Seguridad en la red

Xavier Vilajosana Guillén

PID_00147721

Universitat Oberta de Catalunya
www.uoc.edu

Ninguna parte de esta publicación, incluido el diseño general y la cubierta, puede ser copiada, reproducida, almacenada o transmitida de ninguna forma, ni por ningún medio, sea éste eléctrico, químico, mecánico, óptico, grabación, fotocopia, o cualquier otro, sin la previa autorización escrita de los titulares del copyright.

Índice

Introducción

La seguridad en la red es un aspecto de primordial importancia. Cada día aparecen nuevas aplicaciones remotas o en red; estas aplicaciones almacenan la información en hardware remoto y requieren intercambios de datos constantes entre nuestros ordenadores personales y los servidores remotos. Todo este flujo de información hace necesaria la existencia de mecanismos para evitar el uso malicioso de la información y asegurar la privacidad y protección de los usuarios de la red.

A la hora de proteger las redes de comunicaciones, la **criptografía** es la herramienta fundamental que nos permite evitar que alguien intercepte, manipule o falsifique los datos transmitidos. Dedicaremos una parte de este módulo a introducir los conceptos de la criptografía necesarios para entender cómo se aplica a la protección de las comunicaciones.

La finalidad básica de la criptografía es el envío de información secreta. Si aplicamos una transformación, conocida como **cifrado**, a la información que queremos mantener en privado, aunque un adversario consiga ver qué datos estamos enviando le serán completamente ininteligibles. Sólo el destinatario legítimo será capaz de hacer la transformación inversa y recuperar los datos originales.

En el módulo también estudiaremos los sistemas cortafuegos que se encargarán de proteger el acceso a determinados puntos de la red. Veremos que un sistema cortafuegos actúa como una barrera central para reforzar el control de acceso a los servicios que se ejecutan tanto en el interior como el exterior de la red. El cortafuegos intenta prevenir los ataques del exterior contra las máquinas internas de una red denegando peticiones de conexión desde partes no autorizadas.

Por otra parte veremos también que existen **redes privadas virtuales** que se construyen haciendo uso de los nodos de una red, pero que, mediante técnicas criptográficas y protocolos seguros, permiten aislar a los usuarios de esta red de los de la red de comunicación.

Finalmente haremos una pasada por los diferentes protocolos y aplicaciones más relevantes para asegurar las redes de comunicaciones. Conoceremos las **cookies**, los contenidos dinámicos y cómo éstos pueden afectar a la seguridad de una sitio web. La existencia de protocolos como **SSL/TLS** en el nivel de transporte ha permitido que se pueda dotar de seguridad a las aplicaciones

en red y a sus protocolos. En este módulo también conoceremos el funcionamiento de estos protocolos y, de paso, cómo se pueden hacer transacciones seguras en la red.

Objetivos

El estudio de este módulo os permitirá alcanzar los objetivos siguientes:

1. Conocer los principales mecanismos de seguridad en la red.

2. Conocer los fundamentos de la criptografía y los certificados digitales.

3. Tener una visión global de los elementos de la red que pueden precisar mecanismos de seguridad y conocer los posibles puntos débiles de una red de computadores.

1. Cortafuegos

Cuando un equipo está conectado a una red de computadores se pueden identificar tres áreas de riesgo:

1) El número de puntos que se pueden utilizar como origen para realizar un ataque contra cualquier componente de la red se incrementa. En un sistema aislado (sin conexión), un requisito necesario para ser atacado es forzosamente la existencia de un acceso físico al equipo. Pero en el caso de un sistema en red, cada uno de los equipos que pueda enviar información hacia la víctima podrá ser utilizado por un posible atacante.

Algunos servicios (como por ejemplo web y DNS) necesitan estar abiertos públicamente, de forma que cualquier equipo conectado a Internet podría ser el origen de una posible actividad maliciosa contra ellos. Eso hace que sea muy probable la existencia de ataques regulares contra estos sistemas.

2) La segunda área de riesgo incluye la expansión del perímetro físico del sistema telemático al que acaba de ser conectado el equipo. Cuando la máquina está aislada, cualquier actividad puede ser considerada como interna del equipo (y por lo tanto, de confianza). El procesador trabaja con los datos que encuentra en la memoria, que al mismo tiempo han sido cargados desde un medio de almacenamiento secundario. Estos datos están realmente bien protegidos contra actos de modificación, eliminación, observación maliciosa, etc., ya que se transfieren entre diferentes componentes de confianza.

Pero esta misma premisa no es cierta cuando los datos son transferidos a través de una red. La información transmitida por el medio de comunicación es reenviada por dispositivos que están totalmente fuera del control del receptor. Esta información podría ser leída, almacenada, modificada y más tarde retransmitida hacia el receptor legítimo. Especialmente en grandes redes como Internet, no es trivial la autenticación del origen que se presenta como el emisor de un mensaje.

3) Finalmente, la tercera área de riesgo se debe al aumento del número de servicios de autenticación (generalmente un servicio de login-password) que un sistema conectado a la red tiene que ofrecer, con respecto a un sistema aislado. Estos servicios no dejan de ser simples aplicaciones (con posibles errores de programación o de diseño) que protegen el acceso a los recursos de los equipos del sistema. Un error o vulnerabilidad en alguno de estos servicios puede poner en apuros a todo el sistema.

La prevención de ataques es la suma de una serie de mecanismos de seguridad que proporcionan un primer nivel de defensa contra cierto tipo de ataques antes de que éstos lleguen a su objetivo.

1.1. Sistemas cortafuegos

Los sistemas cortafuegos[1] son un mecanismo de control de acceso sobre la capa de red. La idea básica es separar nuestra red (donde los equipos que intervienen son de confianza) de los equipos del exterior (potencialmente hostiles).

(1)En inglés, *firewall*.

Un sistema cortafuegos actúa como una barrera central para reforzar el control de acceso a los servicios que se ejecutan tanto en el interior como en el exterior de la red. El cortafuegos intentará prevenir los ataques del exterior contra las máquinas internas de nuestra red denegando peticiones de conexión desde partes no autorizadas.

Un cortafuegos puede ser cualquier dispositivo utilizado como mecanismo de control de acceso a nivel de red para proteger una red en particular o un conjunto de redes. En la mayoría de los casos, los sistemas cortafuegos se utilizan para prevenir accesos ilícitos al interior de la red.

Un cortafuegos es aquel sistema de red expresamente encargado de separar redes de computadores y efectuar un control del tráfico existente entre ellas. Este control consiste, en última instancia, en permitir o denegar el paso de la comunicación de una red a otra mediante el control de los protocolos Transmission Control Protocol/Internet Protocol (TCP/IP).

A la hora de instalar y configurar un sistema cortafuegos en nuestra red, se tiene que tener presente lo siguiente:

1) Todo el tráfico que sale del interior hacia el exterior de la red a proteger, y viceversa, tiene que pasar por el cortafuegos. Eso se puede conseguir bloqueando físicamente todo el acceso al interior de la red a través del sistema.

2) Sólo el tráfico autorizado, definido en las políticas de seguridad local del sistema, podrá traspasar el bloqueo.

3) El propio cortafuegos tiene que estar protegido contra posibles intrusiones. Eso implica el uso de un sistema operativo de confianza con suficientes garantías de seguridad.

1.2. Construcción de sistemas cortafuegos

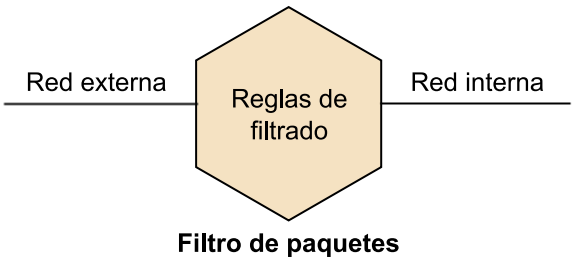
En el sentido más general, un sistema cortafuegos consta de software y hardware. El software puede ser propietario, shareware, o freeware. Por otra parte, el hardware podrá ser cualquiera que pueda soportar este software.

Actualmente, dos de las tecnologías más utilizadas a la hora de construir sistemas cortafuegos son las siguientes:

- Encaminadores con filtrado de paquetes.
- Pasarelas en el ámbito de aplicación.

1.2.1. Encaminadores con filtrado de paquetes

Un encaminador con filtrado de paquetes es un dispositivo que encamina el tráfico TCP/IP (encaminador de TCP/IP) sobre la base de una serie de reglas de filtrado que deciden qué paquetes se encaminan a través de él y cuáles son descartados.



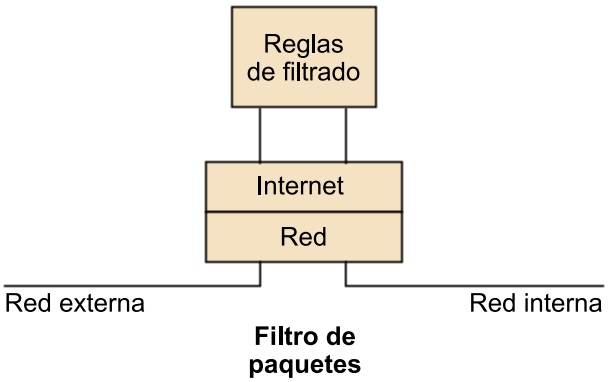
Filtro de paquetes

Las reglas de filtrado se encargan de determinar si a un paquete le está permitido pasar de la parte interna de la red a la parte externa, y viceversa, verificando el tráfico de datos legítimo entre ambas partes.

Los encaminadores con filtrado de paquetes, al trabajar a nivel de red, pueden aceptar o denegar paquetes fijándose en las cabeceras del protocolo (tanto IP como UDP[2], TCP...), en las que se incluyen:

[(2)]UDP es la abreviatura de User Datagram Protocol.

- Direcciones de origen y de destino.
- Tipos de protocolo e indicadores especiales.
- Puertos de origen y de destino o tipo de mensaje (según el protocolo).
- Contenido de los paquetes.
- Tamaño del paquete.



Las reglas están organizadas en conjuntos de listas con una determinada política por defecto (denegarlo todo, aceptarlo todo...).

Cada paquete que llegue al dispositivo será comparado con las reglas, empezando por el principio de la lista hasta que se encuentre la primera coincidencia. Si existe alguna coincidencia, se activará la acción indicada en la regla (denegar, aceptar, redirigir...).

Por el contrario, si no es posible ninguna coincidencia, se consultará la política por defecto para saber qué acción tomar (dejar pasar el paquete, descartarlo, redireccionarlo, etc.). Si se trata, por ejemplo, de una política de denegación por defecto, en el caso de no existir ninguna coincidencia con el paquete será descartado éste.

Una política de denegación por defecto acostumbra a ser más costosa de mantener, ya que será necesario que el administrador indique explícitamente todos los servicios que tienen que permanecer abiertos (el resto, por defecto, serán todos denegados).

En cambio, una política de aceptación por defecto es más sencilla de administrar, pero incrementa el riesgo de permitir ataques contra nuestra red, ya que requiere que el administrador indique explícitamente qué paquetes hay que descartar (el resto, por defecto, serán todos aceptados).

Ventajas y desventajas de los encaminadores con filtrado de paquetes

La construcción de un sistema cortafuegos mediante un encaminador con filtrado de paquetes es realmente económica, ya que generalmente se suele hacer con hardware ya disponible. Además, ofrece un alto rendimiento a redes con una carga de tráfico elevada.

Adicionalmente, esta tecnología permite la implantación de la mayor parte de las políticas de seguridad necesarias.

Iptables

Un ejemplo de encaminador con filtro de paquetes podría ser la **aplicación iptables**, implementada como una parte del software de direccionamiento del kernel Linux.

Las políticas de seguridad son el resultado de documentar las expectativas de seguridad, intentando plasmar en el mundo real los conceptos abstractos de seguridad. Pueden definirse de manera procesal (plasmando de forma práctica las ideas o filosofías de la empresa en cuanto a seguridad) o de manera formal (utilizando un modelo matemático que intente abarcar todos los posibles estados y operaciones).

A pesar de estas ventajas, los encaminadores de red con filtrado de paquetes pueden presentar algunas deficiencias, como, por ejemplo:

- Muchos de los encaminadores utilizados pueden ser vulnerables a ataques existentes (aunque la mayoría de los distribuidores tienen los correspondientes paquetes de actualizaciones para solucionarlo). Por otra parte, no suelen tener capacidades de registro. Eso provoca que al administrador le sea difícil saber si el propio encaminador está siendo atacado.

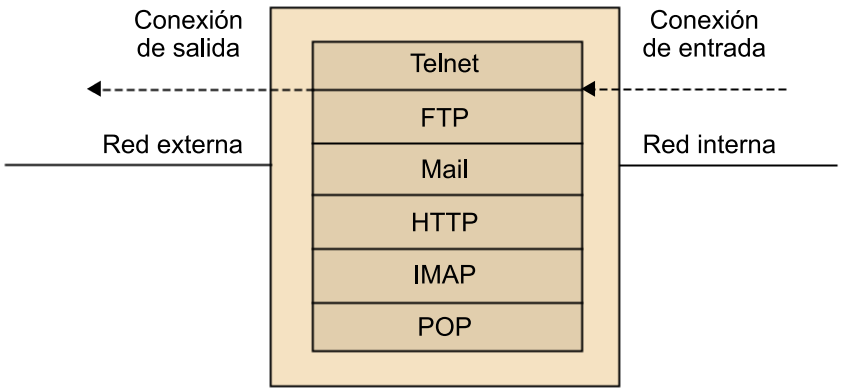
- Su capacidad de actuación puede llegar a deteriorarse a causa de la utilización de un filtrado excesivamente estricto, dificultando también el proceso de gestión del dispositivo si este número de reglas llega a ser muy elevado.

- Las reglas de filtrado pueden llegar a ser muy complicadas, lo que provoca en ocasiones que los posibles descuidos en su configuración sean aprovechados por un atacante para realizar una violación de la política de seguridad.

1.2.2. Pasarela en el ámbito de aplicación

Una pasarela en el ámbito de aplicación, conocida también como servidor intermediario[3], no direcciona paquetes a nivel de red sino que actúa como retransmisor en el ámbito de aplicación. Los usuarios de la red contactarán con el servidor intermediario, que a su vez estará ofreciendo un servicio proxy asociado a una o más aplicaciones determinadas.

[3] En inglés, *proxy*.



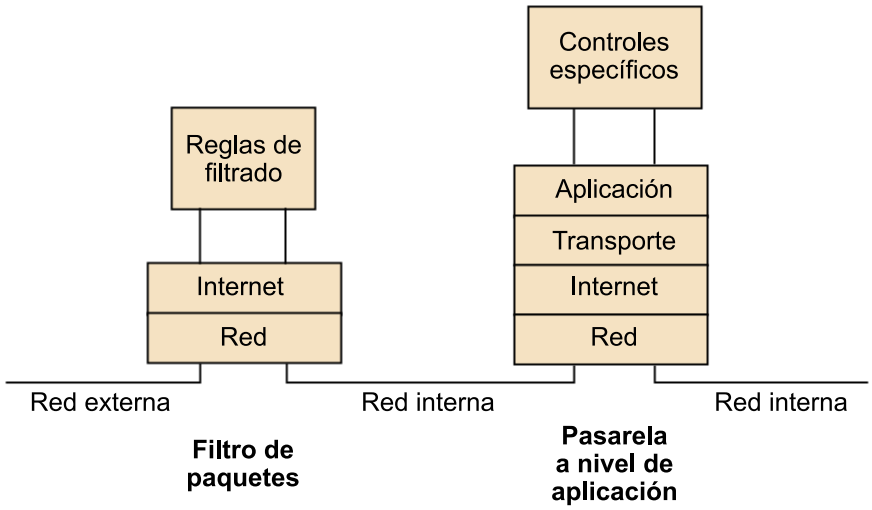
Pasarela a nivel de aplicación

El servicio proxy se encargará de realizar las conexiones solicitadas con el exterior, y cuando reciba una respuesta se encargará de retransmitirla hacia el equipo que había iniciado la conexión. Así, el servicio proxy ejecutado en la pasarela aplicará las normas para decidir si se acepta o se rechaza una petición de conexión.

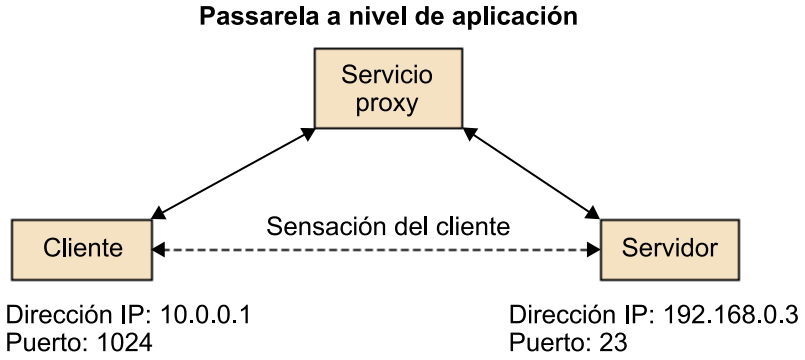
Una pasarela separa completamente el interior del exterior de la red en la capa de enlace, ofreciendo únicamente un conjunto de servicios en el ámbito de aplicación. Eso permite la autenticación de los usuarios que realizan peticiones de conexión y el análisis de conexiones en el ámbito de aplicación.

Estas dos características hacen que las pasarelas ofrezcan una mayor seguridad con respecto a los filtros de paquetes, presentando un rango de posibilidades muy elevado. Por el contrario, la penalización introducida por estos dispositivos es mucho mayor. En el caso de una gran carga de tráfico en la red, el rendimiento puede llegar a reducirse drásticamente.

En la práctica, las pasarelas y los dispositivos de red con filtrado de paquetes son complementarios. Así, estos dos sistemas se pueden combinar, lo que proporcionará más seguridad y flexibilidad que si sólo se utilizara uno, tal como se muestra en la figura siguiente.



Cuando la pasarela autentifica al cliente, abre una conexión al servidor proxy, siendo éste el responsable de transmitir los datos que el cliente recibe del servidor intermediario.



Este funcionamiento particular provoca que las pasarelas en el ámbito de aplicación presenten un rendimiento inferior al de los filtros de paquetes. Con el fin de evitarlo, los servidores intermediarios hacen una copia de los datos transmitidos para entregárselos a otro cuando éste los solicite.

El uso de las pasarelas proporciona diversos beneficios. De entrada, las pasarelas permiten sólo aquellos servicios para los cuales hay un servidor proxy habilitado. Así, si una pasarela contiene servicios intermediarios tan sólo para los servicios HTTP y DNS, entonces sólo HTTP y DNS serán permitidos en la red interna. El resto de los servicios serán completamente rechazados.

Otro beneficio del uso de pasarelas es que el protocolo también puede ser filtrado, prohibiendo así el uso de diferentes subservicios dentro de un mismo servicio permitido. Por ejemplo, mediante una pasarela que filtrara conexiones FTP, sería posible prohibir únicamente el uso del comando PUT de FTP dejando habilitados el resto de comandos. Esta característica no es posible obtenerla sólo con el uso de filtros de paquetes.

Adicionalmente, los servidores intermediarios también pueden implantar el filtro de conexiones por dirección IP de la misma manera que los filtros de paquetes, ya que la dirección IP está disponible en el ámbito de aplicación en el cual se hará el filtrado.

A pesar de obtener más control global sobre los servicios vigilados, las pasarelas también presentan algunas problemáticas. Uno de los primeros inconvenientes a destacar es la necesidad de tener que configurar un servidor proxy para cada servicio de la red que se ha de vigilar (HTTP, DNS, Telnet, FTP...). Además, en el caso de protocolos cliente-servidor, como, por ejemplo, Telnet, pueden llegar a ser necesarios algunos pasos adicionales para conectar el punto final de la comunicación.

2. Redes privadas virtuales

Los protocolos seguros que hemos visto hasta ahora permiten proteger las comunicaciones, por ejemplo, de una aplicación implementada como un proceso cliente que se ejecuta en un ordenador y un proceso servidor que se ejecuta en otro ordenador. Si hay otras aplicaciones que también necesitan una comunicación segura entre estos dos ordenadores, o entre ordenadores situados en las mismas redes locales, pueden hacer uso de otras instancias de los protocolos seguros: nuevas asociaciones de seguridad IPsec, nuevas conexiones SSL TLS, etc.

Una posibilidad alternativa es establecer una red privada virtual[4] entre estos ordenadores o las redes locales en las que están situados.

(4)En inglés, Virtual Private Network (VPN).

2.1. Definición y tipo de redes privadas virtuales

Una **red privada virtual** (VPN) es una configuración que combina el uso de dos tipos de tecnologías:

- Las tecnologías de seguridad que permiten la definición de una **red privada**, es decir, un medio de comunicación confidencial que no puede ser interceptado por usuarios ajenos a la red.
- Las tecnologías de encabezamiento de protocolos que permiten que, en vez de una conexión física dedicada a la red privada, se pueda utilizar una infraestructura de red pública, como es Internet, para definir por encima de ella una **red virtual**.

Por lo tanto, una VPN es una red lógica o virtual creada sobre una infraestructura compartida, pero que proporciona los servicios de protección necesarios para una comunicación segura.

Dependiendo de la situación de los nodos que hacen uso de esta red, podemos considerar tres tipos de VPN:

1) Éste es el caso habitual en que una empresa dispone de redes locales en diferentes sedes, geográficamente separadas, en cada una de las cuales hay una red privada o **intranet**, de acceso restringido a sus empleados. Si interesa que desde una de las sedes se pueda acceder a las intranets de las otras sedes, se puede usar una VPN para interconectar estas redes privadas y formar una intranet única.

2) Cuando un empleado de la empresa quiere acceder a la intranet desde un ordenador remoto, puede establecer una VPN de este tipo entre este ordenador y la intranet de la empresa. El ordenador remoto puede ser, por ejemplo, un PC que el empleado tiene en su casa, o un ordenador portátil desde el que se conecta a la red de la empresa cuando está de viaje.

3) A veces, a una empresa le interesa compartir una parte de los recursos de su intranet con determinados usuarios externos, como, por ejemplo, proveedores o clientes de la empresa. La red que permite estos accesos externos a una intranet se llama **extranet**, y su protección se alcanza mediante una VPN extranet.

2.2. Configuraciones y protocolos utilizados en VPN

A cada uno de los tipos de VPN le suele corresponder una configuración específica.

- En las VPN entre intranets, la situación más habitual es que en cada intranet haya una **pasarela VPN** que conecta la red local con Internet. Esta pasarela se comunica con la de las otras intranets, aplicando el cifrado y las protecciones que sean necesarias para las comunicaciones de pasarela a pasarela a través de Internet. Cuando los paquetes llegan a la intranet de destino, la pasarela correspondiente los descifra y los reenvía por la red local hasta el ordenador que los tenga que recibir. De esta manera se utiliza la infraestructura pública como Internet, en lugar de establecer líneas privadas dedicadas, que supondrían un coste más elevado. También se aprovecha la fiabilidad y redundancia que proporciona Internet, ya que si una ruta no está disponible siempre se pueden encaminar los paquetes por otro lugar, mientras que con una línea dedicada la redundancia supone un coste todavía mayor.

- En las VPN de acceso remoto, a veces llamadas VPDN[5], un usuario se puede comunicar con una intranet a través de un proveedor de acceso a Internet, utilizando tecnología convencional, como, por ejemplo, a través de un módem ADSL. El ordenador del usuario tiene que disponer de software **cliente VPN** para comunicarse con la pasarela VPN de la intranet y llevar a cabo la autenticación necesaria, el cifrado, etc. Así, también se aprovecha la infraestructura de los proveedores de Internet para el acceso a la intranet, sin necesidad de llamadas a un módem de la empresa, que pueden llegar a tener un coste considerable.

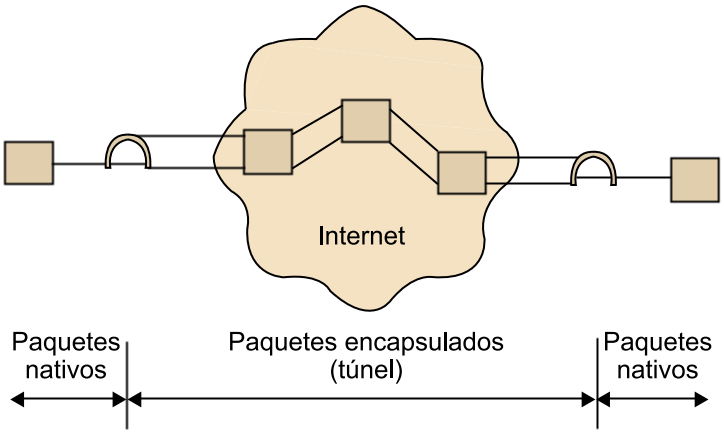
- El caso de las VPN extranet puede ser como el de las VPN entre intranets, en las que la comunicación segura se establece entre pasarelas VPN, o bien como el de las VPN de acceso remoto, en las que un cliente VPN se comunica con la pasarela de la intranet. La diferencia, sin embargo, es que, en este caso, normalmente el control de acceso es más restrictivo para permitir sólo el acceso a los recursos autorizados.

(5)VPDN son las siglas de Virtual Private Dial Network.

La definición de una red virtual se lleva a cabo mediante el establecimiento de **túneles**, que permiten encapsular paquetes de la red virtual, con sus protocolos, dentro de paquetes de otra red, que normalmente es Internet, con su protocolo, es decir IP.

Para la comunicación entre las diferentes intranets, o entre el ordenador que accede remotamente y la intranet, se pueden utilizar los protocolos que sean más convenientes. Los paquetes de estos protocolos, para poder hacerlos llegar a su destino a través de Internet, se pueden encapsular en datagramas IP, que contendrán en su interior los paquetes originales. Cuando llegan a su destino, estos datagramas se desencapsulan para recuperar los paquetes con el formato "nativo" del protocolo correspondiente.

Uso de túneles en una VPN



Hay diversos protocolos que pueden ser utilizados para establecer los túneles, dependiendo del nivel de la comunicación en el que se quiera hacer la protección.

El protocolo utilizado en la gran mayoría de configuraciones VPN es IPsec en modo túnel, generalmente con ESP para cifrar los datos, y opcionalmente con AH para autenticar los paquetes encapsulados. Las pasarelas VPN, en este caso, son pasarelas seguras Ipsec.

En el caso de las VPN de acceso remoto o VPDN, existe la posibilidad de encapsular tramas PPP, que son las que transmite normalmente un cliente VPN de este tipo, sobre datagramas IP. Hay diversas opciones para encapsular PPP (que, a su vez, puede encapsular otros protocolos de red, como IPX, etc., y posiblemente IP) sobre IP:

- El protocolo PPTP[6] (RFC 2637) especifica una técnica para la encapsulación de tramas PPP pero no añade servicios de autenticación. Estos servicios se pueden realizar con los mismos protocolos que utiliza PPP, como Password Authentication Protocol (BUCHE) o Challenge Handshake Authentication Protocol (CHAP).

[6]PPTP es la abreviatura de Point-to-Point Tunneling Protocol.

- El protocolo L2F[7] (RFC 2637) es parecido al PPTP pero también puede trabajar con SLIP, además de con PPP. Para la autenticación puede utilizar protocolos auxiliares como Remote Authentication Dial-In User Service (RADIUS).

(7)L2F es la abreviatura de Layer Two Forwarding.

- El protocolo L2TP[8] (RFC 2661) combina las funcionalidades que ofrecen PPTP y L2F.

(8)L2TP es la abreviatura de Layer Two Tunneling Protocol.

- El protocolo SSH[9] ofrece la posibilidad de redirigir puertos TCP sobre un canal seguro, que podemos considerar como un túnel a nivel de transporte. Desde este punto de vista, también se podría considerar una conexión SSL TLS como un túnel en el ámbito del transporte que proporciona confidencialidad y autenticación. Habitualmente, sin embargo, este último tipo de túnel no sirve para cualquier tipo de tráfico sino sólo para datos TCP, y por lo tanto no se considera parte integrante de una VPN.

(9)SSH es la abreviatura de Secure Shell.

3. Introducción a la criptografía

A lo largo de la historia se han diseñado diferentes técnicas para ocultar el significado de la información que no interesa que sea conocida por extraños. Algunas de ellas ya se utilizaban en tiempo de la antigua Grecia o del Imperio romano: por ejemplo, se atribuye a Julio César la invención de un código para enviar mensajes cifrados que no pudieran ser interpretados por el enemigo.

Criptografía

Los términos *criptografía*, *criptología*, etc. provienen de la raíz griega *kryptós*, que quiere decir 'escondido'.

La **criptografía** estudia, desde un punto de vista matemático, los métodos para proteger la información. Por otra parte, el **criptoanálisis** estudia las posibles técnicas para contrarrestar los métodos criptográficos, y es de gran utilidad para ayudar a hacerlos más robustos y difíciles de atacar. El conjunto formado por estas dos disciplinas, criptografía y criptoanálisis, se llama **criptología**.

Uso del cifrado

El hecho de usar el cifrado parte de la constatación de que intentar evitar la interceptación de la información por parte de un intruso (espía) es muy costoso. Enviar mensajes cifrados es más fácil, y así, aunque un espía los pueda ver, no podrá interpretar la información que contienen.

Cuando la protección que queremos obtener consiste en garantizar el secreto de la información, es decir, la **confidencialidad**, utilizamos el método criptográfico conocido como **cifrado**.

Si M es el mensaje que queremos proteger o **texto en claro**, cifrarlo consiste en aplicarle un **algoritmo** de cifrado f, que lo transforme en otro mensaje que llamaremos **texto cifrado**, C. Eso lo podemos expresar como:

$$C = f(M)$$

A fin de que este cifrado sea útil, tiene que existir otra transformación o **algoritmo** de descifrado f^{-1}, que permita recuperar el mensaje original a partir del texto cifrado:

$$M - f^{-1}(C)$$

La cifra de César

La **cifra de César** consiste en sustituir cada letra del mensaje por la que hay tres posiciones después en el alfabeto (volviendo a empezar por la letra A si llegamos a la Z). Así, si aplicamos este algoritmo de cifrado al texto en claro "ALEA JACTA EST" (y utilizamos el alfabeto latino actual, porque en tiempos de César no había letras como la "W"), obtenemos el texto cifrado "DOHD MDFWD HVW". El descifrado en este caso es bien sencillo: sólo hay que sustituir cada letra por la que hay 3 posiciones después en el alfabeto.

Un esquema como el de la cifra de César tiene el inconveniente de que si el enemigo descubre cuál es el algoritmo de cifrado (y a partir de aquí deduce el algoritmo inverso), será capaz de interpretar todos los mensajes cifrados que capture. Entonces, habría que instruir a todos los "oficiales de comunicaciones" del ejército para que aprendieran un nuevo algoritmo, lo cual podría resultar complicado. En vez de eso, lo que se hace hoy es utilizar como algoritmo una función con un parámetro denominado **clave**.

Ejemplo de uso de una clave

El algoritmo de Julio César se puede generalizar definiendo una clave k que indique cuántas posiciones se adelanta cada letra en el alfabeto. La cifra de César, pues, utiliza $k = 3$. Para el descifrado se puede utilizar el mismo algoritmo, pero con la clave invertida (*de* ^{o} e, $x = -k$).

Entonces, podemos hablar de una función de cifrado e con una **clave** de cifrado k, y una función de descifrado d con una **clave de descifrado** x:

$$C = e(k,M)$$

$$M = d(x,C) = d(x,e(k,M))$$

Así, una solución al problema del espía que se entera de cómo descifrar los mensajes podría ser continuar utilizando el mismo algoritmo, pero con una clave diferente.

Seguridad por ocultismo

A lo largo de la historia ha habido casos que han demostrado la peligrosidad de basar la protección en mantener los algoritmos en secreto (lo que se conoce como "seguridad por ocultismo"). Si el algoritmo es conocido por muchos, es más fácil que se detecten las debilidades o vulnerabilidades y se puedan corregir rápidamente. Si no, un experto podría deducir el algoritmo por ingeniería inversa, y acabar descubriendo que tiene puntos débiles por donde atacarlo, como pasó con el algoritmo A5/1 de la telefonía móvil GSM.

Una premisa fundamental en la criptografía moderna es la llamada **suposición de Kerckhoffs**, de acuerdo con la cual los algoritmos tienen que ser conocidos públicamente y su seguridad sólo tiene que depender de la clave. En lugar de intentar ocultar el funcionamiento de los algoritmos, es mucho más seguro y efectivo mantener en secreto sólo las claves.

Un algoritmo se considera **seguro** si a un adversario le es imposible obtener el texto en claro M aunque conozca el algoritmo e y el texto cifrado C. Es decir, es imposible descifrar el mensaje sin saber cuál es la clave de descifrado. La palabra "imposible", sin embargo, hay que considerarla con diferentes matices. Un algoritmo criptográfico es **computacionalmente seguro** si, aplicando el mejor método conocido, la cantidad de recursos necesarios (tiempo de cálculo, número de procesadores, etc.) para descifrar los mensajes sin conocer la clave es mucho mayor (unos cuantos órdenes de magnitud) que lo que está al

alcance de nadie. En el límite, un algoritmo es **incondicionalmente seguro** si no puede ser invertido ni con recursos infinitos. Los algoritmos que se hacen servir en la práctica son (o intentan ser) computacionalmente seguros.

La acción de intentar descifrar mensajes sin conocer la clave de descifrado se llama "ataque". Si el ataque tiene éxito, se suele decir coloquialmente que se ha conseguido "romper" el algoritmo. Hay dos maneras de llevar a cabo un ataque:

- Mediante el **criptoanálisis**, es decir, estudiando matemáticamente la manera de deducir el texto en claro a partir del texto cifrado.

- Aplicando la **fuerza bruta, es decir, probando uno a uno todos los valores posibles de la clave de descifrado** *x* hasta encontrar uno que produzca un texto en claro con sentido.

Ataques a la cifra de César

Continuando con el ejemplo del algoritmo de César generalizado (con clave), un ataque criptoanalítico podría consistir en analizar las propiedades estadísticas del texto cifrado. Las letras cifradas que más se repiten probablemente corresponden a vocales o a las consonantes más frecuentes, las combinaciones más repetidas de dos (o tres) letras seguidas probablemente corresponden a los dígrafos (o trígrafos) que típicamente aparecen más veces en un texto, etc.

Por otra parte, el ataque con fuerza bruta consistiría en intentar el descifrado con cada uno de los 25 posibles valores de la clave ($1 \leq x \leq 25$, si el alfabeto tiene 26 letras) y mirar cuál da un resultado inteligible. En este caso, la cantidad de recursos necesarios es tan modesta que incluso se puede hacer el ataque a mano. Por lo tanto, este cifrado sería un ejemplo de algoritmo inseguro o débil.

A continuación veremos las características de los principales sistemas criptográficos utilizados en la protección de las comunicaciones. A partir de ahora, consideraremos los mensajes en claro, los mensajes cifrados y las claves como secuencias de bits.

3.1. Criptografía de clave simétrica

Los sistemas criptográficos **de clave simétrica** se caracterizan por que la clave de descifrado *x* es idéntica a la clave de cifrado *k*, o bien se puede deducir directamente a partir de ésta.

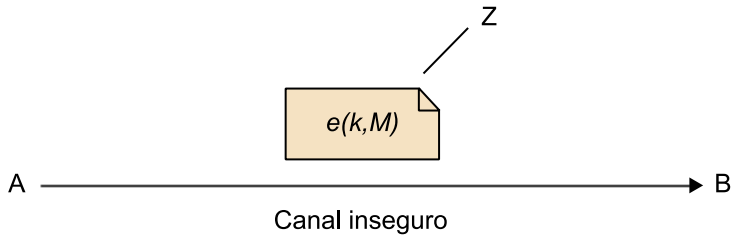
Para simplificar, supondremos que en este tipo de sistemas la clave de descifrado es igual a la de cifrado: $x = k$ (si no, siempre podemos considerar que en el algoritmo de descifrado el primer paso es calcular la clave *x* a partir de la *k*). Por eso, estas técnicas criptográficas se llaman de clave simétrica, y, a veces, también de **clave compartida**. Así, tenemos:

$$C = e(k,M)$$

$$M = d(k,C) = d(k,e(k,M))$$

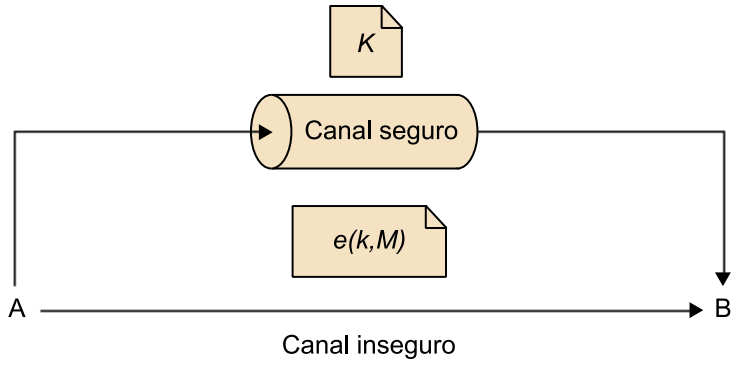
La seguridad del sistema reside, pues, en mantener en secreto la clave *k*. Cuando los participantes en una comunicación quieren intercambiarse mensajes confidenciales, tienen que escoger una clave secreta y utilizarla para cifrar los mensajes. Entonces pueden enviar estos mensajes por cualquier canal de comunicación, en la confianza de que, aunque el canal sea inseguro y susceptible de ser inspeccionado por terceros, ningún espía *Z* será capaz de interpretarlos.

Mensaje *M* cifrado con una clave *k* entre A y B



Si el sistema es de clave compartida, es necesario que el valor de la clave secreta *k* que utilizan *A* y *B* sea el mismo. Ahora bien, ¿cómo se pueden asegurar de que sea así? Está claro que no pueden enviar la clave escogida a través del canal de comunicación de que disponen, porque la hipótesis inicial es que este canal es inseguro y cualquiera podría descubrir la información que se transmite. Una posible solución es utilizar un canal aparte, que pueda ser considerado suficientemente seguro:

Establecimiento de un canal seguro entre A y B



Canales seguros

Podrían ser ejemplos de "canales seguros" el correo tradicional (no electrónico) o un servicio de mensajería "física", una conversación telefónica o cara a cara, etc.

Esta solución, sin embargo, tiene algunos inconvenientes. Por una parte, se supone que el canal seguro no será de uso tan ágil como el canal inseguro (si lo fuera, sería mucho mejor enviar todos los mensajes confidenciales sin cifrar por el canal seguro, y olvidarnos del canal inseguro). Por lo tanto, puede ser difícil ir cambiando la clave. Y por otra parte, este esquema no es bastante general: puede ser que tengamos que enviar información cifrada a alguien

con quien no podemos contactar de ninguna otra manera. Como veremos más adelante, estos problemas relacionados con el **intercambio de claves** se solucionan con la criptografía asimétrica.

A continuación repasaremos las características básicas de los principales algoritmos criptográficos de clave simétrica, los cuales agruparemos en dos categorías: algoritmos de flujo y algoritmos de bloque.

3.1.1. Algoritmos de cifrado de flujo

El funcionamiento de una cifra de flujo consiste en combinar el texto en claro *M* con un texto de cifrado *S* que se obtiene a partir de la clave simétrica *k*. Para descifrar sólo hay que hacer la operación inversa con el texto cifrado y el mismo texto de cifrado *S*.

La operación de combinación que se utiliza típicamente es la suma, y la operación inversa, por lo tanto, es la resta. Si el texto está formado por caracteres, este algoritmo sería como una cifra de César en la que la clave va cambiando de un carácter a otro. La clave que corresponde cada vez viene dada por el texto de cifrado *S* (llamado *keystream* en inglés).

Suma y resta de bits

Cuando trabajamos con aritmética binaria o aritmética módulo 2, se cumple:

$0 + 0 = 0$	$0 - 0 = 0$	$0 \oplus 0 = 0$
$0 + 1 = 1$	$0 - 1 = 1$	$0 \oplus 1 = 1$
$1 + 0 = 1$	$1 - 0 = 1$	$1 \oplus 0 = 1$
$1 + 1 = 0$	$1 - 1 = 0$	$1 \oplus 1 = 0$

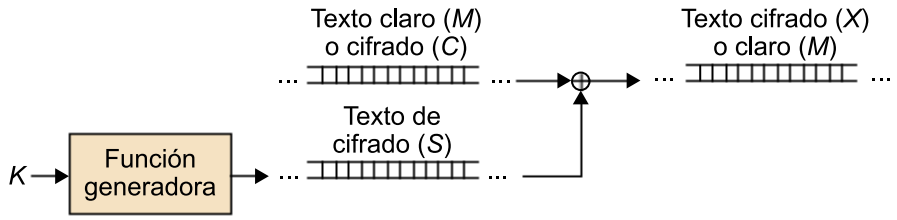
Si consideramos el texto formado por bits, la suma y la resta son equivalentes. En efecto, cuando se aplican bit a bit, las dos son idénticas a la operación lógica "o exclusiva", denotada con el operador XOR (*"eXclusive OR"*) o el símbolo ⊕. Así, pues:

$$C = M \oplus S(k)$$

$$M = C \oplus S(k)$$

En los esquemas de cifrado de flujo, el texto en claro *M* puede ser de cualquier longitud, y el texto de cifrado *S* tiene que ser al menos igual de largo. De hecho, no hay que disponer del mensaje entero antes de empezar a cifrarlo o descifrarlo, ya que se puede implementar el algoritmo para que trabaje con un "flujo de datos" que va llegando (el texto en claro o el texto cifrado) y otro "flujo de datos" que se va generando a partir de la clave (el texto de cifrado). De aquí procede el nombre de este tipo de algoritmos. La figura siguiente ilustra el mecanismo básico de su implementación.

Esquema del cifrado y descifrado de flujo



Hay diferentes maneras de obtener el texto de cifrado *S* en función de la clave *k*:

- Si se escoge una secuencia *k* más corta que el mensaje *M*, una posibilidad sería repetirla cíclicamente tantas veces como haga falta para ir sumándola al texto en claro. El inconveniente de este método es que se puede romper fácilmente, sobre todo cuanto más corta sea la clave (en el caso mínimo, el algoritmo sería equivalente a la cifra de César).

- En el otro extremo, se podría hacer directamente $S(k) = k$. Eso quiere decir que la propia clave tiene que ser tan larga como el mensaje a cifrar. Éste es el principio de la llamada **cifra de Vernam**. Si *k* es una secuencia totalmente aleatoria que no se repite cíclicamente, estamos ante un ejemplo de cifrado incondicionalmente seguro, tal como lo hemos definido al principio de este módulo. Este cifrado se llama en inglés *one-time pad* (cuaderno de un solo uso). El problema en este caso es que el receptor tiene que disponer de la misma secuencia aleatoria para poder hacer el descifrado, y si le tiene que llegar a través de un canal seguro, la pregunta es inmediata: ¿por qué no enviar el mensaje confidencial *M*, que es igual de largo que la clave *k*, directamente por el mismo canal seguro? Es evidente, pues, que este algoritmo es muy seguro, pero no es muy práctico en general.

Uso del cifrado de Vernam

A menudo, las comunicaciones entre los portaaviones y los aviones utilizan el cifrado de Vernam. En este caso, se utiliza el hecho de que en un momento dado (antes de elevarse) tanto el avión como el portaaviones están en el mismo lugar, e intercambiarse, por ejemplo, un disco duro de 20 GB con una secuencia aleatoria no es un problema. Posteriormente, cuando el avión despega puede establecer una comunicación segura con el portaaviones utilizando una cifra de Vernam con la clave aleatoria que ambas partes comparten.

- Lo que se hace en la práctica es utilizar funciones que generan **secuencias pseudoaleatorias** a partir de una **semilla** (un número que actúa como parámetro del generador), y lo que se intercambia como clave secreta *k* es sólo esta semilla.

Ejemplo de funciones pseudoaleatorias

Son ejemplos de funciones pseudoaleatorias las basadas en registros de desplazamiento realimentados (*feedback shift registers* o FSR). El valor inicial del registro es la semilla. Para ir obteniendo cada bit pseudoaleatorio se desplazan todos los bits del registro una posición y se coge el que sale fuera del registro. El bit que queda libre en el otro extremo se llena con un valor que es función del resto de bits.

Las secuencias pseudoaleatorias se llaman así porque intentan parecer aleatorias pero, claro está, son generadas algorítmicamente. En cada paso, el algoritmo estará en un determinado estado, que vendrá dado por sus variables internas. Como las variables serán finitas, habrá un número máximo de posibles estados diferentes. Eso quiere decir que, al cabo de un cierto período, los datos generados se volverán a repetir. Para que el algoritmo sea seguro, interesa que el período de repetición sea cuanto más largo mejor (con relación al mensaje a cifrar), con el fin de dificultar el criptoanálisis. Las secuencias pseudoaleatorias también han de tener otras propiedades estadísticas equivalentes a las de las secuencias aleatorias puras.

Cifras síncronas y asíncronas

Si el texto de cifrado S depende exclusivamente de la clave k, se dice que el cifrado es síncrono. Este cifrado tiene el problema de que, si por algún error de transmisión se pierden bits (o llegan repetidos), el receptor se desincronizará y sumará bits del texto S con bits del texto cifrado C que no tocan, con lo cual el texto descifrado a partir de entonces será incorrecto.

Eso se puede evitar con el cifrado asíncrono (o "autosincronizante"), en el cual el texto S se calcula a partir de la clave k y el mismo texto cifrado C. Es decir, en vez de realimentarse con sus propios bits de estado, el generador se realimenta con los n últimos bits cifrados transmitidos. De esta manera, si se pierden m bits consecutivos en la comunicación, el error afectará como máximo al descifrado de $m + n$ bits del mensaje original.

Ejemplos de algoritmos de cifrado de flujo

Los algoritmos de cifrado de flujo actualmente en uso tienen la propiedad de ser poco costosos de implementar. Las implementaciones en hardware son relativamente simples y, por lo tanto, tienen un rendimiento eficiente (en términos de bits cifrados por segundo). Pero también las implementaciones en software pueden ser bastante eficientes.

Las características del cifrado de flujo lo hacen apropiado para entornos en los que haga falta un rendimiento alto y los recursos (capacidad de cálculo, consumo de energía) sean limitados. Por eso se suelen utilizar en comunicaciones móviles: redes locales sin hilos, telefonía móvil, etc.

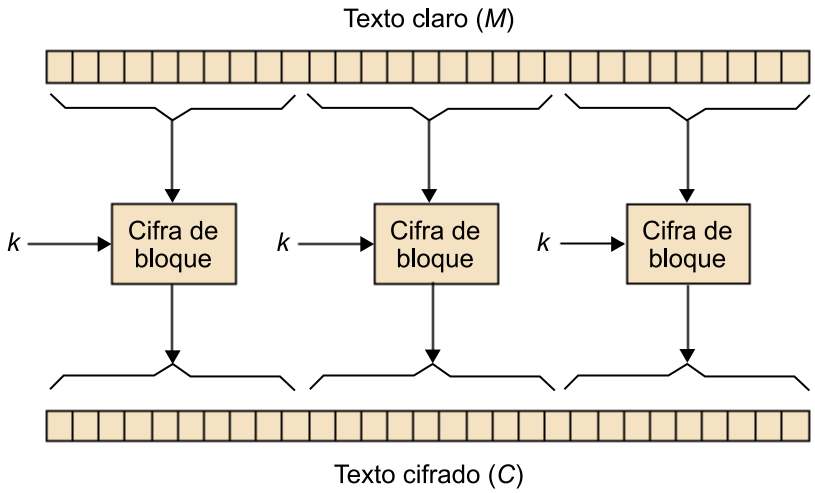
3.1.2. Algoritmos de cifrado de bloque

En una cifra de bloque, el algoritmo de cifrado o descifrado se aplica separadamente a bloques de entrada de longitud fija b, y para cada uno de ellos el resultado es un bloque de la misma longitud.

Para cifrar un texto en claro de L bits hace falta dividirlo en bloques de b bits cada uno y cifrar estos bloques uno a uno. Si L no es múltiplo de b, se pueden añadir bits adicionales hasta llegar a un número entero de bloques,

pero entonces puede ser necesario indicar de alguna manera cuántos bits había realmente en el mensaje original. El descifrado también se tiene que realizar bloque a bloque. La figura siguiente muestra el esquema básico del cifrado de bloque.

Esquema del cifrado de bloque



Muchos de los algoritmos de cifrado de bloque se basan en la combinación de dos operaciones básicas: sustitución y transposición.

La **sustitución** consiste en traducir cada grupo de bits de la entrada a otro, de acuerdo con una permutación determinada.

La cifra de César sería un ejemplo simple de sustitución, en el que cada grupo de bits correspondería a una letra. De hecho, se trata de un caso particular de **sustitución alfabética**. En el caso más general, las letras del texto cifrado no tienen por qué estar a una distancia constante (la *k* del algoritmo, tal como lo hemos definido) de las letras del texto en claro. Clave de la sustitución alfabética

Clave de la sustitución alfabética

Está claro que la clave tiene que ser una **permutación** del alfabeto, es decir, no puede haber letras repetidas ni faltar ninguna. Si no, la transformación no sería invertible en general. La clave se puede expresar entonces como la secuencia correlativa de letras que corresponden a la *A*, la *B*, la *C*, etc.

Ejemplo de sustitución alfabética

Alfabeto: A B C D E F G H Y J K L M N O P Q R S T U V W X Y Z

Clave: Q W E R T Y U Y O P A S D F G H J K L Z X C V B N M

Texto en claro: A L E A J A C T A E S T

Texto cifrado: Q S T Q P Q E Z Q T L Z

La **transposición** consiste en reordenar la información del texto en claro según un patrón determinado.

Ejemplo de transposición

Un ejemplo de transposición podría ser formar grupos de cinco letras, incluidos los espacios en blanco, y reescribir cada grupo (1,2,3,4,5) en el orden (3,1,5,4,2):

Texto en claro: A L E A J A C T A E S T

Texto cifrado: E A A L C J A T A S T E

La transposición por sí sola no dificulta extraordinariamente el criptoanálisis, pero puede combinarse con otras operaciones para añadir complejidad a los algoritmos de cifrado.

El **producto de cifras**, o combinación en cascada de diversas transformaciones criptográficas, es una técnica muy efectiva para implementar algoritmos bastante seguros de manera sencilla. Por ejemplo, muchos algoritmos de cifrado de bloque se basan en una serie de iteraciones de productos sustitución-transposición.

Confusión y difusión

Dos propiedades deseables en un algoritmo criptográfico son la "confusión", que consiste en esconder la relación entre la clave y las propiedades estadísticas del texto cifrado, y la "difusión", que propaga la redundancia del texto en claro a lo largo del texto cifrado para que no sea fácilmente reconocible.

La confusión hace que cambiando un solo bit de la clave cambien muchos bits del texto cifrado, y la difusión hace que el cambio de un solo bit del texto en claro afecte también a muchos bits del texto cifrado.

En un bucle de productos de cifras básicas, la sustitución contribuye a la confusión, mientras que la transposición contribuye a la difusión. La combinación de estas transformaciones simples, repetidas diversas veces, hace que los cambios en la entrada se propaguen por toda la salida por un "efecto avalancha".

Ejemplos de algoritmos de cifrado de bloque

Durante muchos años el algoritmo de cifrado de bloque ha sido el algoritmo más estudiado y, al mismo tiempo, el más utilizado. Desarrollado por IBM durante los años setenta, fue adoptado por el NBS norteamericano (nombre que tenía entonces el actual NIST) como estándar para el cifrado de datos del año 1977. NBS y NISTNBS eran las siglas de National Bureau of Standards, y NIST son las siglas de National Institute of Standards and Technology.

El algoritmo admite una clave de 64 bits, pero sólo 7 de cada 8 intervienen en el cifrado. Un posible uso de los bits de la clave DES que no influyen en el algoritmo es como bits

de paridad, de manera que la longitud efectiva de la clave es de 56 bits. Los bloques de texto en los cuales se aplica el DES tienen que ser de 64 bits cada uno.

La parte central del algoritmo consiste en dividir la entrada en grupos de bits, hacer una sustitución diferente sobre cada grupo y, a continuación, una transposición de todos los bits. Esta transformación se repite 16 veces: a cada iteración, la entrada es una transposición diferente de los bits de la clave sumada bit a bit (XOR) con la salida de la iteración anterior. Tal como está diseñado el algoritmo, el descifrado se realiza igual que el cifrado, pero haciendo las transposiciones de la clave en el orden inverso (empezando por la última).

A pesar de que a lo largo de los años el algoritmo DES ha demostrado ser muy resistente al criptoanálisis, su principal problema actualmente es la vulnerabilidad a los ataques de fuerza bruta, a causa de la longitud de la clave, de sólo 56 bits. Si en los años setenta era muy costoso hacer una búsqueda entre las 2^{56} combinaciones posibles, la tecnología actual permite romper el algoritmo en un tiempo cada vez más corto.

Por eso, en 1999 el NIST cambió el algoritmo DES por el "Triple DES" como estándar oficial, mientras no estuviera disponible el nuevo estándar AES. El Triple DES, como su nombre indica, consiste en aplicar el DES tres veces consecutivas. Eso se puede hacer con tres claves (k_1, k_2, k_3), o bien con sólo dos diferentes (k_1, k_2, y otra vez k_1). La longitud total de la clave en la segunda opción es de 112 bits (dos claves de 56 bits), que hoy ya se considera bastante segura; la primera opción proporciona más seguridad, pero a costa de utilizar una clave total de 168 bits (3 claves de 56 bits), que puede costar un poco más gestionar e intercambiar.

Para hacer el sistema adaptable al estándar antiguo, en el Triple DES se aplica una secuencia cifrado-descifrado-cifrado (E-D-E) en lugar de tres cifrados:

$$C = e(k_3, d(k_2, e(_{k1},M)))$$

o bien: mod 1 $emC = e(_{k1}, d(k_2, e(_{k1},M)))$

Así, haciendo $k_2 = k_1$ tenemos un sistema equivalente al DES simple.

Como el estándar DES empezaba a quedarse anticuado, a causa sobre todo de la longitud tan corta de las claves, y el Triple DES no es muy eficiente cuando se implementa con software, en 1997 el NIST convocó a la comunidad criptológica para que presentara propuestas sobre un nuevo estándar, el AES, que sustituyera al DES. De los quince algoritmos candidatos que se aceptaron, se escogieron cinco como finalistas, y en octubre del 2000 se dio a conocer al ganador, el algoritmo Rijndael, propuesto por los criptógrafos belgas Joan Daemen y Vincent Rijmen.

El Rijndael puede trabajar con bloques de 128, 192 o 256 bits (aunque el estándar AES sólo prevé los de 128), y la longitud de la clave también puede ser 128, 192 o 256 bits. Dependiendo de esta última longitud, el número de iteraciones del algoritmo es 10, 12 o 14, respectivamente. Cada iteración incluye una sustitución fija byte a byte, una transposición, una transformación consistente en desplazamientos de bits y XOR, y una suma binaria (XOR) con bits obtenidos a partir de la clave.

Repetición del DES

La operación de aplicar el cifrado DES con una clave y volver a cifrar el resultado con otra no es equivalente a un solo cifrado DES (no hay ninguna clave única que dé el mismo resultado que dan las otras dos juntas). Si no fuera así, la repetición del DES no sería más segura que el DES simple.

3.1.3. Uso de los algoritmos de clave simétrica

Cuando se usa el cifrado simétrico para proteger las comunicaciones, se puede escoger el algoritmo que sea más apropiado a las necesidades de cada aplicación: normalmente, a más seguridad menos velocidad de cifrado, y viceversa.

Un aspecto a tener en cuenta es que si bien el cifrado puede hacer que un atacante no descubra directamente los datos transmitidos, a veces es posible que se pueda deducir información indirectamente. Por ejemplo, en un protocolo

que utilice mensajes con una cabecera fija, la aparición de los mismos datos cifrados varias veces en una transmisión puede indicar dónde empiezan los mensajes.

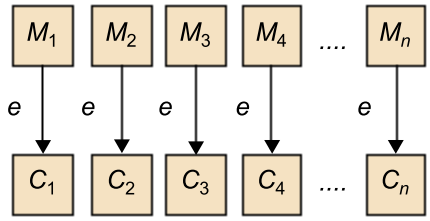
Eso no pasa con las cifras de flujo si su período es lo bastante largo, pero en una cifra de bloque, si dos bloques de texto en claro son iguales y se utiliza la misma clave, los bloques cifrados también serán iguales. Para contrarrestar esta propiedad, se pueden aplicar en diversos **modos de operación** a las cifras de bloque:

1) Modo ECB (*Electronic Codebook*). Es el más sencillo, y consiste simplemente en dividir el texto en bloques y cifrar cada uno de manera independiente. Este modo presenta el problema de dar bloques iguales cuando en la entrada hay bloques iguales.

Nombre del modo ECB

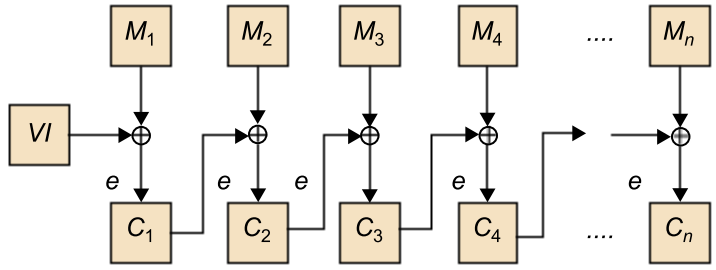
Modo ECB (*Electronic Codebook*) da la idea de que se puede considerar como una simple sustitución bloque a bloque, de acuerdo con un código o diccionario (con muchas entradas, eso sí) que viene dado por la clave.

Modo de operación ECB



2) Modo CBC (*Cipher Block Chaining*). En el modo CBC, antes de cifrar cada bloque de texto en claro se le suma (bit a bit, con XOR) el bloque cifrado anterior. En el primer bloque se le suma un **vector de inicialización** (VI), que es un conjunto de bits aleatorios de la misma longitud que un bloque. Escogiendo vectores diferentes cada vez, aunque el texto en claro sea el mismo, los datos cifrados serán diferentes. El receptor tiene que conocer el valor del vector antes de empezar a descifrar, pero no hay que guardar este valor en secreto, sino que normalmente se transmite como cabecera del texto cifrado.

Modo de operación CBC



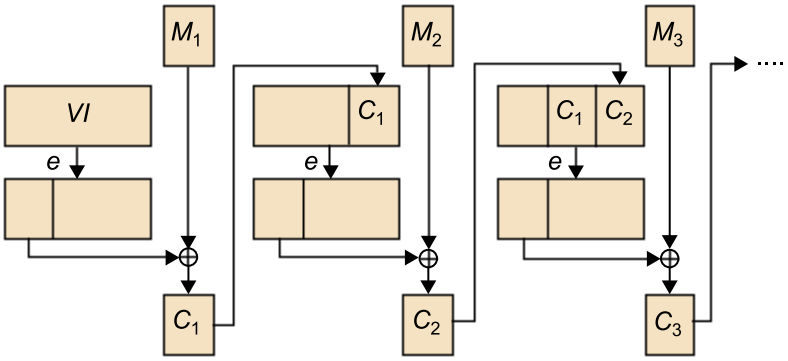
Modo CFB como cifra de flujo

Es fácil ver que el modo CFB (y también el OFB) se puede considerar como una cifra de flujo que utiliza como función generadora una cifra de bloque.

3) Modo CFB (*Cipher Feedback*). En el modo CFB, el algoritmo de cifrado no se aplica directamente al texto en claro sino a un vector auxiliar (inicialmente igual al VI). Del resultado del cifrado se toman n bits que se suman a n bits del texto en claro para obtener n bits de texto cifrado. Estos bits cifrados se usan al mismo tiempo para actualizar el vector auxiliar. El número n de bits

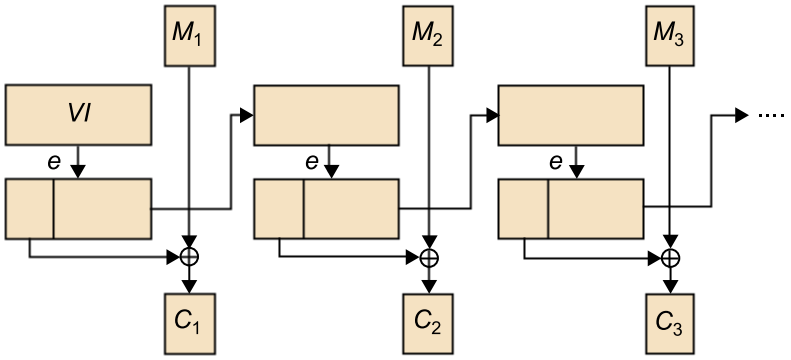
generados en cada iteración puede ser menor o igual que la longitud de bloque *b*. Tomando por ejemplo $n = 8$, tenemos una cifra que genera un byte cada vez sin tener que esperar a tener un bloque entero para poder cifrarlo.

Modo de operación CFB



4) Modo OFB (*Output Feedback*). El modo OFB es como el CFB, pero en lugar de actualizar el vector auxiliar con el texto cifrado, se actualiza con el resultado obtenido del algoritmo de cifrado. La propiedad que diferencia este modo de los otros es que un error en la recepción de un bit cifrado afecta sólo al descifrado de este bit.

Modo de operación OFB



5) Otros modos. A partir de los modos anteriores se pueden definir diversas variantes. Por ejemplo, el modo CTR (*Counter*) es como el OFB, pero el vector auxiliar no se realimenta con el cifrado anterior sino que es simplemente un contador que se va incrementando.

Hay otra técnica para evitar que en textos de entrada iguales se obtengan textos cifrados iguales, que es aplicable también a los cifrados que no usan vector de inicialización (incluidas las cifras de flujo). Esta técnica consiste en modificar la clave secreta con bits aleatorios antes de usarla en el algoritmo de cifrado (o en el de descifrado). Como estos bits aleatorios sirven para dar un "sabor" diferente a la clave, se les suele llamar **bits de sal**. Igual que el vector de inicialización, los bits de sal se envían en claro antes del texto cifrado.

3.1.4. Funciones *hash* seguras

Aparte de cifrar datos, hay algoritmos basados en técnicas criptográficas que se usan para garantizar la autenticidad de los mensajes. Un tipo de algoritmos de estas características son las llamadas **funciones *hash* seguras**, también conocidas como funciones de **resumen de mensaje**[10].

(10)En inglés, *message digest*.

En general, podemos decir que una función *hash* nos permite obtener una cadena de bits de longitud fija, relativamente corta, a partir de un mensaje de longitud arbitraria:

$$H = h(M)$$

Para mensajes M iguales, la función h tiene que dar resúmenes H iguales. Pero si dos mensajes dan el mismo resumen H no necesariamente tienen que ser iguales. Eso es así porque sólo hay un conjunto limitado de posibles valores H, ya que su longitud es fija, y en cambio puede haber muchos más mensajes M (si la longitud puede ser cualquiera, habrá infinitos).

Para poder aplicarla en un sistema de autenticación, la función h tiene que ser una función *hash* segura.

Secreto de los algoritmos

Notad que las funciones *hash* son conocidas ya que todo el mundo tiene que poder calcular los resúmenes de la misma manera.

Se entiende que una función *hash* o de resumen es **segura** si cumple estas condiciones:

1) Es **unidireccional**, es decir, si tenemos $H = h(M)$, es computacionalmente inviable encontrar M *a* partir del resumen H.

2) Es **resistente a colisiones**, es decir, dado un mensaje M cualquiera, es computacionalmente inviable encontrar un mensaje $M' \neq M$ tal que $h(M') = h(M)$.

Estas propiedades permiten el uso de las funciones *hash* seguras para dar un servicio de autenticidad basado en una clave secreta S compartida entre dos partes A y B. Aprovechando la unidireccionalidad, cuando A quiere enviar un mensaje M a B puede preparar otro mensaje M_s, por ejemplo, concatenando el original con la clave: $M_s = (M,s)$. Entonces envía a B el mensaje M y el resumen del mensaje M_s.

Para comprobar la autenticidad del mensaje recibido, B verifica que el resumen corresponde efectivamente a M_s. Si es así, quiere decir que lo ha generado alguien que conoce la clave secreta s (que tendría que ser A), y también que nadie ha modificado el mensaje.

Otra técnica consistiría en calcular el resumen del mensaje *M* y cifrarlo utilizando *s* como clave de cifrado.

Autenticidad y confidencialidad

Cifrar sólo el resumen, en lugar del mensaje entero, es más eficiente porque hay menos bits a cifrar. Eso, claro está, suponiendo que sólo haga falta autenticidad, y no confidencialidad. Si también interesa que el mensaje sea confidencial, entonces sí que hay que cifrarlo entero.

Para verificar la autenticidad hace falta recuperar el resumen enviado, descifrándolo con la clave secreta *s*, y compararlo con el resumen del mensaje *M*. Un atacante que quisiera modificar el mensaje sin conocer la clave podría intentar sustituirlo por otro que dé el mismo resumen, con lo que *B* no detectaría la falsificación. Pero si la función de resumen es resistente a colisiones, eso le tendría que ser imposible al atacante.

Con el fin de dificultar los ataques contra las funciones de resumen, por una parte los algoritmos tienen que definir una relación compleja entre los bits de la entrada y cada bit de la salida. Por otra parte, los ataques de fuerza bruta se contrarrestan haciendo bastante larga la longitud del resumen. Por ejemplo, los algoritmos usados actualmente generan resúmenes de 128 o 160 bits. Eso quiere decir que un atacante podría tener que probar del orden de 2^{128} o 2^{160} mensajes de entrada para encontrar una colisión (es decir, un mensaje diferente que diera el mismo resumen).

Pero hay otro tipo de ataque más ventajoso para el atacante, llamado **ataque del cumpleaños**. Un ataque de este tipo parte del supuesto de que el atacante puede escoger el mensaje que será autenticado. La víctima lee el mensaje y, si está de acuerdo, lo autentica con su clave secreta. Pero el atacante ha presentado este mensaje porque ha encontrado otro que da el mismo resumen, y por lo tanto puede hacer creer al destinatario que el mensaje auténtico es este otro. Y eso se puede conseguir haciendo una búsqueda de fuerza bruta con muchas menos operaciones: del orden de 2^{64} o 2^{80}, si el resumen es de 128 o 160 bits, respectivamente.

Resistencia fuerte a las colisiones

La resistencia de los algoritmos de resumen a las colisiones, tal como la hemos definido, a veces se llama "resistencia débil", mientras que la propiedad de ser resistente a ataques del cumpleaños se llama " resistencia fuerte".

Paradoja del cumpleaños

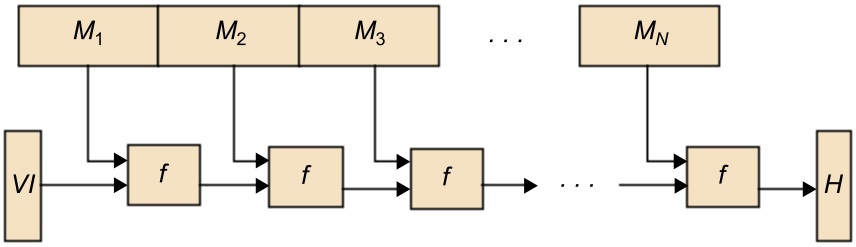
El nombre de este tipo de ataque viene de un problema clásico de probabilidades conocido como la "paradoja del cumpleaños". El problema consiste en encontrar el número mínimo de personas que tiene que haber en un grupo para que la probabilidad de que al menos dos de ellas celebren su cumpleaños el mismo día sea superior al 50%. Una respuesta intuitiva puede ser que la solución es del orden de 200, pero este resultado no es correcto. Podría serlo si se quisiera obtener el número de personas necesarias para que hubiera un 50% de probabilidad de coincidencia con una persona determinada. Si aceptamos que la coincidencia sea entre cualquier pareja de personas, la solución es un número mucho menor: 23.

La conclusión es que si una función de resumen puede dar N valores diferentes, para que la probabilidad de encontrar dos mensajes con el mismo resumen sea del 50% el número de mensajes que hay que probar es del orden de $\sqrt{N}$

Ejemplos de funciones *hash* seguras

El esquema de la mayoría de funciones *hash* usadas actualmente es parecido al de los algoritmos de cifrado de bloque: el mensaje de entrada se divide en bloques de la misma longitud, y a cada uno se le aplica una serie de operaciones junto con el resultado obtenido del bloque anterior. El resultado que queda después de procesar el último bloque es el resumen del mensaje.

Esquema de las funciones de resumen



El objetivo de estos algoritmos es que cada bit de la salida dependa de todos los bits de la entrada. Eso se consigue con diversas iteraciones de operaciones que "mezclen" los bits entre sí, de manera parecida a cómo la sucesión de transposiciones en los cifrados de bloque provoca un "efecto avalancha" que garantiza la difusión de los bits.

Hasta hace poco, el algoritmo de *hash* más usado era el MD5 (Message Digest 5). Pero como el resumen que da es de sólo 128 bits, y aparte se han encontrado otras formas de generar colisiones parciales en el algoritmo, actualmente se recomienda utilizar algoritmos más seguros, como el SHA-1 (Secure Hash Algorithm-1). El algoritmo SHA-1, publicado en 1995 en un estándar del NIST (como revisión de un algoritmo anterior llamado simplemente SHA), da resúmenes de 160 bits. En el 2002 el NIST publicó variantes de este algoritmo que generan resúmenes de 256, 384 y 512 bits.

3.2. Criptografía de clave pública

Trataremos ahora los conceptos fundamentales de la criptología de clave pública.

3.2.1. Algoritmos de clave pública

Como hemos visto en el subapartado anterior, uno de los problemas de la criptografía de clave simétrica es el de la distribución de las claves. Este problema se puede solucionar si utilizamos **algoritmos de clave pública**, también llamados **algoritmos de clave asimétrica**.

En un algoritmo criptográfico de clave pública se utilizan claves diferentes para el cifrado y el descifrado. Una de ellas, la **clave pública**, se puede obtener fácilmente a partir de la otra, la **clave privada**, pero en cambio es computacionalmente muy difícil obtener la clave privada a partir de la clave pública.

Los algoritmos de clave pública típicamente permiten cifrar con la clave pública (k_{pub}) y descifrar con la clave privada (k_{pr}):

$$C = e(k_{pub}, M)$$

$$M = d(k_{pr}, C)$$

Pero también puede haber algoritmos que permitan cifrar con la clave privada y descifrar con la pública (más adelante veremos cómo se puede utilizar esta propiedad):

$$C = e(k_{pr}, M)$$

$$M = d(k_{pub}, C)$$

Adaptación de los problemas difíciles

Si el avance de la tecnología reduce el tiempo de resolución, se puede aumentar la longitud *n*, con lo cual harán falta unas cuantas operaciones más para el planteamiento, pero la complejidad de la solución crecerá exponencialmente.

Los algoritmos de clave pública se basan en problemas matemáticos "fáciles" de plantear a partir de la solución, pero "difíciles" de resolver. En este contexto, se entiende que un problema es fácil si el tiempo para resolverlo, en función de la longitud *n* de los datos, se puede expresar en forma polinómica, como, por ejemplo, $n^2 + 2n$ (en teoría de la complejidad, se dice que estos problemas son de la "clase P"). Si el tiempo de resolución crece más rápidamente, como por ejemplo con 2^n, el problema se considera difícil. Así, se puede escoger un valor de *n* tal que el planteamiento sea viable pero la resolución sea computacionalmente intratable.

Un ejemplo de problema fácil de plantear pero difícil de resolver es el de los logaritmos discretos. Si trabajamos con aritmética módulo *m*, es fácil calcular esta expresión:

$$y = b^x \bmod m$$

El valor x se llama logaritmo discreto de *y* en base *b* módulo *m*. Escogiendo convenientemente *b* y *m*, puede ser difícil calcular el logaritmo discreto de cualquier *y*. Una posibilidad es ir probando todos los valores de x: si *m* es un número de *n* bits, el tiempo para encontrar la solución aumenta proporcionalmente a 2^n. Hay otros métodos más eficientes para calcular logaritmos discretos, pero el mejor algoritmo conocido también necesita más tiempo de lo que se puede expresar polinómicamente.

Ejemplo de operaciones módulo *m*

Para obtener $14^{11} \bmod 19$ podemos multiplicar 11 veces el número 14, dividir el resultado entre 19 y tomar el residuo de la división, que es igual en 13. Pero también se puede aprovechar que el exponente 11 es 1011 en binario ($11 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$), y por lo tanto $14^{11} = 14^8 \cdot 14^2 \cdot 14^1$, para obtener el resultado con menos multiplicaciones:

$$14^1 = 14 \equiv 14 (\text{mod} 19) \rightarrow 14$$
$$14^2 = 14^1 \times 14^1 \equiv 14 \times 14 \equiv 196 \equiv 6 (\text{mod} 19) \rightarrow 6$$
$$14^4 14^2 \times 14^2 \equiv 6 \times 6 \equiv 36 \equiv 17 (\text{mod} 19)$$
$$14^8 = 14^4 \times 14^4 \equiv 17 \times 17 \equiv 289 \equiv 4 (\text{mod} 19) \rightarrow 4$$
$$9 - 9 \qquad 336 \equiv 13 (\text{mod} 19)$$

Así, sabemos que $\log_{14} 13 = 11 \pmod{19}$. Pero si tuviéramos que obtener el logaritmo de cualquier otro número *y* tendríamos que ir probando uno a uno los exponentes hasta encontrar uno que dé como resultado *y*. Y si en vez de tratarse de números de 4 o 5 bits como éstos fueran números de más de 1.000 bits, el problema sería intratable.

Así pues, los algoritmos de clave pública se tienen que diseñar de manera que sea inviable calcular la clave privada a partir de la pública, y lógicamente también tiene que ser inviable invertirlos sin saber la clave privada, pero el cifrado y el descifrado se tienen que poder realizar en un tiempo relativamente corto.

En la práctica, los algoritmos utilizados permiten cifrar y descifrar fácilmente, pero todos ellos son considerablemente más lentos. Por eso, la criptografía de clave pública se suele utilizar sólo en los problemas que la criptografía simétrica no puede resolver: el intercambio de claves y la autenticación con no repudio (firmas digitales).

Velocidad de la criptografía de clave pública

El cifrado y descifrado con algoritmos de clave pública puede llegar a ser dos o tres órdenes de magnitud más lento que con criptografía simétrica. que los equivalentes con criptografía simétrica.

Los mecanismos de **intercambio de claves** permiten que dos partes se pongan de acuerdo en las claves simétricas que utilizarán para comunicarse, sin que un tercero que esté escuchando el diálogo pueda deducir cuáles son estas claves.

Ejemplo de mecanismos de intercambio de claves

A puede escoger una clave simétrica *k*, cifrarla con la clave pública de *B*, y enviar el resultado a *B*. Entonces *B* descifrará con su clave privada el valor recibido, y sabrá cuál es la clave *k* que ha escogido *A*. El resto de la comunicación irá cifrada con un algoritmo simétrico (mucho más rápido), utilizando esta clave *k*. Los atacantes, como no conocerán la clave privada de *B*, no podrán deducir el valor de *k*.

La **autenticación** basada en clave pública se puede llevar a cabo si el algoritmo permite utilizar las claves a la inversa: la clave privada para cifrar y la clave pública para descifrar.

Si *A* envía un mensaje cifrado con su clave privada, todo el mundo podrá descifrarlo con la clave pública de *A*, y al mismo tiempo todo el mundo sabrá que el mensaje sólo lo puede haber generado quien conozca la clave privada asociada (que tendría que ser *A*). Ésta es la base de las **firmas digitales**.

Ejemplos de algoritmos de clave pública

El RSA es el algoritmo más utilizado en la historia de la criptografía de clave pública. Su nombre viene de las iniciales de los que lo diseñaron en 1977: Ronald Rivest, Adi Shamir y Leonard Adleman. La clave pública está formada por un número n, calculado como producto de dos factores primeros muy grandes $(n = p \cdot q)$, y un exponente e. La clave privada es otro exponente d calculado a partir de p, q y e, de tal forma que el cifrado y el descifrado se pueden realizar así:

$$\text{Cifrado: } C = M^e \bmod n$$

$$\text{Descifrado: } M = C^d \bmod n$$

Como se puede ver, la clave pública y la privada son intercambiables: si se usa cualquiera de ellas para cifrar, habrá que usar la otra para descifrar.

La fortaleza del algoritmo RSA se basa, por una parte, en la dificultad de obtener M a partir de C sin conocer d (problema del logaritmo discreto), y por otra parte, en la dificultad de obtener p y q (y por lo tanto d) a partir de n (problema de la factorización de números grandes, que es otro de los problemas considerados difíciles).

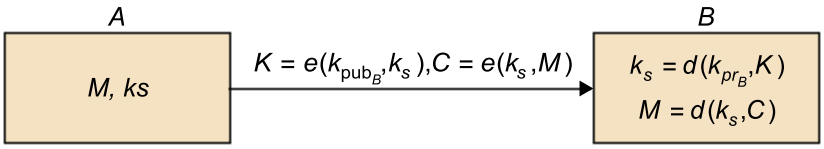
Valores usados en el RSA

Actualmente, el problema de factorizar números de 512 bits es muy complejo, pero abordable si se dispone de bastantes recursos. Por lo tanto, se recomienda utilizar claves públicas con un valor n a partir de 1.024 bits. Como exponente público e típicamente se utilizan valores sencillos como 3 o 65.537 ($12^{16} + 1$) porque hacen más rápido el cifrado.

3.2.2. Uso de la criptografía de clave pública

Hemos visto antes que las principales aplicaciones de la criptografía de clave pública son el intercambio de claves para proporcionar confidencialidad, y la firma digital para proporcionar autenticidad y no repudio.

El problema de la confidencialidad entre dos partes que sólo disponen de un canal inseguro para comunicarse se resuelve con la criptografía de clave pública. Cuando A quiere enviar un mensaje secreto M a B, no hay que cifrar todo el mensaje con un algoritmo de clave pública (eso podría resultar muy lento), sino que se escoge una clave simétrica k_s, llamada a veces **clave de sesión** o **clave de transporte**, y se cifra el mensaje con un algoritmo simétrico utilizando esta clave. Lo único que hay que cifrar con la clave pública de B (k_{pub_B}) es la clave de sesión. En recepción, B utiliza su clave privada (k_{pr_B}) para recuperar la clave de sesión k_s y, entonces, ya puede descifrar el mensaje cifrado.



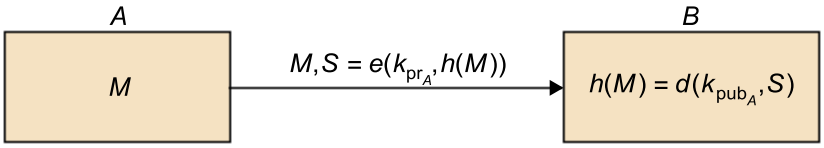
Ya que la clave de sesión es un mensaje relativamente corto (por ejemplo, si es una clave DES sólo tendrá 56 bits), un atacante podría intentar romper el cifrado a la fuerza bruta, pero no intentando descifrar el mensaje con los posibles valores de la clave privada k_{pr_B}, sino cifrando los posibles valores de la

clave de sesión k_s con la clave pública k_{pub_B}. En el caso de una clave de sesión DES, independientemente del número de bits de la clave pública, el atacante sólo necesitaría un esfuerzo del orden de 2^{56} operaciones.

Para evitar este tipo de ataque, lo que se cifra realmente con la clave pública no es directamente el valor secreto (en este caso k_s), sino que a este valor se le añade una cadena más o menos larga de bits aleatorios. El receptor sólo tiene que descartar estos bits aleatorios del resultado que obtenga del descifrado.

Una **firma digital** es básicamente un mensaje cifrado con la clave privada del firmante. Sin embargo, por cuestión de eficiencia, lo que se cifra no es directamente el mensaje a firmar, sino sólo su resumen calculado con una función *hash* segura.

Cuando *A* quiera enviar un mensaje firmado, tendrá que obtener su resumen y cifrarlo con la clave privada k_{pr_A}. Para verificar la firma, el receptor tiene que descifrarla con la clave pública k_{pub_A} y comparar el resultado con el resumen del mensaje: si son iguales, quiere decir que el mensaje lo ha generado *A* y nadie lo ha modificado. Como se supone que la función de resumen es resistente a colisiones, un atacante no podrá modificar el mensaje sin que la firma deje de ser válida.



3.2.3. Infraestructura de clave pública

Tal como hemos visto hasta ahora, la criptografía de clave pública permite resolver el problema del intercambio de claves, haciendo uso de las claves públicas de los participantes. Ahora, sin embargo, se plantea otro problema: si alguien nos dice que es *A* y su clave pública es k_{pub}, ¿cómo poder saber que k_{pub} es realmente la clave pública de *A*? Porque es perfectamente posible que un atacante Z genere su par de claves $(k'_{\text{pr}},k'_{\text{pub}})$ y nos diga "yo soy *A*, y mi clave pública es k'_{pub}".

Una posible solución a este problema es que haya alguien de confianza que nos asegure que efectivamente las claves públicas pertenecen a sus supuestos propietarios. Ese alguien puede firmar un documento que diga "la clave pública de *A* es k_{pub_A}", y publicarlo para que todo el mundo tenga conocimiento de ello. Este tipo de documento se llama **certificado de clave pública** y es la base de lo que se conoce como **infraestructura de clave pública (PKI)**.

4. Certificados digitales

Un certificado de clave pública consta de tres partes básicas:

1) Una identificación de usuario, como, por ejemplo, su nombre.
2) El valor de la clave pública de este usuario.
3) La signatura de las dos partes anteriores.

Si el autor de la signatura es alguien en quien confiamos, el certificado nos sirve como garantía de que la clave pública pertenece al usuario que figura en ella. Quien firma el certificado puede ser una autoridad que se responsabilice de verificar de manera fehaciente la autenticidad de las claves públicas. En este caso, se dice que el certificado ha sido generado por una **autoridad de certificación**[11].

(11)En inglés, Certification Authority (CA).

Puede haber diferentes formatos de certificados, pero el más usado es el de los **certificados X.509**, especificado en la definición del **servicio de directorio X.500**.

El directorio X.500

La especificación del directorio X.500 está publicada en la Serie de Recomendaciones ITU-T X.500, una de las cuales es la Recomendación X.509.

El directorio X.500 permite almacenar y recuperar información, expresada como **atributos**, de un conjunto de **objetos**. Los objetos X.500 pueden representar, por ejemplo, países, ciudades, o bien empresas, universidades (en general, organizaciones), departamentos, facultades (en general, unidades organizativas), personas, etc. Todos estos objetos están organizados jerárquicamente en forma de árbol (en cada nodo del árbol hay un objeto) y, dentro de cada nivel, los objetos se identifican mediante un **atributo distintivo**. A escala global, cada objeto se identifica con un **nombre distintivo**[12], que no es más que la concatenación de los atributos distintivos que hay entre la raíz del árbol y el objeto en cuestión.

(12)En inglés, Distinguished Name (DN).

El sistema de nombres es, pues, parecido al DNS de Internet, con la diferencia de que los componentes de un nombre DNS son simples cadenas de caracteres, y los de un DN X.500 son atributos, cada uno con un tipo y un valor.

Ejemplos de nombre distintivo

Algunos ejemplos de tipos de atributos que se pueden usar como atributos distintivos en un DN son: *countryName* (habitualmente denotado con la abreviatura "c"), *stateOrProvinceName* ("st"), *localityName* ("l"), *organizationName* ("o"), *organizationalUnitName* ("ou"), *commonName* ("cn"), *surname* ("sn"), etc. Un ejemplo de DN es "c=ES, st=Barcelona, l=Barcelona, o=Universitat Oberta de Catalunya, ou=SI, cn=cv.uoc.edu".

X.500 define un protocolo de acceso al servicio que permite realizar operaciones de consulta, y también operaciones de modificación de la información de los objetos. Estas últimas operaciones, sin embargo, normalmente sólo están permitidas a ciertos usuarios autorizados, y por lo tanto hacen falta mecanismos de autenticación de los usuarios, y estos mecanismos están definidos en la Recomendación X.509. Hay un mecanismo básico que utiliza contraseñas, y un mecanismo más avanzado que utiliza certificados.

4.1. Cadenas de certificados y jerarquías de certificación

Un certificado nos soluciona el problema de la autenticidad de la clave pública si está firmado por una CA en la que confiamos. Pero ¿qué pasa si nos comunicamos con un usuario que tiene un certificado emitido por una CA que no conocemos?

Existe la posibilidad que una CA tenga un certificado que garantice la autenticidad de su clave pública, firmado por otra CA. Esta otra CA quizá sí la conocemos, o quizá tiene a su vez un certificado firmado por una tercera CASA, y así sucesivamente. De esta manera, se puede establecer una jerarquía de autoridades de certificación, en la que las CA de nivel más bajo emiten los certificados de usuario, y las CA de cada nivel son certificadas por una de nivel superior.

En un mundo ideal, podría haber una CA raíz que estuviera en la cima de la jerarquía y tuviera la máxima autoridad. En la práctica, esta CA raíz global no existe, y seguramente no existirá nunca (sería difícil que la aceptara todo el mundo). En lugar de haber un solo árbol que comprenda todas las CA del mundo, lo que hay en realidad son árboles independientes (algunos posiblemente con una sola CA), cada uno con su CA raíz.

Para que podamos verificar la autenticidad de su clave pública, un usuario nos puede enviar su certificado, más el certificado de la CA que lo ha emitido, más el de la CA que ha emitido este otro certificado, etc., hasta llegar al certificado de una CA raíz. Eso es lo que se denomina una **cadena de certificados**. Los certificados de las CA raíz tienen la propiedad de ser autofirmados, es decir, están firmados por ellas mismas.

Un posible tipo de extensión de los certificados X.509 es basicConstraints, y un campo de su valor indica si el certificado es de CA (se puede usar su clave para emitir otros certificados) o no. En el caso de que lo sea, otro subcampo (pathLenConstraint) permite indicar el número máximo de niveles de la jerarquía por debajo de esta CA.

4.2. Listas de revocación de certificados

La Recomendación X.509, además de definir el formato de los certificados, también define otra estructura denominada **lista de revocación de certificados**[13].

(13)En inglés, Certificate Revocation List (CRL).

Una lista de este tipo sirve para publicar los certificados que han dejado de ser válidos antes de su fecha de caducidad. Los motivos pueden ser diversos: se ha emitido otro certificado que sustituye al revocado, ha cambiado el DN del titular (por ejemplo, ha dejado de trabajar en la empresa donde estaba), le han robado su clave privada, etc.

Así, si queremos asegurarnos completamente de la validez de un certificado, no basta con verificar su firma, sino que tendremos que obtener la versión actual de la CRL (publicada por la CA que emitió el certificado) y comprobar que el certificado no aparezca en esta lista.

Una CA normalmente actualizará su CRL de forma periódica, añadiendo cada vez los certificados que hayan sido revocados. Cuando llegue la fecha de caducidad que constaba en el certificado, ya no habrá que volver a incluirlo en la CRL. Eso permite que las CRL no crezcan indefinidamente.

5. Seguridad en la red

Hasta el momento hemos visto aspectos teóricos y prácticos relacionados con la arquitectura de la red y los protocolos de comunicación. En este apartado se presenta una visión de los conceptos de seguridad más enfocada a los ámbitos de aplicación y transporte de la red. Trataremos los aspectos a considerar en el desarrollo de aplicaciones como los protocolos de seguridad más usados hoy en día.

5.1. Cookies

Las cookies (galletas) son un mecanismo para facilitar información sobre la navegación realizada. En definitiva, son una herramienta utilizada por los servidores web para almacenar información sobre sus visitantes.

> Una cookie es un fichero de texto que algunos servidores web graban en nuestro ordenador con información sobre la navegación realizada en sus páginas. Este fichero se guarda en nuestro propio disco duro, y será devuelto posteriormente al servidor cuando éste lo solicite.

La caducidad de estas cookies, momento en el cual dejarán de ser activas, la determina el diseñador de la web del servidor, y puede variar entre el tiempo de la propia sesión y el de una fecha especificada. Las cookies no causan daños en el sistema, ya que sólo permiten reconocer al usuario cuando se conecta a un sitio web, registrando las visitas. En concreto, pueden llegar a guardar la contraseña utilizada para acceder a aquella página, datos personales del usuario, etc.; incluso así, muchos usuarios no desean este tipo de control no autorizado y prefieren el anonimato, siendo éste casi el único problema real de las cookies. Destaquemos también que existen virus que buscan esta información "sensible" dentro de las cookies.

Por medio de las opciones de configuración del navegador se puede habilitar o deshabilitar la aceptación de cookies. También podemos configurarlo para que nos avise de la llegada de alguna cookie. Con las últimas versiones de los navegadores se ha ido mejorando el sistema de gestión de las cookies.

Técnicamente, las cookies son trozos de datos arbitrarios definidos por el servidor web y enviados al navegador. El navegador las devuelve sin modificar al servidor, reflejando así un estado (memoria de acontecimientos anteriores) en las transacciones HTTP, que de otra manera serían independientes de estado.

Sin las cookies, cada petición de una página web o un componente de una página web sería un acontecimiento aislado, sin ninguna relación con el resto de peticiones de otras páginas del mismo sitio.

Al devolver una cookie al servidor web, el navegador le proporciona un medio para relacionar la solicitud de la página actual con solicitudes de páginas anteriores. Además de ser definidas por un servidor web, las cookies también pueden ser definidas por un script en un lenguaje como JavaScript, si éste está soportado y habilitado en el navegador web.

Navegadores y cookies

Las especificaciones de cookies sugieren que los navegadores tienen que soportar un número mínimo de cookies o una cantidad mínima de memoria por almacenarlas. En concreto, se espera que un navegador sea capaz de almacenar al menos 300 cookies de 4 kilobytes cada una y al menos 20 cookies por servidor o dominio.

El servidor que establece la cookie puede especificar una fecha de borrado, en cuyo caso la cookie será borrada en dicha fecha. Un sitio de compras podría querer ayudar a sus clientes potenciales recordándoles las cosas que había en su cesta de la compra, incluso si cierran el navegador sin realizar la compra y vuelven más tarde, para evitar que tengan que buscar los productos de nuevo. En este caso, el servidor crearía una cookie con fecha de borrado según el deseo del diseñador del sitio web. Si no se define una fecha de borrado, la cookie es borrada cuando el usuario cierra su navegador. Por lo tanto, definir una fecha de borrado es una manera de hacer que la cookie sobreviva entre una sesión y otra. Por esta razón, a las cookies con fecha de borrado se las llama persistentes.

5.2. Contenidos activos

En los últimos años la web ha evolucionado desde el contenido estático que nos permitía crear HTML hasta contenidos completamente dinámicos, que constituyen ya hoy en día la mayoría de las páginas web. Un contenido dinámico es creado en tiempo de ejecución por un conjunto de procesos que se ejecutan tanto en el cliente como en el servidor, y eso depende de la tecnología utilizada. En este sentido, la seguridad de la web requiere técnicas cada vez más específicas con el fin de asegurar los contenidos y proteger las maquinas de códigos maliciosos. En este sentido introduciremos algunas de las tecnologías usadas para la creación de contenido dinámico.

5.2.1. Applets

Un applet es un componente de software que corre en el contexto de otro programa, por ejemplo, un navegador web. El applet tiene que correr en un contenedor, que lo proporciona un programa anfitrión mediante un *plug-in*, o en aplicaciones, como teléfonos móviles, que soportan el modelo de programación por applets. Los applets se ejecutan en el contexto del cliente y por lo tanto requieren de técnicas que aseguren que la ejecución no dañará la máquina cliente.

A diferencia de un programa, un applet no puede correr de manera independiente, ofrece información gráfica y a veces interactúa con el usuario, típicamente carece de sesión y tiene privilegios de seguridad restringidos. Un applet normalmente lleva a cabo una función muy específica que carece de uso independiente.

Los applets tienen restricciones de seguridad fuertes en lo que respecta a su acceso al ordenador cliente que los está ejecutando. Las políticas de seguridad son, pues, un factor muy importante a tener en cuenta cuando se desarrolla este tipo de software.

5.2.2. Servlets/JSP

Los servlets son objetos Java ejecutados por un servidor de aplicaciones y responden a invocaciones HTTP, sirviendo páginas dinámicas cuyo contenido generalmente es un fichero HTML generado dinámicamente.

Un servlet es capaz de recibir una invocación y generar una respuesta en función de los datos de la invocación, el estado del propio sistema y los datos a que pueda acceder. Los servlets pueden estar empaquetados dentro de un fichero de formato WAR[14], como aplicación web, dentro de un contenedor.

(14)En inglés, Web Application Archive.

La ejecución de un servlet se hace dentro de uno o más procesos del servidor de aplicaciones, de manera que se genera un nuevo flujo. No generar un nuevo proceso (como acostumbran a hacer los CGI) implica un ahorro de recursos que se traduce en un mejor rendimiento del sistema, pero comporta otros problemas de concurrencia.

Los servlets pueden ser objetos java precompilados o JSP compilados en tiempo de ejecución (o en otro momento después del arranque del servidor de aplicaciones).

Por otra parte, JavaServer Pages (JSP) es una tecnología que permite a los desarrolladores de páginas web generar respuestas dinámicamente a peticiones HTTP. La tecnología permite que código Java y ciertas acciones predefinidas sean incrustadas en un contexto estático, es decir, dentro del propio HTML. La sintaxis de JSP incorpora tags XML adicionales, llamados acciones de JSP por ser usados para invocar otras funciones.

5.2.3. CGI

La interfaz de entrada común[15] es una importante tecnología de la World Wide Web que permite a un cliente (navegador web) solicitar datos de un programa ejecutado en un servidor web. CGI especifica un estándar para transferir datos entre el cliente y el programa. Es un mecanismo de comunicación entre el

(15)En inglés, Common Gateway Interface (CGI).

servidor web y una aplicación externa cuyo resultado final de la ejecución son objetos MIME. Las aplicaciones que se ejecutan en el servidor reciben el nombre de CGI.

Las aplicaciones CGI fueron una de las primeras maneras prácticas de crear contenido dinámico para las páginas web. En una aplicación CGI, el servidor web pasa las solicitudes del cliente a un programa externo. Este programa puede estar escrito en cualquier lenguaje que soporte el servidor, aunque por razones de portabilidad se suelen usar lenguajes de script. La salida de este programa es enviada al cliente en lugar del archivo estático tradicional. CGI ha hecho posible la implementación de funciones nuevas y variadas en las páginas web, de tal manera que esta interfaz se ha convertido rápidamente en un estándar, que se implementa en todo tipo de servidores web.

Las CGI son programas que podrían introducir problemas de seguridad si han sido programados malintencionadamente. Por este motivo, muchos de los proveedores de *hosting* de páginas web no permiten la ejecución de este tipo de servicios web.

5.2.4. ASP/PHP

El servidor de páginas activas[16] corresponde a la tecnología introducida por Microsoft en el año 1996, y permite el uso de diferentes scripts y componentes ActiveX al lado del tradicional HTML para mostrar páginas generadas dinámicamente. Se basa en el VBScript, pero existen diversos lenguajes de programación que se pueden utilizar, como, por ejemplo, Perl, Javascript, etc. ASP es una tecnología dinámica que funciona en el lado del servidor, lo que significa que, cuando el usuario solicita un documento ASP, las instrucciones de programación dentro del script se ejecutan en el servidor para enviar al navegador únicamente el código HTML resultante. Al usuario sólo se le envía lo que solicita, y no puede acceder a ningún otro servicio del servidor. Así, una de sus aplicaciones más importantes es la del acceso a bases de datos.

(16)En inglés, Active Server Pages (ASP).

Existen otros lenguajes con funcionalidades parecidas. Entre ellos destacaremos el PHP (Hypertext Preprocessor), que es un lenguaje de programación creado para desarrollar aplicaciones web, muy similar a los lenguajes de programación C o C++. Su código se inserta en las páginas HTML, y se ejecuta en el servidor.

Dirección web recomendada

Para saber más sobre PHP podéis consultar la siguiente dirección: http://www.php.net.

Hay que destacar la aparición de muchos frameworks que facilitan el desarrollo de aplicaciones web, a la vez que integran las funcionalidades de generación de contenido y su almacenaje. Por ejemplo, Cake, Rails, etc.

5.2.5. RIA

RIA (Rich Internet Applications) es un acrónimo que engloba una gran multitud de términos que definen una serie de aplicaciones cuyos contenidos son dinámicos y se cargan en el tiempo de inicialización. La aplicación sólo hace consultas al servidor para obtener datos de las bases de datos mientras que las herramientas (reproductores de vídeo, procesamiento de imágenes) ya están cargadas en el lado del cliente. Eso hace que se puedan ofrecer funcionalidades mucho más ricas y entornos multimedia más conseguidos.

Se puede decir que las RIA son la nueva generación de las aplicaciones y una tendencia ya impuesta por empresas como Macromedia, Magic Software, Sun o Microsoft, que están desarrollando recursos para hacer de este tipo de aplicaciones una realidad. Estas aplicaciones están basadas en plataformas J2EE o.NET, con un front-end Flash, Java Swing, Java FX o Google Web Toolkit, y utilizan una arquitectura cliente/servidor asíncrona, segura y escalable, junto con una interfaz de usuario web. Entre los beneficios principales de las aplicaciones RIA tenemos una mejora importante en la experiencia visual, que hacen del uso de la aplicación alguna cosa muy sencillo, mejoras en la conectividad y despliegue instantáneo de la aplicación, agilizando su acceso, garantizan la desvinculación de la capa de presentación, es decir, el acceso a la aplicación desde cualquier computador en cualquier lugar del mundo. Un framework que permite desarrollar de forma sencilla este tipo de aplicaciones es Google Web Toolkit.

5.3. Protocolos de seguridad

Los protocolos definen las reglas y las normas que utilizan los ordenadores para comunicarse con la red. Internet es un canal inseguro para enviar información. Cuando se tiene que enviar un mensaje por Internet, se le hace pasar por numerosos nodos intermedios antes de llegar a su destino. Alguno de estos nodos intermedios podría interceptar, leer, destruir o modificar la información enviada.

Muchas veces, durante el proceso de diseño de una aplicación la seguridad es un aspecto que se deja de lado y se acaba añadiendo con posterioridad. En muchos casos, estos añadidos no han contemplado todos los posibles problemas y, por lo tanto, hacen que la aplicación pueda ser vulnerable.

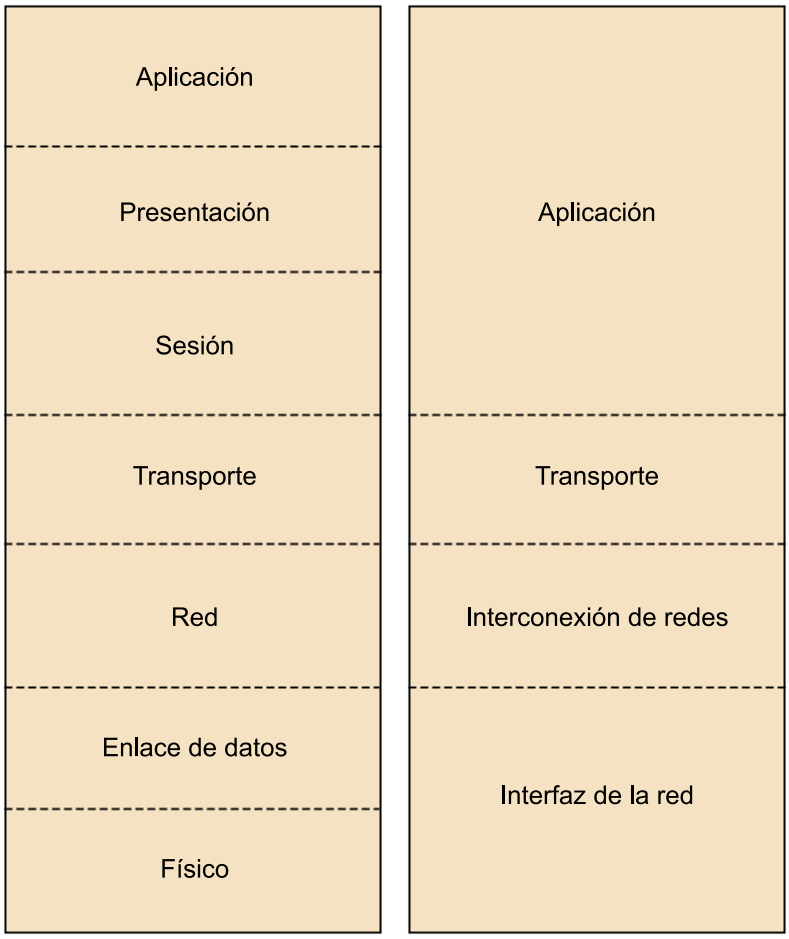
En un principio, en el modelo OSI[17] de comunicaciones, se decidió poner todo lo que se refiere a seguridad en el nivel de presentación. De esta manera, se podría ofrecer este servicio a todas las aplicaciones. No obstante, el modelo OSI no tuvo éxito y, por lo tanto, esta solución no ha sido usada.

(17)OSI es la abreviatura de Open Systems Interconnection, en castellano, interconexión de sistemas abiertos.

Si analizamos el modelo Internet, basado en el protocolo TCP/IP, veremos que no existe ningún nivel de presentación, pero podemos llegar a establecer una cierta equivalencia entre ambos modelos que nos será útil para señalar los

sistemas de seguridad que podemos establecer en cada uno de los niveles. En el nivel de aplicación podemos implementar aquellos servicios de seguridad que sean específicos de cada aplicación.

Comparación del modelo OSI y la arquitectura TCP/IP



El servicio de seguridad pasa a través de aplicaciones intermedias. Por ejemplo, Pretty Good Privacy (PGP), Secure Electronic Transactions (SET), S/MIME, etc. Por debajo del nivel de aplicación, podemos llegar a intercalar servicios de seguridad como Secure Sockets Layer (SSL), Transporte Layer Secure (TLS), etc.

Entre los niveles TCP y IP, podríamos establecer medidas de seguridad transparentes en las aplicaciones. Ejemplo: IPSEC. Incluso por debajo del nivel IP, podemos llegar a cifrar las cabeceras IP, pero ello tiene el inconveniente de que, si no se realiza correctamente el descifrado, los datos no llegarán a su destino.

5.3.1. PGP

Existen herramientas basadas en algoritmos criptográficos que permiten proteger la información que se intercambia a través de la red. Pretty Good Privacy (PGP) es un protocolo que permite cifrar (encriptar) ficheros y mensajes de correo electrónico de manera que sólo puedan acceder a ellos los usuarios que determinemos. De forma adicional, PGP permite realizar una firma digital de

los ficheros y mensajes, con lo que se puede garantizar la integridad de los mismos. Mediante este programa podemos llevar la gestión de claves, cifrar y firmar digitalmente mensajes y ficheros, borrado seguro de ficheros, etc.

Resumiendo, sus dos usos más comunes son los siguientes:

- Garantizar que un fichero informático (mensaje de correo electrónico, fichero de texto, hoja de cálculo, imagen, etc.) ha sido creado por quien dice ser su creador (firma digital).

- Impedir que personas sin autorización lean un fichero informático (encriptación).

No obstante, aunque tiene otras funciones, nos centraremos en las dos formas más comunes de uso de PGP, la firma digital y la encriptación. Para usar cualquiera de estas funciones, el usuario de PGP tiene que crear una pareja de claves (una pública y otra privada), que son la base sobre la que se sostiene la criptografía de claves públicas, basada en los algoritmos de clave asimétrica. Crear la pareja de claves es muy sencillo, ya que es la aplicación PGP la que se encarga prácticamente de todo.

Para que se pueda utilizar PGP, los interlocutores tendrán que tener instalado el programa. Al instalarse el programa se generan unas claves. Una de las claves que forman la pareja es la clave privada, a la que siempre hay que proteger contra los accesos no autorizados, ya que toda nuestra seguridad se basa en que nadie más tenga acceso a ella. Para reducir su vulnerabilidad, la clave privada está protegida por una contraseña compleja en forma de frase que es mucho más segura que una contraseña típica de menos de diez caracteres. Su utilidad es desencriptar los mensajes que nos sean enviados de forma segura. Por el contrario, la clave pública hay que distribuirla a todas las personas con las que queramos mantener comunicaciones seguras. Podría parecer que distribuir libremente nuestra clave pública es una forma de bajar nuestras defensas, pero no es así: la clave pública está encriptada y, además, su única utilidad es cifrar (encriptar) los mensajes y ficheros que los otros nos quieran enviar. Con ella es imposible desencriptar los mensajes o ficheros que alguien nos haya enviado; tampoco es posible hacerse pasar por nosotros firmando un fichero. Para eso sería preciso disponer de la clave privada.

PGP se puede integrar en los programas de correo electrónico más usuales para hacer que encriptar y firmar mensajes no consista más que en pulsar un botón. Al encriptar un mensaje, se nos pedirá que indiquemos a quién lo enviaremos, para así escoger la clave pública correcta con que cifrarlo. Para firmar, en cambio, se nos pedirá que tecleemos la contraseña de nuestra clave privada, con lo cual evitamos también que alguien que use nuestro ordenador en nuestra ausencia pueda hacerse pasar por nosotros.

Para cifrar los datos se utiliza un algoritmo de clave simétrica, cuya clave se cifra con un algoritmo de clave asimétrica. De esta manera se combinan las mejores propiedades de ambos: la seguridad de un algoritmo asimétrico (donde clave pública y privada son diferentes) con la rapidez y robustez de un algoritmo simétrico (cuya clave es única y, por lo tanto, vulnerable). Un tercer algoritmo se utiliza para firmar documentos: se extrae un conjunto de bits del mensaje (resumen) con una función *hash* y se cifra con la clave privada del emisor. Así, por su modo de funcionamiento, PGP (y programas similares), consta de los tres subsistemas: cifrado del documento con clave asimétrica, cifrado con clave simétrica y firma digital.

Cuando se desea enviar un correo o fichero encriptado de *B* a *A*, PGP lo encripta usando un sistema simétrico (generalmente IDEA o DES), usando una clave aleatoria –clave DES–, que posteriormente se encripta (por ejemplo, con RSA con la clave pública de *A*, KAp). Se envían el documento cifrado con la clave aleatoria y ésta encriptada con la clave RSA privada del destinatario.

Cuando *A* recibe el correo y desea desencriptarlo, su programa PGP descifra primero la clave simétrica con su clave privada RSA (Kas), y después descifra el documento usando la clave desencriptada –clave DES–.

5.4. SSL

Secure Socket Layer es un sistema de protocolos de carácter general diseñado en 1994 por la empresa Nestcape Communications Corporation, basado en la aplicación conjunta de criptografía simétrica, criptografía asimétrica (de clave pública), certificados digitales y firmas digitales, para ofrecer conexiones seguras a través de Internet. Este grupo de protocolos comprende:

- El protocolo de transporte Secure Sockets Layer (SSL), desarrollado por Netscape Communications a principios de los años noventa. La primera versión de este protocolo, sobradamente difundida e implementada, fue la 2.0. Poco después, Netscape publicó la versión 3.0, con muchos cambios con respecto a la anterior, que hoy casi ya no se utiliza.

- La especificación Transport Layer Security (TLS), elaborada por la Internet Engineering Task Force (IETF). La versión 1.0 del protocolo TLS está publicada en el documento RFC 2246. Es prácticamente equivalente a SSL 3.0 con algunas pequeñas diferencias, por lo cual en ciertos contextos se considera el TLS 1.0 como si fuera el protocolo SSL 3.1.

- El protocolo Wireless Transport Layer Security (WTLS), perteneciente a la familia de protocolos Wireless Application Protocol (WAP) para el acceso a la red desde dispositivos móviles. La mayoría de los protocolos WAP son adaptaciones de los ya existentes a las características de las comunicaciones sin hilos, y en particular el WTLS está basado en el TLS 1.0. Las diferencias se centran principalmente en aspectos relativos al uso eficiente del

ancho de banda y de la capacidad de cálculo de los dispositivos, que puede ser limitada.

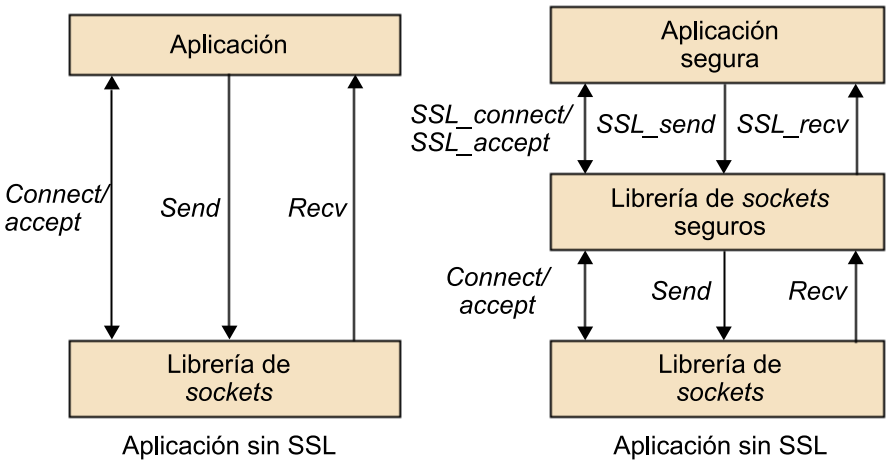
5.4.1. Características del protocolo SSL/TLS

El objetivo inicial del diseño del protocolo SSL fue proteger las conexiones entre clientes y servidores web con el protocolo HTTP. Esta protección tenía que permitir al cliente asegurarse de que se había conectado al servidor auténtico, y enviarle datos confidenciales, como, por ejemplo, un número de tarjeta de crédito, con la certeza de que nadie más que el servidor es capaz de ver estos datos.

Las funciones de seguridad, sin embargo, no se implementaron directamente en la protección de aplicación HTTP, sino que se optó por introducirlas en el nivel de transporte. Así podría haber muchas más aplicaciones que hicieran uso de esta funcionalidad.

A tal fin se desarrolló una interfaz de acceso a los servicios del nivel de transporte basada en la interfaz estándar de los *sockets*. En esta nueva interfaz, funciones como *connect*, *accept*, *send* o *recv* fueron sustituidas por otras equivalentes pero que utilizaban un protocolo de transporte seguro: *SSL_connect*, *SSL_accept*, *SSL_send*, *SSL_recv*, etc. El diseño se hizo de manera tal que cualquier aplicación que utilizara TCP a través de los llamamientos de los *sockets* podía hacer uso del protocolo SSL sólo cambiando estos llamamientos. De aquí viene el nombre del protocolo.

Diseño del protocolo SSL



Datagramas en WTLS

Una característica distintiva del WTLS es que no solamente permite proteger conexiones TCP, como hacen SSL y TLS, sino que también define un mecanismo de protección para las comunicaciones en modo datagrama, usado en diversas aplicaciones móviles.

Los servicios de seguridad que proporcionan los protocolos SSL TLS son:

- **Confidencialidad**. El flujo normal de información en una conexión SSL TLS consiste en intercambiar paquetes con datos cifrados mediante claves simétricas (por motivos de eficiencia y rapidez). Al inicio de cada sesión, cliente y servidor se ponen de acuerdo sobre las claves que utilizarán para cifrar los datos. Siempre se utilizan dos claves diferentes: una para los paquetes enviados por el cliente al servidor, y la otra para los paquetes enviados en sentido contrario. Para evitar que un intruso que esté escuchando el diálogo inicial pueda saber cuáles son las claves acordadas, se sigue un mecanismo seguro de intercambio de claves, basado en criptografía de clave pública. El algoritmo concreto para este intercambio también se negocia durante el establecimiento de la conexión.

- **Autenticación de entidad**. Con un protocolo basado en firmas digitales, el cliente puede confirmar la identidad del servidor al que se ha conectado. Para validar las firmas, el cliente necesita conocer la clave pública del servidor, y eso normalmente se hace a través de certificados digitales. SSL/TLS también prevé la autenticación del cliente de cara al servidor. Esta posibilidad, sin embargo, no se usa tan a menudo porque muchas veces, en vez de autenticar automáticamente al cliente a nivel de transporte, las mismas aplicaciones utilizan su propio método de autenticación.

  Autenticación de cliente

  Un ejemplo de autenticación de cliente en el ámbito de aplicación son las contraseñas que pueden introducir los usuarios en formularios HTML. Si la aplicación utiliza este método, al servidor ya no le hace falta autenticar al cliente a nivel de transporte.

- **Autenticación de mensaje**. Cada paquete enviado en una conexión SSL TLS, además de ir cifrado, puede incorporar un código MAC para que el destinatario compruebe que nadie ha modificado el paquete. Las claves secretas para el cálculo de los códigos MAC (una para cada sentido) también se acuerdan de forma segura en el diálogo inicial.

Además, los protocolos SSL TLS están diseñados con estos criterios adicionales:

- **Eficiencia**. Dos de las características de los SSL TLS, la definición de sesiones y la compresión de los datos, permiten mejorar la eficiencia de la comunicación.
  - Si el cliente pide dos o más conexiones simultáneas o muy seguidas, en lugar de repetir la autenticación y el intercambio de claves (operaciones computacionalmente costosas porque intervienen algoritmos de clave pública), existe la opción de reutilizar los parámetros previamente acordados. Si se hace uso de esta opción, se considera que la nueva conexión pertenece a la misma **sesión** que la anterior. En el establecimiento de cada conexión se especifica un **identificador de sesión**, que permite saber si la conexión empieza una sesión nueva o es continuación de otra.

– SSL TLS prevé la negociación de algoritmos de **compresión** para los datos intercambiados, para compensar el tráfico adicional que introduce la seguridad. Ni SSL 3.0 ni TLS 1.0, sin embargo, especifican ningún algoritmo concreto de compresión.

Conexiones consecutivas o simultáneas

Una situación típica en la que se usa SSL TLS es la de un navegador web que accede a una página HTML que contiene imágenes: con HTTP "no persistente" (el único modo definido en HTTP 1.0), eso requiere una primera conexión para la página y, a continuación, tantas conexiones como imágenes haya. Si las conexiones pertenecen a la misma sesión SSL TLS, sólo hay que hacer la negociación una vez.

- **Extensibilidad**. Al principio de cada sesión, cliente y servidor negocian los algoritmos que utilizarán para el intercambio de claves, la autenticación y el cifrado (además del algoritmo de compresión). Las especificaciones de los protocolos incluyen unas combinaciones predefinidas de algoritmos criptográficos, pero dejan abierta la posibilidad de añadir nuevos algoritmos si se descubren otros que sean más eficientes o más seguros.

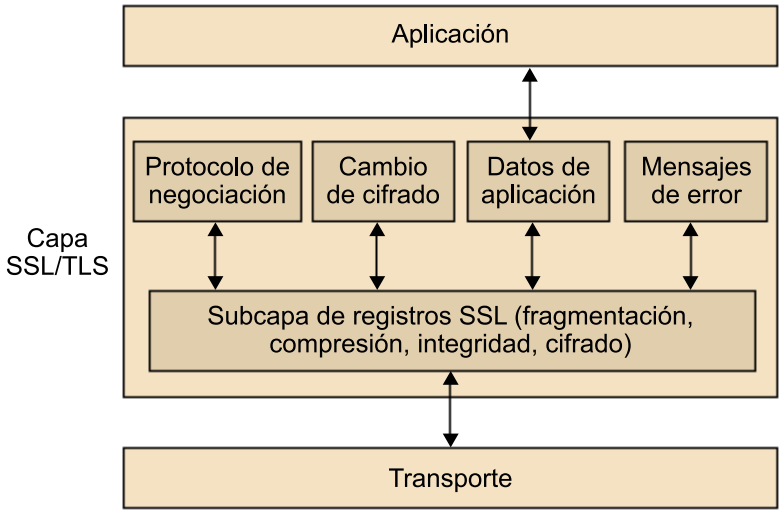
5.4.2. El transporte seguro SSL/TLS

La capa de transporte seguro que proporciona SSL/TLS se puede considerar dividida en dos subcapas.

- La subcapa superior se encarga básicamente de negociar los parámetros de seguridad y transferir los datos de la aplicación. Tanto los datos de negociación como los de aplicación se intercambian en **mensajes**.

- En la subcapa inferior, estos mensajes son estructurados en **registros** a los cuales se aplica, según corresponda, la compresión, la autenticación y el cifrado.

Estructura de la capa SSL/TLS



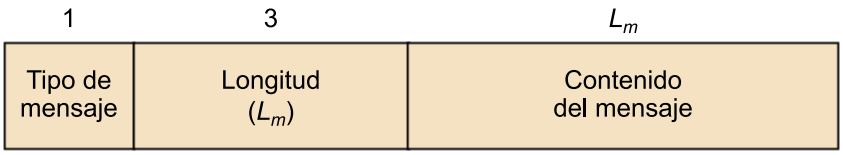
El **protocolo de registros SSL TLS** es el que permite que los datos protegidos sean convenientemente codificados por el emisor e interpretados por el receptor. Los parámetros necesarios para la protección, como los algoritmos y las claves, se establecen de forma segura al inicio de la conexión mediante el **protocolo de negociación SSL/TLS**.

El protocolo de negociación SSL/TLS

El protocolo de negociación SSL TLS, también llamado **protocolo de apretón de manos** (Handshake Protocol), tiene por finalidad autenticar el cliente y/o el servidor, y acordar los algoritmos y claves que utilizarán de una manera segura, es decir, garantizando la confidencialidad y la integridad de la negociación.

Como todos los mensajes SSL/TLS, los mensajes del protocolo de negociación se incluyen dentro del campo de datos de los registros SSL/TLS para ser transmitido al destinatario. La estructura de un mensaje de negociación es la que se muestra en la figura siguiente.

Formato de los mensajes de negociación SSL/TLS



El contenido del mensaje tendrá unos determinados campos dependiendo del tipo de mensaje de negociación de que se trate. En total hay 10 tipos diferentes, que veremos a continuación en el orden en que se tienen que enviar.

1) Petición de saludo (*Hello Request*). Cuando se establece una conexión, el servidor normalmente espera que el cliente inicie la negociación. Alternativamente, puede optar por enviar un mensaje *Hello Request* para indicar al cliente que está preparado para empezar. Si durante la sesión el servidor quiere iniciar una renegociación, también lo puede indicar al cliente enviándole un mensaje de este tipo.

2) Saludo de cliente (*Client Hello*). El cliente envía un mensaje *Client Hello* al inicio de la conexión o como respuesta a un *Hello Request*. Este mensaje contiene la siguiente información:

- La versión del protocolo que el cliente quiere utilizar.
- Una cadena de 32 bytes aleatorios[18].
- Opcionalmente, el identificador de una sesión anterior, si el cliente desea volver a utilizar los parámetros que se le concedieron.

[18]De los 32 bytes aleatorios que se envían en los mensajes de saludo, los 4 primeros tienen que ser una marca de tiempo, con precisión de segundos.

- La lista de las combinaciones de algoritmos criptográficos que el cliente ofrece utilizar, por orden de preferencia. Cada combinación incluye el algoritmo de cifrado, el algoritmo de MAC y el método de intercambio de claves.

- La lista de los algoritmos de compresión ofrecidos, por orden de preferencia (como mínimo tiene que haber uno, aunque sea el algoritmo nulo).

Algoritmos de compresión

El único algoritmo de compresión previsto en SSL TLS es el algoritmo nulo, es decir, ninguna compresión.

Algoritmos criptográficos previstos en SSL/TLS

SSL TLS contempla los algoritmos criptográficos siguientes:

- Cifrado: RC4, DES, Triple DES, RC2, IDEA y FORTEZZA (este último sólo en SSL 3.0).
- MAC: MD5 y SHA-1.
- Intercambio de claves: RSA, Diffie-Hellman y FORTEZZA KEA (este último sólo en SSL 3.0).

Si sólo interesa autenticar la conexión, sin confidencialidad, también se puede usar el algoritmo de cifrado nulo.

3) Saludo de servidor (*Server Hello*). Como respuesta, el servidor envía un mensaje *Server Hello*, que contiene esta información:

- La versión del protocolo que se utilizará en la conexión. La versión será igual a la que envió el cliente, o inferior si ésta no es soportada por el servidor.

- Otra cadena de 32 bytes aleatorios.

- El identificador de la sesión actual. Si el cliente envió uno y el servidor quiere reanudar la sesión correspondiente, tiene que responder con el mismo identificador. Si el servidor no quiere reanudar la sesión (o no puede porque ya no guarda la información necesaria), el identificador enviado será diferente. Opcionalmente, el servidor puede no enviar ningún identificador para indicar que la sesión actual ya no podrá ser reanudada.

- La combinación de algoritmos criptográficos escogida por el servidor en la lista de las enviadas por el cliente. Si se reanuda una sesión anterior, esta combinación tiene que ser la misma que se utilizó entonces. El algoritmo de compresión escogido por el servidor, o el que se utilizó en la sesión que se reanuda.

Si se ha decidido continuar una sesión anterior, cliente y servidor ya pueden empezar a utilizar los algoritmos y claves previamente acordados y se saltan los mensajes que vienen a continuación, pasando directamente a los de finalización de la negociación (mensajes *finished*).

4) Certificado de servidor (*certificate*) o intercambio de claves de servidor (*server key exchange*). Si el servidor puede autenticarse delante del cliente, que es el caso más habitual, envía el mensaje *certificate*. Este mensaje normalmente

contendrá el certificado X.509 del servidor, o una cadena de certificados. Si el servidor no tiene certificado, o se ha acordado un método de intercambio de claves que no utiliza, tiene que enviar un mensaje *server key exchange*, que contiene los parámetros necesarios para el método a seguir.

5) Petición de certificado (*certificate request*)

- Tipo de certificados: En SSL TLS están contemplados los certificados de clave pública RSA, DSA o FORTEZZA KEA (este último tipo sólo en SSL 3.0). En el caso de que se tenga que realizar también la autenticación del cliente, el servidor le envía un mensaje *Certificate Request*. Este mensaje contiene una lista de los posibles tipos de certificado que el servidor puede admitir, por orden de preferencia, y una lista de los DN de las autoridades de certificación que el servidor reconoce.

6) Fin de saludo de servidor (*Server Hello* Done). Para acabar esta primera fase del diálogo, el servidor envía un mensaje *Server Hello Done*.

7) Certificado de cliente (*Certificate*)

- Cliente sin certificado: Si el cliente recibe una petición de certificado pero no tiene ninguno apropiado, en SSL 3.0 tiene que enviar un mensaje de aviso, pero en TLS 1.0 tiene que enviar un mensaje *Certificate* vacío. En cualquier caso, el servidor puede responder con un error fatal, o bien continuar sin autenticar al cliente. Una vez el servidor ha enviado sus mensajes iniciales, el cliente ya sabe cómo continuar el protocolo de negociación. En primer lugar, si el servidor le ha pedido un certificado y el cliente tiene alguno de las características solicitadas, lo envía en un mensaje *Certificate*.

8) Intercambio de claves de cliente (*Client Key Exchange*). El cliente envía un mensaje *Client Key Exchange*, cuyo contenido depende del método de intercambio de claves acordado. En el caso de seguir el método RSA, en este mensaje hay una cadena de 48 bytes que se utilizará como **secreto pre-maestro**, cifrada con la clave pública del servidor.

Un posible ataque contra la negociación es modificar los mensajes para que las dos partes acuerden utilizar el protocolo SSL 2.0, que es más vulnerable. Para evitar este ataque, en los dos primeros bytes del secreto pre-maestro tiene que estar el número de versión que se envió en el mensaje *Client Hello*. Entonces, cliente y servidor calculan el llamado **secreto maestro**, que es otra cadena de 48 bytes. Para hacer este cálculo, se aplican funciones *hash* al secreto pre-maestro y a las cadenas aleatorias que se enviaron en los mensajes de saludo. A partir del secreto maestro y las cadenas aleatorias, se obtienen:

- Las dos claves para el cifrado simétrico de los datos (una para cada sentido: de cliente a servidor y de servidor a cliente).

- Las dos claves MAC (también una para cada sentido).

- Los dos vectores de inicialización para el cifrado, si se utiliza un algoritmo de bloque.

9) Verificación de certificado (*Certificate Verify*). Si el cliente ha enviado un certificado en respuesta a un mensaje *Certificate Request*, ya puede autenticarse demostrando que posee la clave privada correspondiente mediante un mensaje *Certificate Verify*. Este mensaje contiene una firma, generada con la clave privada del cliente, de una cadena de bytes obtenida a partir de la concatenación de todos los mensajes de negociación intercambiados hasta ahora, desde el *Client Hello* hasta el *Client Key Exchange*.

10) Finalización (*Finished*). A partir de este punto ya se pueden utilizar los algoritmos criptográficos negociados. Cada parte envía a la otra una notificación de cambio de cifrado seguida de un mensaje *Finished*. La notificación de cambio de cifrado sirve para indicar que el siguiente mensaje será el primero enviado con los nuevos algoritmos y clavos.

El mensaje *Finished* sigue inmediatamente a la notificación de cambio de cifrado. Su contenido se obtiene aplicando funciones *hash* al secreto maestro y a la concatenación de todos los mensajes de negociación intercambiados, desde el *Client Hello* hasta el anterior a éste (incluido el mensaje *Finished* de la otra parte, si ya lo ha enviado). Normalmente será el cliente el primero en enviar el mensaje *Finished*, pero en el caso de reanudar una sesión anterior, será el servidor quien lo envíe primero, justo después del *Server Hello*.

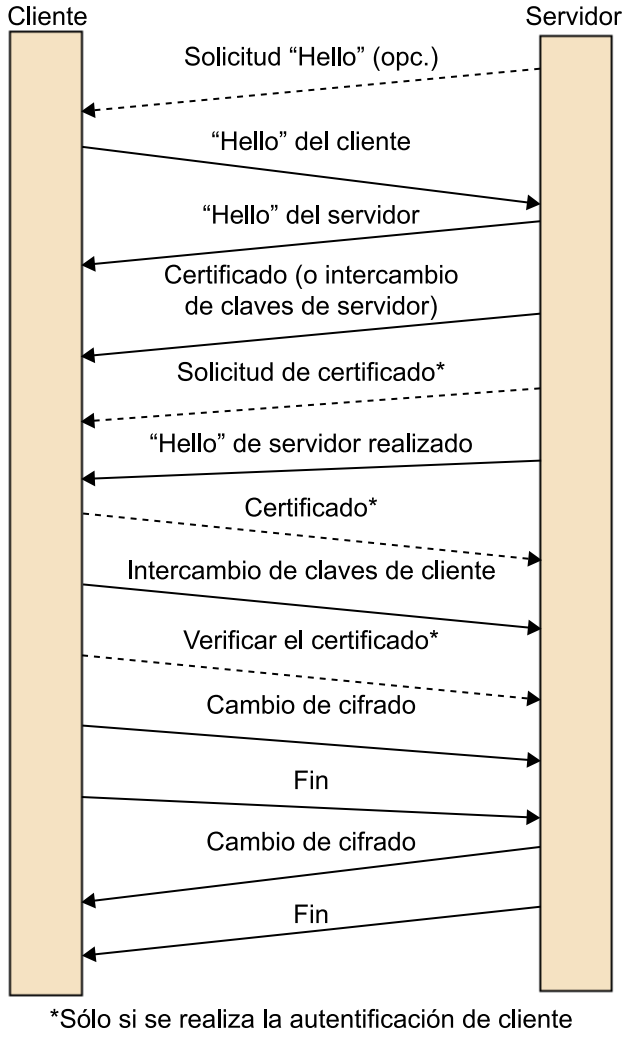
Una de las principales diferencias entre SSL 3.0 y TLS 1.0 está en la técnica usada para obtener los códigos de verificación de los mensajes *Finished*, y también para calcular el secreto maestro y obtener las claves a partir de este secreto (en SSL se utilizan funciones *hash* directamente, y en TLS se utilizan códigos HMAC).

El contenido del mensaje *Finished* sirve para verificar que la negociación se ha llevado a cabo correctamente. Este mensaje también permite autenticar al servidor delante del cliente, ya que el primero necesita su clave privada para descifrar el mensaje *Client Key Exchange* y obtener las claves que se utilizarán en la comunicación.

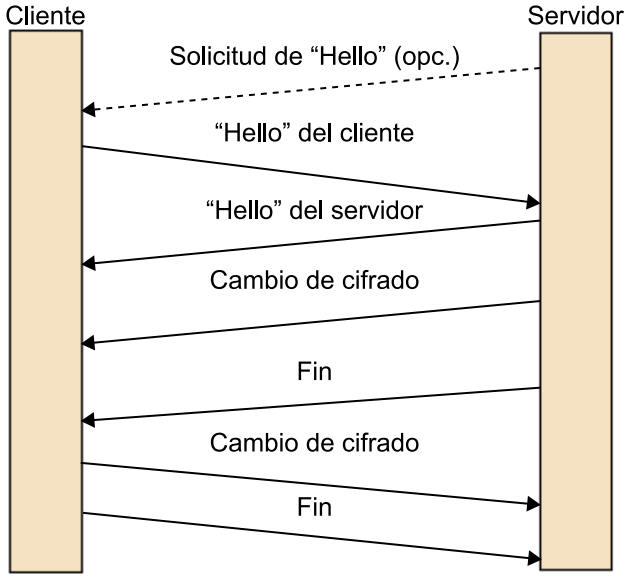
Una vez enviado el mensaje *Finished*, se da por acabada la negociación, y cliente y servidor pueden empezar a enviar los datos de aplicación utilizando los algoritmos y claves acordados.

Los siguientes diagramas resumen los mensajes intercambiados durante la fase de negociación SSL TLS.

Negociación de una sesión SSL/TLS nueva



Negociación de una sesión SSL/TLS que se reanuda



Además de los mensajes de negociación, notificaciones de cambio de cifrado y datos de aplicación, también se pueden enviar mensajes de error. Estos mensajes contienen un código de nivel de gravedad, que puede ser "mensaje de aviso" o " error fatal", y un código de descripción del error. Un error fatal provoca el fin de la conexión y la invalidación del identificador de sesión correspondiente, es decir, la sesión no podrá ser reanudada.

Ejemplos de errores fatales

Son ejemplos de errores fatales: MAC incorrecto, tipo de mensaje inesperado, error de negociación, etc. (TLS 1.0 prevé más códigos de error que SSL 3.0).

También se puede enviar un mensaje de aviso para indicar el fin normal de la conexión. Para evitar ataques de truncamiento, si una conexión acaba sin haber enviado este aviso se invalidará su identificador de sesión.

5.5. Transacciones seguras en red

Conviene conocer también diferentes iniciativas para permitir las transacciones seguras en internet. Presentamos las más importantes.

5.5.1. Secure Electronic Transaction

A raíz de las faltas del protocolo SSL, diferentes empresas y organismos buscaron un sistema que permitiera realizar operaciones sensibles por Internet de forma segura, como los pagos, con el fin de estimular la confianza de los consumidores en el comercio electrónico. En 1996 un grupo de empresas del sector financiero, informático y de seguridad (Visa International, MasterCard, Microsoft, Nestcape, IBM, RSA, etc.) anunció el desarrollo de esta nueva tecnología.

Secure Electronic Transaction[19] (SET) es un protocolo que permite dar seguridad en las transacciones por Internet que utilizan tarjetas de crédito. Sus especificaciones se pueden encontrar en el sitio web oficial de SETco, organismo encargado de homologar los módulos de programación y los certificados desarrollados por empresas privadas que se usan en implementaciones del protocolo SET.

[19]En castellano, transacciones electrónicas seguras.

Dirección web recomendada

Podéis acceder al histórico de la página web de SETco en:
http://web.archive.org/web/*/www.setco.org

Su funcionamiento se basa en el uso de certificados digitales y sistemas criptográficos. Los primeros, para asegurar la perfecta identificación de todas aquellas parejas que intervienen en una transacción en línea basada en el uso de tarjetas de pago, y los segundos, para proteger el envío de los datos sensibles en su viaje entre los diferentes servidores que participan en el proceso. Una de sus características es la de separar el proceso de compra del de pago.

Como inconvenientes de la SET mencionaremos su complejidad y lentitud. Normalmente, con SSL no se necesita certificado ni software adicional, sólo seleccionar los productos que hay que comprar y aceptar el pago. La lentitud de la SET se debe a que se han de realizar diferentes verificaciones de identidad e integridad por parte de diversas entidades a lo largo de una transacción.

5.5.2. 3D-Secure

3D-Secure es un sistema promovido por Visa/Mastercard con la finalidad de incentivar el comercio electrónico, que intenta evitar los fraudes cuando el pago se realiza con tarjetas de crédito. Está basado en el modelo de los "tres dominios", en el cual se requiere:

1) Que la entidad emisora de la tarjeta autentique al cliente o titular de la tarjeta (comprador): dominio del emisor.

2) Que la entidad adquirente autentique al comercio (vendedor): dominio del adquirente.

3) Que ambas partes (entidad emisora y entidad adquirente) sean reconocidas mutuamente como legítimas para efectuar la transacción, y que ésta se complete de forma segura: dominio de intercambio.

En este modelo se deja al arbitrio de cada una de las entidades la elección del "procedimiento de autenticación". De esta manera, tienen flexibilidad para seleccionar el método de autenticación más apropiado para sus comercios y titulares, y sustituirlo por nuevas soluciones cuando lo consideren conveniente. La confidencialidad e integridad de la información de pago se consigue con la utilización del protocolo SSL. El proceso de compra con 3D Secure funciona como una compra normal en un comercio en línea, con la particularidad de que, a la hora de introducir la información de pago, el comprador tendrá que facilitar la contraseña 3D-Secure-Secure. De esta manera, se confirma que es el propio titular de la tarjeta, y no un tercero sin autorización, quien efectúa la transacción.

Resumen

El módulo ha hecho una introducción a los conceptos básicos de la seguridad en las redes de comunicación. Los **cortafuegos** se han presentado como medidas pasivas de seguridad en los sistemas en red. Hemos visto que un cortafuegos puede ser tanto hardware como software y que incluso se pueden combinar diferentes sistemas para asegurar una red.

Se ha presentado el concepto de red privada virtual (VPN), que no es más que una red lógica o virtual creada sobre una infraestructura compartida, pero que proporciona los servicios de protección necesarios para una comunicación segura. Las **redes privadas virtuales** (VPN) permiten utilizar la red pública Internet como si fuera una red privada dedicada, por ejemplo, entre diversas intranets de una misma organización. La técnica básica que utilizan las VPN son los **túneles**, en los que los paquetes protegidos se encapsulan dentro de datagramas IP que circulan de manera normal por la red Internet.

En este módulo también hemos visto que las **técnicas criptográficas** permiten cifrar un texto mediante una **clave de cifrado**, y que basta con conocer la **clave de descifrado** correspondiente para ser capaz de obtener el texto original.

Según la relación que haya entre las dos claves, los algoritmos criptográficos se clasifican en **algoritmos simétricos** si la clave de cifrado y la de descifrado son la misma, y **algoritmos de clave pública** si las claves son diferentes. Los algoritmos simétricos, a su vez, se pueden clasificar en **algoritmos de flujo**, si el cifrado consiste en añadir al texto datos pseudoaleatorios calculados a partir de la clave, o **algoritmos de bloque**, **si el cifrado se hace sobre bloques de tamaño fijo del texto original**.

La particularidad de la criptografía de clave pública es que, a partir de una de las claves, la **clave privada**, se puede deducir fácilmente la otra, la **clave pública**, mientras que la deducción inversa es prácticamente imposible. Eso permite que todo el mundo que conozca la clave pública de un usuario pueda utilizarla para cifrar datos confidenciales, con la seguridad que sólo quien tenga la clave privada podrá descifrarlos, sin necesidad de acordar ninguna clave secreta a través de un canal aparte. El uso de las claves al revés (la privada para cifrar y la pública para descifrar) es la base de las **firmas digitales**.

Como la criptografía de clave pública es computacionalmente más costosa que la simétrica, no se utiliza nunca directamente para obtener confidencialidad, sino siempre a través de una clave de sesión simétrica. De la misma manera, la

firma de un texto no se calcula directamente a partir del texto, sino aplicando una función *hash* segura. La propiedad de este tipo de función es que es muy difícil encontrar un mensaje que dé el mismo *hash* que otro.

Después de esta aproximación más teórica y para concluir el módulo se han presentado diferentes conceptos de seguridad de la red en los niveles de aplicación y transporte. Hemos profundizado en el conocimiento de un protocolo, el SSL/TLS, de la capa de transporte que sirve para dotar de seguridad a las comunicaciones de las aplicaciones en red. El uso típico de los protocolos SSL TLS es para proteger de manera transparente un protocolo de aplicación como es HTTP. El protocolo **HTTPS** es simplemente la combinación de HTTP con el transporte seguro SSL/TLS.

Bibliografía

Menezes, A. J.; Oorschot, P. C. van; Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. Boca Ratón: CRC Press.

SETco. Disponible en web.
<http://web.archive.org/web/*/www.setco.org>. [Fecha de consulta: 12 de abril del 2010.]

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

Yuan, R.; Strayer, W. T. (2001). *Virtual Private Networks, Technologies and Solutions*. Boston: Addison-Wesley.