Servicios Telemáticos

Bibliografía:

- Diapositivas de clase, Universidad de Murcia.
- RFCs 2045, 2046 y 2821 (https://www.ietf.org/standards/rfcs/).
- Wikipedia, the Free Encyclopedia. *IPSec* (https://en.wikipedia.org/wiki/IPsec).

Capítulo 1

Introducción

Internet es una red de redes formada por ISPs que, mediante enlaces de comunicación (fibra, cobre, ondas de radio, etc.) con un cierto ancho de banda y dispositivos de conmutación como *routers* y *switches* que reenvían paquetes (bloques de datos), interconectan *hosts* o dispositivos anfitriones que ejecutan aplicaciones de red en el extremo de la red a través de accesos residenciales, institucionales (centros educativos, empresas, etc.) o móviles, e interconectan sus distintas zonas entre sí y con el resto de ISPs en el núcleo de la red, una malla (*mesh*) de *routers* interconectados que enrutan paquetes de uno al siguiente entre la fuente y el destino (*forwarding*) usando algoritmos de encaminamiento para determinar el siguiente salto (*routing*). Se puede ver como una infraestructura que ofrece servicios a aplicaciones, como las de acceso a la web, voz sobre IP, correo electrónico, juegos multiusuario, comercio electrónico, redes sociales, chat, compartición de ficheros, *streaming* de vídeo, motores de búsqueda, etc., para conectarse entre sí mediante una interfaz.

Internet es muy complejo y ha evolucionado según temas económicos y políticas nacionales. Conectar cada ISP con cada uno de los demás no escala, y conectarlos todos a uno global tampoco, y además habrá competidores que se deben interconectar. El resultado es que los proveedores se interconectan mediante *internet exchange points* y *peering links*, con conexiones entre redes regionales cercanas, con redes de proveedores de contenido (*content provider networks*) que conectan centros de datos para ofrecer servicios y contenidos cerca del usuario final, e ISPs comerciales *tier-I* (Level 3, Sprint, AT&T, NTT, etc.) con cobertura nacional e internacional.

Otra alternativa sería la conmutación de circuitos, usada en la red telefónica, en que se dedican recursos de extremo a extremo para una llamada entre la fuente y el destino. Esto da un rendimiento garantizado y evita los retardos en los paquetes y las pérdidas debidas a la posibilidad de congestión, pero es más complejo y evita la compartición de recursos. El suministro de un ancho de banda garantizado para aplicaciones de audio y vídeo en conmutación de paquetes está sin resolver.

La comunicación se hace siguiendo protocolos para el envío y recepción de mensajes, bien abiertos, muchos recogidos en **RFCs** (*Requests for Comments*) de la **IETF** (*Internet Engineering Task Force*), o privativos, y divididos en niveles:

1. Aplicación: Aplicaciones de red. FTP, SMTP, HTTP. Los protocolos definen los tipos de mensajes intercambiados, su sintaxis (qué campos hay y cómo se delimitan), su semántica

y las reglas de cuándo y cómo los procesos envían y reciben mensajes. Tipos:

   a) **Cliente-servidor**: Los clientes, que pueden estar intermitentemente conectados y tener una IP dinámica, solo se comunican con el servidor, siempre activo y en línea con IP fija y muchas veces en un centro de datos por escalabilidad.

   b) **P2P**: No hay un servidor siempre en línea, sino que los sistemas finales se comunican directamente, solicitando y ofreciendo servicios a otros equipos. Esto es escalable, pues nuevos equipos añaden demanda pero también capacidad, pero su gestión es compleja porque los equipos están intermitentemente conectados y cambian de IP.

2. Transporte: Transferencia de datos entre procesos de distintos *hosts*. TCP y UDP.

   Dos procesos en un mismo *host* se comunican por el sistema operativo. Hay procesos clientes, que inician una comunicación, y servidores, que esperan a ser conectados, y las aplicaciones P2P tienen de ambos tipos. Los procesos envían y reciben mensajes por un *socket*, para lo cual llevan asociados un identificador, formado por la dirección IP del *host* y números de puerto asociados con el proceso en el *host*. Por ejemplo, los servidores HTTP usan en general el puerto 80, y los SMTP, el 25.

3. Red: Enrutamiento de paquetes de la fuente al destino. IP, protocolos de enrutamiento.

4. Enlace: Transferencia de datos entre elementos de red vecinos. Ethernet, 802.11 (WiFi), PPP.

5. Físico: Bits en el cable.

Internet se diseñó pensando en usuarios que confiaban los unos con los otros, pero hoy en día en necesaria seguridad para proteger las comunicaciones. TCP y UDP no ofrecen cifrado, sino que este se hace a nivel de aplicación.

El comando `telnet` *`host`* [*`puerto`*] abre una conexión TCP con el puerto indicado del host (por defecto el 23), que en general podemos considerar cruda, permitiendo interactuar con servidores de forma interactiva.

Capítulo 2

HTTP

Una **página web** es un fichero HTML que puede tener referencias a objetos: otros ficheros HTML, imágenes, vídeos, etc., cada uno direccionable a través de una **URL**, con formato *`protocolo`*`://`*`host`*`/`*`ruta`*.

HTTP, *Hypertext Transfer Protocol*, es el protocolo usado en la web, en el que un cliente, normalmente un navegador, solicita objetos web a un servidor y este se los envía en respuesta, usando TCP normalmente en el puerto 80. Es un protocolo sin estado, en el que el servidor no mantiene información sobre peticiones de cliente anteriores. Los protocolos con estado persistente requieren que el servidor almacene información de forma permanente, y si el cliente o el servidor caen, sus vistas del estado pueden quedar inconsistentes y deben resincronizarse.

Una conexión HTTP es **no persistente** si, una vez se envía un objeto, se cierra la conexión, requiriendo 2 RTTs por objeto más el tiempo de transmisión del fichero y sobrecargado el sistema operativo al necesitar varias conexiones TCP, normalmente paralelas, para traer los objetos. Es **persistente** si la misma conexión se puede usar para varios objetos, en cuyo caso normalmente el cliente envía la petición por la conexión tan pronto encuentra una referencia (*pipeline*) y solo es necesario un RTT por objeto referenciado.

Los mensajes de solicitud están formados por una línea *`MÉTODO /ruta`* `HTTP/`*`versión`*, líneas de cabecera y un cuerpo opcional, y los de respuesta tienen el mismo formato salvo que la primera línea es `HTTP/`*`versión estado descripción`*. En la versión `1.0`, los métodos son:

`GET` Solicita un objeto. No incluye cuerpo.

`POST` Envía datos al servidor en el cuerpo.

`HEAD` Como `GET`, pero indica al servidor que no incluya el cuerpo en la respuesta.

Al rellenar un formulario en una página web, se pasa a la página indicada en el formulario indicando al servidor los datos introducidos. Esto se puede hacer mediante `POST`, con los datos en el cuerpo, en cuyo caso la página no se añade al historial del navegador, o mediante `GET`, en cuyo caso los datos se añaden al final de la URL separado por `?`. Así, `GET` permite al usuario conservar las consultas realizadas en el historial, pero `POST` no está restringido por el tamaño de URL máximo soportado por el servidor.

El tipo MIME `application/x-www-form-urlencoded` se refiere al formato usado para estos datos en peticiones `GET` y, normalmente, también en peticiones `POST`. En este, se envían una

serie de asociaciones *clave*=*valor* separadas por &, donde los espacios se sustituyen por + y los caracteres =, &, +, % y no imprimibles se sustituyen por %XX, donde XX es la representación del byte en hexadecimal en mayúscula. Cuando hay varios tipos de datos en POST, se usa el tipo MIME multipart/form-data.

La versión 1.1 de HTTP incluye también los métodos:

PUT Carga el fichero en el cuerpo del mensaje en el directorio indicado en la URL.

DELETE Elimina el fichero indicado. No incluye cuerpo.

Los estados de respuesta tienen un código numérico y van seguidos de una descripción. Algunos son:

200 OK Solicitud exitosa.

301 Moved Permanently Objeto movido a otra URL.

304 Not Modified Objeto no modificado desde la última vez.

400 Bad Request Solicitud no entendida por el servidor.

404 Not Found Objeto no encontrado.

505 HTTP Version Not Supported Versión de HTTP no soportada.

La cabecera está formada por líneas *Nombre*: *valor*, siendo cabeceras comunes en una solicitud:

Accept Formatos de objeto aceptados por el cliente.

Accept-Language Lenguajes aceptados por el cliente.

Connection Si es close, la conexión es no persistente. Si es Keep-Alive, solicita que sea persistente.

Cookie Devuelve una *cookie* al servidor.

Date Fecha actual.

Host Nombre de dominio del host. Usado para distinguir entre varios nombres de dominio con la misma IP y puerto para HTTP.

If-Modified-Since Solo se quiere el documento si se ha modificado desde la fecha indicada. De lo contrario, el servidor responde con código de estado 304.

Referer URL del documento desde el que se accedió al actual.

User-Agent Navegador del cliente.

Cabeceras comunes en una respuesta:

Content-Length Longitud en bytes del cuerpo de la respuesta.

Content-Type Tipo MIME que identifica al dato de la respuesta.

`Cookie` Solicita al cliente que almacene una *cookie*.

`Last-Modified` Fecha y hora de última modificación.

`Location` Nueva URL del documento en caso de redirección.

`MIME-Version` Versión del protocolo MIME usada por el servidor.

`Server` Nombre y versión del servidor.

El cuerpo de un mensaje está separado de la cabecera mediante una línea vacía, y las líneas se separan por retorno de carro y salto de línea (`\r\n`).

Los clientes pueden tener un fichero de *cookies* donde, para cada nombre de dominio, almacenan asociaciones clave-valor indicadas por el servidor en líneas de cabecera, y que devuelven al servidor en las líneas de cabecera de solicitudes posteriores. El formato de las *cookies* es una lista de campos separados por «`; `»:

`nombre=valor` Asocia a una propiedad un valor específico. Espacio, coma y punto y coma se traducen en secuencias de escape.

`expires=`*`weekday , DD-MM-YY hh:mm:ss`* `GMT` Fecha de caducidad de la *cookie*, por defecto al cerrar el navegador.

`domain=`*`ámbito`* Ámbito de la *cookie*, por defecto el dominio desde donde se genera.

`path=`*`camino`* Recursos a los que se envía la *cookie*, por defecto la página que la genera.

`secure` La *cookie* solo se envía cuando se usa HTTPS.

Son una forma de mantener un estado y se pueden usar para autenticación, preferencias y rastreo a través de la web.

Una **caché web** o **servidor *proxy*** es un servidor distinto al origen al que se envían peticiones web y que puede responder a solicitudes sin involucrar a este a través de una caché, solicitando el objeto desde el servidor origen si no está en caché. Normalmente son instalados por ISPs para reducir el tiempo de respuesta o el tráfico en el enlace de acceso a un servidor, permitiendo que un servidor lento pueda entregar el contenido de modo efectivo sin saturarse.

Un servidor web «genérico» no puede procesar cualquier tipo de consulta, por lo que normalmente se encarga solo del envío de objetos convencional y, para otro tipo de peticiones, como peticiones `POST`, envía el contenido a un programa mediante una interfaz como **CGI**, *Common Gateway Interface*.

En esta, el servidor inicia un proceso con el programa ***gateway*** correspondiente, pasándole los datos de la petición por variables de entorno y parámetros y el cuerpo de la solicitud por la entrada estándar; el programa procesa la solicitud y envía la respuesta por la salida estándar, y el servidor envía al cliente los resultados que emite el programa, normalmente completando las cabeceras de respuesta.

Algunas variables de entorno son:

`DOCUMENT_ROOT` Directorio con las páginas HTML del servidor.

`SERVER_NAME` Nombre del servidor.

`SERVER_PORT` Puerto del servidor.

`HTTP_ACCEPT` Tipos MIME soportados por el cliente.

`HTTP_COOKIE` *Cookies* incluidas en la solicitud.

`REMOTE_ADDR` Dirección IP del cliente.

`CONTENT_LENGTH` Tamaño en bytes del contenido de la información.

`CONTENT_TYPE` Tipo MIME de la petición.

`QUERY_STRING` Parte detrás de la `?` de la ruta solicitada.

En 2015 se define `HTTP/2` en el RFC 7540, que añade:

- Multiplexado de conexiones TCP dividiendo las solicitudes y respuestas en marcos. Todas las conexiones son persistentes.
- Control de flujo interno, direccional basado en crédito o salto a salto.
- Priorizado de transmisiones.
- Varias respuestas a una sola petición mediante envío *push* de recursos todavía no solicitados.
- Compresión de cabeceras, no enviando las cabeceras que no hayan cambiado respecto de la petición anterior, contando también como cabeceras el método y la ruta.

Capítulo 3

Correo electrónico

Un **lector de correo** o **agente de usuario** de correo (**UA**, *User Agent*) es un programa como Thunderbird que permite componer, editar y leer mensajes de correo electrónico. Los mensajes enviados y recibidos se almacenan en un **servidor de correo** (*mail server*), que contiene ***mailboxes*** con los mensajes recibidos de cada usuario y **colas de mensajes** (*message queues*) con los mensajes que se van a enviar.

3.1. SMTP

El envío de correos del agente de usuario a su servidor y entre servidores se hace con el protocolo **SMTP** (*Simple Mail Transfer Protocol*), en el RFC 2821, que funciona sobre TCP, tradicionalmente en el puerto 25. La conexión la inicia el servidor que envía el correo y la transferencia se hace en tres fases, que si no hay problemas transcurren como sigue:

1. ***Handshaking*** o ***greeting***: El receptor envía una línea `220 dominio_receptor`, el emisor responde con una línea `HELO dominio_emisor` y el receptor confirma con una línea `250 saludo_cordial`.

2. Transferencia de mensajes: Para cada mensaje a enviar, el emisor envía una línea `MAIL FROM: emisor` y el receptor responde con `250 mensaje`, el emisor envía una línea `RCPT TO: receptor` y el receptor responde con `250 mensaje`, el emisor envía una línea `DATA`, el receptor envía una línea `354 mensaje`, y el emisor envía el texto del correo seguido de una línea sólo con un punto. Los mensajes están en ASCII de 7 bits y no pueden contener una línea sólo con un punto. El `emisor` y el `receptor` tienen formato `<usuario@dominio>`.

3. Cierre: El emisor envía una línea `QUIT` y el receptor responde con `221 mensaje` y cierra la conexión.

Las líneas se separan con retorno de carro y salto de línea («CRLF»), y no se puede enviar uno de los dos caracteres sin que forme parte de esta combinación. Las líneas tienen un máximo de 1000 caracteres, contando el CRLF. El programa de servidor que recibe y que guarda o reenvía los mensajes SMTP según corresponda es el *Mail Transfer Agent* (**MTA**).

El RFC 822 y sus actualizaciones definen el formato estándar del texto de los correos, con líneas de cabecera, una línea vacía y el cuerpo del mensaje. Las líneas de cabecera tienen formato `Nombre: valor`:

`Bcc` Direcciones para copias «ciegas». Las direcciones indicadas reciben el correo, pero ningún receptor recibe esta cabecera, por lo que solo aparece en carpetas de correo enviado.

`Cc` Direcciones de los destinatarios secundarios.

`Date` Fecha y hora de envío del mensaje.

`From` Quién escribió el mensaje.

`In-Reply-To` Identificador del mensaje al que este responde.

`Message-Id` Identificador único del mensaje.

`Received` MTA por el que ha pasado el mensaje.

`Reply-To` Dirección a la que deben enviarse las respuestas.

`Return-Path` Añadido por el último MTA, indica la trayectoria de regreso calculada según las cabeceras `Received`.

`Sender` Quién envió el mensaje.

`Subject` Asunto del mensaje.

`To` Direcciones de los destinatarios primarios.

`X-Algo` Cabecera no estándar.

3.2. Protocolos de acceso

Hay varios protocolos para acceder al correo recibido en el servidor. **POP** (*Post Office Protocol*), en el RFC 1939, permite control de acceso y descarga. Su evolución, **POP3**, tiene 3 estados:

Autorización (`AUTHORIZATION`) Comandos `user usuario` y `pass password`, en orden, para autenticarse, o `quit` para desconectar, a los que el servidor responde con `+OK` en caso de éxito o `-ERR` si hay fallo. Si ambos tienen éxito, el servidor bloquea el *mailbox* y pasa al siguiente estado.

Transacción (`TRANSACTION`) Comandos `list` para obtener una lista de mensajes del servidor, con una línea `n ID` para cada mensaje, siendo `n` el ordinal del mensaje, que no cambia mientras el *mailbox* siga bloqueado, e `ID` su identificador, ambos números, y finalmente una línea con «`.`»; `retr n` para obtener un mensaje con un cierto ordinal, acabado en una línea con «`.`»; `dele n` para borrar un mensaje, a lo que el servidor no responde, y `quit` para pasar al siguiente estado, con igual respuesta que en `AUTHORIZATION`.

Actualización (`UPDATE`) El servidor libera el bloqueo del *mailbox* y desconecta.

Los clientes POP3 suelen funcionar en modo «descargar y borrar», enviando `list` seguido de `retr` y `dele` repetidamente para descargar los mensajes, lo que impide acceder a estos desde varios dispositivos, o «descargar y mantener», igual pero sin ejecutar `dele`, lo que normalmente hace que se descarguen duplicados.

IMAP (*Internet Mail Access Protocol*), en el RFC 2060 y que funciona en el puerto TCP 143, mantiene los mensajes en el servidor y permite al usuario organizarlos en carpetas y acceder a ellas de forma remota, dando flexibilidad de acceso entre distintos dispositivos, a cambio de que, si no hay conexión a internet, no se puede acceder al correo. Tiene 4 estados:

`NOT AUTHENTICATED` Comandos `LOGIN`, `AUTHENTICATE` y `STARTTLS` para pasar al siguiente estado.

`AUTHENTICATED` Comandos `CREATE`, `DELETE`, `RENAME`, `SUBSCRIBE`, `UNSUBSCRIBE`, `LIST`, `LSUB`, `STATUS` y `APPEND` para manejar el correo y `SELECT` y `EXAMINE` para pasar al siguiente estado.

`SELECTED` Comando `CLOSE` o fallo en `SELECT` o `EXAMINE` para pasar al estado anterior y comando `LOGOUT` para pasar al siguiente.

`LOGOUT` Desconecta.

Muchos servidores de correo proporcionan un servicio *webmail*, que permite enviar y acceder al correo por HTTP a través de página web con un navegador.

3.3. MIME

SMTP no permite caracteres no ASCII, ficheros binarios, líneas largas o líneas sólo con un punto. Para solucionarlo se crea **MIME** (*Multipurpose Internet Mail Extensions*), en los RFCs 2045–9, una extensión del formato de los mensajes de correo que añade las siguientes cabeceras:

`MIME-Version 1.0`

`Content-Type` Tipo de contenido, con formato *`tipo/subtipo`*`[; `*`parámetro=valor`*`; ...]`, de los que vemos algunos en el cuadro 3.1.

`Content-Transfer-Encoding` Codificación del cuerpo del mensaje:

- `7bit` Líneas de hasta 1000 bytes, contando el CRLF, en ASCII. Por defecto.
- `8bit` Igual, pero puede haber caracteres no ASCII. Puede no estar soportado.
- `binary` Igual, pero puede haber líneas arbitrariamente largas. Puede no estar soportado.
- `quoted-printable` Permite añadir caracteres de 8 bits mediante `=XX`, siendo `XX` el caracter en hexadecimal. El caracter `=` también se sustituye. Las líneas tienen tamaño máximo de 76 caracteres sin contar el CRLF, pero un `=` antes de un salto de línea hace que este se ignore, permitiendo líneas más largas.

Tipo	Subtipo	
`text`	`plain`	Texto plano, por defecto.
	`enriched`	Texto enriquecido con *tags* XML como `<bold>`, `<italic>`, `<smaller>` o `<bigger>`.
`multipart`	`mixed`	Parámetro `boundary=`*`boundary`*. Varios ficheros independientes separados por una línea `--`*`boundary`*, con preámbulo antes de la primera línea `--`*`boundary`* y epílogo tras la última ignorados. Cada fichero tiene sus propias cabeceras.
	`parallel`	Igual, pero las partes se abren a la vez. Por ejemplo, un vídeo y un audio a reproducir en paralelo.
	`alternative`	Igual, pero cada parte es una versión alternativa del mismo contenido, en orden creciente de parecido con el original.
	`digest`	Como `mixed` pero el tipo/subtipo por defecto de cada parte es `message/rfc822` en vez de `text/plain`.
`message`	`rfc822`	Mensaje de correo encapsulado.
	`partial`	Usado para permitir fragmentación de objetos grandes de forma transparente al receptor cuando se supera el tamaño máximo permitido por el MTA.
	`external-body`	Puntero a un objeto en otra ubicación.
`image`	`jpeg`	Imagen en formato JPEG/JFIF.
	`gif`	Imagen en formato GIF.
`video`	`mpeg`	Vídeo en formato MPEG.
`audio`	`basic`	Audio PCM mono de 8 bits con ratio de muestreo de 8 kHz.
`application`	`postscript`	Documento en formato Adobe PostScript.
	`octet-stream`	Flujo de bytes de 8 bits no interpretado.

Cuadro 3.1: Algunos tipos MIME.

`base64` Cada 3 caracteres de 8 bits se agrupan en 4 grupos de 6 bits, y los valores resultantes de 0 a 63 se convierten, en orden ascendente, en `A`–`Z`, `a`–`z`, `0`–`9`, `+` y `/`, a representar en líneas de hasta 76 caracteres sin contar el CRLF. Si el tamaño del fichero no es múltiplo de 3, se añaden bits 0 a la derecha para que el resultado sea múltiplo de 6 bits y se añade un `=` al final del mensaje codificado por cada 2 bits añadidos.

`x-`*`algo`* Cabecera no estándar.

Capítulo 4

DNS

Domain Name System es una base de datos distribuida en una jerarquía de servidores de nombres (***name servers***) para traducir entre nombres de dominio y direcciones IP, que permite la creación de alias o nombres de dominio alternativos y la distribución de carga dando al mismo nombre varias direcciones IP, de las que se elige una u otra en orden circular. Las entradas o ***resource records*** (**RR**) tienen forma (nombre, valor, tipo, TTL), donde TTL son los segundos hasta que expire la entrada y algunos tipos son:

`A` El nombre es el de un dominio y el valor es la IPv4.

`CNAME` El nombre es un alias para el valor, ambos nombres de dominio.

`NS` El nombre es el de un dominio y el valor es el servidor DNS responsable (nombre de dominio o IP).

`MX` El valor es un número de prioridad (cuanto mayor, más prioridad) y un servidor de correo asociado con el nombre, un nombre de dominio.

Los nombres de dominio tienen forma jerárquica, *....nivel2.nivel1.*, y un servidor DNS responsable de un dominio se encarga de la dirección de sus **subdominios**, dominios que tienen al dominio principal como sufijo, precedido de un punto, con un nivel mayor. Tipos de servidores DNS:

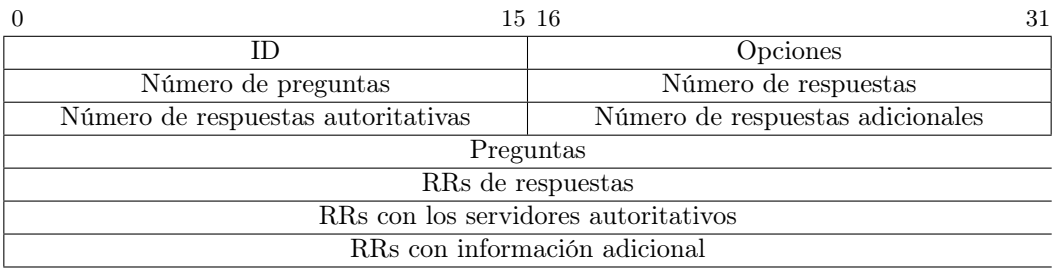
- ***Root name servers***: 13 direcciones IP en el mundo que mantienen el nivel de jerarquía más alto (nivel 1).
- ***Top-level domain*** (**TLD**): Responsables de los dominios de primer nivel: `com`, `org`, `net`, `edu`, `aero`, `jobs`, `museums`, etc., incluyendo los de los países, como `uk`, `fr`, `ca`, `jp`, etc. Network Solutions mantiene servidores para el TLD `com` y Educause para el TLD `edu`.
- ***Authoritative***: Servidor DNS de una organización, con información de mapeo entre nombre e IP para los hosts de la organización. Los mantiene la organización o un proveedor de servicio.

- **Servidores locales**: No son necesariamente responsables de ningún dominio, sino que son servidores más cercanos al usuario a los que este consulta y que actúan como *proxy*, pudiendo responder desde una caché o reenviar la consulta a otro servidor local o a la jerarquía. Llamamos ***default name server*** al servidor local al que pregunta un cierto usuario.

Una consulta hecha por un servidor es **iterativa** si, cuando el servidor contactado no conoce un mapeo, responde con el nombre del servidor al que contactar para obtenerlo, y es **recursiva** si es este el que contacta. La recursiva mejora la caché del servidor contactado, pero sobrecarga los niveles altos de la jerarquía si la consulta se reenvía a esta.

Cuando un servidor de nombre aprende un mapeo, lo guarda en una caché (***caching***) hasta que expira el TTL. Los TLD se cachean en servidores locales, por lo que los servidores raíz no se consultan frecuentemente. El RFC 2136 propone mecanismos de actualización y modificación.

Los mensajes se envían sobre UDP con el siguiente formato:



El ID es el mismo en la petición que en la respuesta. Las opciones indican si se trata de una petición (*query*) o respuesta (*reply*), si se desea recursividad, si esta está disponible, etc.

Para registrar un nombre en un dominio, se guardan en un ***DNS registrar*** un registro NS con el nombre de dominio *authoritative* y un registro A para dicho servidor, y en el servidor *authoritative* se incluyen los registros A para el dominio y subdominios, MX, etc.

Para un dominio un servidor DNS puede ser **primario** o **maestro** si almacena la información sobre la zona o **secundario** o **esclavo** si obtiene información de esta copiándola del primario.

DNS podría ser vulnerable a ataques de solicitud de servicio distribuidos (DDoS). Esto no se ha conseguido para los servidores raíz, pues estos implementan filtrado de tráfico y, además, los servidores locales rara vez contactan con estos porque tienen las IPs de los TLDs en caché; sin embargo, esto es potencialmente más peligroso si el objetivo son los TLDs. DNS se puede explotar para DDoS enviando solicitudes con una dirección IP fuente que se pretende atacar, aunque se requiere algún mecanismo de amplificación de las respuestas. También es vulnerable a ataques *man-in-the-middle* (MITM); en concreto, el ***DNS poisoning*** consiste en enviar respuestas DNS con información falsa a los servidores locales para que la cacheen.

Capítulo 5

DHCP

Un administrador de red puede asignar direcciones IP:

- **Fijas**, de forma **estática**, útil para equipos que no deben cambiar de IP como routers y servidores. En Linux, la IP de una interfaz se indica en `/etc/network/interfaces`.
- **Temporales**, de forma **dinámica**, útil en equipos de usuarios que no requieren direcciones IP fijas, el número de equipos es alto y la gestión de direcciones es compleja. Esto permite reutilizar direcciones, que solo se ocupan cuando el equipo está conectado, y es especialmente útil para usuarios móviles que entran en la red de forma esporádica.

DHCP (*Dynamic Host Configuration Protocol*) es un protocolo sobre UDP para obtener una IP dinámica de un servidor DHCP, junto con más información como la dirección del router de salida, el nombre y la dirección del servidor DNS o la máscara de red. Un ***DHCP client*** es un equipo que solicita una IP dinámica por DHCP, un ***DHCP server*** es uno que la asigna y un ***DHCP relay*** redirige solicitudes DHCP para permitir que el cliente y el servidor estén en subredes distintas.

Tipos de mensaje:

- ***Discover***: Enviado por un cliente a *broadcast* (IP 255.255.255.255, MAC ff:ff:ff:ff:ff:ff) al conectarse a la red para saber si hay algún servidor DHCP.
- ***Offer***: Respuesta *broadcast* de un servidor a un *DHCP Discover*, que incluye una dirección libre y la duración que tendría el «**alquiler**» (*lease*) de la dirección por parte del cliente.
- ***Request***: Solicitud del cliente de alquilar una IP, enviada tras recibir un *DHCP Offer* o al expirar el tiempo de alquiler para renovarlo.
- ***ACK***: Confirmación de que el servidor ha recibido la *DHCP Request* y el cliente puede usar la IP.
- ***Nak***: Indica que no se pudo llevar a cabo la petición del cliente, por ejemplo, porque la dirección ya esté ocupada. En una renovación, hace que el cliente reinicie el proceso general.
- ***Inform***: Usado por el cliente tras un *ACK* para solicitar más información.

Capítulo 6

FTP

El protocolo **FTP** (*File Transfer Protocol*, RFC 959) permite enviar y recibir ficheros de un *host* remoto. El cliente contacta con el puerto TCP 21 del servidor para iniciar una **conexión de control**, por la que envía comandos ASCII y recibe códigos de estado, también en ASCII y con una frase, y metadatos. Esta conexión es *out of band*, separada del canal de datos.

La conexión de datos se crea para transferir un fichero y es cerrada por el cliente o el servidor al terminar la transferencia. Modos:

1. **Modo activo**: El cliente envía un comando `PORT` con un número de puerto y, para el envío de datos, el servidor crea una conexión TCP con el cliente con puerto origen 20 y el puerto destino indicado.
2. **Modo pasivo**: El cliente envía un comando `PASV` y el servidor le devuelve un número de puerto libre, al que el cliente se conecta para el envío de datos.

Otros comandos:

`USER` *username* Indica el nombre de usuario.

`PASS` *password* Completa el inicio de sesión.

`LIST` Lista los ficheros en el directorio actual.

`RETR` *filename* Descarga un fichero.

`STOR` *filename* Sube un fichero.

Algunos códigos de estado:

```
125 data connection already open; transfer starting
```

```
331 Username OK, password required
```

```
425 Can't open data connection
```

```
452 Error writing file
```

Capítulo 7

Seguridad en redes

La seguridad en red se refiere a:

- **Confidencialidad**: Evitar que terceras partes entiendan el mensaje. Si K_A es la clave de cifrado o encriptado de Alice y K_B es la clave de descifrado correspondiente de Bob, Alice encripta un ***plaintext*** o mensaje en claro m con K_A y envía el ***ciphertext*** o texto cifrado $K_A(m)$ a Bob, que obtiene $m = K_B(K_A(m))$.
- **Autenticación**: Confirmar que la otra parte de la comunicación es quien dice ser.
- **Integridad**: Asegurar que los mensajes no han sido alterados.
- **Acceso** y **disponibilidad**: Los servicios deben estar accesibles y disponibles a los usuarios.

Alice y Bob pueden ser personas, un navegador y un servidor web, servidores DNS, routers intercambiando información de enrutamiento, etc. Trudy, un intruso, puede:

- Espiar mensajes (***eavesdrop***).
- Insertar o filtrar mensajes.
- Modificar mensajes, por ejemplo para impersonar (***spoof***) a Alice o a Bob cambiando la IP fuente en los paquetes.
- Robar una conexión abierta sustituyendo al emisor o receptor (***hijacking***).
- Prevenir que un servicio sea usado por otros (***denial of service***), por ejemplo sobrecargando los recursos.

Clasificamos los ataques a un sistema de cifrado o encriptado en:

- Ataques de solo *ciphertext* en que solo se analiza este, como la **fuerza bruta**, en que se prueban todas las claves hasta encontrar la correcta, o ataques por análisis estadístico.
- Con *plaintext* conocido, en que Trudy conoce el *plaintext* correspondiente a algún *ciphertext*.
- Con *plaintext* escogido, en que Trudy puede obtener el *ciphertext* de un *plaintext*.

7.1. Criptografía de clave simétrica

Las claves de cifrado y descifrado son la misma, aunque el procedimiento para cifrar y descifrar puede ser distinto.

- **Cifrado de sustitución**: Sustituir un objeto por otro. La clave es una biyección entre objetos. En el **cifrado monoalfabético** se sustituye una letra por otra.
- Se puede usar un patrón cíclico para alternar entre distintos cifrados de sustitución, y la clave es una lista de cifrados de sustitución y el patrón cíclico.
- **Cifrado de flujo**: La clave tiene el mismo número de bits que el mensaje a cifrar y se aplica XOR entre el mensaje y la clave.
- Se puede usar una clave corta como semilla de un generador de números pseudo-aleatorio (**PRNG**) para generar la clave en un cifrado de flujo.
- **Cifrado de bloque**: El mensaje se divide en bloques de longitud fija cifrados por separado, usando *padding* para ajustar los mensajes al tamaño del bloque. Es importante la relación entre los bits.
- ***Cipher Book Chaining*** (**CBC**): Para ocular patrones, cada bloque se cifra con una clave prefijada combinada, por ejemplo con XOR, con el *ciphertext* del bloque anterior, o con un **vector de inicialización** para el primer bloque.
- **DES** (*Data Encryption Standard*): Estándar estadounidense NIST 1993 que usa CDC con bloques de 64 bits y una clave simétrica de 56 bits. A cada bloque se le hace una permutación inicial, 16 *rounds* idénticos de una función que usa distintos 48 bits de la clave y una permutación final, obteniendo un *ciphertext* de 64 bits. En el ***DES Challenge*** se vio que la clave se puede obtener por fuerza bruta en menos de un día, pero no se conoce un buen ataque analítico.
- **3DES**: Cifrar 3 veces seguidas con DES con 3 claves distintas.
- **AES** (*Advanced Encryption Standard*): Estándar NIST que sustituye a DES desde noviembre de 2001. Usa bloques de 128 bits y claves de 128, 192 y 256 bits. Si descifrar DES por fuerza bruta tardara un segundo, con AES tardaría 149 billones de años.

7.2. Criptografía de clave pública

En la **criptografía de clave pública**, cada entidad tiene una **clave pública** K^+ conocida por todos y una **clave privada** K^- solo conocida por la entidad, de forma que lo que se encripta con la clave pública se puede desencriptar con la privada y es inviable calcular la privada a partir de la pública.

En **RSA** (*Rivest-Shamir-Adelson Algorithm*), para generar las claves se escogen dos primos grandes p y q; se calculan $n \coloneqq pq$ y $z \coloneqq \phi(n) = (p-1)(q-1)$; se escogen $e < n$ con e y z coprimos y d tal que $ed \equiv 1$ mód z, y se toma (n, e) como clave pública y (n, d) como clave privada. Los mensajes se interpretan como enteros positivos. Cifrar un mensaje m es calcular m^e mód n, y descifrar un texto cifrado c es calcular c^d mód n.

Con esto, para todo mensaje m, $K^-(K^+(m)) = m$. En efecto, sea k tal que $ed = kz + 1$, por el teorema de Euler, $m^z = m^{\phi(n)} \equiv 1$ mód n, luego trabajando módulo n, $K^-(K^+(m)) \equiv (m^e)^d = m^{ed} = m^{kz+1} = (m^z)^k m \equiv 1^k m = m$. Además, para determinar d a partir de (n, e), en esencia hay que calcular $\phi(n)$ factorizando n en p y q, pero factorizar un entero es NP-completo.

7.3. Intercambio de claves

Encriptar en RSA es computacionalmente costoso, siendo DES al menos 100 veces más rápido en software, por lo que se suele usar RSA para enviar una clave simétrica encriptada que es la que se usa para cifrar los datos.

Otra opción es el **algoritmo de Diffie-Hellman**:

1. Alice y Bob acuerdan, sin necesidad de cifrado, un número primo q y un entero $\alpha < q$ coprimo con q.
2. Generan claves privadas respectivas $X_A, X_B < q$.
3. Generan claves públicas respectivas $Y_A := \alpha^{X_A}$ mód q e $Y_B := \alpha^{X_B}$ mód q y las intercambian.
4. Alice calcula $Y_B^{X_A}$ mód q y Bob calcula $Y_A^{X_B}$ mód q. El resultado coincide y se usa como clave simétrica.

En efecto, $Y_B^{X_A} = (\alpha^{X_B})^{X_A} = (\alpha^{X_A})^{X_B} = Y_A^{X_B}$, y la seguridad se debe a que es muy difícil calcular **logaritmos discretos** (la inversa del exponente en aritmética modular). Sin embargo, este algoritmo no ofrece autenticación de las partes, por lo que es vulnerable a ataques ***man-in-the-middle*** (**MITM**), en los que un atacante se hace pasar por Bob para Alice y por Alice para Bob haciendo de intermediario en la comunicación y viendo los mensajes en claro.

7.4. Autenticación

La autenticación permite a las partes de la comunicación saber que los mensajes que reciben son realmente de quien dicen ser y que no han sido alterados.

Un mecanismo de autenticación es **verificable**, **no falsificable** o **no repudiable** si el receptor puede probar ante otros que un mensaje realmente ha sido enviado por el emisor, en cuyo caso el mecanismo es de **firma digital**.

Una forma de firma digital es adjuntar al mensaje una copia de este cifrada con la clave privada del emisor, que el receptor comprobará aplicando la clave pública del emisor a esta copia y comparándola con el mensaje.

Esto es computacionalmente costoso, por lo que en la práctica lo que se cifra es una **huella** (*fingerprint*) del mensaje, obtenida de aplicarle a este una **función *hash* criptográfica** o ***message digest***, una función fácil de calcular que genera una huella de longitud fija a partir del mensaje y para la que es computacionalmente inviable encontrar dos mensajes con un mismo *hash* y por tanto un mensaje que tenga un *hash* determinado.

El *Internet checksum* no sirve, pues a partir de un mensaje es fácil encontrar otro con el mismo *checksum*. Algunos *hash* criptográficos son **MD5** (RFC 1321), que calcula una huella

de 128 bits en 4 pasos, y **SHA-1** (NIST FIPS PUB 180-1), que calcula una de 160 bits, pero ambos han sido atacados al encontrar colisiones, por lo que ahora se usa **SHA-2**, que produce huellas de 224, 256, 384 o 512 bits.

Otra forma de autenticación más eficiente, especialmente para sesiones interactivas, es adjuntar a los mensajes un **MAC** (*Message Authentication Code*), la huella de la concatenación del mensaje y una clave simétrica preestablecida (*authentication key*), aunque esto no sirve de firma digital porque el receptor también puede calcular el MAC.

Las **autoridades de certificación** (**CA**) sirven para evitar recibir la clave pública de una persona indicando que es de otra. Para ello, una entidad registra su clave pública en la CA aportando una prueba de identidad, y la CA crea un certificado firmando la clave pública a registrar con su clave privada.

Capítulo 8

X.509

X.509 es un estándar ISO/ITU de certificados electrónicos que asocian una clave pública con la identidad de su propietario. Se creó en 1988 basándose en X.500 y actualmente está en su versión 3, descrita en el RFC 5280. Se usa en S/MIME, TLS, SSH, etc.

El formato es:

Versión
Número de serie
Algoritmo y parámetros (p. ej., SHA-256 con RSA)
Nombre de la CA
Intervalo temporal de validez
Identidad del propietario
Algoritmos, parámetros y clave pública en sí
ID único de CA, opcional
ID único de propietario, opcional
Extensiones, opcional
Firma de la CA: algoritmos, parámetros y huella *hash* encriptada

Registrar un certificado consiste en enviarlo a la CA sin firmar para que lo verifiquen y obtener una firma.

Los certificados **expiran** al terminar su periodo de validez, y la **renovación** consiste en obtener un nuevo certificado antes de que esto ocurra. La **revocación** de un certificado no expirado consiste en marcarlo como no válido añadiéndolo a la **CRL**, una lista pública mantenida por la CA de certificados revocados no expirados, lo que se hace porque se comprometa la clave pública o el certificado de la CA o porque el usuario ya no pertenezca a la CA.

Formato de la CRL:

Algoritmo y parámetros
Nombre de la CA
Fecha de actualización
Fecha de siguiente actualización
Número de serie de certificado y fecha de validez
⋮
Firma de la CA

Capítulo 9

SSL y TLS

SSL es un protocolo sobre TCP para añadir cifrado, integridad y autenticación a una conexión creado por Netscape. Está formado por el ***SSL Record Protocol*** (**RP**), para dividir la conexión en segmentos que se encriptan y por separado, y protocolos de control que funcionan por encima de este.

El formato de los segmentos es:

Tipo de contenido
Número de versión mayor
Número de versión menor
Longitud del texto comprimido
Texto plano, opcionalmente comprimido
MAC de lo anterior
Encriptado
Al menos 1 byte de contenido aleatorio

La MAC se calcula como

```
hash(shared_key || pad2 ||
     hash(shared_key || pad1 || seq_num || compression_type ||
          compressed_length || compressed_text))
```

donde `||` es la concatenación, `shared_key` es la clave compartida, `seq_num` es un número de secuencia, `compression_type` es el mecanismo de compresión, `compressed_length` es la longitud del texto comprimido, `compressed_text` es el texto plano comprimido y `pad1` y `pad2` son respectivamente `0x36` y `0x5C` repetidos un número de veces, 48 cuando el `hash` es MD5 y 40 cuando es SHA-1.

Los algoritmos de cifrado de bloque son DES (tamaño de clave 40 o 56), AES (128 o 256), RC2 (40), IDEA (128), 3DES (168) y Fortezza (80), y para cifrado de flujo se usa RC4.

9.1. CCSP

El ***Change Cipher Spec Protocol*** está compuesto de un único mensaje con un bit 1, y lo inicia cualquiera de los participantes para actualizar los parámetros criptográficos.

9.2. HP

El ***Handshake Protocol*** se usa al principio de una conexión para el intercambio de claves. Se intercambian los parámetros para rellenar una **tabla de sesión** y una **tabla de conexión** en 4 fases:

1. El cliente envía un `client_hello` al servidor con sus parámetros, y el servidor responde con un `server_hello` con sus parámetros.

   Estos son un ID de sesión (`session_identifier`), algoritmos de cifrado (`cipher_spec`), si la sesión se puede restaurar (`is_resumable`) y un *nonce* aleatorio (`client_random` en el `client_hello` y `server_random` en el `server_hello`, que se concatenan en un `client_server_random`). Los 3 primeros parámetros van a la tabla de sesión y el último a la de conexión.

   La `cipher_spec` tiene formato ***PROT_AUTH_SYM_SIZE_BLK_HASH***, donde ***PROT*** es el protocolo (`SSL`, `TLS`, ...), ***AUTH*** es el algoritmo de clave pública para autenticación (`RSA`, `DSS`, ...), ***SYM*** es el algoritmo de clave simétrica para cifrado (`AES`, `DES`, ...), ***SIZE*** es el tamaño de bloque (`64`, `128`, `256`, ...), ***BLK*** es el modo de cifrado de bloque (`CBC`, `OCF`, ...) y ***HASH*** es la función *hash* (`MD5`, `SHA_1`, ...). Si se establece el secreto compartido por *Elliptic Curve Diffie-Hellman*, se añade `ECDHE` antes de ***AUTH*** como método de intercambio de claves.

2. El servidor envía un `certificate` al cliente con su certificado (`peer_certificate`), solicita opcionalmente un certificado del cliente con un `certificate_request`, envía un mensaje `server_key_exchange` si el intercambio se hace por un método Diffie-Hellman y envía un `server_hello_done`. El cliente verifica la confianza en el certificado, crea una clave *pre-master-secret* y de esta deriva una *master-secret*, usada como valor pseudo aleatorio.

   ```
   master_secret :=
       MD5(pre_master ||
           SHA('A' || pre_master || client_server_random)) ||
       MD5(pre_master ||
           SHA('BB' || pre_master || client_server_random)) ||
       MD5(pre_master ||
           SHA('CCC' || pre_master || client_server_random))
   ```

3. El cliente envía un `client_key_exchange` con la *pre-master-secret*, de la que el servidor deriva la *master-secret*. El `peer_certificate` y la `master_secret` van a la tabla de sesión.

   Si se usa autenticación de cliente, este envía también un mensaje `certificate` y un `client_verify` para demostrar que posee la clave privada asociada a su certificado, que contiene

   ```
   hash(master_secret || pad2 ||
        hash(handshake_messages || master_secret || pad1))
   ```

   Si se usa Diffie-Hellman, en vez de usar una *pre-master-secret*, la *master-secret* se establece con los mensajes `server_key_exchange` y `client_key_exchange`.

4. De la *master-secret* se derivan las subclaves `server_write_MAC_secret`, `client_write_MAC_secret`, `server_write_key` y `client_write_key`, y vectores de inicialización del cifrado de bloques en ambas direcciones (`initialization_vector`), que guardan en la **tabla de conexión** junto a `sequence_numbers`. El cliente envía un mensaje CCSP para pasar a conexión cifrada y un `finish`, que verifica que el proceso concluyó con éxito, va ya cifrado y contiene

```
MD5(master_secret || pad2 ||
    MD5(handshake_messages || sender || master_secret || pad1)) ||
SHA(master_secret || pad2 ||
    SHA(handshake_messages || sender || master_secret || pad1))
```

   El servidor hace lo mismo.

9.3. AP

El ***Alert Protocol*** permite notificar avisos y errores con encriptación. Los mensajes tienen un byte de nivel, que vale 1 para notificar avisos (*warning*) o 2 para errores irrecuperables que requieren abortar la conexión (*fatal*), y un byte de código de alerta. Algunos de los códigos son:

`Close_notify` El emisor no enviará más mensajes.

`Unexpected_message` Se ha recibido un mensaje inapropiado.

`Bad_record_mac` Se ha recibido un segmento con MAC incorrecta.

`Decompression_failure` El texto no cumple el formato de compresión establecido.

`Handshake_failure` El emisor es incapaz de negociar parámetros de seguridad aceptables.

`No_certificate` No hay un certificado disponible.

`Bad_certificate` El certificado tiene una firma no verificada o ha sido dañado.

`Unsupported_certificate` Tipo de certificado no soportado.

`Certificate_revoked` El certificado está revocado.

`Certificate_expired` El certificado ha expirado.

`Certificate_unknown` Otra propiedad del certificado lo hace no aceptable.

`Illegal_parameter` Un campo está fuera de rango o es inconsistente con otros campos.

9.4. TLS

Transport Layer Security es el sucesor de SSL. La versión 1.0 es el RFC 2246 y añade algunos cambios a SSL 3.0. Su número de versión en el RP es 3.1; el algoritmo de firma se basa en HMAC estándar (RFC 2104); se añaden una función pseudoaleatoria, más secretos compartidos para mayor seguridad, nuevos códigos de alerta, *padding* variable múltiplo de la longitud del bloque de cifrado y más tipos de certificados de cliente; se eliminan los cifrados nulo y Fortezza y cambia el cálculo del *hash* en `certificate_verify` y `finished`.

La versión 1.1, de 2006, añade protección contra ataques a CBC. La 1.2, de 2008, reemplaza MD5 y SHA-1 por SHA-2 en los distintos mensajes, mejora la especificación de algoritmos soportados y añade algoritmos de cifrado autenticado y AES.

La versión 1.3, de 2018, está propuesta como *Internet Standard*. Mejora el rendimiento reduciendo el establecimiento de conexión en el caso general a 1 RTT, separa los algoritmos de intercambio de claves y autenticación, añade un modo 0-RTT, cifra todos los mensajes después del `server_hello`, rediseña las funciones de derivación de clave (**HDKF**), y añade algoritmos basados en curvas elípticas y un resumen de sesión basado en PSK.

Capítulo 10

IPSec

IPSec es una extensión de IPv4 e IPv6 para proteger comunicaciones que funciona de forma transparente en redes existentes. Su arquitectura se especifica en el RFC 4301, que actualiza el 2401. Modos de operación:

1. **Transporte**: Se establece una sesión entre los dos extremos que encripta o autentica los datos del nivel de transporte.

2. **Túnel**: Se establece una sesión entre puertas de enlace de redes normalmente corporativas, que encripta o autentica los datos del nivel de red y añade una cabecera IP que tiene como direcciones fuente y destino las de las puertas de enlace.

Cada extremo tiene una **SPD** (*Security Policy Database*), cuyas entradas contienen un nombre, direcciones IP remotas (una o varias direcciones, un rango o una subred), direcciones locales (mismo formato), el protocolo a nivel de transporte (*Next Layer Protocol*), puertos locales y remotos (uno o varios puertos o un comodín *) y la acción a realizar.

También tiene una **SAD** (*Security Association Database*) con **SAs** (*Security Associations*), relaciones unidireccionales entre emisor y receptor que dotan de seguridad a un flujo de tráfico. Un SA está formado por un **SPI** (*Security Parameters Index*), número de 32 bits seleccionado por el receptor que identifica la SA, una dirección IP de destino, un identificador del protocolo de seguridad (AH o ESP), algoritmos de cifrado y autenticación, vectores de inicialización, claves de autenticación y cifrado, tiempo de vida de las claves, tiempo de vida (intervalo de tiempo o número de bytes) tras el cual hay que renovar la SA, modo túnel o transporte, *path MTU* descubierto, contador de número de secuencia, número de secuencia máximo, ventana, etc.

Al enviar un paquete, se busca en la SPD. Si no se encuentra una entrada correspondiente o la acción es descartar, se descarta. Si la acción es dejar pasar, se reenvía el paquete por IP. Si es proteger, se busca la entrada en la SAD para asegurar el paquete antes de enviarlo o se establece la SA.

Al recibir un paquete, si es IPSec, se busca la entrada en la SAD para desencriptar o autenticar el contenido antes de enviarlo a la capa superior, descartándolo si no se encuentra la entrada. En otro caso se busca en la SPD y, si la acción es dejar pasar, se envía a la capa superior, y de lo contrario se descarta.

10.1. IKE

El ***Internet Key Exchange*** (**IKE**), con versión 1 en el RFC 2409 y versión 2 en el 7296, es el protocolo para establecer SAs entre un **iniciador** y un **contestador**:

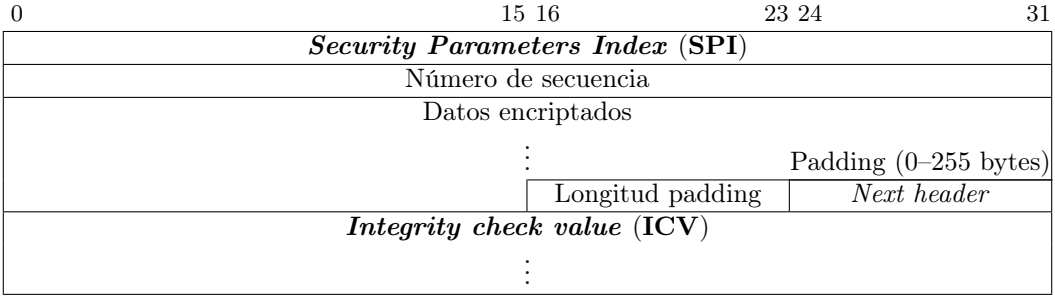
1. `IKE_SA_INIT`: El iniciador envía al contestador un mensaje con la suite criptográfica (incluyendo si usar AH o ESP), una clave Diffie-Hellman y un número aleatorio. Este le responde con los mismos datos y una solicitud de certificado.

2. `IKE_AUTH`: El iniciador envía al contestador un mensaje con su identidad, información de autenticación, la suite criptográfica, las políticas de iniciador y contestador (por ejemplo, modo túnel o transporte) y, según se requiera, un certificado, una solicitud de certificado y la identidad de contestador que espera. El contestador le responde con los mismos campos y, según se requiera, un certificado.

10.2. ESP

Encapsulating Security Payload (**ESP**), en el RFC 4303, que actualiza el 2406 y el 1827, proporciona confidencialidad; opcionalmente integridad sin conexión (por paquete), con autenticación implícita de los datos de origen, y opcionalmente anti-reenvío mediante números de secuencia. Se identifica como protocolo IP 50 en el campo *Protocol* de IPv4 y el *Next Header* de IPv6.

Formato:



El ICV, si aparece, proporciona la integridad de los datos encriptados.

10.3. AH

Authentication Header (**AH**), en el RFC 4302, que actualiza el 2402 y el 1826, proporciona integridad mediante una cabecera inmediatamente después de la cabecera IP con SPI, número de secuencia e ICV, que autentica no solo la siguiente capa sino también campos seleccionados de la cabecera IP externa.

Capítulo 11

SSH

Es un protocolo sencillo cliente-servidor para asegurar comunicaciones que funciona sobre el puerto TCP 22. SSH 1 se centraba en hacer seguro el acceso remoto por Telnet, y SSH 2 resuelve problemas de seguridad en SSH 1.

11.1. *Transport Layer Protocol*

Proporciona autenticación, confidencialidad, integridad y compresión opcional a los protocolos superiores. Para la autenticación, el cliente tiene una base de datos local con las claves públicas de los servidores en que confía, que suele estar en `~/.ssh/known_hosts`, y certificados CA para verificar los certificados de servidor.

Establecimiento de conexión segura o **túnel**:

1. ***Identification string exchange***: El cliente envía su versión del protocolo (1.0 y 2.0 son incompatibles) y del software al servidor, y este responde con lo mismo.
2. ***Algorithm negotiation***: El cliente envía un `SSH_MSG_KEXINIT` al servidor y este le responde con otro. Se negocian los algoritmos de intercambio de claves, cifrado simétrico, MAC y compresión.
3. ***Key exchange***: Diffie-Hellman autenticado con firma digital.
4. ***End of key exchange***: El cliente envía un `SSH_MSG_NEWKEYS` al servidor y este le responde con otro.
5. ***Service Request***: El cliente solicita el comienzo de la autenticación de usuario o el establecimiento de una conexión.

11.2. *User Authentication Protocol*

Para la autenticación de cliente, este envía un `SSH_MSG_USERAUTH_REQUEST` con las credenciales y un método de autenticación, y el servidor responde con `SSH_MSG_USERAUTH_FAILURE`, en cuyo caso el cliente puede volver a intentarlo, o `SSH_MSG_USERAUTH_SUCCESS`.

Normalmente el cliente va probando métodos de autenticación, empezando por `none` (sin autenticación, solo se indica el nombre de usuario).

11.3. *Connection Protocol*

Funciona sobre el *Transport Layer Protocol* y multiplexa el túnel en **canales**. El cliente y el servidor pueden iniciar el establecimiento de un canal, al que se asigna un identificador y se realiza control de flujo.

Para abrir un canal, el iniciador envía un mensaje `SSH_MSG_CHANNEL_OPEN` a la otra parte, que responde con un `SSH_MSG_CHANNEL_OPEN_CONFIRMATION`. La transferencia de datos ocurre en mensajes `SSH_MSG_CHANNEL_DATA`, y el canal se cierra con un `SSH_MSG_CHANNEL_CLOSE`.

Algunos tipos de canales son *interactive session* (sesión interactiva de *shell*), *X11 redirect* (redirección de una conexión X11 para usar aplicaciones gráficas de forma remota), *forwarded-tcpip* y *direct-tcpip*.