

SecDevOps y Administración de Redes para Cloud

---

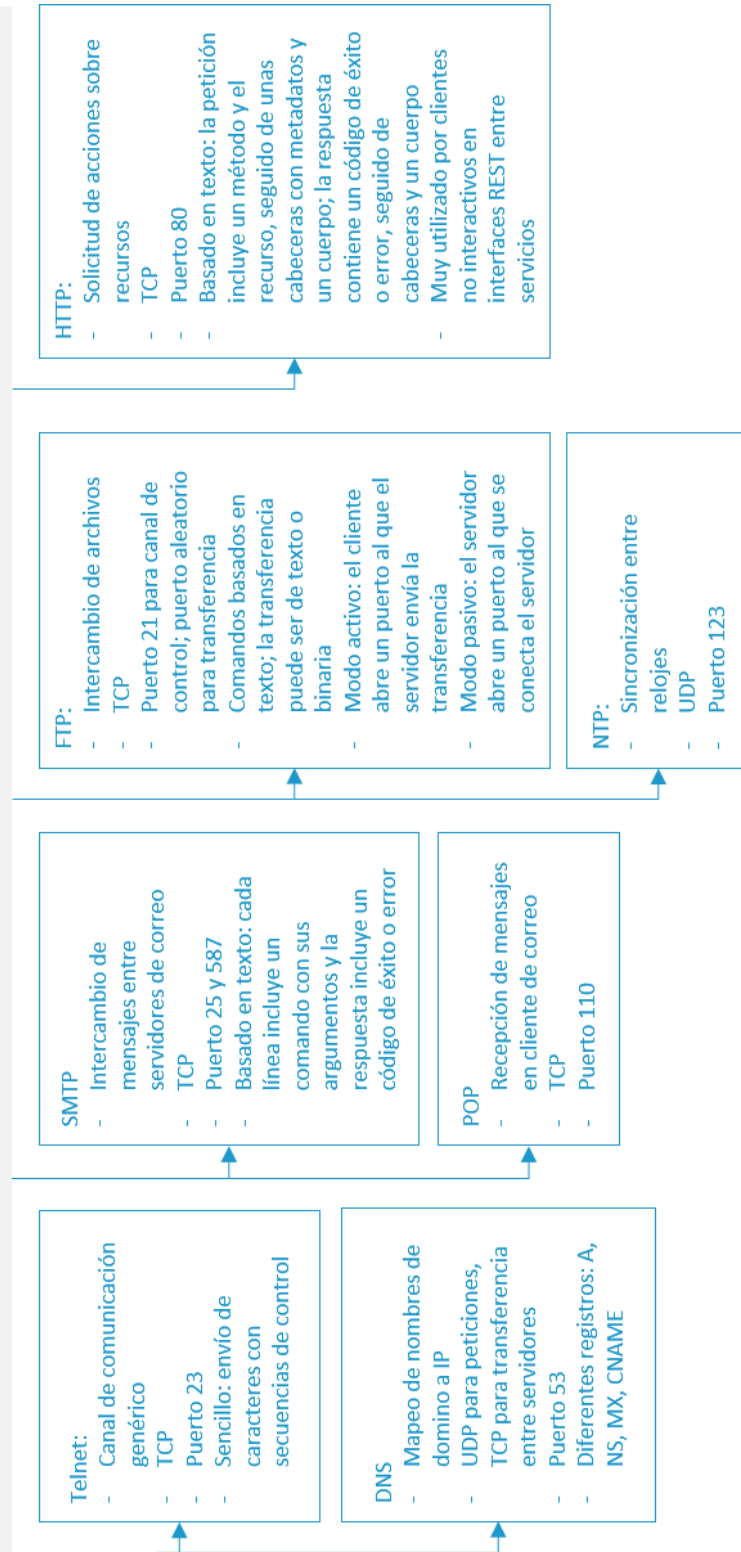
# Fundamentos de redes 3

# Índice

Esquema	3
Ideas clave	4
4.1. Introducción y objetivos	4
4.2. Introducción a la capa de aplicación	5
4.3. Telnet	7
4.4. DNS	11
4.5. SMTP	14
4.6. POP	16
4.7. FTP	17
4.8. HTTP	19
4.9. NTP	28
4.10. SSL	29
4.11. Referencias bibliográficas	43
A fondo	45
Test	47

## FUNDAMENTOS DE REDES 3

## PROTOCOLOS DE CAPA DE APLICACIÓN



## 4.1. Introducción y objetivos

Este tema trata algunos de los protocolos más relevantes de la **capa de aplicación**. Mientras que el número de protocolos de las otras redes es relativamente bajo, la cantidad de protocolos de aplicación está al nivel del número de **aplicaciones disponibles**. Por tanto, conocer en profundidad todos ellos es una tarea para los administradores más exigentes. En este tema solo se analizarán algunos de ellos para ofrecer una **idea general** sobre cómo funcionan y qué tipo de tareas intentan resolver.

Los **objetivos** que se pretenden conseguir en este tema son:

- ▶ Conocer algunos de los protocolos de aplicación más comunes.
- ▶ Entender la diferencia entre protocolos orientados a conexión y no orientados a conexión a nivel de aplicación.
- ▶ Entender cómo funcionan los protocolos basados en texto.

En el siguiente vídeo, titulado «**Capa de aplicación**», se muestra de manera interactiva el funcionamiento de las diferentes capas, mediante capturas de red con Wireshark.



Accede al vídeo

## 4.2. Introducción a la capa de aplicación

La capa de aplicación es la **capa superior del modelo TCP/IP**. Un usuario puede, o no, interactuar directamente con las aplicaciones. La capa de aplicación es donde se inicia y refleja la comunicación real. Debido a que esta capa está en la parte superior de la pila de capas, no ofrece servicios a ninguna otra capa. La **capa de aplicación** se ayuda de la capa de transporte, y de todas las capas inferiores, para comunicar o transferir sus datos al servidor remoto.

Cuando un protocolo de capa de aplicación desea comunicarse con su aplicación de protocolo homóloga en el *host* remoto, **entrega los datos** o la información a la **capa de transporte**. Esta gestiona el resto con la ayuda de todas las capas que están debajo.

Existe una ambigüedad en la comprensión de la capa de aplicación y su protocolo. No todas las **aplicaciones de usuario** se pueden considerar elementos de la capa de aplicación, sino solo aquellas aplicaciones que interactúan con el **sistema de comunicación**.

Por ejemplo, un **editor de texto** es una aplicación de usuario, pero no puede considerarse como un programa de la capa de aplicación. Por otro lado, un **navegador web** en realidad está usando el protocolo *Hyper Text Transfer Protocol* (HTTP) para interactuar con la red. El protocolo de la capa de aplicación es HTTP.

Otro ejemplo es *File Transfer Protocol* (FTP), que permite, al usuario, **transferir archivos basados en texto** o archivos binarios a través de la red. Un usuario puede utilizar este protocolo en cualquier programa de escritorio como FileZilla, CuteFTP o en línea de comandos. El protocolo de capa de aplicación es **FTP**, no las aplicaciones de usuario.

Por lo tanto, independientemente del software que se utilice, el protocolo usado por el software es el agente que se considera parte de la capa de aplicación.

## Modelo cliente-servidor

En el modelo cliente-servidor, cualquier proceso puede actuar como servidor o cliente. No es el tipo o tamaño de la máquina, ni sus **especificaciones de hardware**, lo que lo convierten en servidor, sino su capacidad de **atender las peticiones** de los procesos cliente.

Un sistema puede actuar simultáneamente como servidor y cliente. Es decir, uno de sus procesos está actuando como **servidor**, y otro está actuando como **cliente**. Tanto cliente como servidor pueden residir en la **misma máquina**.

Dos procesos en el modelo cliente-servidor pueden **interactuar** de varias maneras:

- ▶ *Sockets.*
- ▶ Llamada a procedimiento remoto (*Remote Procedure Calls* o RPC).

### Sockets

En este **paradigma**, el proceso que actúa como servidor abre un *socket* utilizando un puerto conocido por el cliente y espera hasta que llegue alguna **solicitud** de este.

Un *socket* es un objeto en la **pila de protocolos** de red del **sistema operativo** al que se entregan los paquetes recibidos por el *host* y dirigidos al *socket*, identificado por el número de puerto. El segundo proceso, actuando como cliente, también abre un *socket*, pero, en lugar de esperar una solicitud entrante, **envía paquetes** a través de dicho socket. Cuando llega la petición al servidor, esta es dirigida al *socket* y atendida por el proceso.

## RPC

Este es un **mecanismo** en el que un proceso interactúa con otro mediante el **procedimiento de llamada**. Un proceso cliente llama al procedimiento que se encuentra en el *host* remoto, que, a su vez, actúa como **servidor**. La comunicación ocurre de la siguiente manera:

El proceso de cliente invoca el código que simula la funcionalidad remota como si fuera una llamada local. La llamada incluye todos los parámetros necesarios.
La función, junto con los parámetros, se empaqueta y se realiza una llamada al sistema para enviarlos a través de la red.
El sistema operativo envía los datos a través de la red.
El <i>host</i> remoto recibe los datos y los pasa al proceso del servidor, que desempaqueta la función y los parámetros.
Los parámetros se pasan al procedimiento local y el procedimiento es entonces ejecutado.
El resultado se envía al cliente de la misma manera.

Tabla 1. Comunicación en la llamada a procedimiento remoto. Fuente: elaboración propia.

Los siguientes apartados tratarán sobre algunos **protocolos de nivel de aplicación**. La lista de ejemplos es muy corta comparada con la cantidad de protocolos existentes.

### 4.3. Telnet

Telnet ofrece un **canal de comunicación** relativamente genérico y bidireccional de 8 bits. El objetivo principal es servir de **método estándar de interfaz** para dispositivos de terminal y procesos orientados a terminal. Una **sesión de Telnet** consiste en un **terminal virtual de red** o *Network Virtual Terminal* (NVT).

---

Para obtener más información sobre el protocolo, puedes consultar completo el documento *Telnet Protocol Specification* a través del aula virtual o desde la siguiente dirección web: <https://datatracker.ietf.org/doc/html/rfc854>

---

Un NVT es un dispositivo lógico que ofrece una representación a través de la red de un terminal estándar.

## Protocolo

Telnet usa TCP como **protocolo de capa 4**, por lo que aprovecha su garantía de entrega en orden. El protocolo, como tal, es extremadamente sencillo: cada *byte* tecleado por el usuario en el terminal virtual, o enviado por el servidor, se transmite tal cual, sin cabeceras adicionales.

El carácter 255 (0xFF en hexadecimal) tiene el significado especial de *Interpret as command* (IAC), e indica que el siguiente *byte* o grupo de *bytes* será un **comando especial de Telnet** y no un **carácter de usuario**. Algunos de los comandos de Telnet se detallan en la Tabla 2.

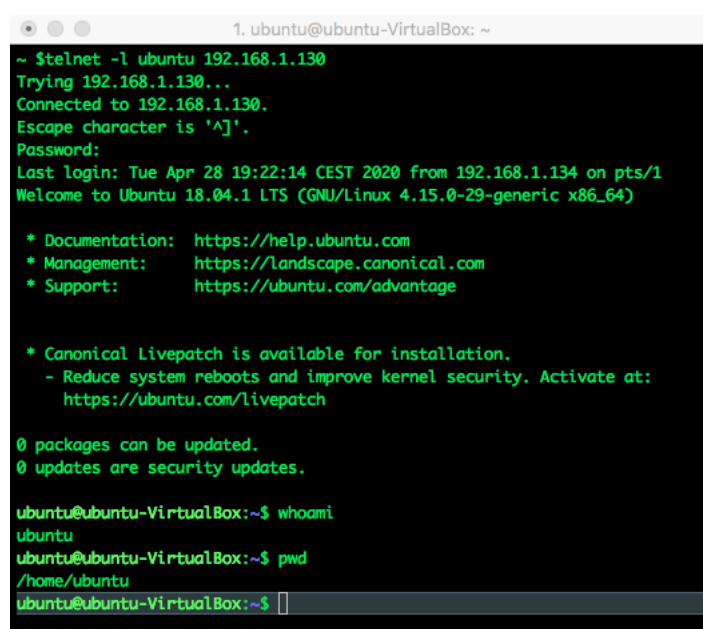
Comandos de Telnet		
	Código	Significado
NOP	0xF1	No Operation
Interrupt Process	0xF4	Función interna IP
Abort Output	0xF5	Función interna AO
Are You There	0xF6	Función interna AYT
Erase character	0xF7	Borrar carácter
Erase line	0xF8	Borrar línea

Tabla 2. Comandos de Telnet. Fuente: adaptado de IETF, s. f.



Las **funciones IP, AO y AYT** son internas de Telnet. IP solicita la terminación del proceso al que está conectado el terminal virtual. **AO** indica que el proceso puede terminar satisfactoriamente, pero sin enviar la salida al cliente y **AYT** solicita que el otro extremo envíe un carácter visible a modo de acuse de recibo.

La utilidad más inmediata es el uso para inicio de **sesión remoto**. Un servidor Telnet escucha en el puerto **23** por defecto. Este uso está desaconsejado, ya que todo el contenido enviado durante la sesión podría ser leído por un **atacante**, que pudiera **acceder al flujo**. SSH es la alternativa habitual para inicio de sesión seguro, ya que todo el contenido en tránsito está cifrado.

A screenshot of a terminal window titled '1. ubuntu@ubuntu-VirtualBox: ~'. The terminal shows a Telnet session initiated with the command '\$ telnet -l ubuntu 192.168.1.130'. The output shows the connection process, including the escape character and password prompt. After logging in, the user is greeted with the Ubuntu version and system information. The terminal also displays links for documentation, management, and support, as well as information about Canonical Livepatch. Finally, the user runs 'whoami' and 'pwd' commands, which return 'ubuntu' and '/home/ubuntu' respectively.

```
~ $ telnet -l ubuntu 192.168.1.130
Trying 192.168.1.130...
Connected to 192.168.1.130.
Escape character is '^]'.
Password:
Last login: Tue Apr 28 19:22:14 CEST 2020 from 192.168.1.134 on pts/1
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-29-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch
0 packages can be updated.
0 updates are security updates.

ubuntu@ubuntu-VirtualBox:~$ whoami
ubuntu
ubuntu@ubuntu-VirtualBox:~$ pwd
/home/ubuntu
ubuntu@ubuntu-VirtualBox:~$
```

Figura 1. Inicio de sesión con Telnet en un *host* Linux. Fuente: elaboración propia.

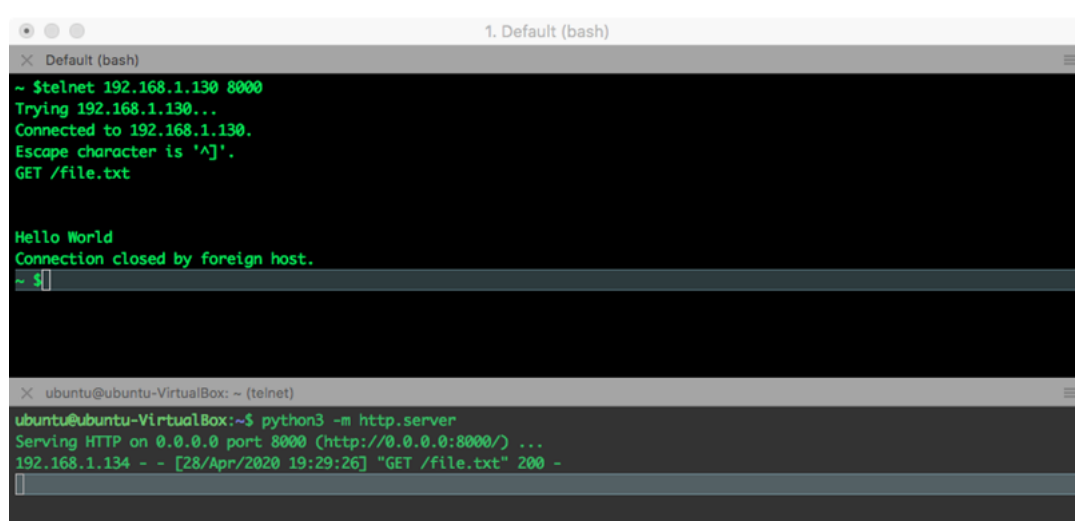
El protocolo es tan sencillo que es usado habitualmente para **abrir una conexión TCP**, para interactuar con el otro *host* en modo texto. Esta técnica puede ser para:

- ▶ Detectar si un equipo remoto tiene un *socket* activo en un puerto concreto.
- ▶ Interactuar con protocolos no binarios.

Por ejemplo, en la Figura 2 aparece un servidor HTTP sencillo que escucha en el puerto 8000 (en la parte inferior de la consola). El **cliente Telnet** se puede usar para

abrir una conexión con el *host* remoto en el puerto 8000, incluso cuando el proceso que ha abierto ese *socket* no es un **servidor de Telnet**, sino un **servidor HTTP**.

Dado que HTTP es un protocolo basado en texto, es posible enviar un **comando concreto**, en este caso, se envía `GET /file.txt`, seguido de dos saltos de línea. El servidor contesta con el **contenido del fichero**. Además, al ser un servidor de prueba, también muy sencillo, ni siquiera ha incluido **cabeceras**. Dado que el protocolo es HTTP 1.0, el servidor cierra la conexión al terminar la transferencia.



```
1. Default (bash)
X Default (bash)
~ $telnet 192.168.1.130 8000
Trying 192.168.1.130...
Connected to 192.168.1.130.
Escape character is '^]'.
GET /file.txt

Hello World
Connection closed by foreign host.
~ $

ubuntu@ubuntu-VirtualBox: ~ (telnet)
ubuntu@ubuntu-VirtualBox:~$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.1.134 - - [28/Apr/2020 19:29:26] "GET /file.txt" 200 -
```

Figura 2. Telnet como cliente HTTP. Fuente: elaboración propia.

## Aplicaciones

Las **utilidades de línea de comandos** `telnet` suelen estar disponibles en Windows, Linux y Mac OS, por defecto, o son fácilmente instalables. Los clientes de terminal que soportan SSH suelen soportar también Telnet. Por ejemplo, Putty soporta Telnet directamente cambiando el tipo de conexión, como se puede ver en la Figura 3.

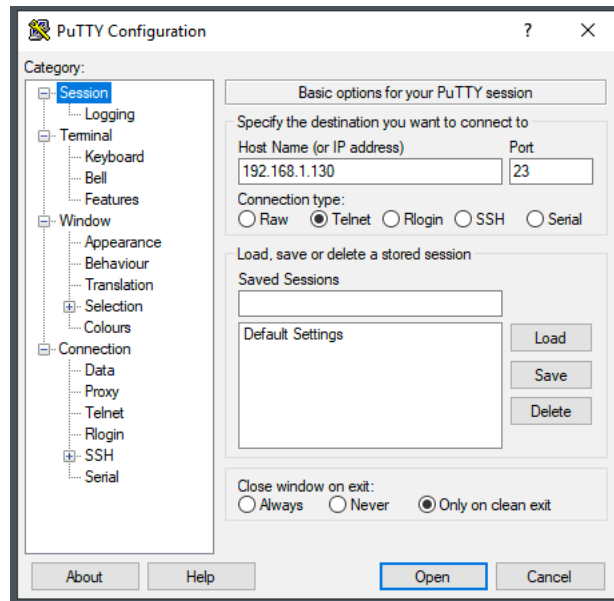


Figura 3. Putty configurado para conexión Telnet. Fuente: elaboración propia.

## 4.4. DNS

El **protocolo de nombres de dominio** o *Domain Name System* (DNS) facilita el mapeo entre nombres de equipos y direcciones IP. Los programas de software pueden trabajar con direcciones IP sin problema, pero, para usuarios finales, es más fácil **recordar** un **nombre de equipo** como ftp.rediris.es, que su IP 130.206.13.2.

DNS también facilita las **tareas de configuración**, ya que el administrador del *host* ftp.rediris.es puede decidir **cambiar la IP del equipo** (por una tarea de mantenimiento o por un fallo en el equipo), y no tiene que **notificar** a todos los usuarios del cambio de IP; simplemente debe **actualizar el registro** en el servidor DNS, y todos los clientes recibirán la nueva IP en la próxima consulta.

El volumen de nombres de equipo en Internet es muy elevado, así que DNS trabaja con un **modelo jerárquico basado en dominios** y una **base de datos distribuida**. Así, si un equipo necesita resolver la dirección del equipo ftp.rediris.es, envía una solicitud DNS a su servidor DNS local. Este no tiene por qué conocer la dirección, pero

enviará, a su vez, una solicitud a uno de los servidores raíz preguntando por el dominio `rediris.es`.

---

Puedes acceder al documento completo *Domain Names – Implementation and Specification* a través del aula virtual o desde la siguiente dirección web:

<https://datatracker.ietf.org/doc/html/rfc1035>

---

El **servidor raíz** contestará con la dirección de otro servidor DNS, el que aloja el dominio `rediris.es`. El servidor DNS local podrá, entonces, **preguntar al servidor** que aloja el dominio `rediris.es` por el nombre completo, `ftp.rediris.es`.

Los registros DNS no se limitan a nombres de equipo. Los **tipos más relevantes** de registros DNS se pueden consultar en la Tabla 3.

Registros DNS	
<b>A</b>	Dirección de un <i>host</i>
<b>NS</b>	Nombre de servidor DNS acreditado para esta zona
<b>CNAME</b>	Alias de un nombre de <i>host</i>
<b>MX</b>	Nombre de servidor de correo asociado al dominio
<b>TXT</b>	Cadena de texto

Tabla 3. Tipos de registros DNS. Fuente: elaboración propia.

## Protocolo

El protocolo trabaja sobre **UDP** usando el **puerto 53** por defecto. La elección de UDP se debe, principalmente, a razones de rendimiento. El número de peticiones DNS necesarias para resolver un único nombre es **elevado**, y el contenido de cada petición es relativamente pequeño, así que, el tamaño de las cabeceras TCP es comparable y añade una sobrecarga importante.

Además, las peticiones DNS son una **herramienta auxiliar** antes de iniciar el protocolo deseado, por lo que es deseable una reducción en la latencia inicial. Aunque, formalmente, **DNS no es confiable**, no hay más que usar Internet para comprobar que funciona como se espera.

Aunque las peticiones de resolución de nombres usan UDP como protocolo de transporte, los servidores DNS se comunican entre ellos mediante **TCP en el puerto 53**. Esta comunicación se usa para el **intercambio y sincronización** de zonas DNS. Este mecanismo permite que una actualización de un registro DNS se pueda replicar a otros servidores sin intervención manual.

---

Puedes acceder al documento completo *DNS Zone Transfer Protocol (AXFR)* a través del aula virtual o desde la siguiente dirección web:

<https://datatracker.ietf.org/doc/html/rfc5936#section-2>

---

## Aplicaciones

Los sistemas operativos incorporan llamadas de sistema para la resolución de nombres, sin necesidad de que las aplicaciones lo implementen independientemente.

De manera práctica, es habitual usar las **utilidades de línea** de comandos `ping` y `host` para obtener la IP de un nombre de *host*. Una utilidad avanzada es `dig`: permite ejecutar peticiones de diferentes tipos usando servidores DNS específicos (es decir, no los configurados por defecto en el sistema operativo). La Figura 4 muestra una petición MX a un servidor DNS específico.

```
1. ubuntu@ubuntu-VirtualBox: ~  
ubuntu@ubuntu-VirtualBox:~$ dig mx unir.net 8.8.8.8  
  
;<> DIG 9.11.3-1ubuntu1.1-Ubuntu <> mx unir.net 8.8.8.8  
;; global options: +cmd  
;; Got answer:  
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 43936  
;; Flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1  
  
;; OPT PSEUDOSECTION:  
; EDNS: version: 0, flags:; udp: 65494  
;; QUESTION SECTION:  
;unir.net. IN MX  
  
;; ANSWER SECTION:  
unir.net. 3556 IN MX 0 unir-net.mail.protection.outlook.com.  
  
;; Query time: 0 msec  
;; SERVER: 127.0.0.53#53(127.0.0.53)  
;; WHEN: Wed Apr 29 10:26:05 CEST 2020  
;; MSG SIZE rcvd: 89  
  
;; Got answer:  
;; ->HEADER<- opcode: QUERY, status: NXDOMAIN, id: 61932  
;; Flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1  
  
;; OPT PSEUDOSECTION:  
; EDNS: version: 0, flags:; udp: 65494  
;; QUESTION SECTION:  
;8.8.8.8. IN MX  
  
;; Query time: 1 msec  
;; SERVER: 127.0.0.53#53(127.0.0.53)  
;; WHEN: Wed Apr 29 10:26:05 CEST 2020  
;; MSG SIZE rcvd: 36  
  
ubuntu@ubuntu-VirtualBox:~$
```

Figura 4. Comando dig con una petición de DNS de tipo MX en un servidor DNS específico. Fuente: elaboración propia.

En el mercado hay **múltiples opciones** de servidores DNS. Algunos de los habituales son BIND para equipos Unix y Linux, y la implementación de DNS de Microsoft, incorporada en **Windows Server**.

## 4.5. SMTP

El **protocolo de transferencia de correo simple** (SMTP) se utiliza para el intercambio de mensajes de correo electrónico entre un cliente y un servidor, o entre servidores (IETF, s. f.). En un **cliente de escritorio**, un agente de la aplicación se encarga de **gestionar** el envío al servidor mediante SMTP, mientras que el usuario final utiliza SMTP solo para **enviar** los correos electrónicos, los servidores normalmente utilizan SMTP tanto para **enviar** como para **recibir correos**.

---

Para obtener más información sobre el protocolo de transferencia, puedes acceder al documento *Simple Mail Transfer Protocol* a través del aula virtual o desde la siguiente dirección web: <https://datatracker.ietf.org/doc/html/rfc2821>

---

La recepción de correos en los clientes se suele llevar a cabo con POP o IMAP.

## Protocolo

Los clientes usan el **puerto TCP 587** como destino para el **envío de mensajes**, mientras que los servidores usan el puerto **TCP 25** tanto para el envío como para la recepción.

SMTP es un protocolo, basado en texto, en el que el cliente y el servidor **intercambian comandos y respuestas**. Los comandos son cadenas de texto, seguidas de los parámetros, por ejemplo, `MAIL FROM:user@example.com`. Las respuestas empiezan por un **código numérico** y una **descripción**, por ejemplo, `250 Ok`.

Al ser un protocolo de texto, es posible **simular una sesión** usando Telnet. La Figura 5 muestra un ejemplo en el que se usan los comandos `HELO`, `MAIL FROM`, `RCPT TO`, `DATA` y `QUIT`.

```
2. Default (bash)
~ $telnet mailserver.local 25
Trying 192.168.1.130...
Connected to mailserver.local.
Escape character is '^J'.
220 SMTP Servidor de Prueba (Ubuntu)
HELO cliente.local
250 ubuntu-VirtualBox.home
MAIL FROM:<ubuntu@mailserver.local>
250 2.1.0 Ok
RCPT TO:<root@mailserver.local>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
From: "Ubuntu" <ubuntu@mailserver.local>
To: ROOT <root@mailserver.local>
Date: Wed, Apr 29 2020 11:09:45
Subject: Mensaje de prueba

Hello Root:
Este mensaje de prueba se envia a traves de telnet.

Greetings,
Ubuntu
.
250 2.0.0 Ok: queued as 3A04926D21
QUIT
221 2.0.0 Bye
Connection closed by foreign host.
~ $
```

Figura 5. Sesión SMTP usando Telnet. Fuente: elaboración propia.

## Aplicaciones

Las aplicaciones de cliente implementan **SMTP** para el envío de correos, y **POP** e **IMAP** para la recepción. Se ha extendido el **uso de protocolos** propietarios como Exchange ActiveSync, que ofrecen funcionalidades adicionales.

A **nivel de servidor** se pueden citar Postfix para Linux y Microsoft Exchange para Windows.

## 4.6. POP

Aunque los clientes de correo usan SMTP para el envío de mensajes, usan POP (*Post Office Protocol*) para la **descarga de mensajes nuevos**. IMAP es otro protocolo que también permite el envío y la recepción.



El protocolo funciona sobre el **puerto TCP 110**. En el modo *delete*, el servidor borra los mensajes una vez descargados por el cliente. En el modo *keep*, tanto el servidor como el cliente mantienen los mensajes, hasta que el usuario los borra.

## 4.7. FTP

El *File Transfer Protocol* es el protocolo más utilizado para la **transferencia de archivos en la red**, o, al menos, lo era antes de la popularización de las redes de intercambio *peer-to-peer*. FTP utiliza TCP como capa de transporte, y utiliza **diferentes puertos** en función del modo de funcionamiento:

- ▶ El cliente siempre inicia una conexión de control al puerto TCP 21.
- ▶ En modo activo, el cliente abre un *socket* en un puerto aleatorio e indica al servidor que inicie la transferencia de datos en ese puerto.
- ▶ En modo pasivo, es el servidor el que abre un *socket* adicional en un puerto aleatorio y le indica al cliente que lo use para sus transferencias.

El modo activo no funcionará en situaciones con **firewalls** o en las que el cliente está detrás de un **NAT**, ya que el puerto, en el cliente, no será accesible desde la red del servidor.

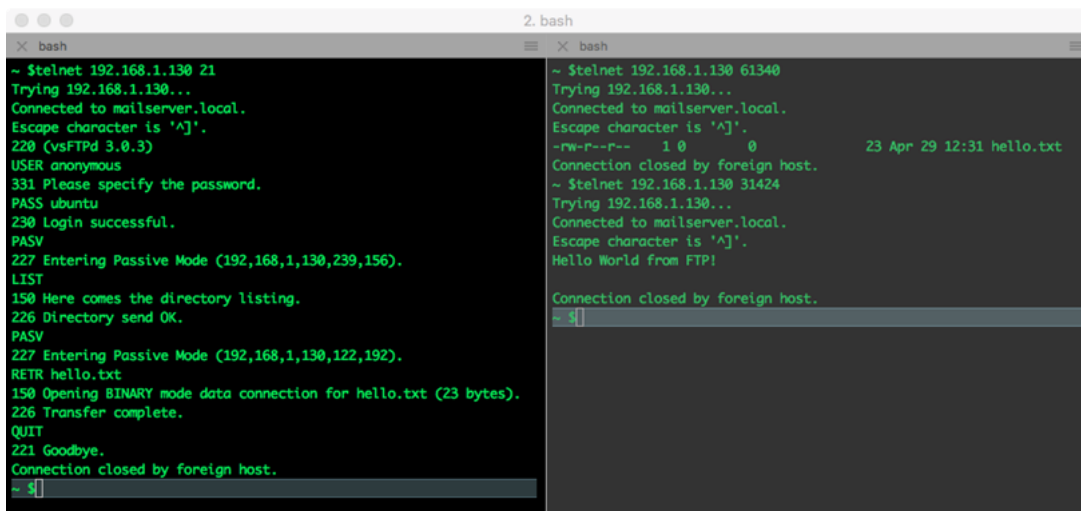
### Protocolo

FTP también es un protocolo basado en texto. Los comandos se transmiten en el **canal de control abierto** al iniciar la sesión en el puerto 21, y el resultado de los comandos se devuelve bien en el puerto abierto por el servidor en **modo pasivo**, o en el puerto abierto por el cliente en **modo activo**.

El ejemplo de la Figura 6 muestra una **conexión de control** en la consola izquierda en la que se inicia sesión con los comandos USER y PASS. A continuación, se activa el

modo pasivo con PASV, a lo que el servidor responde con una dirección IP y un puerto (los dos últimos números de la respuesta a PASV son el **número de puerto**, pero representado con el valor en decimal de 2 *bytes*, que, en realidad, es un número entero de 16 bits).

El siguiente comando es LIST para solicitar un **listado de ficheros**, y en la consola derecha se abre un canal TCP al **puerto abierto** por el servidor para este comando. Para recuperar el archivo hello.txt, se activa de nuevo el modo pasivo, se ejecuta el comando RETR hello.txt y se abre otra conexión TCP, también en la consola de la derecha, donde aparece el contenido del fichero. Para terminar, se cierra la **conexión de control** con el comando QUIT.



```
2. bash
~ $telnet 192.168.1.130 21
Trying 192.168.1.130...
Connected to mailserver.local.
Escape character is '^J'.
220 (vsFTPd 3.0.3)
USER anonymous
331 Please specify the password.
PASS ubuntu
230 Login successful.
PASV
227 Entering Passive Mode (192,168,1,130,239,156).
LIST
150 Here comes the directory listing.
226 Directory send OK.
PASV
227 Entering Passive Mode (192,168,1,130,122,192).
RETR hello.txt
150 Opening BINARY mode data connection for hello.txt (23 bytes).
226 Transfer complete.
QUIT
221 Goodbye.
Connection closed by foreign host.
~ $

~ $telnet 192.168.1.130 61340
Trying 192.168.1.130...
Connected to mailserver.local.
Escape character is '^J'.
-rw-r--r--  1 0      0      23 Apr 29 12:31 hello.txt
~ $telnet 192.168.1.130 31424
Trying 192.168.1.130...
Connected to mailserver.local.
Escape character is '^J'.
Hello World from FTP!
Connection closed by foreign host.
~ $
```

Figura 6. Sesión FTP sobre Telnet. Fuente: elaboración propia.

## Aplicaciones

Hay **múltiples clientes de FTP**: Filezilla, Cute FTP, Cyberduck, etc. Los navegadores web suelen incluir un cliente FTP para poder **recuperar ficheros** a partir de enlace en las páginas web, aunque no ofrecen toda la **funcionalidad** de un cliente habitual.

## 4.8. HTTP

HTTP es el protocolo por antonomasia en Internet. En el escenario más sencillo, un **navegador web** muestra una página HTML, recuperado del servidor, con una petición HTTP. Los **recursos adicionales** referenciados en el HTML, como ficheros de scripts, estilos CSS e imágenes, se recuperan con más peticiones HTTP.

Este modelo sencillo se puede complicar, ya que HTTP **no se limita** a páginas HTML estáticas. Los servidores pueden generar las respuestas de manera **dinámica** y los *scripts* puede hacer, a su vez, llamadas adicionales para incrementar la **interactividad** de las páginas.

### Protocolo

Al igual que SMTP, es un protocolo basado en texto que funciona sobre TCP (IETF, s. f.). El **puerto por defecto es 80**. Las peticiones consisten en:

- ▶ Una primera línea con un «método», también llamados «verbo», seguido de una «URL». La URL define un recurso en el servidor al que se envía la petición. El método solicita una acción sobre el recurso. La línea termina con la versión HTTP que el cliente desea utilizar.
- ▶ Una lista de «cabeceras de petición» (o *request headers*), uno por línea, que actúan como metadatos de la petición. Cada cabecera incluye información sobre el cliente, la petición, el contenido de la petición o lo que se espera en la respuesta. El estándar define una serie de cabeceras, pero se pueden definir cabeceras personalizadas, siempre y cuando, tanto el servidor como el cliente, estén de acuerdo en su significado. De las cabeceras definidas por el estándar, solo *Host* es estrictamente necesaria, aunque un servidor puede exigir otras, según proceda. Las cabeceras más habituales se detallan en la Tabla 6.

- ▶ Una línea vacía, seguida del cuerpo de la petición, si procede.
- ▶ Una última línea vacía. Si la petición no incluye cuerpo, la petición acaba con dos saltos de línea tras las cabeceras.

Los principales métodos y su significado están detallados en la Tabla 4.

Principales métodos HTTP			
	Significado	La petición incluye cuerpo	La respuesta incluye cuerpo
<b>GET</b>	Solicita la transmisión del recurso.	Opcional	Si
<b>HEAD</b>	Solicita solo las cabeceras de la petición GET del recurso. Útil para comprobar si un recurso ha cambiado, o el tamaño que tiene, sin solicitar el recurso completo.	Opcional	No
<b>POST</b>	Solicita el procesamiento del cuerpo de la petición por parte del recurso. Puede implicar la creación de un archivo, de un objeto en una base de datos o el procesamiento de un formulario web.	Si	Si
<b>PUT</b>	Solicita el reemplazo del recurso con el contenido del cuerpo de la petición.	Si	Si
<b>DELETE</b>	Solicita el borrado del recurso	Opcional	Opcional
<b>OPTIONS</b>	Solicita información sobre las opciones disponibles en el recurso. El servidor puede, por ejemplo, contestar con los métodos disponibles en el recurso	Opcional	Opcional

Tabla 4. Principales métodos HTTP. Fuente: elaboración propia.

La **respuesta**, también en modo texto, consiste en:

- ▶ Una línea de estado con la versión del protocolo y un **código** de respuesta, seguido, opcionalmente, de una descripción del código. El código es un número de tres cifras.
- ▶ Una lista de **cabeceras de respuesta**. Al igual que las cabeceras de la petición, el estándar define algunos, pero es una lista extensible. Pueden contener información del servidor, del recurso, etc.
- ▶ Una línea vacía, seguida del cuerpo de la respuesta.

El **estándar** define una lista de códigos de respuesta agrupados en cinco secciones. Al igual que las cabeceras, los códigos son **extensibles**. El primer dígito del código define la clase de respuesta:

- ▶ 1xx – Información: petición recibida, continuando el procesado.
- ▶ 2xx – Éxito: la petición se recibió y ha sido procesada con éxito.
- ▶ 3xx – Redirección: el cliente debe tomar acción para terminar de completar la petición.
- ▶ 4xx – Error de cliente: la petición no puede completarse por un error en la petición.
- ▶ 5xx – Error de servidor: la petición puede ser válida, pero el servidor no ha podido procesarla por algún otro error.

Los códigos más habituales, y su significado, se detallan en la Tabla 5.

---

La lista completa de códigos puede consultarse a través del aula virtual o desde la siguiente dirección web:

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

---

Principales métodos HTTP		
	Código y descripción	Significado
1xx	100 Continue	El cliente ha enviado una petición sin cuerpo y el servidor ha aceptado que el cliente envíe ahora la petición.
	101 Switching Protocols	El servidor acepta el cambio de protocolo que ha solicitado el cliente. Se usa, por ejemplo, para abrir túneles de <i>websockets</i> .
2xx	200 OK	Respuesta estándar para una petición completada con éxito.
	201 Created	La petición fue completada, resultando en la creación de un recurso.
	202 Accepted	El servidor ha aceptado la petición, pero no ha terminado de procesarla. El cliente deberá tomar acción si quiere confirmar que la petición ha sido completada.
	204 No Content	La petición se ha completado con éxito, pero el servidor no incluye contenido en la respuesta.
3xx	302 Found	El recurso se encuentra ubicado en otra URL. El servidor incluirá la URL correcta en la cabecera «Location».
4xx	400 Bad Request	La petición tiene algún tipo de error: desde cuerpo inválido, tamaño excesivo, hasta error genérico. El cuerpo de la petición suele contener un mensaje de error más específico.
	401 Unauthorized	El cliente debe incluir detalles de autenticación. Los detalles serán especificados por el servidor: puede ser una autenticación básica, incluyendo usuario y contraseña en la cabecera «Authentication», o incluir algún tipo de <i>cookie</i> .
	403 Forbidden	La petición es válida, pero el servidor ha decidido no atenderla por falta de permisos. Es diferente de 401 Unauthorized en el sentido de que el usuario se ha identificado correctamente, pero no tiene permisos sobre el recurso.
	404 Not Found	El recurso solicitado no existe.
	405 Method Not Allowed	El recurso puede existir, pero se ha solicitado con un método que no es válido. Por ejemplo, una petición POST puede no tener sentido sobre un contenido estático.
	429 Too Many Requests	El cliente ha enviado más peticiones en un intervalo de tiempo de lo que permite el servidor.
5xx	500 Internal Server Error	Error genérico interno del servidor. La petición podría ser válida, pero, en el peor de los casos, el servidor no ha llegado a evaluarla.
	502 Bad Gateway	El servidor que ha recibido la petición actúa como intermediario (por ejemplo, como <i>proxy</i> inverso) y el servidor al que ha delegado la petición no está disponible.
	504 Gateway Timeout	Similar al error 502, pero más específico.

Tabla 5. Códigos de respuesta HTTP más habituales. Fuente: elaboración propia.

Principales cabeceras HTTP		
	Significado	Tipo
<b>Authorization</b>	Contiene credenciales para identificar al usuario. Puede ser de tipo Basic, que incluye un usuario y contraseña en formato Base64, o en formato Bearer, con una cadena de texto, normalmente firmada, que contiene información de identificación, pero sin credenciales.	Petición
<b>Accept</b>	Informa al servidor del formato de datos MIME esperado por el cliente.	Petición
<b>Cookie</b>	Contiene las <i>cookies</i> fijadas por el servidor en un dominio y ruta dados.	Petición
<b>Set-Cookie</b>	Entrega una cookie, cuyo contenido es texto arbitrario, que el cliente deberá incluir en futuras peticiones al mismo dominio de esta respuesta. La aplicación se encargará de interpretar el contenido de la <i>cookie</i> , que puede ser desde texto plano a tokens JWT o contenido cifrado. Se puede usar para mantener estado de la sesión, detalles de autenticación, <i>tracking</i> de publicidad, etc.	Respuesta
<b>Content-Length</b>	Tamaño del contenido (sin contar cabeceras) en <i>bytes</i> .	Petición o respuesta
<b>Content-Type</b>	Tipo de datos MIME del contenido.	Petición o respuesta
<b>Location</b>	URL a la que debe dirigirse el cliente para conseguir el recurso. Suele incluirse en respuestas con código 302.	Respuesta
<b>Host</b>	Especifica el dominio del servidor al que se dirige la petición. Permite que un único proceso atienda peticiones para más de un dominio.	Petición
<b>User-Agent</b>	Identifica al cliente: aplicación, versión, sistema operativo, etc.	Petición
<b>Server</b>	Identifica al servidor: aplicación, versión, etc.	Respuesta

Tabla 6. Cabeceras HTTP más comunes. Fuente: elaboración propia.

Puedes conocer más acerca de los *tokens* JWT a través del aula virtual o desde la siguiente dirección web:

<https://jwt.io/>

## Web Services

Una arquitectura basada en *web services* es un **modelo de interoperabilidad** de sistemas mediante servicios distribuidos, basados en **estándares web**. Un caso concreto son los **sistemas REST o RESTful** (*Representational State Transfer*).

Aunque no hay una definición concreta, se puede hablar de REST como un **estilo de arquitectura** con una serie de restricciones para el desarrollo de interfaces de aplicaciones (Archip, *et al*, 2018), concretamente:

- ▶ Los recursos se identifican con un esquema de URI (*Uniform Resource Identifier*), similar a las URL de las páginas web.
- ▶ Los recursos tienen una representación en *bytes* y unos metadatos asociados.
- ▶ Solo unos pocos métodos son permitidos sobre los recursos, y tienen el mismo significado sobre todos los recursos.
- ▶ Las interacciones no mantienen estado: cada petición termina completamente, bien con éxito o con fallo.
- ▶ El comportamiento idempotente de las operaciones es deseable.
- ▶ Entidades intermedias para *proxy*, caché u otras interacciones son deseables.

En la práctica, es habitual encontrar **interfaces REST** basadas en **HTTP**. Cuanto más se ciña una API al estándar HTTP, más fácil será que otras aplicaciones se **integren** con ella, ya que aprovechará el comportamiento de clientes HTTP ya existentes.

Estos servicios web se ofrecen para ser **consumidos programáticamente**, no interactivamente, por un usuario.



## API REST de *Oxford Languages*



Figura 7. Página de inicio de Oxford English Dictionary. Fuente: Oxford English Dictionary, 2021.

La plataforma de diccionarios `languages.oup.com` es un claro ejemplo de un servicio ofrecido con una interfaz REST por HTTP.

La web funciona como servicio de diccionarios a usuarios de la manera habitual: algunas consultas no necesitan inicio de sesión, otros servicios necesitan inicio de sesión, pero son gratuitos, y otros requieren una suscripción. Dispone de blog, noticias, redes sociales, etc. La interfaz REST, sin embargo, no está pensada para el consumo por parte de los usuarios directamente. El objetivo de la API es ofrecer el servicio a otras aplicaciones.

Por ejemplo, una compañía cliente podría integrar una aplicación de lectura con las traducciones ofrecidas por la API, mientras que otra podría diseñar una aplicación móvil que se alimente de la API, delegando en un tercero la funcionalidad de diccionario para centrarse en la experiencia de usuario.

El funcionamiento de esta API se basa, totalmente, en el protocolo HTTP. Para solicitar la definición de una palabra, por ejemplo, hay que solicitar el recurso `/entries/<idioma>/<palabra>` con el método GET, incluyendo dos cabeceras propias de la aplicación, `app_id` y `app_key`, que sirven de autenticación y autorización. Una petición para la definición de libro sería como sigue:

```
GET /api/v2/entries/es/libro?fields=definitions HTTP/1.0
Host: od-api.oxforddictionaries.com
User-Agent: curl/7.54.0
Accept: */*
app_id: 123456
app_key: 5241fe132e2ee98fd76bb558013a568e
```

Las cabeceras de la respuesta serían similares a las siguientes:

```
HTTP/1.1 200 OK
Date: Thu, 30 Apr 2020 07:11:59 GMT
Content-Type: application/json;charset=utf-8
Content-Length: 3232
Connection: close
Server: openresty/1.13.6.2
X-Request-Id: 1-5eaa7a3f-a56cd13c60501714ccd629bd
code_version: leap2-v2.6.0-ge749fd1
api_version: v2
```

Y el contenido sería un documento JSON (tal como indica la cabecera Content-Type) de 3.2 KB (tal como indica Content-Length). El JSON contiene metadatos y un *array senses* con múltiples elementos. Algunos de ellos se muestran a continuación.

```
{
  "definitions": [
    "Conjunto de hojas de papel, pergamino, vitela, etc., manuscritas
    o impresas, unidas por uno de sus lados y normalmente encuadernadas,
    formando un solo volumen"
  ],
  "id": "idc0605f74-7d2a-4571-8e3c-2220dc44d012"
},
{
  "definitions": [
    "Conjunto de hojas unidas formando un volumen que se rellena con
    distintos datos para llevar un registro; suele tener una parte impresa
    con blancos para ser rellenos con los datos"
  ],
  "id": "id2b812530-0983-4fba-a378-ce476dc52901"
},
{
  "definitions": [
    "División de una obra o de códigos y leyes de gran extensión que
    tiene unidad de contenido y podría constituir un volumen"
  ],
  "id": "idb3ffd7d4-cf85-4963-9173-15a72c7bf8c2"
},
...
```

Las aplicaciones integradas con esta API podrían entonces interpretar este contenido para mostrar definiciones, etimología o pronunciaciones con una simple llamada HTTP.

## Aplicaciones

El ejemplo más claro de **aplicaciones HTTP cliente** son los **navegadores web** (que, además, incorporan la funcionalidad de renderización HTML y de ejecución de JavaScript, que no es parte del protocolo HTTP). Cada vez más aplicaciones hacen uso de **HTTP** y de **interfaces REST** para funcionalidades que tradicionalmente se habrían implementado con protocolos propietarios.

También, hay clientes que exponen la **funcionalidad HTTP al usuario**. Postman, por ejemplo, permite construir peticiones HTTP, indicando todos los parámetros: métodos, cabeceras, contenido, etc. La Figura 8 muestra la misma llamada a `oxforddictionaries.com` del caso práctico de este mismo apartado. La utilidad de línea de comandos `curl` ofrece la misma **funcionalidad**.

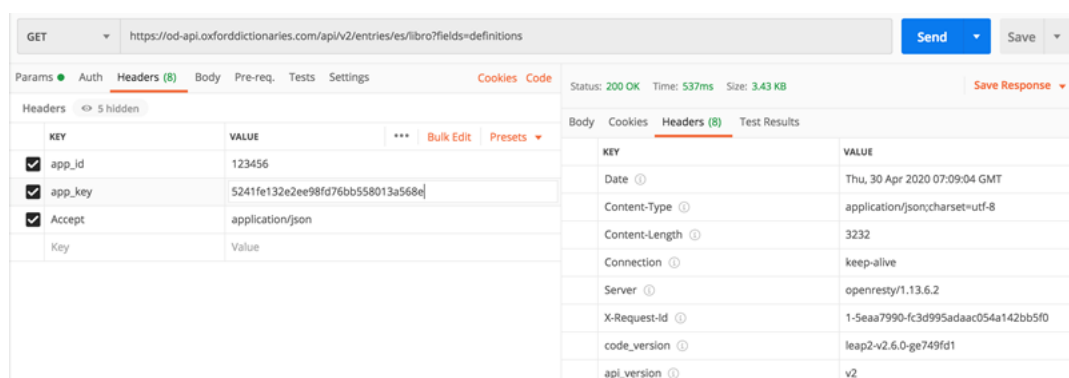


Figura 8. Interfaz de Postman con las cabeceras de la petición HTTP (izquierda) y de la respuesta (derecha) en una llamada a `oxforddictionaries.com`. Fuente: elaboración propia.

En el lado del servidor también hay una gran oferta. El servidor [web Apache](#) es uno de los más extendidos. Aparte de **software específico**, hay multitud de *frameworks* que facilitan el desarrollo de servidores web a medida, como [Ruby on Rails](#) o [Express.js](#).

## 4.9. NTP

*Network Time Protocol* (NTP), es un **protocolo de sincronización de reloj**. Un equipo puede fijar su reloj local tras una petición a un servidor, sin necesidad de intervención del usuario. Funciona a través de **redes con una latencia variable**.

---

Puedes acceder al documento *Network Time Protocolo (NTP)* a través del aula virtual o desde la siguiente dirección web:

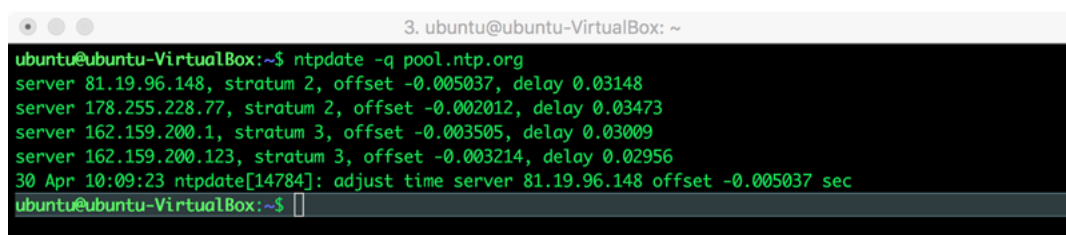
<https://datatracker.ietf.org/doc/html/rfc958>

---

Usa una **arquitectura de cliente-servidor** sobre UDP en el puerto 123. Cliente y servidor usan un algoritmo sencillo para calcular el **retardo en la transmisión** y la diferencia entre los relojes, en función de los tiempos en los que se **emite y recibe** cada paquete.

Dado que estos tiempos son relevantes para el **cálculo de la diferencia entre relojes**, NTP no puede depender de TCP, ya que este añade retardo adicional durante el inicio de la sesión.

Prácticamente todos los **sistemas operativos** incorporan un **cliente NTP** para la sincronización del reloj del sistema. En Linux es posible sincronizar el reloj manualmente con *ntpdate*. Este comando, además, muestra en la consola la comprobación, la diferencia del retardo y la diferencia de los relojes, como se muestra en la Figura 9.



```
3. ubuntu@ubuntu-VirtualBox: ~  
ubuntu@ubuntu-VirtualBox:~$ ntpdate -q pool.ntp.org  
server 81.19.96.148, stratum 2, offset -0.005037, delay 0.03148  
server 178.255.228.77, stratum 2, offset -0.002012, delay 0.03473  
server 162.159.200.1, stratum 3, offset -0.003505, delay 0.03009  
server 162.159.200.123, stratum 3, offset -0.003214, delay 0.02956  
30 Apr 10:09:23 ntpdate[14784]: adjust time server 81.19.96.148 offset -0.005037 sec  
ubuntu@ubuntu-VirtualBox:~$
```

Figura 9. *Ntpdate* sincronizado con el servidor público pool.ntp.org. Fuente: elaboración propia.

## 4.10. SSL

Varios de los protocolos mencionados, hasta ahora, se podían simular a través de una **conexión Telnet** porque estaban basados en texto. Ese modelo hace que el protocolo sea **sencillo de implementar**, pero a costa de la privacidad y seguridad: cualquier dispositivo de red, en el camino de tránsito, podría **capturar los paquetes** y descifrar o manipular el contenido.

Algunas soluciones de VPN incorporan **cifrado a nivel 2 y 3** en la interconexión de red, pero el **tráfico de aplicación** sería aún vulnerable en la red local. **SSL** viene a cubrir ese hueco (Hirsh, 1997) (Apache, s. f.).

Se presenta el **protocolo SSL** en relación con los servidores web, aunque ese no es su único ámbito de funcionamiento. Antes de hablar del protocolo en sí, se presentarán **conceptos de criptografía** necesarios para entender por qué SSL es seguro y confiable:

- ▶ Algoritmos.
- ▶ Funciones de resumen.
- ▶ Firmas.
- ▶ Certificados.
- ▶ Autoridades de certificación.

---

Los algoritmos criptográficos y las funciones resumen requieren de una base matemática muy profunda. En la sección A fondo se menciona el libro *Cryptography Engineering*, de Ferguson, *et al*, sobre estos temas, con un enfoque muy práctico.

---

## Algoritmos

En una situación de ejemplo, un usuario A quiere enviar un mensaje confidencial a un usuario B. El mensaje contiene información personal, por lo que el usuario A querría que sea privado y solo legible por el usuario B. Una solicitud de transferencia a un banco es un claro ejemplo, ya que el mensaje incluye datos personales del usuario A. Un algoritmo criptográfico permitirá al usuario A convertir el mensaje inicial en un mensaje ilegible, es decir, cifrarlo, a menos que se le aplique la técnica complementaria de descifrado. Para el cifrado se usa una clave secreta que solo deben conocer los usuarios A y B, ya que esa clave se usará en el proceso de descifrado. Se podría intentar descifrar el mensaje por fuerza bruta, pero el objetivo del algoritmo es que no merezca la pena.

Según Apache (s. f.): «Good cryptographic algorithms make it so difficult for intruders to decode the original text that it isn't worth their effort» que se traduce como: «Los buenos algoritmos criptográficos hacen que sea tan **difícil** para los **intrusos** decodificar el texto original, que no vale la pena su esfuerzo».

No siempre la **clave** que cifra el mensaje original es la misma que descifra el mensaje. Esto depende del tipo de algoritmo.

## Criptografía simétrica

Esta familia de **técnicas**, también conocida como criptografía convencional, usa la **misma clave** para cifrar y para descifrar los mensajes. Por tanto, requiere que ambos extremos de la comunicación (es decir, los usuarios o los agentes de software) compartan la clave.

Estas técnicas tienen un **punto débil** en el intercambio de la clave: necesitan un **canal seguro** para compartirla. Si se ha podido compartir en secreto (por un segundo canal confiable o en persona), el algoritmo puede ser tan **seguro** como permita técnicamente. Es decir, el problema del intercambio de la clave no es una **propiedad de seguridad** de cada algoritmo simétrico, sino un problema de esta familia en general. La siguiente familia de algoritmos viene a dar una **solución al problema**.

## Criptografía asimétrica

Estos algoritmos también se conocen como de **clave pública**. Resuelven el problema del intercambio de claves definiendo un **algoritmo** que usa **dos claves** (conocidas como *key pair* o pareja de claves), cada una de las cuales puede usarse para **cifrar un mensaje**.

Cuando se **cifra un mensaje** con una de las **claves**, hay que usar la otra para descifrarlo. Los usuarios ya no tienen que compartir una misma clave, sino que es suficiente que compartan una de las dos claves. Siguiendo con el ejemplo anterior, si el usuario A quiere que solo el usuario B lea el contenido del mensaje, cifrará el mensaje con la clave pública de B. Es decir, B siempre compartirá la misma clave, considerada pública, con **cualquier usuario** de quien necesite recibir mensajes.

Deberá mantener la otra **clave**, la **privada**, en **secreto**. Esta clave privada será la única que permite descifrar los mensajes cifrados con la clave pública de B. No hace falta compartir la clave pública por un canal seguro, hay **otras implicaciones** a la hora de compartir esta clave.

## Funciones de resumen

Se ha hablado del problema de la **confidencialidad**, pero no de la **integridad**. Si un usuario C intercepta el mensaje de A, C puede modificar el mensaje a su gusto, o sustituirlo completamente, sin que B tenga constancia de si el mensaje es el mismo que ha enviado A.

La integridad se puede conseguir si el **usuario A** genera un **resumen** a partir del mensaje original y lo envía a B. A la recepción de ambos, B calcula el resumen con el mismo **algoritmo** que usó A, y compara el resumen recibido con el generado. Si los resúmenes coinciden, B puede estar seguro de que el mensaje **no ha sido alterado** en tránsito.

Estos resúmenes se calculan con **funciones de resumen**, también llamadas *hash* o *digest*. Las funciones *hash* reciben un mensaje de longitud variable y generan un resumen de longitud fija. Entre las propiedades deseables de estas funciones están:

PROPIEDADES DESEABLES DE LAS FUNCIONES DE RESUMEN
Que sean rápidas.
Que sea difícil obtener el mensaje a partir del resumen.
Que dos mensajes diferentes no puedan generar el mismo resumen. Esto, en teoría, no puede ocurrir, ya que el tamaño de los mensajes puede ser mucho mayor que el tamaño del resumen. Por ejemplo, si el resumen es de 40 bits, y se firman mensajes de 100 bits, forzosamente habrá más de un mensaje original para cada combinación de esos 40 bits. La propiedad deseable de las funciones resumen no es que esto no ocurra, sino que sea muy poco probable.

Tabla 7. Propiedades deseables de las funciones de resumen. Fuente: elaboración propia.

Estas propiedades **aumentan la dificultad** de sustituir un mensaje por otro para un mismo resumen dado.

El problema de la **integridad** del mensaje que solucionan los resúmenes queda eclipsado con el mismo problema que había al enviar el mensaje original: si el resumen **no se transmite de forma segura**, no sirve de nada. Si el usuario C intercepta tanto el mensaje como el resumen, nada le impide **crear su propio mensaje** y generar un resumen acorde al mismo. La solución a este problema son las firmas digitales.

### Firmas digitales

En el escenario de ejemplo, el **interés** de ambas partes es que el mensaje sea **secreto** y que el mensaje que llega a B proceda de A, sin que C haya alterado su **integridad**.



Una solución pasa por **generar el resumen** a partir del mensaje y firmar el resumen con la clave privada de A (ojo, no con la clave pública). B sigue estos **pasos** cuando recibe la transmisión:

- ▶ Descifra el mensaje usando su clave privada. Con esto han asegurado la privacidad del mensaje.
- ▶ Descifra el resumen, usando la clave pública de A. Con esto B se asegura de la autoría del mensaje.
- ▶ Calcula el resumen del mensaje localmente y lo compara con el resumen descifrado. Si coinciden, la integridad del mensaje está también asegurada.

## Certificados

El proceso anterior es teóricamente **válido** salvo por un aspecto: con la información que tienen, ninguno de los dos puede asegurar que la **clave pública** del otro es realmente de quien dice ser.

En estos casos, hace falta una tercera parte en la que **ambos confíen**. Esta tercera parte se denomina **autoridad de certificación** (CA, de *Certificate Authority*). Las CA emiten certificados que contienen la clave pública del sujeto en cuestión y están firmados por la propia CA. Así, si tanto A como B confían en la CA, pueden confiar en que la clave pública del certificado de A es realmente de A, y lo mismo ocurre con el certificado de B.

## Contenido del certificado

Esta confianza en una tercera parte permite validar la **identidad real** asociada a un certificado. Esta identidad puede ser una **persona física o jurídica**, un **servidor** (por ejemplo, identificado por el dominio) u otro tipo de **entidad**. La Tabla 8 resume el tipo de información que incluye un certificado y que identifica al sujeto.

Aparte de los datos identificativos, los certificados incluyen otros metadatos como el **período de validez** o **identificadores para uso de la CA**.

Información contenida en un certificado	
Sujeto	Nombre ( <i>distinguished name</i> ), clave pública
Emisor	Nombre ( <i>distinguished name</i> ), firma
Periodo de validez	Fecha de inicio y fecha de fin de validez
Información administrativa	Versión, número de serie
Información extendida	Uso del certificado, información de revocación, etc.

Tabla 8. Información contenida en un certificado. Fuente: elaboración propia.

El sujeto se identifica con un *distinguished name*, o **nombre distinguido**, en vez de un **nombre común**. Por ejemplo, el usuario A puede disponer de un **certificado digital** firmado por la **policía** del país en el que el nombre distinguido contenga su número de DNI, ya que, en el contexto de los individuos del país, el DNI lo identifica unívocamente.

Por otro lado, el mismo usuario A puede recibir un segundo certificado digital por parte de su **empresa**, en la que el nombre distinguido sea el **número de empleado**, seguido de su departamento y los datos de la empresa. Ambos certificados identifican al mismo individuo, pero cada uno en su contexto. Se definen mediante el **estándar X.509** (International Telecommunication Union, s. f.) y siguen una estructura a base de componentes y siglas que pueden consultarse en la Tabla 9.

Información de nombre distinguido			
	Abr.	Descripción	Ejemplo
Nombre común	CN	Nombre que identifica el certificado	CN=John Doe
Organización, compañía	O	El CN está asociado con esta organización	O=UNIR
Unidad organizativa	OU	El CN está asociado con esta OU (departamento, sección, etc.)	OU=Facultad de Historia
Ciudad, localidad	L	El CN está ubicado en esta ciudad	L=Madrid
Estado, provincia	ST	El CN está ubicado en este estado	ST=Madrid
País	C	El CN está ubicado en este país	C=ES

Tabla 9. Campos del nombre distinguido. Fuente: elaboración propia.

El formato de los nombres distinguidos es **versátil** y cada CA puede especificar su propia **estructura**, definiendo qué campos son requeridos. Por ejemplo, los navegadores web requieren que el CN de un certificado, que representa un servidor, coincida con un **patrón comodín** para el nombre de dominio de ese servidor, como \*.unir.net. La Figura 10 muestra un certificado de este tipo en el que el *Common Name* coincide con este dominio.

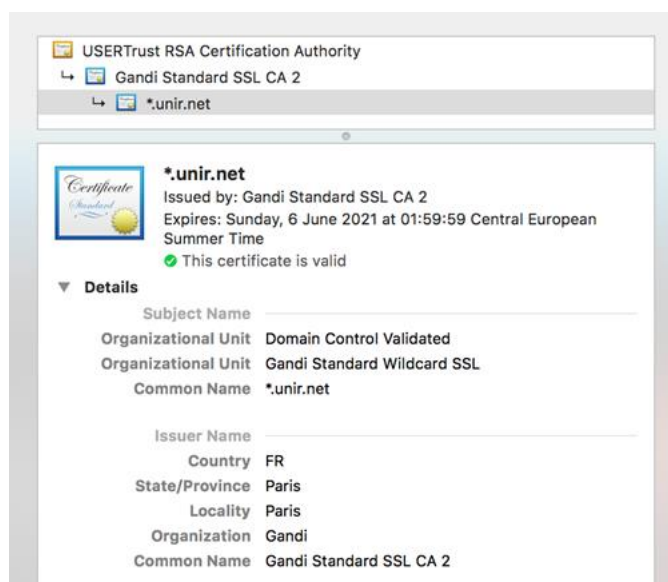


Figura 10. Certificado de \*.unir.net que muestra el CN. Fuente: elaboración propia.

Los certificados siguen el **formato binario ASN.1**, aunque, cuando la transmisión no puede ser binaria, se traducen a **formato ASCII**, utilizando la **codificación Base64**. Este contenido se encapsula entonces en un **archivo PEM** (*Privacy Enhanced Mail*) en el que se delimita el contenido en Base64 entre dos líneas, tal como se muestra a continuación.

```
-----BEGIN CERTIFICATE-----
MIICkzCCAFwCCQCNTXKhyVwgRzANBgkqhkiG9w0BAQsFADCBjTElMAkGA1UEBhMC
RVMxDzANBgNVBAGMBk1hZHJpZDEPMA0GA1UEBwwGTWFKcm1kMQ0wCwYDVQQKDARV
Tk1SMR0wGwYDVQQQLDBRGYWN1bHRhZCBkZSBIaXN0b3JpYTERMA8GA1UEAwwISm9o
biBEb2UxGzAZBgkqhkiG9w0BCQEWDGRvZUB1bmlyLm5ldDAeFw0yMDA1MTAxNDU3
NDZaFw0zMDA1MDgxNDU3NDZaMIGNMQswCQYDVQQGEwJFUzEPMA0GA1UECAwGTWFK
cm1kMQ8wDQYDVQQHDAZNYWRyaWQxDALBgNVBAoMBFVOSVIXHTAbBgNVBAsMFEZh
Y3VsdGFkIGRlIEhpc3RvcmlhMREwDwYDVQQDDAhkb2huIERvZTEbMBkGCSqGSIb3
DQEJARYMZG91QHVuaXIubmV0MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCl
Tjo92ngKZmfTA3oiJh8VDr13Nz66w7wH1Fv4KV/aDCQJwJ3Wtmfb+1EFsDgV48qE
c+j5ck3ZEKO0kcUu7iJKHgY57tPJLhaUCeJs4C6RYKdLG31A0XEYxj6fqHps0HhC
lXSfPZpb2p1nMeNVdreTfWoMlCxXbk9ueGqjR0NgtQIDAQABMA0GCSqGSIb3DQEB
CwUAA4GBAAX5j+cImEDjWYqBQUZYtYZHt6Ur+DdA3tAHh4F0HOTLxCxPjJ1/9bAJ
RS8TaUJ2vdex+Rj43wFch/1Muay1WzbuhRQPEXfzVW5ZtUMdxyFY//JjrDZ8M7FU
pK93iw3r7c9sH8Ql+IPNVj588FVw1CkK1HS2nbsuxyTiRzzRNWCh
-----END CERTIFICATE-----
```

## Autoridades de certificación

Las CA se encargan de **verificar la información** en una solicitud de certificado antes de concederlo. Es habitual que la CA solo se asegure de que el individuo posee las claves, **no de generarlas**.

Por ejemplo, un método para verificar los certificados para dominios de Internet es que el propietario del dominio incluya un **contenido firmado** en un **registro DNS** de tipo **TXT**.

Si la **autoridad de certificación** puede leer el **registro DNS**, y descifrarlo con la clave pública del solicitante, podrá emitir un certificado porque ha verificado que la clave que ha usado es del **propietario del dominio**, ya que este ha sido capaz de modificar los registros DNS de ese dominio. Otras formas de verificación pueden incluir una **visita presencial**, por ejemplo, a una comisaría para conseguir un **DNI electrónico**.

### Cadenas de certificados

Las CA pueden **emitir certificados** para otras CA. Por ejemplo, en los certificados de dominio se permite que haya muchas más autoridades de **segundo nivel** que de **primero**. Las de primer nivel emiten certificados de CA; y las de segundo, **certificados de usuario**.

Así se consigue, por ejemplo, que los usuarios tengan más **oferta** y no dependan de un número limitado de CA de **nivel raíz**. La Figura 10 muestra la cadena de certificados del dominio `unir.net` tal como aparece en un **navegador web**.

### CA de nivel raíz

Se han mencionado algunos métodos para verificar la identidad del sujeto antes de **emitir un certificado** (modificación de registros DNS y visita presencial). También se ha hablado de que los certificados pueden formar una cadena hasta la CA de nivel raíz. ¿Cómo se garantiza entonces que el certificado de la CA de nivel raíz es realmente de la CA que dice ser? Los certificados de estas CA están firmados por la propia **clave privada de la CA**, ya que no hay un certificado de una **capa superior** que lo pueda firmar.

En el caso de los certificados de dominio, los navegadores distribuyen los certificados de las CA de nivel raíz más extendidas. En una organización con una CA interna, una posibilidad es distribuir los certificados a través de las directivas de grupo de *Active Directory*.

Varias empresas, como Thawte, VeriSign o Let's Encrypt, se han establecido como **autoridades de certificación raíz**. Estas empresas se encargan de verificar las solicitudes de certificado (con el método de los registros DNS, por ejemplo), procesarlas, emitir los certificados y gestionarlos.

## Gestión de certificados

La tarea de emisión de certificados no es la única que llevan a cabo las **autoridades de certificación**. Los certificados tienen un **ciclo de vida**, y uno de los momentos de este ciclo es la posible **invalidación** de estos. La fecha de expiración invalida un certificado de manera **automática**, de modo que un cliente puede verificar la validez temporal simplemente leyendo los **metadatos del certificado** (que, al estar firmados, se tiene garantía de que son válidos) y teniendo el **reloj local en hora**.

Sin embargo, un certificado puede ser invalidado, antes de su fecha de expiración, por parte de una CA, por varias razones. Por ejemplo, se puede considerar que el **algoritmo** con el que se firmó ya no es lo suficientemente **seguro**, o que el tamaño de la clave es muy **pequeño**. También puede ocurrir que un individuo deje de cumplir los **requisitos** que le permitieron conseguir el certificado y, si este sigue en vigor, le otorgarían permisos que ya no debe tener.

En ambos casos, una CA puede, proactivamente, **invalidar un certificado**. Dado que los certificados son **autocontenidos** y se pueden distribuir libremente, la invalidación no puede perseguir el borrado ni el envío de un **contracertificado** a todos aquellos usuarios que recibieron el mensaje original, por motivos de escala.

El método que siguen las CA es **publicar** listas de certificados que han sido **revocados**. Estas listas se conocen como *Certificate Revocation List* (CRL) o **listas de revocación de certificados**. En el escenario que se ha usado como ejemplo en este capítulo, B tiene una tarea más para **comprobar** que el mensaje viene realmente de A: además de comprobar la firma, el resumen y que el certificado ha sido emitido por una CA de

confianza, también debe consultar la CRL de dicha CA. Si el certificado de A se encuentra en esta lista, B no deberá confiar en el mensaje.

### ***Secure Sockets Layer***

SSL es un **protocolo de aplicación** que actúa como **capa intermedia** entre TCP y otro protocolo de aplicación. El ejemplo más conocido es HTTPS, en el que HTTP se transmite sobre SSL y usa el puerto 443 por defecto, pero se usa con otras aplicaciones conocidas, como **FTP** y **SMTP**.

Ofrece un **canal seguro** que soporta autenticación, integridad y cifrado mediante certificados, firmas y cifrado. El protocolo usa **criptografía asimétrica** para la autenticación y el establecimiento de parámetros de la sesión; y **criptografía simétrica** para el intercambio de los datos.

El protocolo es versátil en cuanto a la variedad de algoritmos que soporta.

Ambos extremos de la comunicación deben ponerse de acuerdo en un conjunto de algoritmos de firma y cifrado durante el establecimiento.

Esto permite que una organización **restrinja el uso de algoritmos** menos seguros, mediante opciones de configuración, sin necesidad de cambiar la capa de software completamente. Esto hace, además, que el protocolo sea **extensible**. Las nuevas versiones soportan algoritmos nuevos, a medida que están disponibles.

Aunque se hable de SSL, el **protocolo soportado**, actualmente, es **TLS**. Hubo varias versiones de SSL, pero TLS incluyó **características nuevas** que impidieron la compatibilidad hacia atrás.

## Establecer una sesión

El establecimiento de un **canal SSL** se lleva a cabo con una negociación en la que cliente y servidor **acuerdan los algoritmos** que van a usar, e **intercambian** los certificados. Este paso es opcional en ambos extremos; por ejemplo, los servidores web envían su certificado durante una petición HTTPS, pero no tienen por qué solicitar un **certificado de cliente**.

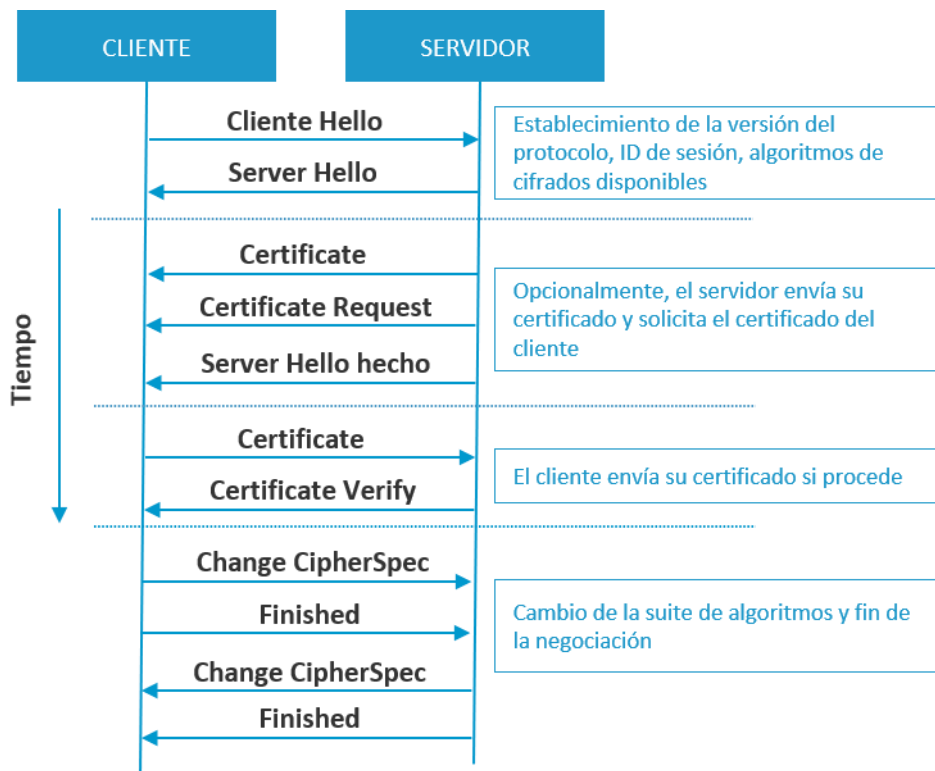


Figura 11. Establecimiento de conexión SSL. Fuente: elaboración propia.

Esta negociación establece una **sesión** que se puede reaprovechar para **intercambiar tráfico adicional**, después del intercambio inicial. Esto reduce la **latencia inicial** para el envío de nuevas peticiones.

La negociación como tal, así como la fase de intercambio de suites, se llevan a cabo con **protocolos específicos**, todos ellos incluidos en el *SSL Record Protocol*, como se muestra en la Figura 13.



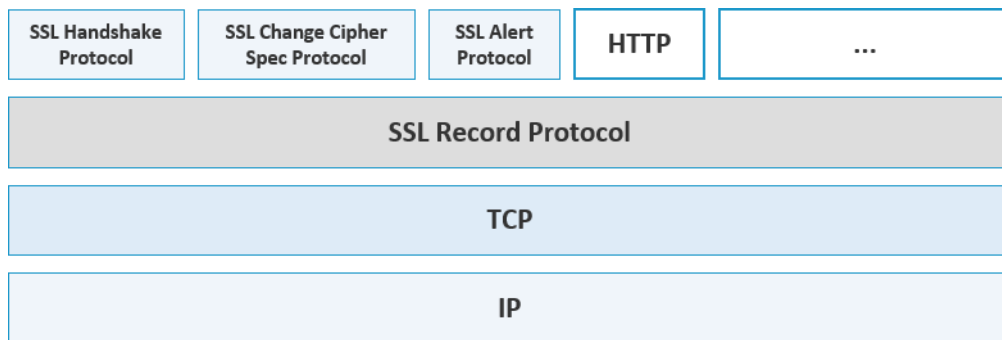


Figura 12. Pila de protocolos SSL. Fuente: elaboración propia.

La negociación incluye los siguientes **pasos**:

- ▶ Negociación del conjunto de algoritmos.
- ▶ Establecimiento de una clave de sesión (esta clave se usará para el cifrado simétrico durante la sesión, y no es la clave privada ni pública de ninguno de los extremos).
- ▶ El cliente valida el certificado del servidor, si procede, y viceversa.

Durante el primer paso, cliente y servidor acuerdan qué algoritmos usarán en función de los que tengan **disponibles**: un cliente puede no disponer de los algoritmos más nuevos, y un servidor puede no permitir el uso de algoritmos menos seguros, como se ha mencionado. Los algoritmos se acuerdan en **suites**, que son **conjuntos de algoritmos** que abarcan el intercambio de claves, el algoritmo de **cifrado de mensajes** y la **función de firma**.

### Método de intercambio de claves

Los datos se intercambian con un **cifrado simétrico** para reducir el **coste computacional del cifrado**. Este tipo de cifrado requiere una **clave compartida** por ambos extremos y, para asegurar la privacidad, esta clave debe intercambiarse de **manera segura**. Algunas **técnicas** de intercambio de esta clave son **RSA** y **Diffie-Hellman**.

## Cifrado para transferencia de datos

Este componente de la *suite* de cifrado se refiere al algoritmo simétrico que se usará durante el **intercambio de datos** (este algoritmo será el que use la clave intercambiada con el método anterior).

Las versiones anteriores a TLS1.3 ofrecían la opción no cifrar el contenido (por lo que ofrecían autenticación e integridad, pero no privacidad). El cifrado puede conseguirse con cifrados de flujo (*stream cipher*) o con cifrados de bloque. TLS 1.2 soporta, entre otros, RC4 como cifrado de flujo y algunas variantes de AES como cifrado de bloque.

## Función de resumen o *hash*

TLS 1.2 admite, entre otras funciones, **MD5** (un *hash* de 128 bits), **SHA-1** (de 160 bits) y variantes de **SHA-2** (de hasta 512 bits). Estas funciones generan un **código de autenticación de mensaje** o *Message Authentication Code* (MAC), a partir del mensaje original. Este código se cifra junto al mensaje como **comprobación de integridad**.

## Transferencia de datos

El *SSL Record Protocol*, que se muestra dentro de la pila de protocolos en la Figura 12, encapsula tanto los **datos de clientes** como la **información de control**. Realiza las funciones mostradas en Figura 13, entre las que se encuentran:

- ▶ Fragmenta estos datos en unidades más pequeñas, si es necesario, y combina múltiples mensajes de datos de protocolo de nivel superior en paquetes individuales.
- ▶ Genera el código de autenticación aplicando la función de firma al mensaje.
- ▶ Cifra cada fragmento y su firma.
- ▶ Delega la transmisión a la capa TCP.

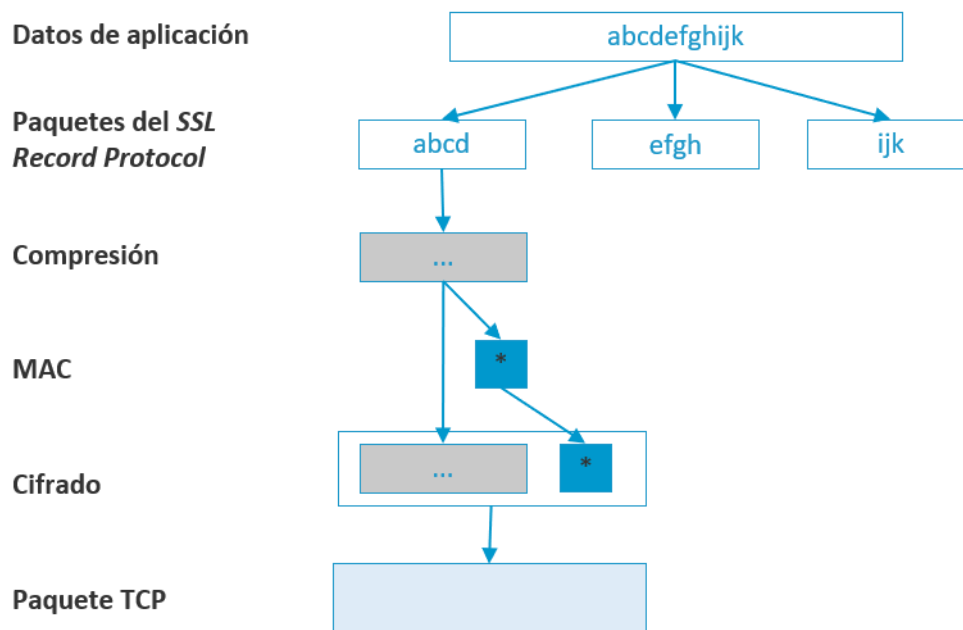


Figura 13. SSL Record Protocol. Fuente: elaboración propia.

## 4.11. Referencias bibliográficas

Apache. (S. f.). *SSL/TLS Strong Encryption: An Introduction*.  
[http://httpd.apache.org/docs/2.2/ssl/ssl\\_intro.html](http://httpd.apache.org/docs/2.2/ssl/ssl_intro.html)

Archip, A., Amarandei, C., Herghelegiu, P., Mironeanu, C., Șerban, E. (2018). *RESTful Web Services – A Question of Standards*. International Conference on System Theory, Control and Computing (ICSTCC), 677 - 682.  
<https://ieeexplore.ieee.org/document/8540763>

Hirsch, F. (1997). Introducing SSL and certificates using SSLeay. *World Wide Web Journal* 2.3, 141-173.

IEFT. (S. f.). *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, RFC 7231*.  
<https://tools.ietf.org/html/rfc7231>

IETF. (S. f.). *Simple Mail Transfer Protocol*, RFC 2821. <https://tools.ietf.org/html/rfc2821>

IETF. (S. f.). *DNS Zone Transfer Protocol (AXFR)*, RFC 5936. <https://tools.ietf.org/html/rfc5936#section-2>

IETF. (S. f.). *Domain Names - Implementation and Specification*, RFC 1035. <https://tools.ietf.org/html/rfc1035>

IETF. (S. f.). *Network Time Protocol (NTP)*, RFC 958. <https://tools.ietf.org/html/rfc958>

IETF. (S. f.). *Telnet Protocol Specification*, RFC 854. <https://tools.ietf.org/html/rfc854>

International Telecommunication Union. (S. f.). *Recommendation X.509*. <https://www.itu.int/rec/T-REC-X.509/e>

MDN Web Docs. (2021, mayo 18). *HTTP response status codes*. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

Oxford English Dictionary. (<https://www.oed.com/>).

Tanenbaum, A. y Wetherall, D. (2011). *Computer Networks*. (5ª ed). Pearson New International.

## ***Oxford Dictionaries***

Oxford Dictionaries. (S. f.). *Oxford Dictionaries API Documentation*.  
<https://developer.oxforddictionaries.com/documentation>

Cada vez más servicios implementan APIs REST como interfaz. La API de *Oxford Dictionaries* es un claro ejemplo de API pensada para el consumo por otros servicios. Su documentación expone todos los detalles del protocolo HTTP necesarios para construir las peticiones. Incluso, incluyen ejemplos de código en varios lenguajes para facilitar la vida de los desarrolladores. No es necesario leer la documentación de principio a fin, pero un vistazo a algunos de los recursos que ofrecen puede servir para aclarar el concepto de API.

## **Httpbin.org**

Httpbin. (<https://httpbin.org/>)

Httpbin es otro ejemplo de API REST, pensada, en este caso, para probar clientes y librerías HTTP. Las peticiones a Httpbin solicitan recursos imaginarios en los que la respuesta incluye los detalles HTTP solicitados por la petición. Por ejemplo, GET <https://httpbin.org/get> devuelve las cabeceras, URL y parámetros en el cuerpo de la respuesta. Es un buen campo de prácticas.

## Criptografía

Ferguson, N., Schneier, B. y Kohno, T. (2010). *Cryptography Engineering*. John Wiley & Sons.

La criptografía no es un tema sencillo y entrar en el detalle técnico de los algoritmos daría para muchas asignaturas de la misma longitud que esta. El prólogo y el primer capítulo de este libro dan una idea de por qué la criptografía es necesaria y por qué es tal difícil. El resto de libro es solo para aquellos que se queden con ganas de más.

1. ¿Por qué se pueden hacer peticiones SMTP o HTTP usando Telnet?
  - A. No es posible.
  - B. Porque ambos son protocolos basados en texto que envían cadenas de texto plano sobre una conexión TCP.
  - C. Porque Telnet es un protocolo genérico que actúa como capa de nivel 4.
  - D. Porque HTTP y SMTP se encapsulan siempre sobre Telnet.
  
2. ¿Por qué FTP ofrece un modo pasivo?
  - A. Porque los puertos del cliente pueden no estar accesibles al servidor debido a *firewalls*, NAT, etc.
  - B. Para hacer el protocolo más completo.
  - C. El modo pasivo está desaconsejado y es preferible usar el modo activo.
  - D. Para la interacción entre dos servidores FTP, no entre cliente y servidor.
  
3. ¿A qué capa ofrece servicios la capa de aplicación?
  - A. A la capa de presentación.
  - B. A la capa de transporte.
  - C. A ninguna.
  - D. Ninguna de las anteriores.
  
4. ¿Por qué DNS funciona sobre UDP?
  - A. Porque el contenido de las peticiones es pequeño y, por tanto, comparable a la sobrecarga que añade TCP.
  - B. Para reducir la latencia inicial que añade TCP.
  - C. Todas las anteriores.
  - D. Ninguna de las anteriores.

5. Relaciona cada protocolo con su transporte capa 4.

HTTP	1	A	TCP
Petición DNS	2	B	UDP

6. ¿Qué diferencia hay entre un registro DNS de tipo A y otro de tipo MX?
- A. Ninguna.
  - B. Un registro A contiene una IP, mientras que un registro MX contiene el nombre de *host* de un servidor de correo.
  - C. Un registro A contiene una IP, mientras que un registro MX contiene un alias de otro nombre de dominio.
  - D. Un registro A contiene una IP, mientras que un registro MX contiene el nombre del servidor DNS.
7. ¿Cuáles de los siguientes son ejemplos de protocolos de aplicación que funcionan en modo texto? Escoge todas las opciones correctas.
- A. NTP.
  - B. HTTP.
  - C. FTP.
  - D. DNS.
8. ¿Cuáles de los siguientes son ejemplos de métodos HTTP? Escoge todas las opciones correctas.
- A. GET.
  - B. POST.
  - C. FETCH.
  - D. PUT.



9. Relaciona el protocolo con el puerto correspondiente.

HTTP	1
FTP	2
Telnet	3
NTP	4

A	123
B	80
C	23
D	21

10. ¿Cuál fue la última versión de SSL?

- A. SSL 2.0.
- B. SSL 3.0.
- C. TLS 1.3.
- D. TLS 1.0.