AGRADECIMIENTOS

Quiero darle las gracias a mi mujer, por su tremenda paciencia mientras las horas pasaban pegado a la pantalla, y su insistencia para que siguiese trabajando cuando mi eterna pereza me hacía frenar.

También quiero agradecer esa misma paciencia en mis editores, que amablemente toleraron mis retrasos sin enfadarse demasiado (léase la referencia a "mi eterna pereza").

Introducción

Los **sistemas electrónicos de información** incluyen tanto la informática como la telemática (telefonía, para los amigos). Aparentemente ambos campos no tienen mucho en común. La deducción más obvia es que los sistemas telemáticos rebosan elementos informáticos, pero lo mismo ocurre con los demás campos técnicos; es raro el sistema en el que no encontramos uno o más ordenadores.

Si nos centramos en los propios equipos informáticos veremos que sus funciones se limitan a **procesar la información** y **transmitirla**, ya sea internamente o a través de una red, y es el proceso de transmisión el que difumina las fronteras entre la informática y la telemática, ya que ambas utilizan los mismos principios para ello. Por eso es tan interesante estudiarlas a la vez.

Respecto a los contenidos, tenemos el problema clásico: ¿qué fue primero: el huevo o la gallina, los "pesados" fundamentos o las partes más jugosas sin establecer una base previa? Vamos a optar por el camino del medio; utilizar el **capítulo 1** para estudiar ciertos conceptos fundamentales, profundizando lo justo, en un intento de mantenerlo ameno.

Los **capítulos 2 a 4** se dedicarán a la microinformática, proporcionando los conocimientos necesarios para utilizar un PC a nivel de usuario, lo que incluye el hardware, las aplicaciones básicas y la instalación, configuración y reparación del sistema.

El **capítulo 5** trata las bases de los sistemas telemáticos, empezando por su elemento más famoso, el teléfono, y continuando por todos los pasos que atraviesa la señal, estudiando los distintos protocolos y servicios utilizados en la transmisión de datos. Esto nos permitirá comprender mejor las redes de área extensa (WAN), base de la telemática.

Por último, el **capítulo 6** estudia el diseño, administración y montaje físico de las redes de área local (LAN), a la que se añade una introducción a la defensa contra intrusos informáticos.

Resumiendo, el objetivo de este libro es proporcionar los conocimientos teóricos necesarios para comprender el entorno informático/telemático en el que vivimos, y los prácticos para modificarlo, ya sea configurando y reparando ordenadores, o instalando sencillas redes telefónicas y de datos.

1 Tratamiento electrónico de la información

Objetivos del capítulo

- ✓ Comprender el proceso de digitalización y los sistemas numéricos que utiliza.
- ✓ Entender los fundamentos de los diversos algoritmos de compresión y codificación.
- ✓ Conocer los medios físicos utilizados en el almacenamiento de información digital.

1.1 INFORMACIÓN DIGITAL

La información gobierna nuestras vidas, ya que dependemos de nuestros sentidos para proporcionarnos un flujo constante de datos que nos mantenga en contacto con el mundo. Transmitimos información mediante el lenguaje, la conservamos con la escritura, la utilizamos a través de las ciencias. También están las artes, que manipulan nuestras emociones en forma difícilmente mensurable.

El advenimiento de los sistemas informáticos y su capacidad para gestionar millones de datos en un parpadeo ponen de manifiesto la importancia de convertir la información a un formato que puedan entender. Y como estos sistemas son simples apisonadoras matemáticas, dicho formato debe ser numérico.

La escritura es fácil de expresar en esta forma, ya que dispone de un juego de caracteres limitado y basta con asignar un valor a cada uno de ellos. Uno de los códigos más conocidos es el **ASCII**, que utiliza 128 valores para identificar las distintas letras (mayúsculas, minúsculas, etc.) y algunos caracteres especiales, como retornos de carro.

Si abrimos un editor de texto, mantenemos pulsada la tecla *ALT*, escribimos en el teclado numérico un valor decimal entre 0 y 127 y soltamos *ALT*, aparecerá el carácter ASCII correspondiente. Por ejemplo, *ALT+100* es la *d* minúscula, mientras que *ALT+68* es la mayúscula. Este método es muy útil para obtener ciertos caracteres que no se encuentran en el teclado español, como ~, accesible a través de ALT+126.

Actividades

- Utilizando *ALT* y el teclado numérico, localizar los equivalentes ASCII de la *Z* (mayúscula) y la *a* (minúscula), y comprobar si son contiguos.

Por lo tanto, la **digitalización de la información** es el proceso de convertirla en valores numéricos, con el grado de exactitud deseado. La frase tiene trampa, ya que por una parte pide exactitud numérica y por otra concede un margen de error. Por ejemplo, la frase "tres patatas" puede almacenarse mediante un código numérico, pero tenemos que entender que representa tres unidades de una definición genérica, la patata, a la que no hemos digitalizado. No hemos expresado su estado (pelada, cocida), peso, estructura molecular, etc.

Puede que el ejemplo anterior parezca una exageración, pero la verdad es que la digitalización rara vez está a la altura del concepto real. Basta con estudiar de cerca el código ASCII: nos permite expresar numéricamente una gran cantidad de caracteres, pero carece de ciertos símbolos especiales necesarios para representar textos en idiomas distintos del inglés; entre ellos, las letras con acento del espa-

ñol. La representación digital del lenguaje escrito es incompleta. Este problema se solucionó aumentando el código hasta 256 caracteres, dejando así sitio para nuevos elementos.

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	◘	BS	040	(	072	H	104	h
009	(tab)	HT	041	)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	'	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♫	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	►	DLE	048	0	080	P	112	p
017	◄	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	!!	DC3	051	3	083	S	115	s
020	π	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↨	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[	123	{
028	(cursor right)	FS	060	<	092	\	124	¦
029	(cursor left)	GS	061	=	093	]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	⌂

Figura 1.1. *Código ASCII.*

Una vez ampliado ASCII los desarrolladores agregaron caracteres a su antojo en la segunda mitad, con la consiguiente confusión, hasta que la especificación **ISO-8859** estableció un estándar para los diversos **juegos de caracteres**. España utiliza ISO-8859-1, también llamado Latín-1; los países eslavos con alfabeto latino recurren a ISO-8859-2 (Latín-2), los que utilizan el alfabeto cirílico a ISO-8859-5, etc. Cada uno de estos elementos dispone de los 128 caracteres del ASCII clásico y una segunda mitad ajustada a las necesidades de cada lenguaje.

¿Hemos conseguido digitalizar los distintos alfabetos? Sí. ¿Hemos conseguido representar la forma escrita exactamente? Definitivamente no. Basta con abrir un documento escrito bajo ISO-8859-1 en un equipo con ISO-8859-5 para que las letras acentuadas se conviertan en garabatos; no podemos mezclar distintos idiomas a menos que todos ellos compartan un juego de caracteres común.

Actualmente las soluciones unificadas son **ISO 10646** y **Unicode**, ambas lo bastante compatibles entre sí para que no debamos preocuparnos de ellas. Basta con tener en cuenta que disponen de capacidad más que suficiente para contener

todos los símbolos utilizados en los diversos idiomas, y que crecen continuamente para aumentar su representación.

Ahora nos enfrentamos a una de las paradojas clásicas de la informática. Disponemos de un método normalizado para representar de forma conjunta todos los lenguajes, pero existe una enorme cantidad de documentos previos a los que debemos poder acceder, manteniendo la **compatibilidad hacia atrás**. Uno de los estándares más populares para implementar Unicode es **UTF-8**, que soporta elementos tan antiguos como ASCII de 128 caracteres, aunque tiene algunos problemas de compatibilidad.

Uno de los ejemplos más famosos es el acceso desde Linux a archivos creados en Windows XP y cuyo nombre contiene acentos; si no advertimos al sistema de archivos para que utilice UTF-8 (formato nativo en XP) las letras acentuadas serán sustituidas por caracteres ilegibles, bloqueando el acceso por parte de cualquier aplicación al archivo hasta que sea renombrado. Esto ya no pasa en las versiones modernas de Linux, que son totalmente compatibles con el estándar UTF-8.

La información digital no se limita a las matemáticas y los documentos de texto. Todos hemos visto imágenes digitales, archivos de audio y películas de vídeo, y más tarde estudiaremos el proceso que se lleva a cabo para realizar la conversión, pero algo debe quedar claro: **la digitalización no alcanza la realidad**, aunque sólo sea porque no podemos disfrutar de la textura y el olor del papel original. Ahora bien, una vez digitalizada la información se "congela" en una estructura numérica, **resistente al paso del tiempo y maleable**. Las fotos digitales no amarillean, y podemos hacer maravillas con un programa de retoque.

1.2 BITS Y BYTES

Al hablar de la digitalización hemos visto la importancia de convertir los distintos datos en valores numéricos. Aunque en teoría podemos construir un sistema electrónico que recurra a diez niveles de tensión distintos para indicar los valores del 0 al 9, cualquier pequeña variación representaría una alteración de los datos, causando errores. Por eso los sistemas digitales establecen un único valor **umbral** a partir del cual se considera que existe señal, por ejemplo 5 voltios. De esta forma las pequeñas alteraciones no afectarán al resultado pero nos vemos limitados a sólo dos opciones, que podemos llamar **activo** o **inactivo**, **sí** o **no**, **1** ó **0**… lo que se denomina unidad mínima de información, o **bit**.

Sabemos que podemos utilizar el sistema **decimal** para expresar cualquier cifra a pesar de tener sólo diez dígitos, pero estamos tan acostumbrados al proceso que es difícil darnos cuenta de cómo funciona. La cuenta de 0 a 9 es fácil, pero, ¿qué ocurre después?

Actividades

- Antes de seguir leyendo, explicar en uno o dos párrafos cómo se calcula el valor siguiente a 20199.

Es fácil calcular el valor correcto para la pregunta anterior (20200), pero ya no tanto explicar cómo hemos llegado a él. Cuando contamos valores consecutivos en realidad añadimos 1 a la primera cifra, por ejemplo, 233 + 1 = 234. Si dicha cifra es la última posible (9), la ponemos a cero y sumamos uno a la siguiente: 239 + 1 = 240. Hay un número infinito de ceros a la izquierda aunque no se escriban, por lo que podemos seguir incrementando indefinidamente: 99 + 1 = 099 + 1 = 100.

El proceso es muy sencillo y no sólo funciona con el sistema decimal, sino con cualquier otro. Por ejemplo, si sólo tenemos tres cifras con las que trabajar, 0, 1 y 2, la cuenta se realizaría así: 0, 1, 2, 10 (puesta a 0 de la primera cifra y sumamos 1 a la siguiente), 11, 12, 20 (puesta a cero y sumamos 1), 21, 22, 100 (puesta a cero de la primera cifra, sumamos 1 a la siguiente, que también se pone a cero, sumamos uno a la siguiente).

Existen sistemas que utilizan más de 10 dígitos, como el **hexadecimal**, que recurre a las letras de la "A" a la "F" para expresar los seis valores por encima de 9. Después de contar de 0 a 9 viene A, B, C, D, E, F, y vuelta a empezar sumando uno (10, 11…, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20…). Si sólo disponemos de los dígitos 0 y 1 se trata de un sistema **binario** y el proceso es el mismo, aunque mucho más brusco, ya que la primera cifra llega al final cada dos valores, obteniendo una cuenta del tipo 0, 1, 10, 11, 100, 101, 110, 111, 1000, etc.

Para evitar confusiones entre cifras decimales y hexadecimales suele agregarse a estas últimas una hache minúscula. Por ejemplo 10 puede ser un valor decimal, o el valor hexadecimal equivalente al 16 decimal, pero **10h** es claramente un valor hexadecimal. Lo mismo ocurre con el binario, agregando una b minúscula en este caso, aunque se utiliza menos, porque habitualmente se trata de cifras de gran tamaño compuestas únicamente de ceros y unos, y las posibilidades de error son reducidas.

Considerando que la digitalización nos permite expresar información en forma numérica, y que los sistemas electrónicos digitales pueden manipular la información si viene dada en forma de bits, es decir, dividida en elementos cuyo valor es 0 ó 1, resulta evidente que basta con convertir la información decimal a binaria para poder utilizarlos en un sistema electrónico digital.

¡Que no cunda el pánico! No vamos a sumergirnos en operaciones matemáticas o aritméticas entre distintas bases; no es que no sea interesante, pero al final su aplicación práctica es limitada. En su lugar vamos a estudiar un par conceptos y métodos sencillos que nos permitirán comprender mejor el funcionamiento de los sistemas informáticos.

1.2.1 CAMBIO DE BASE A DECIMAL

El número de dígitos distintos del que dispone un sistema numérico se llama base. El decimal es **base 10**, el binario **base 2** y el hexadecimal **base 16**. El proceso de convertir un número de un sistema a otro se llama **cambio de base**, y aunque puede llegar a ser muy complejo, el paso a decimal es terriblemente simple, si entendemos una sola cosa: por qué 1.397 es mil trescientos noventa y siete.

Empezando por la derecha, la primera cifra representa las **unidades**. La segunda son las **decenas**, e indican 10 veces su valor; el 9 de 1.397 vale 90, debido a su posición como decena. La tercera son las **centenas**, multiplicadas por 100 (300 en 1.393), la siguiente los **millares**, etc. Dicho de otra forma, las unidades se multiplican por uno (en forma exponencial 10^0), las decenas por diez (10^1), las centenas por cien (10x10, o 10^2), los millares por mil (10x10x10, o 10^3), etc.

De aquí se extrapola la regla para convertir un número en cualquier base a decimal; simplemente tendremos que multiplicar el valor de la primera cifra por 1, la segunda por la base, la tercera por la base al cuadrado, la cuarta por la base al cubo, etc. Por ejemplo, las "unidades" del **octal** (base 8) valen 1, las "decenas" 8, las "centenas" 64, los "millares" 512, etc. El número octal 1072o se convierte a decimal multiplicando cada cifra por su valor y sumando el total:

1072o = (1 x 512) + (0 x 64) + (7 x 8) + (2 x 1) = 570

En binario la base es 2, y los valores de cada cifra, empezando de derecha a izquierda, son: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, etc. En hexadecimal la base es 16 y los valores son: 1, 16, 256 (16x16), 4.096 (16x16x16), etc. El único problema en este caso es que podemos encontrarnos con letras, debiendo convertirlas antes a decimal; por ejemplo, para obtener el equivalente decimal de 2A5h hay que saber que Ah es 10, y entonces no hay problema en calcular el resto:

2A5h = (2 x 256) + (10 x 16) + (5 x 1) = 677

Actividades

- Convertir los siguientes números al formato decimal: 630o, 11h, 1001010b, 2Ch.

1.2.2 CAMBIO DE DECIMAL A UNA BASE DADA

Hay reglas para realizar operaciones matemáticas en una base, por ejemplo tablas de multiplicación hexadecimal, pero no resulta demasiado interesante aprenderlas, porque se olvidan con demasiada facilidad si no las utilizamos. Es más sencillo convertir los operandos al sistema decimal y después devolver el resultado a la base original. Lo mismo ocurre con los cambios de base entre dos sistemas no decimales, es posible hacerlo directamente, pero resulta más fácil utilizar el decimal como elemento intermedio (base X a decimal, decimal a base Y).

El procedimiento estándar para pasar de decimal a una base dada son las divisiones sucesivas. Por ejemplo, para convertir el decimal 2.754 a octal dividiremos este valor por 8, es decir, por la base de destino, y el resto será la primera cifra del número en la nueva base empezando por la derecha:

2.754 / 8 = cociente 344, resto 2
Cálculo parcial del equivalente octal: …2o

Ahora dividimos de nuevo el cociente por la base, y el resto será la segunda cifra:

344 / 8 = cociente 43, resto 0
Cálculo del equivalente octal: …02o

Repetimos el proceso hasta que el cociente es más pequeño que la base, anotamos el resto y después el propio cociente como la última cifra.

43 / 8 = cociente 5, resto 3
Equivalente octal de 2.754: 5302o

Recapitulando: dividimos la cifra por la base y anotamos el resto como la primera cifra, las unidades. Dividimos el cociente obtenido por la base y anotamos el nuevo resto como la segunda cifra, repitiendo hasta que dicho cociente sea menor que la base, momento en el que lo anotaremos como la última cifra, después del resto.

Las bases superiores a 10 funcionan igual, siempre y cuando nos demos cuenta de convertir los restos superiores a 9 a su equivalente alfabético. Por ejemplo, veamos el paso del decimal 420 a hexadecimal:

420 / 16 = cociente 26, resto 4
Cálculo parcial del equivalente hexadecimal: …4h
26 / 16 = cociente 1, resto 10 (A)
Equivalente hexadecimal de 398: 1A4h

Actividades

- Convertir a hexadecimal las siguientes cifras: 198, 1.000,34.

Para las conversiones a binario podemos aplicar el mismo método, pero en cuanto la cifra es un poco grande las divisiones se disparan, lo que resulta tedioso. Por ejemplo, pasar 1.058 de decimal a binario requeriría las siguientes divisiones:

1.058 / 2 = cociente 529, resto 0
529 / 2 = cociente 264, resto 1
264 / 2 = cociente 132, resto 0
132 / 2 = cociente 66, resto 0
66 / 2 = cociente 33, resto 0
33 / 2 = cociente 16, resto 1
16 / 2 = cociente 8, resto 0
8 / 2 = cociente 4, resto 0
4 / 2 = cociente 2, resto 0
2 / 2 = cociente 1, resto 0
Equivalente binario de 1.058: 10000100010

Es fácil comprobar que el resultado es correcto realizando otra vez la conversión a decimal, sumando los valores de las posiciones donde hay un uno: segunda ($2^1 = 2$), sexta ($2^5 = 32$) y undécima ($2^{10} = 1.024$); 2 +32 + 1.024 = 1.058.

Si tenemos que convertir a menudo valores decimales a binarios podemos recordar las primeras potencias de dos, buscando a continuación la mayor potencia que esté por debajo de la cifra a convertir, restándola de ella, y repitiendo el proceso con el resto hasta llegar a cero. Por ejemplo:

1.058 – 1024 = 34
36 – 32 = 2
2 – 2 = 0

Con esto hemos conseguido obtener las potencias que componen el decimal 1.058. Ahora sólo tendremos que escribir unos en las posiciones obtenidas y ceros en todas las demás:

***1**(1.024) **0**(512) **0**(256) **0**(128) **0**(64) **1**(32) **0**(16) **0**(8) **0**(4) **1** (2) **0**(1)*

Admito que parece complicado, pero es muy rápido, aunque ante la duda siempre podemos recurrir a las divisiones sucesivas.

Actividades

- Convertir a binario las siguientes cifras: 198, 1.000,34.

Sea como sea, el cambio de decimal a binario no resulta demasiado cómodo, sobre todo cuando utilizamos cifras grandes. A veces se opta en su lugar por utilizar sistemas de representación abreviados, como el **Binario Codificado a Decimal**, o **BCD**, muy popular en circuitos de lógica digital.

En BCD cada dígito decimal se sustituye por su equivalente binario en paquetes de 4 bits. Por ejemplo 209 en formato BCD sería la combinación de 0010 (2), 0000 (0) y 1001 (9): 00100001001. Hay que tener mucho cuidado con las confusiones, ya que este código no es compatible en absoluto con el binario convencional; si hacemos un paso binario-decimal el resultado sería 265.

Es importante conocer los diversos métodos de conversión manual, pero hay una forma mucho más sencilla: las calculadoras científicas. Windows nos suministra una, a la que podemos acceder desde *Accesorios* en el *menú Inicio*, o escribiendo **calc** en *Inicio-Ejecutar*. Para pasar a modo científico pulsaremos *Ver-Científica* en la barra de menús.

Figura 1.2. *Calculadora científica.*

Esta calculadora puede realizar operaciones en hexadecimal, octal, binario y, por supuesto, decimal. Para realizar un cambio de base bastará con seleccionar la base origen en los botones de la izquierda (Hex, Oct, Bin, Dec), escribir la cifra y pulsar en el botón de la base a la que queremos convertirla.

1.2.3 UNIDAD BÁSICA DE INFORMACIÓN. BYTE

Cuando hablamos del código ASCII mencionamos que soportaba 128 caracteres distintos. Si asignamos un valor de 0 a 127 para cada uno de ellos veremos que sus equivalentes binarios van desde 0000000 hasta 1111111, es decir, que necesitaremos 7 bits para expresarlos. Análogamente, una vez ampliado el código a 256 caracteres necesitaremos 8 bits para representar los valores de 0 a 255.

Estos paquetes de 8 bits se llaman **bytes** y tienen la ventaja de ser totalmente compatibles con la estructura interna del ordenador, que originalmente transmitían la información de 8 en 8 bits (byte), después de 16 en 16 (doble byte, o palabra), 32 en 32 (doble palabra) y últimamente de 64 en 64 (palabra cuádruple).

Se considera de forma aproximada que cada carácter de un documento equivale a un byte, y aunque en la realidad eso depende mucho del estándar usado y la compresión del mismo, se cumple de forma exacta con los documentos de **Notepad** de Windows. Después de crear un archivo mediante *botón derecho-Nuevo documento de texto* sobre un espacio libre de la pantalla y escribir en él seis caracteres, podemos pulsar *botón derecho-Propiedades* sobre él para ver que "pesa" seis bytes.

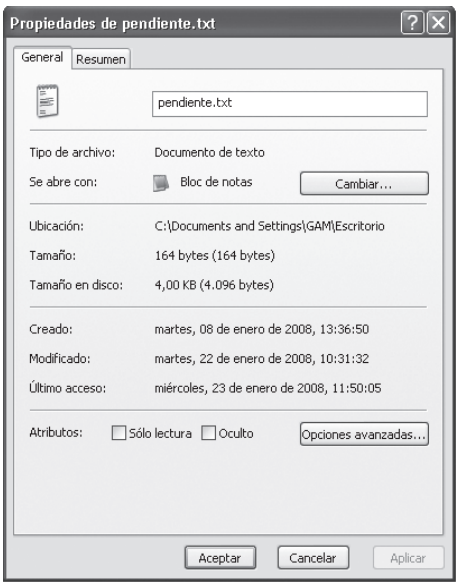

Figura 1.3. *Tamaño de un documento.*

Como hemos visto, 7 bits tenían 128 combinaciones posibles. Al agregar un nuevo bit se doblan las posibilidades, llegando a 256, el valor asociado con los 8 bits del byte. En el sistema decimal, con un dígito alcanzamos 10 combinaciones (0 a 9), con dos 100, y con tres 1.000. Si continuamos agregando bits al byte, con 9 conseguimos 512 combinaciones y con 10 alcanzamos las 1.024, un valor muy cercano al 1.000 decimal. Por eso el millar binario se compone de 10 bits y tiene

un valor real de 1.024, otro de los números famosos de la informática. Medio millar será 512 en lugar de 500, un cuarto 256 en lugar de 250, etc.

La analogía se mantiene; si mil metros son un kilómetro decimal, 1.024 bytes son un Kilobyte (KB) binario. Si un millón decimal son 1.000 millares, un Megabyte (millón de bytes) son 1.024 Kilobytes. La tabla 1.1 muestra los principales múltiplos y sus abreviaturas.

Tabla 1.1. Principales múltiplos binarios

Direccionamiento	Múltiplo	Abreviatura	Valor
Byte	B	8 bits	X
Kilobyte	KB	1.024 bytes	10 bits
Megabyte	MB	1.024 KB	20 bits
Gigabyte	GB	1.024 MB	30 bits
Terabyte	TB	1.024 GB	40 bits
Petabyte	PB	1.024 TB	50 bits

La tabla 1.1 incluye una columna llamada "Direccionamiento". Se trata de una referencia a que un equipo que maneja una cierta cantidad de bytes necesitará identificarlos a todos independientemente, un concepto en el que profundizaremos más adelante. Como vimos antes, 10 bits tienen 1.024 combinaciones posibles, y por eso son necesarios para identificar 1.024 bytes distintos, un KB. Últimamente se habla mucho de **PCs de 32 y 64 bits**; aunque las principales diferencias de rendimiento entre ellos vienen determinadas por ciertos cambios en su arquitectura interna, sí que es cierto que los equipos de 32 bits sólo pueden direccionar 4GB de memoria, mientras que los de 64 no tienen ese límite (30 bits para un GB, según la tabla. Uno más dobla a 2GB y otro más dobla de nuevo, alcanzando los 4GB).

Normalmente podemos considerar los múltiplos de 1.000 para realizar cálculos aproximados sobre cifras pequeñas, pero siendo conscientes de que el valor real corresponde a incrementos de 1.024. Por ejemplo, los fabricantes utilizan el sistema decimal como estándar a la hora de determinar el **tamaño de un disco duro**; así, un disco de 200GB tiene capacidad para 200.000.000.000 bytes. En cambio, el PC utiliza el sistema binario, dividiendo esa cifra por 1.024 para obtener KB, otra vez por 1024 para MB, y otra más para los GB. El resultado es que la gente compra un disco de 200 y el PC le dice que sólo tiene 186GB, con el consiguiente desconcierto sobre los 14GB "desaparecidos".

Actividades

- Calcular el tamaño real de un disco duro de 700 Gigabytes.

1.2.4 VELOCIDADES DE TRANSMISIÓN

En sistemas de transmisión serie, como Internet y las redes de área local, la cantidad de información que puede transmitirse en la unidad de tiempo se denomina **velocidad de transferencia** o **ancho de banda** y suele medirse en bits por segundo bajo el sistema decimal. Una conexión estándar con una capacidad de 1 Megabit por segundo (1 Mb/s) podrá transmitir 1.000.000 bits por segundo, no 1.024 x 1.024 bits por segundo.

En teoría debemos utilizar b y B para distinguir entre bits y bytes en las abreviaturas (por ejemplo, Kb y KB), pero no es un estándar oficial, así que deberemos tener cuidado, aunque por lo general los valores y entorno nos permiten deducir fácilmente el valor correcto.

Para calcular la velocidad de transmisión "real" de una conexión empezaremos dividiendo por 8, para pasar de bits a bytes (un byte = 8 bits), y si la cifra resultante es muy grande la dividiremos por 1.024 para pasarla a KB, otra vez para pasar a MB, etc. En el caso anterior, 1 Megabit por segundo = 1.000.000 bits/s = 125.000 bytes/s; aproximadamente 122 KB/s.

Para calcular el **tiempo** que se tarda en transferir un archivo de 100MB a través de esa conexión, multiplicaremos su tamaño (100MB) por 1.024 para convertirlo a KB (102.400KB), el mismo múltiplo de la velocidad "real" que calculamos (122KB/s) y poder así dividirlo por ella, lo que nos da un tiempo aproximado de 840 segundos. Podemos dividir este valor por 60, descubriendo así que, con mucha suerte, tardaremos unos 14 minutos en realizar la transferencia.

Si esto nos resulta complicado podemos convertir el valor en bytes a bits, multiplicándolo por 8, y eliminar los multiplicadores de ambos elementos (el archivo a transmitir y la velocidad de transmisión) como si los dos fueran decimales. Aunque el resultado no es exacto, si será lo bastante aproximado. Volviendo al caso anterior, multiplicando el archivo de 100MB por 8 para pasarlo a bits y dividiendo obtendremos: 800.000.000 bits / 1.000.000 bits/s = 800 segundos, menos de un minuto de diferencia respecto al cálculo anterior.

Actividades

- Calcular de forma aproximada el tiempo que tardaremos en transmitir un CD de 100MB a través de una conexión de 2Megabits por segundo.
- Repetir el cálculo para obtener la diferencia de tiempo entre el valor real y el aproximado.

1.2.5 CONVERSIÓN BINARIA DIRECTA

Es posible que los ordenadores utilicen nativamente valores binarios, pero las personas no estamos diseñadas para manejar bytes. Es muy difícil recordar valores como 11101001, 11001001 y 11100101 sin confundirnos, pero sus equivalentes decimales (233, 201 y 229) son mucho más asequibles. Este problema se agrava, y mucho, cuando tenemos que trabajar con paquetes de 16, 32 y 64 bits.

Convertir continuamente entre binario y decimal sólo porque las cifras son difíciles de leer resulta incómodo. Una de las grandes ventajas de los sistemas octal y hexadecimal es que podemos convertirlos directamente a binario, y las cifras en estos sistemas son mucho más legibles.

Para preparar una cifra binaria para su paso a octal la dividiremos en grupos de tres dígitos **empezando por la derecha**, rellenando si nos resulta más cómodo a la izquierda con ceros. A continuación bastará con escribir el equivalente octal de cada uno de los grupos, sumando el valor de los bits que tienen un 1, o usar la tabla 1.2 para realizar la conversión.

Tabla 1.2. Binario - Octal

Binario	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Por ejemplo, 10000011101001 quedaría como 010 (2)- 000 (0)- 011 (3)- 101 (5)- 001 (1), 20351o mientras que 100000011101001 se divide en 100 (4)- 000 (0)- 011 (3)- 101 (5)- 001 (1), 40351o. La diferencia entre ambos valores octales es evidente, y el cálculo muy simple.

Actividades

- Calcular el equivalente octal de la siguiente cifra 1100100100000010. Comprobar si es posible con la calculadora que ambas cifras equivalen al mismo valor decimal.

El paso de octal a binario es tan sencillo como sustituir cada dígito por su equivalente binario, cuidando siempre que el resultado sea de tres cifras, rellenando con ceros a la izquierda si es necesario. Por ejemplo, el paso a binario de 724o se obtiene con 111 (7), 010 (2) y 100 (4): 1111010100.

Actividades

- Calcular el equivalente binario de la siguiente cifra 75301o. Comprobar con la calculadora.

El mayor problema del octal es que aunque podemos añadir un cero a la izquierda a un byte solitario para convertirlo en un grupo de 9, cuando trabajamos con una "palabra" ese bit deberá tomarse del segundo byte, rompiéndolo. Por eso el octal se utiliza muy poco, recurriéndose en su lugar al hexadecimal, que funciona de la misma forma pero dividiendo los valores binarios en grupos de 4 bits, tal y como se ve en la tabla 1.3.

Tabla 1.3. Binario - Hexadecimal

Binario	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Al recurrir al hexadecimal un byte se representa con dos dígitos hexadecimales, una palabra por 4 y una doble palabra por 8. Por ejemplo, el byte 01101101 se separa en 0110 (6) y 1101 (13, D en hexadecimal), convirtiéndose en 6Dh. Inversamente, para convertir una cifra hexadecimal en binaria bastará con escribir el equivalente de cada dígito en paquetes de cuatro bits; 2Ah será 00101010.

Actividades

- Convertir la palabra 0010000011101001 a hexadecimal, y el valor FF3Eh a binario. Comprobar con la calculadora.

Resumiendo, el estudio del cambio de base, y en particular del binario, nos permite:

- ✓ Comprender por qué valores como 256, 512 ó 1024 son recurrentes en la informática.
- ✓ Ser capaz de convertir, en caso necesario, valores binarios.
- ✓ Conocer la diferencia entre los sistemas que utilizan múltiplos basados en el sistema binario y el decimal, y convertir de uno a otro, en particular cuando calculamos anchos de banda y el tamaño real de un disco duro.
- ✓ Reconocer y utilizar con soltura valores hexadecimales.

1.3 MUESTREO

La codificación del alfabeto es sencilla, ya que basta con establecer un vínculo entre las letras y distintos valores numéricos; lo mismo ocurre con factores fácilmente cuantificables, como pesos y distancias. El problema viene con ciertos procesos **continuos**, como las fotografías convencionales, que se crean a partir de la exposición a la luz y consiguiente cambio de color de las moléculas de un compuesto químico, siendo imposible identificar y asignar un valor numérico a cada una de ellas.

La solución en estos casos es el **muestreo**, que divide en fragmentos (muestras) el elemento continuo y asigna un valor a cada uno de ellos; la equivalencia nunca será exacta, pero cuantas más muestras tomemos más nos acercaremos al original.

1.3.1 MUESTREO DE UNA IMAGEN

La digitalización de una imagen es como poner una rejilla sobre una fotografía, asignando un valor numérico al color, o a la media de colores, que aparece en cada cuadro. A la hora de guardar estos valores tendremos que anotar también su posición, es decir, la fila y columna a la que pertenecen. Existen tres factores que determinan la calidad y tipo de una imagen muestreada: resolución, profundidad de color y formato.

La **resolución** es el número de filas y columnas en que dividimos la imagen. Cuantas más sean más pequeños serán los cuadros de la cuadrícula, llamados **puntos**, y más aproximado el resultado.

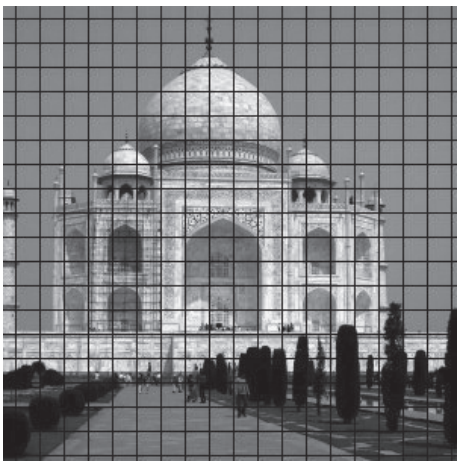

Figura 1.4. *Rejilla de muestreo.*

A partir de 300 puntos por pulgada (unos 118 puntos por centímetro) el ojo humano no puede distinguir los puntos independientes, y por ello se considera la resolución óptima. Por ejemplo, una fotografía de 20x15 tendría una **resolución ideal** de unos 2352x1764 puntos.

Los principales dispositivos de muestreo de imagen son las cámaras digitales fotográficas, de vídeo y escáneres. Estos últimos suelen ofrecer resoluciones muy superiores a los 300 ppp (puntos por pulgada), innecesarias desde un punto de vista convencional, pero muy útiles para ampliaciones o retoques.

La **profundidad de color** es el rango máximo de tonalidades que podemos definir, normalmente determinada por la asignación de un número fijo de bits a cada punto, llamado píxel. Dicho de otra forma, una imagen que asigna un byte a cada punto (píxel) tiene una profundidad de 256 colores, ya que 1 byte = 8 bits = 2^8 combinaciones = 256 combinaciones.

Tabla 1.4. Profundidades de color típicas

Bits	Bytes	Nº de colores	Nombre
1	0	2	Monocromo
8	1	256	256 colores
16	2	65.536	64K colores
24	3	16.777.216	Millones de colores, color verdadero
32	4	4.294.967.296	Color verdadero con canal alfa

Un caso particular son las imágenes en **escala de grises**, que equivalen a la fotografía química en blanco y negro y pueden utilizar el mismo número de bits por píxel que las convencionales, es decir, hasta 32, pero en lugar de contener tonalidades de color representan niveles de gris. No hay que confundir la escala de gris con el **monocromo**, ya que en este último caso los puntos obtenidos son siempre blancos o negros, lo que hace a este sistema ideal para representaciones precisas, como planos, logos, texto, etc.

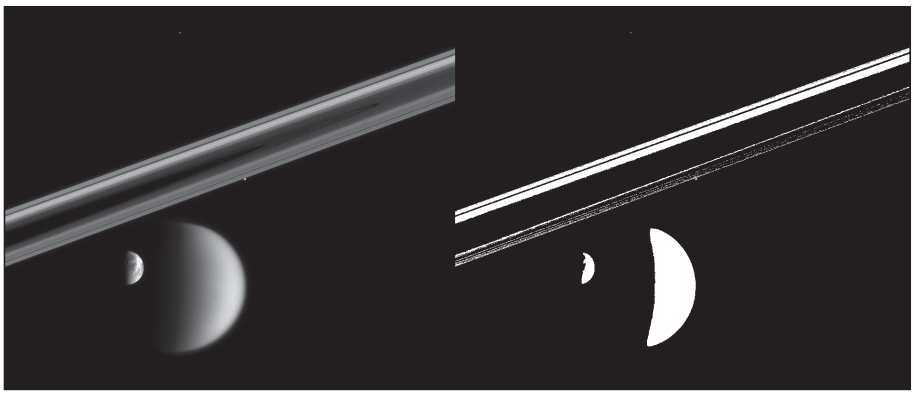

Figura 1.5. *Monocromo y escala de grises.*

La profundidad de 24 bits se denomina **color verdadero** porque el ojo humano es incapaz de distinguir variaciones tonales por encima de este valor, tal y como ocurría con resoluciones superiores a los 300 ppp. Aunque en teoría la profundidad de 16 bits no es idónea, la verdad es que es bastante difícil apreciar la diferencia.

En **sistemas luminiscentes**, como monitores y televisiones, el color se obtiene mediante la combinación de los tres primarios (rojo, verde y azul) con distintas intensidades. Con la profundidad de 24 bits lo más fácil es asignar cada uno de los tres bytes a cada canal **RGB** (**R**ed/rojo – **G**reen/verde – **B**lue/azul), con lo que se obtienen 256 posibles valores de rojo, 256 de verde y 256 de azul.

La profundidad de 32 bits mantiene los tres bytes RGB y añade un cuarto que suele utilizarse para el **canal alfa**, que define qué puntos de la imagen pueden considerarse transparentes, y el grado de transparencia.

Si los monitores crean imágenes mediante la combinación de canales de luz, los sistemas de impresión trabajan agregando **tintas**, que absorben la luz, oscureciendo el medio sobre el que se aplican; por eso los colores básicos son muy claros: Cyan, Magenta y Amarillo. La mezcla de luces RGB en partes iguales da blanco, y la de tintas CMA debería dar negro (absorción total), pero en la práctica se obtiene una especie de marrón sucio, por lo que es necesario agregar tinta negra al conjunto, obteniendo el sistema **CMYK** (**C**yan – **M**agenta –**Y**ellow – **B**lack).

Otro problema de los dispositivos de impresión es que la tinta tiene un color fijo, mientras que cada canal RGB dispone de 256 valores distintos, por lo que para obtener una tonalidad específica será necesario combinar varias gotas en un punto dado. Por eso las impresoras ofrecen "resoluciones" muy superiores a los 300 ppp, en referencia a las gotas que aplican, no a los puntos reales obtenidos. La cantidad de tinta que podemos aplicar depende en gran medida de su densidad y la absorción del papel; es raro que una impresora doméstica supere los 200 ppp reales, incluso utilizando papel fotográfico.

Por último, el **formato** es el método utilizado para almacenar la información obtenida en un archivo digital, y que aunque no tiene relación directa con el muestreo puede afectar al resultado final. Veamos algunos ejemplos:

- **BMP:** también llamado bitmap o mapa de bits, ya que se limita a contener los puntos y la información de color (no acepta el canal de transparencia). Su tamaño es una función directa de su resolución y su profundidad de color; por ejemplo, una resolución de 1.024 x 768 representa un total de 786.432 píxeles. Con una profundidad de color de 16 bits (2 bytes) obtendríamos un bitmap con un tamaño total de 2 bytes/píxel x 786.432 píxeles= 1.572.864 bytes; algo más de lo que entra en un disquete.
- **TGA:** Targa es similar al BMP pero con soporte para información avanzada como capas y transparencia. También dispone de un cierto nivel de compresión sin pérdida de calidad, es decir, que reduce un poco el tamaño final del archivo sin alterar su visualización.
- **JPG:** también llamado JPEG, tiene las mismas características que el BMP pero está "limitado" a una profundidad de color de 16 bits y utiliza un sistema de **compresión con pérdida** que permite reducir mucho el tamaño final del archivo, al precio de eliminar detalles en la fotografía, de forma que cuanto mayor sea la reducción elegida más dañaremos el resultado final. Suele ser posible utilizar valores superiores al 50% sin que el daño sea fácilmente apreciable, excepto por una pérdida de nitidez en las fronteras entre distintos colores y tonos. JPEG es un formato muy popular en Internet y en las cámaras fotográficas digitales, debido a su buena relación tamaño/calidad.

- **GIF:** este formato dispone de compresión sin pérdida y permite agregar información de transparencia simple, pero está limitado a un máximo de 256 colores que se almacenan en una **paleta**. Distintas imágenes tendrán paletas distintas, según las tonalidades predominantes en el original. Actualmente se utiliza básicamente para publicar en Internet imágenes que necesitan transparencia o que tienen muy pocos colores, como los logotipos, ya que en estos casos es posible obtener relaciones tamaño/calidad superiores a JPEG.
- **Otros formatos:** hay cientos de formatos en el mercado; por ejemplo, mejoras como PING (PNG), una evolución de GIF que mantiene su rendimiento pero supera el límite de los 256 colores. También hay muchos formatos propietarios asociados a programas comerciales, como PSD de Photoshop.

Actividades

- Calcular el tamaño en MB de un BMP con una resolución de 1280 x 1024 píxeles y 32 bits de color.

Hasta ahora nos hemos limitado a las **imágenes matriciales**, que como vimos, almacenan la información en forma de rejilla de puntos, pero también existen las **imágenes vectoriales**, basadas en el dibujo geométrico, que guardan las coordenadas de puntos críticos como extremos, vértices, centros o tangencias, y las líneas que los unen.

Representar una fotografía de forma vectorial es poco práctico, pero cualquier dibujo geométrico ocupa muy poco espacio de esta forma, y puede ser modificado con gran facilidad; en particular, las imágenes vectoriales pueden **ampliarse** de forma indefinida sin que aparezcan bloques, ya que lo único que cambia es la relación entre los puntos.

Muchos programas de edición gráfica que manejan ambos tipos de gráficos disponen de opciones para convertir imágenes vectoriales en matriciales de un tamaño dado (**rasterizado**), o matriciales en vectoriales (**trazado**), aunque en este último caso los resultados dependen en gran medida de la capacidad del programa y la habilidad del usuario.

Figura 1.6. *Trazado de una imagen matricial.*

1.3.2 MUESTREO DE UNA SEÑAL DE AUDIO

Una señal es un valor que varía en función del tiempo. En el caso de las señales de audio la velocidad de variación determina la frecuencia (tono), mientras que la amplitud de la misma determina el volumen.

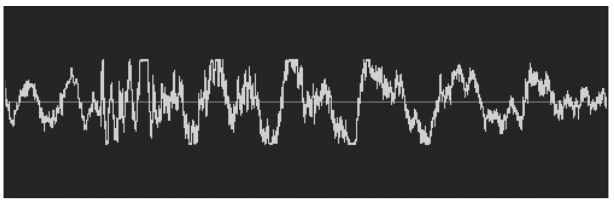

Figura 1.7. *Señal de audio.*

Para muestrear una señal deberemos medir su valor cada cierto tiempo. Se denomina **frecuencia de muestreo** al número de mediciones que realizamos por segundo, y debería doblar a la frecuencia máxima que contiene la señal; en el

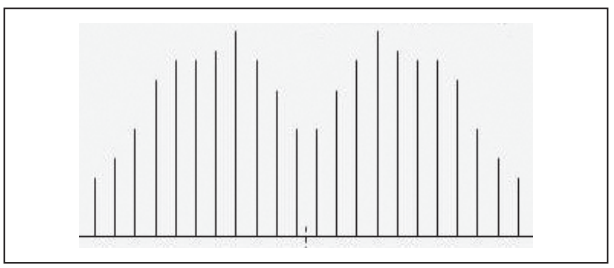

Figura 1.8. *Rejillas de muestreo.*

caso de las señales de audio el oído humano tiene un límite de unos 22KHz, por lo que la frecuencia de muestreo debería ser de al menos 44KHz (44.000 mediciones por segundo) para obtener un resultado preciso.

Digitalmente nos vemos limitados a la hora de anotar el valor muestreado por el número de bits que queramos asignar a cada medición, teniendo en cuenta que afectará a la calidad resultante y al tamaño final del archivo. Para reproducir una señal de audio con **calidad de CD** necesitaremos una resolución de 16 bits para medir la amplitud, lo que implica 2 bytes por cada medición realizada. Como también necesitamos una frecuencia de 44KHz, obtendremos un total de 88.000 bytes por segundo, algo más de 300MB en una hora.

Esto no quiere decir que estos valores sean obligatorios. El teléfono no se preocupa demasiado de la calidad final, sólo de que la voz sea inteligible, y por eso utiliza una frecuencia de muestreo de 8KHz y 8 bits.

El ser humano utiliza las diferencias de percepción entre los dos oídos para ubicar espacialmente un sonido, y por eso los equipos **estéreo** reproducen dos canales distintos. Dos canales representan doble tamaño, así que si antes teníamos unos 300MB por hora en calidad CD, ahora tendremos unos 600MB por hora.

Actividades

- Calcular el tamaño en MB de una hora y 15 minutos (1,25 horas) de audio estéreo en calidad CD.

Rara vez se utiliza la resolución en bits para identificar una señal muestreada, sino que en su lugar se recurre a su producto por la frecuencia de muestreo, obteniendo así el número de bits por segundo, o **bitrate**. La calidad telefónica tiene un bitrate de 8KHz x 8 bits = 64 Kbps, y el audio estéreo con calidad de CD 1.408 Kbps (44KHz x 16 bits x 2).

Al igual que ocurría con las imágenes digitales, la forma en la que almacenamos los datos, el formato, afecta al resultado final. Algunos ejemplos son:

- **Sin compresión:** simplemente contienen en su interior los valores muestreados, sin compresión. Los CDs de audio utilizan el estándar **CDA**, y los sistemas digitales suelen recurrir a **PCM** o a **WAV** de Microsoft.
- **Codificación perceptual:** agrega compresión con pérdida, pero dicha pérdida se limita en gran medida a lo que el oído humano no puede percibir, con lo que se obtienen archivos muy pequeños sin apenas afectar a la calidad final. El formato más extendido es el MP3, pero hay muchos otros que ofrecen mejores prestaciones, como la evolución a **MP4** en su versión libre **OGG Vorbis**. Los formatos propietarios como **WMA** de Microsoft y **AAC**

de Apple también utilizan este método de codificación, aunque cada uno emplea sus propios algoritmos para ello.

- **Audio envolvente:** los sistemas estéreo ofrecen posicionamiento lateral mediante el uso de dos canales de audio independientes y dos altavoces. Formatos como **AC-3** y **Surround** agregan múltiples canales para posicionar altavoces frontales y traseros, mejorando la experiencia ambiental, sobre todo en películas.

1.4 ALGORITMOS DE COMPRESIÓN

Las abreviaturas se utilizan para reducir el tamaño de textos, sustituyendo ciertas palabras por versiones más breves y aplicando ciertas reglas: duplicación de letras en plurales (ferrocarril/F.C., ferrocarriles/FF.CC.), terminación en punto, etc. (etcétera).

Hay una multitud de **algoritmos** (conjuntos de instrucciones para realizar una acción dada) para reducir el tamaño de archivos digitales, algunos muy complejos, pero todos ellos se basan en la creación de un diccionario en el que se sustituyen términos grandes por pequeños, que puede ser externo o interno.

Las abreviaturas son un caso de **diccionario externo**, ya que no acompaña al documento abreviado. El formato gráfico GIF sustituye secuencias de colores iguales por su total (blanco, blanco, rojo, rojo, rojo por 2 blanco, 3 rojo), y cualquier programa preparado para reproducir archivos GIF puede interpretar este código. El problema de estos diccionarios es que el nivel de compresión es muy bajo, excepto en circunstancias especiales.

1.4.1 DICCIONARIO INTERNO

Supongamos que sustituimos las palabras de cierto tamaño en un texto por abreviaturas de cinco letras creadas para la ocasión, por ejemplo, *ab-aa*, *ab-3U*, *ab-g4*, etc. Si consideramos que disponemos de 58 caracteres distintos (24 minúsculas, 24 mayúsculas y 10 cifras), tenemos un total de 58 x 58 = 3.364 posibles abreviaturas para sustituir palabras de seis o más letras por valores desde *ab-00* a *ab-ZZ*. Aplicando este método, la primera línea del párrafo anterior quedaría así:

ab-00 que ab-01 las ab-02 de ab-03 ab-04 en un texto por ab-05 de cinco ab-06 ab-07 para la ab-08, por ab-09, ab-10, ab-11, ab-12, etc.

Actividades

- Descubrir por qué el algoritmo utilizado ha sustituido las expresiones *ab-aa*, *ab-3U* y *ab-g4* del final de la línea por las abreviaturas *ab-10*, *ab-11* y *ab-12*, a pesar de que tienen el mismo tamaño.

En apariencia, hemos eliminado 29 caracteres de 165, una reducción cercana al 18%, pero en realidad sólo hemos conseguido aumentar el texto. El problema es que el documento no será legible a menos que agreguemos un **diccionario interno** que permita invertir el proceso, por lo que al total anterior deberemos sumar la siguiente tabla:

Tabla 1.5. Ejemplo de diccionario

Abreviatura	Palabra
ab-00	Supongamos
ab-01	sustituimos
ab-02	palabras
ab-03	cierto
ab-04	tamaño
ab-05	abreviaturas
ab-06	letras
ab-07	creadas
ab-08	ocasión
ab-09	ejemplo
ab-10	ab-aa
ab-11	ab-3U
ab-12	ab-g4

En lugar de reducir el tamaño un 18%, hemos incrementado el resultado final en un 80%. Por supuesto podemos mejorar el algoritmo, por ejemplo haciendo que cualquier palabra que empiece por *Z* sea una abreviatura (*Z000*, *ZabD*, etc.), pero no obtendremos rendimiento hasta que las palabras sustituidas se repitan varias veces, compensando así el tamaño del diccionario.

Por ejemplo, cambiar la palabra *autorreferencialidad* (20 caracteres) por la abreviatura *Z0d* ahorra 17 caracteres (descontando el gasto de incorporarla al diccionario), pero dada su naturaleza es fácil que no vuelva a aparecer. En cambio, sustituir la palabra *para* por *ZBB* sólo ahorra un carácter, pero al ser inmensamente común resulta un cambio muy rentable en documentos de cierto tamaño.

1.4.2 CODIFICACIÓN CON ANÁLISIS PREVIO

Los sistemas de **doble pasada** realizan un análisis previo del archivo a codificar, localizando los elementos más interesantes antes de crear el diccionario. Siguiendo con el ejemplo anterior, el algoritmo debería leer todo el documento, anotar el ahorro total que representaría la sustitución de cada palabra por una

abreviatura de tres letras y seleccionar las 3.364 que ofrezcan un mejor resultado (análisis), sustituyéndolas por los valores desde Z00 a ZZZ en la segunda pasada (codificación).

Analizar por completo los millones de palabras posibles de un documento de gran tamaño puede llevar mucho tiempo y requerir la creación de una enorme base de datos, pero por lo menos conseguiremos la máxima compresión… ¿o no? En realidad, no hay ninguna necesidad de utilizar las palabras propiamente dichas; primero, porque el espacio es un carácter más, y es muy posible que si analizamos un texto encontremos múltiples repeticiones de cadenas complejas.

Por ejemplo, consideremos la palabra *son*, ampliamente utilizada. Al tratarse de un plural es muy fácil que siempre la encontremos precedida de un espacio y una ese (ellos son, algunos son, etc.), por lo que la compresión óptima seguramente sería *s_son*.

Admito que no tiene mucho sentido implementar miles y miles de reglas basadas en estructuras semánticas o gramaticales en un algoritmo de compresión, pero el ejemplo anterior nos demuestra que no es tan fácil alcanzar la compresión perfecta.

Otra cosa que no podemos olvidar cuando hablamos de codificación es que disponemos de dos fuentes con las que trabajar: el texto original y su equivalente binario. Por ejemplo, el **algoritmo de Huffman** es una variación del método del diccionario en la que se organiza los caracteres en forma de árbol, dando prioridad a los más utilizados de forma que el primero de ellos (la base) se asocia a un único bit, y cada nivel de las ramas a un número creciente de ellos. A medida que nos alejamos cada carácter necesita más bits para representarlo, pero como alejarse también significa que los utilizamos menos, se consigue un importante ahorro.

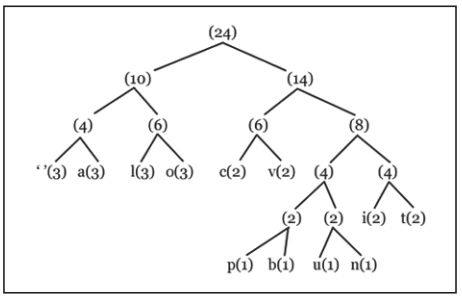

Figura 1.9. *Árbol de Huffman.*

El concepto parece sencillo, pero el proceso resulta algo complejo; bastante complejo. Peor aún, es sólo uno de los muchos métodos posibles para realizar una compresión sobre el código binario. Por eso no voy a explicar paso a paso la implementación del algoritmo de Huffman, aunque comprenderlo resulta un pasatiempo interesante.

Actividades

- Utilizar Internet para localizar y comprender el funcionamiento del algoritmo de Huffman (una sugerencia, empezar en: http://es.wikipedia.org).

Crear algoritmos que ignoren el origen y trabajen directamente sobre la codificación binaria hace que perdamos posibilidades de compresión, pero simplifica el proceso y lo generaliza; no podemos aplicar la sustitución de palabras a una imagen, pero sí a los bytes de los que se compone su equivalente binario.

Cuando estudiamos la digitalización de imágenes mencionamos la **codificación con pérdida**, en la que se eliminan detalles difíciles de apreciar para "suavizar" la imagen; ahora es evidente que dicho suavizado reduce los elementos no comunes y permite optimizar el uso de algoritmos basados en diccionarios. Lo mismo ocurre con la **codificación perceptual**, sólo que en este caso apuntamos a detalles que escapan al ojo y al oído humano, dependiendo la calidad del resultado final del umbral elegido.

Cada formato o elemento tiene su truco; por ejemplo, los archivos de vídeo agregan un nuevo factor a la ecuación: el tiempo. Durante muchas escenas hay elementos que permanecen constantes, típicamente el fondo; casi todos los sistemas de codificación de vídeo disponen de fotogramas clave, que contienen la imagen completa, y fotogramas progresivos, que sólo almacenan los cambios en la imagen desde el último fotograma clave. Por eso cuando una película se daña y perdemos un fotograma clave la imagen va apareciendo gradualmente, a medida que sus elementos se mueven o cambian.

Otro caso: los discos duros tienen unos espacios de almacenamiento mínimos, llamados clusters. Un archivo puede y suele ocupar varios, pero dos archivos no pueden compartir un mismo cluster, lo que lleva a que siempre se desperdicie un poco de espacio; 1.000 archivos de 1 KB pueden ocupar 4.000 KB, es decir, desperdiciar 3 MB. El **empaquetamiento** es la unión de varios archivos en uno, lo que implica la recuperación de dicho espacio, aunque después suele ser necesario desempaquetarlos para utilizarlos.

La verdad es que el ahorro obtenido al empaquetar no suele ser gran cosa, pero en combinación con los algoritmos de compresión nos permite agrupar múltiples archivos bajo el mismo diccionario, y debemos recordar que los diccionarios son más eficientes cuanto más grande sea el archivo a codificar.

También hay ocasiones en las que no merece la pena perder el tiempo comprimiendo pero sí queremos disfrutar de las ventajas del empaquetado, por ejemplo para enviar múltiples archivos adjuntos en un correo electrónico. A veces utilizamos un compresor que dispone de un sistema de recuperación para empaquetar datos vitales, sabiendo que en caso de un fallo menor del soporte podremos reparar los archivos.

1.5 PROTECCIÓN DE LA INTEGRIDAD

Tendemos a olvidar que los sistemas informáticos son falibles, pero la verdad es que alteraciones temporales de niveles de tensión, fallos en la emulsión del disco duro o incluso pequeños defectos de fabricación, hacen que los ordenadores cometan errores continuamente, sobre todo durante la transmisión de datos.

Pero si los equipos son tan susceptibles a este tipo de problemas deberíamos sufrir continuamente pérdidas de datos, y sabemos que no es así. Bien, lo que ocurre es que dichos datos se verifican cada vez que los manipulamos, permitiendo al sistema corregir los fallos que se producen.

1.5.1 PARIDAD

Un pico de tensión puede hacer que un cero se convierta en uno, y una caída tener el efecto contrario. El método más clásico para saber si un conjunto de bits ha sido alterado es sumar el número de unos y añadir un bit de paridad. En los sistemas de **paridad par** el bit hace que el número de unos sea par, y en los sistemas de **paridad impar**, el bit hace que el número de unos sea impar.

Mejor con un ejemplo, ¿verdad?

Supongamos que queremos aplicar protección de paridad al byte *10000111*. Como el número de unos que contiene es cuatro, el bit de paridad para un sistema de paridad par será *0* (*0+10000111*, ya que cuatro unos es par), mientras que para un sistema de paridad impar será *1* (*1+10000111*, convirtiendo el número de unos en cinco, impar).

De esta forma, si estamos utilizando paridad par y recibimos una serie de bytes, en el momento que el número de unos sea impar el ordenador sabrá que se ha producido un error y solicitará que se envíe de nuevo la información.

Actividades

- Descubrir el byte dañado en esta transmisión basada en paridad par:

  0-11001010 1-10001111 1-11100001 0-11000011

El sistema de paridad es popular porque es muy rápido, pero ofrece muy poca seguridad, ya que la alteración de dos bits, o de un número par de bits, mantendrá la paridad pero habrá dañado los datos.

Por ejemplo, el byte *1-11000001* es correcto bajo paridad par (cuatro unos), y si modificamos un bit saltaría la alarma (*1-10000001* es impar), pero cambiando dos sigue cumpliendo la regla (*1-10000000* sigue siendo par), aunque no se pare-

ce en nada al original. Hay que darse cuenta de que la información de paridad también se transmite, por lo que es posible que ésta sufra daños y se produzca un falso error.

Como la paridad simple falla en un cincuenta por ciento de los casos, es evidente que necesitamos un sistema mejor. Una solución mucho más eficiente es la **paridad cruzada**, que agrupa un conjunto de bytes en forma de columna, almacenando tanto la paridad de las filas como la de las columnas.

Tabla 1.6. Paridad Cruzada

Original	Errónea		
10011101	***1***	10011110	***1***
11001100	***0***	11001100	***0***
00010011	***1***	00010011	***1***
10010110	***0***	10010110	***0***
11010100	***11010100***		

La tabla muestra un ejemplo de paridad par, y una transmisión en la que se ha producido un fallo en los dos últimos bits del primer byte, indetectable para la paridad horizontal pero que aparece en la vertical.

Actividades

- Modificar la transmisión de la tabla anterior hasta obtener un error indetectable.

Es evidente que para que un error pase desapercibido, deberá modificar un número par de bits en las filas y a su vez en las columnas correspondientes; es decir, que el mínimo error indetectable posible serán cuatro bits dispuestos formando un rectángulo, algo bastante improbable.

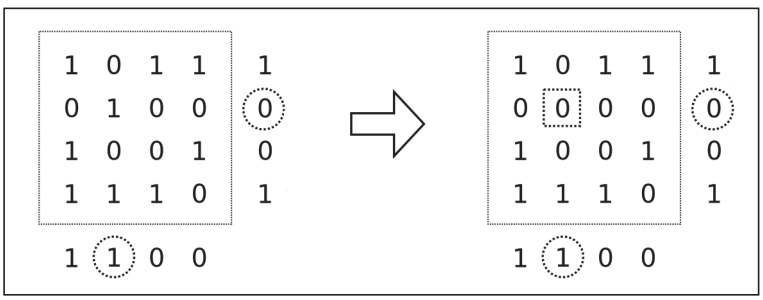

Figura 1.10. *Fallo de la paridad cruzada.*

Pero la improbabilidad no asegura que dichos errores indetectables no ocurran; es más, dada la ingente cantidad de datos digitales que se manipulan diariamente, la ley de probabilidades nos indica que ocurrirán continuamente. Pero antes de estudiarlos, hay algo que debemos tener claro; no hay un solo método que sea totalmente seguro, aunque a medida que los mejoramos es cada vez más difícil que se produzca una combinación de errores que pueda pasar desapercibida.

Uno de los sistemas de detección de errores más utilizado son los **Códigos de Redundancia Cíclica**, o **CRC**. Se basan en la división del código a verificar por un valor acordado previamente, llamado **polinomio generador**. El cociente se combina con el valor original en un proceso matemático relativamente complejo y se envía. En el destino se divide el código por el polinomio generador, debiendo dar resto cero para saber que no se han producido alteraciones, y regenerándose a continuación el mensaje original.

Los sistemas basados en redundancia cíclica suelen añadir 16 bits por cada 128 transmitidos, lo que da un buen nivel de eficiencia. Detectan todos los errores de uno, dos bits, bits impares, cadenas de pequeño tamaño y casi el 100% de los errores en cadenas de gran tamaño, lo que les convierte en uno de los mejores métodos para asegurar la integridad de la información.

1.5.2 RECUPERACIÓN DE ERRORES

Pensamos que cuando grabamos un archivo en un CD no va a cambiar, pero la verdad es que un rayonazo, o simplemente la forma en la que el paso del tiempo afecta a la emulsión del mismo, pueden provocar una alteración de los datos. Lo mismo ocurre con todos los demás soportes; errores de superficie, picos de tensión o la degeneración del medio pueden alterar datos vitales.

La aproximación clásica a la protección de datos es la **redundancia**, guardando duplicados de la información original en otros medios, es decir, realizando **copias de seguridad**. Estos sistemas han evolucionado mucho, por ejemplo ahorrando espacio mediante **copias incrementales** (guardan automáticamente los archivos modificados desde la última vez que se hizo una copia de seguridad), agregando sistemas de compresión y realizando verificaciones de la integridad de los datos guardados.

Una de las aplicaciones más fascinantes del concepto de redundancia son los sistemas **RAID**, que combinan varios discos haciendo que colaboren entre sí. Como su nombre indica, **RAID 0** no tiene redundancia, simplemente reparte fragmentos de información entre los distintos discos, haciéndolos trabajar en **paralelo**, lo que significa un gran aumento de las velocidades de lectura y escritura al precio de que si uno de los discos falla, perderemos la información de todos ellos.

RAID 1 trabaja en modo **espejo**, donde la información de un disco se duplica en otro. Si uno de ellos se avería basta con retirarlo (lo que se llama *romper el espejo*), agregar uno nuevo e indicar al espejo que se regenere. La idea se parece a

una copia de seguridad pero hay que tener en cuenta que esto sólo protege contra fallos físicos, ya que un borrado accidental o una alteración causada por virus se transmitirá a ambos elementos del espejo. Además, como realizamos una copia 1:1 se desperdicia muchísimo espacio; el 50% se asigna a la copia de seguridad.

RAID 5 es una mezcla de los dos procedimientos anteriores. Combina múltiples discos duros trabajando en paralelo, como ocurría con RAID 0, pero además utiliza un disco para guardar la información de paridad de los demás, tal y como muestra la siguiente tabla:

Tabla 1.7. Raid 5

Disco 1	Disco 2	Disco 3	Disco 4 (paridad par)
0000 0001	0100 0000	0000 0000	0100 0001
1100 1010	0011 1001	1110 1010	0001 1001
1001 1001	0001 0110	1000 1111	0000 0000
0000 0010	0010 0010	0010 1001	0000 1001

Si uno de los discos falla, podremos sustituirlo y hacer que el RAID recupere la información almacenada en su interior haciendo que se cumpla la paridad. Además, al utilizar un único disco para guardar la información de recuperación, el espacio desperdiciado disminuye; en un RAID 5 de seis discos sólo perderemos un sexto del espacio total.

Por ejemplo, consideremos la primera línea de la tabla anterior conteniendo los cuatro discos duros, suponiendo que el segundo se avería con lo que tenemos la siguiente situación: *0000 0001 xxxx xxxx 0000 0000 0100 0001*

Si aplicamos paridad a la información de los otros tres discos, obtendremos de nuevo el contenido original: *0000 0001 0100 0000 0000 0000 0100 0001*

Actividades

- Eliminar el tercer disco duro y recuperarlo aplicando paridad.

Por supuesto, si el daño afecta a más de un disco duro estaremos perdidos; al igual que ocurría con la verificación de errores, no hay un sistema seguro al 100%.

Volviendo a los archivos digitales convencionales, por ejemplo almacenados en un CD, lo habitual es que la verificación CRC descubra que el archivo ha sido dañado, y eso es todo. Si no disponemos de otra copia, habremos perdido la infor-

mación. Pero al hablar de la redundancia cíclica mencionábamos que al dividir la información obtenida por el polinomio generador el resto debía ser cero. ¿Influyen los distintos tipos de resto? ¿Es posible regenerar el código original según el resto obtenido?

Existen programas de recuperación de datos que se basan en este principio. Dado que cada tipo de archivo utiliza una estructura interna distinta, suele ser necesario un programa para cada uno de ellos, lo que reduce su eficiencia.

Acabamos de ver que RAID 5 sólo necesita utilizar una fracción del espacio total para almacenar la información que le permitirá recuperar datos perdidos. Análogamente, y utilizando ciertos procesos matemáticos, algunos sistemas de archivo pueden aumentar las posibilidades de recuperación. Por ejemplo, el compresor **WinRAR** nos permite añadir un **registro de recuperación**, cuyo tamaño oscila del 1 al 3% del total, que le permitirá restaurar archivos dañados… o al menos ligeramente dañados.

Resumiendo, el primer paso para proteger la integridad de los datos es descubrir errores, típicamente a través de una verificación CRC, y a partir de aquí la solución dependerá de los sistemas de copia de seguridad o restauración que estemos utilizando, siendo una buena idea empaquetar los archivos de las copias de seguridad mediante algún sistema con redundancia como el registro de recuperación de WinRAR, para reducir el riesgo de que una posible degeneración del medio de destino destruya la copia de seguridad.

1.6 PROTECCIÓN DE LA CONFIDENCIALIDAD

La integridad de los datos no es la única preocupación de los usuarios; en ocasiones es preferible perder ciertos archivos que dejar que sean conocidos públicamente. Día a día el riesgo aumenta; equipos de uso compartido, intrusiones informáticas, transmisión de datos a través de Internet, unidades de almacenamiento portátiles que pueden perderse o robarse.

1.6.1 CIFRADO

Cuando estudiamos la codificación de elementos analógicos, vimos que obteníamos un archivo digital. Igualmente, es posible codificar un archivo digital en forma de otro archivo digital, por ejemplo sustituyendo los unos por ceros, y viceversa, lo que haría imposible su lectura a menos que lo hagamos pasar por el proceso inverso.

La ciencia que estudia este tipo de problemas es la **criptografía**, que utiliza procedimientos matemáticos para **cifrar** un documento o archivo (también se utiliza la palabra **encriptar**, derivada del inglés, para describir este proceso) median-

te un documento que lo hace ilegible, y que se puede aplicar posteriormente para restaurarlo a su estado original.

El ejemplo de la transposición de ceros y unos es muy simple, pero podemos imaginar métodos más complicados. Por ejemplo, podemos sumar a cada byte uno de los decimales del **número trascendente Pi** (es trascendente porque tiene un número infinito de decimales sin una secuencia de repetición deducible), creando así resultados casi aleatorios; pero si todo el mundo utiliza codificadores basados en *Pi*, todos podrían utilizarlos para leer los archivos codificados de otras personas.

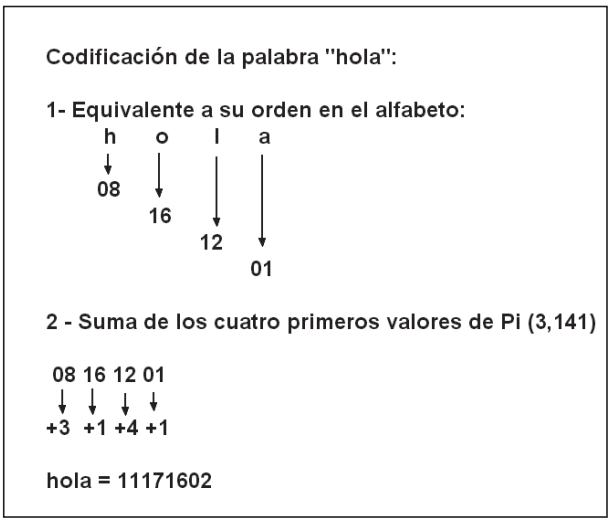

Figura 1.11. *Codificación en PI.*

Si bien es cierto que cada usuario podría utilizar números trascendentes distintos, pero tampoco es que crezcan en los árboles. Por tanto, se introduce un factor aleatorio en los algoritmos de cifrado, llamado **clave**, de forma que el resultado obtenido con un determinado algoritmo cambiará según la clave elegida.

Esto deja abiertas dos vías de ataque: una, la deducción del funcionamiento del algoritmo, encontrando alguna vulnerabilidad que permita deducir su contenido, y dos, la obtención de dicha clave.

De ahí la importancia de elegir claves complejas, que incluyan valores numéricos e incluso símbolos especiales, y preferiblemente que no tengan ningún sentido o paralelismo con la vida real; es decir, no vale sumar la fecha de boda con la de nacimiento, o agregar un par de ceros en medio de un nombre. Tampoco es conveniente utilizar durante mucho tiempo las mismas claves, ya que un cambio periódico puede echar por tierra los avances de un agresor.

Tampoco debemos despreciar el tamaño de la clave. Cuatro caracteres ASCII son 32 bits, lo que representa más de cuatro mil millones de combinaciones, pero si nos limitamos a escribir números y minúsculas tenemos poco más de un millón de variaciones (muchas menos si consideramos que el ser humano tiende a usar ciertas letras y combinaciones inconscientemente), un parpadeo para los sistemas de análisis informático. Normalmente se considera una longitud segura a partir de los ocho caracteres, siempre y cuando tengamos un poco de imaginación.

El problema de las claves muy complejas es que es fácil olvidarlas, y ponerlas por escrito no ayuda a proteger su confidencialidad. Una solución de compromiso es utilizar múltiples claves, por ejemplo una sencilla para el acceso a ciertas páginas web, mediana para el correo electrónico y una extremadamente compleja para aquellas operaciones que impliquen efectivo. Otra solución es utilizar un programa que almacene de forma codificada nuestras contraseñas, haciendo así que sólo tengamos que recordar la clave de acceso al mismo (y hacer copias de seguridad del archivo asociado, por supuesto).

1.6.2 CIFRADO DE CLAVE PÚBLICA

El cifrado mediante claves funciona mientras que el usuario accede a sus archivos, pero ¿qué pasa cuando quiere transmitirlos a otra persona? Para empezar tendría que inventar una nueva clave, ya que no debe revelar una de las suyas. Después está el problema de enviarla; no puede utilizar el mismo medio, ya que entonces cualquiera que esté espiando la transmisión de datos la recibiría.

Esto es especialmente importante en Internet, ya que la codificación debe establecerse antes de que utilicemos nuestra contraseña de acceso a un sitio dado. De nuevo el huevo o la gallina.

Los sistemas de **cifrado de clave pública** solucionan este problema utilizando parejas de claves, también llamadas llaves. La **llave pública** se entrega a todo el mundo pero sólo sirve para codificar la información, siendo necesaria la **llave privada** en poder del usuario para decodificarla.

El ejemplo típico es la conexión a un banco on-line. Cuando el navegador accede a la página recibe automáticamente la llave pública del mismo, de forma que cuando introducimos nuestros datos de acceso y contraseña viajan en forma codificada, estableciendo un **canal seguro de comunicación** que únicamente el servidor del banco puede leer, ya que nadie más posee la llave privada. Esto se muestra mediante la aparición de un pequeño candado en la barra de navegación, y un cambio de color de la misma. La cabecera también cambia, pasando de *http://* a *https://* (donde la *ese* significa *secure*).

Figura 1.12. *Canal seguro.*

Muchos programas relacionados con Internet, como los clientes de correo, disponen de opciones para el uso de llaves públicas. Eso significa que podemos "coleccionar" las llaves de nuestros corresponsales y utilizarlas para que nuestras comunicaciones sean seguras, a la vez que ellos hacen lo mismo.

A medida que se extiende el uso de llaves públicas se hace necesario el uso de un programa **llavero**, que genera, contiene y administra nuestras llaves pública y privada (que, por cierto, necesita una clave para ser activada, lo que asegura su integridad en caso de pérdida o copia), así como las llaves públicas de los demás usuarios; estos programas también se ocupan de suministrarlas a otros programas, como los clientes de correo que mencionamos antes, haciendo que toda la experiencia sea simple.

Actividades

- Generar un conjunto de llaves y utilizarlas para codificar y decodificar un archivo (sugerencia: utilizar el programa gratuito *Windows Privacy Tools*, disponible en www.winpt.org).

Parece que disponemos de una forma segura de comunicación… al menos, en teoría.

Uno de los peligros de Internet es el **spoofing**, en el que se engaña al usuario haciéndole creer que se está conectando a un sitio determinado, ya sea mediante métodos relacionados con el funcionamiento de Internet (como el envenenamiento ARP), o utilizando una dirección similar a la de destino, lo que se denomina **phishing**.

Por ejemplo, es posible que creamos que estamos accediendo a un banco a través de un canal seguro, cuando en realidad hemos abierto una información codificada con el falso sitio, que a su vez ha abierto otro canal al banco. Al actuar como **proxy** (intermediario), el sitio intermedio es capaz de ver todo el tráfico que le atraviesa, incluyendo nuestras contraseñas de acceso.

Incluso sin recurrir a direcciones falsas, nuestro tráfico codificado puede ser espiado desde elementos

Figura 1.13. *Ejemplo de spoofing.*

intermedios si nuestro equipo es parte de una red de área local, como las utilizadas en empresas o cibercafés. Por tanto, debemos considerar inseguras estas redes, a menos que conozcamos su estructura. En particular, nunca deberíamos transmitir información confidencial a través de una red wireless que "casualmente" encontremos abierta al público, si bien es cierto que la mayoría se deben al despiste congénito de sus propietarios, nada nos dice que no se trate de una trampa para ver el tráfico que las atraviesa.

1.6.3 CERTIFICADOS

Para evitar el **phishing** basta con ser capaces de distinguir entre la clave pública del banco y otro cualquiera. Los **certificados** son claves públicas que han sido asociadas a una determinada empresa por una **entidad certificadora**. Cuando un navegador recibe un certificado automáticamente se conecta a dicha entidad, o a una que pertenezca a su círculo de confianza, para comprobar que es correcto.

Para verificar el certificado de un sitio web una vez establecido un canal seguro bastará con hacer **doble clic sobre el candado del navegador**… pero no solemos hacerlo, ¿no es así? De forma que los certificados son una gran herramienta para evitar estafas, pero rara vez los usamos correctamente.

Afortunadamente los diseñadores de navegadores van agregando funciones anti-phishing, ayudando con mayor o menor éxito en la detección de falsas páginas web.

Actividades

- Acceder a la página web de un banco y comprobar su certificado.

Un último comentario: hay gran cantidad de sitios web que solicitan un certificado sin necesitarlo, y luego olvidan renovarlo pero mantienen la misma clave pública, de forma que la gente que accede a ellos obtiene mensajes del tipo "El certificado ha expirado, ¿seguro que quiere continuar?", lo que provoca cierta ansiedad. Así que si la información que suministramos no es estrictamente confidencial, no deberíamos preocuparnos demasiado.

1.6.4 FIRMA DIGITAL

En muchos casos, necesitamos enviar documentos importantes a través de Internet, asegurando su integridad y su origen; es decir, que el receptor sepa con toda seguridad que no es falso, o que no ha sido interceptado y modificado por alguien. Los certificados digitales podrían ayudar, pero no todo el mundo puede tenerlos ni disponemos de un servidor para colgarlos.

Antes mencionamos que en los sistemas de clave pública la llave privada (codificadora) tiene una contraseña para que, en caso de pérdida o robo, no pueda ser utilizada por otra persona. ¿Y si lo hacemos al revés? ¿Y si repartimos la llave codificadora sin clave, mientras que guardamos una decodificadora con ella? Tendremos justo lo que buscábamos, un sistema de **firma digital**, en el que todo el mundo puede leer lo que enviamos, asociándolo con nosotros a través de nuestra llave, pero nadie puede modificarlo.

El problema de este sistema es que, como hemos dicho, todo el mundo puede decodificar y leer el documento; pero hay una solución. Veamos paso a paso el proceso utilizado por los sistemas de llave pública para asegurar la confidencialidad, integridad e identidad:

- **Uso de la llave pública codificadora del destinatario sobre el documento:** este primer paso asegura la confidencialidad, ya que a partir de este momento sólo el destinatario será capaz de abrirlo, pero no la identidad ni la integridad, ya que alguien podría utilizar dicha llave pública para enviar un documento falso.
- **Creación de un código suma (hash) asociado al documento codificado:** se trata de un proceso matemático que crea un número único asociado a la estructura binaria de un documento; si éste es modificado el cálculo del hash en destino no coincidirá; se trata de un método muy parecido a los sistemas de recuperación de errores que ya hemos visto (paridad, CRC), con la diferencia de que el hash siempre tiene el mismo tamaño.
- **Firma digital del hash:** en este momento se asegura simultáneamente la identidad y la integridad; todo el mundo puede utilizar nuestra llave decodificadora pública para obtener el hash y aplicarlo al documento, comprobando que es correcto y nuestro, pero no podrán leer el contenido a menos que sean sus destinatarios.

Actividades

- Firmar digitalmente los archivos codificados en la actividad del punto 1.6.2.

Por supuesto, para que este sistema funcione correctamente tenemos que hacer llegar la llave decodificadora pública al destinatario de una forma segura, ya sea a través de un e-mail previo y confirmado telefónicamente, o depositando su creación en la entidad destinataria. Por ejemplo, el Ministerio de Hacienda permite el envío telemático de documentación si solicitamos previamente un certificado digital que contiene las llaves necesarias para la codificación del documento y su firma digital.

1.7 ALMACENAMIENTO FÍSICO DE LA INFORMACIÓN

Ya hemos visto que la protección de la información y su confidencialidad dependen en gran medida del medio que estemos usando; no hay RAID para CD-ROM, ni necesitamos un certificado digital si no disponemos de conexión a Internet.

Por eso, y aunque en los próximos capítulos estudiaremos en profundidad la estructura interna de los sistemas informáticos, incluyendo los sistemas de almacenamiento y los procesos asociados a ellos, como la grabación de CD/DVD, vamos a efectuar un breve repaso de los que podemos encontrar, para que al menos conozcamos sus características principales:

- **Unidades de cinta:** también llamadas **streamers**, trabajan de forma similar a una cinta magnetofónica y proporcionan la mejor relación entre espacio de almacenamiento y precio, por lo que son muy populares en los sistemas de copia de seguridad. Lamentablemente resultan muy lentas, sobre todo teniendo en cuenta que debemos acceder a ellas secuencialmente, es decir, empezando por el principio hasta que encontramos los datos buscados, y además existe el riesgo de que la emulsión se degrade con el paso del tiempo, dañando su contenido.

Figura 1.14. *Unidad de cinta.*

- **Disquetes:** utilizan el mismo principio que las unidades de cinta, pero en este caso la emulsión se aplica sobre un disco de plástico lo que permite evitar el acceso secuencial y acelerar el proceso de lectura y escritura (esta última sólo será posible si la pestaña de protección está bajada); lógicamente, la superficie útil es mucho más pequeña que en una cinta, por lo

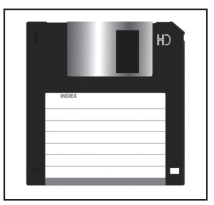

Figura 1.15. *Disquete.*

que el espacio de almacenamiento será muy reducido. Por lo demás los problemas son los mismos, tiempos de acceso elevados en comparación con otros medios, degeneración de la emulsión, etc.

- **Discos duros:** también son sistemas magnéticos, pero en este caso la emulsión se encuentra sobre uno o varios discos de metal que giran a gran velocidad combinando así durabilidad, elevado espacio de almacenamiento a un bajo precio y altas velocidades de acceso, lo que les convierte en el medio de almacenamiento por excelencia. El mayor problema de los discos duros es que el cabezal de lectura/escritura nunca debe tocar la superficie del disco, por lo que un pequeño golpe durante su funcionamiento puede averiarlos (en cambio, soportan tremendos impactos cuando están apagados); esto es particularmente cierto en los discos duros externos, debiendo extremar las precauciones cuando los estemos utilizando.

Figura 1.16. *Disco duro.*

- **CD:** sustituye el soporte magnético por marcas en una emulsión reflectante, sobre las que se refleja un haz láser. Esto permite obtener precisiones muy elevadas y una gran durabilidad, ya que no hay contacto físico con el medio. Un CD puede tener una capacidad de 640 MB a 700 MB en los formatos estándar y hasta 900 MB en los extendidos, que sólo son soportados por ciertas grabadoras. La velocidad de transmisión de datos básica es de 150 KB/s (1x), pero no es raro que las unidades modernas alcancen los 40x en lectura (150x40/1000 = 6 MB/s) y 10x en escritura (1,5 MBps). Además del clásico **CD-ROM** que sólo se puede grabar una vez, el mercado también nos ofrece **CD-RW**, que pueden ser borrados y reescritos un elevado número de veces.

Figura 1.17. *CD/DVD.*

- **DVD:** los DVD son una mejora del formato CD que utiliza un sistema láser más preciso, obteniendo una densidad de datos de 4,7 GB sobre una superficie similar a la de un CD-ROM, y el doble (9,2 GB) si el dispositivo es capaz de grabar las dos caras. Algunas grabadoras también pueden modificar la emisión láser para acceder a una segunda capa, obteniendo 8,5 GB al grabar una cara a doble capa o 17 GB si grabamos dos caras a doble capa. Respecto a la velocidad, los multiplicadores de DVD no coinciden con los de CD, siendo un DVD 1x aproximadamente igual a un CD 9x.

- **Otros sistemas ópticos:** en este momento disponemos de variaciones del sistema CD/DVD que ofrecen espacios de almacenamiento de 25GB a 300GB, dependiendo del sistema utilizado (Blue Ray, HD-DVD, Tapestry). Es cierto que estos enormes tamaños de almacenamiento son muy jugosos, pero hasta que el mercado no se decida por un formato resulta muy arriesgado elegir uno de ellos debido a la elevada inversión que representan.

- **Memorias no volátiles; de la ROM a la EEPROM:** todos los sistemas informáticos utilizan **unidades de memoria volátiles** (RAM, SDRAM, DDR, etc.) para procesar la información; su velocidad es elevadísima, pero al apagar el equipo pierden los datos almacenados. Desde hace muchos años existen versiones permanentes de estas memorias, pero el proceso de grabación es complicado y lento; lo que las hace poco prácticas. Las **ROM/PROM** sólo se grababan una vez, las **EPROM** necesitaban una exposición a luz ultravioleta, y las **EEPROM** un borrado completo mediante un pulso eléctrico.

Figura 1.18. *Memoria EPROM.*

- **Memorias FLASH:** se trata de memorias no volátiles (es decir, no pierden la información cuando desconectamos la alimentación eléctrica) que pueden ser modificadas dato a dato, evitando los problemas asociados a las EEPROM. El acceso es mucho más lento que en las volátiles, sobre todo durante la escritura, pero aun así algunas son más rápidas que los discos duros. Su uso se ha extendido a través del medio digital, siendo el soporte favorito de cámaras, agendas electrónicas y en general cualquier medio portátil, sobre todo en forma de pequeños "lápices" USB cuya capacidad puede ser de varios GB. Hay que tener en cuenta que aunque estas unidades soportan un elevado número de escrituras, pueden acabar fallando con el uso, por lo que es importante disponer de copias de seguridad de sus contenidos.

Figura 1.19. *Memoria FLASH.*

- **Papel:** ya hemos visto que es una mala política anotar las contraseñas, también es cierto que olvidarlas no nos ayuda en nada, sobre todo en el caso de domicilios domésticos y archivos personales no relacionados con actividades económicas, donde es raro que un intruso se tome la molestia de utilizarlas. Además, no es la primera vez que alguien pierde toda su agenda telefónica al dañarse la memoria FLASH de la tarjeta SIM. Y por supuesto, es mucho más cómodo leer ciertos documentos sobre el papel… como este libro.

RESUMEN DEL CAPÍTULO

- La información digital sólo puede representar de forma precisa valores matemáticos, pero puede obtener resultados aproximados de elementos analógicos mediante el proceso conocido como **digitalización**. Con esto se produce una inevitable pérdida de calidad, pero se gana durabilidad y facilidad de manejo.
- El **código binario** es la base del funcionamiento de los sistemas digitales y el origen de muchas de sus características, como tamaños de almacenamiento, velocidades de transmisión, etc. Por eso es importante entender sus fundamentos, aunque no se utilice habitualmente.
- Las **imágenes digitales matriciales** constan de dos factores principales: su **resolución** (número de puntos de que se componen) y **profundidad de color** (información dedicada a cada punto), pudiendo afectar al resultado final el formato elegido para guardarlas, sobre todo en aquellos que utilizan **compresión con pérdida**.
- Los **archivos de audio digital** se definen por el **número de canales** de que se componen, la **frecuencia de muestreo** (número de mediciones por segundo de la señal analógica) y el **bitrate** (bits por segundo dedicados al almacenamiento de la información). Al igual que ocurría con las imágenes el formato elegido para el almacenaje es importante, ya que en muchos de ellos es habitual utilizar distintos métodos de **codificación perceptual** (comprime la señal de audio eliminando aquellos elementos que el oído humano no detecta).
- Los **sistemas de compresión** utilizan algoritmos para reducir el tamaño ocupado por un archivo digital, siendo necesario descomprimir el archivo para utilizarlo normalmente.
- Todos los sistemas informáticos utilizan algoritmos de detección de errores, como la paridad, para comprobar la integridad de las comunicaciones. Entre los más eficientes se cuentan los códigos de redundancia cíclica, con una eficiencia cercana al 100%.
- Algunos sistemas de compresión y copia de seguridad ofrecen la capacidad de añadir datos extra al algoritmo de verificación para facilitar la **reparación** del archivo en caso de que sufra daños.

- La **codificación basada en contraseñas** es el principal método para la protección de la confidencialidad. En el caso de comunicaciones a través de sistemas abiertos, como Internet, suele ser necesario crear previamente un **canal seguro** mediante el uso de **claves públicas**.
- Los certificados se utilizan para determinar de forma segura la identidad de un interlocutor, normalmente una empresa. **Siempre que vayamos a transferir datos confidenciales a través de un canal seguro de un navegador deberemos verificar el certificado haciendo doble clic en el símbolo del candado**.
- La **firma digital** sirve para identificar la identidad del emisor de un documento, así como la integridad del mismo; cualquier modificación del documento hará que el **hash** de la firma no coincida.

EJERCICIOS PROPUESTOS

- **1.** Obtener el tiempo mínimo aproximado que tardará en transmitirse 1GB a través de una conexión ADSL de 1Mb/s.
- **2.** Calcular el tamaño real de un disco duro de 300GB (recordemos que los fabricantes utilizan múltiplos decimales en lugar de binarios).
- **3.** Un sistema utiliza direcciones desde la 0000h hasta la FFFFh. ¿De cuántas combinaciones posibles estamos hablando?
- **4.** Queremos imprimir una imagen de 1024x768 píxeles. Indicar los tamaños en centímetros con los que obtendremos calidad óptima (300ppp) y media-baja (150ppp).
- **5.** Calcular la resolución en píxeles de una imagen para su impresión en un tamaño de 50x40cm, a 300ppp.
- **6.** Un archivo de audio estéreo de una hora de duración ha sido muestreado con un bitrate de 192Kbps. ¿Qué tamaño ocupará?
- **7.** Dibujar un esquema de la distribución de llaves públicas y privadas para el intercambio de información codificada entre cuatro interlocutores.

TEST DE CONOCIMIENTOS

1 16 bits tienen un total de:
- **a)** 64.000 combinaciones posibles.
- **b)** 1.024 combinaciones posibles.
- **c)** 65.536 combinaciones posibles.
- **d)** 1KB de combinaciones posibles.

2 10 bits tienen un total de:
- **a)** 64.000 combinaciones posibles.
- **b)** 1.024 combinaciones posibles.
- **c)** 65.536 combinaciones posibles.
- **d)** 1KB de combinaciones posibles.

3 El formato GIF tiene:
- **a)** Compresión con pérdida de calidad.
- **b)** Un máximo de 256 colores.
- **c)** Una resolución fija de 300ppp.
- **d)** Compresión perceptual.

4 El formato JPEG tiene:
- **a)** Compresión con pérdida de calidad.
- **b)** Un máximo de 256 colores.
- **c)** Una resolución fija de 300ppp.
- **d)** Compresión perceptual.

5 La paridad detecta los errores:
- **a)** De los bits pares.
- **b)** De los bits impares.
- **c)** De un número par de bits.
- **d)** De un número impar de bits.

6 La paridad se utiliza para recuperar datos en:
- **a)** Los sistemas RAID 0.
- **b)** Los sistemas RAID 1.
- **c)** Los sistemas RAID 5.
- **d)** Los bits impares del primer viernes de cada mes.

7 Los sistemas de compresión de archivos:
- **a)** Siempre reducen el resultado a la mitad del original.
- **b)** Utilizan códigos de redundancia para la detección de errores.
- **c)** Eliminan los elementos no perceptibles por el oído humano.
- **d)** Pueden recuperar cualquier archivo dañado en su interior.

8 Las contraseñas:
- **a)** Son seguras a partir de una longitud determinada.
- **b)** Son seguras si no se corresponden con hechos deducibles.
- **c)** Son seguras si no se escriben.
- **d)** Nunca son totalmente seguras.

9 Los sistemas de llave pública son vulnerables a:
- **a)** Que un usuario remoto descubra la clave de codificación.
- **b)** Que un usuario remoto modifique la clave de codificación.
- **c)** Que un usuario remoto utilice su propia clave de codificación.
- **d)** Nada de lo anterior.

10 Para que un canal seguro sea realmente seguro deberemos:
- **a)** Comprobar que realmente estamos conectados al destinatario.
- **b)** Generar nuestras propias parejas de llaves.
- **c)** No revelar nunca la clave de codificación.
- **d)** Todo lo anterior.