expanding, disk costs are dropping dramatically, so storing the entire Web may continue to be feasible for large companies for the foreseeable future.

Making sense of this data is another matter. You can appreciate how XML can help programs extract the structure of the data easily, while ad hoc formats will lead to much guesswork. There is also the issue of conversion between formats, and even translation between languages. But even knowing the structure of data is only part of the problem. The hard bit is to understand what it means. This is where much value can be unlocked, starting with more relevant result pages for search queries. The ultimate goal is to be able to answer questions, for example, where to buy a cheap but decent toaster oven in your city.

A third aspect of Web search is that it has come to provide a higher level of naming. There is no need to remember a long URL if it is just as reliable (or perhaps more) to search for a Web page by a person's name, assuming that you are better at remembering names than URLs. This strategy is increasingly successful. In the same way that DNS names relegated IP addresses to computers, Web search is relegating URLs to computers. Also in favor of search is that it corrects spelling and typing errors, whereas if you type in a URL wrong, you get the wrong page.

Finally, Web search shows us something that has little to do with network design but much to do with the growth of some Internet services: there is much money in advertising. Advertising is the economic engine that has driven the growth of Web search. The main change from print advertising is the ability to target advertisements depending on what people are searching for, to increase the relevance of the advertisements. Variations on an auction mechanism are used to match the search query to the most valuable advertisement (Edelman et al., 2007). This new model has given rise to new problems, of course, such as **click fraud**, in which programs imitate users and click on advertisements to cause payments that have not been fairly earned.

7.4 STREAMING AUDIO AND VIDEO

Web applications and the mobile Web are not the only exciting developments in the use of networks. For many people, audio and video are the holy grail of networking. When the word "multimedia" is mentioned, both the propellerheads and the suits begin salivating as if on cue. The former see immense technical challenges in providing voice over IP and video-on-demand to every computer. The latter see equally immense profits in it.

While the idea of sending audio and video over the Internet has been around since the 1970s at least, it is only since roughly 2000 that **real-time audio** and **real-time video** traffic has grown with a vengeance. Real-time traffic is different from Web traffic in that it must be played out at some predetermined rate to be useful. After all, watching a video in slow motion with fits and starts is not most

people's idea of fun. In contrast, the Web can have short interruptions, and page loads can take more or less time, within limits, without it being a major problem.

Two things happened to enable this growth. First, computers have became much more powerful and are equipped with microphones and cameras so that they can input, process, and output audio and video data with ease. Second, a flood of Internet bandwidth has come to be available. Long-haul links in the core of the Internet run at many gigabits/sec, and broadband and 802.11 wireless reaches users at the edge of the Internet. These developments allow ISPs to carry tremendous levels of traffic across their backbones and mean that ordinary users can connect to the Internet 100–1000 times faster than with a 56-kbps telephone modem.

The flood of bandwidth caused audio and video traffic to grow, but for different reasons. Telephone calls take up relatively little bandwidth (in principle 64 kbps but less when compressed) yet telephone service has traditionally been expensive. Companies saw an opportunity to carry voice traffic over the Internet using existing bandwidth to cut down on their telephone bills. Startups such as Skype saw a way to let customers make free telephone calls using their Internet connections. Upstart telephone companies saw a cheap way to carry traditional voice calls using IP networking equipment. The result was an explosion of voice data carried over Internet networks that is called **voice over IP** or **Internet telephony**.

Unlike audio, video takes up a large amount of bandwidth. Reasonable quality Internet video is encoded with compression at rates of around 1 Mbps, and a typical DVD movie is 2 GB of data. Before broadband Internet access, sending movies over the network was prohibitive. Not so any more. With the spread of broadband, it became possible for the first time for users to watch decent, streamed video at home. People love to do it. Around a quarter of the Internet users on any given day are estimated to visit YouTube, the popular video sharing site. The movie rental business has shifted to online downloads. And the sheer size of videos has changed the overall makeup of Internet traffic. The majority of Internet traffic is already video, and it is estimated that 90% of Internet traffic will be video within a few years (Cisco, 2010).

Given that there is enough bandwidth to carry audio and video, the key issue for designing streaming and conferencing applications is network delay. Audio and video need real-time presentation, meaning that they must be played out at a predetermined rate to be useful. Long delays mean that calls that should be interactive no longer are. This problem is clear if you have ever talked on a satellite phone, where the delay of up to half a second is quite distracting. For playing music and movies over the network, the absolute delay does not matter, because it only affects when the media starts to play. But the variation in delay, called **jitter**, still matters. It must be masked by the player or the audio will sound unintelligible and the video will look jerky.

In this section, we will discuss some strategies to handle the delay problem, as well as protocols for setting up audio and video sessions. After an introduction to

digital audio and video, our presentation is broken into three cases for which different designs are used. The first and easiest case to handle is streaming stored media, like watching a video on YouTube. The next case in terms of difficulty is streaming live media. Two examples are Internet radio and IPTV, in which radio and television stations broadcast to many users live on the Internet. The last and most difficult case is a call as might be made with Skype, or more generally an interactive audio and video conference.

As an aside, the term **multimedia** is often used in the context of the Internet to mean video and audio. Literally, multimedia is just two or more media. That definition makes this book a multimedia presentation, as it contains text and graphics (the figures). However, that is probably not what you had in mind, so we use the term "multimedia" to imply two or more **continuous media**, that is, media that have to be played during some well-defined time interval. The two media are normally video with audio, that is, moving pictures with sound. Many people also refer to pure audio, such as Internet telephony or Internet radio, as multimedia as well, which it is clearly not. Actually, a better term for all these cases is **streaming media**. Nonetheless, we will follow the herd and consider real-time audio to be multimedia as well.

7.4.1 Digital Audio

An audio (sound) wave is a one-dimensional acoustic (pressure) wave. When an acoustic wave enters the ear, the eardrum vibrates, causing the tiny bones of the inner ear to vibrate along with it, sending nerve pulses to the brain. These pulses are perceived as sound by the listener. In a similar way, when an acoustic wave strikes a microphone, the microphone generates an electrical signal, representing the sound amplitude as a function of time.

The frequency range of the human ear runs from 20 Hz to 20,000 Hz. Some animals, notably dogs, can hear higher frequencies. The ear hears loudness logarithmically, so the ratio of two sounds with power A and B is conventionally expressed in **dB** (**decibels**) as the quantity $10 \log_{10}(A/B)$. If we define the lower limit of audibility (a sound pressure of about 20 μPascals) for a 1-kHz sine wave as 0 dB, an ordinary conversation is about 50 dB and the pain threshold is about 120 dB. The dynamic range is a factor of more than 1 million.

The ear is surprisingly sensitive to sound variations lasting only a few milliseconds. The eye, in contrast, does not notice changes in light level that last only a few milliseconds. The result of this observation is that jitter of only a few milliseconds during the playout of multimedia affects the perceived sound quality much more than it affects the perceived image quality.

Digital audio is a digital representation of an audio wave that can be used to recreate it. Audio waves can be converted to digital form by an **ADC** (**Analog-to-Digital Converter**). An ADC takes an electrical voltage as input and generates a binary number as output. In Fig. 7-42(a) we see an example of a sine wave.

To represent this signal digitally, we can sample it every ΔT seconds, as shown by the bar heights in Fig. 7-42(b). If a sound wave is not a pure sine wave but a linear superposition of sine waves where the highest frequency component present is f, the Nyquist theorem (see Chap. 2) states that it is sufficient to make samples at a frequency $2f$. Sampling more often is of no value since the higher frequencies that such sampling could detect are not present.

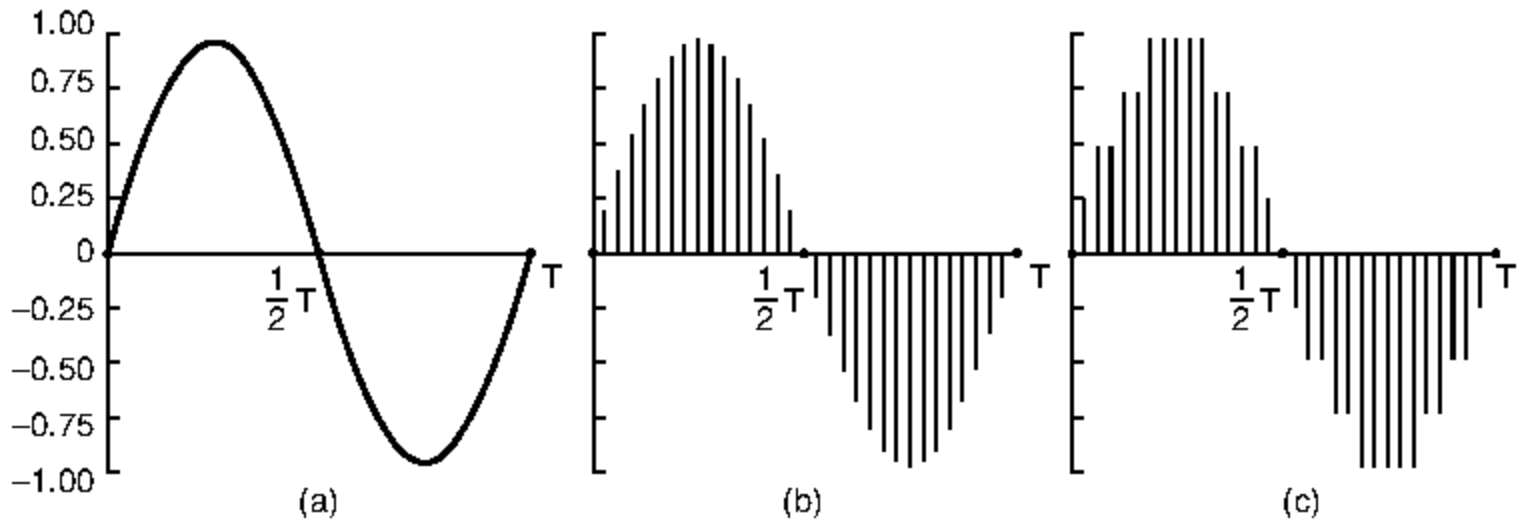


Figure 7-42. (a) A sine wave. (b) Sampling the sine wave. (c) Quantizing the samples to 4 bits.

The reverse process takes digital values and produces an analog electrical voltage. It is done by a **DAC** (**Digital-to-Analog Converter**). A loudspeaker can then convert the analog voltage to acoustic waves so that people can hear sounds.

Digital samples are never exact. The samples of Fig. 7-42(c) allow only nine values, from −1.00 to +1.00 in steps of 0.25. An 8-bit sample would allow 256 distinct values. A 16-bit sample would allow 65,536 distinct values. The error introduced by the finite number of bits per sample is called the **quantization noise**. If it is too large, the ear detects it.

Two well-known examples where sampled sound is used are the telephone and audio compact discs. Pulse code modulation, as used within the telephone system, uses 8-bit samples made 8000 times per second. The scale is nonlinear to minimize perceived distortion, and with only 8000 samples/sec, frequencies above 4 kHz are lost. In North America and Japan, the **μ-law** encoding is used. In Europe and internationally, the **A-law** encoding is used. Each encoding gives a data rate of 64,000 bps.

Audio CDs are digital with a sampling rate of 44,100 samples/sec, enough to capture frequencies up to 22,050 Hz, which is good enough for people but bad for canine music lovers. The samples are 16 bits each and are linear over the range of amplitudes. Note that 16-bit samples allow only 65,536 distinct values, even though the dynamic range of the ear is more than 1 million. Thus, even though CD-quality audio is much better than telephone-quality audio, using only 16 bits per sample introduces noticeable quantization noise (although the full dynamic range is not covered—CDs are not supposed to hurt). Some fanatic audiophiles

still prefer 33-RPM LP records to CDs because records do not have a Nyquist frequency cutoff at 22 kHz and have no quantization noise. (But they do have scratches unless handled very carefully) With 44,100 samples/sec of 16 bits each, uncompressed CD-quality audio needs a bandwidth of 705.6 kbps for monaural and 1.411 Mbps for stereo.

Audio Compression

Audio is often compressed to reduce bandwidth needs and transfer times, even though audio data rates are much lower than video data rates. All compression systems require two algorithms: one for compressing the data at the source, and another for decompressing it at the destination. In the literature, these algorithms are referred to as the **encoding** and **decoding** algorithms, respectively. We will use this terminology too.

Compression algorithms exhibit certain asymmetries that are important to understand. Even though we are considering audio first, these asymmetries hold for video as well. For many applications, a multimedia document will only be encoded once (when it is stored on the multimedia server) but will be decoded thousands of times (when it is played back by customers). This asymmetry means that it is acceptable for the encoding algorithm to be slow and require expensive hardware provided that the decoding algorithm is fast and does not require expensive hardware. The operator of a popular audio (or video) server might be quite willing to buy a cluster of computers to encode its entire library, but requiring customers to do the same to listen to music or watch movies is not likely to be a big success. Many practical compression systems go to great lengths to make decoding fast and simple, even at the price of making encoding slow and complicated.

On the other hand, for live audio and video, such as a voice-over-IP calls, slow encoding is unacceptable. Encoding must happen on the fly, in real time. Consequently, real-time multimedia uses different algorithms or parameters than stored audio or videos on disk, often with appreciably less compression.

A second asymmetry is that the encode/decode process need not be invertible. That is, when compressing a data file, transmitting it, and then decompressing it, the user expects to get the original back, accurate down to the last bit. With multimedia, this requirement does not exist. It is usually acceptable to have the audio (or video) signal after encoding and then decoding be slightly different from the original as long as it sounds (or looks) the same. When the decoded output is not exactly equal to the original input, the system is said to be **lossy**. If the input and output are identical, the system is **lossless**. Lossy systems are important because accepting a small amount of information loss normally means a huge payoff in terms of the compression ratio possible.

Historically, long-haul bandwidth in the telephone network was very expensive, so there is a substantial body of work on **vocoders** (short for "voice coders") that compress audio for the special case of speech. Human speech tends to be in

the 600-Hz to 6000-Hz range and is produced by a mechanical process that depends on the speaker's vocal tract, tongue, and jaw. Some vocoders make use of models of the vocal system to reduce speech to a few parameters (e.g., the sizes and shapes of various cavities) and a data rate of as little as 2.4 kbps. How these vocoders work is beyond the scope of this book, however.

We will concentrate on audio as sent over the Internet, which is typically closer to CD-quality. It is also desirable to reduce the data rates for this kind of audio. At 1.411 Mbps, stereo audio would tie up many broadband links, leaving less room for video and other Web traffic. Its data rate with compression can be reduced by an order of magnitude with little to no perceived loss of quality.

Compression and decompression require signal processing. Fortunately, digitized sound and movies can be easily processed by computers in software. In fact, dozens of programs exist to let users record, display, edit, mix, and store media from multiple sources. This has led to large amounts of music and movies being available on the Internet—not all of it legal—which has resulted in numerous lawsuits from the artists and copyright owners.

Many audio compression algorithms have been developed. Probably the most popular formats are **MP3** (**MPEG audio layer 3**) and **AAC** (**Advanced Audio Coding**) as carried in **MP4** (**MPEG-4**) files. To avoid confusion, note that MPEG provides audio and video compression. MP3 refers to the audio compression portion (part 3) of the MPEG-1 standard, not the third version of MPEG. In fact, no third version of MPEG was released, only MPEG-1, MPEG-2, and MPEG-4. AAC is the successor to MP3 and the default audio encoding used in MPEG-4. MPEG-2 allows both MP3 and AAC audio. Is that clear now? The nice thing about standards is that there are so many to choose from. And if you do not like any of them, just wait a year or two.

Audio compression can be done in two ways. In **waveform coding**, the signal is transformed mathematically by a Fourier transform into its frequency components. In Chap. 2, we showed an example function of time and its Fourier amplitudes in Fig. 2-1(a). The amplitude of each component is then encoded in a minimal way. The goal is to reproduce the waveform fairly accurately at the other end in as few bits as possible.

The other way, **perceptual coding**, exploits certain flaws in the human auditory system to encode a signal in such a way that it sounds the same to a human listener, even if it looks quite different on an oscilloscope. Perceptual coding is based on the science of **psychoacoustics**—how people perceive sound. Both MP3 and AAC are based on perceptual coding.

The key property of perceptual coding is that some sounds can **mask** other sounds. Imagine you are broadcasting a live flute concert on a warm summer day. Then all of a sudden, out of the blue, a crew of workmen nearby turn on their jackhammers and start tearing up the street. No one can hear the flute any more. Its sounds have been masked by the jackhammers. For transmission purposes, it is now sufficient to encode just the frequency band used by the jackhammers

because the listeners cannot hear the flute anyway. This is called **frequency masking**—the ability of a loud sound in one frequency band to hide a softer sound in another frequency band that would have been audible in the absence of the loud sound. In fact, even after the jackhammers stop, the flute will be inaudible for a short period of time because the ear turns down its gain when they start and it takes a finite time to turn it up again. This effect is called **temporal masking**.

To make these effects more quantitative, imagine experiment 1. A person in a quiet room puts on headphones connected to a computer's sound card. The computer generates a pure sine wave at 100 Hz at low, but gradually increasing, power. The subject is instructed to strike a key when she hears the tone. The computer records the current power level and then repeats the experiment at 200 Hz, 300 Hz, and all the other frequencies up to the limit of human hearing. When averaged over many people, a log-log graph of how much power it takes for a tone to be audible looks like that of Fig. 7-43(a). A direct consequence of this curve is that it is never necessary to encode any frequencies whose power falls below the threshold of audibility. For example, if the power at 100 Hz were 20 dB in Fig. 7-43(a), it could be omitted from the output with no perceptible loss of quality because 20 dB at 100 Hz falls below the level of audibility.

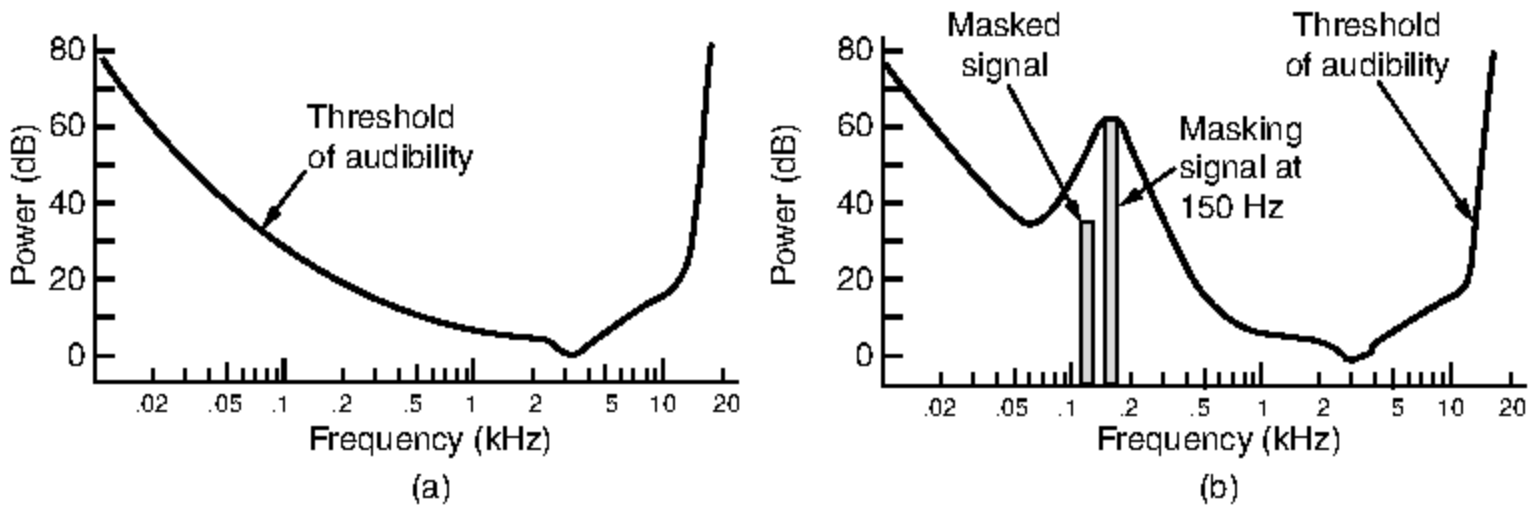


Figure 7-43. (a) The threshold of audibility as a function of frequency. (b) The masking effect.

Now consider experiment 2. The computer runs experiment 1 again, but this time with a constant-amplitude sine wave at, say, 150 Hz superimposed on the test frequency. What we discover is that the threshold of audibility for frequencies near 150 Hz is raised, as shown in Fig. 7-43(b).

The consequence of this new observation is that by keeping track of which signals are being masked by more powerful signals in nearby frequency bands, we can omit more and more frequencies in the encoded signal, saving bits. In Fig. 7-43, the 125-Hz signal can be completely omitted from the output and no one will be able to hear the difference. Even after a powerful signal stops in some frequency band, knowledge of its temporal masking properties allows us to continue to omit the masked frequencies for some time interval as the ear recovers. The

essence of MP3 and AAC is to Fourier-transform the sound to get the power at each frequency and then transmit only the unmasked frequencies, encoding these in as few bits as possible.

With this information as background, we can now see how the encoding is done. The audio compression is done by sampling the waveform at a rate from 8 to 96 kHz for AAC, often at 44.1 kHz, to mimic CD sound. Sampling can be done on one (mono) or two (stereo) channels. Next, the output bit rate is chosen. MP3 can compress a stereo rock 'n roll CD down to 96 kbps with little perceptible loss in quality, even for rock 'n roll fans with no hearing loss. For a piano concert, AAC with at least 128 kbps is needed. The difference is because the signal-to-noise ratio for rock 'n roll is much higher than for a piano concert (in an engineering sense, anyway). It is also possible to choose lower output rates and accept some loss in quality.

The samples are processed in small batches. Each batch is passed through a bank of digital filters to get frequency bands. The frequency information is fed into a psychoacoustic model to determine the masked frequencies. Then the available bit budget is divided among the bands, with more bits allocated to the bands with the most unmasked spectral power, fewer bits allocated to unmasked bands with less spectral power, and no bits allocated to masked bands. Finally, the bits are encoded using Huffman encoding, which assigns short codes to numbers that appear frequently and long codes to those that occur infrequently. There are many more details for the curious reader. For more information, see Brandenburg (1999).

7.4.2 Digital Video

Now that we know all about the ear, it is time to move on to the eye. (No, this section is not followed by one on the nose.) The human eye has the property that when an image appears on the retina, the image is retained for some number of milliseconds before decaying. If a sequence of images is drawn at 50 images/sec, the eye does not notice that it is looking at discrete images. All video systems exploit this principle to produce moving pictures.

The simplest digital representation of video is a sequence of frames, each consisting of a rectangular grid of picture elements, or **pixels**. Each pixel can be a single bit, to represent either black or white. However, the quality of such a system is awful. Try using your favorite image editor to convert the pixels of a color image to black and white (and *not* shades of gray).

The next step up is to use 8 bits per pixel to represent 256 gray levels. This scheme gives high-quality "black-and-white" video. For color video, many systems use 8 bits for each of the red, green and blue (RGB) primary color components. This representation is possible because any color can be constructed from a linear superposition of red, green, and blue with the appropriate intensities. With

24 bits per pixel, there are about 16 million colors, which is more than the human eye can distinguish.

On color LCD computer monitors and televisions, each discrete pixel is made up of closely spaced red, green and blue subpixels. Frames are displayed by setting the intensity of the subpixels, and the eye blends the color components.

Common frame rates are 24 frames/sec (inherited from 35mm motion-picture film), 30 frames/sec (inherited from NTSC U.S. televisions), and 30 frames/sec (inherited from the PAL television system used in nearly all the rest of the world). (For the truly picky, NTSC color television runs at 29.97 frames/sec. The original black-and-white system ran at 30 frames/sec, but when color was introduced, the engineers needed a bit of extra bandwidth for signaling so they reduced the frame rate to 29.97. NTSC videos intended for computers really use 30.) PAL was invented after NTSC and really uses 25.000 frames/sec. To make this story complete, a third system, SECAM, is used in France, Francophone Africa, and Eastern Europe. It was first introduced into Eastern Europe by then Communist East Germany so the East German people could not watch West German (PAL) television lest they get Bad Ideas. But many of these countries are switching to PAL. Technology and politics at their best.

Actually, for broadcast television, 25 frames/sec is not quite good enough for smooth motion so the images are split into two **fields**, one with the odd-numbered scan lines and one with the even-numbered scan lines. The two (half-resolution) fields are broadcast sequentially, giving almost 60 (NTSC) or exactly 50 (PAL) fields/sec, a system known as **interlacing**. Videos intended for viewing on a computer are **progressive**, that is, do not use interlacing because computer monitors have buffers on their graphics cards, making it possible for the CPU to put a new image in the buffer 30 times/sec but have the graphics card redraw the screen 50 or even 100 times/sec to eliminate flicker. Analog television sets do not have a frame buffer the way computers do. When an interlaced video with rapid movement is displayed on a computer, short horizontal lines will be visible near sharp edges, an effect known as **combing**.

The frame sizes used for video sent over the Internet vary widely for the simple reason that larger frames require more bandwidth, which may not always be available. Low-resolution video might be 320 by 240 pixels, and "full-screen" video is 640 by 480 pixels. These dimensions approximate those of early computer monitors and NTSC television, respectively. The **aspect ratio**, or width to height ratio, of 4:3, is the same as a standard television. **HDTV** (**High-Definition TeleVision**) videos can be downloaded with 1280 by 720 pixels. These "widescreen" images have an aspect ratio of 16:9 to more closely match the 3:2 aspect ratio of film. For comparison, standard DVD video is usually 720 by 480 pixels, and video on Blu-ray discs is usually HDTV at 1080 by 720 pixels.

On the Internet, the number of pixels is only part of the story, as media players can present the same image at different sizes. Video is just another window on a computer screen that can be blown up or shrunk down. The role of more

pixels is to increase the quality of the image, so that it does not look blurry when it is expanded. However, many monitors can show images (and hence videos) with even more pixels than even HDTV.

Video Compression

It should be obvious from our discussion of digital video that compression is critical for sending video over the Internet. Even a standard-quality video with 640 by 480 pixel frames, 24 bits of color information per pixel, and 30 frames/sec takes over 200 Mbps. This far exceeds the bandwidth by which most company offices are connected to the Internet, let alone home users, and this is for a single video stream. Since transmitting uncompressed video is completely out of the question, at least over wide area networks, the only hope is that massive compression is possible. Fortunately, a large body of research over the past few decades has led to many compression techniques and algorithms that make video transmission feasible.

Many formats are used for video that is sent over the Internet, some proprietary and some standard. The most popular encoding is MPEG in its various forms. It is an open standard found in files with mpg and mp4 extensions, as well as in other container formats. In this section, we will look at MPEG to study how video compression is accomplished. To begin, we will look at the compression of still images with JPEG. A video is just a sequence of images (plus sound). One way to compress video is to encode each image in succession. To a first approximation, MPEG is just the JPEG encoding of each frame, plus some extra features for removing the redundancy across frames.

The JPEG Standard

The **JPEG** (**Joint Photographic Experts Group**) standard for compressing continuous-tone still pictures (e.g., photographs) was developed by photographic experts working under the joint auspices of ITU, ISO, and IEC, another standards body. It is widely used (look for files with the extension jpg) and often provides compression ratios of 10:1 or better for natural images.

JPEG is defined in International Standard 10918. Really, it is more like a shopping list than a single algorithm, but of the four modes that are defined only the lossy sequential mode is relevant to our discussion. Furthermore, we will concentrate on the way JPEG is normally used to encode 24-bit RGB video images and will leave out some of the options and details for the sake of simplicity.

The algorithm is illustrated in Fig. 7-44. Step 1 is block preparation. For the sake of specificity, let us assume that the JPEG input is a 640×480 RGB image with 24 bits/pixel, as shown in Fig. 7-44(a). RGB is not the best color model to use for compression. The eye is much more sensitive to the **luminance**, or brightness, of video signals than the **chrominance**, or color, of video signals. Thus, we

first compute the luminance, Y, and the two chrominances, Cb and Cr, from the R, G, and B components. The following formulas are used for 8-bit values that range from 0 to 255:

$$\begin{aligned} Y &= 16 + 0.26R + 0.50G + 0.09B \\ Cb &= 128 + 0.15R - 0.29G - 0.44B \\ Cr &= 128 + 0.44R - 0.37G + 0.07B \end{aligned}$$

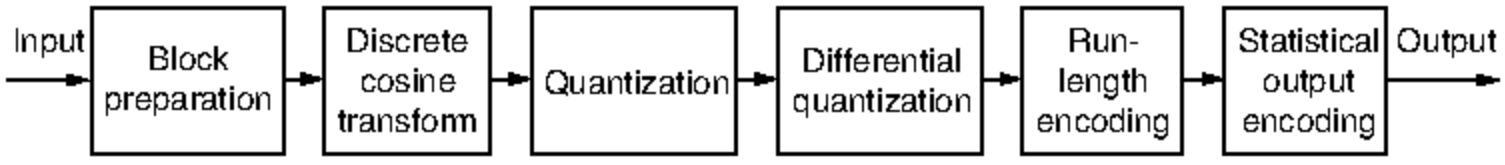


Figure 7-44. Steps in JPEG lossy sequential encoding.

Separate matrices are constructed for Y, Cb, and Cr. Next, square blocks of four pixels are averaged in the Cb and Cr matrices to reduce them to 320×240. This reduction is lossy, but the eye barely notices it since the eye responds to luminance more than to chrominance. Nevertheless, it compresses the total amount of data by a factor of two. Now 128 is subtracted from each element of all three matrices to put 0 in the middle of the range. Finally, each matrix is divided up into 8×8 blocks. The Y matrix has 4800 blocks; the other two have 1200 blocks each, as shown in Fig. 7-45(b).

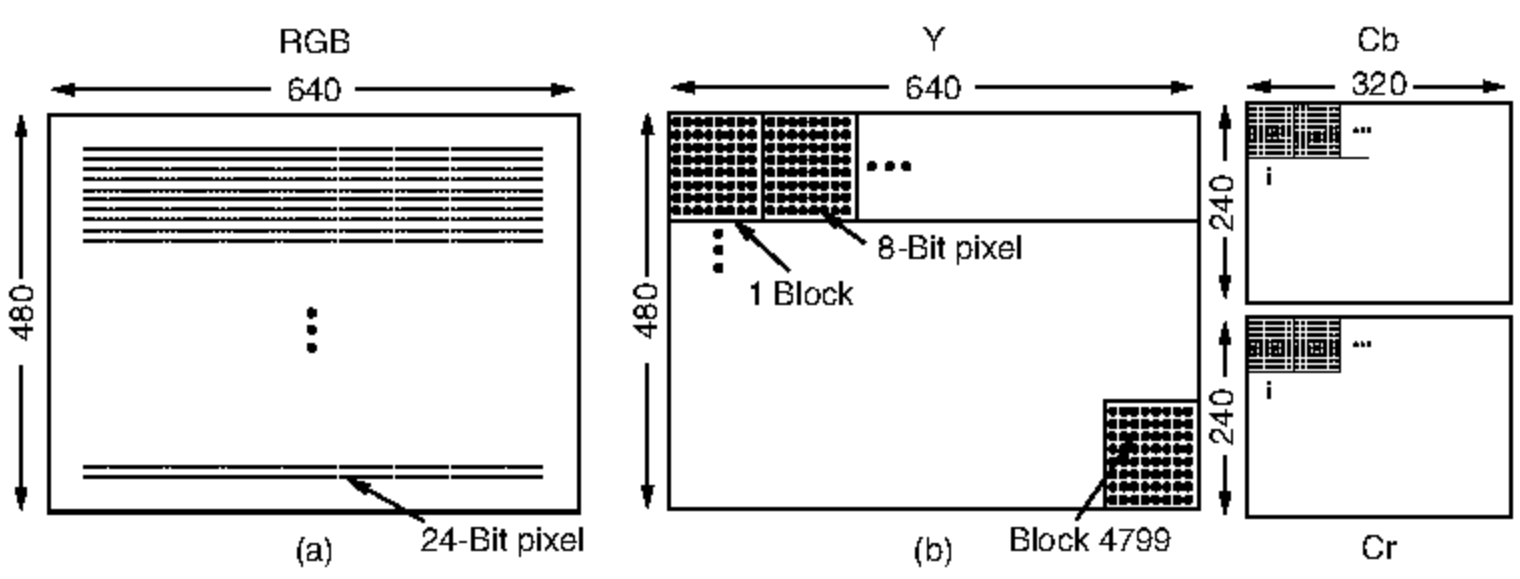


Figure 7-45. (a) RGB input data. (b) After block preparation.

Step 2 of JPEG encoding is to apply a **DCT** (**Discrete Cosine Transformation**) to each of the 7200 blocks separately. The output of each DCT is an 8×8 matrix of DCT coefficients. DCT element (0, 0) is the average value of the block. The other elements tell how much spectral power is present at each spatial frequency. Normally, these elements decay rapidly with distance from the origin, (0, 0), as suggested by Fig. 7-46.

Once the DCT is complete, JPEG encoding moves on to step 3, called **quantization**, in which the less important DCT coefficients are wiped out. This (lossy)

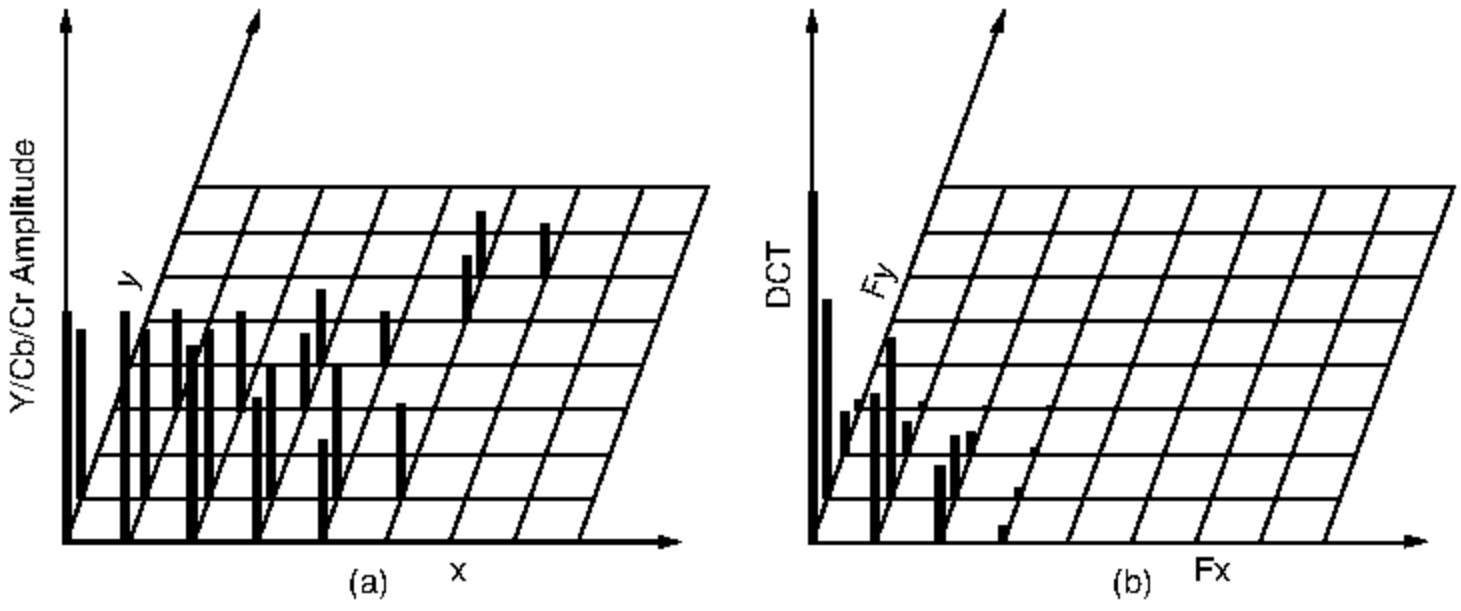


Figure 7-46. (a) One block of the Y matrix. (b) The DCT coefficients.

transformation is done by dividing each of the coefficients in the 8×8 DCT matrix by a weight taken from a table. If all the weights are 1, the transformation does nothing. However, if the weights increase sharply from the origin, higher spatial frequencies are dropped quickly.

An example of this step is given in Fig. 7-47. Here we see the initial DCT matrix, the quantization table, and the result obtained by dividing each DCT element by the corresponding quantization table element. The values in the quantization table are not part of the JPEG standard. Each application must supply its own, allowing it to control the loss-compression trade-off.

DCT coefficients

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	38	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantization table

1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

Quantized coefficients

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 7-47. Computation of the quantized DCT coefficients.

Step 4 reduces the (0, 0) value of each block (the one in the upper-left corner) by replacing it with the amount it differs from the corresponding element in the previous block. Since these elements are the averages of their respective blocks, they should change slowly, so taking the differential values should reduce most of them to small values. No differentials are computed from the other values.

Step 5 linearizes the 64 elements and applies run-length encoding to the list. Scanning the block from left to right and then top to bottom will not concentrate the zeros together, so a zigzag scanning pattern is used, as shown in Fig. 7-48. In this example, the zigzag pattern produces 38 consecutive 0s at the end of the matrix. This string can be reduced to a single count saying there are 38 zeros, a technique known as **run-length encoding**.

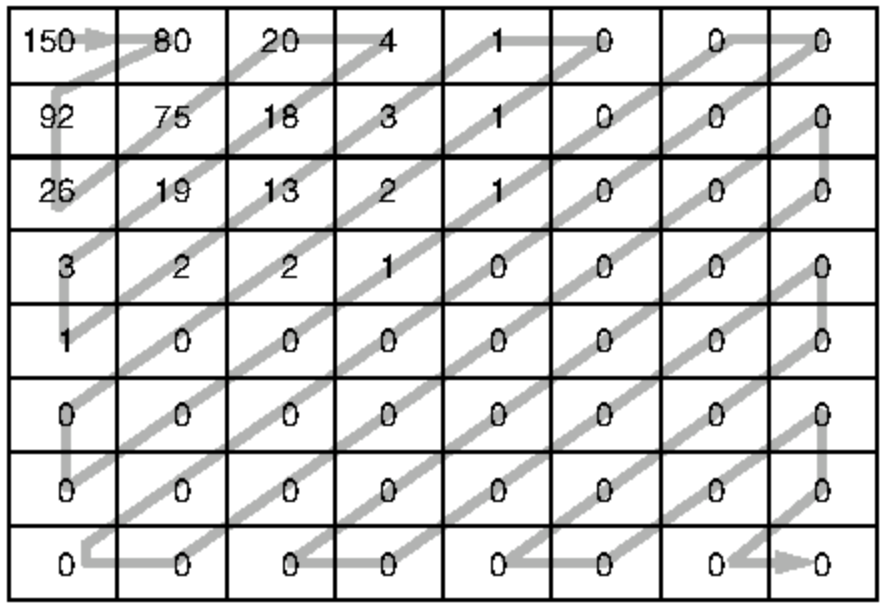


Figure 7-48. The order in which the quantized values are transmitted.

Now we have a list of numbers that represent the image (in transform space). Step 6 Huffman-encodes the numbers for storage or transmission, assigning common numbers shorter codes than uncommon ones.

JPEG may seem complicated, but that is because it *is* complicated. Still, the benefits of up to 20:1 compression are worth it. Decoding a JPEG image requires running the algorithm backward. JPEG is roughly symmetric: decoding takes as long as encoding. This property is not true of all compression algorithms, as we shall now see.

The MPEG Standard

Finally, we come to the heart of the matter: the **MPEG** (**Motion Picture Experts Group**) standards. Though there are many proprietary algorithms, these standards define the main algorithms used to compress videos. They have been international standards since 1993. Because movies contain both images and sound, MPEG can compress both audio and video. We have already examined audio compression and still image compression, so let us now examine video compression.

The MPEG-1 standard (which includes MP3 audio) was first published in 1993 and is still widely used. Its goal was to produce video-recorder-quality output that was compressed 40:1 to rates of around 1 Mbps. This video is suitable for

broad Internet use on Web sites. Do not worry if you do not remember video recorders—MPEG-1 was also used for storing movies on CDs when they existed. If you do not know what CDs are, we will have to move on to MPEG-2.

The MPEG-2 standard, released in 1996, was designed for compressing broadcast-quality video. It is very common now, as it is used as the basis for video encoded on DVDs (which inevitably finds its way onto the Internet) and for digital broadcast television (as DVB). DVD quality video is typically encoded at rates of 4–8 Mbps.

The MPEG-4 standard has two video formats. The first format, released in 1999, encodes video with an object-based representation. This allows for the mixing of natural and synthetic images and other kinds of media, for example, a weatherperson standing in front of a weather map. With this structure, it is easy to let programs interact with movie data. The second format, released in 2003, is known as **H.264** or **AVC** (**Advanced Video Coding**). Its goal is to encode video at half the rate of earlier encoders for the same quality level, all the better to support the transmission of video over networks. This encoder is used for HDTV on most Blu-ray discs.

The details of all these standards are many and varied. The later standards also have many more features and encoding options than the earlier standards. However, we will not go into the details. For the most part, the gains in video compression over time have come from numerous small improvements, rather than fundamental shifts in how video is compressed. Thus, we will sketch the overall concepts.

MPEG compresses both audio and video. Since the audio and video encoders work independently, there is an issue of how the two streams get synchronized at the receiver. The solution is to have a single clock that outputs timestamps of the current time to both encoders. These timestamps are included in the encoded output and propagated all the way to the receiver, which can use them to synchronize the audio and video streams.

MPEG video compression takes advantage of two kinds of redundancies that exist in movies: spatial and temporal. Spatial redundancy can be utilized by simply coding each frame separately with JPEG. This approach is occasionally used, especially when random access to each frame is needed, as in editing video productions. In this mode, JPEG levels of compression are achieved.

Additional compression can be achieved by taking advantage of the fact that consecutive frames are often almost identical. This effect is smaller than it might first appear since many movie directors cut between scenes every 3 or 4 seconds (time a movie fragment and count the scenes). Nevertheless, runs of 75 or more highly similar frames offer the potential of a major reduction over simply encoding each frame separately with JPEG.

For scenes in which the camera and background are stationary and one or two actors are moving around slowly, nearly all the pixels will be identical from frame to frame. Here, just subtracting each frame from the previous one and running

JPEG on the difference would do fine. However, for scenes where the camera is panning or zooming, this technique fails badly. What is needed is some way to compensate for this motion. This is precisely what MPEG does; it is the main difference between MPEG and JPEG.

MPEG output consists of three kinds of frames:

1. I- (Intracoded) frames: self-contained compressed still pictures.
2. P- (Predictive) frames: block-by-block difference with the previous frames.
3. B- (Bidirectional) frames: block-by-block differences between previous and future frames.

I-frames are just still pictures. They can be coded with JPEG or something similar. It is valuable to have I-frames appear in the output stream periodically (e.g., once or twice per second) for three reasons. First, MPEG can be used for a multicast transmission, with viewers tuning in at will. If all frames depended on their predecessors going back to the first frame, anybody who missed the first frame could never decode any subsequent frames. Second, if any frame were received in error, no further decoding would be possible: everything from then on would be unintelligble junk. Third, without I-frames, while doing a fast forward or rewind the decoder would have to calculate every frame passed over so it would know the full value of the one it stopped on.

P-frames, in contrast, code interframe differences. They are based on the idea of **macroblocks**, which cover, for example, 16×16 pixels in luminance space and 8×8 pixels in chrominance space. A macroblock is encoded by searching the previous frame for it or something only slightly different from it.

An example of where P-frames would be useful is given in Fig. 7-49. Here we see three consecutive frames that have the same background, but differ in the position of one person. The macroblocks containing the background scene will match exactly, but the macroblocks containing the person will be offset in position by some unknown amount and will have to be tracked down.

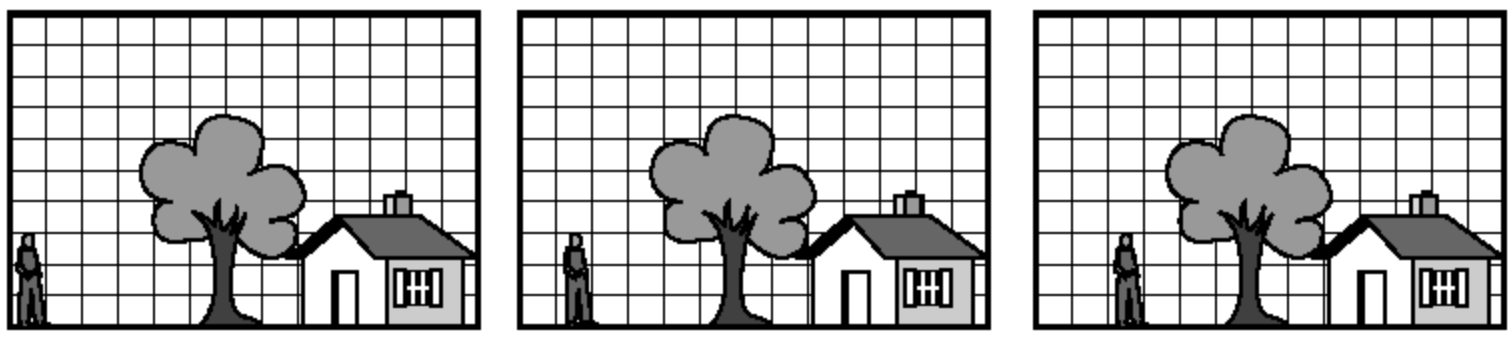


Figure 7-49. Three consecutive frames.

The MPEG standards do not specify how to search, how far to search, or how good a match has to be in order to count. This is up to each implementation. For

example, an implementation might search for a macroblock at the current position in the previous frame, and all other positions offset $\pm\Delta x$ in the x direction and $\pm\Delta y$ in the y direction. For each position, the number of matches in the luminance matrix could be computed. The position with the highest score would be declared the winner, provided it was above some predefined threshold. Otherwise, the macroblock would be said to be missing. Much more sophisticated algorithms are also possible, of course.

If a macroblock is found, it is encoded by taking the difference between its current value and the one in the previous frame (for luminance and both chrominances). These difference matrices are then subjected to the discrete cosine transformation, quantization, run-length encoding, and Huffman encoding, as usual. The value for the macroblock in the output stream is then the motion vector (how far the macroblock moved from its previous position in each direction), followed by the encoding of its difference. If the macroblock is not located in the previous frame, the current value is encoded, just as in an I-frame.

Clearly, this algorithm is highly asymmetric. An implementation is free to try every plausible position in the previous frame if it wants to, in a desperate attempt to locate every last macroblock, no matter where it has moved to. This approach will minimize the encoded MPEG stream at the expense of very slow encoding. This approach might be fine for a one-time encoding of a film library but would be terrible for real-time videoconferencing.

Similarly, each implementation is free to decide what constitutes a "found" macroblock. This freedom allows implementers to compete on the quality and speed of their algorithms, but always produce compliant MPEG output.

So far, decoding MPEG is straightforward. Decoding I-frames is similar to decoding JPEG images. Decoding P-frames requires the decoder to buffer the previous frames so it can build up the new one in a separate buffer based on fully encoded macroblocks and macroblocks containing differences from the previous frames. The new frame is assembled macroblock by macroblock.

B-frames are similar to P-frames, except that they allow the reference macroblock to be in either previous frames or succeeding frames. This additional freedom allows for improved motion compensation. It is useful, for example, when objects pass in front of, or behind, other objects. To do B-frame encoding, the encoder needs to hold a sequence of frames in memory at once: past frames, the current frame being encoded, and future frames. Decoding is similarly more complicated and adds some delay. This is because a given B-frame cannot be decoded until the successive frames on which it depends are decoded. Thus, although B-frames give the best compression, they are not always used due to their greater complexity and buffering requirements.

The MPEG standards contain many enhancements to these techniques to achieve excellent levels of compression. AVC can be used to compress video at ratios in excess of 50:1, which reduces network bandwidth requirements by the same factor. For more information on AVC, see Sullivan and Wiegand (2005).

7.4.3 Streaming Stored Media

Let us now move on to network applications. Our first case is streaming media that is already stored in files. The most common example of this is watching videos over the Internet. This is one form of **VoD** (**Video on Demand**). Other forms of video on demand use a provider network that is separate from the Internet to deliver the movies (e.g., the cable network).

In the next section, we will look at streaming live media, for example, broadcast IPTV and Internet radio. Then we will look at the third case of real-time conferencing. An example is a voice-over-IP call or video conference with Skype. These three cases place increasingly stringent requirements on how we can deliver the audio and video over the network because we must pay increasing attention to delay and jitter.

The Internet is full of music and video sites that stream stored media files. Actually, the easiest way to handle stored media is *not* to stream it. Imagine you want to create an online movie rental site to compete with Apple's iTunes. A regular Web site will let users download and then watch videos (after they pay, of course). The sequence of steps is shown in Fig. 7-50. We will spell them out to contrast them with the next example.

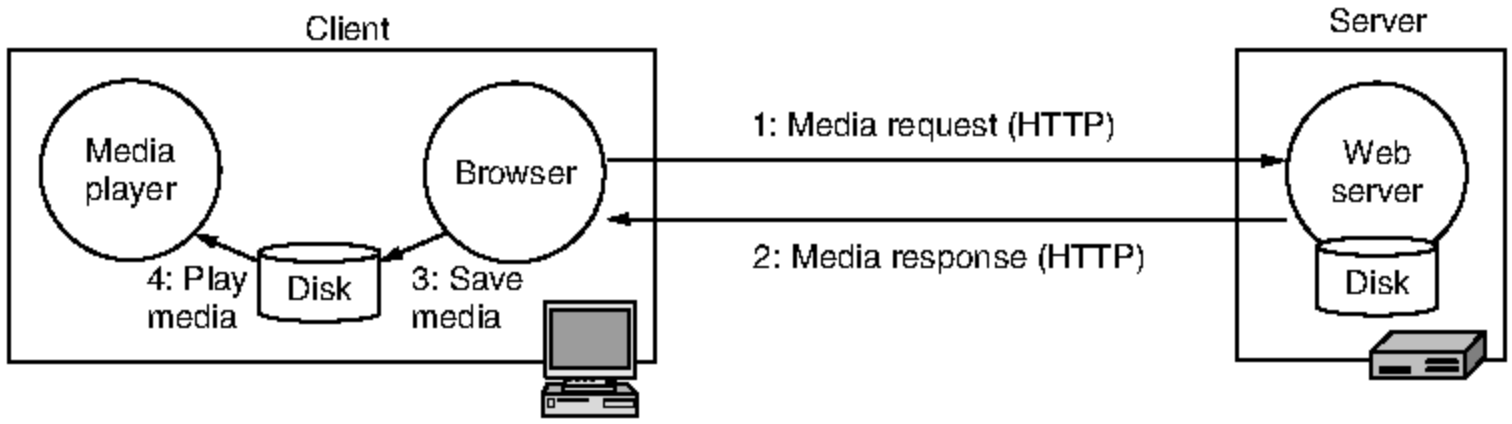


Figure 7-50. Playing media over the Web via simple downloads.

The browser goes into action when the user clicks on a movie. In step 1, it sends an HTTP request for the movie to the Web server to which the movie is linked. In step 2, the server fetches the movie (which is just a file in MP4 or some other format) and sends it back to the browser. Using the MIME type, for example, *video/mp4*, the browser looks up how it is supposed to display the file. In this case, it is with a media player that is shown as a helper application, though it could also be a plug-in. The browser saves the entire movie to a scratch file on disk in step 3. It then starts the media player, passing it the name of the scratch file. Finally, in step 4 the media player starts reading the file and playing the movie.

In principle, this approach is completely correct. It will play the movie. There is no real-time network issue to address either because the download is simply a

file download. The only trouble is that the entire video must be transmitted over the network before the movie starts. Most customers do not want to wait an hour for their "video on demand." This model can be problematic even for audio. Imagine previewing a song before purchasing an album. If the song is 4 MB, which is a typical size for an MP3 song, and the broadband connectivity is 1 Mbps, the user will be greeted by half a minute of silence before the preview starts. This model is unlikely to sell many albums.

To get around this problem without changing how the browser works, sites can use the design shown in Fig. 7-51. The page linked to the movie is not the actual movie file. Instead, it is what is called a **metafile**, a very short file just naming the movie (and possibly having other key descriptors). A simple metafile might be only one line of ASCII text and look like this:

```
rtsp://joes-movie-server/movie-0025.mp4
```

The browser gets the page as usual, now a one-line file, in steps 1 and 2. Then it starts the media player and hands it the one-line file in step 3, all as usual. The media player reads the metafile and sees the URL of where to get the movie. It contacts *joes-video-server* and asks for the movie in step 4. The movie is then streamed back to the media player in step 5. The advantage of this arrangement is that the media player starts quickly, after only a very short metafile is downloaded. Once this happens, the browser is not in the loop any more. The media is sent directly to the media player, which can start showing the movie before the entire file has been downloaded.

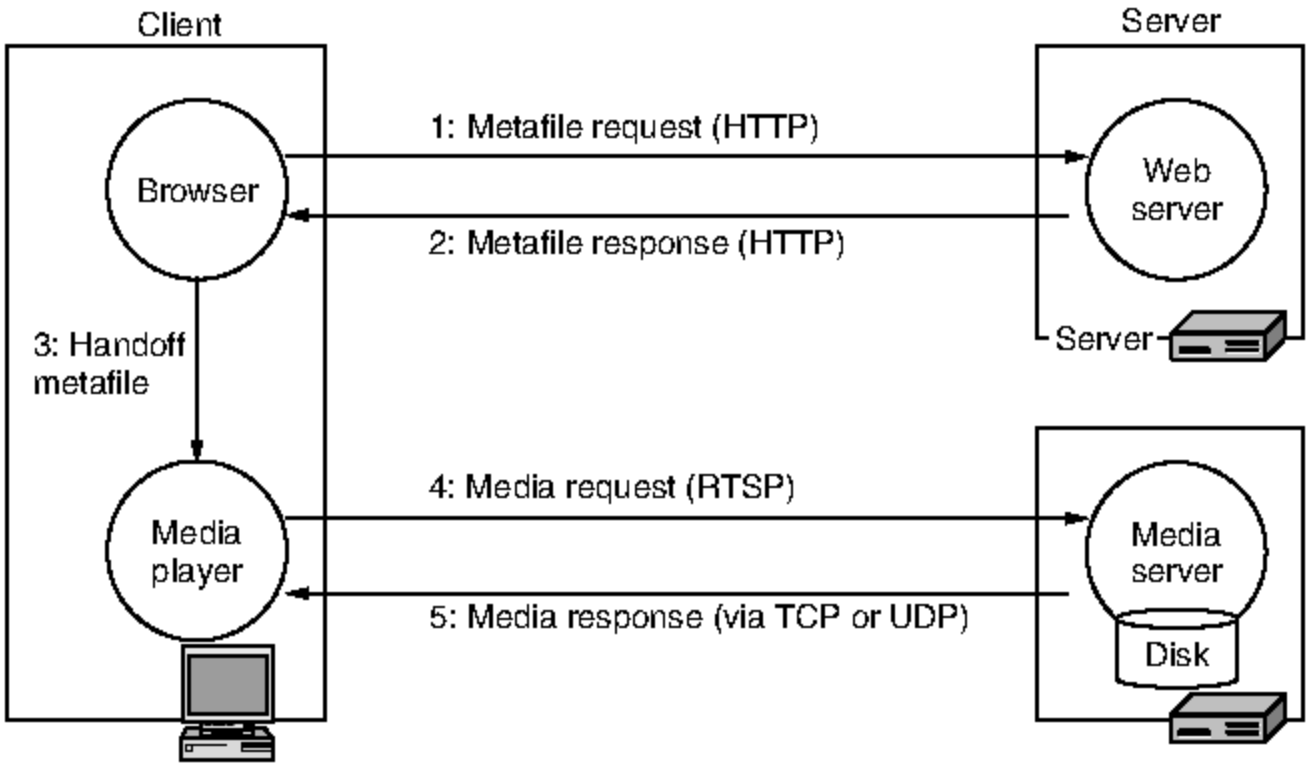


Figure 7-51. Streaming media using the Web and a media server.

We have shown two servers in Fig. 7-51 because the server named in the metafile is often not the same as the Web server. In fact, it is generally not even

an HTTP server, but a specialized media server. In this example, the media server uses **RTSP** (**Real Time Streaming Protocol**), as indicated by the scheme name *rtsp*.

The media player has four major jobs to do:

1. Manage the user interface.
2. Handle transmission errors.
3. Decompress the content.
4. Eliminate jitter.

Most media players nowadays have a glitzy user interface, sometimes simulating a stereo unit, with buttons, knobs, sliders, and visual displays. Often there are interchangeable front panels, called **skins**, that the user can drop onto the player. The media player has to manage all this and interact with the user.

The other jobs are related and depend on the network protocols. We will go through each one in turn, starting with handling transmission errors. Dealing with errors depends on whether a TCP-based transport like HTTP is used to transport the media, or a UDP-based transport like RTP is used. Both are used in practice. If a TCP-based transport is being used then there are no errors for the media player to correct because TCP already provides reliability by using retransmissions. This is an easy way to handle errors, at least for the media player, but it does complicate the removal of jitter in a later step.

Alternatively, a UDP-based transport like RTP can be used to move the data. We studied it in Chap. 6. With these protocols, there are no retransmissions. Thus, packet loss due to congestion or transmission errors will mean that some of the media does not arrive. It is up to the media player to deal with this problem.

Let us understand the difficulty we are up against. The loss is a problem because customers do not like large gaps in their songs or movies. However, it is not as much of a problem as loss in a regular file transfer because the loss of a small amount of media need not degrade the presentation for the user. For video, the user is unlikely to notice if there are occasionally 24 new frames in some second instead of 25 new frames. For audio, short gaps in the playout can be masked with sounds close in time. The user is unlikely to detect this substitution unless they are paying *very* close attention.

The key to the above reasoning, however, is that the gaps are very short. Network congestion or a transmission error will generally cause an entire packet to be lost, and packets are often lost in small bursts. Two strategies can be used to reduce the impact of packet loss on the media that is lost: FEC and interleaving. We will describe each in turn.

FEC (**Forward Error Correction**) is simply the error-correcting coding that we studied in Chap. 3 applied at the application level. Parity across packets provides an example (Shacham and McKenny, 1990). For every four data packets

that are sent, a fifth **parity packet** can be constructed and sent. This is shown in Fig. 7-52 with packets *A*, *B*, *C*, and *D*. The parity packet, *P*, contains redundant bits that are the parity or exclusive-OR sums of the bits in each of the four data packets. Hopefully, all of the packets will arrive for most groups of five packets. When this happens, the parity packet is simply discarded at the receiver. Or, if only the parity packet is lost, no harm is done.

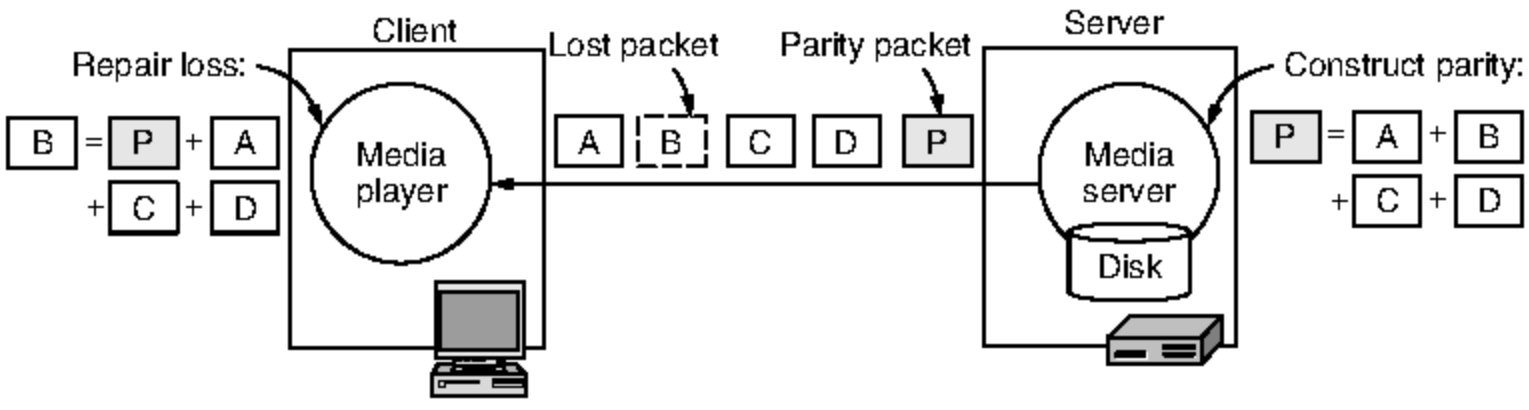


Figure 7-52. Using a parity packet to repair loss.

Occasionally, however, a data packet may be lost during transmission, as *B* is in Fig. 7-52. The media player receives only three data packets, *A*, *C*, and *D*, plus the parity packet, *P*. By design, the bits in the missing data packet can be reconstructed from the parity bits. To be specific, using "+" to represent exclusive-OR or modulo 2 addition, *B* can be reconstructed as $B = P + A + C + D$ by the properties of exclusive-OR (i.e., $X + Y + Y = X$).

FEC can reduce the level of loss seen by the media player by repairing some of the packet losses, but it only works up to a certain level. If two packets in a group of five are lost, there is nothing we can do to recover the data. The other property to note about FEC is the cost that we have paid to gain this protection. Every four packets have become five packets, so the bandwidth requirements for the media are 25% larger. The latency of decoding has increased too, as we may need to wait until the parity packet has arrived before we can reconstruct a data packet that came before it.

There is also one clever trick in the technique above. In Chap. 3, we described parity as providing error detection. Here we are providing error-correction. How can it do both? The answer is that in this case it is known which packet was lost. The lost data is called an **erasure**. In Chap. 3, when we considered a frame that was received with some bits in error, we did not know which bit was errored. This case is harder to deal with than erasures. Thus, with erasures parity can provide error correction, and without erasures parity can only provide error detection. We will see another unexpected benefit of parity soon, when we get to multicast scenarios.

The second strategy is called **interleaving**. This approach is based on mixing up or interleaving the order of the media before transmission and unmixing or

deinterleaving it on reception. That way, if a packet (or burst of packets) is lost, the loss will be spread out over time by the unmixing. It will not result in a single, large gap when the media is played out. For example, a packet might contain 220 stereo samples, each containing a pair of 16-bit numbers, normally good for 5 msec of music. If the samples were sent in order, a lost packet would represent a 5 msec gap in the music. Instead, the samples are transmitted as shown in Fig. 7-53. All the even samples for a 10-msec interval are sent in one packet, followed by all the odd samples in the next one. The loss of packet 3 now does not represent a 5-msec gap in the music, but the loss of every other sample for 10 msec. This loss can be handled easily by having the media player interpolate using the previous and succeeding samples. The result is lower temporal resolution for 10 msec, but not a noticeable time gap in the media.

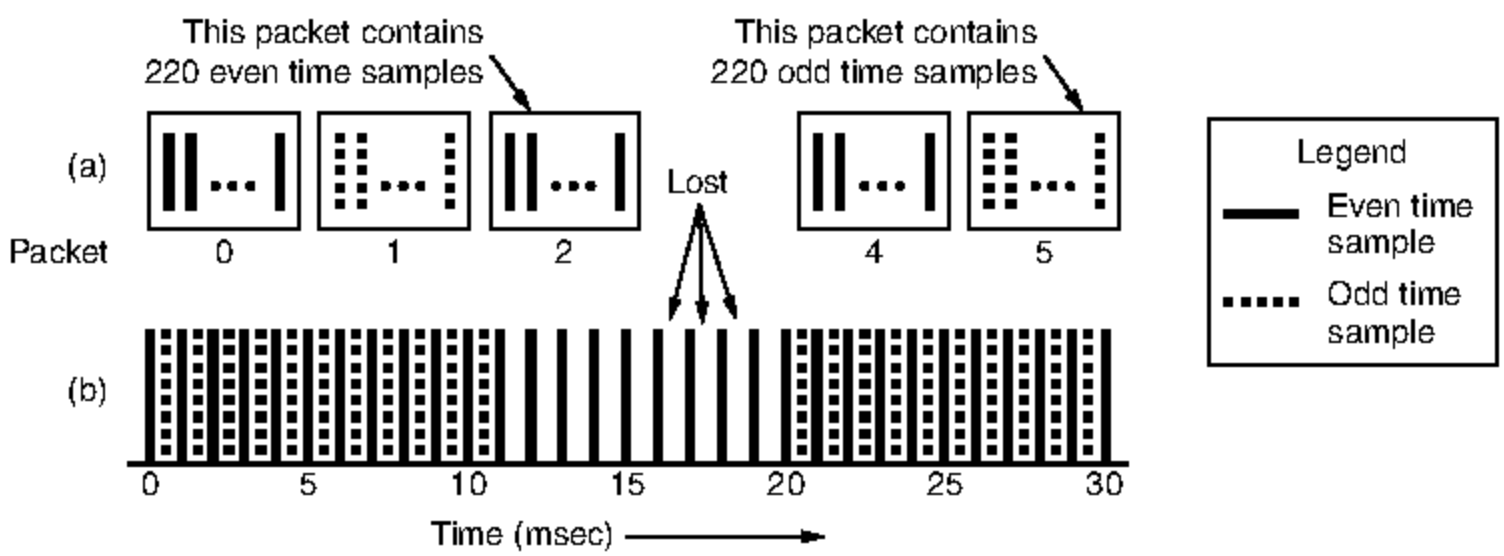


Figure 7-53. When packets carry alternate samples, the loss of a packet reduces the temporal resolution rather than creating a gap in time.

This interleaving scheme above only works with uncompressed sampling. However, interleaving (over short periods of time, not individual samples) can also be applied after compression as long as there is a way to find sample boundaries in the compressed stream. RFC 3119 gives a scheme that works with compressed audio.

Interleaving is an attractive technique when it can be used because it needs no additional bandwidth, unlike FEC. However, interleaving adds to the latency, just like FEC, because of the need to wait for a group of packets to arrive (so they can be de-interleaved).

The media player's third job is decompressing the content. Although this task is computationally intensive, it is fairly straightforward. The thorny issue is how to decode media if the network protocol does not correct transmission errors. In many compression schemes, later data cannot be decompressed until the earlier data has been decompressed, because the later data is encoded relative to the earlier data. For a UDP-based transport, there can be packet loss. Thus, the encoding

process must be designed to permit decoding despite packet loss. This requirement is why MPEG uses I-, P- and B-frames. Each I-frame can be decoded independently of the other frames to recover from the loss of any earlier frames.

The fourth job is to eliminate jitter, the bane of all real-time systems. The general solution that we described in Sec. 6.4.3 is to use a playout buffer. All streaming systems start by buffering 5–10 sec worth of media before starting to play, as shown in Fig. 7-54. Playing drains media regularly from the buffer so that the audio is clear and the video is smooth. The startup delay gives the buffer a chance to fill to the **low-water mark**. The idea is that data should now arrive regularly enough that the buffer is never completely emptied. If that were to happen, the media playout would stall. The value of buffering is that if the data are sometimes slow to arrive due to congestion, the buffered media will allow the playout to continue normally until new media arrive and the buffer is replenished.

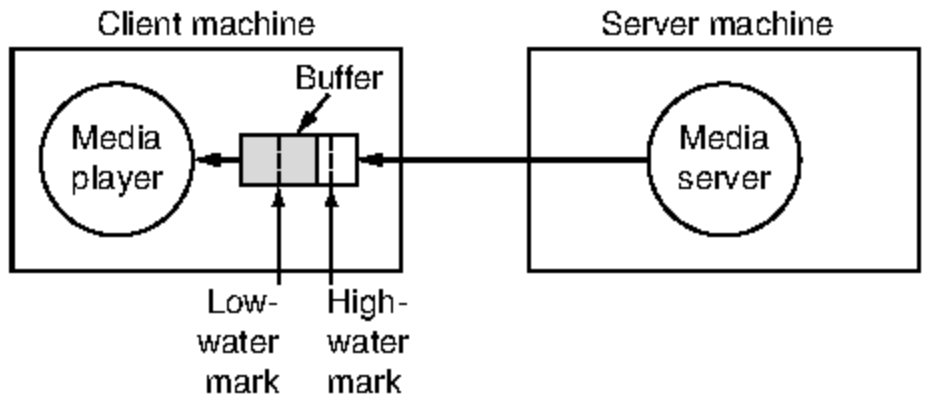


Figure 7-54. The media player buffers input from the media server and plays from the buffer rather than directly from the network.

How much buffering is needed, and how fast the media server sends media to fill up the buffer, depend on the network protocols. There are many possibilities. The largest factor in the design is whether a UDP-based transport or a TCP-based transport is used.

Suppose that a UDP-based transport like RTP is used. Further suppose that there is ample bandwidth to send packets from the media server to the media player with little loss, and little other traffic in the network. In this case, packets can be sent at the exact rate that the media is being played. Each packet will transit the network and, after a propagation delay, arrive at about the right time for the media player to present the media. Very little buffering is needed, as there is no variability in delay. If interleaving or FEC is used, more buffering is needed for at least the group of packets over which the interleaving or FEC is performed. However, this adds only a small amount of buffering.

Unfortunately, this scenario is unrealistic in two respects. First, bandwidth varies over network paths, so it is usually not clear to the media server whether there will be sufficient bandwidth before it tries to stream the media. A simple solution is to encode media at multiple resolutions and let each user choose a

resolution that is supported by his Internet connectivity. Often there are just two levels: high quality, say, encoded at 1.5 Mbps or better, and low quality, say encoded at 512 kbps or less.

Second, there will be some jitter, or variation in how long it takes media samples to cross the network. This jitter comes from two sources. There is often an appreciable amount of competing traffic in the network—some of which can come from multitasking users themselves browsing the Web while ostensibly watching a streamed movie). This traffic will cause fluctuations in when the media arrives. Moreover, we care about the arrival of video frames and audio samples, not packets. With compression, video frames in particular may be larger or smaller depending on their content. An action sequence will typically take more bits to encode than a placid landscape. If the network bandwidth is constant, the rate of media delivery versus time will vary. The more jitter, or variation in delay, from these sources, the larger the low-water mark of the buffer needs to be to avoid underrun.

Now suppose that a TCP-based transport like HTTP is used to send the media. By performing retransmissions and waiting to deliver packets until they are in order, TCP will increase the jitter that is observed by the media player, perhaps significantly. The result is that a larger buffer and higher low-water mark are needed. However, there is an advantage. TCP will send data as fast as the network will carry it. Sometimes media may be delayed if loss must be repaired. But much of the time, the network will be able to deliver media faster than the player consumes it. In these periods, the buffer will fill and prevent future underruns. If the network is significantly faster than the average media rate, as is often the case, the buffer will fill rapidly after startup such that emptying it will soon cease to be a concern.

With TCP, or with UDP and a transmission rate that exceeds the playout rate, a question is how far ahead of the playout point the media player and media server are willing to proceed. Often they are willing to download the entire file.

However, proceeding far ahead of the playout point performs work that is not yet needed, may require significant storage, and is not necessary to avoid buffer underruns. When it is not wanted, the solution is for the media player to define a **high-water mark** in the buffer. Basically, the server just pumps out data until the buffer is filled to the high-water mark. Then the media player tells it to pause. Since data will continue to pour in until the server has gotten the pause request, the distance between the high-water mark and the end of the buffer has to be greater than the bandwidth-delay product of the network. After the server has stopped, the buffer will begin to empty. When it hits the low-water mark, the media player tells the media server to start again. To avoid underrun, the low-water mark must also take the bandwidth-delay product of the network into account when asking the media server to resume sending the media.

To start and stop the flow of media, the media player needs a remote control for it. This is what RTSP provides. It is defined in RFC 2326 and provides the

mechanism for the player to control the server. As well as starting and stopping the stream, it can seek back or forward to a position, play specified intervals, and play at fast or slow speeds. It does not provide for the data stream, though, which is usually RTP over UDP or RTP over HTTP over TCP.

The main commands provided by RTSP are listed in Fig. 7-55. They have a simple text format, like HTTP messages, and are usually carried over TCP. RTSP can run over UDP too, since each command is acknowledged (and so can be resent if it is not acknowledged).

Command	Server action
DESCRIBE	List media parameters
SETUP	Establish a logical channel between the player and the server
PLAY	Start sending data to the client
RECORD	Start accepting data from the client
PAUSE	Temporarily stop sending data
TEARDOWN	Release the logical channel

Figure 7-55. RTSP commands from the player to the server.

Even though TCP would seem a poor fit to real-time traffic, it is often used in practice. The main reason is that it is able to pass through firewalls more easily than UDP, especially when run over the HTTP port. Most administrators configure firewalls to protect their networks from unwelcome visitors. They almost always allow TCP connections from remote port 80 to pass through for HTTP and Web traffic. Blocking that port quickly leads to unhappy campers. However, most other ports are blocked, including for RSTP and RTP, which use ports 554 and 5004, amongst others. Thus, the easiest way to get streaming media through the firewall is for the Web site to pretend it is an HTTP server sending a regular HTTP response, at least to the firewall.

There are some other advantages of TCP, too. Because it provides reliability, TCP gives the client a complete copy of the media. This makes it easy for a user to rewind to a previously viewed playout point without concern for lost data. Finally, TCP will buffer as much of the media as possible as quickly as possible. When buffer space is cheap (which it is when the disk is used for storage), the media player can download the media while the user watches. Once the download is complete, the user can watch uninterrupted, even if he loses connectivity. This property is helpful for mobiles because connectivity can change rapidly with motion.

The disadvantage of TCP is the added startup latency (because of TCP startup) and also a higher low-water mark. However, this is rarely much of a penalty as long as the network bandwidth exceeds the media rate by a large factor.

7.4.4 Streaming Live Media

It is not only recorded videos that are tremendously popular on the Web. Live media streaming is very popular too. Once it became possible to stream audio and video over the Internet, commercial radio and TV stations got the idea of broadcasting their content over the Internet as well as over the air. Not so long after that, college stations started putting their signals out over the Internet. Then college *students* started their own Internet broadcasts.

Today, people and companies of all sizes stream live audio and video. The area is a hotbed of innovation as the technologies and standards evolve. Live streaming is used for an online presence by major television stations. This is called **IPTV** (**IP TeleVision**). It is also used to broadcast radio stations like the BBC. This is called **Internet radio**. Both IPTV and Internet radio reach audiences worldwide for events ranging from fashion shows to World Cup soccer and test matches live from the Melbourne Cricket Ground. Live streaming over IP is used as a technology by cable providers to build their own broadcast systems. And it is widely used by low-budget operations from adult sites to zoos. With current technology, virtually anyone can start live streaming quickly and with little expense.

One approach to live streaming is to record programs to disk. Viewers can connect to the server's archives, pull up any program, and download it for listening. A **podcast** is an episode retrieved in this manner. For scheduled events, it is also possible to store content just after it is broadcast live, so the archive is only running, say, half an hour or less behind the live feed.

In fact, this approach is exactly the same as that used for the streaming media we just discussed. It is easy to do, all the techniques we have discussed work for it, and viewers can pick and choose among all the programs in the archive.

A different approach is to broadcast live over the Internet. Viewers tune in to an ongoing media stream, just like turning on the television. However, media players provide the added features of letting the user pause or rewind the playout. The live media will continue to be streamed and will be buffered by the player until the user is ready for it. From the browser's point of view, it looks exactly like the case of streaming stored media. It does not matter to the player whether the content comes from a file or is being sent live, and usually the player will not be able to tell (except that it is not possible to skip forward with a live stream). Given the similarity of mechanism, much of our previous discussion applies, but there are also some key differences.

Importantly, there is still the need for buffering at the client side to smooth out jitter. In fact, a larger amount of buffering is often needed for live streaming (independent of the consideration that the user may pause playback). When streaming from a file, the media can be pushed out at a rate that is greater than the playback rate. This will build up a buffer quickly to compensate for network jitter (and the player will stop the stream if it does not want to buffer more data). In contrast, live media streaming is always transmitted at precisely the rate it is

generated, which is the same as the rate at which it is played back. It cannot be sent faster than this. As a consequence, the buffer must be large enough to handle the full range of network jitter. In practice, a 10–15 second startup delay is usually adequate, so this is not a large problem.

The other important difference is that live streaming events usually have hundreds or thousands of simultaneous viewers of the same content. Under these circumstances, the natural solution for live streaming is to use multicasting. This is not the case for streaming stored media because the users typically stream different content at any given time. Streaming to many users then consists of many individual streaming sessions that happen to occur at the same time.

A multicast streaming scheme works as follows. The server sends each media packet once using IP multicast to a group address. The network delivers a copy of the packet to each member of the group. All of the clients who want to receive the stream have joined the group. The clients do this using IGMP, rather than sending an RTSP message to the media server. This is because the media server is already sending the live stream (except before the first user joins). What is needed is to arrange for the stream to be received locally.

Since multicast is a one-to-many delivery service, the media is carried in RTP packets over a UDP transport. TCP only operates between a single sender and a single receiver. Since UDP does not provide reliability, some packets may be lost. To reduce the level of media loss to an acceptable level, we can use FEC and interleaving, as before.

In the case of FEC, there is a beneficial interaction with multicast that is shown in the parity example of Fig. 7-56. When the packets are multicast, different clients may lose different packets. For example, client 1 has lost packet B, client 2 lost the parity packet P, client 3 lost D, and client 4 did not lose any packets. However, even though three different packets are lost across the clients, each client can recover all of the data packets in this example. All that is required is that each client lose no more than one packet, whichever one it may be, so that the missing packet can be recovered by a parity computation. Nonnenmacher et al. (1997) describe how this idea can be used to boost reliability.

For a server with a large number of clients, multicast of media in RTP and UDP packets is clearly the most efficient way to operate. Otherwise, the server must transmit N streams when it has N clients, which will require a very large amount of network bandwidth at the server for large streaming events.

It may surprise you to learn that the Internet does not work like this in practice. What usually happens is that each user establishes a separate TCP connection to the server, and the media is streamed over that connection. To the client, this is the same as streaming stored media. And as with streaming stored media, there are several reasons for this seemingly poor choice.

The first reason is that IP multicast is not broadly available on the Internet. Some ISPs and networks support it internally, but it is usually not available across network boundaries as is needed for wide-area streaming. The other reasons are

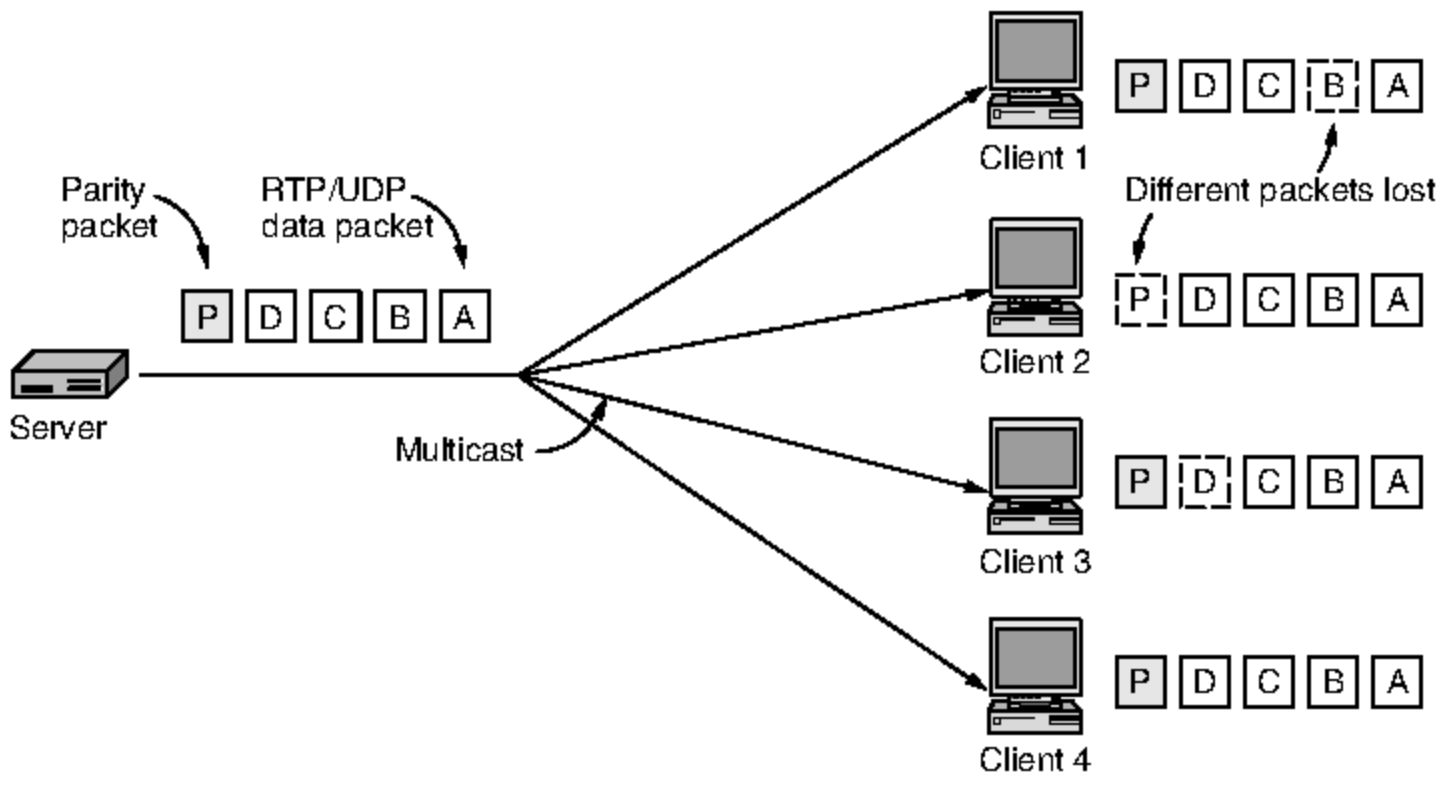


Figure 7-56. Multicast streaming media with a parity packet.

the same advantages of TCP over UDP as discussed earlier. Streaming with TCP will reach nearly all clients on the Internet, particularly when disguised as HTTP to pass through firewalls, and reliable media delivery allows users to rewind easily.

There is one important case in which UDP and multicast can be used for streaming, however: within a provider network. For example, a cable company might decide to broadcast TV channels to customer set-top boxes using IP technology instead of traditional video broadcasts. The use of IP to distribute broadcast video is broadly called IPTV, as discussed above. Since the cable company has complete control of its own network, it can engineer it to support IP multicast and have sufficient bandwidth for UDP-based distribution. All of this is invisible to the customer, as the IP technology exists within the **walled garden** of the provider. It looks just like cable TV in terms of service, but it is IP underneath, with the set-top box being a computer running UDP and the TV set being simply a monitor attached to the computer.

Back to the Internet case, the disadvantage of live streaming over TCP is that the server must send a separate copy of the media for each client. This is feasible for a moderate number of clients, especially for audio. The trick is to place the server at a location with good Internet connectivity so that there is sufficient bandwidth. Usually this means renting a server in a data center from a hosting provider, not using a server at home with only broadband Internet connectivity. There is a very competitive hosting market, so this need not be expensive.

In fact, it is easy for anybody, even a student, to set up and operate a streaming media server such as an Internet radio station. The main components of this

station are illustrated in Fig. 7-57. The basis of the station is an ordinary PC with a decent sound card and microphone. Popular software is used to capture audio and encode it in various formats, for example, MP4, and media players are used to listen to the audio as usual.

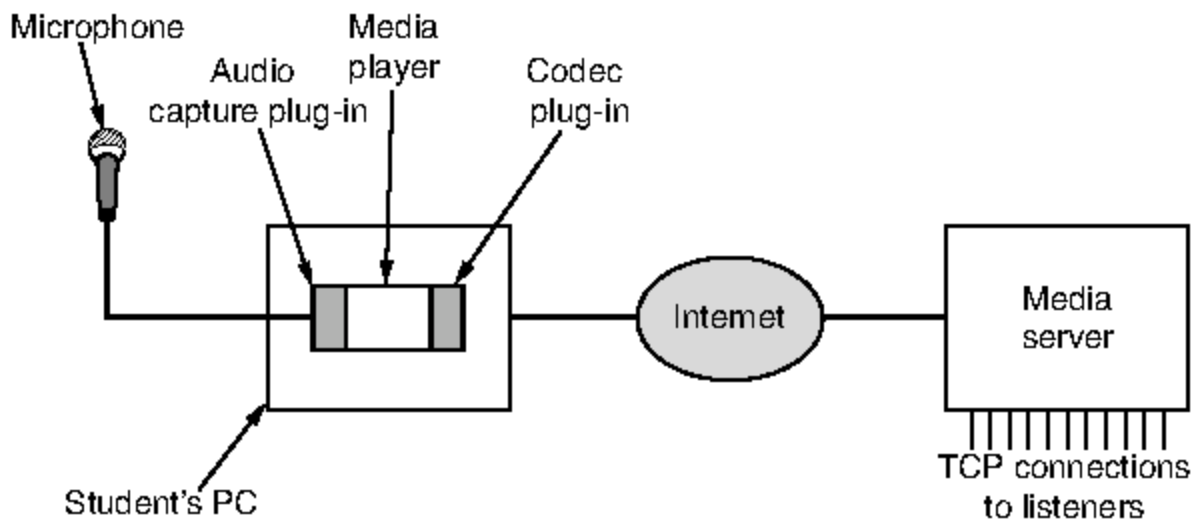


Figure 7-57. A student radio station.

The audio stream captured on the PC is then fed over the Internet to a media server with good network connectivity, either as podcasts for stored file streaming or for live streaming. The server handles the task of distributing the media via large numbers of TCP connections. It also presents a front-end Web site with pages about the station and links to the content that is available for streaming. There are commercial software packages for managing all the pieces, as well as open source packages such as icecast.

However, for a very large number of clients, it becomes infeasible to use TCP to send media to each client from a single server. There is simply not enough bandwidth to the one server. For large streaming sites, the streaming is done using a set of servers that are geographically spread out, so that a client can connect to the nearest server. This is a content distribution network that we will study at the end of the chapter.

7.4.5 Real-Time Conferencing

Once upon a time, voice calls were carried over the public switched telephone network, and network traffic was primarily voice traffic, with a little bit of data traffic here and there. Then came the Internet, and the Web. The data traffic grew and grew, until by 1999 there was as much data traffic as voice traffic (since voice is now digitized, both can be measured in bits). By 2002, the volume of data traffic was an order of magnitude more than the volume of voice traffic and still growing exponentially, with voice traffic staying almost flat.

The consequence of this growth has been to flip the telephone network on its head. Voice traffic is now carried using Internet technologies, and represents only

a tiny fraction of the network bandwidth. This disruptive technology is known as **voice over IP**, and also as **Internet telephony**.

Voice-over-IP is used in several forms that are driven by strong economic factors. (English translation: it saves money so people use it.) One form is to have what look like regular (old-fashioned?) telephones that plug into the Ethernet and send calls over the network. Pehr Anderson was an undergraduate student at M.I.T. when he and his friends prototyped this design for a class project. They got a "B" grade. Not content, he started a company called NBX in 1996, pioneered this kind of voice over IP, and sold it to 3Com for $90 million three years later. Companies love this approach because it lets them do away with separate telephone lines and make do with the networks that they have already.

Another approach is to use IP technology to build a long-distance telephone network. In countries such as the U.S., this network can be accessed for competitive long-distance service by dialing a special prefix. Voice samples are put into packets that are injected into the network and pulled out of the packets when they leave it. Since IP equipment is much cheaper than telecommunications equipment this leads to cheaper services.

As an aside, the difference in price is not entirely technical. For many decades, telephone service was a regulated monopoly that guaranteed the phone companies a fixed percentage profit over their costs. Not surprisingly, this led them to run up costs, for example, by having lots and lots of redundant hardware, justified in the name of better reliability (the telephone system was only allowed to be down for a total of 2 hours every 40 years, or 3 min/year on average). This effect was often referred to as the "gold-plated telephone pole syndrome." Since deregulation, the effect has decreased, of course, but legacy equipment still exists. The IT industry never had any history operating like this, so it has always been lean and mean.

However, we will concentrate on the form of voice over IP that is likely the most visible to users: using one computer to call another computer. This form became commonplace as PCs began shipping with microphones, speakers, cameras, and CPUs fast enough to process media, and people started connecting to the Internet from home at broadband rates. A well-known example is the Skype software that was released starting in 2003. Skype and other companies also provide gateways to make it easy to call regular telephone numbers as well as computers with IP addresses.

As network bandwidth increased, video calls joined voice calls. Initially, video calls were in the domain of companies. Videoconferencing systems were designed to exchange video between two or more locations enabling executives at different locations to see each other while they held their meetings. However, with good broadband Internet connectivity and video compression software, home users can also videoconference. Tools such as Skype that started as audio-only now routinely include video with the calls so that friends and family across the world can see as well as hear each other.

From our point of view, Internet voice or video calls are also a media streaming problem, but one that is much more constrained than streaming a stored file or a live event. The added constraint is the low latency that is needed for a two-way conversation. The telephone network allows a one-way latency of up to 150 msec for acceptable usage, after which delay begins to be perceived as annoying by the participants. (International calls may have a latency of up to 400 msec, by which point they are far from a positive user experience.)

This low latency is difficult to achieve. Certainly, buffering 5–10 seconds of media is not going to work (as it would for broadcasting a live sports event). Instead, video and voice-over-IP systems must be engineered with a variety of techniques to minimize latency. This goal means starting with UDP as the clear choice rather than TCP, because TCP retransmissions introduce at least one round-trip worth of delay. Some forms of latency cannot be reduced, however, even with UDP. For example, the distance between Seattle and Amsterdam is close to 8,000 km. The speed-of-light propagation delay for this distance in optical fiber is 40 msec. Good luck beating that. In practice, the propagation delay through the network will be longer because it will cover a larger distance (the bits do not follow a great circle route) and have transmission delays as each IP router stores and forwards a packet. This fixed delay eats into the acceptable delay budget.

Another source of latency is related to packet size. Normally, large packets are the best way to use network bandwidth because they are more efficient. However, at an audio sampling rate of 64 kbps, a 1-KB packet would take 125 msec to fill (and even longer if the samples are compressed). This delay would consume most of the overall delay budget. In addition, if the 1-KB packet is sent over a broadband access link that runs at just 1 Mbps, it will take 8 msec to transmit. Then add another 8 msec for the packet to go over the broadband link at the other end. Clearly, large packets will not work.

Instead, voice-over-IP systems use short packets to reduce latency at the cost of bandwidth efficiency. They batch audio samples in smaller units, commonly 20 msec. At 64 kbps, this is 160 bytes of data, less with compression. However, by definition the delay from this packetization will be 20 msec. The transmission delay will be smaller as well because the packet is shorter. In our example, it would reduce to around 1 msec. By using short packets, the minimum one-way delay for a Seattle-to-Amsterdam packet has been reduced from an unacceptable 181 msec (40 + 125 + 16) to an acceptable 62 msec (40 + 20 + 2).

We have not even talked about the software overhead, but it, too, will eat up some of the delay budget. This is especially true for video, since compression is usually needed to fit video into the available bandwidth. Unlike streaming from a stored file, there is no time to have a computationally intensive encoder for high levels of compression. The encoder and the decoder must both run quickly.

Buffering is still needed to play out the media samples on time (to avoid unintelligible audio or jerky video), but the amount of buffering must be kept very small since the time remaining in our delay budget is measured in milliseconds.

When a packet takes too long to arrive, the player will skip over the missing samples, perhaps playing ambient noise or repeating a frame to mask the loss to the user. There is a trade-off between the size of the buffer used to handle jitter and the amount of media that is lost. A smaller buffer reduces latency but results in more loss due to jitter. Eventually, as the size of the buffer shrinks, the loss will become noticeable to the user.

Observant readers may have noticed that we have said nothing about the network layer protocols so far in this section. The network can reduce latency, or at least jitter, by using quality of service mechanisms. The reason that this issue has not come up before is that streaming is able to operate with substantial latency, even in the live streaming case. If latency is not a major concern, a buffer at the end host is sufficient to handle the problem of jitter. However, for real-time conferencing, it is usually important to have the network reduce delay and jitter to help meet the delay budget. The only time that it is not important is when there is so much network bandwidth that everyone gets good service.

In Chap. 5, we described two quality of service mechanisms that help with this goal. One mechanism is DS (Differentiated Services), in which packets are marked as belonging to different classes that receive different handling within the network. The appropriate marking for voice-over-IP packets is low delay. In practice, systems set the DS codepoint to the well-known value for the *Expedited Forwarding* class with *Low Delay* type of service. This is especially useful over broadband access links, as these links tend to be congested when Web traffic or other traffic competes for use of the link. Given a stable network path, delay and jitter are increased by congestion. Every 1-KB packet takes 8 msec to send over a 1-Mbps link, and a voice-over-IP packet will incur these delays if it is sitting in a queue behind Web traffic. However, with a low delay marking the voice-over-IP packets will jump to the head of the queue, bypassing the Web packets and lowering their delay.

The second mechanism that can reduce delay is to make sure that there is sufficient bandwidth. If the available bandwidth varies or the transmission rate fluctuates (as with compressed video) and there is sometimes not sufficient bandwidth, queues will build up and add to the delay. This will occur even with DS. To ensure sufficient bandwidth, a reservation can be made with the network. This capability is provided by integrated services. Unfortunately, it is not widely deployed. Instead, networks are engineered for an expected traffic level or network customers are provided with service-level agreements for a given traffic level. Applications must operate below this level to avoid causing congestion and introducing unnecessary delays. For casual videoconferencing at home, the user may choose a video quality as a proxy for bandwidth needs, or the software may test the network path and select an appropriate quality automatically.

Any of the above factors can cause the latency to become unacceptable, so real-time conferencing requires that attention be paid to all of them. For an overview of voice over IP and analysis of these factors, see Goode (2002).

Now that we have discussed the problem of latency in the media streaming path, we will move on to the other main problem that conferencing systems must address. This problem is how to set up and tear down calls. We will look at two protocols that are widely used for this purpose, H.323 and SIP. Skype is another important system, but its inner workings are proprietary.

H.323

One thing that was clear to everyone before voice and video calls were made over the Internet was that if each vendor designed its own protocol stack, the system would never work. To avoid this problem, a number of interested parties got together under ITU auspices to work out standards. In 1996, ITU issued recommendation **H.323**, entitled "Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-Guaranteed Quality of Service." Only the telephone industry would think of such a name. It was quickly changed to "Packet-based Multimedia Communications Systems" in the 1998 revision. H.323 was the basis for the first widespread Internet conferencing systems. It remains the most widely deployed solution, in its seventh version as of 2009.

H.323 is more of an architectural overview of Internet telephony than a specific protocol. It references a large number of specific protocols for speech coding, call setup, signaling, data transport, and other areas rather than specifying these things itself. The general model is depicted in Fig. 7-58. At the center is a **gateway** that connects the Internet to the telephone network. It speaks the H.323 protocols on the Internet side and the PSTN protocols on the telephone side. The communicating devices are called **terminals**. A LAN may have a **gatekeeper**, which controls the end points under its jurisdiction, called a **zone**.

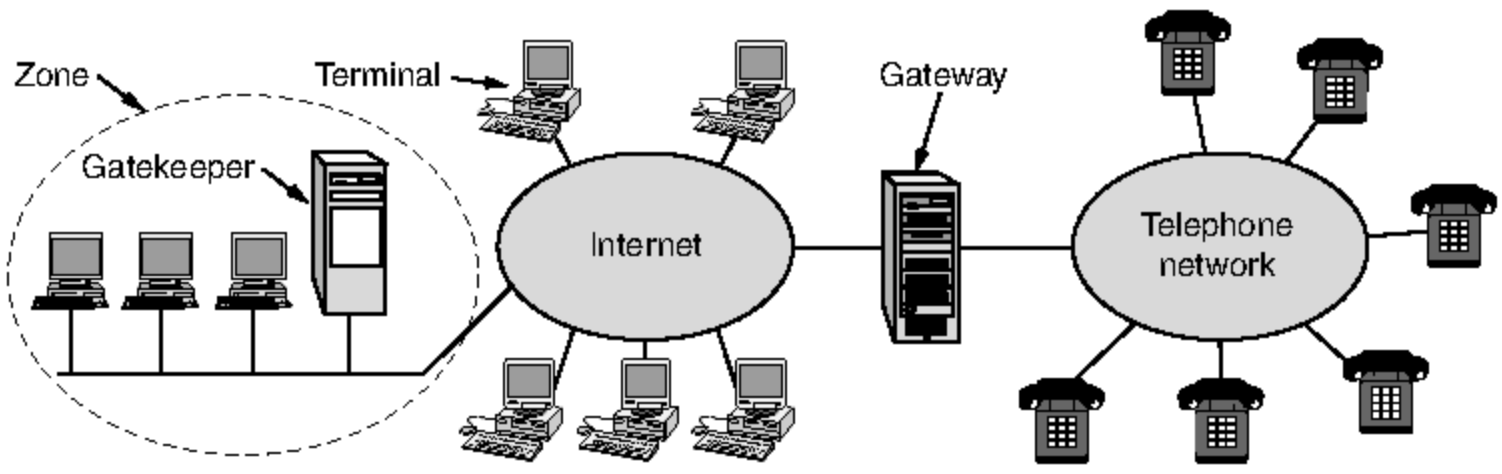


Figure 7-58. The H.323 architectural model for Internet telephony.

A telephone network needs a number of protocols. To start with, there is a protocol for encoding and decoding audio and video. Standard telephony representations of a single voice channel as 64 kbps of digital audio (8000 samples of 8 bits per second) are defined in ITU recommendation **G.711**. All H.323 systems

must support G.711. Other encodings that compress speech are permitted, but not required. They use different compression algorithms and make different trade-offs between quality and bandwidth. For video, the MPEG forms of video compression that we described above are supported, including H.264.

Since multiple compression algorithms are permitted, a protocol is needed to allow the terminals to negotiate which one they are going to use. This protocol is called **H.245**. It also negotiates other aspects of the connection such as the bit rate. RTCP is need for the control of the RTP channels. Also required is a protocol for establishing and releasing connections, providing dial tones, making ringing sounds, and the rest of the standard telephony. ITU **Q.931** is used here. The terminals need a protocol for talking to the gatekeeper (if present) as well. For this purpose, **H.225** is used. The PC-to-gatekeeper channel it manages is called the **RAS** (**Registration/Admission/Status**) channel. This channel allows terminals to join and leave the zone, request and return bandwidth, and provide status updates, among other things. Finally, a protocol is needed for the actual data transmission. RTP over UDP is used for this purpose. It is managed by RTCP, as usual. The positioning of all these protocols is shown in Fig. 7-59.

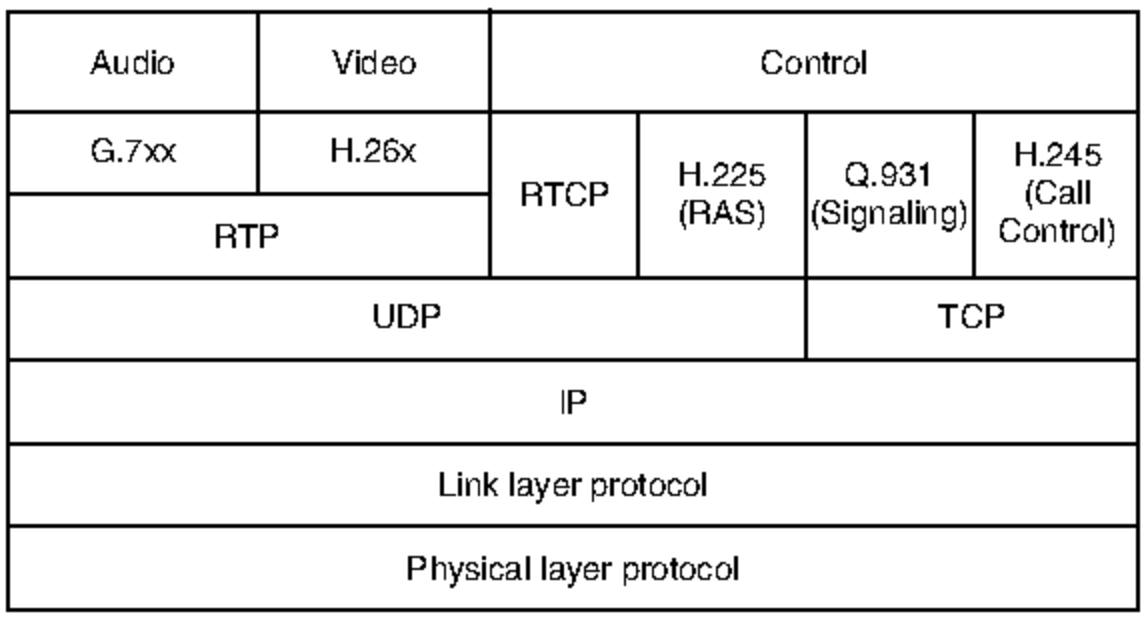


Figure 7-59. The H.323 protocol stack.

To see how these protocols fit together, consider the case of a PC terminal on a LAN (with a gatekeeper) calling a remote telephone. The PC first has to discover the gatekeeper, so it broadcasts a UDP gatekeeper discovery packet to port 1718. When the gatekeeper responds, the PC learns the gatekeeper's IP address. Now the PC registers with the gatekeeper by sending it a RAS message in a UDP packet. After it has been accepted, the PC sends the gatekeeper a RAS admission message requesting bandwidth. Only after bandwidth has been granted may call setup begin. The idea of requesting bandwidth in advance is to allow the gatekeeper to limit the number of calls. It can then avoid oversubscribing the outgoing line in order to help provide the necessary quality of service.

As an aside, the telephone system does the same thing. When you pick up the receiver, a signal is sent to the local end office. If the office has enough spare capacity for another call, it generates a dial tone. If not, you hear nothing. Nowadays, the system is so overdimensioned that the dial tone is nearly always instantaneous, but in the early days of telephony, it often took a few seconds. So if your grandchildren ever ask you "Why are there dial tones?" now you know. Except by then, probably telephones will no longer exist.

The PC now establishes a TCP connection to the gatekeeper to begin call setup. Call setup uses existing telephone network protocols, which are connection oriented, so TCP is needed. In contrast, the telephone system has nothing like RAS to allow telephones to announce their presence, so the H.323 designers were free to use either UDP or TCP for RAS, and they chose the lower-overhead UDP.

Now that it has bandwidth allocated, the PC can send a Q.931 *SETUP* message over the TCP connection. This message specifies the number of the telephone being called (or the IP address and port, if a computer is being called). The gatekeeper responds with a Q.931 *CALL PROCEEDING* message to acknowledge correct receipt of the request. The gatekeeper then forwards the *SETUP* message to the gateway.

The gateway, which is half computer, half telephone switch, then makes an ordinary telephone call to the desired (ordinary) telephone. The end office to which the telephone is attached rings the called telephone and also sends back a Q.931 *ALERT* message to tell the calling PC that ringing has begun. When the person at the other end picks up the telephone, the end office sends back a Q.931 *CONNECT* message to signal the PC that it has a connection.

Once the connection has been established, the gatekeeper is no longer in the loop, although the gateway is, of course. Subsequent packets bypass the gatekeeper and go directly to the gateway's IP address. At this point, we just have a bare tube running between the two parties. This is just a physical layer connection for moving bits, no more. Neither side knows anything about the other one.

The H.245 protocol is now used to negotiate the parameters of the call. It uses the H.245 control channel, which is always open. Each side starts out by announcing its capabilities, for example, whether it can handle video (H.323 can handle video) or conference calls, which codecs it supports, etc. Once each side knows what the other one can handle, two unidirectional data channels are set up and a codec and other parameters are assigned to each one. Since each side may have different equipment, it is entirely possible that the codecs on the forward and reverse channels are different. After all negotiations are complete, data flow can begin using RTP. It is managed using RTCP, which plays a role in congestion control. If video is present, RTCP handles the audio/video synchronization. The various channels are shown in Fig. 7-60. When either party hangs up, the Q.931 call signaling channel is used to tear down the connection after the call has been completed in order to free up resources no longer needed.

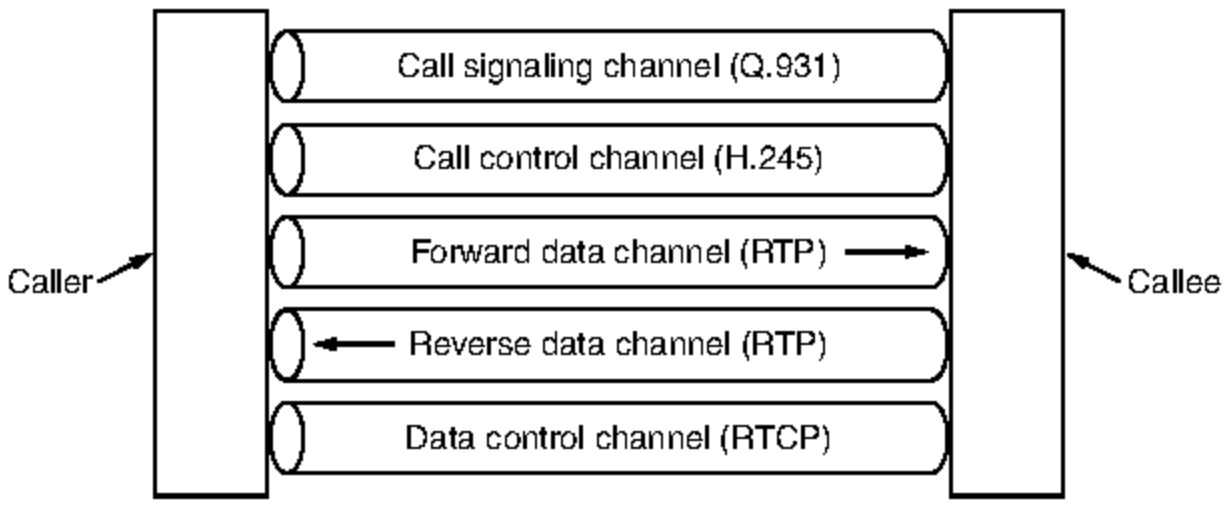


Figure 7-60. Logical channels between the caller and callee during a call.

When the call is terminated, the calling PC contacts the gatekeeper again with a RAS message to release the bandwidth it has been assigned. Alternatively, it can make another call.

We have not said anything about quality of service as part of H.323, even though we have said it is an important part of making real-time conferencing a success. The reason is that QoS falls outside the scope of H.323. If the underlying network is capable of producing a stable, jitter-free connection from the calling PC to the gateway, the QoS on the call will be good; otherwise, it will not be. However, any portion of the call on the telephone side will be jitter-free, because that is how the telephone network is designed.

SIP—The Session Initiation Protocol

H.323 was designed by ITU. Many people in the Internet community saw it as a typical telco product: large, complex, and inflexible. Consequently, IETF set up a committee to design a simpler and more modular way to do voice over IP. The major result to date is **SIP** (**Session Initiation Protocol**). The latest version is described in RFC 3261, which was written in 2002. This protocol describes how to set up Internet telephone calls, video conferences, and other multimedia connections. Unlike H.323, which is a complete protocol suite, SIP is a single module, but it has been designed to interwork well with existing Internet applications. For example, it defines telephone numbers as URLs, so that Web pages can contain them, allowing a click on a link to initiate a telephone call (the same way the *mailto* scheme allows a click on a link to bring up a program to send an email message).

SIP can establish two-party sessions (ordinary telephone calls), multiparty sessions (where everyone can hear and speak), and multicast sessions (one sender, many receivers). The sessions may contain audio, video, or data, the latter being useful for multiplayer real-time games, for example. SIP just handles setup, management, and termination of sessions. Other protocols, such as RTP/RTCP, are

also used for data transport. SIP is an application-layer protocol and can run over UDP or TCP, as required.

SIP supports a variety of services, including locating the callee (who may not be at his home machine) and determining the callee's capabilities, as well as handling the mechanics of call setup and termination. In the simplest case, SIP sets up a session from the caller's computer to the callee's computer, so we will examine that case first.

Telephone numbers in SIP are represented as URLs using the *sip* scheme, for example, *sip:ilse@cs.university.edu* for a user named Ilse at the host specified by the DNS name *cs.university.edu*. SIP URLs may also contain IPv4 addresses, IPv6 addresses, or actual telephone numbers.

The SIP protocol is a text-based protocol modeled on HTTP. One party sends a message in ASCII text consisting of a method name on the first line, followed by additional lines containing headers for passing parameters. Many of the headers are taken from MIME to allow SIP to interwork with existing Internet applications. The six methods defined by the core specification are listed in Fig. 7-61.

Method	Description
INVITE	Request initiation of a session
ACK	Confirm that a session has been initiated
BYE	Request termination of a session
OPTIONS	Query a host about its capabilities
CANCEL	Cancel a pending request
REGISTER	Inform a redirection server about the user's current location

Figure 7-61. SIP methods.

To establish a session, the caller either creates a TCP connection with the callee and sends an *INVITE* message over it or sends the *INVITE* message in a UDP packet. In both cases, the headers on the second and subsequent lines describe the structure of the message body, which contains the caller's capabilities, media types, and formats. If the callee accepts the call, it responds with an HTTP-type reply code (a three-digit number using the groups of Fig. 7-38, 200 for acceptance). Following the reply-code line, the callee also may supply information about its capabilities, media types, and formats.

Connection is done using a three-way handshake, so the caller responds with an *ACK* message to finish the protocol and confirm receipt of the 200 message.

Either party may request termination of a session by sending a message with the *BYE* method. When the other side acknowledges it, the session is terminated.

The *OPTIONS* method is used to query a machine about its own capabilities. It is typically used before a session is initiated to find out if that machine is even capable of voice over IP or whatever type of session is being contemplated.

The *REGISTER* method relates to SIP's ability to track down and connect to a user who is away from home. This message is sent to a SIP location server that keeps track of who is where. That server can later be queried to find the user's current location. The operation of redirection is illustrated in Fig. 7-62. Here, the caller sends the *INVITE* message to a proxy server to hide the possible redirection. The proxy then looks up where the user is and sends the *INVITE* message there. It then acts as a relay for the subsequent messages in the three-way handshake. The *LOOKUP* and *REPLY* messages are not part of SIP; any convenient protocol can be used, depending on what kind of location server is used.

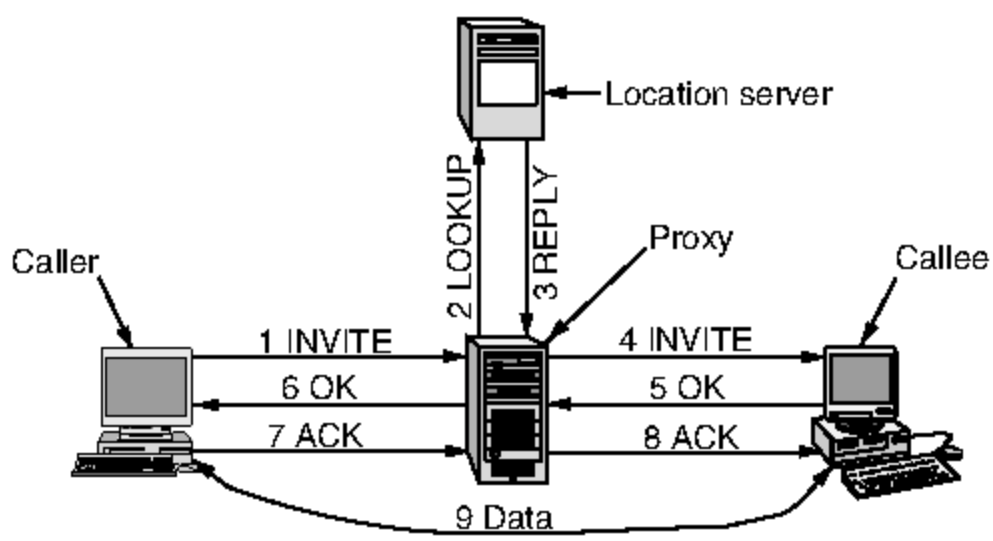


Figure 7-62. Use of a proxy server and redirection with SIP.

SIP has a variety of other features that we will not describe here, including call waiting, call screening, encryption, and authentication. It also has the ability to place calls from a computer to an ordinary telephone, if a suitable gateway between the Internet and telephone system is available.

Comparison of H.323 and SIP

Both H.323 and SIP allow two-party and multiparty calls using both computers and telephones as end points. Both support parameter negotiation, encryption, and the RTP/RTCP protocols. A summary of their similarities and differences is given in Fig. 7-63.

Although the feature sets are similar, the two protocols differ widely in philosophy. H.323 is a typical, heavyweight, telephone-industry standard, specifying the complete protocol stack and defining precisely what is allowed and what is forbidden. This approach leads to very well defined protocols in each layer, easing the task of interoperability. The price paid is a large, complex, and rigid standard that is difficult to adapt to future applications.

In contrast, SIP is a typical Internet protocol that works by exchanging short lines of ASCII text. It is a lightweight module that interworks well with other Internet protocols but less well with existing telephone system signaling protocols.

Item	H.323	SIP
Designed by	ITU	IETF
Compatibility with PSTN	Yes	Largely
Compatibility with Internet	Yes, over time	Yes
Architecture	Monolithic	Modular
Completeness	Full protocol stack	SIP just handles setup
Parameter negotiation	Yes	Yes
Call signaling	Q.931 over TCP	SIP over TCP or UDP
Message format	Binary	ASCII
Media transport	RTP/RTCP	RTP/RTCP
Multiparty calls	Yes	Yes
Multimedia conferences	Yes	No
Addressing	URL or phone number	URL
Call termination	Explicit or TCP release	Explicit or timeout
Instant messaging	No	Yes
Encryption	Yes	Yes
Size of standards	1400 pages	250 pages
Implementation	Large and complex	Moderate, but issues
Status	Widespread, esp. video	Alternative, esp. voice

Figure 7-63. Comparison of H.323 and SIP.

Because the IETF model of voice over IP is highly modular, it is flexible and can be adapted to new applications easily. The downside is that is has suffered from ongoing interoperability problems as people try to interpret what the standard means.

7.5 CONTENT DELIVERY

The Internet used to be all about communication, like the telephone network. Early on, academics would communicate with remote machines, logging in over the network to perform tasks. People have used email to communicate with each other for a long time, and now use video and voice over IP as well. Since the Web grew up, however, the Internet has become more about content than communication. Many people use the Web to find information, and there is a tremendous amount of peer-to-peer file sharing that is driven by access to movies, music, and programs. The switch to content has been so pronounced that the majority of Internet bandwidth is now used to deliver stored videos.

Because the task of distributing content is different from that of communication, it places different requirements on the network. For example, if Sally wants to talk to Jitu, she may make a voice-over-IP call to his mobile. The communication must be with a particular computer; it will do no good to call Paul's computer. But if Jitu wants to watch his team's latest cricket match, he is happy to stream video from whichever computer can provide the service. He does not mind whether the computer is Sally's or Paul's, or, more likely, an unknown server in the Internet. That is, location does not matter for content, except as it affects performance (and legality).

The other difference is that some Web sites that provide content have become tremendously popular. YouTube is a prime example. It allows users to share videos of their own creation on every conceivable topic. Many people want to do this. The rest of us want to watch. With all of these bandwidth-hungry videos, it is estimated that YouTube accounts for up to 10% of Internet traffic today.

No single server is powerful or reliable enough to handle such a startling level of demand. Instead, YouTube and other large content providers build their own content distribution networks. These networks use data centers spread around the world to serve content to an extremely large number of clients with good performance and availability.

The techniques that are used for content distribution have been developed over time. Early in the growth of the Web, its popularity was almost its undoing. More demands for content led to servers and networks that were frequently overloaded. Many people began to call the WWW the World Wide Wait.

In response to consumer demand, very large amounts of bandwidth were provisioned in the core of the Internet, and faster broadband connectivity was rolled out at the edge of the network. This bandwidth was key to improving performance, but it is only part of the solution. To reduce the endless delays, researchers also developed different architectures to use the bandwidth for distributing content.

One architecture is a **CDN** (**Content Distribution Network**). In it, a provider sets up a distributed collection of machines at locations inside the Internet and uses them to serve content to clients. This is the choice of the big players. An alternative architecture is a **P2P** (**Peer-to-Peer**) network. In it, a collection of computers pool their resources to serve content to each other, without separately provisioned servers or any central point of control. This idea has captured people's imagination because, by acting together, many little players can pack an enormous punch.

In this section, we will look at the problem of distributing content on the Internet and some of the solutions that are used in practice. After briefly discussing content popularity and Internet traffic, we will describe how to build powerful Web servers and use caching to improve performance for Web clients. Then we will come to the two main architectures for distributing content: CDNs and P2P networks. There design and properties are quite different, as we will see.

7.5.1 Content and Internet Traffic

To design and engineer networks that work well, we need an understanding of the traffic that they must carry. With the shift to content, for example, servers have migrated from company offices to Internet data centers that provide large numbers of machines with excellent network connectivity. To run even a small server nowadays, it is easier and cheaper to rent a virtual server hosted in an Internet data center than to operate a real machine in a home or office with broadband connectivity to the Internet.

Fortunately, there are only two facts about Internet traffic that is it essential to know. The first fact is that it changes quickly, not only in the details but in the overall makeup. Before 1994, most traffic was traditional FTP file transfer (for moving programs and data sets between computers) and email. Then the Web arrived and grew exponentially. Web traffic left FTP and email traffic in the dust long before the dot com bubble of 2000. Starting around 2000, P2P file sharing for music and then movies took off. By 2003, most Internet traffic was P2P traffic, leaving the Web in the dust. Sometime in the late 2000s, video streamed using content distribution methods by sites like YouTube began to exceed P2P traffic. By 2014, Cisco predicts that 90% of all Internet traffic will be video in one form or another (Cisco, 2010).

It is not always traffic volume that matters. For instance, while voice-over-IP traffic boomed even before Skype started in 2003, it will always be a minor blip on the chart because the bandwidth requirements of audio are two orders of magnitude lower than for video. However, voice-over-IP traffic stresses the network in other ways because it is sensitive to latency. As another example, online social networks have grown furiously since Facebook started in 2004. In 2010, for the first time, Facebook reached more users on the Web per day than Google. Even putting the traffic aside (and there is an awful lot of traffic), online social networks are important because they are changing the way that people interact via the Internet.

The point we are making is that seismic shifts in Internet traffic happen quickly, and with some regularity. What will come next? Please check back in the 6th edition of this book and we will let you know.

The second essential fact about Internet traffic is that it is highly skewed. Many properties with which we are familiar are clustered around an average. For instance, most adults are close to the average height. There are some tall people and some short people, but few very tall or very short people. For these kinds of properties, it is possible to design for a range that is not very large but nonetheless captures the majority of the population.

Internet traffic is not like this. For a long time, it has been known that there are a small number of Web sites with massive traffic and a vast number of Web site with much smaller traffic. This feature has become part of the language of networking. Early papers talked about traffic in terms of **packet trains**, the idea

being that express trains with a large number of packets would suddenly travel down a link (Jain and Routhier, 1986). This was formalized as the notion of **self-similarity**, which for our purposes can be thought of as network traffic that exhibits many short and many long gaps even when viewed at different time scales (Leland et al., 1994). Later work spoke of long traffic flows as **elephants** and short traffic flows as **mice**. The idea is that there are only a few elephants and many mice, but the elephants matter because they are so big.

Returning to Web content, the same sort of skew is evident. Experience with video rental stores, public libraries, and other such organizations shows that not all items are equally popular. Experimentally, when N movies are available, the fraction of all requests for the kth most popular one is approximately C/k. Here, C is computed to normalize the sum to 1, namely,

$$C = 1/(1 + 1/2 + 1/3 + 1/4 + 1/5 + \cdots + 1/N)$$

Thus, the most popular movie is seven times as popular as the number seven movie. This result is known as **Zipf's law** (Zipf, 1949). It is named after George Zipf, a professor of linguistics at Harvard University who noted that the frequency of a word's usage in a large body of text is inversely proportional to its rank. For example, the 40th most common word is used twice as much as the 80th most common word and three times as much as the 120th most common word.

A Zipf distribution is shown in Fig. 7-64(a). It captures the notion that there are a small number of popular items and a great many unpopular items. To recognize distributions of this form, it is convenient to plot the data on a log scale on both axes, as shown in Fig. 7-64(b). The result should be a straight line.

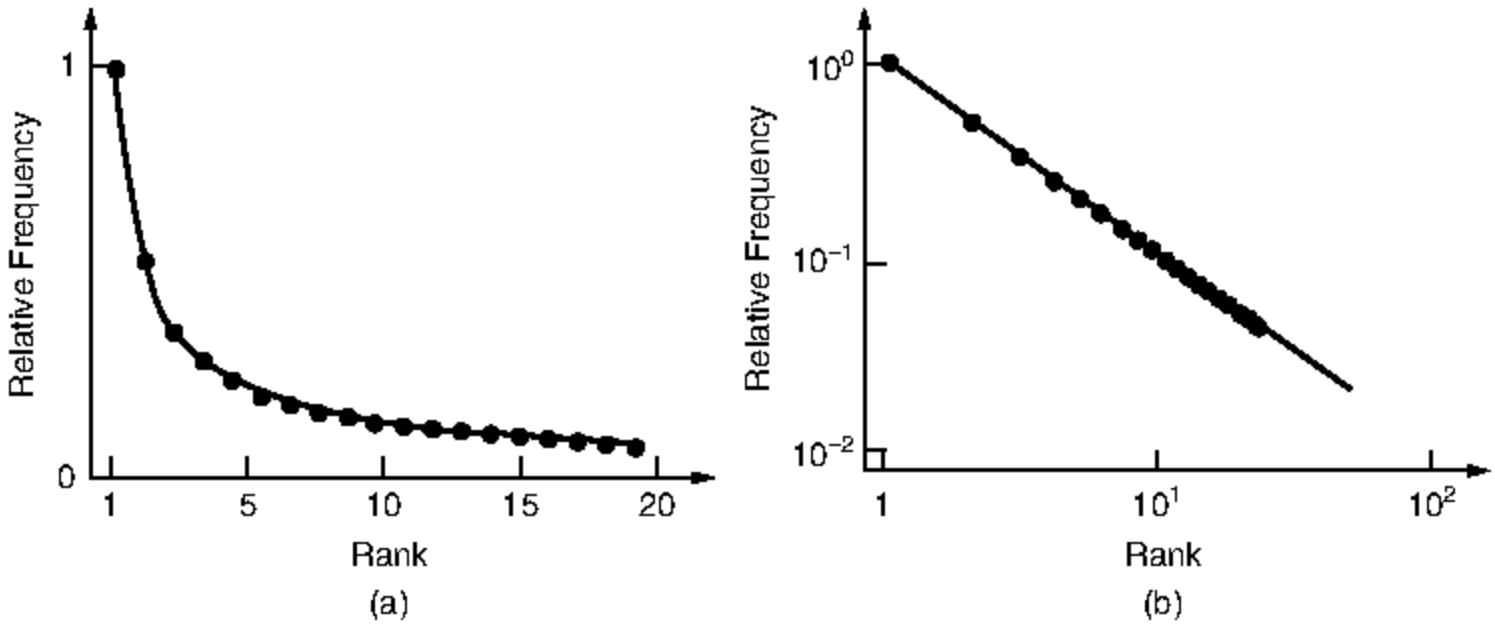


Figure 7-64. Zipf distribution (a) On a linear scale. (b) On a log-log scale.

When people looked at the popularity of Web pages, it also turned out to roughly follow Zipf's law (Breslau et al., 1999). A Zipf distribution is one example in a family of distributions known as **power laws**. Power laws are evident

in many human phenomena, such as the distribution of city populations and of wealth. They have the same propensity to describe a few large players and a great many smaller players, and they too appear as a straight line on a log-log plot. It was soon discovered that the topology of the Internet could be roughly described with power laws (Faloutsos et al., 1999). Next, researchers began plotting every imaginable property of the Internet on a log scale, observing a straight line, and shouting: "Power law!"

However, what matters more than a straight line on a log-log plot is what these distributions mean for the design and use of networks. Given the many forms of content that have Zipf or power law distributions, it seems fundamental that Web sites on the Internet are Zipf-like in popularity. This in turn means that an average site is not a useful representation. Sites are better described as either popular or unpopular. Both kinds of sites matter. The popular sites obviously matter, since a few popular sites may be responsible for most of the traffic on the Internet. Perhaps surprisingly, the unpopular sites can matter too. This is because the total amount of traffic directed to the unpopular sites can add up to a large fraction of the overall traffic. The reason is that there are so many unpopular sites. The notion that, collectively, many unpopular choices can matter has been popularized by books such as *The Long Tail* (Anderson, 2008a).

Curves showing decay like that of Fig. 7-64(a) are common, but they are not all the same. In particular, situations in which the rate of decay is proportional to how much material is left (such as with unstable radioactive atoms) exhibit **exponential decay**, which drops off much faster than Zipf's Law. The number of items, say atoms, left after time t is usually expressed as $e^{-t/\alpha}$, where the constant α determines how fast the decay is. The difference between exponential decay and Zipf's Law is that with exponential decay, it is safe to ignore the end of tail but with Zipf's Law the total weight of the tail is significant and cannot be ignored.

To work effectively in this skewed world, we must be able to build both kinds of Web sites. Unpopular sites are easy to handle. By using DNS, many different sites may actually point to the same computer in the Internet that runs all of the sites. On the other hand, popular sites are difficult to handle. There is no single computer even remotely powerful enough, and using a single computer would make the site inaccessible for millions of users if it fails. To handle these sites, we must build content distribution systems. We will start on that quest next.

7.5.2 Server Farms and Web Proxies

The Web designs that we have seen so far have a single server machine talking to multiple client machines. To build large Web sites that perform well, we can speed up processing on either the server side or the client side. On the server side, more powerful Web servers can be built with a server farm, in which a cluster of computers acts as a single server. On the client side, better performance can

be achieved with better caching techniques. In particular, proxy caches provide a large shared cache for a group of clients.

We will describe each of these techniques in turn. However, note that neither technique is sufficient to build the largest Web sites. Those popular sites require the content distribution methods that we describe in the following sections, which combine computers at many different locations.

Server Farms

No matter how much bandwidth one machine has, it can only serve so many Web requests before the load is too great. The solution in this case is to use more than one computer to make a Web server. This leads to the **server farm** model of Fig. 7-65.

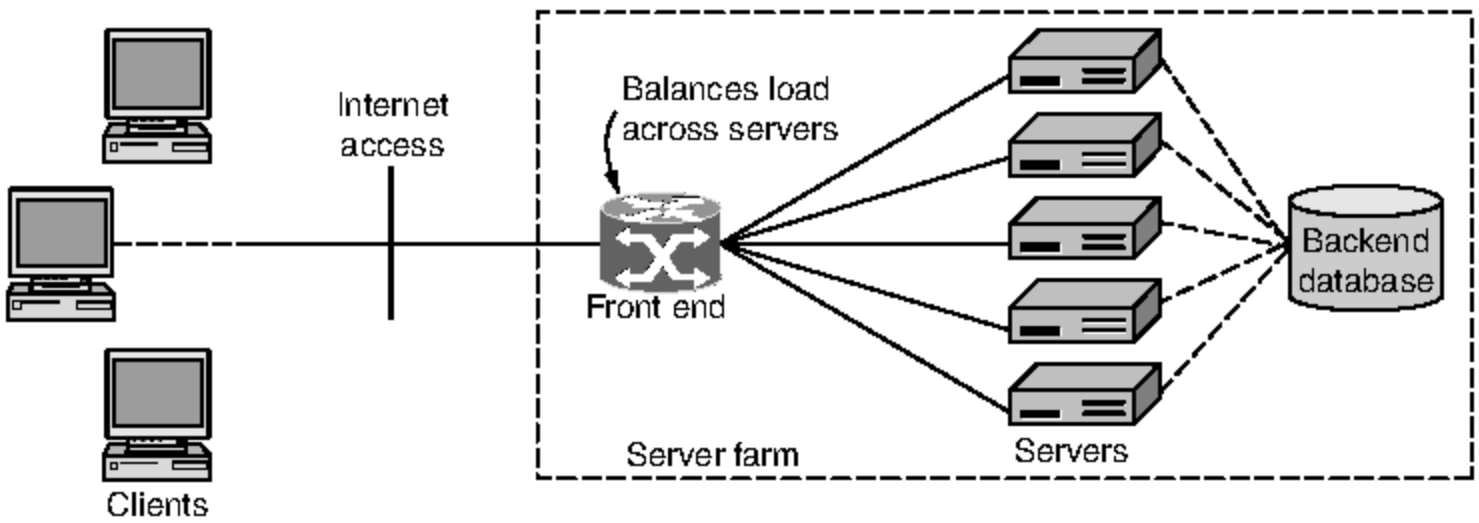


Figure 7-65. A server farm.

The difficulty with this seemingly simple model is that the set of computers that make up the server farm must look like a single logical Web site to clients. If they do not, we have just set up different Web sites that run in parallel.

There are several possible solutions to make the set of servers appear to be one Web site. All of the solutions assume that any of the servers can handle a request from any client. To do this, each server must have a copy of the Web site. The servers are shown as connected to a common back-end database by a dashed line for this purpose.

One solution is to use DNS to spread the requests across the servers in the server farm. When a DNS request is made for the Web URL, the DNS server returns a rotating list of the IP addresses of the servers. Each client tries one IP address, typically the first on the list. The effect is that different clients contact different servers to access the same Web site, just as intended. The DNS method is at the heart of CDNs, and we will revisit it later in this section.

The other solutions are based on a **front end** that sprays incoming requests over the pool of servers in the server farm. This happens even when the client

contacts the server farm using a single destination IP address. The front end is usually a link-layer switch or an IP router, that is, a device that handles frames or packets. All of the solutions are based on it (or the servers) peeking at the network, transport, or application layer headers and using them in nonstandard ways. A Web request and response are carried as a TCP connection. To work correctly, the front end must distribute all of the packets for a request to the same server.

A simple design is for the front end to broadcast all of the incoming requests to all of the servers. Each server answers only a fraction of the requests by prior agreement. For example, 16 servers might look at the source IP address and reply to the request only if the last 4 bits of the source IP address match their configured selectors. Other packets are discarded. While this is wasteful of incoming bandwidth, often the responses are much longer than the request, so it is not nearly as inefficient as it sounds.

In a more general design, the front end may inspect the IP, TCP, and HTTP headers of packets and arbitrarily map them to a server. The mapping is called a **load balancing** policy as the goal is to balance the workload across the servers. The policy may be simple or complex. A simple policy might be to use the servers one after the other in turn, or round-robin. With this approach, the front end must remember the mapping for each request so that subsequent packets that are part of the same request will be sent to the same server. Also, to make the site more reliable than a single server, the front end should notice when servers have failed and stop sending them requests.

Much like NAT, this general design is perilous, or at least fragile, in that we have just created a device that violates the most basic principle of layered protocols: each layer must use its own header for control purposes and may not inspect and use information from the payload for any purpose. But people design such systems anyway and when they break in the future due to changes in higher layers, they tend to be surprised. The front end in this case is a switch or router, but it may take action based on transport layer information or higher. Such a box is called a **middlebox** because it interposes itself in the middle of a network path in which it has no business, according to the protocol stack. In this case, the front end is best considered an internal part of a server farm that terminates all layers up to the application layer (and hence can use all of the header information for those layers).

Nonetheless, as with NAT, this design is useful in practice. The reason for looking at TCP headers is that it is possible to do a better job of load balancing than with IP information alone. For example, one IP address may represent an entire company and make many requests. It is only by looking at TCP or higher-layer information that these requests can be mapped to different servers.

The reason for looking at the HTTP headers is somewhat different. Many Web interactions access and update databases, such as when a customer looks up her most recent purchase. The server that fields this request will have to query the back-end database. It is useful to direct subsequent requests from the same user to

the same server, because that server has already cached information about the user. The simplest way to cause this to happen is to use Web cookies (or other information to distinguish the user) and to inspect the HTTP headers to find the cookies.

As a final note, although we have described this design for Web sites, a server farm can be built for other kinds of servers as well. An example is servers streaming media over UDP. The only change that is required is for the front end to be able to load balance these requests (which will have different protocol header fields than Web requests).

Web Proxies

Web requests and responses are sent using HTTP. In Sec. 7.3, we described how browsers can cache responses and reuse them to answer future requests. Various header fields and rules are used by the browser to determine if a cached copy of a Web page is still fresh. We will not repeat that material here.

Caching improves performance by shortening the response time and reducing the network load. If the browser can determine that a cached page is fresh by itself, the page can be fetched from the cache immediately, with no network traffic at all. However, even if the browser must ask the server for confirmation that the page is still fresh, the response time is shortened and the network load is reduced, especially for large pages, since only a small message needs to be sent.

However, the best the browser can do is to cache all of the Web pages that the user has previously visited. From our discussion of popularity, you may recall that as well as a few popular pages that many people visit repeatedly, there are many, many unpopular pages. In practice, this limits the effectiveness of browser caching because there are a large number of pages that are visited just once by a given user. These pages always have to be fetched from the server.

One strategy to make caches more effective is to share the cache among multiple users. That way, a page already fetched for one user can be returned to another user when that user makes the same request. Without browser caching, both users would need to fetch the page from the server. Of course, this sharing cannot be done for encrypted traffic, pages that require authentication, and uncacheable pages (e.g., current stock prices) that are returned by programs. Dynamic pages created by programs, especially, are a growing case for which caching is not effective. Nonetheless, there are plenty of Web pages that are visible to many users and look the same no matter which user makes the request (e.g., images).

A **Web proxy** is used to share a cache among users. A proxy is an agent that acts on behalf of someone else, such as the user. There are many kinds of proxies. For instance, an ARP proxy replies to ARP requests on behalf of a user who is elsewhere (and cannot reply for himself). A Web proxy fetches Web requests on behalf of its users. It normally provides caching of the Web responses, and since it is shared across users it has a substantially larger cache than a browser.

When a proxy is used, the typical setup is for an organization to operate one Web proxy for all of its users. The organization might be a company or an ISP. Both stand to benefit by speeding up Web requests for its users and reducing its bandwidth needs. While flat pricing, independent of usage, is common for end users, most companies and ISPs are charged according to the bandwidth that they use.

This setup is shown in Fig. 7-66. To use the proxy, each browser is configured to make page requests to the proxy instead of to the page's real server. If the proxy has the page, it returns the page immediately. If not, it fetches the page from the server, adds it to the cache for future use, and returns it to the client that requested it.

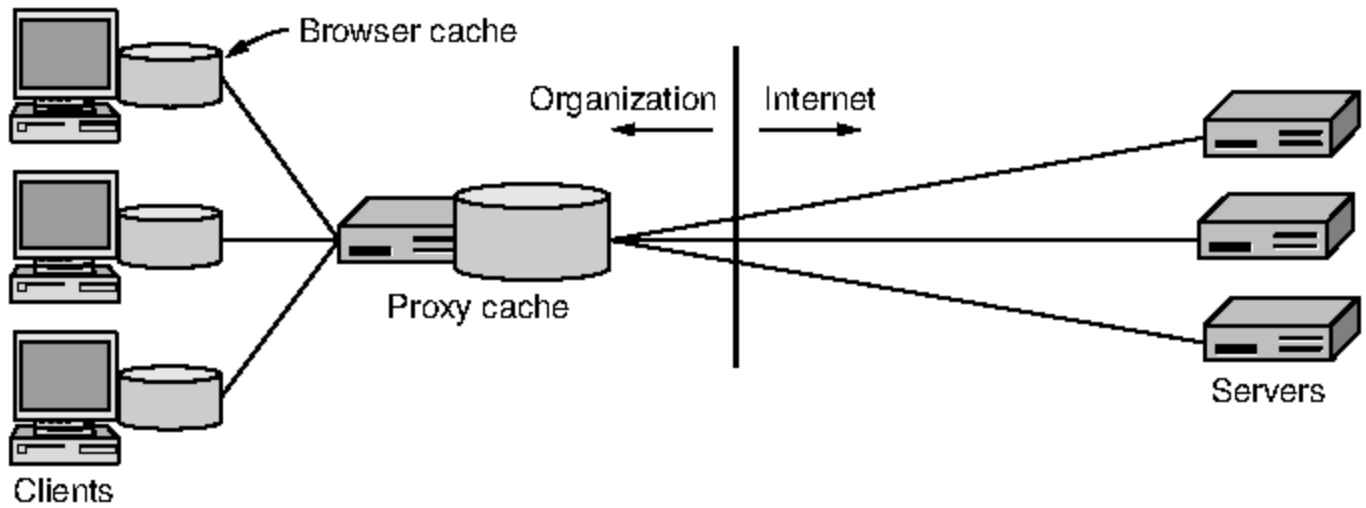


Figure 7-66. A proxy cache between Web browsers and Web servers.

As well as sending Web requests to the proxy instead of the real server, clients perform their own caching using its browser cache. The proxy is only consulted after the browser has tried to satisfy the request from its own cache. That is, the proxy provides a second level of caching.

Further proxies may be added to provide additional levels of caching. Each proxy (or browser) makes requests via its **upstream proxy**. Each upstream proxy caches for the **downstream proxies** (or browsers). Thus, it is possible for browsers in a company to use a company proxy, which uses an ISP proxy, which contacts Web servers directly. However, the single level of proxy caching we have shown in Fig. 7-66 is often sufficient to gain most of the potential benefits, in practice. The problem again is the long tail of popularity. Studies of Web traffic have shown that shared caching is especially beneficial until the number of users reaches about the size of a small company (say, 100 people). As the number of people grows larger, the benefits of sharing a cache become marginal because of the unpopular requests that cannot be cached due to lack of storage space (Wolman et al., 1999).

Web proxies provide additional benefits that are often a factor in the decision to deploy them. One benefit is to filter content. The administrator may configure

the proxy to blacklist sites or otherwise filter the requests that it makes. For example, many administrators frown on employees watching YouTube videos (or worse yet, pornography) on company time and set their filters accordingly. Another benefit of having proxies is privacy or anonymity, when the proxy shields the identity of the user from the server.

7.5.3 Content Delivery Networks

Server farms and Web proxies help to build large sites and to improve Web performance, but they are not sufficient for truly popular Web sites that must serve content on a global scale. For these sites, a different approach is needed.

CDNs (**Content Delivery Networks**) turn the idea of traditional Web caching on its head. Instead, of having clients look for a copy of the requested page in a nearby cache, it is the provider who places a copy of the page in a set of nodes at different locations and directs the client to use a nearby node as the server.

An example of the path that data follows when it is distributed by a CDN is shown in Fig. 7-67. It is a tree. The origin server in the CDN distributes a copy of the content to other nodes in the CDN, in Sydney, Boston, and Amsterdam, in this example. This is shown with dashed lines. Clients then fetch pages from the nearest node in the CDN. This is shown with solid lines. In this way, the clients in Sydney both fetch the page copy that is stored in Sydney; they do not both fetch the page from the origin server, which may be in Europe.

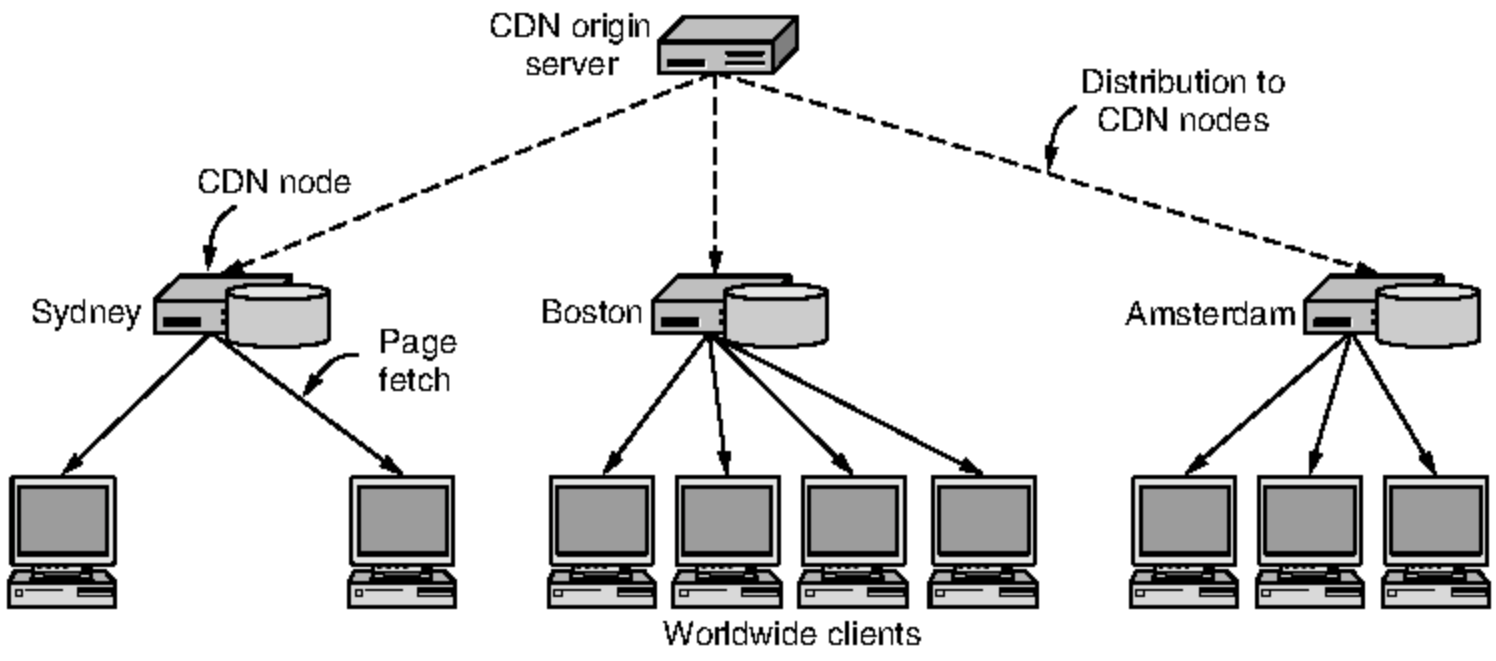


Figure 7-67. CDN distribution tree.

Using a tree structure has three virtues. First, the content distribution can be scaled up to as many clients as needed by using more nodes in the CDN, and more levels in the tree when the distribution among CDN nodes becomes the bottleneck. No matter how many clients there are, the tree structure is efficient. The origin server is not overloaded because it talks to the many clients via the tree

of CDN nodes; it does not have to answer each request for a page by itself. Second, each client gets good performance by fetching pages from a nearby server instead of a distant server. This is because the round-trip time for setting up a connection is shorter, TCP slow-start ramps up more quickly because of the shorter round-trip time, and the shorter network path is less likely to pass through regions of congestion in the Internet. Finally, the total load that is placed on the network is also kept at a minimum. If the CDN nodes are well placed, the traffic for a given page should pass over each part of the network only once. This is important because someone pays for network bandwidth, eventually.

The idea of using a distribution tree is straightforward. What is less simple is how to organize the clients to use this tree. For example, proxy servers would seem to provide a solution. Looking at Fig. 7-67, if each client was configured to use the Sydney, Boston or Amsterdam CDN node as a caching Web proxy, the distribution would follow the tree. However, this strategy falls short in practice, for three reasons. The first reason is that the clients in a given part of the network probably belong to different organizations, so they are probably using different Web proxies. Recall that caches are not usually shared across organizations because of the limited benefit of caching over a large number of clients, and for security reasons too. Second, there can be multiple CDNs, but each client uses only a single proxy cache. Which CDN should a client use as its proxy? Finally, perhaps the most practical issue of all is that Web proxies are configured by clients. They may or may not be configured to benefit content distribution by a CDN, and there is little that the CDN can do about it.

Another simple way to support a distribution tree with one level is to use **mirroring**. In this approach, the origin server replicates content over the CDN nodes as before. The CDN nodes in different network regions are called **mirrors**. The Web pages on the origin server contain explicit links to the different mirrors, usually telling the user their location. This design lets the user manually select a nearby mirror to use for downloading content. A typical use of mirroring is to place a large software package on mirrors located in, for example, the East and West coasts of the U.S., Asia, and Europe. Mirrored sites are generally completely static, and the choice of sites remains stable for months or years. They are a tried and tested technique. However, they depend on the user to do the distribution as the mirrors are really different Web sites, even if they are linked together.

The third approach, which overcomes the difficulties of the previous two approaches, uses DNS and is called **DNS redirection**. Suppose that a client wants to fetch a page with the URL *http://www.cdn.com/page.html*. To fetch the page, the browser will use DNS to resolve *www.cdn.com* to an IP address. This DNS lookup proceeds in the usual manner. By using the DNS protocol, the browser learns the IP address of the name server for *cdn.com*, then contacts the name server to ask it to resolve *www.cdn.com*. Now comes the really clever bit. The name server is run by the CDN. Instead, of returning the same IP address for each request, it will look at the IP address of the client making the request and return

different answers. The answer will be the IP address of the CDN node that is nearest the client. That is, if a client in Sydney asks the CDN name server to resolve *www.cdn.com*, the name server will return the IP address of the Sydney CDN node, but if a client in Amsterdam makes the same request, the name server will return the IP address of the Amsterdam CDN node instead.

This strategy is perfectly legal according to the semantics of DNS. We have previously seen that name servers may return changing lists of IP addresses. After the name resolution, the Sydney client will fetch the page directly from the Sydney CDN node. Further pages on the same "server" will be fetched directly from the Sydney CDN node as well because of DNS caching. The overall sequence of steps is shown in Fig. 7-68.

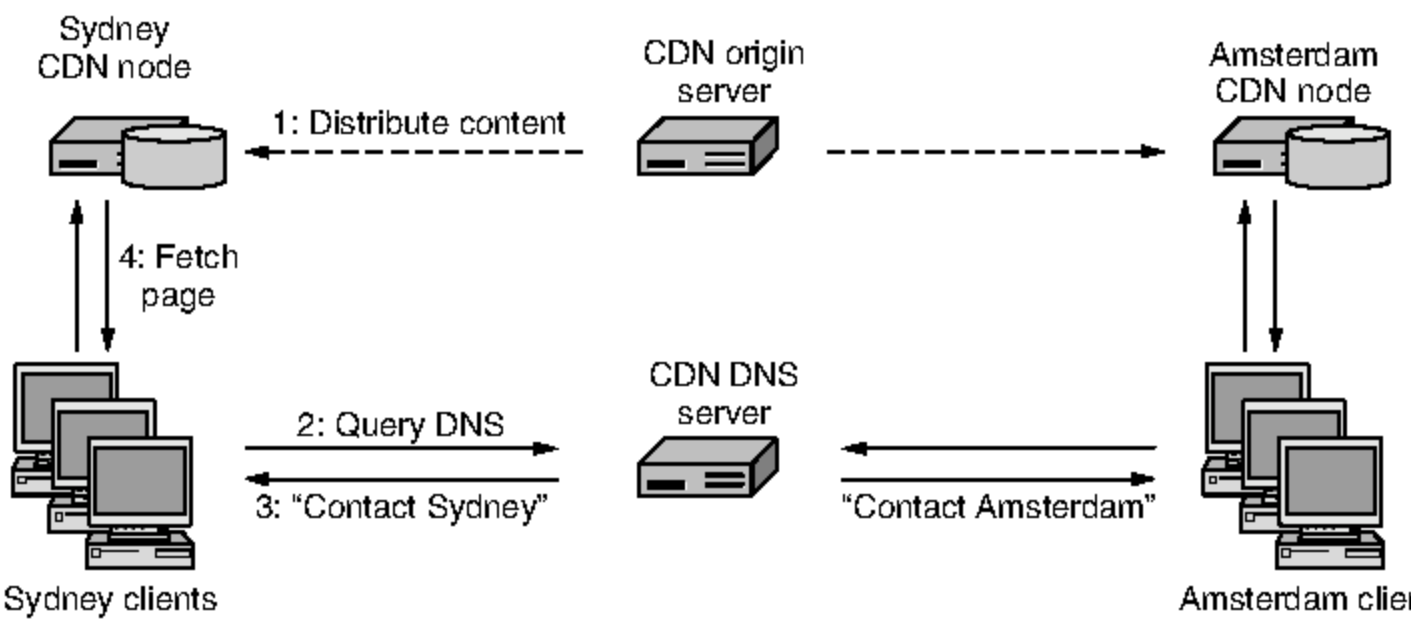


Figure 7-68. Directing clients to nearby CDN nodes using DNS.

A complex question in the above process is what it means to find the nearest CDN node, and how to go about it. To define nearest, it is not really geography that matters. There are at least two factors to consider in mapping a client to a CDN node. One factor is the network distance. The client should have a short and high-capacity network path to the CDN node. This situation will produce quick downloads. CDNs use a map they have previously computed to translate between the IP address of a client and its network location. The CDN node that is selected might be the one at the shortest distance as the crow flies, or it might not. What matters is some combination of the length of the network path and any capacity limits along it. The second factor is the load that is already being carried by the CDN node. If the CDN nodes are overloaded, they will deliver slow responses, just like the overloaded Web server that we sought to avoid in the first place. Thus, it may be necessary to balance the load across the CDN nodes, mapping some clients to nodes that are slightly further away but more lightly loaded.

The techniques for using DNS for content distribution were pioneered by Akamai starting in 1998, when the Web was groaning under the load of its early

growth. Akamai was the first major CDN and became the industry leader. Probably even more clever than the idea of using DNS to connect clients to nearby nodes was the incentive structure of their business. Companies pay Akamai to deliver their content to clients, so that they have responsive Web sites that customers like to use. The CDN nodes must be placed at network locations with good connectivity, which initially meant inside ISP networks. For the ISPs, there is a benefit to having a CDN node in their networks, namely that the CDN node cuts down the amount of upstream network bandwidth that they need (and must pay for), just as with proxy caches. In addition, the CDN node improves responsiveness for the ISP's customers, which makes the ISP look good in their eyes, giving them a competitive advantage over ISPs that do not have a CDN node. These benefits (at no cost to the ISP) makes installing a CDN node a no brainer for the ISP. Thus, the content provider, the ISP, and the customers all benefit and the CDN makes money. Since 1998, other companies have gotten into the business, so it is now a competitive industry with multiple providers.

As this description implies, most companies do not build their own CDN. Instead, they use the services of a CDN provider such as Akamai to actually deliver their content. To let other companies use the service of a CDN, we need to add one last step to our picture.

After the contract is signed for a CDN to distribute content on behalf of a Web site owner, the owner gives the CDN the content. This content is pushed to the CDN nodes. In addition, the owner rewrites any of its Web pages that link to the content. Instead of linking to the content on their Web site, the pages link to the content via the CDN. As an example of how this scheme works, consider the source code for Fluffy Video's Web page, given in Fig. 7-69(a). After preprocessing, it is transformed to Fig. 7-69(b) and placed on Fluffy Video's server as *www.fluffyvideo.com/index.html*.

When a user types in the URL *www.fluffyvideo.com* to his browser, DNS returns the IP address of Fluffy Video's own Web site, allowing the main (HTML) page to be fetched in the normal way. When the user clicks on any of the hyperlinks, the browser asks DNS to look up *www.cdn.com*. This lookup contacts the CDN's DNS server, which returns the IP address of the nearby CDN node. The browser then sends a regular HTTP request to the CDN node, for example, for */fluffyvideo/koalas.mpg*. The URL identifies the page to return, starting the path with *fluffyvideo* so that the CDN node can separate requests for the different companies that it serves. Finally, the video is returned and the user sees cute fluffy animals.

The strategy behind this split of content hosted by the CDN and entry pages hosted by the content owner is that it gives the content owner control while letting the CDN move the bulk of the data. Most entry pages are tiny, being just HTML text. These pages often link to large files, such as videos and images. It is precisely these large files that are served by the CDN, even though the use of a CDN is completely transparent to users. The site looks the same, but performs faster.

```
<html>
<head> <title> Fluffy Video </title> </head>
<body>
<h1> Fluffy Video's Product List </h1>
<p> Click below for free samples. </p>

<a href="koalas.mpg"> Koalas Today </a> <br>
<a href="kangaroos.mpg"> Funny Kangaroos </a> <br>
<a href="wombats.mpg"> Nice Wombats </a> <br>
</body>
</html>
```

(a)

```
<html>
<head> <title> Fluffy Video </title> </head>
<body>
<h1> Fluffy Video's Product List </h1>
<p> Click below for free samples. </p>

<a href="http://www.cdn.com/fluffyvideo/koalas.mpg"> Koalas Today </a> <br>
<a href="http://www.cdn.com/fluffyvideo/kangaroos.mpg"> Funny Kangaroos </a> <br>
<a href="http://www.cdn.com/fluffyvideo/wombats.mpg"> Nice Wombats </a> <br>
</body>
</html>
```

(b)

Figure 7-69. (a) Original Web page. (b) Same page after linking to the CDN.

There is another advantage for sites using a shared CDN. The future demand for a Web site can be difficult to predict. Frequently, there are surges in demand known as **flash crowds**. Such a surge may happen when the latest product is released, there is a fashion show or other event, or the company is otherwise in the news. Even a Web site that was a previously unknown, unvisited backwater can suddenly become the focus of the Internet if it is newsworthy and linked from popular sites. Since most sites are not prepared to handle massive increases in traffic, the result is that many of them crash when traffic surges.

Case in point. Normally the Florida Secretary of State's Web site is not a busy place, although you can look up information about Florida corporations, notaries, and cultural affairs, as well as information about voting and elections there. For some odd reason, on Nov. 7, 2000 (the date of the U.S. presidential election with Bush vs. Gore), a whole lot of people were suddenly interested in the election results page of this site. The site suddenly became one of the busiest Web sites in the world and naturally crashed as a result. If it had been using a CDN, it would probably have survived.

By using a CDN, a site has access to a very large content-serving capacity. The largest CDNs have tens of thousands of servers deployed in countries all over the world. Since only a small number of sites will be experiencing a flash crowd

at any one time (by definition), those sites may use the CDN's capacity to handle the load until the storm passes. That is, the CDN can quickly scale up a site's serving capacity.

The preceding discussion above is a simplified description of how Akamai works. There are many more details that matter in practice. The CDN nodes pictured in our example are normally clusters of machines. DNS redirection is done with two levels: one to map clients to the approximate network location, and another to spread the load over nodes in that location. Both reliability and performance are concerns. To be able to shift a client from one machine in a cluster to another, DNS replies at the second level are given with short TTLs so that the client will repeat the resolution after a short while. Finally, while we have concentrated on distributing static objects like images and videos, CDNs can also support dynamic page creation, streaming media, and more. For more information about CDNs, see Dilley et al. (2002).

7.5.4 Peer-to-Peer Networks

Not everyone can set up a 1000-node CDN at locations around the world to distribute their content. (Actually, it is not hard to rent 1000 virtual machines around the globe because of the well-developed and competitive hosting industry. However, setting up a CDN only starts with getting the nodes.) Luckily, there is an alternative for the rest of us that is simple to use and can distribute a tremendous amount of content. It is a P2P (Peer-to-Peer) network.

P2P networks burst onto the scene starting in 1999. The first widespread application was for mass crime: 50 million Napster users were exchanging copyrighted songs without the copyright owners' permission until Napster was shut down by the courts amid great controversy. Nevertheless, peer-to-peer technology has many interesting and legal uses. Other systems continued development, with such great interest from users that P2P traffic quickly eclipsed Web traffic. Today, BitTorrent is the most popular P2P protocol. It is used so widely to share (licensed and public domain) videos, as well as other content, that it accounts for a large fraction of all Internet traffic. We will look at it in this section.

The basic idea of a **P2P** (**Peer-to-Peer**) file-sharing network is that many computers come together and pool their resources to form a content distribution system. The computers are often simply home computers. They do not need to be machines in Internet data centers. The computers are called peers because each one can alternately act as a client to another peer, fetching its content, and as a server, providing content to other peers. What makes peer-to-peer systems interesting is that there is no dedicated infrastructure, unlike in a CDN. Everyone participates in the task of distributing content, and there is often no central point of control.

Many people are excited about P2P technology because it is seen as empowering the little guy. The reason is not only that it takes a large company to run a

CDN, while anyone with a computer can join a P2P network. It is that P2P networks have a formidable capacity to distribute content that can match the largest of Web sites.

Consider a P2P network made up of N average users, each with broadband connectivity at 1 Mbps. The aggregate upload capacity of the P2P network, or rate at which the users can send traffic into the Internet, is N Mbps. The download capacity, or rate at which the users can receive traffic, is also N Mbps. Each user can upload and download at the same time, too, because they have a 1-Mbps link in each direction.

It is not obvious that this should be true, but it turns out that all of the capacity can be used productively to distribute content, even for the case of sharing a single copy of a file with all the other users. To see how this can be so, imagine that the users are organized into a binary tree, with each non-leaf user sending to two other users. The tree will carry the single copy of the file to all the other users. To use the upload bandwidth of as many users as possible at all times (and hence distribute the large file with low latency), we need to pipeline the network activity of the users. Imagine that the file is divided into 1000 pieces. Each user can receive a new piece from somewhere up the tree and send the previously received piece down the tree at the same time. This way, once the pipeline is started, after a small number of pieces (equal to the depth of the tree) are sent, all non-leaf users will be busy uploading the file to other users. Since there are approximately $N/2$ non-leaf users, the upload bandwidth of this tree is $N/2$ Mbps. We can repeat this trick and create another tree that uses the other $N/2$ Mbps of upload bandwidth by swapping the roles of leaf and non-leaf nodes. Together, this construction uses all of the capacity.

This argument means that P2P networks are self-scaling. Their usable upload capacity grows in tandem with the download demands that can be made by their users. They are always "large enough" in some sense, without the need for any dedicated infrastructure. In contrast, the capacity of even a large Web site is fixed and will either be too large or too small. Consider a site with only 100 clusters, each capable of 10 Gbps. This enormous capacity does not help when there are a small number of users. The site cannot get information to N users at a rate faster than N Mbps because the limit is at the users and not the Web site. And when there are more than one million 1-Mbps users, the Web site cannot pump out data fast enough to keep all the users busy downloading. That may seem like a large number of users, but large BitTorrent networks (e.g., Pirate Bay) claim to have more than 10,000,000 users. That is more like 10 terabits/sec in terms of our example!

You should take these back-of-the-envelope numbers with a grain (or better yet, a metric ton) of salt because they oversimplify the situation. A significant challenge for P2P networks is to use bandwidth well when users can come in all shapes and sizes, and have different download and upload capacities. Nevertheless, these numbers do indicate the enormous potential of P2P.

There is another reason that P2P networks are important. CDNs and other centrally run services put the providers in a position of having a trove of personal information about many users, from browsing preferences and where people shop online, to people's locations and email addresses. This information can be used to provide better, more personalized service, or it can be used to intrude on people's privacy. The latter may happen either intentionally—say as part of a new product—or through an accidental disclosure or compromise. With P2P systems, there can be no single provider that is capable of monitoring the entire system. This does not mean that P2P systems will necessarily provide privacy, as users are trusting each other to some extent. It only means that they can provide a different form of privacy than centrally managed systems. P2P systems are now being explored for services beyond file sharing (e.g., storage, streaming), and time will tell whether this advantage is significant.

P2P technology has followed two related paths as it has been developed. On the more practical side, there are the systems that are used every day. The most well known of these systems are based on the BitTorrent protocol. On the more academic side, there has been intense interest in DHT (Distributed Hash Table) algorithms that let P2P systems perform well as a whole, yet rely on no centralized components at all. We will look at both of these technologies.

BitTorrent

The BitTorrent protocol was developed by Brahm Cohen in 2001 to let a set of peers share files quickly and easily. There are dozens of freely available clients that speak this protocol, just as there are many browsers that speak the HTTP protocol to Web servers. The protocol is available as an open standard at *www.bittorrent.org*.

In a typical peer-to-peer system, like that formed with BitTorrent, the users each have some information that may be of interest to other users. This information may be free software, music, videos, photographs, and so on. There are three problems that need to be solved to share content in this setting:

1. How does a peer find other peers that have the content it wants to download?
2. How is content replicated by peers to provide high-speed downloads for everyone?
3. How do peers encourage each other to upload content to others as well as download content for themselves?

The first problem exists because not all peers will have all of the content, at least initially. The approach taken in BitTorrent is for every content provider to create a content description called a **torrent**. The torrent is much smaller than the

content, and is used by a peer to verify the integrity of the data that it downloads from other peers. Other users who want to download the content must first obtain the torrent, say, by finding it on a Web page advertising the content.

The torrent is just a file in a specified format that contains two key kinds of information. One kind is the name of a **tracker**, which is a server that leads peers to the content of the torrent. The other kind of information is a list of equal-sized pieces, or **chunks**, that make up the content. Different chunk sizes can be used for different torrents, typically 64 KB to 512 KB. The torrent file contains the name of each chunk, given as a 160-bit SHA-1 hash of the chunk. We will cover cryptographic hashes such as SHA-1 in Chap. 8. For now, you can think of a hash as a longer and more secure checksum. Given the size of chunks and hashes, the torrent file is at least three orders of magnitude smaller than the content, so it can be transferred quickly.

To download the content described in a torrent, a peer first contacts the tracker for the torrent. The **tracker** is a server that maintains a list of all the other peers that are actively downloading and uploading the content. This set of peers is called a **swarm**. The members of the swarm contact the tracker regularly to report that they are still active, as well as when they leave the swarm. When a new peer contacts the tracker to join the swarm, the tracker tells it about other peers in the swarm. Getting the torrent and contacting the tracker are the first two steps for downloading content, as shown in Fig. 7-70.

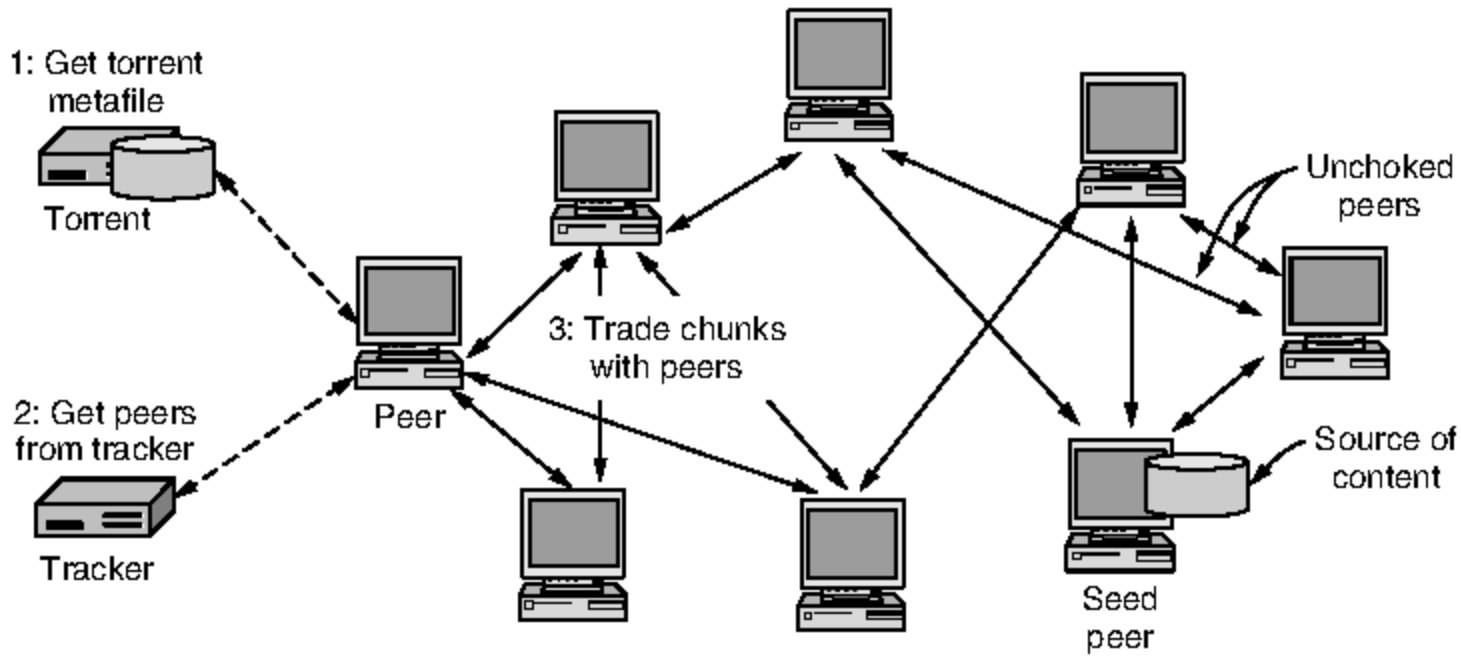


Figure 7-70. BitTorrent.

The second problem is how to share content in a way that gives rapid downloads. When a swarm is first formed, some peers must have all of the chunks that make up the content. These peers are called **seeders**. Other peers that join the swarm will have no chunks; they are the peers that are downloading the content.

While a peer participates in a swarm, it simultaneously downloads chunks that it is missing from other peers, and uploads chunks that it has to other peers who

need them. This trading is shown as the last step of content distribution in Fig. 7-70. Over time, the peer gathers more chunks until it has downloaded all of the content. The peer can leave the swarm (and return) at any time. Normally a peer will stay for a short period after finishes its own download. With peers coming and going, the rate of churn in a swarm can be quite high.

For the above method to work well, each chunk should be available at many peers. If everyone were to get the chunks in the same order, it is likely that many peers would depend on the seeders for the next chunk. This would create a bottleneck. Instead, peers exchange lists of the chunks they have with each other. Then they select rare chunks that are hard to find to download. The idea is that downloading a rare chunk will make a copy of it, which will make the chunk easier for other peers to find and download. If all peers do this, after a short while all chunks will be widely available.

The third problem is perhaps the most interesting. CDN nodes are set up exclusively to provide content to users. P2P nodes are not. They are users' computers, and the users may be more interested in getting a movie than helping other users with their downloads. Nodes that take resources from a system without contributing in kind are called **free-riders** or **leechers**. If there are too many of them, the system will not function well. Earlier P2P systems were known to host them (Saroiu et al., 2003) so BitTorrent sought to minimize them.

The approach taken in BitTorrent clients is to reward peers who show good upload behavior. Each peer randomly samples the other peers, retrieving chunks from them while it uploads chunks to them. The peer continues to trade chunks with only a small number of peers that provide the highest download performance, while also randomly trying other peers to find good partners. Randomly trying peers also allows newcomers to obtain initial chunks that they can trade with other peers. The peers with which a node is currently exchanging chunks are said to be **unchoked**.

Over time, this algorithm is intended to match peers with comparable upload and download rates with each other. The more a peer is contributing to the other peers, the more it can expect in return. Using a set of peers also helps to saturate a peer's download bandwidth for high performance. Conversely, if a peer is not uploading chunks to other peers, or is doing so very slowly, it will be cut off, or **choked**, sooner or later. This strategy discourages antisocial behavior in which peers free-ride on the swarm.

The choking algorithm is sometimes described as implementing the **tit-for-tat** strategy that encourages cooperation in repeated interactions. However, it does not prevent clients from gaming the system in any strong sense (Piatek et al., 2007). Nonetheless, attention to the issue and mechanisms that make it more difficult for casual users to free-ride have likely contributed to the success of BitTorrent.

As you can see from our discussion, BitTorrent comes with a rich vocabulary. There are torrents, swarms, leechers, seeders, and trackers, as well as snubbing,

choking, lurking, and more. For more information see the short paper on BitTorrent (Cohen, 2003) and look on the Web starting with *www.bittorrent.org*.

DHTs—Distributed Hash Tables

The emergence of P2P file sharing networks around 2000 sparked much interest in the research community. The essence of P2P systems is that they avoid the centrally managed structures of CDNs and other systems. This can be a significant advantage. Centrally managed components become a bottleneck as the system grows very large and are a single point of failure. Central components can also be used as a point of control (e.g., to shut off the P2P network). However, the early P2P systems were only partly decentralized, or, if they were fully decentralized, they were inefficient.

The traditional form of BitTorrent that we just described uses peer-to-peer transfers and a centralized tracker for each swarm. It is the tracker that turns out to be the hard part to decentralize in a peer-to-peer system. The key problem is how to find out which peers have specific content that is being sought. For example, each user might have one or more data items such as songs, photographs, programs, files, and so on that other users might want to read. How do the other users find them? Making one index of who has what is simple, but it is centralized. Having every peer keep its own index does not help. True, it is distributed. However, it requires so much work to keep the indexes of all peers up to date (as content is moved about the system) that it is not worth the effort.

The question tackled by the research community was whether it was possible to build P2P indexes that were entirely distributed but performed well. By perform well, we mean three things. First, each node keeps only a small amount of information about other nodes. This property means that it will not be expensive to keep the index up to date. Second, each node can look up entries in the index quickly. Otherwise, it is not a very useful index. Third, each node can use the index at the same time, even as other nodes come and go. This property means the performance of the index grows with the number of nodes.

The answer is to the question was: "Yes." Four different solutions were invented in 2001. They are Chord (Stoica et al., 2001), CAN (Ratnasamy et al., 2001), Pastry (Rowstron and Druschel, 2001), and Tapestry (Zhao et al., 2004). Other solutions were invented soon afterwards, including Kademlia, which is used in practice (Maymounkov and Mazieres, 2002). The solutions are known as **DHTs** (**Distributed Hash Tables**) because the basic functionality of an index is to map a key to a value. This is like a hash table, and the solutions are distributed versions, of course.

DHTs do their work by imposing a regular structure on the communication between the nodes, as we will see. This behavior is quite different than that of traditional P2P networks that use whatever connections peers happen to make.

For this reason, DHTs are called **structured P2P networks**. Traditional P2P protocols build **unstructured P2P networks**.

The DHT solution that we will describe is Chord. As a scenario, consider how to replace the centralized tracker traditionally used in BitTorrent with a fully-distributed tracker. Chord can be used to solve this problem. In this scenario, the overall index is a listing of all of the swarms that a computer may join to download content. The key used to look up the index is the torrent description of the content. It uniquely identifies a swarm from which content can be downloaded as the hashes of all the content chunks. The value stored in the index for each key is the list of peers that comprise the swarm. These peers are the computers to contact to download the content. A person wanting to download content such as a movie has only the torrent description. The question the DHT must answer is how, lacking a central database, does a person find out which peers (out of the millions of BitTorrent nodes) to download the movie from?

A Chord DHT consists of n participating nodes. They are nodes running BitTorrent in our scenario. Each node has an IP address by which it may be contacted. The overall index is spread across the nodes. This implies that each node stores bits and pieces of the index for use by other nodes. The key part of Chord is that it navigates the index using identifiers in a virtual space, not the IP addresses of nodes or the names of content like movies. Conceptually, the identifiers are simply m-bit numbers that can be arranged in ascending order into a ring.

To turn a node address into an identifier, it is mapped to an m-bit number using a hash function, *hash*. Chord uses SHA-1 for *hash*. This is the same hash that we mentioned when describing BitTorrent. We will look at it when we discuss cryptography in Chap. 8. For now, suffice it to say that it is just a function that takes a variable-length byte string as an argument and produces a highly random 160-bit number. Thus, we can use it to convert any IP address to a 160-bit number called the **node identifier**.

In Fig. 7-71(a), we show the node identifier circle for $m = 5$. (Just ignore the arcs in the middle for the moment.) Some of the identifiers correspond to nodes, but most do not. In this example, the nodes with identifiers 1, 4, 7, 12, 15, 20, and 27 correspond to actual nodes and are shaded in the figure; the rest do not exist.

Let us now define the function *successor*(k) as the node identifier of the first actual node following k around the circle, clockwise. For example, *successor* (6) = 7, *successor* (8) = 12, and *successor* (22) = 27.

A **key** is also produced by hashing a content name with *hash* (i.e., SHA-1) to generate a 160-bit number. In our scenario, the content name is the torrent. Thus, in order to convert *torrent* (the torrent description file) to its key, we compute *key* = *hash*(*torrent*). This computation is just a local procedure call to *hash*.

To start a new a swarm, a node needs to insert a new key-value pair consisting of (*torrent*, *my-IP-address*) into the index. To accomplish this, the node asks *successor*(*hash*(*torrent*)) to store *my-IP-address*. In this way, the index is distributed over the nodes at random. For fault tolerance, p different hash functions

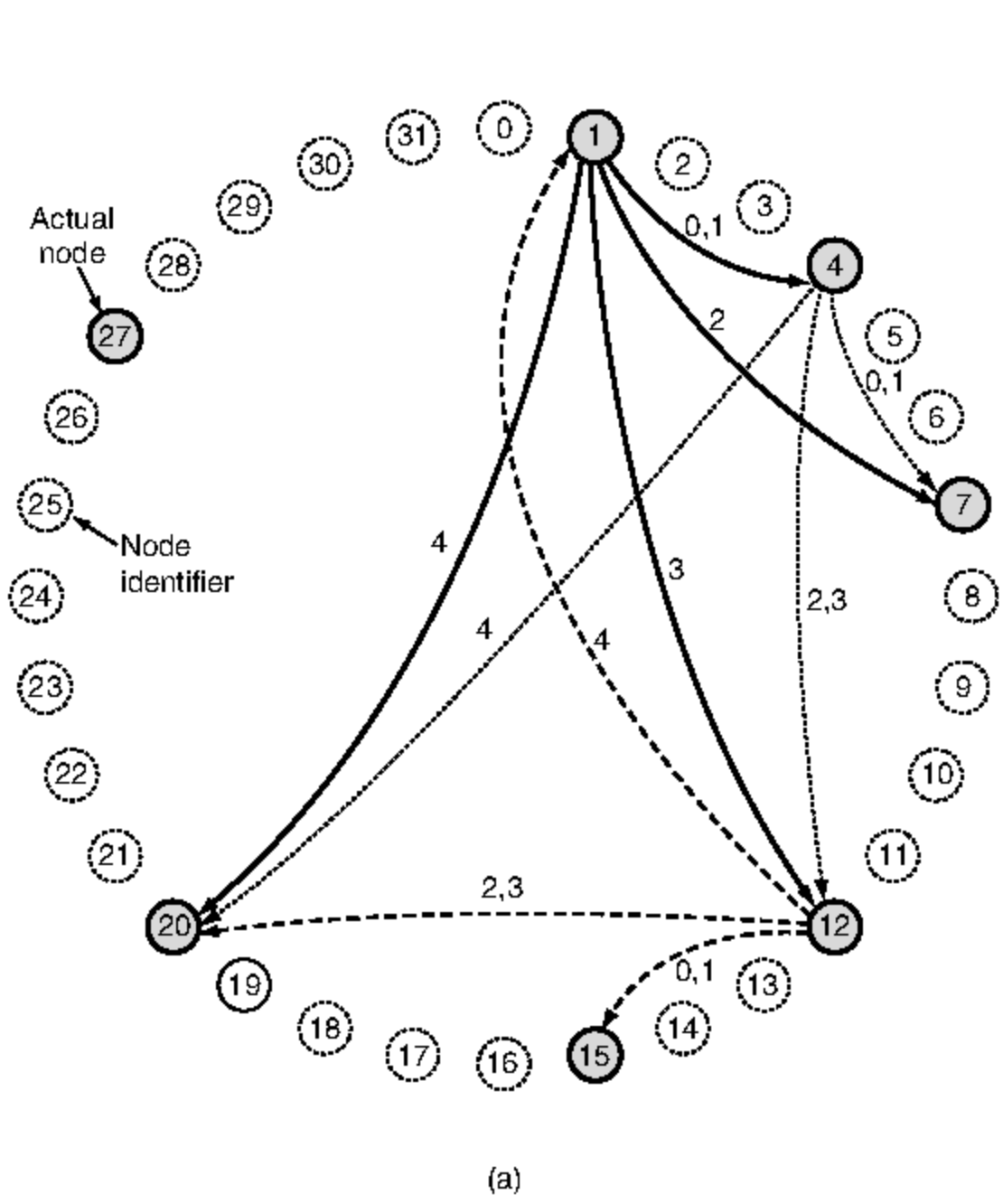


Figure 7-71. (a) A set of 32 node identifiers arranged in a circle. The shaded ones correspond to actual machines. The arcs show the fingers from nodes 1, 4, and 12. The labels on the arcs are the table indices. (b) Examples of the finger tables.

could be used to store the data at p nodes, but we will not consider the subject of fault tolerance further here.

Some time after the DHT is constructed, another node wants to find a torrent so that it can join the swarm and download content. A node looks up *torrent* by first hashing it to get *key*, and second using *successor* (*key*) to find the IP address of the node storing the corresponding value. The value is the list of peers in the swarm; the node can add its IP address to the list and contact the other peers to download content with the BitTorrent protocol.

The first step is easy; the second one is not easy. To make it possible to find the IP address of the node corresponding to a certain key, each node is required to

maintain certain administrative data structures. One of these is the IP address of its successor node along the node identifier circle. For example, in Fig. 7-71, node 4's successor is 7 and node 7's successor is 12.

Lookup can now proceed as follows. The requesting node sends a packet to its successor containing its IP address and the key it is looking for. The packet is propagated around the ring until it locates the successor to the node identifier being sought. That node checks to see if it has any information matching the key, and if so, returns it directly to the requesting node, whose IP address it has.

However, linearly searching all the nodes is very inefficient in a large peer-to-peer system since the mean number of nodes required per search is $n/2$. To greatly speed up the search, each node also maintains what Chord calls a **finger table**. The finger table has m entries, indexed by 0 through $m-1$, each one pointing to a different actual node. Each of the entries has two fields: *start* and the IP address of *successor*(*start*), as shown for three example nodes in Fig. 7-71(b). The values of the fields for entry i at a node with identifier k are:

$start = k + 2^i \ \ (modulo \ \ 2^m)$
IP address of *successor*(*start* [i])

Note that each node stores the IP addresses of a relatively small number of nodes and that most of these are fairly close by in terms of node identifier.

Using the finger table, the lookup of *key* at node k proceeds as follows. If *key* falls between k and *successor*(k), the node holding information about *key* is *successor*(k) and the search terminates. Otherwise, the finger table is searched to find the entry whose *start* field is the closest predecessor of *key*. A request is then sent directly to the IP address in that finger table entry to ask it to continue the search. Since it is closer to *key* but still below it, chances are good that it will be able to return the answer with only a small number of additional queries. In fact, since every lookup halves the remaining distance to the target, it can be shown that the average number of lookups is $\log_2 n$.

As a first example, consider looking up $key = 3$ at node 1. Since node 1 knows that 3 lies between it and its successor, 4, the desired node is 4 and the search terminates, returning node 4's IP address.

As a second example, consider looking up $key = 16$ at node 1. Since 16 does not lie between 1 and 4, the finger table is consulted. The closest predecessor to 16 is 9, so the request is forwarded to the IP address of 9's entry, namely, that of node 12. Node 12 also does not know the answer itself, so it looks for the node most closely preceding 16 and finds 14, which yields the IP address of node 15. A query is then sent there. Node 15 observes that 16 lies between it and its successor (20), so it returns the IP address of 20 to the caller, which works its way back to node 1.

Since nodes join and leave all the time, Chord needs a way to handle these operations. We assume that when the system began operation it was small enough that the nodes could just exchange information directly to build the first circle and

finger tables. After that, an automated procedure is needed. When a new node, r, wants to join, it must contact some existing node and ask it to look up the IP address of *successor*(r) for it. Next, the new node then asks *successor*(r) for its predecessor. The new node then asks both of these to insert r in between them in the circle. For example, if 24 in Fig. 7-71 wants to join, it asks any node to look up *successor*(24), which is 27. Then it asks 27 for its predecessor (20). After it tells both of those about its existence, 20 uses 24 as its successor and 27 uses 24 as its predecessor. In addition, node 27 hands over those keys in the range 21–24, which now belong to 24. At this point, 24 is fully inserted.

However, many finger tables are now wrong. To correct them, every node runs a background process that periodically recomputes each finger by calling *successor*. When one of these queries hits a new node, the corresponding finger entry is updated.

When a node leaves gracefully, it hands its keys over to its successor and informs its predecessor of its departure so the predecessor can link to the departing node's successor. When a node crashes, a problem arises because its predecessor no longer has a valid successor. To alleviate this problem, each node keeps track not only of its direct successor but also its s direct successors, to allow it to skip over up to $s - 1$ consecutive failed nodes and reconnect the circle if disaster strikes.

There has been a tremendous amount of research on DHTs since they were invented. To give you an idea of just how much research, let us pose a question: what is the most-cited networking paper of all time? You will find it difficult to come up with a paper that is cited more than the seminal Chord paper (Stoica et al., 2001). Despite this veritable mountain of research, applications of DHTs are only slowly beginning to emerge. Some BitTorrent clients use DHTs to provide a fully distributed tracker of the kind that we described. Large commercial cloud services such as Amazon's Dynamo also incorporate DHT techniques (DeCandia et al., 2007).

7.6 SUMMARY

Naming in the ARPANET started out in a very simple way: an ASCII text file listed the names of all the hosts and their corresponding IP addresses. Every night all the machines downloaded this file. But when the ARPANET morphed into the Internet and exploded in size, a far more sophisticated and dynamic naming scheme was required. The one used now is a hierarchical scheme called the Domain Name System. It organizes all the machines on the Internet into a set of trees. At the top level are the well-known generic domains, including *com* and *edu*, as well as about 200 country domains. DNS is implemented as a distributed database with servers all over the world. By querying a DNS server, a process

can map an Internet domain name onto the IP address used to communicate with a computer for that domain.

Email is the original killer app of the Internet. It is still widely used by everyone from small children to grandparents. Most email systems in the world use the mail system now defined in RFCs 5321 and 5322. Messages have simple ASCII headers, and many kinds of content can be sent using MIME. Mail is submitted to message transfer agents for delivery and retrieved from them for presentation by a variety of user agents, including Web applications. Submitted mail is delivered using SMTP, which works by making a TCP connection from the sending message transfer agent to the receiving one.

The Web is the application that most people think of as being the Internet. Originally, it was a system for seamlessly linking hypertext pages (written in HTML) across machines. The pages are downloaded by making a TCP connection from the browser to a server and using HTTP. Nowadays, much of the content on the Web is produced dynamically, either at the server (e.g., with PHP) or in the browser (e.g., with JavaScript). When combined with back-end databases, dynamic server pages allow Web applications such as e-commerce and search. Dynamic browser pages are evolving into full-featured applications, such as email, that run inside the browser and use the Web protocols to communicate with remote servers.

Caching and persistent connections are widely used to enhance Web performance. Using the Web on mobile devices can be challenging, despite the growth in the bandwidth and processing power of mobiles. Web sites often send tailored versions of pages with smaller images and less complex navigation to devices with small displays.

The Web protocols are increasingly being used for machine-to-machine communication. XML is preferred to HTML as a description of content that is easy for machines to process. SOAP is an RPC mechanism that sends XML messages using HTTP.

Digital audio and video have been key drivers for the Internet since 2000. The majority of Internet traffic today is video. Much of it is streamed from Web sites over a mix of protocols (including RTP/UDP and RTP/HTTP/TCP). Live media is streamed to many consumers. It includes Internet radio and TV stations that broadcast all manner of events. Audio and video are also used for real-time conferencing. Many calls use voice over IP, rather than the traditional telephone network, and include videoconferencing.

There are a small number of tremendously popular Web sites, as well as a very large number of less popular ones. To serve the popular sites, content distribution networks have been deployed. CDNs use DNS to direct clients to a nearby server; the servers are placed in data centers all around the world. Alternatively, P2P networks let a collection of machines share content such as movies among themselves. They provide a content distribution capacity that scales with the number of machines in the P2P network and which can rival the largest of sites.

PROBLEMS

1. Many business computers have three distinct and worldwide unique identifiers. What are they?
2. In Fig. 7-4, there is no period after *laserjet*. Why not?
3. Consider a situation in which a cyberterrorist makes all the DNS servers in the world crash simultaneously. How does this change one's ability to use the Internet?
4. DNS uses UDP instead of TCP. If a DNS packet is lost, there is no automatic recovery. Does this cause a problem, and if so, how is it solved?
5. John wants to have an original domain name and uses a randomized program to generate a secondary domain name for him. He wants to register this domain name in the *com* generic domain. The domain name that was generated is 253 characters long. Will the *com* registrar allow this domain name to be registered?
6. Can a machine with a single DNS name have multiple IP addresses? How could this occur?
7. The number of companies with a Web site has grown explosively in recent years. As a result, thousands of companies are registered in the *com* domain, causing a heavy load on the top-level server for this domain. Suggest a way to alleviate this problem without changing the naming scheme (i.e., without introducing new top-level domain names). It is permitted that your solution requires changes to the client code.
8. Some email systems support a *Content Return:* header field. It specifies whether the body of a message is to be returned in the event of nondelivery. Does this field belong to the envelope or to the header?
9. Electronic mail systems need directories so people's email addresses can be looked up. To build such directories, names should be broken up into standard components (e.g., first name, last name) to make searching possible. Discuss some problems that must be solved for a worldwide standard to be acceptable.
10. A large law firm, which has many employees, provides a single email address for each employee. Each employee's email address is *<login>@lawfirm.com*. However, the firm did not explicitly define the format of the login. Thus, some employees use their first names as their login names, some use their last names, some use their initials, etc. The firm now wishes to make a fixed format, for example:

    firstname.lastname@lawfirm.com,

    that can be used for the email addresses of all its employees. How can this be done without rocking the boat too much?
11. A binary file is 4560 bytes long. How long will it be if encoded using base64 encoding, with a CR+LF pair inserted after every 110 bytes sent and at the end?
12. Name five MIME types not listed in this book. You can check your browser or the Internet for information.

13. Suppose that you want to send an MP3 file to a friend, but your friend's ISP limits the size of each incoming message to 1 MB and the MP3 file is 4 MB. Is there a way to handle this situation by using RFC 5322 and MIME?

14. Suppose that John just set up an auto-forwarding mechanism on his work email address, which receives all of his business-related emails, to forward them to his personal email address, which he shares with his wife. John's wife was unaware of this, and activated a vacation agent on their personal account. Because John forwarded his email, he did not set up a vacation daemon on his work machine. What happens when an email is received at John's work email address?

15. In any standard, such as RFC 5322, a precise grammar of what is allowed is needed so that different implementations can interwork. Even simple items have to be defined carefully. The SMTP headers allow white space between the tokens. Give *two* plausible alternative definitions of white space between tokens.

16. Is the vacation agent part of the user agent or the message transfer agent? Of course, it is set up using the user agent, but does the user agent actually send the replies? Explain your answer.

17. In a simple version of the Chord algorithm for peer-to-peer lookup, searches do not use the finger table. Instead, they are linear around the circle, in either direction. Can a node accurately predict which direction it should search in? Discuss your answer.

18. IMAP allows users to fetch and download email from a remote mailbox. Does this mean that the internal format of mailboxes has to be standardized so any IMAP program on the client side can read the mailbox on any mail server? Discuss your answer.

19. Consider the Chord circle of Fig. 7-71. Suppose that node 18 suddenly goes online. Which of the finger tables shown in the figure are affected? how?

20. Does Webmail use POP3, IMAP, or neither? If one of these, why was that one chosen? If neither, which one is it closer to in spirit?

21. When Web pages are sent out, they are prefixed by MIME headers. Why?

22. Is it possible that when a user clicks on a link with Firefox, a particular helper is started, but clicking on the same link in Internet Explorer causes a completely different helper to be started, even though the MIME type returned in both cases is identical? Explain your answer.

23. Although it was not mentioned in the text, an alternative form for a URL is to use the IP address instead of its DNS name. Use this information to explain why a DNS name cannot end with a digit.

24. Imagine that someone in the math department at Stanford has just written a new document including a proof that he wants to distribute by FTP for his colleagues to review. He puts the program in the FTP directory *ftp/pub/forReview/newProof.pdf*. What is the URL for this program likely to be?

25. In Fig. 7-22, *www.aportal.com* keeps track of user preferences in a cookie. A disadvantage of this scheme is that cookies are limited to 4 KB, so if the preferences are

extensive, for example, many stocks, sports teams, types of news stories, weather for multiple cities, specials in numerous product categories, and more, the 4-KB limit may be reached. Design an alternative way to keep track of preferences that does not have this problem.

26. Sloth Bank wants to make online banking easy for its lazy customers, so after a customer signs up and is authenticated by a password, the bank returns a cookie containing a customer ID number. In this way, the customer does not have to identify himself or type a password on future visits to the online bank. What do you think of this idea? Will it work? Is it a good idea?

27. (a) Consider the following HTML tag:

```
<h1 title="this is the header"> HEADER 1 </h1>
```

Under what conditions does the browser use the *TITLE* attribute, and how?
(b) How does the *TITLE* attribute differ from the *ALT* attribute?

28. How do you make an image clickable in HTML? Give an example.

29. Write an HTML page that includes a link to the email address *username@DomainName.com*. What happens when a user clicks this link?

30. Write an XML page for a university registrar listing multiple students, each having a name, an address, and a GPA.

31. For each of the following applications, tell whether it would be (1) possible and (2) better to use a PHP script or JavaScript, and why:
(a) Displaying a calendar for any requested month since September 1752.
(b) Displaying the schedule of flights from Amsterdam to New York.
(c) Graphing a polynomial from user-supplied coefficients.

32. Write a program in JavaScript that accepts an integer greater than 2 and tells whether it is a prime number. Note that JavaScript has if and while statements with the same syntax as C and Java. The modulo operator is %. If you need the square root of x, use *Math.sqrt*(x).

33. An HTML page is as follows:

```
<html> <body>
<a href="www.info-source.com/welcome.html"> Click here for info </a>
</body> </html>
```

If the user clicks on the hyperlink, a TCP connection is opened and a series of lines is sent to the server. List all the lines sent.

34. The *If-Modified-Since* header can be used to check whether a cached page is still valid. Requests can be made for pages containing images, sound, video, and so on, as well as HTML. Do you think the effectiveness of this technique is better or worse for JPEG images as compared to HTML? Think carefully about what "effectiveness'' means and explain your answer.

35. On the day of a major sporting event, such as the championship game in some popular sport, many people go to the official Web site. Is this a flash crowd in the same sense as the 2000 Florida presidential election? Why or why not?

36. Does it make sense for a single ISP to function as a CDN? If so, how would that work? If not, what is wrong with the idea?

37. Assume that compression is not used for audio CDs. How many MB of data must the compact disc contain in order to be able to play two hours of music?

38. In Fig. 7-42(c), quantization noise occurs due to the use of 4-bit samples to represent nine signal values. The first sample, at 0, is exact, but the next few are not. What is the percent error for the samples at 1/32, 2/32, and 3/32 of the period?

39. Could a psychoacoustic model be used to reduce the bandwidth needed for Internet telephony? If so, what conditions, if any, would have to be met to make it work? If not, why not?

40. An audio streaming server has a one-way "distance" of 100 msec to a media player. It outputs at 1 Mbps. If the media player has a 2-MB buffer, what can you say about the position of the low-water mark and the high-water mark?

41. Does voice over IP have the same problems with firewalls that streaming audio does? Discuss your answer.

42. What is the bit rate for transmitting uncompressed 1200×800 pixel color frames with 16 bits/pixel at 50 frames/sec?

43. Can a 1-bit error in an MPEG frame affect more than the frame in which the error occurs? Explain your answer.

44. Consider a 50,000-customer video server, where each customer watches three movies per month. Two-thirds of the movies are served at 9 P.M. How many movies does the server have to transmit at once during this time period? If each movie requires 6 Mbps, how many OC-12 connections does the server need to the network?

45. Suppose that Zipf's law holds for accesses to a 10,000-movie video server. If the server holds the most popular 1000 movies in memory and the remaining 9000 on disk, give an expression for the fraction of all references that will be to memory. Write a little program to evaluate this expression numerically.

46. Some cybersquatters have registered domain names that are misspellings of common corporate sites, for example, *www.microsfot.com*. Make a list of at least five such domains.

47. Numerous people have registered DNS names that consist of *www.word.com*, where *word* is a common word. For each of the following categories, list five such Web sites and briefly summarize what it is (e.g., *www.stomach.com* belongs to a gastroenterologist on Long Island). Here is the list of categories: animals, foods, household objects, and body parts. For the last category, please stick to body parts above the waist.

48. Rewrite the server of Fig. 6-6 as a true Web server using the *GET* command for HTTP 1.1. It should also accept the *Host* message. The server should maintain a cache of files recently fetched from the disk and serve requests from the cache when possible.

8

NETWORK SECURITY

For the first few decades of their existence, computer networks were primarily used by university researchers for sending email and by corporate employees for sharing printers. Under these conditions, security did not get a lot of attention. But now, as millions of ordinary citizens are using networks for banking, shopping, and filing their tax returns, and weakness after weakness has been found, network security has become a problem of massive proportions. In this chapter, we will study network security from several angles, point out numerous pitfalls, and discuss many algorithms and protocols for making networks more secure.

Security is a broad topic and covers a multitude of sins. In its simplest form, it is concerned with making sure that nosy people cannot read, or worse yet, secretly modify messages intended for other recipients. It is concerned with people trying to access remote services that they are not authorized to use. It also deals with ways to tell whether that message purportedly from the IRS "Pay by Friday, or else" is really from the IRS and not from the Mafia. Security also deals with the problems of legitimate messages being captured and replayed, and with people later trying to deny that they sent certain messages.

Most security problems are intentionally caused by malicious people trying to gain some benefit, get attention, or harm someone. A few of the most common perpetrators are listed in Fig. 8-1. It should be clear from this list that making a network secure involves a lot more than just keeping it free of programming errors. It involves outsmarting often intelligent, dedicated, and sometimes well-funded adversaries. It should also be clear that measures that will thwart casual

attackers will have little impact on the serious ones. Police records show that the most damaging attacks are not perpetrated by outsiders tapping a phone line but by insiders bearing a grudge. Security systems should be designed accordingly.

Adversary	Goal
Student	To have fun snooping on people's email
Cracker	To test out someone's security system; steal data
Sales rep	To claim to represent all of Europe, not just Andorra
Corporation	To discover a competitor's strategic marketing plan
Ex-employee	To get revenge for being fired
Accountant	To embezzle money from a company
Stockbroker	To deny a promise made to a customer by email
Identity thief	To steal credit card numbers for sale
Government	To learn an enemy's military or industrial secrets
Terrorist	To steal biological warfare secrets

Figure 8-1. Some people who may cause security problems, and why.

Network security problems can be divided roughly into four closely intertwined areas: secrecy, authentication, nonrepudiation, and integrity control. Secrecy, also called confidentiality, has to do with keeping information out of the grubby little hands of unauthorized users. This is what usually comes to mind when people think about network security. Authentication deals with determining whom you are talking to before revealing sensitive information or entering into a business deal. Nonrepudiation deals with signatures: how do you prove that your customer really placed an electronic order for ten million left-handed doohickeys at 89 cents each when he later claims the price was 69 cents? Or maybe he claims he never placed any order. Finally, integrity control has to do with how you can be sure that a message you received was really the one sent and not something that a malicious adversary modified in transit or concocted.

All these issues (secrecy, authentication, nonrepudiation, and integrity control) occur in traditional systems, too, but with some significant differences. Integrity and secrecy are achieved by using registered mail and locking documents up. Robbing the mail train is harder now than it was in Jesse James' day.

Also, people can usually tell the difference between an original paper document and a photocopy, and it often matters to them. As a test, make a photocopy of a valid check. Try cashing the original check at your bank on Monday. Now try cashing the photocopy of the check on Tuesday. Observe the difference in the bank's behavior. With electronic checks, the original and the copy are indistinguishable. It may take a while for banks to learn how to handle this.

People authenticate other people by various means, including recognizing their faces, voices, and handwriting. Proof of signing is handled by signatures on letterhead paper, raised seals, and so on. Tampering can usually be detected by

handwriting, ink, and paper experts. None of these options are available electronically. Clearly, other solutions are needed.

Before getting into the solutions themselves, it is worth spending a few moments considering where in the protocol stack network security belongs. There is probably no one single place. Every layer has something to contribute. In the physical layer, wiretapping can be foiled by enclosing transmission lines (or better yet, optical fibers) in sealed tubes containing an inert gas at high pressure. Any attempt to drill into a tube will release some gas, reducing the pressure and triggering an alarm. Some military systems use this technique.

In the data link layer, packets on a point-to-point line can be encrypted as they leave one machine and decrypted as they enter another. All the details can be handled in the data link layer, with higher layers oblivious to what is going on. This solution breaks down when packets have to traverse multiple routers, however, because packets have to be decrypted at each router, leaving them vulnerable to attacks from within the router. Also, it does not allow some sessions to be protected (e.g., those involving online purchases by credit card) and others not. Nevertheless, **link encryption**, as this method is called, can be added to any network easily and is often useful.

In the network layer, firewalls can be installed to keep good packets and bad packets out. IP security also functions in this layer.

In the transport layer, entire connections can be encrypted end to end, that is, process to process. For maximum security, end-to-end security is required.

Finally, issues such as user authentication and nonrepudiation can only be handled in the application layer.

Since security does not fit neatly into any layer, it does not fit into any chapter of this book. For this reason, it rates its own chapter.

While this chapter is long, technical, and essential, it is also quasi-irrelevant for the moment. It is well documented that most security failures at banks, for example, are due to lax security procedures and incompetent employees, numerous implementation bugs that enable remote break-ins by unauthorized users, and so-called social engineering attacks, where customers are tricked into revealing their account details. All of these security problems are more prevalent than clever criminals tapping phone lines and then decoding encrypted messages. If a person can walk into a random branch of a bank with an ATM slip he found on the street claiming to have forgotten his PIN and get a new one on the spot (in the name of good customer relations), all the cryptography in the world will not prevent abuse. In this respect, Ross Anderson's (2008a) book is a real eye-opener, as it documents hundreds of examples of security failures in numerous industries, nearly all of them due to what might politely be called sloppy business practices or inattention to security. Nevertheless, the technical foundation on which e-commerce is built when all of these other factors are done well is cryptography.

Except for physical layer security, nearly all network security is based on cryptographic principles. For this reason, we will begin our study of security by

examining cryptography in some detail. In Sec. 8.1, we will look at some of the basic principles. In Sec. 8-2 through Sec. 8-5, we will examine some of the fundamental algorithms and data structures used in cryptography. Then we will examine in detail how these concepts can be used to achieve security in networks. We will conclude with some brief thoughts about technology and society.

Before starting, one last thought is in order: what is not covered. We have tried to focus on networking issues, rather than operating system and application issues, although the line is often hard to draw. For example, there is nothing here about user authentication using biometrics, password security, buffer overflow attacks, Trojan horses, login spoofing, code injection such as cross-site scripting, viruses, worms, and the like. All of these topics are covered at length in Chap. 9 of *Modern Operating Systems* (Tanenbaum, 2007). The interested reader is referred to that book for the systems aspects of security. Now let us begin our journey.

8.1 CRYPTOGRAPHY

Cryptography comes from the Greek words for "secret writing." It has a long and colorful history going back thousands of years. In this section, we will just sketch some of the highlights, as background information for what follows. For a complete history of cryptography, Kahn's (1995) book is recommended reading. For a comprehensive treatment of modern security and cryptographic algorithms, protocols, and applications, and related material, see Kaufman et al. (2002). For a more mathematical approach, see Stinson (2002). For a less mathematical approach, see Burnett and Paine (2001).

Professionals make a distinction between ciphers and codes. A **cipher** is a character-for-character or bit-for-bit transformation, without regard to the linguistic structure of the message. In contrast, a **code** replaces one word with another word or symbol. Codes are not used any more, although they have a glorious history. The most successful code ever devised was used by the U.S. armed forces during World War II in the Pacific. They simply had Navajo Indians talking to each other using specific Navajo words for military terms, for example *chay-da-gahi-nail-tsaidi* (literally: tortoise killer) for antitank weapon. The Navajo language is highly tonal, exceedingly complex, and has no written form. And not a single person in Japan knew anything about it.

In September 1945, the *San Diego Union* described the code by saying "For three years, wherever the Marines landed, the Japanese got an earful of strange gurgling noises interspersed with other sounds resembling the call of a Tibetan monk and the sound of a hot water bottle being emptied." The Japanese never broke the code and many Navajo code talkers were awarded high military honors for extraordinary service and bravery. The fact that the U.S. broke the Japanese code but the Japanese never broke the Navajo code played a crucial role in the American victories in the Pacific.

8.1.1 Introduction to Cryptography

Historically, four groups of people have used and contributed to the art of cryptography: the military, the diplomatic corps, diarists, and lovers. Of these, the military has had the most important role and has shaped the field over the centuries. Within military organizations, the messages to be encrypted have traditionally been given to poorly paid, low-level code clerks for encryption and transmission. The sheer volume of messages prevented this work from being done by a few elite specialists.

Until the advent of computers, one of the main constraints on cryptography had been the ability of the code clerk to perform the necessary transformations, often on a battlefield with little equipment. An additional constraint has been the difficulty in switching over quickly from one cryptographic method to another one, since this entails retraining a large number of people. However, the danger of a code clerk being captured by the enemy has made it essential to be able to change the cryptographic method instantly if need be. These conflicting requirements have given rise to the model of Fig. 8-2.

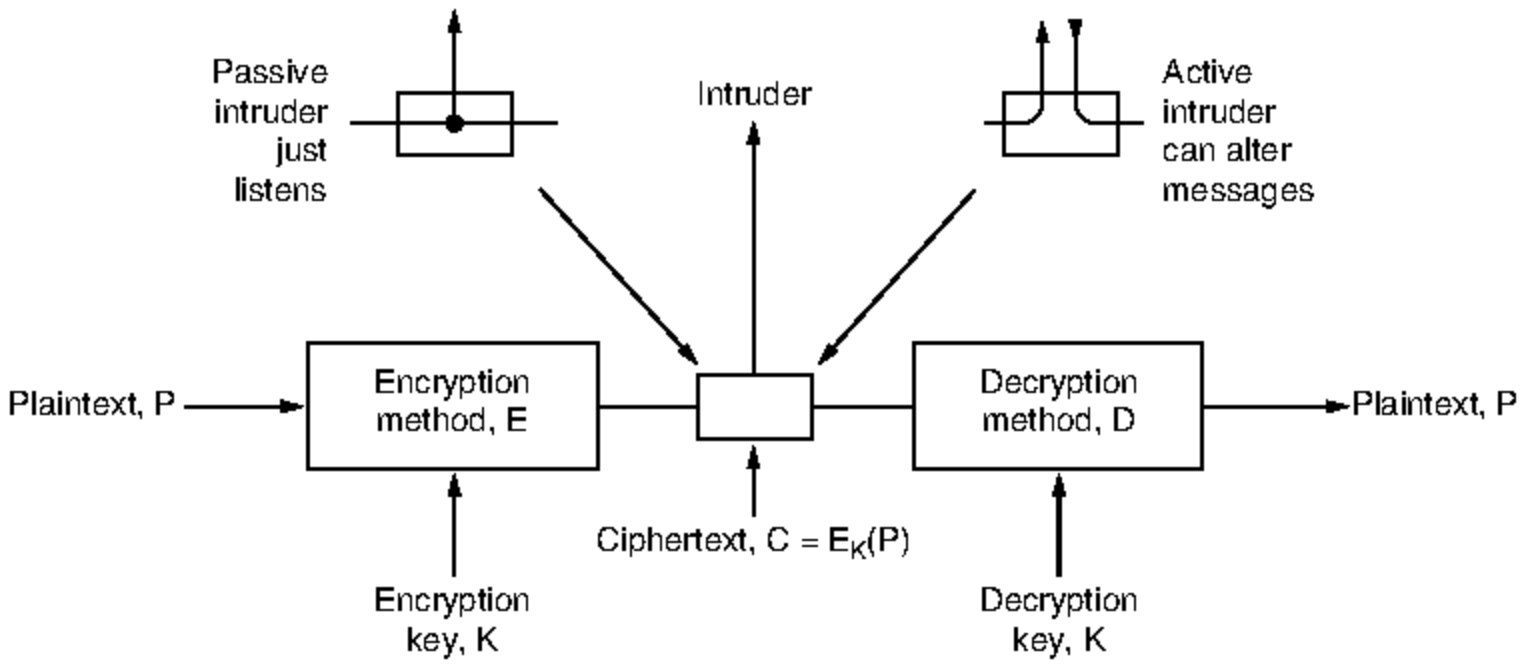


Figure 8-2. The encryption model (for a symmetric-key cipher).

The messages to be encrypted, known as the **plaintext**, are transformed by a function that is parameterized by a **key**. The output of the encryption process, known as the **ciphertext**, is then transmitted, often by messenger or radio. We assume that the enemy, or **intruder**, hears and accurately copies down the complete ciphertext. However, unlike the intended recipient, he does not know what the decryption key is and so cannot decrypt the ciphertext easily. Sometimes the intruder can not only listen to the communication channel (passive intruder) but can also record messages and play them back later, inject his own messages, or modify legitimate messages before they get to the receiver (active intruder). The art of

breaking ciphers, known as **cryptanalysis**, and the art of devising them (cryptography) are collectively known as **cryptology**.

It will often be useful to have a notation for relating plaintext, ciphertext, and keys. We will use $C = E_K(P)$ to mean that the encryption of the plaintext P using key K gives the ciphertext C. Similarly, $P = D_K(C)$ represents the decryption of C to get the plaintext again. It then follows that

$$D_K(E_K(P)) = P$$

This notation suggests that E and D are just mathematical functions, which they are. The only tricky part is that both are functions of two parameters, and we have written one of the parameters (the key) as a subscript, rather than as an argument, to distinguish it from the message.

A fundamental rule of cryptography is that one must assume that the cryptanalyst knows the methods used for encryption and decryption. In other words, the cryptanalyst knows how the encryption method, E, and decryption, D, of Fig. 8-2 work in detail. The amount of effort necessary to invent, test, and install a new algorithm every time the old method is compromised (or thought to be compromised) has always made it impractical to keep the encryption algorithm secret. Thinking it is secret when it is not does more harm than good.

This is where the key enters. The key consists of a (relatively) short string that selects one of many potential encryptions. In contrast to the general method, which may only be changed every few years, the key can be changed as often as required. Thus, our basic model is a stable and publicly known general method parameterized by a secret and easily changed key. The idea that the cryptanalyst knows the algorithms and that the secrecy lies exclusively in the keys is called **Kerckhoff's principle**, named after the Flemish military cryptographer Auguste Kerckhoff who first stated it in 1883 (Kerckhoff, 1883). Thus, we have

Kerckhoff's principle: All algorithms must be public; only the keys are secret

The nonsecrecy of the algorithm cannot be emphasized enough. Trying to keep the algorithm secret, known in the trade as **security by obscurity**, never works. Also, by publicizing the algorithm, the cryptographer gets free consulting from a large number of academic cryptologists eager to break the system so they can publish papers demonstrating how smart they are. If many experts have tried to break the algorithm for a long time after its publication and no one has succeeded, it is probably pretty solid.

Since the real secrecy is in the key, its length is a major design issue. Consider a simple combination lock. The general principle is that you enter digits in sequence. Everyone knows this, but the key is secret. A key length of two digits means that there are 100 possibilities. A key length of three digits means 1000 possibilities, and a key length of six digits means a million. The longer the key, the higher the **work factor** the cryptanalyst has to deal with. The work factor for breaking the system by exhaustive search of the key space is exponential in the

key length. Secrecy comes from having a strong (but public) algorithm and a long key. To prevent your kid brother from reading your email, 64-bit keys will do. For routine commercial use, at least 128 bits should be used. To keep major governments at bay, keys of at least 256 bits, preferably more, are needed.

From the cryptanalyst's point of view, the cryptanalysis problem has three principal variations. When he has a quantity of ciphertext and no plaintext, he is confronted with the **ciphertext-only** problem. The cryptograms that appear in the puzzle section of newspapers pose this kind of problem. When the cryptanalyst has some matched ciphertext and plaintext, the problem is called the **known plaintext** problem. Finally, when the cryptanalyst has the ability to encrypt pieces of plaintext of his own choosing, we have the **chosen plaintext** problem. Newspaper cryptograms could be broken trivially if the cryptanalyst were allowed to ask such questions as "What is the encryption of ABCDEFGHIJKL?"

Novices in the cryptography business often assume that if a cipher can withstand a ciphertext-only attack, it is secure. This assumption is very naive. In many cases, the cryptanalyst can make a good guess at parts of the plaintext. For example, the first thing many computers say when you call them up is "login:". Equipped with some matched plaintext-ciphertext pairs, the cryptanalyst's job becomes much easier. To achieve security, the cryptographer should be conservative and make sure that the system is unbreakable even if his opponent can encrypt arbitrary amounts of chosen plaintext.

Encryption methods have historically been divided into two categories: substitution ciphers and transposition ciphers. We will now deal with each of these briefly as background information for modern cryptography.

8.1.2 Substitution Ciphers

In a **substitution cipher**, each letter or group of letters is replaced by another letter or group of letters to disguise it. One of the oldest known ciphers is the **Caesar cipher**, attributed to Julius Caesar. With this method, *a* becomes *D*, *b* becomes *E*, *c* becomes *F*, . . . , and *z* becomes *C*. For example, *attack* becomes *DWWDFN*. In our examples, plaintext will be given in lowercase letters, and ciphertext in uppercase letters.

A slight generalization of the Caesar cipher allows the ciphertext alphabet to be shifted by k letters, instead of always three. In this case, k becomes a key to the general method of circularly shifted alphabets. The Caesar cipher may have fooled Pompey, but it has not fooled anyone since.

The next improvement is to have each of the symbols in the plaintext, say, the 26 letters for simplicity, map onto some other letter. For example,

plaintext:	a b c d e f g h i j k l m n o p q r s t u v w x y z
ciphertext:	Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

The general system of symbol-for-symbol substitution is called a **monoalphabetic substitution cipher**, with the key being the 26-letter string corresponding to the full alphabet. For the key just given, the plaintext *attack* would be transformed into the ciphertext *QZZQEA*.

At first glance this might appear to be a safe system because although the cryptanalyst knows the general system (letter-for-letter substitution), he does not know which of the $26! \approx 4 \times 10^{26}$ possible keys is in use. In contrast with the Caesar cipher, trying all of them is not a promising approach. Even at 1 nsec per solution, a million computer chips working in parallel would take 10,000 years to try all the keys.

Nevertheless, given a surprisingly small amount of ciphertext, the cipher can be broken easily. The basic attack takes advantage of the statistical properties of natural languages. In English, for example, *e* is the most common letter, followed by *t*, *o*, *a*, *n*, *i*, etc. The most common two-letter combinations, or **digrams**, are *th*, *in*, *er*, *re*, and *an*. The most common three-letter combinations, or **trigrams**, are *the*, *ing*, *and*, and *ion*.

A cryptanalyst trying to break a monoalphabetic cipher would start out by counting the relative frequencies of all letters in the ciphertext. Then he might tentatively assign the most common one to *e* and the next most common one to *t*. He would then look at trigrams to find a common one of the form *tXe*, which strongly suggests that *X* is *h*. Similarly, if the pattern *thYt* occurs frequently, the *Y* probably stands for *a*. With this information, he can look for a frequently occurring trigram of the form *aZW*, which is most likely *and*. By making guesses at common letters, digrams, and trigrams and knowing about likely patterns of vowels and consonants, the cryptanalyst builds up a tentative plaintext, letter by letter.

Another approach is to guess a probable word or phrase. For example, consider the following ciphertext from an accounting firm (blocked into groups of five characters):

```
CTBMN BYCTC BTJDS QXBNS GSTJC BTSWX CTQTZ CQVUJ
QJSGS TJQZZ MNQJS VLNSX VSZJU JDSTS JQUUS JUBXJ
DSKSU JSNTK BGAQJ ZBGYQ TLCTZ BNYBN QJSW
```

A likely word in a message from an accounting firm is *financial*. Using our knowledge that *financial* has a repeated letter (*i*), with four other letters between their occurrences, we look for repeated letters in the ciphertext at this spacing. We find 12 hits, at positions 6, 15, 27, 31, 42, 48, 56, 66, 70, 71, 76, and 82. However, only two of these, 31 and 42, have the next letter (corresponding to *n* in the plaintext) repeated in the proper place. Of these two, only 31 also has the *a* correctly positioned, so we know that *financial* begins at position 30. From this point on, deducing the key is easy by using the frequency statistics for English text and looking for nearly complete words to finish off.

8.1.3 Transposition Ciphers

Substitution ciphers preserve the order of the plaintext symbols but disguise them. **Transposition ciphers**, in contrast, reorder the letters but do not disguise them. Figure 8-3 depicts a common transposition cipher, the columnar transposition. The cipher is keyed by a word or phrase not containing any repeated letters. In this example, MEGABUCK is the key. The purpose of the key is to order the columns, with column 1 being under the key letter closest to the start of the alphabet, and so on. The plaintext is written horizontally, in rows, padded to fill the matrix if need be. The ciphertext is read out by columns, starting with the column whose key letter is the lowest.

M	E	G	A	B	U	C	K
7	4	5	1	2	8	3	6
p	l	e	a	s	e	t	r
a	n	s	f	e	r	o	n
e	m	i	l	l	i	o	n
d	o	l	l	a	r	s	t
o	m	y	s	w	i	s	s
b	a	n	k	a	c	c	o
u	n	t	s	i	x	t	w
o	t	w	o	a	b	c	d

Plaintext

pleasetransferonemilliondollarsto
myswissbankaccountsixtwotwo

Ciphertext

AFLLSKSOSELAWAIATOOSSCTCLNMOMANT
ESILYNTWRNNTSOWDPAEDOBUOERIRICXB

Figure 8-3. A transposition cipher.

To break a transposition cipher, the cryptanalyst must first be aware that he is dealing with a transposition cipher. By looking at the frequency of *E*, *T*, *A*, *O*, *I*, *N*, etc., it is easy to see if they fit the normal pattern for plaintext. If so, the cipher is clearly a transposition cipher, because in such a cipher every letter represents itself, keeping the frequency distribution intact.

The next step is to make a guess at the number of columns. In many cases, a probable word or phrase may be guessed at from the context. For example, suppose that our cryptanalyst suspects that the plaintext phrase *milliondollars* occurs somewhere in the message. Observe that digrams *MO*, *IL*, *LL*, *LA*, *IR*, and *OS* occur in the ciphertext as a result of this phrase wrapping around. The ciphertext letter *O* follows the ciphertext letter *M* (i.e., they are vertically adjacent in column 4) because they are separated in the probable phrase by a distance equal to the key length. If a key of length seven had been used, the digrams *MD*, *IO*, *LL*, *LL*, *IA*, *OR*, and *NS* would have occurred instead. In fact, for each key length, a different set of digrams is produced in the ciphertext. By hunting for the various possibilities, the cryptanalyst can often easily determine the key length.

The remaining step is to order the columns. When the number of columns, k, is small, each of the $k(k-1)$ column pairs can be examined in turn to see if its digram frequencies match those for English plaintext. The pair with the best match is assumed to be correctly positioned. Now each of the remaining columns is tentatively tried as the successor to this pair. The column whose digram and trigram frequencies give the best match is tentatively assumed to be correct. The next column is found in the same way. The entire process is continued until a potential ordering is found. Chances are that the plaintext will be recognizable at this point (e.g., if *milloin* occurs, it is clear what the error is).

Some transposition ciphers accept a fixed-length block of input and produce a fixed-length block of output. These ciphers can be completely described by giving a list telling the order in which the characters are to be output. For example, the cipher of Fig. 8-3 can be seen as a 64 character block cipher. Its output is 4, 12, 20, 28, 36, 44, 52, 60, 5, 13, . . . , 62. In other words, the fourth input character, *a*, is the first to be output, followed by the twelfth, *f*, and so on.

8.1.4 One-Time Pads

Constructing an unbreakable cipher is actually quite easy; the technique has been known for decades. First choose a random bit string as the key. Then convert the plaintext into a bit string, for example, by using its ASCII representation. Finally, compute the XOR (eXclusive OR) of these two strings, bit by bit. The resulting ciphertext cannot be broken because in a sufficiently large sample of ciphertext, each letter will occur equally often, as will every digram, every trigram, and so on. This method, known as the **one-time pad**, is immune to all present and future attacks, no matter how much computational power the intruder has. The reason derives from information theory: there is simply no information in the message because all possible plaintexts of the given length are equally likely.

An example of how one-time pads are used is given in Fig. 8-4. First, message 1, "I love you." is converted to 7-bit ASCII. Then a one-time pad, pad 1, is chosen and XORed with the message to get the ciphertext. A cryptanalyst could try all possible one-time pads to see what plaintext came out for each one. For example, the one-time pad listed as pad 2 in the figure could be tried, resulting in plaintext 2, "Elvis lives'', which may or may not be plausible (a subject beyond the scope of this book). In fact, for every 11-character ASCII plaintext, there is a one-time pad that generates it. That is what we mean by saying there is no information in the ciphertext: you can get any message of the correct length out of it.

One-time pads are great in theory but have a number of disadvantages in practice. To start with, the key cannot be memorized, so both sender and receiver must carry a written copy with them. If either one is subject to capture, written keys are clearly undesirable. Additionally, the total amount of data that can be transmitted is limited by the amount of key available. If the spy strikes it rich and discovers a wealth of data, he may find himself unable to transmit them back to

Message 1:	1001001	0100000	1101100	1101111	1110110	1100101	0100000	1111001	1101111	1110101	0101110
Pad 1:	1010010	1001011	1110010	1010101	1010010	1100011	0001011	0101010	1010111	1100110	0101011
Ciphertext:	0011011	1101011	0011110	0111010	0100100	0000110	0101011	1010011	0111000	0010011	0000101
Pad 2:	1011110	0000111	1101000	1010011	1010111	0100110	1000111	0111010	1001110	1110110	1110110
Plaintext 2:	1000101	1101100	1110110	1101001	1110011	0100000	1101100	1101001	1110110	1100101	1110011

Figure 8-4. The use of a one-time pad for encryption and the possibility of getting any possible plaintext from the ciphertext by the use of some other pad.

headquarters because the key has been used up. Another problem is the sensitivity of the method to lost or inserted characters. If the sender and receiver get out of synchronization, all data from then on will appear garbled.

With the advent of computers, the one-time pad might potentially become practical for some applications. The source of the key could be a special DVD that contains several gigabytes of information and, if transported in a DVD movie box and prefixed by a few minutes of video, would not even be suspicious. Of course, at gigabit network speeds, having to insert a new DVD every 30 sec could become tedious. And the DVDs must be personally carried from the sender to the receiver before any messages can be sent, which greatly reduces their practical utility.

Quantum Cryptography

Interestingly, there may be a solution to the problem of how to transmit the one-time pad over the network, and it comes from a very unlikely source: quantum mechanics. This area is still experimental, but initial tests are promising. If it can be perfected and be made efficient, virtually all cryptography will eventually be done using one-time pads since they are provably secure. Below we will briefly explain how this method, **quantum cryptography**, works. In particular, we will describe a protocol called **BB84** after its authors and publication year (Bennet and Brassard, 1984).

Suppose that a user, Alice, wants to establish a one-time pad with a second user, Bob. Alice and Bob are called **principals**, the main characters in our story. For example, Bob is a banker with whom Alice would like to do business. The names "Alice" and "Bob" have been used for the principals in virtually every paper and book on cryptography since Ron Rivest introduced them many years ago (Rivest et al., 1978). Cryptographers love tradition. If we were to use "Andy" and "Barbara" as the principals, no one would believe anything in this chapter. So be it.

If Alice and Bob could establish a one-time pad, they could use it to communicate securely. The question is: how can they establish it without previously exchanging DVDs? We can assume that Alice and Bob are at the opposite ends

of an optical fiber over which they can send and receive light pulses. However, an intrepid intruder, Trudy, can cut the fiber to splice in an active tap. Trudy can read all the bits sent in both directions. She can also send false messages in both directions. The situation might seem hopeless for Alice and Bob, but quantum cryptography can shed some new light on the subject.

Quantum cryptography is based on the fact that light comes in little packets called **photons**, which have some peculiar properties. Furthermore, light can be polarized by being passed through a polarizing filter, a fact well known to both sunglasses wearers and photographers. If a beam of light (i.e., a stream of photons) is passed through a polarizing filter, all the photons emerging from it will be polarized in the direction of the filter's axis (e.g., vertically). If the beam is now passed through a second polarizing filter, the intensity of the light emerging from the second filter is proportional to the square of the cosine of the angle between the axes. If the two axes are perpendicular, no photons get through. The absolute orientation of the two filters does not matter; only the angle between their axes counts.

To generate a one-time pad, Alice needs two sets of polarizing filters. Set one consists of a vertical filter and a horizontal filter. This choice is called a **rectilinear basis**. A basis (plural: bases) is just a coordinate system. The second set of filters is the same, except rotated 45 degrees, so one filter runs from the lower left to the upper right and the other filter runs from the upper left to the lower right. This choice is called a **diagonal basis**. Thus, Alice has two bases, which she can rapidly insert into her beam at will. In reality, Alice does not have four separate filters, but a crystal whose polarization can be switched electrically to any of the four allowed directions at great speed. Bob has the same equipment as Alice. The fact that Alice and Bob each have two bases available is essential to quantum cryptography.

For each basis, Alice now assigns one direction as 0 and the other as 1. In the example presented below, we assume she chooses vertical to be 0 and horizontal to be 1. Independently, she also chooses lower left to upper right as 0 and upper left to lower right as 1. She sends these choices to Bob as plaintext.

Now Alice picks a one-time pad, for example based on a random number generator (a complex subject all by itself). She transfers it bit by bit to Bob, choosing one of her two bases at random for each bit. To send a bit, her photon gun emits one photon polarized appropriately for the basis she is using for that bit. For example, she might choose bases of diagonal, rectilinear, rectilinear, diagonal, rectilinear, etc. To send her one-time pad of 1001110010100110 with these bases, she would send the photons shown in Fig. 8-5(a). Given the one-time pad and the sequence of bases, the polarization to use for each bit is uniquely determined. Bits sent one photon at a time are called **qubits**.

Bob does not know which bases to use, so he picks one at random for each arriving photon and just uses it, as shown in Fig. 8-5(b). If he picks the correct basis, he gets the correct bit. If he picks the incorrect basis, he gets a random bit

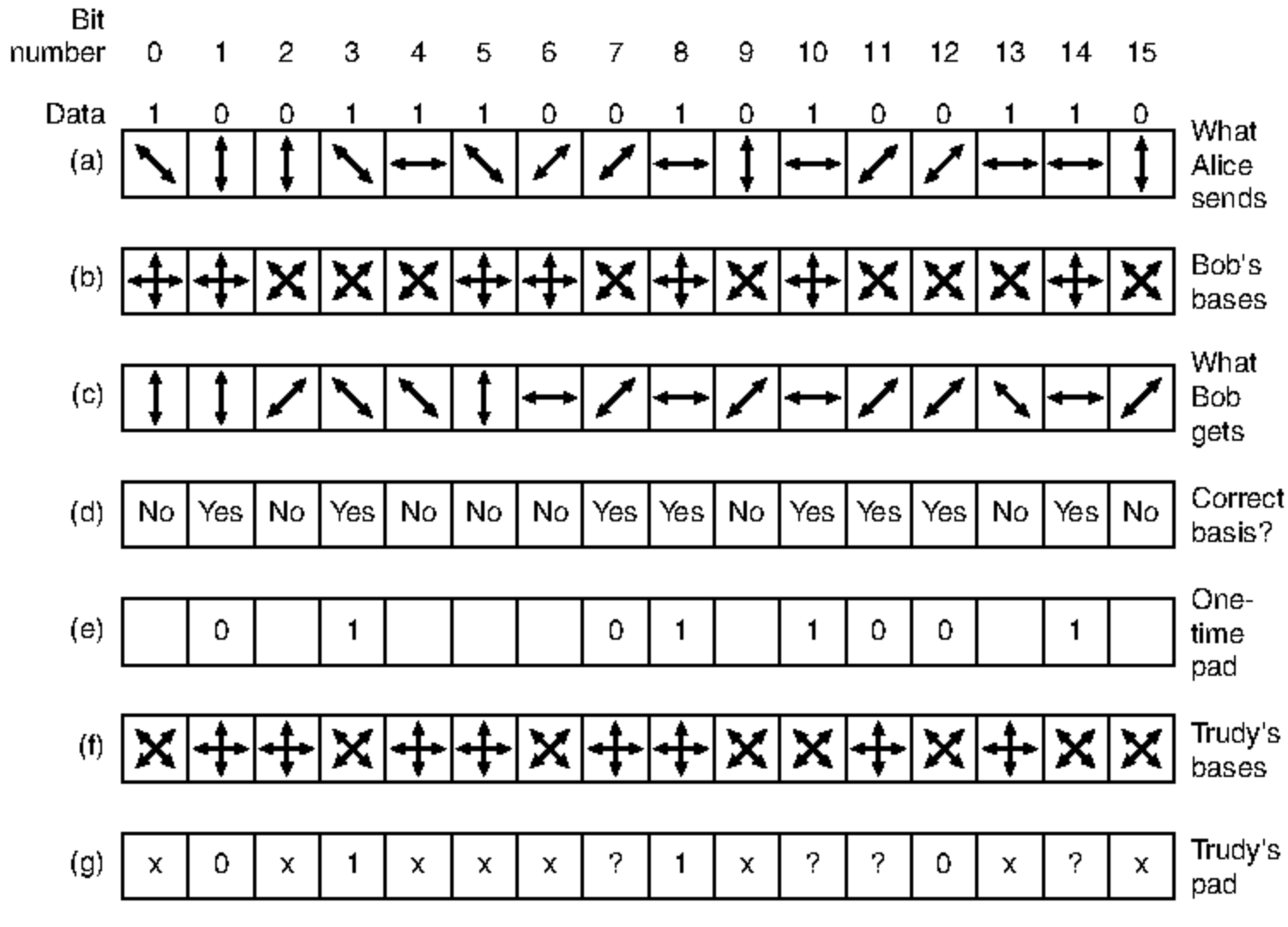


Figure 8-5. An example of quantum cryptography.

because if a photon hits a filter polarized at 45 degrees to its own polarization, it randomly jumps to the polarization of the filter or to a polarization perpendicular to the filter, with equal probability. This property of photons is fundamental to quantum mechanics. Thus, some of the bits are correct and some are random, but Bob does not know which are which. Bob's results are depicted in Fig. 8-5(c).

How does Bob find out which bases he got right and which he got wrong? He simply tells Alice which basis he used for each bit in plaintext and she tells him which are right and which are wrong in plaintext, as shown in Fig. 8-5(d). From this information, both of them can build a bit string from the correct guesses, as shown in Fig. 8-5(e). On the average, this bit string will be half the length of the original bit string, but since both parties know it, they can use it as a one-time pad. All Alice has to do is transmit a bit string slightly more than twice the desired length, and she and Bob will have a one-time pad of the desired length. Done.

But wait a minute. We forgot Trudy. Suppose that she is curious about what Alice has to say and cuts the fiber, inserting her own detector and transmitter. Unfortunately for her, she does not know which basis to use for each photon either. The best she can do is pick one at random for each photon, just as Bob does. An example of her choices is shown in Fig. 8-5(f). When Bob later reports (in plaintext) which bases he used and Alice tells him (in plaintext) which ones are

correct, Trudy now knows when she got it right and when she got it wrong. In Fig. 8-5, she got it right for bits 0, 1, 2, 3, 4, 6, 8, 12, and 13. But she knows from Alice's reply in Fig. 8-5(d) that only bits 1, 3, 7, 8, 10, 11, 12, and 14 are part of the one-time pad. For four of these bits (1, 3, 8, and 12), she guessed right and captured the correct bit. For the other four (7, 10, 11, and 14), she guessed wrong and does not know the bit transmitted. Thus, Bob knows the one-time pad starts with 01011001, from Fig. 8-5(e) but all Trudy has is 01?1??0?, from Fig. 8-5(g).

Of course, Alice and Bob are aware that Trudy may have captured part of their one-time pad, so they would like to reduce the information Trudy has. They can do this by performing a transformation on it. For example, they could divide the one-time pad into blocks of 1024 bits, square each one to form a 2048-bit number, and use the concatenation of these 2048-bit numbers as the one-time pad. With her partial knowledge of the bit string transmitted, Trudy has no way to generate its square and so has nothing. The transformation from the original one-time pad to a different one that reduces Trudy's knowledge is called **privacy amplification**. In practice, complex transformations in which every output bit depends on every input bit are used instead of squaring.

Poor Trudy. Not only does she have no idea what the one-time pad is, but her presence is not a secret either. After all, she must relay each received bit to Bob to trick him into thinking he is talking to Alice. The trouble is, the best she can do is transmit the qubit she received, using the polarization she used to receive it, and about half the time she will be wrong, causing many errors in Bob's one-time pad.

When Alice finally starts sending data, she encodes it using a heavy forward-error-correcting code. From Bob's point of view, a 1-bit error in the one-time pad is the same as a 1-bit transmission error. Either way, he gets the wrong bit. If there is enough forward error correction, he can recover the original message despite all the errors, but he can easily count how many errors were corrected. If this number is far more than the expected error rate of the equipment, he knows that Trudy has tapped the line and can act accordingly (e.g., tell Alice to switch to a radio channel, call the police, etc.). If Trudy had a way to clone a photon so she had one photon to inspect and an identical photon to send to Bob, she could avoid detection, but at present no way to clone a photon perfectly is known. And even if Trudy could clone photons, the value of quantum cryptography to establish one-time pads would not be reduced.

Although quantum cryptography has been shown to operate over distances of 60 km of fiber, the equipment is complex and expensive. Still, the idea has promise. For more information about quantum cryptography, see Mullins (2002).

8.1.5 Two Fundamental Cryptographic Principles

Although we will study many different cryptographic systems in the pages ahead, two principles underlying all of them are important to understand. Pay attention. You violate them at your peril.

Redundancy

The first principle is that all encrypted messages must contain some redundancy, that is, information not needed to understand the message. An example may make it clear why this is needed. Consider a mail-order company, The Couch Potato (TCP), with 60,000 products. Thinking they are being very efficient, TCP's programmers decide that ordering messages should consist of a 16-byte customer name followed by a 3-byte data field (1 byte for the quantity and 2 bytes for the product number). The last 3 bytes are to be encrypted using a very long key known only by the customer and TCP.

At first, this might seem secure, and in a sense it is because passive intruders cannot decrypt the messages. Unfortunately, it also has a fatal flaw that renders it useless. Suppose that a recently fired employee wants to punish TCP for firing her. Just before leaving, she takes the customer list with her. She works through the night writing a program to generate fictitious orders using real customer names. Since she does not have the list of keys, she just puts random numbers in the last 3 bytes, and sends hundreds of orders off to TCP.

When these messages arrive, TCP's computer uses the customers' name to locate the key and decrypt the message. Unfortunately for TCP, almost every 3-byte message is valid, so the computer begins printing out shipping instructions. While it might seem odd for a customer to order 837 sets of children's swings or 540 sandboxes, for all the computer knows, the customer might be planning to open a chain of franchised playgrounds. In this way, an active intruder (the ex-employee) can cause a massive amount of trouble, even though she cannot understand the messages her computer is generating.

This problem can be solved by the addition of redundancy to all messages. For example, if order messages are extended to 12 bytes, the first 9 of which must be zeros, this attack no longer works because the ex-employee can no longer generate a large stream of valid messages. The moral of the story is that all messages must contain considerable redundancy so that active intruders cannot send random junk and have it be interpreted as a valid message.

However, adding redundancy makes it easier for cryptanalysts to break messages. Suppose that the mail-order business is highly competitive, and The Couch Potato's main competitor, The Sofa Tuber, would dearly love to know how many sandboxes TCP is selling so it taps TCP's phone line. In the original scheme with 3-byte messages, cryptanalysis was nearly impossible because after guessing a key, the cryptanalyst had no way of telling whether it was right because almost every message was technically legal. With the new 12-byte scheme, it is easy for the cryptanalyst to tell a valid message from an invalid one. Thus, we have

Cryptographic principle 1: Messages must contain some redundancy

In other words, upon decrypting a message, the recipient must be able to tell whether it is valid by simply inspecting the message and perhaps performing a

simple computation. This redundancy is needed to prevent active intruders from sending garbage and tricking the receiver into decrypting the garbage and acting on the "plaintext." However, this same redundancy makes it much easier for passive intruders to break the system, so there is some tension here. Furthermore, the redundancy should never be in the form of n 0s at the start or end of a message, since running such messages through some cryptographic algorithms gives more predictable results, making the cryptanalysts' job easier. A CRC polynomial is much better than a run of 0s since the receiver can easily verify it, but it generates more work for the cryptanalyst. Even better is to use a cryptographic hash, a concept we will explore later. For the moment, think of it as a better CRC.

Getting back to quantum cryptography for a moment, we can also see how redundancy plays a role there. Due to Trudy's interception of the photons, some bits in Bob's one-time pad will be wrong. Bob needs some redundancy in the incoming messages to determine that errors are present. One very crude form of redundancy is repeating the message two times. If the two copies are not identical, Bob knows that either the fiber is very noisy or someone is tampering with the transmission. Of course, sending everything twice is overkill; a Hamming or Reed-Solomon code is a more efficient way to do error detection and correction. But it should be clear that some redundancy is needed to distinguish a valid message from an invalid message, especially in the face of an active intruder.

Freshness

The second cryptographic principle is that measures must be taken to ensure that each message received can be verified as being fresh, that is, sent very recently. This measure is needed to prevent active intruders from playing back old messages. If no such measures were taken, our ex-employee could tap TCP's phone line and just keep repeating previously sent valid messages. Thus,

Cryptographic principle 2: Some method is needed to foil replay attacks

One such measure is including in every message a timestamp valid only for, say, 10 seconds. The receiver can then just keep messages around for 10 seconds and compare newly arrived messages to previous ones to filter out duplicates. Messages older than 10 seconds can be thrown out, since any replays sent more than 10 seconds later will be rejected as too old. Measures other than timestamps will be discussed later.

8.2 SYMMETRIC-KEY ALGORITHMS

Modern cryptography uses the same basic ideas as traditional cryptography (transposition and substitution), but its emphasis is different. Traditionally, cryptographers have used simple algorithms. Nowadays, the reverse is true: the object

is to make the encryption algorithm so complex and involuted that even if the cryptanalyst acquires vast mounds of enciphered text of his own choosing, he will not be able to make any sense of it at all without the key.

The first class of encryption algorithms we will study in this chapter are called **symmetric-key algorithms** because they use the same key for encryption and decryption. Fig. 8-2 illustrates the use of a symmetric-key algorithm. In particular, we will focus on **block ciphers**, which take an n-bit block of plaintext as input and transform it using the key into an n-bit block of ciphertext.

Cryptographic algorithms can be implemented in either hardware (for speed) or software (for flexibility). Although most of our treatment concerns the algorithms and protocols, which are independent of the actual implementation, a few words about building cryptographic hardware may be of interest. Transpositions and substitutions can be implemented with simple electrical circuits. Figure 8-6(a) shows a device, known as a **P-box** (P stands for permutation), used to effect a transposition on an 8-bit input. If the 8 bits are designated from top to bottom as 01234567, the output of this particular P-box is 36071245. By appropriate internal wiring, a P-box can be made to perform any transposition and do it at practically the speed of light since no computation is involved, just signal propagation. This design follows Kerckhoff's principle: the attacker knows that the general method is permuting the bits. What he does not know is which bit goes where.

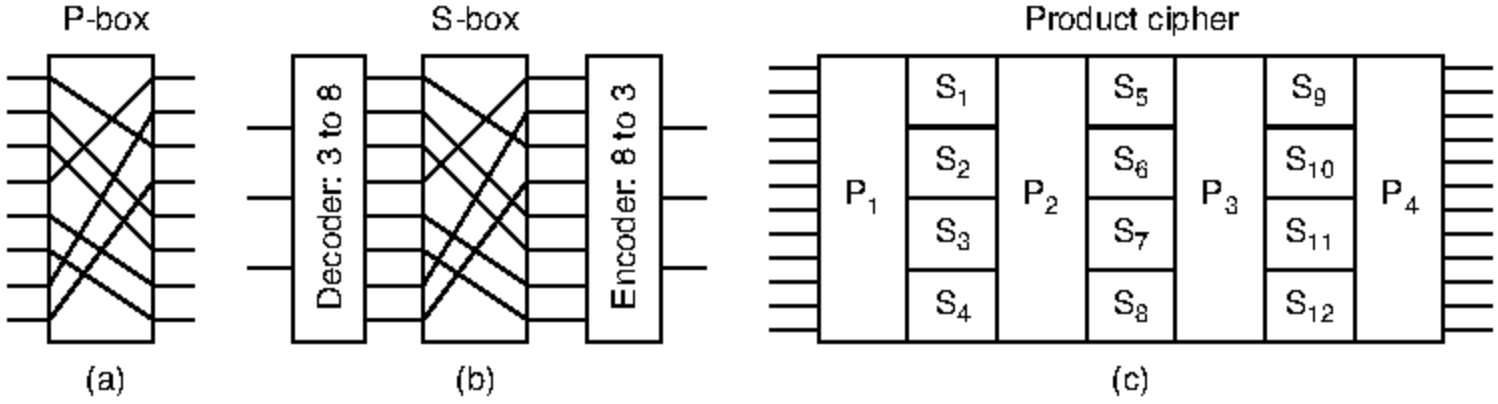


Figure 8-6. Basic elements of product ciphers. (a) P-box. (b) S-box. (c) Product.

Substitutions are performed by **S-boxes**, as shown in Fig. 8-6(b). In this example, a 3-bit plaintext is entered and a 3-bit ciphertext is output. The 3-bit input selects one of the eight lines exiting from the first stage and sets it to 1; all the other lines are 0. The second stage is a P-box. The third stage encodes the selected input line in binary again. With the wiring shown, if the eight octal numbers 01234567 were input one after another, the output sequence would be 24506713. In other words, 0 has been replaced by 2, 1 has been replaced by 4, etc. Again, by appropriate wiring of the P-box inside the S-box, any substitution can be accomplished. Furthermore, such a device can be built in hardware to achieve great speed, since encoders and decoders have only one or two (subnanosecond) gate delays and the propagation time across the P-box may well be less than 1 picosec.

The real power of these basic elements only becomes apparent when we cascade a whole series of boxes to form a **product cipher**, as shown in Fig. 8-6(c). In this example, 12 input lines are transposed (i.e., permuted) by the first stage (P_1). In the second stage, the input is broken up into four groups of 3 bits, each of which is substituted independently of the others (S_1 to S_4). This arrangement shows a method of approximating a larger S-box from multiple, smaller S-boxes. It is useful because small S-boxes are practical for a hardware implementation (e.g., an 8-bit S-box can be realized as a 256-entry lookup table), but large S-boxes become unwieldy to build (e.g., a 12-bit S-box would at a minimum need $2^{12} = 4096$ crossed wires in its middle stage). Although this method is less general, it is still powerful. By inclusion of a sufficiently large number of stages in the product cipher, the output can be made to be an exceedingly complicated function of the input.

Product ciphers that operate on k-bit inputs to produce k-bit outputs are very common. Typically, k is 64 to 256. A hardware implementation usually has at least 10 physical stages, instead of just 7 as in Fig. 8-6(c). A software implementation is programmed as a loop with at least eight iterations, each one performing S-box-type substitutions on subblocks of the 64- to 256-bit data block, followed by a permutation that mixes the outputs of the S-boxes. Often there is a special initial permutation and one at the end as well. In the literature, the iterations are called **rounds**.

8.2.1 DES—The Data Encryption Standard

In January 1977, the U.S. Government adopted a product cipher developed by IBM as its official standard for unclassified information. This cipher, **DES** (**Data Encryption Standard**), was widely adopted by the industry for use in security products. It is no longer secure in its original form, but in a modified form it is still useful. We will now explain how DES works.

An outline of DES is shown in Fig. 8-7(a). Plaintext is encrypted in blocks of 64 bits, yielding 64 bits of ciphertext. The algorithm, which is parameterized by a 56-bit key, has 19 distinct stages. The first stage is a key-independent transposition on the 64-bit plaintext. The last stage is the exact inverse of this transposition. The stage prior to the last one exchanges the leftmost 32 bits with the rightmost 32 bits. The remaining 16 stages are functionally identical but are parameterized by different functions of the key. The algorithm has been designed to allow decryption to be done with the same key as encryption, a property needed in any symmetric-key algorithm. The steps are just run in the reverse order.

The operation of one of these intermediate stages is illustrated in Fig. 8-7(b). Each stage takes two 32-bit inputs and produces two 32-bit outputs. The left output is simply a copy of the right input. The right output is the bitwise XOR of the left input and a function of the right input and the key for this stage, K_i. Pretty much all the complexity of the algorithm lies in this function.

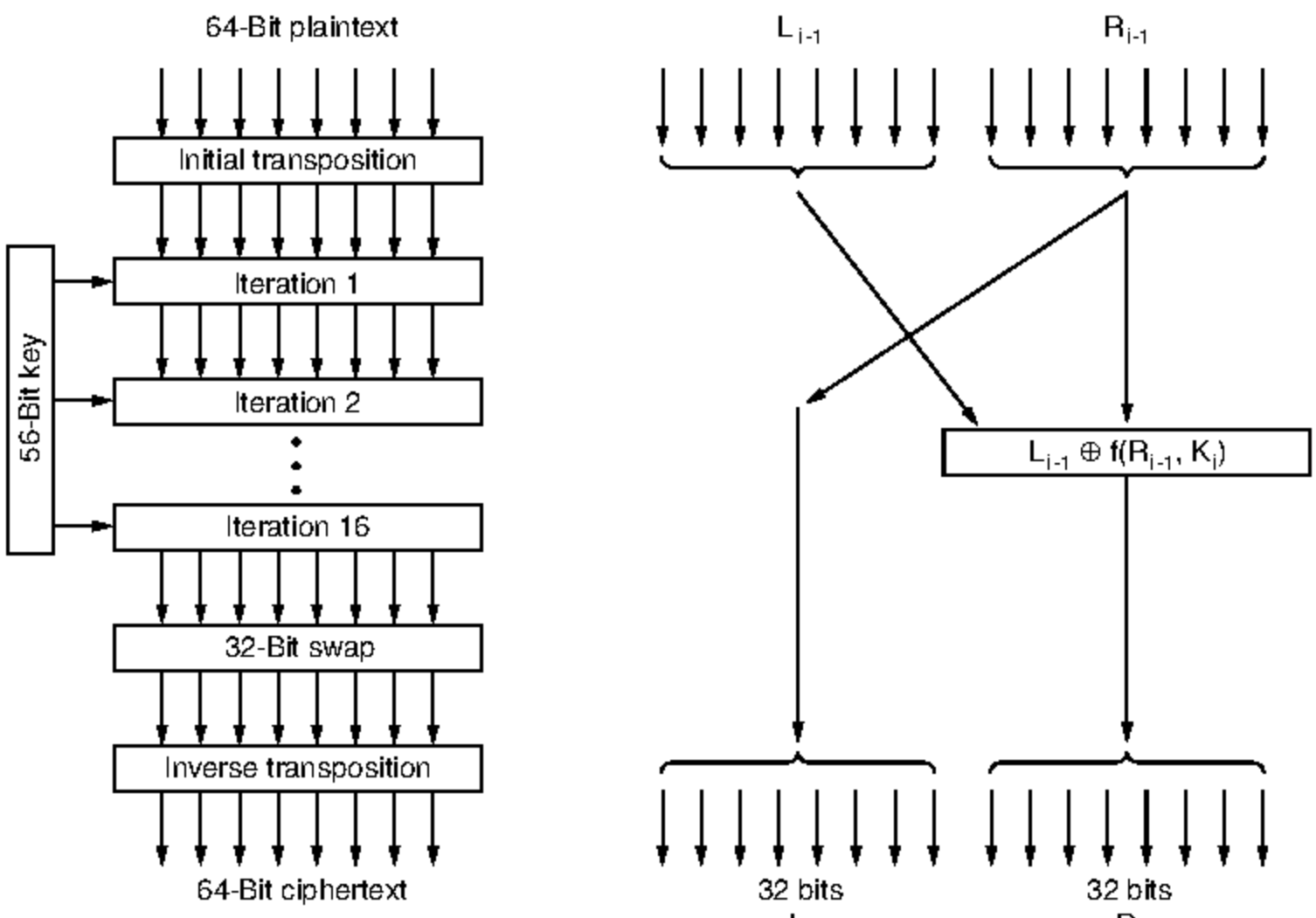


Figure 8-7. The Data Encryption Standard. (a) General outline. (b) Detail of one iteration. The circled + means exclusive OR.

The function consists of four steps, carried out in sequence. First, a 48-bit number, E, is constructed by expanding the 32-bit R_{i-1} according to a fixed transposition and duplication rule. Second, E and K_i are XORed together. This output is then partitioned into eight groups of 6 bits each, each of which is fed into a different S-box. Each of the 64 possible inputs to an S-box is mapped onto a 4-bit output. Finally, these 8×4 bits are passed through a P-box.

In each of the 16 iterations, a different key is used. Before the algorithm starts, a 56-bit transposition is applied to the key. Just before each iteration, the key is partitioned into two 28-bit units, each of which is rotated left by a number of bits dependent on the iteration number. K_i is derived from this rotated key by applying yet another 56-bit transposition to it. A different 48-bit subset of the 56 bits is extracted and permuted on each round.

A technique that is sometimes used to make DES stronger is called **whitening**. It consists of XORing a random 64-bit key with each plaintext block before feeding it into DES and then XORing a second 64-bit key with the resulting ciphertext before transmitting it. Whitening can easily be removed by running the

reverse operations (if the receiver has the two whitening keys). Since this technique effectively adds more bits to the key length, it makes an exhaustive search of the key space much more time consuming. Note that the same whitening key is used for each block (i.e., there is only one whitening key).

DES has been enveloped in controversy since the day it was launched. It was based on a cipher developed and patented by IBM, called Lucifer, except that IBM's cipher used a 128-bit key instead of a 56-bit key. When the U.S. Federal Government wanted to standardize on one cipher for unclassified use, it "invited" IBM to "discuss" the matter with NSA, the U.S. Government's code-breaking arm, which is the world's largest employer of mathematicians and cryptologists. NSA is so secret that an industry joke goes:

Q: What does NSA stand for?
A: No Such Agency.

Actually, NSA stands for National Security Agency.

After these discussions took place, IBM reduced the key from 128 bits to 56 bits and decided to keep secret the process by which DES was designed. Many people suspected that the key length was reduced to make sure that NSA could just break DES, but no organization with a smaller budget could. The point of the secret design was supposedly to hide a back door that could make it even easier for NSA to break DES. When an NSA employee discreetly told IEEE to cancel a planned conference on cryptography, that did not make people any more comfortable. NSA denied everything.

In 1977, two Stanford cryptography researchers, Diffie and Hellman (1977), designed a machine to break DES and estimated that it could be built for 20 million dollars. Given a small piece of plaintext and matched ciphertext, this machine could find the key by exhaustive search of the 2^{56}-entry key space in under 1 day. Nowadays, the game is up. Such a machine exists, is for sale, and costs less than $10,000 to make (Kumar et al., 2006).

Triple DES

As early as 1979, IBM realized that the DES key length was too short and devised a way to effectively increase it, using triple encryption (Tuchman, 1979). The method chosen, which has since been incorporated in International Standard 8732, is illustrated in Fig. 8-8. Here, two keys and three stages are used. In the first stage, the plaintext is encrypted using DES in the usual way with K_1. In the second stage, DES is run in decryption mode, using K_2 as the key. Finally, another DES encryption is done with K_1.

This design immediately gives rise to two questions. First, why are only two keys used, instead of three? Second, why is **EDE** (**Encrypt Decrypt Encrypt**) used, instead of **EEE** (**Encrypt Encrypt Encrypt**)? The reason that two keys are used is that even the most paranoid of cryptographers believe that 112 bits is

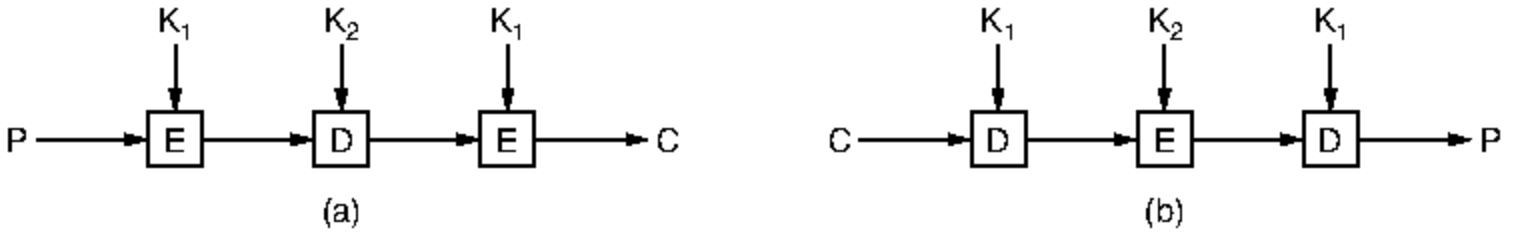


Figure 8-8. (a) Triple encryption using DES. (b) Decryption.

adequate for routine commercial applications for the time being. (And among cryptographers, paranoia is considered a feature, not a bug.) Going to 168 bits would just add the unnecessary overhead of managing and transporting another key for little real gain.

The reason for encrypting, decrypting, and then encrypting again is backward compatibility with existing single-key DES systems. Both the encryption and decryption functions are mappings between sets of 64-bit numbers. From a cryptographic point of view, the two mappings are equally strong. By using EDE, however, instead of EEE, a computer using triple encryption can speak to one using single encryption by just setting $K_1 = K_2$. This property allows triple encryption to be phased in gradually, something of no concern to academic cryptographers but of considerable importance to IBM and its customers.

8.2.2 AES—The Advanced Encryption Standard

As DES began approaching the end of its useful life, even with triple DES, **NIST** (**National Institute of Standards and Technology**), the agency of the U.S. Dept. of Commerce charged with approving standards for the U.S. Federal Government, decided that the government needed a new cryptographic standard for unclassified use. NIST was keenly aware of all the controversy surrounding DES and well knew that if it just announced a new standard, everyone knowing anything about cryptography would automatically assume that NSA had built a back door into it so NSA could read everything encrypted with it. Under these conditions, probably no one would use the standard and it would have died quietly.

So, NIST took a surprisingly different approach for a government bureaucracy: it sponsored a cryptographic bake-off (contest). In January 1997, researchers from all over the world were invited to submit proposals for a new standard, to be called **AES** (**Advanced Encryption Standard**). The bake-off rules were:

1. The algorithm must be a symmetric block cipher.
2. The full design must be public.
3. Key lengths of 128, 192, and 256 bits must be supported.

4. Both software and hardware implementations must be possible.
5. The algorithm must be public or licensed on nondiscriminatory terms.

Fifteen serious proposals were made, and public conferences were organized in which they were presented and attendees were actively encouraged to find flaws in all of them. In August 1998, NIST selected five finalists, primarily on the basis of their security, efficiency, simplicity, flexibility, and memory requirements (important for embedded systems). More conferences were held and more potshots taken.

In October 2000, NIST announced that it had selected Rijndael, by Joan Daemen and Vincent Rijmen. The name Rijndael, pronounced Rhine-doll (more or less), is derived from the last names of the authors: Rijmen + Daemen. In November 2001, Rijndael became the AES U.S. Government standard, published as FIPS (Federal Information Processing Standard) 197. Due to the extraordinary openness of the competition, the technical properties of Rijndael, and the fact that the winning team consisted of two young Belgian cryptographers (who were unlikely to have built in a back door just to please NSA), Rijndael has become the world's dominant cryptographic cipher. AES encryption and decryption is now part of the instruction set for some microprocessors (e.g., Intel).

Rijndael supports key lengths and block sizes from 128 bits to 256 bits in steps of 32 bits. The key length and block length may be chosen independently. However, AES specifies that the block size must be 128 bits and the key length must be 128, 192, or 256 bits. It is doubtful that anyone will ever use 192-bit keys, so de facto, AES has two variants: a 128-bit block with a 128-bit key and a 128-bit block with a 256-bit key.

In our treatment of the algorithm, we will examine only the 128/128 case because this is likely to become the commercial norm. A 128-bit key gives a key space of $2^{128} \approx 3 \times 10^{38}$ keys. Even if NSA manages to build a machine with 1 billion parallel processors, each being able to evaluate one key per picosecond, it would take such a machine about 10^{10} years to search the key space. By then the sun will have burned out, so the folks then present will have to read the results by candlelight.

Rijndael

From a mathematical perspective, Rijndael is based on Galois field theory, which gives it some provable security properties. However, it can also be viewed as C code, without getting into the mathematics.

Like DES, Rijndael uses substitution and permutations, and it also uses multiple rounds. The number of rounds depends on the key size and block size, being 10 for 128-bit keys with 128-bit blocks and moving up to 14 for the largest key or the largest block. However, unlike DES, all operations involve entire bytes, to

allow for efficient implementations in both hardware and software. An outline of the code is given in Fig. 8-9. Note that this code is for the purpose of illustration. Good implementations of security code will follow additional practices, such as zeroing out sensitive memory after it has been used. See, for example, Ferguson et al. (2010).

```
#define LENGTH 16                                   /* # bytes in data block or key */
#define NROWS 4                                     /* number of rows in state */
#define NCOLS 4                                     /* number of columns in state */
#define ROUNDS 10                                   /* number of iterations */
typedef unsigned char byte;                         /* unsigned 8-bit integer */

rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
  int r;                                            /* loop index */
  byte state[NROWS][NCOLS];                         /* current state */
  struct {byte k[NROWS][NCOLS];} rk[ROUNDS + 1];    /* round keys */

  expand_key(key, rk);                              /* construct the round keys */
  copy_plaintext_to_state(state, plaintext);        /* init current state */
  xor_roundkey_into_state(state, rk[0]);            /* XOR key into state */

  for (r = 1; r <= ROUNDS; r++) {
      substitute(state);                            /* apply S-box to each byte */
      rotate_rows(state);                           /* rotate row i by i bytes */
      if (r < ROUNDS) mix_columns(state);           /* mix function */
      xor_roundkey_into_state(state, rk[r]);        /* XOR key into state */
  }
  copy_state_to_ciphertext(ciphertext, state);      /* return result */
}
```

Figure 8-9. An outline of Rijndael in C.

The function *rijndael* has three parameters. They are: *plaintext*, an array of 16 bytes containing the input data; *ciphertext*, an array of 16 bytes where the enciphered output will be returned; and *key*, the 16-byte key. During the calculation, the current state of the data is maintained in a byte array, *state*, whose size is $NROWS \times NCOLS$. For 128-bit blocks, this array is 4×4 bytes. With 16 bytes, the full 128-bit data block can be stored.

The *state* array is initialized to the plaintext and modified by every step in the computation. In some steps, byte-for-byte substitution is performed. In others, the bytes are permuted within the array. Other transformations are also used. At the end, the contents of the *state* are returned as the ciphertext.

The code starts out by expanding the key into 11 arrays of the same size as the state. They are stored in *rk*, which is an array of structs, each containing a state array. One of these will be used at the start of the calculation and the other 10 will be used during the 10 rounds, one per round. The calculation of the round

keys from the encryption key is too complicated for us to get into here. Suffice it to say that the round keys are produced by repeated rotation and XORing of various groups of key bits. For all the details, see Daemen and Rijmen (2002).

The next step is to copy the plaintext into the *state* array so it can be processed during the rounds. It is copied in column order, with the first 4 bytes going into column 0, the next 4 bytes going into column 1, and so on. Both the columns and the rows are numbered starting at 0, although the rounds are numbered starting at 1. This initial setup of the 12 byte arrays of size 4×4 is illustrated in Fig. 8-10.

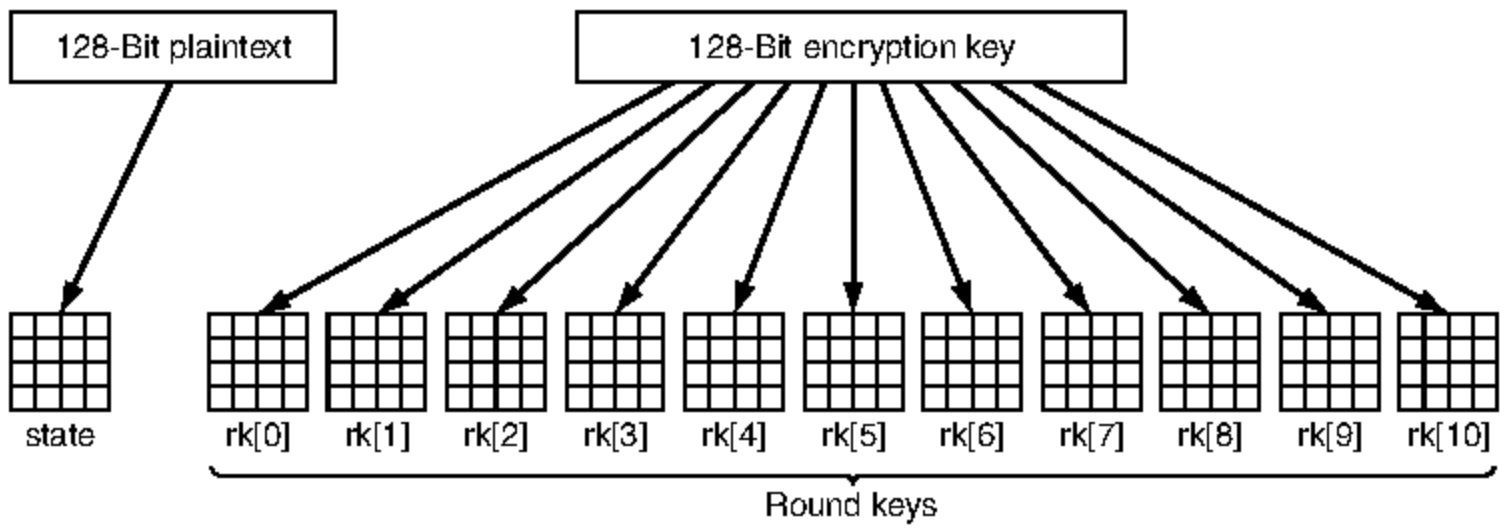


Figure 8-10. Creating the *state* and *rk* arrays.

There is one more step before the main computation begins: *rk*[0] is XORed into *state*, byte for byte. In other words, each of the 16 bytes in *state* is replaced by the XOR of itself and the corresponding byte in *rk*[0].

Now it is time for the main attraction. The loop executes 10 iterations, one per round, transforming *state* on each iteration. The contents of each round is produced in four steps. Step 1 does a byte-for-byte substitution on *state*. Each byte in turn is used as an index into an S-box to replace its value by the contents of that S-box entry. This step is a straight monoalphabetic substitution cipher. Unlike DES, which has multiple S-boxes, Rijndael has only one S-box.

Step 2 rotates each of the four rows to the left. Row 0 is rotated 0 bytes (i.e., not changed), row 1 is rotated 1 byte, row 2 is rotated 2 bytes, and row 3 is rotated 3 bytes. This step diffuses the contents of the current data around the block, analogous to the permutations of Fig. 8-6.

Step 3 mixes up each column independently of the other ones. The mixing is done using matrix multiplication in which the new column is the product of the old column and a constant matrix, with the multiplication done using the finite Galois field, $GF(2^8)$. Although this may sound complicated, an algorithm exists that allows each element of the new column to be computed using two table lookups and three XORs (Daemen and Rijmen, 2002, Appendix E).

Finally, step 4 XORs the key for this round into the *state* array for use in the next round.

Since every step is reversible, decryption can be done just by running the algorithm backward. However, there is also a trick available in which decryption can be done by running the encryption algorithm using different tables.

The algorithm has been designed not only for great security, but also for great speed. A good software implementation on a 2-GHz machine should be able to achieve an encryption rate of 700 Mbps, which is fast enough to encrypt over 100 MPEG-2 videos in real time. Hardware implementations are faster still.

8.2.3 Cipher Modes

Despite all this complexity, AES (or DES, or any block cipher for that matter) is basically a monoalphabetic substitution cipher using big characters (128-bit characters for AES and 64-bit characters for DES). Whenever the same plaintext block goes in the front end, the same ciphertext block comes out the back end. If you encrypt the plaintext *abcdefgh* 100 times with the same DES key, you get the same ciphertext 100 times. An intruder can exploit this property to help subvert the cipher.

Electronic Code Book Mode

To see how this monoalphabetic substitution cipher property can be used to partially defeat the cipher, we will use (triple) DES because it is easier to depict 64-bit blocks than 128-bit blocks, but AES has exactly the same problem. The straightforward way to use DES to encrypt a long piece of plaintext is to break it up into consecutive 8-byte (64-bit) blocks and encrypt them one after another with the same key. The last piece of plaintext is padded out to 64 bits, if need be. This technique is known as **ECB mode** (**Electronic Code Book mode**) in analogy with old-fashioned code books where each plaintext word was listed, followed by its ciphertext (usually a five-digit decimal number).

In Fig. 8-11, we have the start of a computer file listing the annual bonuses a company has decided to award to its employees. This file consists of consecutive 32-byte records, one per employee, in the format shown: 16 bytes for the name, 8 bytes for the position, and 8 bytes for the bonus. Each of the sixteen 8-byte blocks (numbered from 0 to 15) is encrypted by (triple) DES.

Leslie just had a fight with the boss and is not expecting much of a bonus. Kim, in contrast, is the boss' favorite, and everyone knows this. Leslie can get access to the file after it is encrypted but before it is sent to the bank. Can Leslie rectify this unfair situation, given only the encrypted file?

No problem at all. All Leslie has to do is make a copy of the 12th ciphertext block (which contains Kim's bonus) and use it to replace the fourth ciphertext block (which contains Leslie's bonus). Even without knowing what the 12th

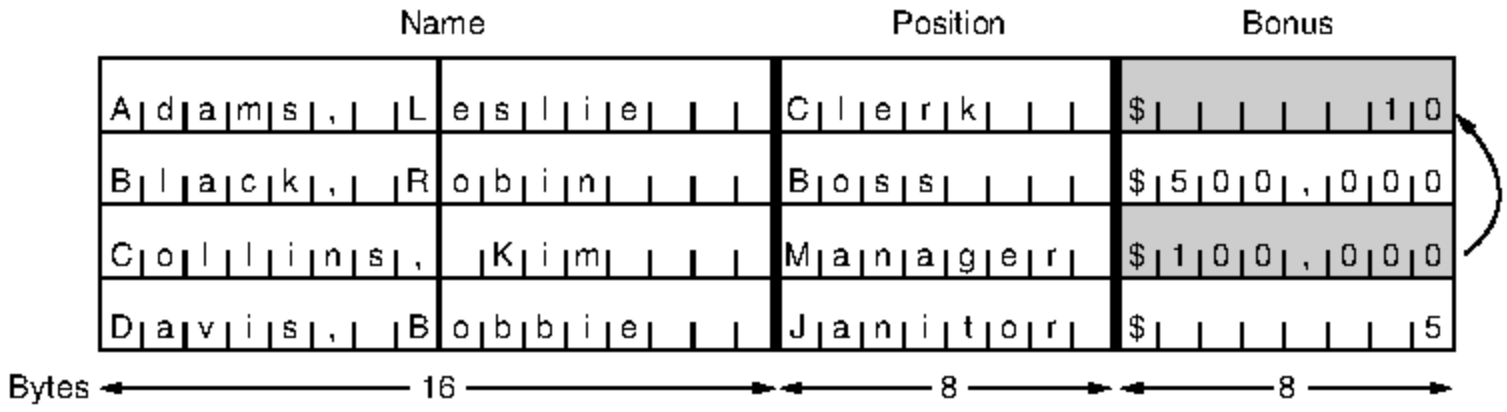


Figure 8-11. The plaintext of a file encrypted as 16 DES blocks.

block says, Leslie can expect to have a much merrier Christmas this year. (Copying the eighth ciphertext block is also a possibility, but is more likely to be detected; besides, Leslie is not a greedy person.)

Cipher Block Chaining Mode

To thwart this type of attack, all block ciphers can be chained in various ways so that replacing a block the way Leslie did will cause the plaintext decrypted starting at the replaced block to be garbage. One way of chaining is **cipher block chaining**. In this method, shown in Fig. 8-12, each plaintext block is XORed with the previous ciphertext block before being encrypted. Consequently, the same plaintext block no longer maps onto the same ciphertext block, and the encryption is no longer a big monoalphabetic substitution cipher. The first block is XORed with a randomly chosen **IV** (**Initialization Vector**), which is transmitted (in plaintext) along with the ciphertext.

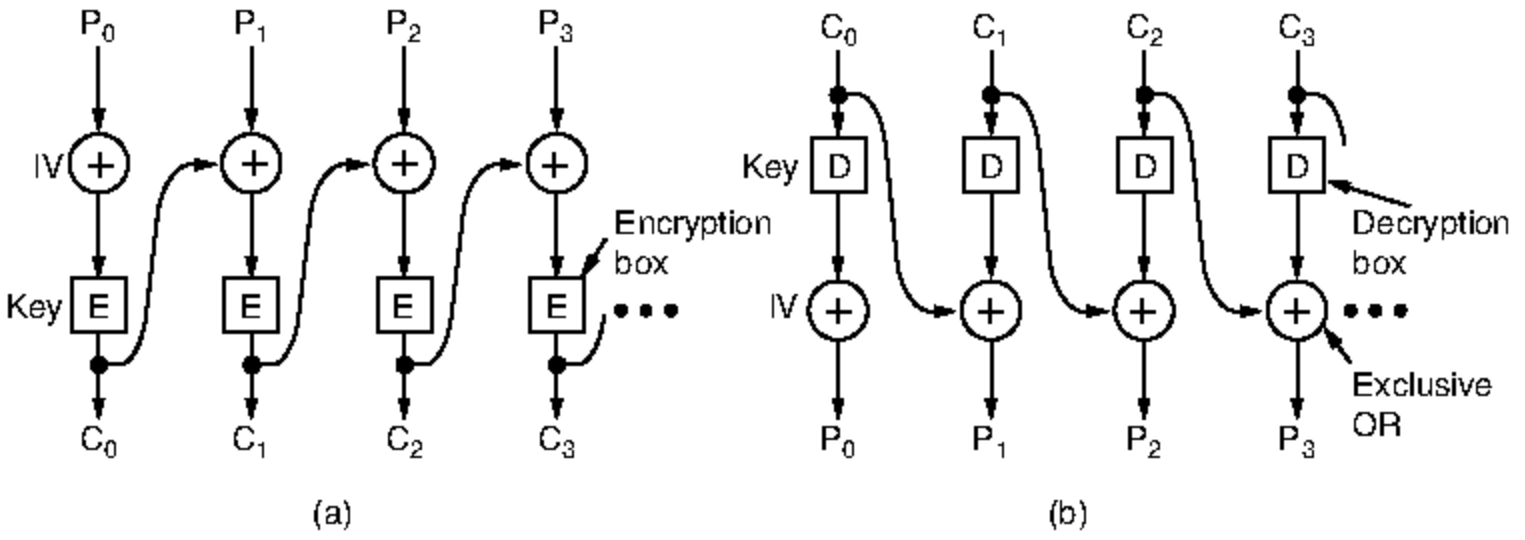


Figure 8-12. Cipher block chaining. (a) Encryption. (b) Decryption.

We can see how cipher block chaining mode works by examining the example of Fig. 8-12. We start out by computing $C_0 = E(P_0 \text{ XOR } IV)$. Then we compute $C_1 = E(P_1 \text{ XOR } C_0)$, and so on. Decryption also uses XOR to reverse the process, with $P_0 = IV \text{ XOR } D(C_0)$, and so on. Note that the encryption of block i is a

function of all the plaintext in blocks 0 through $i-1$, so the same plaintext generates different ciphertext depending on where it occurs. A transformation of the type Leslie made will result in nonsense for two blocks starting at Leslie's bonus field. To an astute security officer, this peculiarity might suggest where to start the ensuing investigation.

Cipher block chaining also has the advantage that the same plaintext block will not result in the same ciphertext block, making cryptanalysis more difficult. In fact, this is the main reason it is used.

Cipher Feedback Mode

However, cipher block chaining has the disadvantage of requiring an entire 64-bit block to arrive before decryption can begin. For byte-by-byte encryption, **cipher feedback mode** using (triple) DES is used, as shown in Fig. 8-13. For AES, the idea is exactly the same, only a 128-bit shift register is used. In this figure, the state of the encryption machine is shown after bytes 0 through 9 have been encrypted and sent. When plaintext byte 10 arrives, as illustrated in Fig. 8-13(a), the DES algorithm operates on the 64-bit shift register to generate a 64-bit ciphertext. The leftmost byte of that ciphertext is extracted and XORed with P_{10}. That byte is transmitted on the transmission line. In addition, the shift register is shifted left 8 bits, causing C_2 to fall off the left end, and C_{10} is inserted in the position just vacated at the right end by C_9.

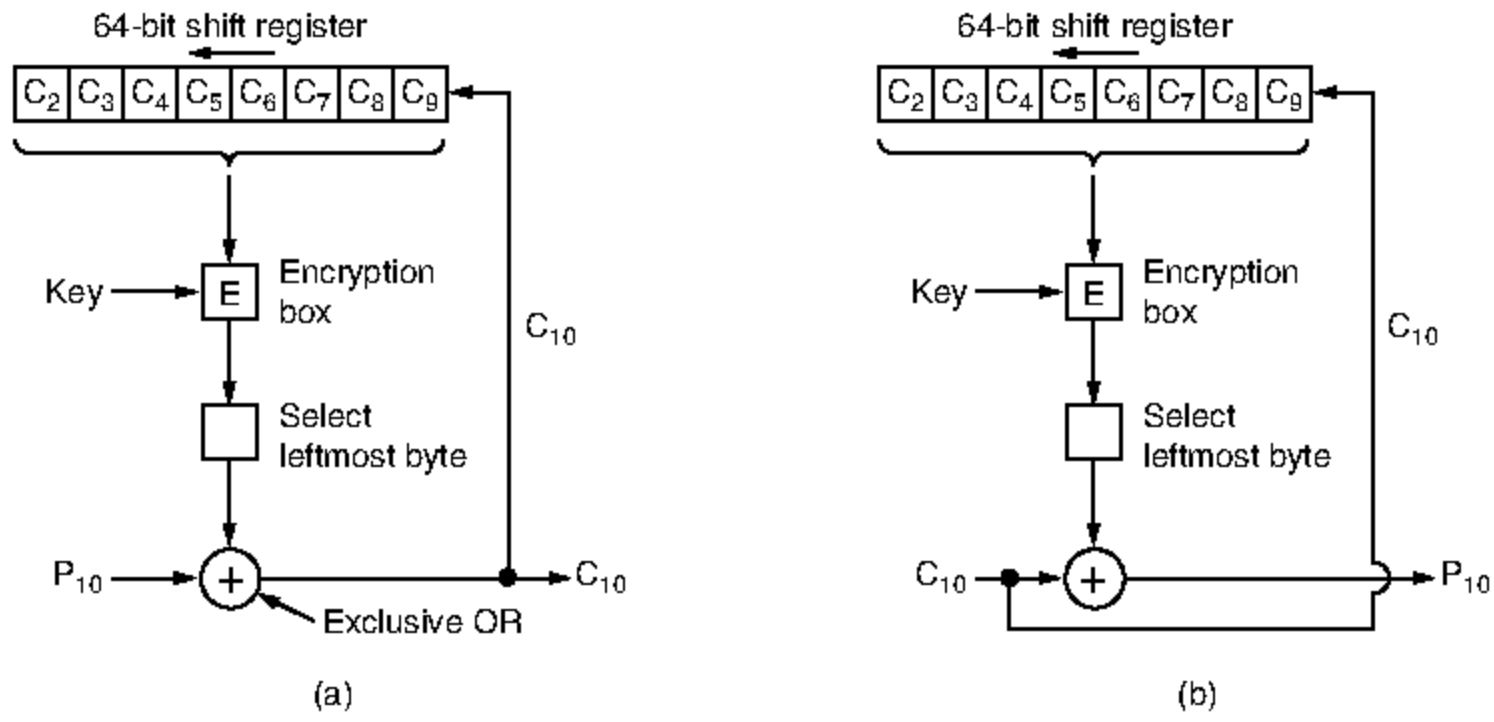


Figure 8-13. Cipher feedback mode. (a) Encryption. (b) Decryption.

Note that the contents of the shift register depend on the entire previous history of the plaintext, so a pattern that repeats multiple times in the plaintext will be encrypted differently each time in the ciphertext. As with cipher block chaining, an initialization vector is needed to start the ball rolling.

Decryption with cipher feedback mode works the same way as encryption. In particular, the content of the shift register is *encrypted*, not *decrypted*, so the selected byte that is XORed with C_{10} to get P_{10} is the same one that was XORed with P_{10} to generate C_{10} in the first place. As long as the two shift registers remain identical, decryption works correctly. This is illustrated in Fig. 8-13(b).

A problem with cipher feedback mode is that if one bit of the ciphertext is accidentally inverted during transmission, the 8 bytes that are decrypted while the bad byte is in the shift register will be corrupted. Once the bad byte is pushed out of the shift register, correct plaintext will once again be generated. Thus, the effects of a single inverted bit are relatively localized and do not ruin the rest of the message, but they do ruin as many bits as the shift register is wide.

Stream Cipher Mode

Nevertheless, applications exist in which having a 1-bit transmission error mess up 64 bits of plaintext is too large an effect. For these applications, a fourth option, **stream cipher mode**, exists. It works by encrypting an initialization vector, using a key to get an output block. The output block is then encrypted, using the key to get a second output block. This block is then encrypted to get a third block, and so on. The (arbitrarily large) sequence of output blocks, called the **keystream**, is treated like a one-time pad and XORed with the plaintext to get the ciphertext, as shown in Fig. 8-14(a). Note that the IV is used only on the first step. After that, the output is encrypted. Also note that the keystream is independent of the data, so it can be computed in advance, if need be, and is completely insensitive to transmission errors. Decryption is shown in Fig. 8-14(b).

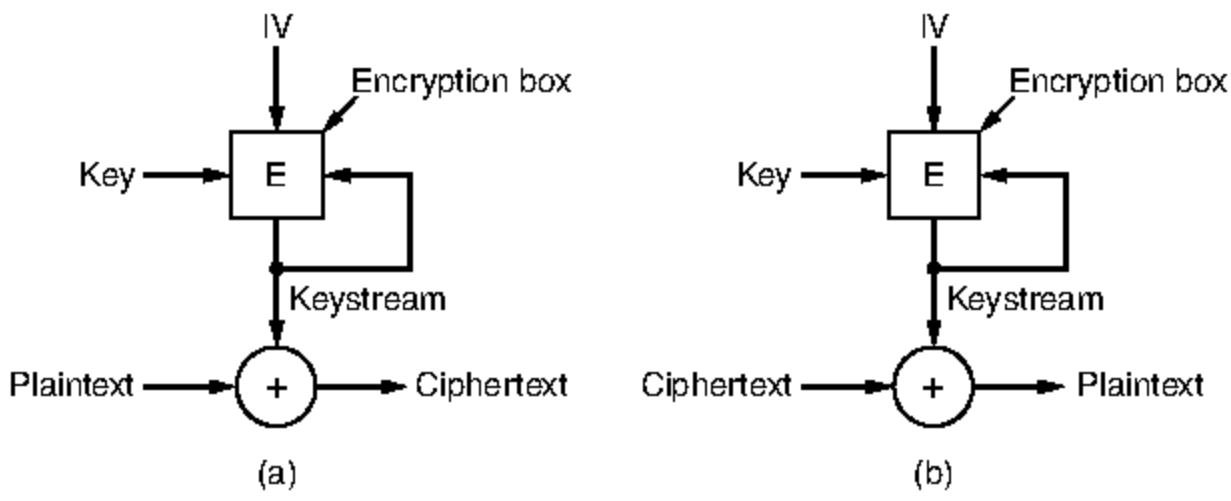


Figure 8-14. A stream cipher. (a) Encryption. (b) Decryption.

Decryption occurs by generating the same keystream at the receiving side. Since the keystream depends only on the IV and the key, it is not affected by transmission errors in the ciphertext. Thus, a 1-bit error in the transmitted ciphertext generates only a 1-bit error in the decrypted plaintext.

It is essential never to use the same (key, IV) pair twice with a stream cipher because doing so will generate the same keystream each time. Using the same keystream twice exposes the ciphertext to a **keystream reuse attack**. Imagine that the plaintext block, P_0, is encrypted with the keystream to get P_0 XOR K_0. Later, a second plaintext block, Q_0, is encrypted with the same keystream to get Q_0 XOR K_0. An intruder who captures both of these ciphertext blocks can simply XOR them together to get P_0 XOR Q_0, which eliminates the key. The intruder now has the XOR of the two plaintext blocks. If one of them is known or can be guessed, the other can also be found. In any event, the XOR of two plaintext streams can be attacked by using statistical properties of the message. For example, for English text, the most common character in the stream will probably be the XOR of two spaces, followed by the XOR of space and the letter "e", etc. In short, equipped with the XOR of two plaintexts, the cryptanalyst has an excellent chance of deducing both of them.

Counter Mode

One problem that all the modes except electronic code book mode have is that random access to encrypted data is impossible. For example, suppose a file is transmitted over a network and then stored on disk in encrypted form. This might be a reasonable way to operate if the receiving computer is a notebook computer that might be stolen. Storing all critical files in encrypted form greatly reduces the damage due to secret information leaking out in the event that the computer falls into the wrong hands.

However, disk files are often accessed in nonsequential order, especially files in databases. With a file encrypted using cipher block chaining, accessing a random block requires first decrypting all the blocks ahead of it, an expensive proposition. For this reason, yet another mode has been invented: **counter mode**, as illustrated in Fig. 8-15. Here, the plaintext is not encrypted directly. Instead, the initialization vector plus a constant is encrypted, and the resulting ciphertext is XORed with the plaintext. By stepping the initialization vector by 1 for each new block, it is easy to decrypt a block anywhere in the file without first having to decrypt all of its predecessors.

Although counter mode is useful, it has a weakness that is worth pointing out. Suppose that the same key, K, is used again in the future (with a different plaintext but the same IV) and an attacker acquires all the ciphertext from both runs. The keystreams are the same in both cases, exposing the cipher to a keystream reuse attack of the same kind we saw with stream ciphers. All the cryptanalyst has to do is XOR the two ciphertexts together to eliminate all the cryptographic protection and just get the XOR of the plaintexts. This weakness does not mean counter mode is a bad idea. It just means that both keys and initialization vectors should be chosen independently and at random. Even if the same key is accidentally used twice, if the IV is different each time, the plaintext is safe.

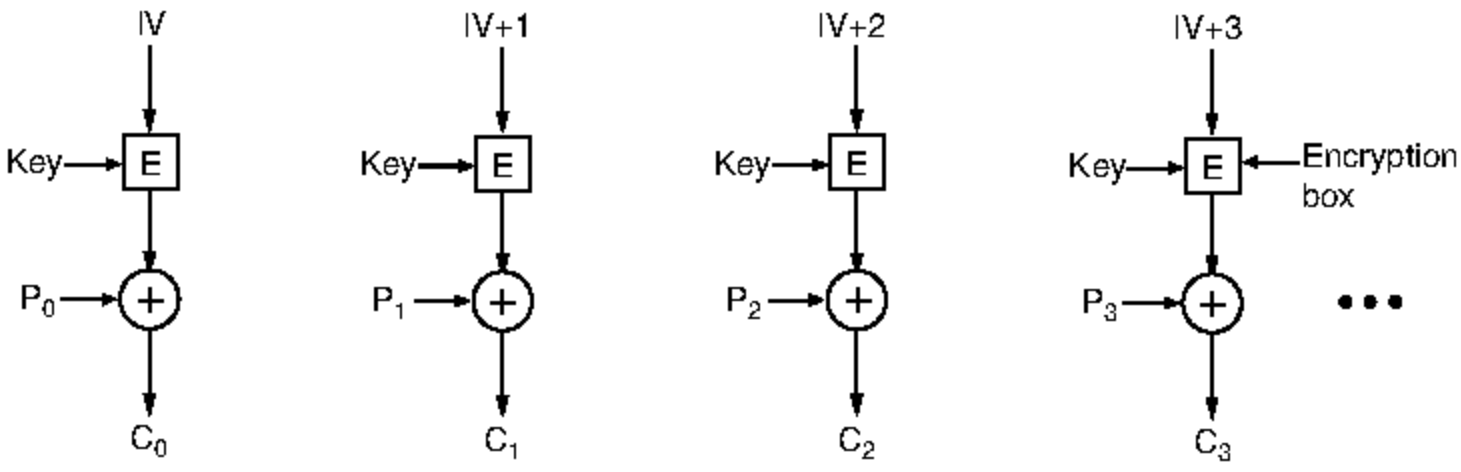


Figure 8-15. Encryption using counter mode.

8.2.4 Other Ciphers

AES (Rijndael) and DES are the best-known symmetric-key cryptographic algorithms, and the standard industry choices, if only for liability reasons. (No one will blame you if you use AES in your product and AES is cracked, but they will certainly blame you if you use a nonstandard cipher and it is later broken.) However, it is worth mentioning that numerous other symmetric-key ciphers have been devised. Some of these are embedded inside various products. A few of the more common ones are listed in Fig. 8-16. It is possible to use combinations of these ciphers, for example, AES over Twofish, so that both ciphers need to be broken to recover the data.

Cipher	Author	Key length	Comments
DES	IBM	56 bits	Too weak to use now
RC4	Ronald Rivest	1–2048 bits	Caution: some keys are weak
RC5	Ronald Rivest	128–256 bits	Good, but patented
AES (Rijndael)	Daemen and Rijmen	128–256 bits	Best choice
Serpent	Anderson, Biham, Knudsen	128–256 bits	Very strong
Triple DES	IBM	168 bits	Good, but getting old
Twofish	Bruce Schneier	128–256 bits	Very strong; widely used

Figure 8-16. Some common symmetric-key cryptographic algorithms.

8.2.5 Cryptanalysis

Before leaving the subject of symmetric-key cryptography, it is worth at least mentioning four developments in cryptanalysis. The first development is **differential cryptanalysis** (Biham and Shamir, 1997). This technique can be used

to attack any block cipher. It works by beginning with a pair of plaintext blocks differing in only a small number of bits and watching carefully what happens on each internal iteration as the encryption proceeds. In many cases, some bit patterns are more common than others, which can lead to probabilistic attacks.

The second development worth noting is **linear cryptanalysis** (Matsui, 1994). It can break DES with only 2^{43} known plaintexts. It works by XORing certain bits in the plaintext and ciphertext together and examining the result. When done repeatedly, half the bits should be 0s and half should be 1s. Often, however, ciphers introduce a bias in one direction or the other, and this bias, however small, can be exploited to reduce the work factor. For the details, see Matsui's paper.

The third development is using analysis of electrical power consumption to find secret keys. Computers typically use around 3 volts to represent a 1 bit and 0 volts to represent a 0 bit. Thus, processing a 1 takes more electrical energy than processing a 0. If a cryptographic algorithm consists of a loop in which the key bits are processed in order, an attacker who replaces the main n-GHz clock with a slow (e.g., 100-Hz) clock and puts alligator clips on the CPU's power and ground pins can precisely monitor the power consumed by each machine instruction. From this data, deducing the key is surprisingly easy. This kind of cryptanalysis can be defeated only by carefully coding the algorithm in assembly language to make sure power consumption is independent of the key and also independent of all the individual round keys.

The fourth development is timing analysis. Cryptographic algorithms are full of if statements that test bits in the round keys. If the then and else parts take different amounts of time, by slowing down the clock and seeing how long various steps take, it may also be possible to deduce the round keys. Once all the round keys are known, the original key can usually be computed. Power and timing analysis can also be employed simultaneously to make the job easier. While power and timing analysis may seem exotic, in reality they are powerful techniques that can break any cipher not specifically designed to resist them.

8.3 PUBLIC-KEY ALGORITHMS

Historically, distributing the keys has always been the weakest link in most cryptosystems. No matter how strong a cryptosystem was, if an intruder could steal the key, the system was worthless. Cryptologists always took for granted that the encryption key and decryption key were the same (or easily derived from one another). But the key had to be distributed to all users of the system. Thus, it seemed as if there was an inherent problem. Keys had to be protected from theft, but they also had to be distributed, so they could not be locked in a bank vault.

In 1976, two researchers at Stanford University, Diffie and Hellman (1976), proposed a radically new kind of cryptosystem, one in which the encryption and decryption keys were so different that the decryption key could not feasibly be

derived from the encryption key. In their proposal, the (keyed) encryption algorithm, E, and the (keyed) decryption algorithm, D, had to meet three requirements. These requirements can be stated simply as follows:

1. $D(E(P)) = P$.
2. It is exceedingly difficult to deduce D from E.
3. E cannot be broken by a chosen plaintext attack.

The first requirement says that if we apply D to an encrypted message, $E(P)$, we get the original plaintext message, P, back. Without this property, the legitimate receiver could not decrypt the ciphertext. The second requirement speaks for itself. The third requirement is needed because, as we shall see in a moment, intruders may experiment with the algorithm to their hearts' content. Under these conditions, there is no reason that the encryption key cannot be made public.

The method works like this. A person, say, Alice, who wants to receive secret messages, first devises two algorithms meeting the above requirements. The encryption algorithm and Alice's key are then made public, hence the name **public-key cryptography**. Alice might put her public key on her home page on the Web, for example. We will use the notation E_A to mean the encryption algorithm parameterized by Alice's public key. Similarly, the (secret) decryption algorithm parameterized by Alice's private key is D_A. Bob does the same thing, publicizing E_B but keeping D_B secret.

Now let us see if we can solve the problem of establishing a secure channel between Alice and Bob, who have never had any previous contact. Both Alice's encryption key, E_A, and Bob's encryption key, E_B, are assumed to be in publicly readable files. Now Alice takes her first message, P, computes $E_B(P)$, and sends it to Bob. Bob then decrypts it by applying his secret key D_B [i.e., he computes $D_B(E_B(P)) = P$]. No one else can read the encrypted message, $E_B(P)$, because the encryption system is assumed to be strong and because it is too difficult to derive D_B from the publicly known E_B. To send a reply, R, Bob transmits $E_A(R)$. Alice and Bob can now communicate securely.

A note on terminology is perhaps useful here. Public-key cryptography requires each user to have two keys: a public key, used by the entire world for encrypting messages to be sent to that user, and a private key, which the user needs for decrypting messages. We will consistently refer to these keys as the *public* and *private* keys, respectively, and distinguish them from the *secret* keys used for conventional symmetric-key cryptography.

8.3.1 RSA

The only catch is that we need to find algorithms that indeed satisfy all three requirements. Due to the potential advantages of public-key cryptography, many researchers are hard at work, and some algorithms have already been published.

One good method was discovered by a group at M.I.T. (Rivest et al., 1978). It is known by the initials of the three discoverers (Rivest, Shamir, Adleman): **RSA**. It has survived all attempts to break it for more than 30 years and is considered very strong. Much practical security is based on it. For this reason, Rivest, Shamir, and Adleman were given the 2002 ACM Turing Award. Its major disadvantage is that it requires keys of at least 1024 bits for good security (versus 128 bits for symmetric-key algorithms), which makes it quite slow.

The RSA method is based on some principles from number theory. We will now summarize how to use the method; for details, consult the paper.

1. Choose two large primes, p and q (typically 1024 bits).
2. Compute $n = p \times q$ and $z = (p - 1) \times (q - 1)$.
3. Choose a number relatively prime to z and call it d.
4. Find e such that $e \times d = 1 \ mod \ z$.

With these parameters computed in advance, we are ready to begin encryption. Divide the plaintext (regarded as a bit string) into blocks, so that each plaintext message, P, falls in the interval $0 \le P < n$. Do that by grouping the plaintext into blocks of k bits, where k is the largest integer for which $2^k < n$ is true.

To encrypt a message, P, compute $C = P^e \pmod{n}$. To decrypt C, compute $P = C^d \pmod{n}$. It can be proven that for all P in the specified range, the encryption and decryption functions are inverses. To perform the encryption, you need e and n. To perform the decryption, you need d and n. Therefore, the public key consists of the pair (e, n) and the private key consists of (d, n).

The security of the method is based on the difficulty of factoring large numbers. If the cryptanalyst could factor the (publicly known) n, he could then find p and q, and from these z. Equipped with knowledge of z and e, d can be found using Euclid's algorithm. Fortunately, mathematicians have been trying to factor large numbers for at least 300 years, and the accumulated evidence suggests that it is an exceedingly difficult problem.

According to Rivest and colleagues, factoring a 500-digit number would require 10^{25} years using brute force. In both cases, they assumed the best known algorithm and a computer with a 1-μsec instruction time. With a million chips running in parallel, each with an instruction time of 1 nsec, it would still take 10^{16} years. Even if computers continue to get faster by an order of magnitude per decade, it will be many years before factoring a 500-digit number becomes feasible, at which time our descendants can simply choose p and q still larger.

A trivial pedagogical example of how the RSA algorithm works is given in Fig. 8-17. For this example, we have chosen $p = 3$ and $q = 11$, giving $n = 33$ and $z = 20$. A suitable value for d is $d = 7$, since 7 and 20 have no common factors. With these choices, e can be found by solving the equation $7e = 1 \pmod{20}$, which yields $e = 3$. The ciphertext, C, corresponding to a plaintext message, P, is

given by $C = P^3 \pmod{33}$. The ciphertext is decrypted by the receiver by making use of the rule $P = C^7 \pmod{33}$. The figure shows the encryption of the plaintext "SUZANNE" as an example.

Plaintext (P)			Ciphertext (C)		After decryption	
Symbolic	Numeric	P^3	$P^3 \pmod{33}$	C^7	$C^7 \pmod{33}$	Symbolic
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	01	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	05	E
Sender's computation				Receiver's computation		

Figure 8-17. An example of the RSA algorithm.

Because the primes chosen for this example are so small, P must be less than 33, so each plaintext block can contain only a single character. The result is a monoalphabetic substitution cipher, not very impressive. If instead we had chosen p and $q \approx 2^{512}$, we would have $n \approx 2^{1024}$, so each block could be up to 1024 bits or 128 eight-bit characters, versus 8 characters for DES and 16 characters for AES.

It should be pointed out that using RSA as we have described is similar to using a symmetric algorithm in ECB mode—the same input block gives the same output block. Therefore, some form of chaining is needed for data encryption. However, in practice, most RSA-based systems use public-key cryptography primarily for distributing one-time session keys for use with some symmetric-key algorithm such as AES or triple DES. RSA is too slow for actually encrypting large volumes of data but is widely used for key distribution.

8.3.2 Other Public-Key Algorithms

Although RSA is widely used, it is by no means the only public-key algorithm known. The first public-key algorithm was the knapsack algorithm (Merkle and Hellman, 1978). The idea here is that someone owns a large number of objects, each with a different weight. The owner encodes the message by secretly selecting a subset of the objects and placing them in the knapsack. The total weight of the objects in the knapsack is made public, as is the list of all possible objects and their corresponding weights. The list of objects in the knapsack is kept secret. With certain additional restrictions, the problem of figuring out a possible list of objects with the given weight was thought to be computationally infeasible and formed the basis of the public-key algorithm.

The algorithm's inventor, Ralph Merkle, was quite sure that this algorithm could not be broken, so he offered a $100 reward to anyone who could break it. Adi Shamir (the "S" in RSA) promptly broke it and collected the reward. Undeterred, Merkle strengthened the algorithm and offered a $1000 reward to anyone who could break the new one. Ronald Rivest (the "R" in RSA) promptly broke the new one and collected the reward. Merkle did not dare offer $10,000 for the next version, so "A" (Leonard Adleman) was out of luck. Nevertheless, the knapsack algorithm is not considered secure and is not used in practice any more.

Other public-key schemes are based on the difficulty of computing discrete logarithms. Algorithms that use this principle have been invented by El Gamal (1985) and Schnorr (1991).

A few other schemes exist, such as those based on elliptic curves (Menezes and Vanstone, 1993), but the two major categories are those based on the difficulty of factoring large numbers and computing discrete logarithms modulo a large prime. These problems are thought to be genuinely difficult to solve—mathematicians have been working on them for many years without any great breakthroughs.

8.4 DIGITAL SIGNATURES

The authenticity of many legal, financial, and other documents is determined by the presence or absence of an authorized handwritten signature. And photocopies do not count. For computerized message systems to replace the physical transport of paper-and-ink documents, a method must be found to allow documents to be signed in an unforgeable way.

The problem of devising a replacement for handwritten signatures is a difficult one. Basically, what is needed is a system by which one party can send a signed message to another party in such a way that the following conditions hold:

1. The receiver can verify the claimed identity of the sender.
2. The sender cannot later repudiate the contents of the message.
3. The receiver cannot possibly have concocted the message himself.

The first requirement is needed, for example, in financial systems. When a customer's computer orders a bank's computer to buy a ton of gold, the bank's computer needs to be able to make sure that the computer giving the order really belongs to the customer whose account is to be debited. In other words, the bank has to authenticate the customer (and the customer has to authenticate the bank).

The second requirement is needed to protect the bank against fraud. Suppose that the bank buys the ton of gold, and immediately thereafter the price of gold

drops sharply. A dishonest customer might then proceed to sue the bank, claiming that he never issued any order to buy gold. When the bank produces the message in court, the customer may deny having sent it. The property that no party to a contract can later deny having signed it is called **nonrepudiation**. The digital signature schemes that we will now study help provide it.

The third requirement is needed to protect the customer in the event that the price of gold shoots up and the bank tries to construct a signed message in which the customer asked for one bar of gold instead of one ton. In this fraud scenario, the bank just keeps the rest of the gold for itself.

8.4.1 Symmetric-Key Signatures

One approach to digital signatures is to have a central authority that knows everything and whom everyone trusts, say, Big Brother (*BB*). Each user then chooses a secret key and carries it by hand to *BB*'s office. Thus, only Alice and *BB* know Alice's secret key, K_A, and so on.

When Alice wants to send a signed plaintext message, *P*, to her banker, Bob, she generates $K_A(B, R_A, t, P)$, where *B* is Bob's identity, R_A is a random number chosen by Alice, *t* is a timestamp to ensure freshness, and $K_A(B, R_A, t, P)$ is the message encrypted with her key, K_A. Then she sends it as depicted in Fig. 8-18. *BB* sees that the message is from Alice, decrypts it, and sends a message to Bob as shown. The message to Bob contains the plaintext of Alice's message and also the signed message $K_{BB}(A, t, P)$. Bob now carries out Alice's request.

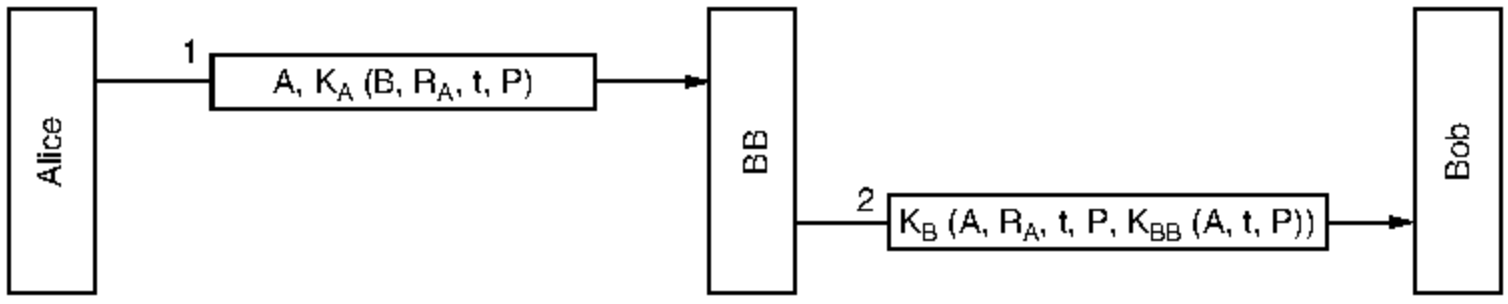


Figure 8-18. Digital signatures with Big Brother.

What happens if Alice later denies sending the message? Step 1 is that everyone sues everyone (at least, in the United States). Finally, when the case comes to court and Alice vigorously denies sending Bob the disputed message, the judge will ask Bob how he can be sure that the disputed message came from Alice and not from Trudy. Bob first points out that *BB* will not accept a message from Alice unless it is encrypted with K_A, so there is no possibility of Trudy sending *BB* a false message from Alice without BB detecting it immediately.

Bob then dramatically produces Exhibit A: $K_{BB}(A, t, P)$. Bob says that this is a message signed by *BB* that proves Alice sent *P* to Bob. The judge then asks *BB* (whom everyone trusts) to decrypt Exhibit A. When *BB* testifies that Bob is telling the truth, the judge decides in favor of Bob. Case dismissed.

One potential problem with the signature protocol of Fig. 8-18 is Trudy replaying either message. To minimize this problem, timestamps are used throughout. Furthermore, Bob can check all recent messages to see if R_A was used in any of them. If so, the message is discarded as a replay. Note that based on the timestamp, Bob will reject very old messages. To guard against instant replay attacks, Bob just checks the R_A of every incoming message to see if such a message has been received from Alice in the past hour. If not, Bob can safely assume this is a new request.

8.4.2 Public-Key Signatures

A structural problem with using symmetric-key cryptography for digital signatures is that everyone has to agree to trust Big Brother. Furthermore, Big Brother gets to read all signed messages. The most logical candidates for running the Big Brother server are the government, the banks, the accountants, and the lawyers. Unfortunately, none of these inspire total confidence in all citizens. Hence, it would be nice if signing documents did not require a trusted authority.

Fortunately, public-key cryptography can make an important contribution in this area. Let us assume that the public-key encryption and decryption algorithms have the property that $E(D(P)) = P$, in addition, of course, to the usual property that $D(E(P)) = P$. (RSA has this property, so the assumption is not unreasonable.) Assuming that this is the case, Alice can send a signed plaintext message, P, to Bob by transmitting $E_B(D_A(P))$. Note carefully that Alice knows her own (private) key, D_A, as well as Bob's public key, E_B, so constructing this message is something Alice can do.

When Bob receives the message, he transforms it using his private key, as usual, yielding $D_A(P)$, as shown in Fig. 8-19. He stores this text in a safe place and then applies E_A to get the original plaintext.

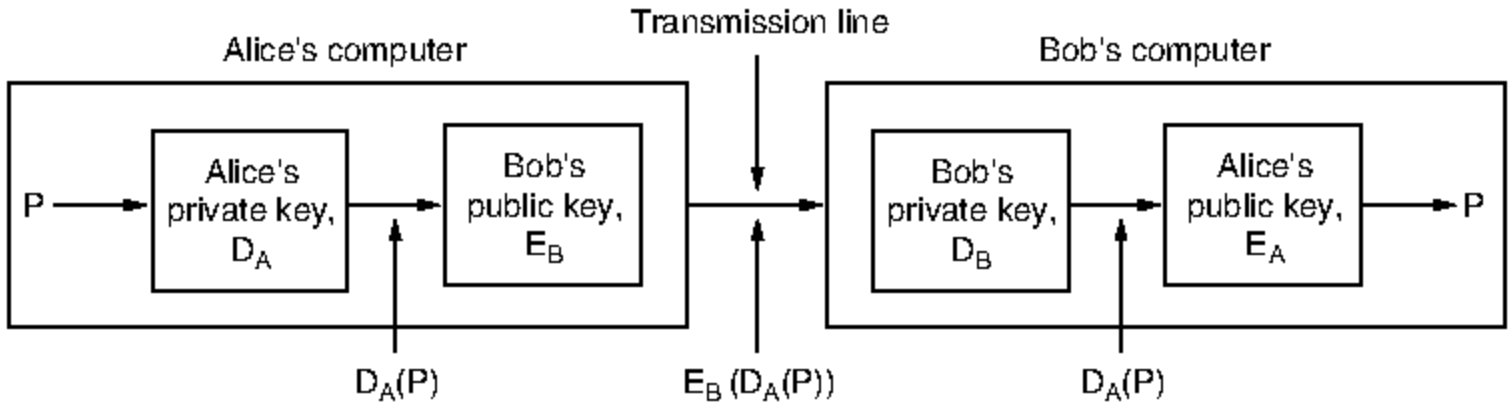


Figure 8-19. Digital signatures using public-key cryptography.

To see how the signature property works, suppose that Alice subsequently denies having sent the message P to Bob. When the case comes up in court, Bob can produce both P and $D_A(P)$. The judge can easily verify that Bob indeed has a valid message encrypted by D_A by simply applying E_A to it. Since Bob does not

know what Alice's private key is, the only way Bob could have acquired a message encrypted by it is if Alice did indeed send it. While in jail for perjury and fraud, Alice will have much time to devise interesting new public-key algorithms.

Although using public-key cryptography for digital signatures is an elegant scheme, there are problems that are related to the environment in which they operate rather than to the basic algorithm. For one thing, Bob can prove that a message was sent by Alice only as long as D_A remains secret. If Alice discloses her secret key, the argument no longer holds, because anyone could have sent the message, including Bob himself.

The problem might arise, for example, if Bob is Alice's stockbroker. Suppose that Alice tells Bob to buy a certain stock or bond. Immediately thereafter, the price drops sharply. To repudiate her message to Bob, Alice runs to the police claiming that her home was burglarized and the PC holding her key was stolen. Depending on the laws in her state or country, she may or may not be legally liable, especially if she claims not to have discovered the break-in until getting home from work, several hours after it allegedly happened.

Another problem with the signature scheme is what happens if Alice decides to change her key. Doing so is clearly legal, and it is probably a good idea to do so periodically. If a court case later arises, as described above, the judge will apply the *current* E_A to $D_A(P)$ and discover that it does not produce P. Bob will look pretty stupid at this point.

In principle, any public-key algorithm can be used for digital signatures. The de facto industry standard is the RSA algorithm. Many security products use it. However, in 1991, NIST proposed using a variant of the El Gamal public-key algorithm for its new **Digital Signature Standard** (**DSS**). El Gamal gets its security from the difficulty of computing discrete logarithms, rather than from the difficulty of factoring large numbers.

As usual when the government tries to dictate cryptographic standards, there was an uproar. DSS was criticized for being

1. Too secret (NSA designed the protocol for using El Gamal).
2. Too slow (10 to 40 times slower than RSA for checking signatures).
3. Too new (El Gamal had not yet been thoroughly analyzed).
4. Too insecure (fixed 512-bit key).

In a subsequent revision, the fourth point was rendered moot when keys up to 1024 bits were allowed. Nevertheless, the first two points remain valid.

8.4.3 Message Digests

One criticism of signature methods is that they often couple two distinct functions: authentication and secrecy. Often, authentication is needed but secrecy is not always needed. Also, getting an export license is often easier if the system in

question provides only authentication but not secrecy. Below we will describe an authentication scheme that does not require encrypting the entire message.

This scheme is based on the idea of a one-way hash function that takes an arbitrarily long piece of plaintext and from it computes a fixed-length bit string. This hash function, *MD*, often called a **message digest**, has four important properties:

1. Given P, it is easy to compute $MD(P)$.
2. Given $MD(P)$, it is effectively impossible to find P.
3. Given P, no one can find P' such that $MD(P') = MD(P)$.
4. A change to the input of even 1 bit produces a very different output.

To meet criterion 3, the hash should be at least 128 bits long, preferably more. To meet criterion 4, the hash must mangle the bits very thoroughly, not unlike the symmetric-key encryption algorithms we have seen.

Computing a message digest from a piece of plaintext is much faster than encrypting that plaintext with a public-key algorithm, so message digests can be used to speed up digital signature algorithms. To see how this works, consider the signature protocol of Fig. 8-18 again. Instead, of signing P with $K_{BB}(A, t, P)$, *BB* now computes the message digest by applying *MD* to P, yielding $MD(P)$. *BB* then encloses $K_{BB}(A, t, MD(P))$ as the fifth item in the list encrypted with K_B that is sent to Bob, instead of $K_{BB}(A, t, P)$.

If a dispute arises, Bob can produce both P and $K_{BB}(A, t, MD(P))$. After Big Brother has decrypted it for the judge, Bob has $MD(P)$, which is guaranteed to be genuine, and the alleged P. However, since it is effectively impossible for Bob to find any other message that gives this hash, the judge will easily be convinced that Bob is telling the truth. Using message digests in this way saves both encryption time and message transport costs.

Message digests work in public-key cryptosystems, too, as shown in Fig. 8-20. Here, Alice first computes the message digest of her plaintext. She then signs the message digest and sends both the signed digest and the plaintext to Bob. If Trudy replaces P along the way, Bob will see this when he computes $MD(P)$.

Figure 8-20. Digital signatures using message digests.

SHA-1 and SHA-2

A variety of message digest functions have been proposed. One of the most widely used functions is **SHA-1** (**Secure Hash Algorithm 1**) (NIST, 1993). Like all message digests, it operates by mangling bits in a sufficiently complicated way that every output bit is affected by every input bit. SHA-1 was developed by NSA and blessed by NIST in FIPS 180-1. It processes input data in 512-bit blocks, and it generates a 160-bit message digest. A typical way for Alice to send a nonsecret but signed message to Bob is illustrated in Fig. 8-21. Here, her plaintext message is fed into the SHA-1 algorithm to get a 160-bit SHA-1 hash. Alice then signs the hash with her RSA private key and sends both the plaintext message and the signed hash to Bob.

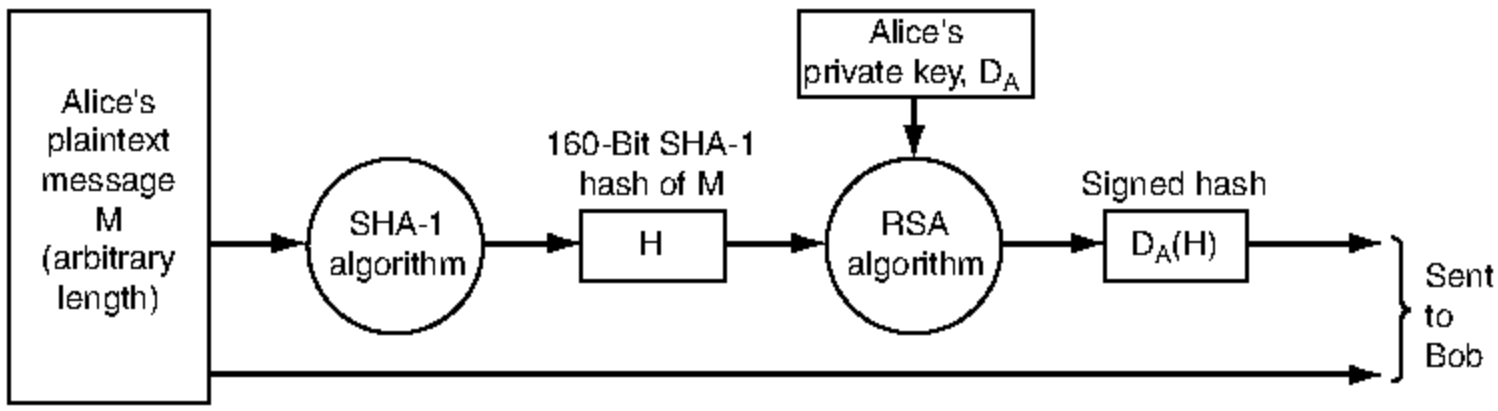


Figure 8-21. Use of SHA-1 and RSA for signing nonsecret messages.

After receiving the message, Bob computes the SHA-1 hash himself and also applies Alice's public key to the signed hash to get the original hash, H. If the two agree, the message is considered valid. Since there is no way for Trudy to modify the (plaintext) message while it is in transit and produce a new one that hashes to H, Bob can easily detect any changes Trudy has made to the message. For messages whose integrity is important but whose contents are not secret, the scheme of Fig. 8-21 is widely used. For a relatively small cost in computation, it guarantees that any modifications made to the plaintext message in transit can be detected with very high probability.

Now let us briefly see how SHA-1 works. It starts out by padding the message by adding a 1 bit to the end, followed by as many 0 bits as are necessary, but at least 64, to make the length a multiple of 512 bits. Then a 64-bit number containing the message length before padding is ORed into the low-order 64 bits. In Fig. 8-22, the message is shown with padding on the right because English text and figures go from left to right (i.e., the lower right is generally perceived as the end of the figure). With computers, this orientation corresponds to big-endian machines such as the SPARC and the IBM 360 and its successors, but SHA-1 always pads the end of the message, no matter which endian machine is used.

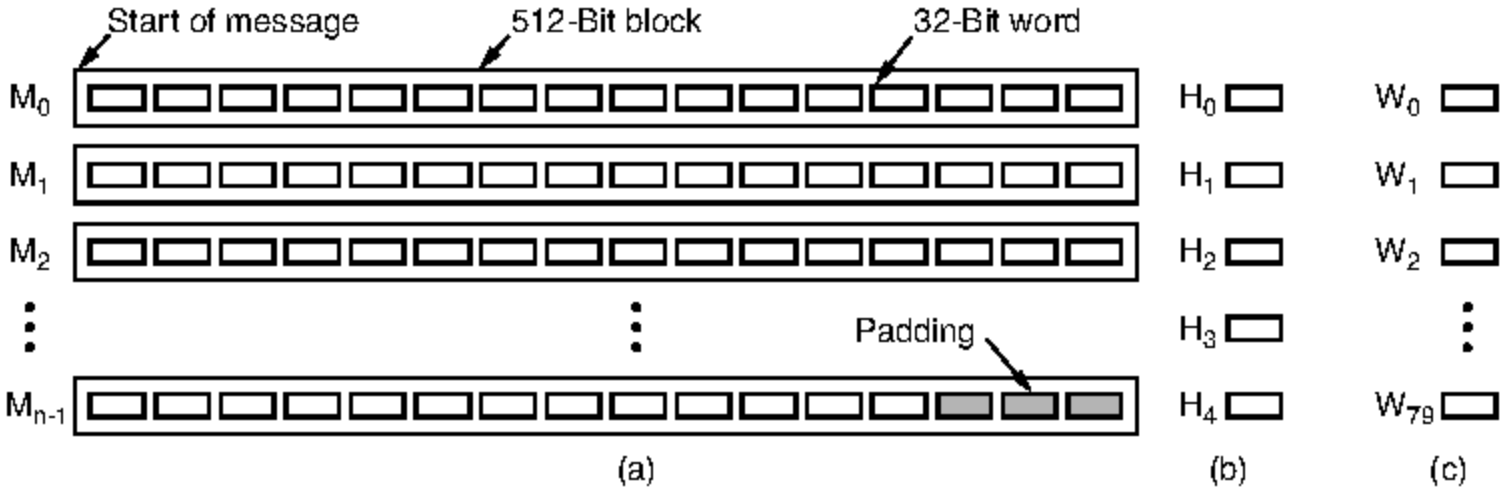


Figure 8-22. (a) A message padded out to a multiple of 512 bits. (b) The output variables. (c) The word array.

During the computation, SHA-1 maintains five 32-bit variables, H_0 through H_4, where the hash accumulates. These are shown in Fig. 8-22(b). They are initialized to constants specified in the standard.

Each of the blocks M_0 through M_{n-1} is now processed in turn. For the current block, the 16 words are first copied into the start of an auxiliary 80-word array, W, as shown in Fig. 8-22(c). Then the other 64 words in W are filled in using the formula

$$W_i = S^1(W_{i-3} \text{ XOR } W_{i-8} \text{ XOR } W_{i-14} \text{ XOR } W_{i-16}) \quad (16 \le i \le 79)$$

where $S^b(W)$ represents the left circular rotation of the 32-bit word, W, by b bits. Now five scratch variables, A through E, are initialized from H_0 through H_4, respectively.

The actual calculation can be expressed in pseudo-C as

```
for (i = 0; i < 80; i++) {
    temp = S^5(A) + f_i (B, C, D) + E + W_i + K_i;
    E = D;  D = C;  C = S^30(B);  B = A;  A = temp;
}
```

where the K_i constants are defined in the standard. The mixing functions f_i are defined as

$$f_i(B,C,D) = (B \text{ AND } C) \text{ OR } (\text{NOT } B \text{ AND } D) \qquad (0 \le i \le 19)$$
$$f_i(B,C,D) = B \text{ XOR } C \text{ XOR } D \qquad (20 \le i \le 39)$$
$$f_i(B,C,D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D) \qquad (40 \le i \le 59)$$
$$f_i(B,C,D) = B \text{ XOR } C \text{ XOR } D \qquad (60 \le i \le 79)$$

When all 80 iterations of the loop are completed, A through E are added to H_0 through H_4, respectively.

Now that the first 512-bit block has been processed, the next one is started. The W array is reinitialized from the new block, but H is left as it was. When this

block is finished, the next one is started, and so on, until all the 512-bit message blocks have been tossed into the soup. When the last block has been finished, the five 32-bit words in the *H* array are output as the 160-bit cryptographic hash. The complete C code for SHA-1 is given in RFC 3174.

New versions of SHA-1 have been developed that produce hashes of 224, 256, 384, and 512 bits. Collectively, these versions are called SHA-2. Not only are these hashes longer than SHA-1 hashes, but the digest function has been changed to combat some potential weaknesses of SHA-1. SHA-2 is not yet widely used, but it is likely to be in the future.

MD5

For completeness, we will mention another digest that is popular. **MD5** (Rivest, 1992) is the fifth in a series of message digests designed by Ronald Rivest. Very briefly, the message is padded to a length of 448 bits (modulo 512). Then the original length of the message is appended as a 64-bit integer to give a total input whose length is a multiple of 512 bits. Each round of the computation takes a 512-bit block of input and mixes it thoroughly with a running 128-bit buffer. For good measure, the mixing uses a table constructed from the sine function. The point of using a known function is to avoid any suspicion that the designer built in a clever back door through which only he can enter. This process continues until all the input blocks have been consumed. The contents of the 128-bit buffer form the message digest.

After more than a decade of solid use and study, weaknesses in MD5 have led to the ability to find collisions, or different messages with the same hash (Sotirov, et al., 2008). This is the death knell for a digest function because it means that the digest cannot safely be used to represent a message. Thus, the security community considers MD5 to be broken; it should be replaced where possible and no new systems should use it as part of their design. Nevertheless, you may still see MD5 used in existing systems.

8.4.4 The Birthday Attack

In the world of crypto, nothing is ever what it seems to be. One might think that it would take on the order of 2^m operations to subvert an m-bit message digest. In fact, $2^{m/2}$ operations will often do using the **birthday attack,** an approach published by Yuval (1979) in his now-classic paper "How to Swindle Rabin."

The idea for this attack comes from a technique that math professors often use in their probability courses. The question is: how many students do you need in a class before the probability of having two people with the same birthday exceeds 1/2? Most students expect the answer to be way over 100. In fact, probability theory says it is just 23. Without giving a rigorous analysis, intuitively, with 23

people, we can form $(23 \times 22)/2 = 253$ different pairs, each of which has a probability of 1/365 of being a hit. In this light, it is not really so surprising any more.

More generally, if there is some mapping between inputs and outputs with n inputs (people, messages, etc.) and k possible outputs (birthdays, message digests, etc.), there are $n(n-1)/2$ input pairs. If $n(n-1)/2 > k$, the chance of having at least one match is pretty good. Thus, approximately, a match is likely for $n > \sqrt{k}$. This result means that a 64-bit message digest can probably be broken by generating about 2^{32} messages and looking for two with the same message digest.

Let us look at a practical example. The Department of Computer Science at State University has one position for a tenured faculty member and two candidates, Tom and Dick. Tom was hired two years before Dick, so he goes up for review first. If he gets it, Dick is out of luck. Tom knows that the department chairperson, Marilyn, thinks highly of his work, so he asks her to write him a letter of recommendation to the Dean, who will decide on Tom's case. Once sent, all letters become confidential.

Marilyn tells her secretary, Ellen, to write the Dean a letter, outlining what she wants in it. When it is ready, Marilyn will review it, compute and sign the 64-bit digest, and send it to the Dean. Ellen can send the letter later by email.

Unfortunately for Tom, Ellen is romantically involved with Dick and would like to do Tom in, so she writes the following letter with the 32 bracketed options:

Dear Dean Smith,

This [*letter* | *message*] is to give my [*honest* | *frank*] opinion of Prof. Tom Wilson, who is [*a candidate* | *up*] for tenure [*now* | *this year*]. I have [*known* | *worked with*] Prof. Wilson for [*about* | *almost*] six years. He is an [*outstanding* | *excellent*] researcher of great [*talent* | *ability*] known [*worldwide* | *internationally*] for his [*brilliant* | *creative*] insights into [*many* | *a wide variety of*] [*difficult* | *challenging*] problems.

He is also a [*highly* | *greatly*] [*respected* | *admired*] [*teacher* | *educator*]. His students give his [*classes* | *courses*] [*rave* | *spectacular*] reviews. He is [*our* | *the Department's*] [*most popular* | *best-loved*] [*teacher* | *instructor*].

[*In addition* | *Additionally*] Prof. Wilson is a [*gifted* | *effective*] fund raiser. His [*grants* | *contracts*] have brought a [*large* | *substantial*] amount of money into [*the* | *our*] Department. [*This money has* | *These funds have*] [*enabled* | *permitted*] us to [*pursue* | *carry out*] many [*special* | *important*] programs, [*such as* | *for example*] your State 2000 program. Without these funds we would [*be unable* | *not be able*] to continue this program, which is so [*important* | *essential*] to both of us. I strongly urge you to grant him tenure.

Unfortunately for Tom, as soon as Ellen finishes composing and typing in this letter, she also writes a second one:

Dear Dean Smith,

This [*letter* | *message*] is to give my [*honest* | *frank*] opinion of Prof. Tom Wilson, who is [*a candidate* | *up*] for tenure [*now* | *this year*]. I have [*known* | *worked with*] Tom for [*about* | *almost*] six years. He is a [*poor* | *weak*] researcher not well known in his [*field* | *area*]. His research [*hardly ever* | *rarely*] shows [*insight in* | *understanding of*] the [*key* | *major*] problems of [*the* | *our*] day.

Furthermore, he is not a [*respected* | *admired*] [*teacher* | *educator*]. His students give his [*classes* | *courses*] [*poor* | *bad*] reviews. He is [*our* | *the Department's*] least popular [*teacher* | *instructor*], known [*mostly* | *primarily*] within [*the* | *our*] Department for his [*tendency* | *propensity*] to [*ridicule* | *embarrass*] students [*foolish* | *imprudent*] enough to ask questions in his classes.

[*In addition* | *Additionally*] Tom is a [*poor* | *marginal*] fund raiser. His [*grants* | *contracts*] have brought only a [*meager* | *insignificant*] amount of money into [*the* | *our*] Department. Unless new [*money is* | *funds are*] quickly located, we may have to cancel some essential programs, such as your State 2000 program. Unfortunately, under these [*conditions* | *circumstances*] I cannot in good [*conscience* | *faith*] recommend him to you for [*tenure* | *a permanent position*].

Now Ellen programs her computer to compute the 2^{32} message digests of each letter overnight. Chances are, one digest of the first letter will match one digest of the second. If not, she can add a few more options and try again tonight. Suppose that she finds a match. Call the "good" letter *A* and the "bad" one *B*.

Ellen now emails letter *A* to Marilyn for approval. Letter *B* she keeps secret, showing it to no one. Marilyn, of course, approves it, computes her 64-bit message digest, signs the digest, and emails the signed digest off to Dean Smith. Independently, Ellen emails letter *B* to the Dean (not letter *A*, as she is supposed to).

After getting the letter and signed message digest, the Dean runs the message digest algorithm on letter *B*, sees that it agrees with what Marilyn sent him, and fires Tom. The Dean does not realize that Ellen managed to generate two letters with the same message digest and sent her a different one than the one Marilyn saw and approved. (Optional ending: Ellen tells Dick what she did. Dick is appalled and breaks off the affair. Ellen is furious and confesses to Marilyn. Marilyn calls the Dean. Tom gets tenure after all.) With SHA-1, the birthday attack is difficult because even at the ridiculous speed of 1 trillion digests per second, it would take over 32,000 years to compute all 2^{80} digests of two letters with 80 variants each, and even then a match is not guaranteed. With a cloud of 1,000,000 chips working in parallel, 32,000 years becomes 2 weeks.

8.5 MANAGEMENT OF PUBLIC KEYS

Public-key cryptography makes it possible for people who do not share a common key in advance to nevertheless communicate securely. It also makes signing messages possible without the presence of a trusted third party. Finally,

signed message digests make it possible for the recipient to verify the integrity of received messages easily and securely.

However, there is one problem that we have glossed over a bit too quickly: if Alice and Bob do not know each other, how do they get each other's public keys to start the communication process? The obvious solution—put your public key on your Web site—does not work, for the following reason. Suppose that Alice wants to look up Bob's public key on his Web site. How does she do it? She starts by typing in Bob's URL. Her browser then looks up the DNS address of Bob's home page and sends it a *GET* request, as shown in Fig. 8-23. Unfortunately, Trudy intercepts the request and replies with a fake home page, probably a copy of Bob's home page except for the replacement of Bob's public key with Trudy's public key. When Alice now encrypts her first message with E_T, Trudy decrypts it, reads it, re-encrypts it with Bob's public key, and sends it to Bob, who is none the wiser that Trudy is reading his incoming messages. Worse yet, Trudy could modify the messages before reencrypting them for Bob. Clearly, some mechanism is needed to make sure that public keys can be exchanged securely.

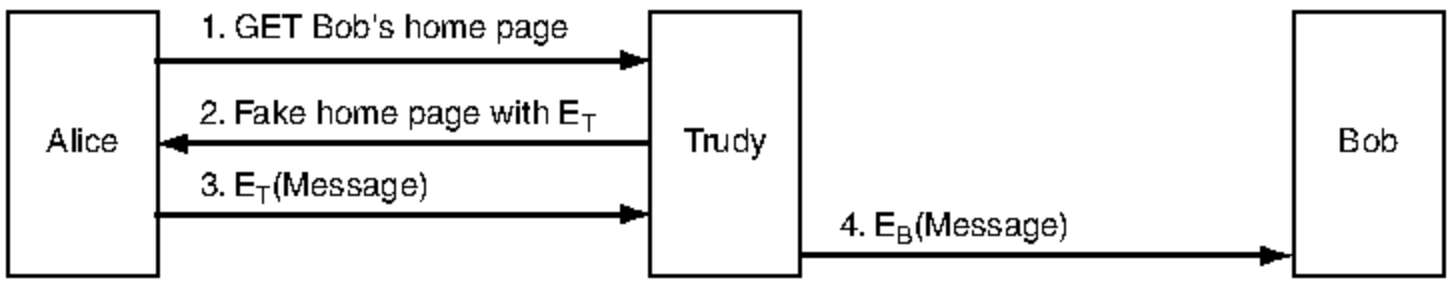


Figure 8-23. A way for Trudy to subvert public-key encryption.

8.5.1 Certificates

As a first attempt at distributing public keys securely, we could imagine a **KDC key distribution center** available online 24 hours a day to provide public keys on demand. One of the many problems with this solution is that it is not scalable, and the key distribution center would rapidly become a bottleneck. Also, if it ever went down, Internet security would suddenly grind to a halt.

For these reasons, people have developed a different solution, one that does not require the key distribution center to be online all the time. In fact, it does not have to be online at all. Instead, what it does is certify the public keys belonging to people, companies, and other organizations. An organization that certifies public keys is now called a **CA** (**Certification Authority**).

As an example, suppose that Bob wants to allow Alice and other people he does not know to communicate with him securely. He can go to the CA with his public key along with his passport or driver's license and ask to be certified. The CA then issues a certificate similar to the one in Fig. 8-24 and signs its SHA-1

hash with the CA's private key. Bob then pays the CA's fee and gets a CD-ROM containing the certificate and its signed hash.

Figure 8-24. A possible certificate and its signed hash.

The fundamental job of a certificate is to bind a public key to the name of a principal (individual, company, etc.). Certificates themselves are not secret or protected. Bob might, for example, decide to put his new certificate on his Web site, with a link on the main page saying: Click here for my public-key certificate. The resulting click would return both the certificate and the signature block (the signed SHA-1 hash of the certificate).

Now let us run through the scenario of Fig. 8-23 again. When Trudy intercepts Alice's request for Bob's home page, what can she do? She can put her own certificate and signature block on the fake page, but when Alice reads the contents of the certificate she will immediately see that she is not talking to Bob because Bob's name is not in it. Trudy can modify Bob's home page on the fly, replacing Bob's public key with her own. However, when Alice runs the SHA-1 algorithm on the certificate, she will get a hash that does not agree with the one she gets when she applies the CA's well-known public key to the signature block. Since Trudy does not have the CA's private key, she has no way of generating a signature block that contains the hash of the modified Web page with her public key on it. In this way, Alice can be sure she has Bob's public key and not Trudy's or someone else's. And as we promised, this scheme does not require the CA to be online for verification, thus eliminating a potential bottleneck.

While the standard function of a certificate is to bind a public key to a principal, a certificate can also be used to bind a public key to an **attribute**. For example, a certificate could say: "This public key belongs to someone over 18." It could be used to prove that the owner of the private key was not a minor and thus allowed to access material not suitable for children, and so on, but without disclosing the owner's identity. Typically, the person holding the certificate would send it to the Web site, principal, or process that cared about age. That site, principal, or process would then generate a random number and encrypt it with the public key in the certificate. If the owner were able to decrypt it and send it back,

that would be proof that the owner indeed had the attribute stated in the certificate. Alternatively, the random number could be used to generate a session key for the ensuing conversation.

Another example of where a certificate might contain an attribute is in an object-oriented distributed system. Each object normally has multiple methods. The owner of the object could provide each customer with a certificate giving a bit map of which methods the customer is allowed to invoke and binding the bit map to a public key using a signed certificate. Again, if the certificate holder can prove possession of the corresponding private key, he will be allowed to perform the methods in the bit map. This approach has the property that the owner's identity need not be known, a property useful in situations where privacy is important.

8.5.2 X.509

If everybody who wanted something signed went to the CA with a different kind of certificate, managing all the different formats would soon become a problem. To solve this problem, a standard for certificates has been devised and approved by ITU. The standard is called **X.509** and is in widespread use on the Internet. It has gone through three versions since the initial standardization in 1988. We will discuss V3.

X.509 has been heavily influenced by the OSI world, borrowing some of its worst features (e.g., naming and encoding). Surprisingly, IETF went along with X.509, even though in nearly every other area, from machine addresses to transport protocols to email formats, IETF generally ignored OSI and tried to do it right. The IETF version of X.509 is described in RFC 5280.

At its core, X.509 is a way to describe certificates. The primary fields in a certificate are listed in Fig. 8-25. The descriptions given there should provide a general idea of what the fields do. For additional information, please consult the standard itself or RFC 2459.

For example, if Bob works in the loan department of the Money Bank, his X.500 address might be

```
/C=US/O=MoneyBank/OU=Loan/CN=Bob/
```

where *C* is for country, *O* is for organization, *OU* is for organizational unit, and *CN* is for common name. CAs and other entities are named in a similar way. A substantial problem with X.500 names is that if Alice is trying to contact *bob@moneybank.com* and is given a certificate with an X.500 name, it may not be obvious to her that the certificate refers to the Bob she wants. Fortunately, starting with version 3, DNS names are now permitted instead of X.500 names, so this problem may eventually vanish.

Certificates are encoded using OSI **ASN.1** (**Abstract Syntax Notation 1**), which is sort of like a struct in C, except with a extremely peculiar and verbose notation. More information about X.509 is given by Ford and Baum (2000).

Field	Meaning
Version	Which version of X.509
Serial number	This number plus the CA's name uniquely identifies the certificate
Signature algorithm	The algorithm used to sign the certificate
Issuer	X.500 name of the CA
Validity period	The starting and ending times of the validity period
Subject name	The entity whose key is being certified
Public key	The subject's public key and the ID of the algorithm using it
Issuer ID	An optional ID uniquely identifying the certificate's issuer
Subject ID	An optional ID uniquely identifying the certificate's subject
Extensions	Many extensions have been defined
Signature	The certificate's signature (signed by the CA's private key)

Figure 8-25. The basic fields of an X.509 certificate.

8.5.3 Public Key Infrastructures

Having a single CA to issue all the world's certificates obviously would not work. It would collapse under the load and be a central point of failure as well. A possible solution might be to have multiple CAs, all run by the same organization and all using the same private key to sign certificates. While this would solve the load and failure problems, it introduces a new problem: key leakage. If there were dozens of servers spread around the world, all holding the CA's private key, the chance of the private key being stolen or otherwise leaking out would be greatly increased. Since the compromise of this key would ruin the world's electronic security infrastructure, having a single central CA is very risky.

In addition, which organization would operate the CA? It is hard to imagine any authority that would be accepted worldwide as legitimate and trustworthy. In some countries, people would insist that it be a government, while in other countries they would insist that it not be a government.

For these reasons, a different way for certifying public keys has evolved. It goes under the general name of **PKI** (**Public Key Infrastructure**). In this section, we will summarize how it works in general, although there have been many proposals, so the details will probably evolve in time.

A PKI has multiple components, including users, CAs, certificates, and directories. What the PKI does is provide a way of structuring these components and define standards for the various documents and protocols. A particularly simple form of PKI is a hierarchy of CAs, as depicted in Fig. 8-26. In this example we have shown three levels, but in practice there might be fewer or more. The top-level CA, the root, certifies second-level CAs, which we here call **RA**s (**Regional**

Authorities) because they might cover some geographic region, such as a country or continent. This term is not standard, though; in fact, no term is really standard for the different levels of the tree. These in turn certify the real CAs, which issue the X.509 certificates to organizations and individuals. When the root authorizes a new RA, it generates an X.509 certificate stating that it has approved the RA, includes the new RA's public key in it, signs it, and hands it to the RA. Similarly, when an RA approves a new CA, it produces and signs a certificate stating its approval and containing the CA's public key.

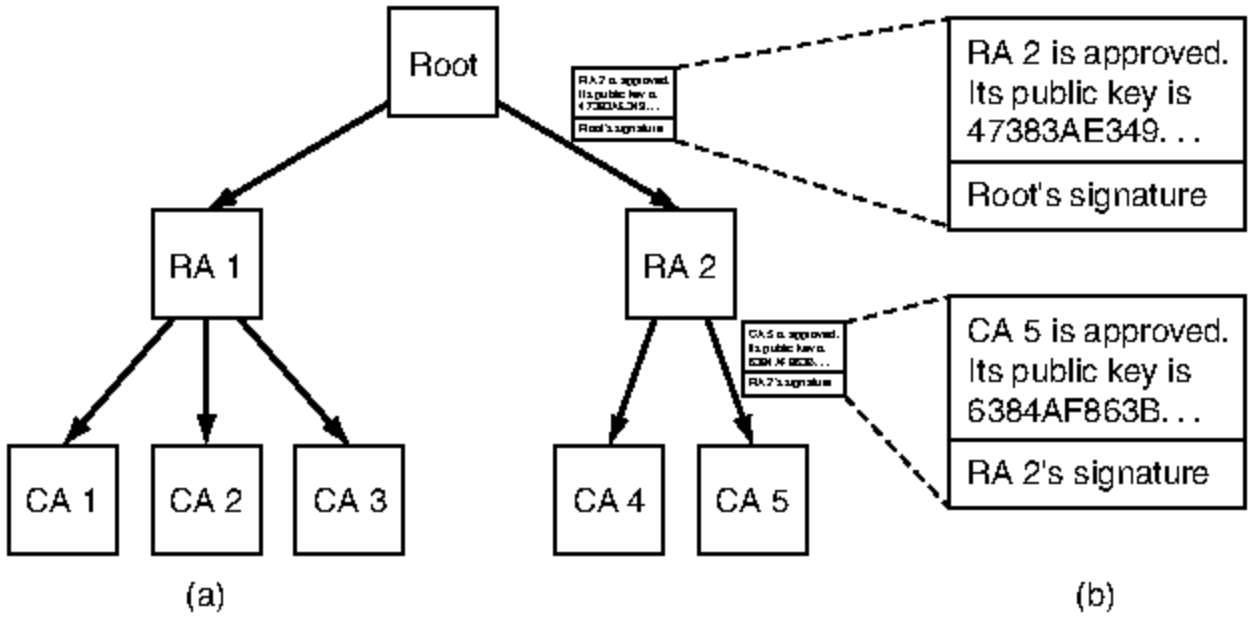


Figure 8-26. (a) A hierarchical PKI. (b) A chain of certificates.

Our PKI works like this. Suppose that Alice needs Bob's public key in order to communicate with him, so she looks for and finds a certificate containing it, signed by CA 5. But Alice has never heard of CA 5. For all she knows, CA 5 might be Bob's 10-year-old daughter. She could go to CA 5 and say: "Prove your legitimacy." CA 5 will respond with the certificate it got from RA 2, which contains CA 5's public key. Now armed with CA 5's public key, she can verify that Bob's certificate was indeed signed by CA 5 and is thus legal.

Unless RA 2 is Bob's 12-year-old son. So, the next step is for her to ask RA 2 to prove it is legitimate. The response to her query is a certificate signed by the root and containing RA 2's public key. Now Alice is sure she has Bob's public key.

But how does Alice find the root's public key? Magic. It is assumed that everyone knows the root's public key. For example, her browser might have been shipped with the root's public key built in.

Bob is a friendly sort of guy and does not want to cause Alice a lot of work. He knows that she is going to have to check out CA 5 and RA 2, so to save her some trouble, he collects the two needed certificates and gives her the two certificates along with his. Now she can use her own knowledge of the root's public key to verify the top-level certificate and the public key contained therein to verify the second one. Alice does not need to contact anyone to do the verification.

Because the certificates are all signed, she can easily detect any attempts to tamper with their contents. A chain of certificates going back to the root like this is sometimes called a **chain of trust** or a **certification path**. The technique is widely used in practice.

Of course, we still have the problem of who is going to run the root. The solution is not to have a single root, but to have many roots, each with its own RAs and CAs. In fact, modern browsers come preloaded with the public keys for over 100 roots, sometimes referred to as **trust anchors**. In this way, having a single worldwide trusted authority can be avoided.

But there is now the issue of how the browser vendor decides which purported trust anchors are reliable and which are sleazy. It all comes down to the user trusting the browser vendor to make wise choices and not simply approve all trust anchors willing to pay its inclusion fee. Most browsers allow users to inspect the root keys (usually in the form of certificates signed by the root) and delete any that seem shady.

Directories

Another issue for any PKI is where certificates (and their chains back to some known trust anchor) are stored. One possibility is to have each user store his or her own certificates. While doing this is safe (i.e., there is no way for users to tamper with signed certificates without detection), it is also inconvenient. One alternative that has been proposed is to use DNS as a certificate directory. Before contacting Bob, Alice probably has to look up his IP address using DNS, so why not have DNS return Bob's entire certificate chain along with his IP address?

Some people think this is the way to go, but others would prefer dedicated directory servers whose only job is managing X.509 certificates. Such directories could provide lookup services by using properties of the X.500 names. For example, in theory such a directory service could answer a query such as: "Give me a list of all people named Alice who work in sales departments anywhere in the U.S. or Canada."

Revocation

The real world is full of certificates, too, such as passports and drivers' licenses. Sometimes these certificates can be revoked, for example, drivers' licenses can be revoked for drunken driving and other driving offenses. The same problem occurs in the digital world: the grantor of a certificate may decide to revoke it because the person or organization holding it has abused it in some way. It can also be revoked if the subject's private key has been exposed or, worse yet, the CA's private key has been compromised. Thus, a PKI needs to deal with the issue of revocation. The possibility of revocation complicates matters.

A first step in this direction is to have each CA periodically issue a **CRL** (**Certificate Revocation List**) giving the serial numbers of all certificates that it has revoked. Since certificates contain expiry times, the CRL need only contain the serial numbers of certificates that have not yet expired. Once its expiry time has passed, a certificate is automatically invalid, so no distinction is needed between those that just timed out and those that were actually revoked. In both cases, they cannot be used any more.

Unfortunately, introducing CRLs means that a user who is about to use a certificate must now acquire the CRL to see if the certificate has been revoked. If it has been, it should not be used. However, even if the certificate is not on the list, it might have been revoked just after the list was published. Thus, the only way to really be sure is to ask the CA. And on the next use of the same certificate, the CA has to be asked again, since the certificate might have been revoked a few seconds ago.

Another complication is that a revoked certificate could conceivably be reinstated, for example, if it was revoked for nonpayment of some fee that has since been paid. Having to deal with revocation (and possibly reinstatement) eliminates one of the best properties of certificates, namely, that they can be used without having to contact a CA.

Where should CRLs be stored? A good place would be the same place the certificates themselves are stored. One strategy is for the CA to actively push out CRLs periodically and have the directories process them by simply removing the revoked certificates. If directories are not used for storing certificates, the CRLs can be cached at various places around the network. Since a CRL is itself a signed document, if it is tampered with, that tampering can be easily detected.

If certificates have long lifetimes, the CRLs will be long, too. For example, if credit cards are valid for 5 years, the number of revocations outstanding will be much longer than if new cards are issued every 3 months. A standard way to deal with long CRLs is to issue a master list infrequently, but issue updates to it more often. Doing this reduces the bandwidth needed for distributing the CRLs.

8.6 COMMUNICATION SECURITY

We have now finished our study of the tools of the trade. Most of the important techniques and protocols have been covered. The rest of the chapter is about how these techniques are applied in practice to provide network security, plus some thoughts about the social aspects of security at the end of the chapter.

In the following four sections, we will look at communication security, that is, how to get the bits secretly and without modification from source to destination and how to keep unwanted bits outside the door. These are by no means the only security issues in networking, but they are certainly among the most important ones, making this a good place to start our study.

8.6.1 IPsec

IETF has known for years that security was lacking in the Internet. Adding it was not easy because a war broke out about where to put it. Most security experts believe that to be really secure, encryption and integrity checks have to be end to end (i.e., in the application layer). That is, the source process encrypts and/or integrity protects the data and sends them to the destination process where they are decrypted and/or verified. Any tampering done in between these two processes, including within either operating system, can then be detected. The trouble with this approach is that it requires changing all the applications to make them security aware. In this view, the next best approach is putting encryption in the transport layer or in a new layer between the application layer and the transport layer, making it still end to end but not requiring applications to be changed.

The opposite view is that users do not understand security and will not be capable of using it correctly and nobody wants to modify existing programs in any way, so the network layer should authenticate and/or encrypt packets without the users being involved. After years of pitched battles, this view won enough support that a network layer security standard was defined. In part, the argument was that having network layer encryption does not prevent security-aware users from doing it right and it does help security-unaware users to some extent.

The result of this war was a design called **IPsec** (**IP security**), which is described in RFCs 2401, 2402, and 2406, among others. Not all users want encryption (because it is computationally expensive). Rather than make it optional, it was decided to require encryption all the time but permit the use of a null algorithm. The null algorithm is described and praised for its simplicity, ease of implementation, and great speed in RFC 2410.

The complete IPsec design is a framework for multiple services, algorithms, and granularities. The reason for multiple services is that not everyone wants to pay the price for having all the services all the time, so the services are available a la carte. The major services are secrecy, data integrity, and protection from replay attacks (where the intruder replays a conversation). All of these are based on symmetric-key cryptography because high performance is crucial.

The reason for having multiple algorithms is that an algorithm that is now thought to be secure may be broken in the future. By making IPsec algorithm-independent, the framework can survive even if some particular algorithm is later broken.

The reason for having multiple granularities is to make it possible to protect a single TCP connection, all traffic between a pair of hosts, or all traffic between a pair of secure routers, among other possibilities.

One slightly surprising aspect of IPsec is that even though it is in the IP layer, it is connection oriented. Actually, that is not so surprising because to have any security, a key must be established and used for some period of time—in essence, a kind of connection by a different name. Also, connections amortize the setup

costs over many packets. A "connection" in the context of IPsec is called an **SA** (**Security Association**). An SA is a simplex connection between two endpoints and has a security identifier associated with it. If secure traffic is needed in both directions, two security associations are required. Security identifiers are carried in packets traveling on these secure connections and are used to look up keys and other relevant information when a secure packet arrives.

Technically, IPsec has two principal parts. The first part describes two new headers that can be added to packets to carry the security identifier, integrity control data, and other information. The other part, **ISAKMP** (**Internet Security Association and Key Management Protocol**), deals with establishing keys. ISAKMP is a framework. The main protocol for carrying out the work is **IKE** (**Internet Key Exchange**). Version 2 of IKE as described in RFC 4306 should be used, as the earlier version was deeply flawed, as pointed out by Perlman and Kaufman (2000).

IPsec can be used in either of two modes. In **transport mode**, the IPsec header is inserted just after the IP header. The *Protocol* field in the IP header is changed to indicate that an IPsec header follows the normal IP header (before the TCP header). The IPsec header contains security information, primarily the SA identifier, a new sequence number, and possibly an integrity check of the payload.

In **tunnel mode**, the entire IP packet, header and all, is encapsulated in the body of a new IP packet with a completely new IP header. Tunnel mode is useful when the tunnel ends at a location other than the final destination. In some cases, the end of the tunnel is a security gateway machine, for example, a company firewall. This is commonly the case for a VPN (Virtual Private Network). In this mode, the security gateway encapsulates and decapsulates packets as they pass through it. By terminating the tunnel at this secure machine, the machines on the company LAN do not have to be aware of IPsec. Only the security gateway has to know about it.

Tunnel mode is also useful when a bundle of TCP connections is aggregated and handled as one encrypted stream because it prevents an intruder from seeing who is sending how many packets to whom. Sometimes just knowing how much traffic is going where is valuable information. For example, if during a military crisis, the amount of traffic flowing between the Pentagon and the White House were to drop sharply, but the amount of traffic between the Pentagon and some military installation deep in the Colorado Rocky Mountains were to increase by the same amount, an intruder might be able to deduce some useful information from these data. Studying the flow patterns of packets, even if they are encrypted, is called **traffic analysis**. Tunnel mode provides a way to foil it to some extent. The disadvantage of tunnel mode is that it adds an extra IP header, thus increasing packet size substantially. In contrast, transport mode does not affect packet size as much.

The first new header is **AH** (**Authentication Header**). It provides integrity checking and antireplay security, but not secrecy (i.e., no data encryption). The

use of AH in transport mode is illustrated in Fig. 8-27. In IPv4, it is interposed between the IP header (including any options) and the TCP header. In IPv6, it is just another extension header and is treated as such. In fact, the format is close to that of a standard IPv6 extension header. The payload may have to be padded out to some particular length for the authentication algorithm, as shown.

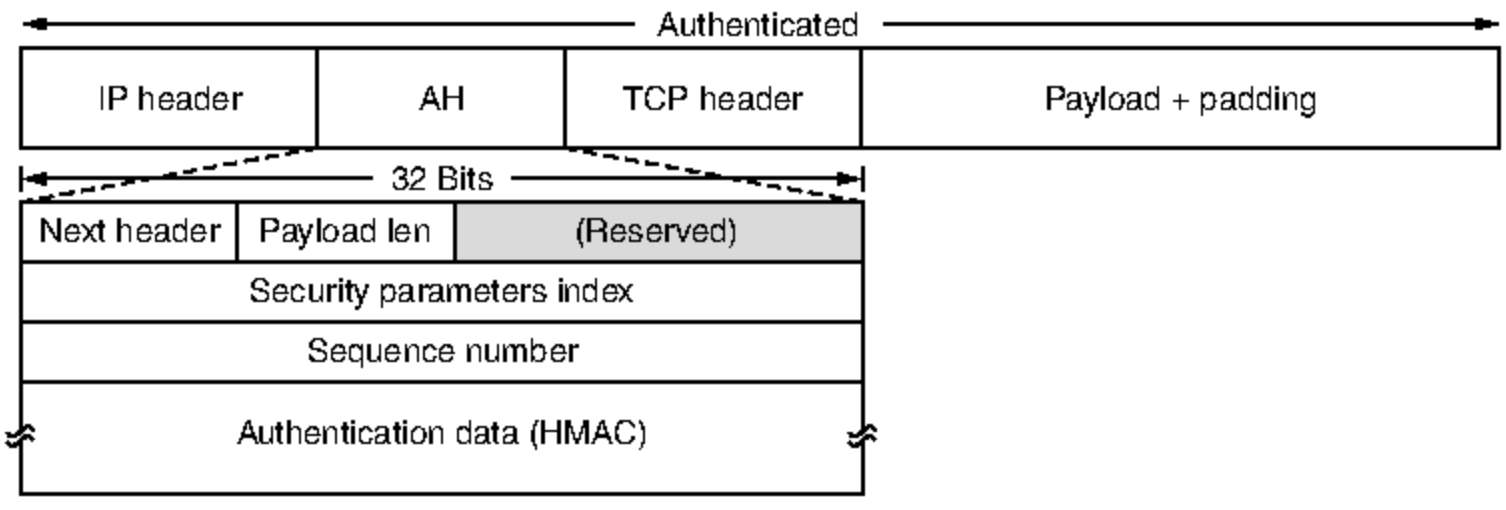


Figure 8-27. The IPsec authentication header in transport mode for IPv4.

Let us now examine the AH header. The *Next header* field is used to store the value that the IP *Protocol* field had before it was replaced with 51 to indicate that an AH header follows. In most cases, the code for TCP (6) will go here. The *Payload length* is the number of 32-bit words in the AH header minus 2.

The *Security parameters index* is the connection identifier. It is inserted by the sender to indicate a particular record in the receiver's database. This record contains the shared key used on this connection and other information about the connection. If this protocol had been invented by ITU rather than IETF, this field would have been called *Virtual circuit number*.

The *Sequence number* field is used to number all the packets sent on an SA. Every packet gets a unique number, even retransmissions. In other words, the retransmission of a packet gets a different number here than the original (even though its TCP sequence number is the same). The purpose of this field is to detect replay attacks. These sequence numbers may not wrap around. If all 2^{32} are exhausted, a new SA must be established to continue communication.

Finally, we come to *Authentication data*, which is a variable-length field that contains the payload's digital signature. When the SA is established, the two sides negotiate which signature algorithm they are going to use. Normally, public-key cryptography is not used here because packets must be processed extremely rapidly and all known public-key algorithms are too slow. Since IPsec is based on symmetric-key cryptography and the sender and receiver negotiate a shared key before setting up an SA, the shared key is used in the signature computation. One simple way is to compute the hash over the packet plus the shared key. The shared key is not transmitted, of course. A scheme like this is called an **HMAC**

(**Hashed Message Authentication Code**). It is much faster to compute than first running SHA-1 and then running RSA on the result.

The AH header does not allow encryption of the data, so it is mostly useful when integrity checking is needed but secrecy is not needed. One noteworthy feature of AH is that the integrity check covers some of the fields in the IP header, namely, those that do not change as the packet moves from router to router. The *Time to live* field changes on each hop, for example, so it cannot be included in the integrity check. However, the IP source address is included in the check, making it impossible for an intruder to falsify the origin of a packet.

The alternative IPsec header is **ESP** (**Encapsulating Security Payload**). Its use for both transport mode and tunnel mode is shown in Fig. 8-28.

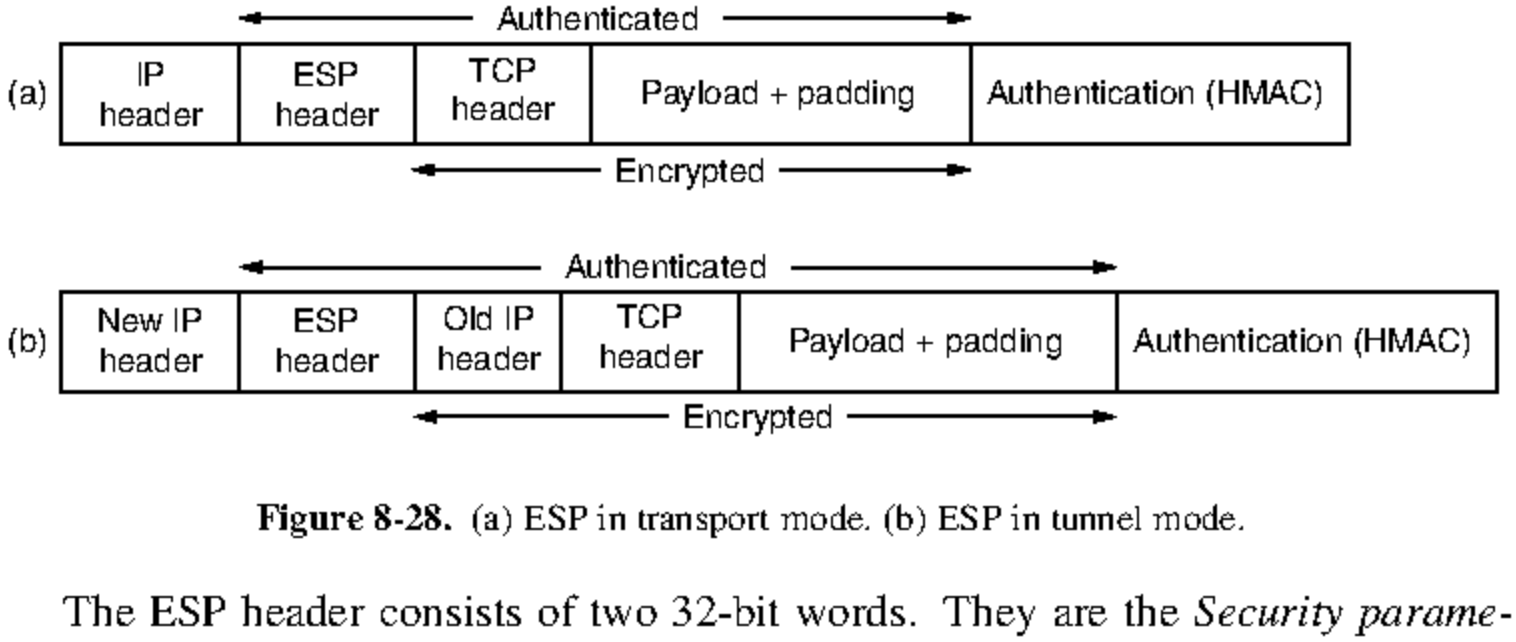


Figure 8-28. (a) ESP in transport mode. (b) ESP in tunnel mode.

The ESP header consists of two 32-bit words. They are the *Security parameters index* and *Sequence number* fields that we saw in AH. A third word that generally follows them (but is technically not part of the header) is the *Initialization vector* used for the data encryption, unless null encryption is used, in which case it is omitted.

ESP also provides for HMAC integrity checks, as does AH, but rather than being included in the header, they come after the payload, as shown in Fig. 8-28. Putting the HMAC at the end has an advantage in a hardware implementation: the HMAC can be calculated as the bits are going out over the network interface and appended to the end. This is why Ethernet and other LANs have their CRCs in a trailer, rather than in a header. With AH, the packet has to be buffered and the signature computed before the packet can be sent, potentially reducing the number of packets/sec that can be sent.

Given that ESP can do everything AH can do and more and is more efficient to boot, the question arises: why bother having AH at all? The answer is mostly historical. Originally, AH handled only integrity and ESP handled only secrecy. Later, integrity was added to ESP, but the people who designed AH did not want to let it die after all that work. Their only real argument is that AH checks part of the IP header, which ESP does not, but other than that it is really a weak argument. Another weak argument is that a product supporting AH but not ESP might

have less trouble getting an export license because it cannot do encryption. AH is likely to be phased out in the future.

8.6.2 Firewalls

The ability to connect any computer, anywhere, to any other computer, anywhere, is a mixed blessing. For individuals at home, wandering around the Internet is lots of fun. For corporate security managers, it is a nightmare. Most companies have large amounts of confidential information online—trade secrets, product development plans, marketing strategies, financial analyses, etc. Disclosure of this information to a competitor could have dire consequences.

In addition to the danger of information leaking out, there is also a danger of information leaking in. In particular, viruses, worms, and other digital pests can breach security, destroy valuable data, and waste large amounts of administrators' time trying to clean up the mess they leave. Often they are imported by careless employees who want to play some nifty new game.

Consequently, mechanisms are needed to keep "good" bits in and "bad" bits out. One method is to use IPsec. This approach protects data in transit between secure sites. However, IPsec does nothing to keep digital pests and intruders from getting onto the company LAN. To see how to accomplish this goal, we need to look at firewalls.

Firewalls are just a modern adaptation of that old medieval security standby: digging a deep moat around your castle. This design forced everyone entering or leaving the castle to pass over a single drawbridge, where they could be inspected by the I/O police. With networks, the same trick is possible: a company can have many LANs connected in arbitrary ways, but all traffic to or from the company is forced through an electronic drawbridge (firewall), as shown in Fig. 8-29. No other route exists.

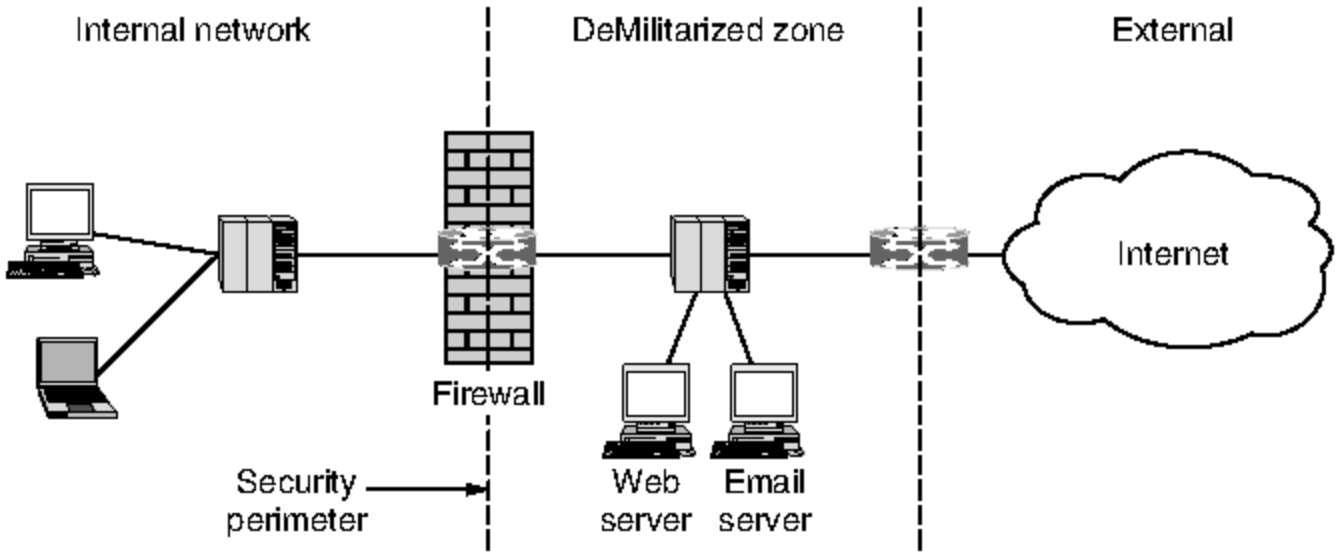


Figure 8-29. A firewall protecting an internal network.

The firewall acts as a **packet filter**. It inspects each and every incoming and outgoing packet. Packets meeting some criterion described in rules formulated by the network administrator are forwarded normally. Those that fail the test are uncermoniously dropped.

The filtering criterion is typically given as rules or tables that list sources and destinations that are acceptable, sources and destinations that are blocked, and default rules about what to do with packets coming from or going to other machines. In the common case of a TCP/IP setting, a source or destination might consist of an IP address and a port. Ports indicate which service is desired. For example, TCP port 25 is for mail, and TCP port 80 is for HTTP. Some ports can simply be blocked. For example, a company could block incoming packets for all IP addresses combined with TCP port 79. It was once popular for the Finger service to look up people's email addresses but is little used today.

Other ports are not so easily blocked. The difficulty is that network administrators want security but cannot cut off communication with the outside world. That arrangement would be much simpler and better for security, but there would be no end to user complaints about it. This is where arrangements such as the **DMZ** (**DeMilitarized Zone**) shown in Fig. 8-29 come in handy. The DMZ is the part of the company network that lies outside of the security perimeter. Anything goes here. By placing a machine such as a Web server in the DMZ, computers on the Internet can contact it to browse the company Web site. Now the firewall can be configured to block incoming TCP traffic to port 80 so that computers on the Internet cannot use this port to attack computers on the internal network. To allow the Web server to be managed, the firewall can have a rule to permit connections between internal machines and the Web server.

Firewalls have become much more sophisticated over time in an arms race with attackers. Originally, firewalls applied a rule set independently for each packet, but it proved difficult to write rules that allowed useful functionality but blocked all unwanted traffic. **Stateful firewalls** map packets to connections and use TCP/IP header fields to keep track of connections. This allows for rules that, for example, allow an external Web server to send packets to an internal host, but only if the internal host first establishes a connection with the external Web server. Such a rule is not possible with stateless designs that must either pass or drop all packets from the external Web server.

Another level of sophistication up from stateful processing is for the firewall to implement **application-level gateways**. This processing involves the firewall looking inside packets, beyond even the TCP header, to see what the application is doing. With this capability, it is possible to distinguish HTTP traffic used for Web browsing from HTTP traffic used for peer-to-peer file sharing. Administrators can write rules to spare the company from peer-to-peer file sharing but allow Web browsing that is vital for business. For all of these methods, outgoing traffic can be inspected as well as incoming traffic, for example, to prevent sensitive documents from being emailed outside of the company.

As the above discussion should make clear, firewalls violate the standard layering of protocols. They are network layer devices, but they peek at the transport and applications layers to do their filtering. This makes them fragile. For instance, firewalls tend to rely on standard port numbering conventions to determine what kind of traffic is carried in a packet. Standard ports are often used, but not by all computers, and not by all applications either. Some peer-to-peer applications select ports dynamically to avoid being easily spotted (and blocked). Encryption with IPSEC or other schemes hides higher-layer information from the firewall. Finally, a firewall cannot readily talk to the computers that communicate through it to tell them what policies are being applied and why their connection is being dropped. It must simply pretend to be a broken wire. For all these reasons, networking purists consider firewalls to be a blemish on the architecture of the Internet. However, the Internet can be a dangerous place if you are a computer. Firewalls help with that problem, so they are likely to stay.

Even if the firewall is perfectly configured, plenty of security problems still exist. For example, if a firewall is configured to allow in packets from only specific networks (e.g., the company's other plants), an intruder outside the firewall can put in false source addresses to bypass this check. If an insider wants to ship out secret documents, he can encrypt them or even photograph them and ship the photos as JPEG files, which bypasses any email filters. And we have not even discussed the fact that, although three-quarters of all attacks come from outside the firewall, the attacks that come from inside the firewall, for example, from disgruntled employees, are typically the most damaging (Verizon, 2009).

A different problem with firewalls is that they provide a single perimeter of defense. If that defense is breached, all bets are off. For this reason, firewalls are often used in a layered defense. For example, a firewall may guard the entrance to the internal network and each computer may also run its own firewall. Readers who think that one security checkpoint is enough clearly have not made an international flight on a scheduled airline recently.

In addition, there is a whole other class of attacks that firewalls cannot deal with. The basic idea of a firewall is to prevent intruders from getting in and secret data from getting out. Unfortunately, there are people who have nothing better to do than try to bring certain sites down. They do this by sending legitimate packets at the target in great numbers until it collapses under the load. For example, to cripple a Web site, an intruder can send a TCP *SYN* packet to establish a connection. The site will then allocate a table slot for the connection and send a *SYN + ACK* packet in reply. If the intruder does not respond, the table slot will be tied up for a few seconds until it times out. If the intruder sends thousands of connection requests, all the table slots will fill up and no legitimate connections will be able to get through. Attacks in which the intruder's goal is to shut down the target rather than steal data are called **DoS** (**Denial of Service**) attacks. Usually, the request packets have false source addresses so the intruder cannot be traced easily. DoS attacks against major Web sites are common on the Internet.

An even worse variant is one in which the intruder has already broken into hundreds of computers elsewhere in the world, and then commands all of them to attack the same target at the same time. Not only does this approach increase the intruder's firepower, but it also reduces his chances of detection since the packets are coming from a large number of machines belonging to unsuspecting users. Such an attack is called a **DDoS** (**Distributed Denial of Service**) attack. This attack is difficult to defend against. Even if the attacked machine can quickly recognize a bogus request, it does take some time to process and discard the request, and if enough requests per second arrive, the CPU will spend all its time dealing with them.

8.6.3 Virtual Private Networks

Many companies have offices and plants scattered over many cities, sometimes over multiple countries. In the olden days, before public data networks, it was common for such companies to lease lines from the telephone company between some or all pairs of locations. Some companies still do this. A network built up from company computers and leased telephone lines is called a **private network**.

Private networks work fine and are very secure. If the only lines available are the leased lines, no traffic can leak out of company locations and intruders have to physically wiretap the lines to break in, which is not easy to do. The problem with private networks is that leasing a dedicated T1 line between two points costs thousands of dollars a month, and T3 lines are many times more expensive. When public data networks and later the Internet appeared, many companies wanted to move their data (and possibly voice) traffic to the public network, but without giving up the security of the private network.

This demand soon led to the invention of **VPN**s (**Virtual Private Networks**), which are overlay networks on top of public networks but with most of the properties of private networks. They are called "virtual" because they are merely an illusion, just as virtual circuits are not real circuits and virtual memory is not real memory.

One popular approach is to build VPNs directly over the Internet. A common design is to equip each office with a firewall and create tunnels through the Internet between all pairs of offices, as illustrated in Fig. 8-30(a). A further advantage of using the Internet for connectivity is that the tunnels can be set up on demand to include, for example, the computer of an employee who is at home or traveling as long as the person has an Internet connection. This flexibility is much greater then is provided with leased lines, yet from the perspective of the computers on the VPN, the topology looks just like the private network case, as shown in Fig. 8-30(b). When the system is brought up, each pair of firewalls has to negotiate the parameters of its SA, including the services, modes, algorithms, and keys. If IPsec is used for the tunneling, it is possible to aggregate all traffic between any

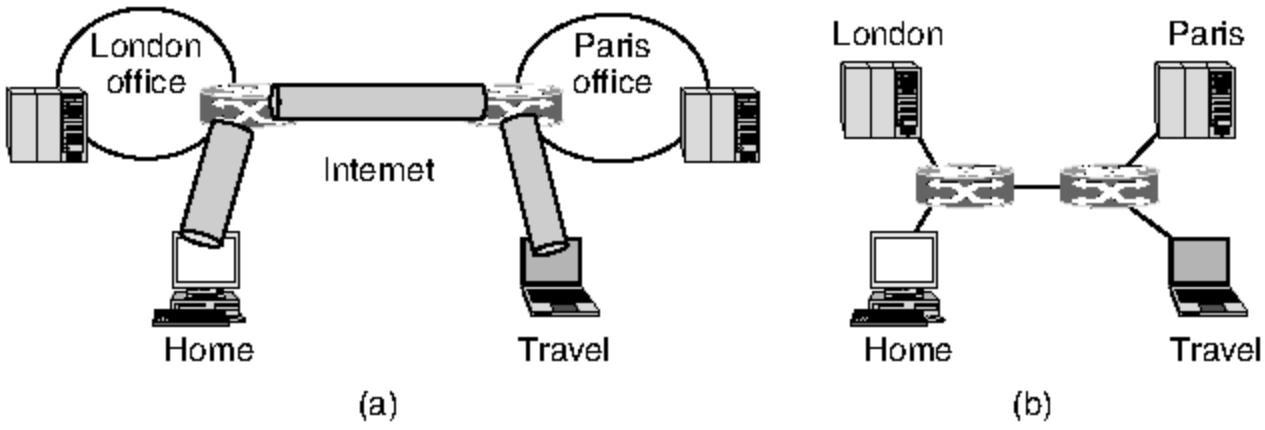


Figure 8-30. (a) A virtual private network. (b) Topology as seen from the inside.

two pairs of offices onto a single authenticated, encrypted SA, thus providing integrity control, secrecy, and even considerable immunity to traffic analysis. Many firewalls have VPN capabilities built in. Some ordinary routers can do this as well, but since firewalls are primarily in the security business, it is natural to have the tunnels begin and end at the firewalls, providing a clear separation between the company and the Internet. Thus, firewalls, VPNs, and IPsec with ESP in tunnel mode are a natural combination and widely used in practice.

Once the SAs have been established, traffic can begin flowing. To a router within the Internet, a packet traveling along a VPN tunnel is just an ordinary packet. The only thing unusual about it is the presence of the IPsec header after the IP header, but since these extra headers have no effect on the forwarding process, the routers do not care about this extra header.

Another approach that is gaining popularity is to have the ISP set up the VPN. Using MPLS (as discussed in Chap. 5), paths for the VPN traffic can be set up across the ISP network between the company offices. These paths keep the VPN traffic separate from other Internet traffic and can be guaranteed a certain amount of bandwidth or other quality of service.

A key advantage of a VPN is that it is completely transparent to all user software. The firewalls set up and manage the SAs. The only person who is even aware of this setup is the system administrator who has to configure and manage the security gateways, or the ISP administrator who has to configure the MPLS paths. To everyone else, it is like having a leased-line private network again. For more about VPNs, see Lewis (2006).

8.6.4 Wireless Security

It is surprisingly easy to design a system using VPNs and firewalls that is logically completely secure but that, in practice, leaks like a sieve. This situation can occur if some of the machines are wireless and use radio communication, which passes right over the firewall in both directions. The range of 802.11 networks is

often a few hundred meters, so anyone who wants to spy on a company can simply drive into the employee parking lot in the morning, leave an 802.11-enabled notebook computer in the car to record everything it hears, and take off for the day. By late afternoon, the hard disk will be full of valuable goodies. Theoretically, this leakage is not supposed to happen. Theoretically, people are not supposed to rob banks, either.

Much of the security problem can be traced to the manufacturers of wireless base stations (access points) trying to make their products user friendly. Usually, if the user takes the device out of the box and plugs it into the electrical power socket, it begins operating immediately—nearly always with no security at all, blurting secrets to everyone within radio range. If it is then plugged into an Ethernet, all the Ethernet traffic suddenly appears in the parking lot as well. Wireless is a snooper's dream come true: free data without having to do any work. It therefore goes without saying that security is even more important for wireless systems than for wired ones. In this section, we will look at some ways wireless networks handle security. Some additional information is given by Nichols and Lekkas (2002).

802.11 Security

Part of the 802.11 standard, originally called **802.11i**, prescribes a data link-level security protocol for preventing a wireless node from reading or interfering with messages sent between another pair of wireless nodes. It also goes by the trade name **WPA2** (**WiFi Protected Access 2**). Plain WPA is an interim scheme that implements a subset of 802.11i. It should be avoided in favor of WPA2.

We will describe 802.11i shortly, but will first note that it is a replacement for **WEP** (**Wired Equivalent Privacy**), the first generation of 802.11 security protocols. WEP was designed by a networking standards committee, which is a completely different process than, for example, the way NIST selected the design of AES. The results were devastating. What was wrong with it? Pretty much everything from a security perspective as it turns out. For example, WEP encrypted data for confidentiality by XORing it with the output of a stream cipher. Unfortunately, weak keying arrangements meant that the output was often reused. This led to trivial ways to defeat it. As another example, the integrity check was based on a 32-bit CRC. That is an efficient code for detecting transmission errors, but it is not a cryptographically strong mechanism for defeating attackers.

These and other design flaws made WEP very easy to compromise. The first practical demonstration that WEP was broken came when Adam Stubblefield was an intern at AT&T (Stubblefield et al., 2002). He was able to code up and test an attack outlined by Fluhrer et al. (2001) in one week, of which most of the time was spent convincing management to buy him a WiFi card to use in his experiments. Software to crack WEP passwords within a minute is now freely available and the use of WEP is very strongly discouraged. While it does prevent casual

access it does not provide any real form of security. The 802.11i group was put together in a hurry when it was clear that WEP was seriously broken. It produced a formal standard by June 2004.

Now we will describe 802.11i, which does provide real security if it is set up and used properly. There are two common scenarios in which WPA2 is used. The first is a corporate setting, in which a company has a separate authentication server that has a username and password database that can be used to determine if a wireless client is allowed to access the network. In this setting, clients use standard protocols to authenticate themselves to the network. The main standards are **802.1X**, with which the access point lets the client carry on a dialogue with the authentication server and observes the result, and **EAP** (**Extensible Authentication Protocol**) (RFC 3748), which tells how the client and the authentication server interact. Actually, EAP is a framework and other standards define the protocol messages. However, we will not delve into the many details of this exchange because they do not much matter for an overview.

The second scenario is in a home setting in which there is no authentication server. Instead, there is a single shared password that is used by clients to access the wireless network. This setup is less complex than having an authentication server, which is why it is used at home and in small businesses, but it is less secure as well. The main difference is that with an authentication server each client gets a key for encrypting traffic that is not known by the other clients. With a single shared password, different keys are derived for each client, but all clients have the same password and can derive each others' keys if they want to.

The keys that are used to encrypt traffic are computed as part of an authentication handshake. The handshake happens right after the client associates with a wireless network and authenticates with an authentication server, if there is one. At the start of the handshake, the client has either the shared network password or its password for the authentication server. This password is used to derive a master key. However, the master key is not used directly to encrypt packets. It is standard cryptographic practice to derive a session key for each period of usage, to change the key for different sessions, and to expose the master key to observation as little as possible. It is this session key that is computed in the handshake.

The session key is computed with the four-packet handshake shown in Fig. 8-31. First, the AP (access point) sends a random number for identification. Random numbers used just once in security protocols like this one are called **nonces**, which is more-or-less a contraction of "number used once." The client also picks its own nonce. It uses the nonces, its MAC address and that of the AP, and the master key to compute a session key, K_S. The session key is split into portions, each of which is used for different purposes, but we have omitted this detail. Now the client has session keys, but the AP does not. So the client sends its nonce to the AP, and the AP performs the same computation to derive the same session keys. The nonces can be sent in the clear because the keys cannot be derived from them without extra, secret information. The message from the client is protected

with an integrity check called a **MIC** (**Message Integrity Check**) based on the session key. The AP can check that the MIC is correct, and so the message indeed must have come from the client, after it computes the session keys. A MIC is just another name for a message authentication code, as in an HMAC. The term MIC is often used instead for networking protocols because of the potential for confusion with MAC (Medium Access Control) addresses.

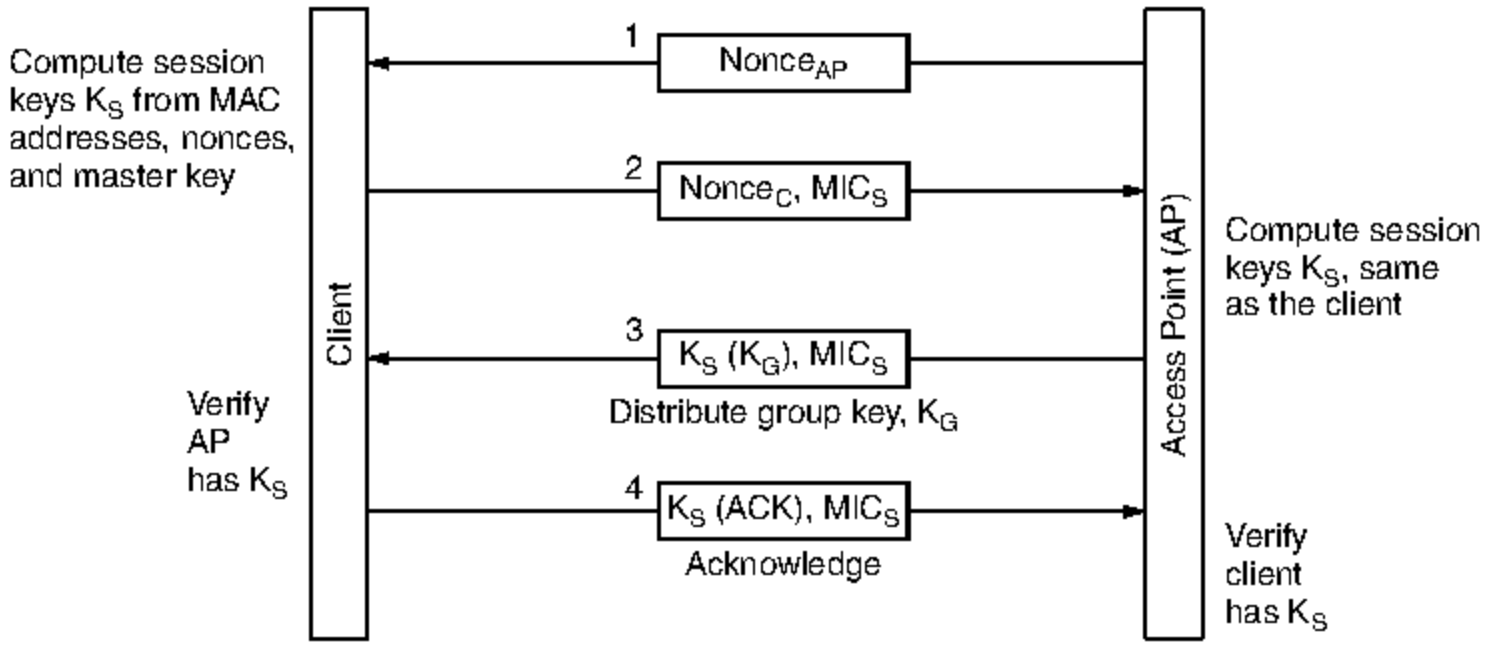


Figure 8-31. The 802.11i key setup handshake.

In the last two messages, the AP distributes a group key, K_G, to the client, and the client acknowledges the message. Receipt of these messages lets the client verify that the AP has the correct session keys, and vice versa. The group key is used for broadcast and multicast traffic on the 802.11 LAN. Because the result of the handshake is that every client has its own encryption keys, none of these keys can be used by the AP to broadcast packets to all of the wireless clients; a separate copy would need to be sent to each client using its key. Instead, a shared key is distributed so that broadcast traffic can be sent only once and received by all the clients. It must be updated as clients leave and join the network.

Finally, we get to the part where the keys are actually used to provide security. Two protocols can be used in 802.11i to provide message confidentiality, integrity, and authentication. Like WPA, one of the protocols, called **TKIP** (**Temporary Key Integrity Protocol**), was an interim solution. It was designed to improve security on old and slow 802.11 cards, so that at least some security that is better than WEP can be rolled out as a firmware upgrade. However, it, too, has now been broken so you are better off with the other, recommended protocol, **CCMP**. What does CCMP stand for? It is short for the somewhat spectacular name Counter mode with Cipher block chaining Message authentication code Protocol. We will just call it CCMP. You can call it anything you want.

CCMP works in a fairly straightforward way. It uses AES encryption with a 128-bit key and block size. The key comes from the session key. To provide confidentiality, messages are encrypted with AES in counter mode. Recall that we discussed cipher modes in Sec. 8.2.3. These modes are what prevent the same message from being encrypted to the same set of bits each time. Counter mode mixes a counter into the encryption. To provide integrity, the message, including header fields, is encrypted with cipher block chaining mode and the last 128-bit block is kept as the MIC. Then both the message (encrypted with counter mode) and the MIC are sent. The client and the AP can each perform this encryption, or verify this encryption when a wireless packet is received. For broadcast or multicast messages, the same procedure is used with the group key.

Bluetooth Security

Bluetooth has a considerably shorter range than 802.11, so it cannot easily be attacked from the parking lot, but security is still an issue here. For example, imagine that Alice's computer is equipped with a wireless Bluetooth keyboard. In the absence of security, if Trudy happened to be in the adjacent office, she could read everything Alice typed in, including all her outgoing email. She could also capture everything Alice's computer sent to the Bluetooth printer sitting next to it (e.g., incoming email and confidential reports). Fortunately, Bluetooth has an elaborate security scheme to try to foil the world's Trudies. We will now summarize the main features of it.

Bluetooth version 2.1 and later has four security modes, ranging from nothing at all to full data encryption and integrity control. As with 802.11, if security is disabled (the default for older devices), there is no security. Most users have security turned off until a serious breach has occurred; then they turn it on. In the agricultural world, this approach is known as locking the barn door after the horse has escaped.

Bluetooth provides security in multiple layers. In the physical layer, frequency hopping provides a tiny little bit of security, but since any Bluetooth device that moves into a piconet has to be told the frequency hopping sequence, this sequence is obviously not a secret. The real security starts when the newly arrived slave asks for a channel with the master. Before Bluetooth 2.1, two devices were assumed to share a secret key set up in advance. In some cases, both are hardwired by the manufacturer (e.g., for a headset and mobile phone sold as a unit). In other cases, one device (e.g., the headset) has a hardwired key and the user has to enter that key into the other device (e.g., the mobile phone) as a decimal number. These shared keys are called **passkeys**. Unfortunately, the passkeys are often hardcoded to "1234" or another predictable value, and in any case are four decimal digits, allowing only 10^4 choices. With simple secure pairing in Bluetooth 2.1, devices pick a code from a six-digit range, which makes the passkey much less predictable but still far from secure.

To establish a channel, the slave and master each check to see if the other one knows the passkey. If so, they negotiate whether that channel will be encrypted, integrity controlled, or both. Then they select a random 128-bit session key, some of whose bits may be public. The point of allowing this key weakening is to comply with government restrictions in various countries designed to prevent the export or use of keys longer than the government can break.

Encryption uses a stream cipher called $\mathbf{E_0}$; integrity control uses **SAFER+**. Both are traditional symmetric-key block ciphers. SAFER+ was submitted to the AES bake-off but was eliminated in the first round because it was slower than the other candidates. Bluetooth was finalized before the AES cipher was chosen; otherwise, it would most likely have used Rijndael.

The actual encryption using the stream cipher is shown in Fig. 8-14, with the plaintext XORed with the keystream to generate the ciphertext. Unfortunately, E_0 itself (like RC4) may have fatal weaknesses (Jakobsson and Wetzel, 2001). While it was not broken at the time of this writing, its similarities to the A5/1 cipher, whose spectacular failure compromises all GSM telephone traffic, are cause for concern (Biryukov et al., 2000). It sometimes amazes people (including the authors of this book), that in the perennial cat-and-mouse game between the cryptographers and the cryptanalysts, the cryptanalysts are so often on the winning side.

Another security issue is that Bluetooth authenticates only devices, not users, so theft of a Bluetooth device may give the thief access to the user's financial and other accounts. However, Bluetooth also implements security in the upper layers, so even in the event of a breach of link-level security, some security may remain, especially for applications that require a PIN code to be entered manually from some kind of keyboard to complete the transaction.

8.7 AUTHENTICATION PROTOCOLS

Authentication is the technique by which a process verifies that its communication partner is who it is supposed to be and not an imposter. Verifying the identity of a remote process in the face of a malicious, active intruder is surprisingly difficult and requires complex protocols based on cryptography. In this section, we will study some of the many authentication protocols that are used on insecure computer networks.

As an aside, some people confuse authorization with authentication. Authentication deals with the question of whether you are actually communicating with a specific process. Authorization is concerned with what that process is permitted to do. For example, say a client process contacts a file server and says: "I am Scott's process and I want to delete the file *cookbook.old*." From the file server's point of view, two questions must be answered:

1. Is this actually Scott's process (authentication)?
2. Is Scott allowed to delete *cookbook.old* (authorization)?

Only after both of these questions have been unambiguously answered in the affirmative can the requested action take place. The former question is really the key one. Once the file server knows to whom it is talking, checking authorization is just a matter of looking up entries in local tables or databases. For this reason, we will concentrate on authentication in this section.

The general model that essentially all authentication protocols use is this. Alice starts out by sending a message either to Bob or to a trusted **KDC** (**Key Distribution Center**), which is expected to be honest. Several other message exchanges follow in various directions. As these messages are being sent, Trudy may intercept, modify, or replay them in order to trick Alice and Bob or just to gum up the works.

Nevertheless, when the protocol has been completed, Alice is sure she is talking to Bob and Bob is sure he is talking to Alice. Furthermore, in most of the protocols, the two of them will also have established a secret **session key** for use in the upcoming conversation. In practice, for performance reasons, all data traffic is encrypted using symmetric-key cryptography (typically AES or triple DES), although public-key cryptography is widely used for the authentication protocols themselves and for establishing the session key.

The point of using a new, randomly chosen session key for each new connection is to minimize the amount of traffic that gets sent with the users' secret keys or public keys, to reduce the amount of ciphertext an intruder can obtain, and to minimize the damage done if a process crashes and its core dump falls into the wrong hands. Hopefully, the only key present then will be the session key. All the permanent keys should have been carefully zeroed out after the session was established.

8.7.1 Authentication Based on a Shared Secret Key

For our first authentication protocol, we will assume that Alice and Bob already share a secret key, K_{AB}. This shared key might have been agreed upon on the telephone or in person, but, in any event, not on the (insecure) network.

This protocol is based on a principle found in many authentication protocols: one party sends a random number to the other, who then transforms it in a special way and returns the result. Such protocols are called **challenge-response** protocols. In this and subsequent authentication protocols, the following notation will be used:

A, B are the identities of Alice and Bob.
R_i's are the challenges, where i identifies the challenger.
K_i's are keys, where i indicates the owner.
K_S is the session key.

The message sequence for our first shared-key authentication protocol is illustrated in Fig. 8-32. In message 1, Alice sends her identity, A, to Bob in a way that Bob understands. Bob, of course, has no way of knowing whether this message came from Alice or from Trudy, so he chooses a challenge, a large random number, R_B, and sends it back to "Alice" as message 2, in plaintext. Alice then encrypts the message with the key she shares with Bob and sends the ciphertext, $K_{AB}(R_B)$, back in message 3. When Bob sees this message, he immediately knows that it came from Alice because Trudy does not know K_{AB} and thus could not have generated it. Furthermore, since R_B was chosen randomly from a large space (say, 128-bit random numbers), it is very unlikely that Trudy would have seen R_B and its response in an earlier session. It is equally unlikely that she could guess the correct response to any challenge.

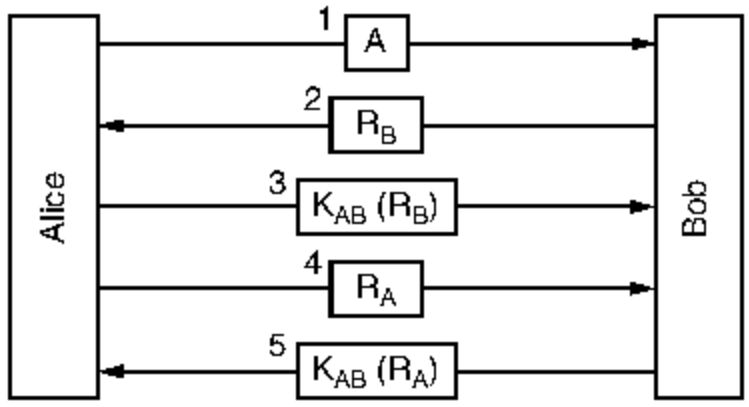


Figure 8-32. Two-way authentication using a challenge-response protocol.

At this point, Bob is sure he is talking to Alice, but Alice is not sure of anything. For all Alice knows, Trudy might have intercepted message 1 and sent back R_B in response. Maybe Bob died last night. To find out to whom she is talking, Alice picks a random number, R_A, and sends it to Bob as plaintext, in message 4. When Bob responds with $K_{AB}(R_A)$, Alice knows she is talking to Bob. If they wish to establish a session key now, Alice can pick one, K_S, and send it to Bob encrypted with K_{AB}.

The protocol of Fig. 8-32 contains five messages. Let us see if we can be clever and eliminate some of them. One approach is illustrated in Fig. 8-33. Here Alice initiates the challenge-response protocol instead of waiting for Bob to do it. Similarly, while he is responding to Alice's challenge, Bob sends his own. The entire protocol can be reduced to three messages instead of five.

Is this new protocol an improvement over the original one? In one sense it is: it is shorter. Unfortunately, it is also wrong. Under certain circumstances, Trudy can defeat this protocol by using what is known as a **reflection attack**. In particular, Trudy can break it if it is possible to open multiple sessions with Bob at once. This situation would be true, for example, if Bob is a bank and is prepared to accept many simultaneous connections from teller machines at once.

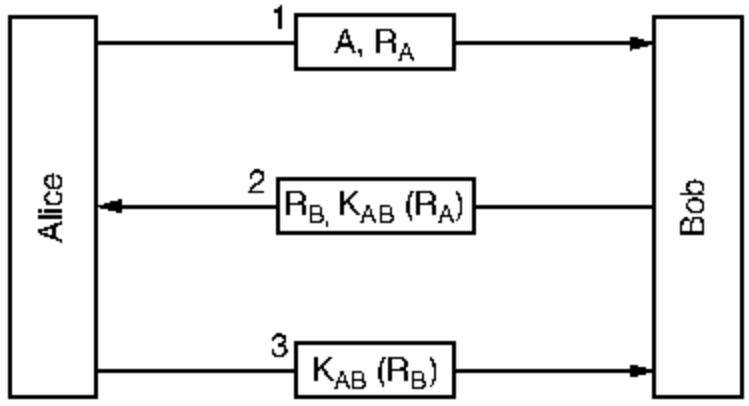


Figure 8-33. A shortened two-way authentication protocol.

Trudy's reflection attack is shown in Fig. 8-34. It starts out with Trudy claiming she is Alice and sending R_T. Bob responds, as usual, with his own challenge, R_B. Now Trudy is stuck. What can she do? She does not know $K_{AB}(R_B)$.

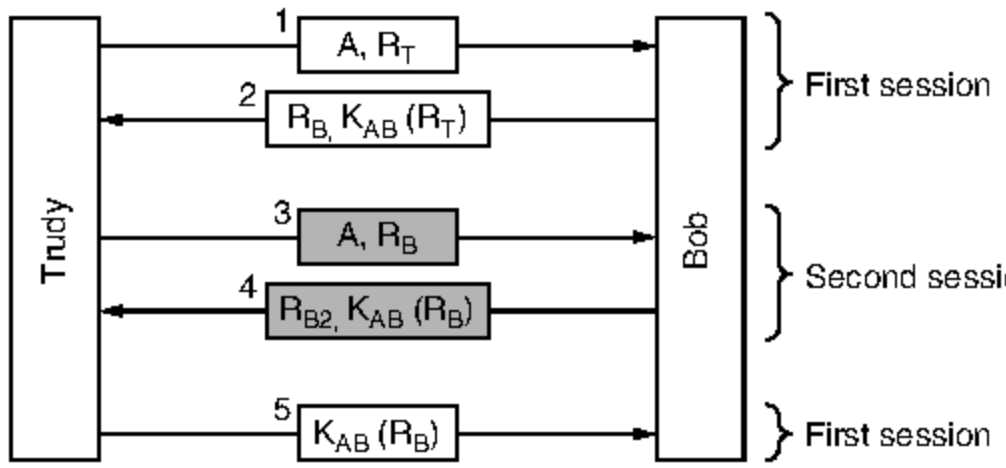


Figure 8-34. The reflection attack.

She can open a second session with message 3, supplying the R_B taken from message 2 as her challenge. Bob calmly encrypts it and sends back $K_{AB}(R_B)$ in message 4. We have shaded the messages on the second session to make them stand out. Now Trudy has the missing information, so she can complete the first session and abort the second one. Bob is now convinced that Trudy is Alice, so when she asks for her bank account balance, he gives it to her without question. Then when she asks him to transfer it all to a secret bank account in Switzerland, he does so without a moment's hesitation.

The moral of this story is:

Designing a correct authentication protocol is much harder than it looks.

The following four general rules often help the designer avoid common pitfalls:

1. Have the initiator prove who she is before the responder has to. This avoids Bob giving away valuable information before Trudy has to give any evidence of who she is.
2. Have the initiator and responder use different keys for proof, even if this means having two shared keys, K_{AB} and K'_{AB}.
3. Have the initiator and responder draw their challenges from different sets. For example, the initiator must use even numbers and the responder must use odd numbers.
4. Make the protocol resistant to attacks involving a second parallel session in which information obtained in one session is used in a different one.

If even one of these rules is violated, the protocol can frequently be broken. Here, all four rules were violated, with disastrous consequences.

Now let us go take a closer look at Fig. 8-32. Surely that protocol is not subject to a reflection attack? Maybe. It is quite subtle. Trudy was able to defeat our protocol by using a reflection attack because it was possible to open a second session with Bob and trick him into answering his own questions. What would happen if Alice were a general-purpose computer that also accepted multiple sessions, rather than a person at a computer? Let us take a look what Trudy can do.

To see how Trudy's attack works, see Fig. 8-35. Alice starts out by announcing her identity in message 1. Trudy intercepts this message and begins her own session with message 2, claiming to be Bob. Again we have shaded the session 2 messages. Alice responds to message 2 by saying in message 3: "You claim to be Bob? Prove it." At this point, Trudy is stuck because she cannot prove she is Bob.

What does Trudy do now? She goes back to the first session, where it is her turn to send a challenge, and sends the R_A she got in message 3. Alice kindly responds to it in message 5, thus supplying Trudy with the information she needs to send in message 6 in session 2. At this point, Trudy is basically home free because she has successfully responded to Alice's challenge in session 2. She can now cancel session 1, send over any old number for the rest of session 2, and she will have an authenticated session with Alice in session 2.

But Trudy is nasty, and she really wants to rub it in. Instead, of sending any old number over to complete session 2, she waits until Alice sends message 7, Alice's challenge for session 1. Of course, Trudy does not know how to respond, so she uses the reflection attack again, sending back R_{A2} as message 8. Alice conveniently encrypts R_{A2} in message 9. Trudy now switches back to session 1 and sends Alice the number she wants in message 10, conveniently copied from what Alice sent in message 9. At this point Trudy has two fully authenticated sessions with Alice.

This attack has a somewhat different result than the attack on the three-message protocol that we saw in Fig. 8-34. This time, Trudy has two authenticated

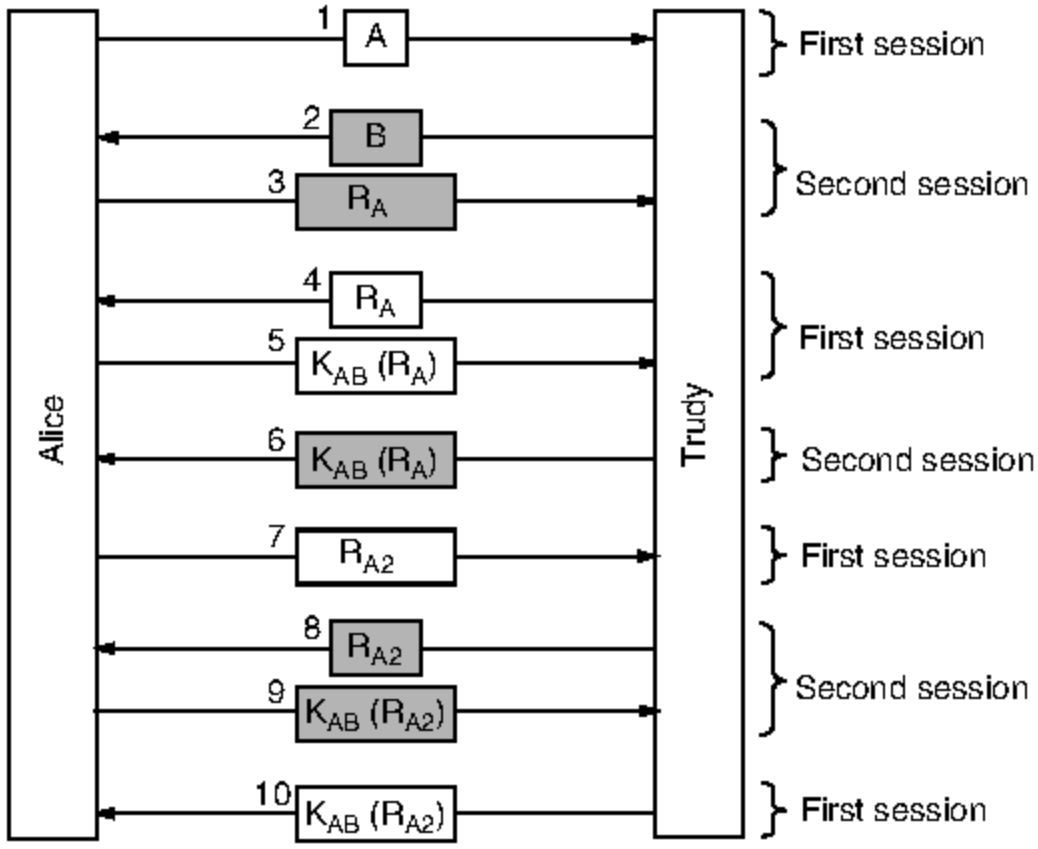


Figure 8-35. A reflection attack on the protocol of Fig. 8-32.

connections with Alice. In the previous example, she had one authenticated connection with Bob. Again here, if we had applied all the general authentication protocol rules discussed earlier, this attack could have been stopped. For a detailed discussion of these kinds of attacks and how to thwart them, see Bird et al. (1993). They also show how it is possible to systematically construct protocols that are provably correct. The simplest such protocol is nevertheless a bit complicated, so we will now show a different class of protocol that also works.

The new authentication protocol is shown in Fig. 8-36 (Bird et al., 1993). It uses an HMAC of the type we saw when studying IPsec. Alice starts out by sending Bob a nonce, R_A, as message 1. Bob responds by selecting his own nonce, R_B, and sending it back along with an HMAC. The HMAC is formed by building a data structure consisting of Alice's nonce, Bob's nonce, their identities, and the shared secret key, K_{AB}. This data structure is then hashed into the HMAC, for example, using SHA-1. When Alice receives message 2, she now has R_A (which she picked herself), R_B, which arrives as plaintext, the two identities, and the secret key, K_{AB}, which she has known all along, so she can compute the HMAC herself. If it agrees with the HMAC in the message, she knows she is talking to Bob because Trudy does not know K_{AB} and thus cannot figure out which HMAC to send. Alice responds to Bob with an HMAC containing just the two nonces.

Can Trudy somehow subvert this protocol? No, because she cannot force either party to encrypt or hash a value of her choice, as happened in Fig. 8-34 and Fig. 8-35. Both HMACs include values chosen by the sending party, something that Trudy cannot control.

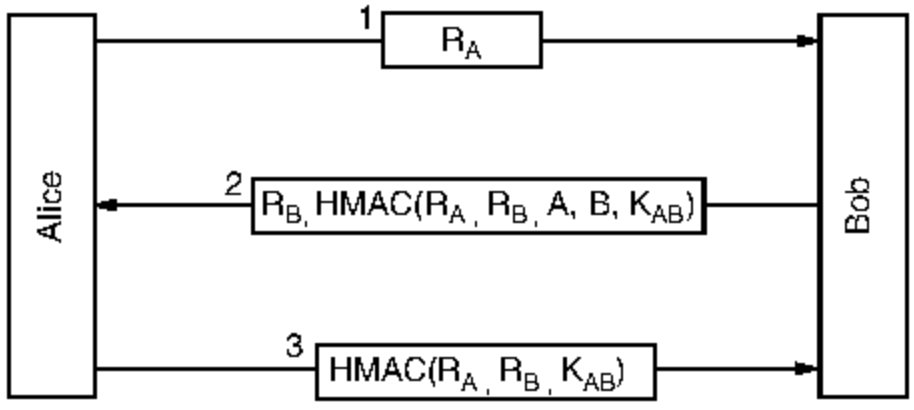


Figure 8-36. Authentication using HMACs.

Using HMACs is not the only way to use this idea. An alternative scheme that is often used instead of computing the HMAC over a series of items is to encrypt the items sequentially using cipher block chaining.

8.7.2 Establishing a Shared Key: The Diffie-Hellman Key Exchange

So far, we have assumed that Alice and Bob share a secret key. Suppose that they do not (because so far there is no universally accepted PKI for signing and distributing certificates). How can they establish one? One way would be for Alice to call Bob and give him her key on the phone, but he would probably start out by saying: "How do I know you are Alice and not Trudy?" They could try to arrange a meeting, with each one bringing a passport, a driver's license, and three major credit cards, but being busy people, they might not be able to find a mutually acceptable date for months. Fortunately, incredible as it may sound, there is a way for total strangers to establish a shared secret key in broad daylight, even with Trudy carefully recording every message.

The protocol that allows strangers to establish a shared secret key is called the **Diffie-Hellman key exchange** (Diffie and Hellman, 1976) and works as follows. Alice and Bob have to agree on two large numbers, n and g, where n is a prime, $(n - 1)/2$ is also a prime, and certain conditions apply to g. These numbers may be public, so either one of them can just pick n and g and tell the other openly. Now Alice picks a large (say, 1024-bit) number, x, and keeps it secret. Similarly, Bob picks a large secret number, y.

Alice initiates the key exchange protocol by sending Bob a message containing $(n, g, g^x \bmod n)$, as shown in Fig. 8-37. Bob responds by sending Alice a message containing $g^y \bmod n$. Now Alice raises the number Bob sent her to the xth power modulo n to get $(g^y \bmod n)^x \bmod n$. Bob performs a similar operation to get $(g^x \bmod n)^y \bmod n$. By the laws of modular arithmetic, both calculations yield $g^{xy} \bmod n$. Lo and behold, as if by magic, Alice and Bob suddenly share a secret key, $g^{xy} \bmod n$.

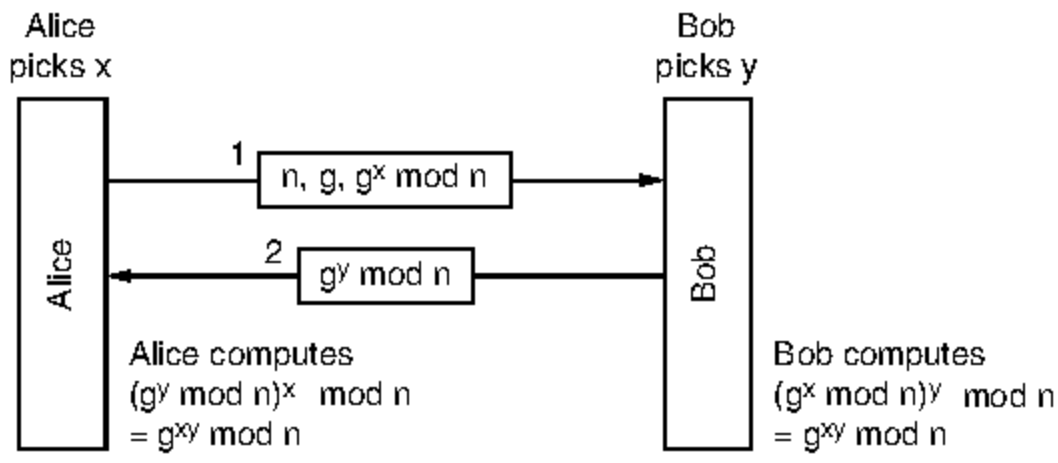


Figure 8-37. The Diffie-Hellman key exchange.

Trudy, of course, has seen both messages. She knows g and n from message 1. If she could compute x and y, she could figure out the secret key. The trouble is, given only $g^x \bmod n$, she cannot find x. No practical algorithm for computing discrete logarithms modulo a very large prime number is known.

To make this example more concrete, we will use the (completely unrealistic) values of $n = 47$ and $g = 3$. Alice picks $x = 8$ and Bob picks $y = 10$. Both of these are kept secret. Alice's message to Bob is (47, 3, 28) because $3^8 \bmod 47$ is 28. Bob's message to Alice is (17). Alice computes $17^8 \bmod 47$, which is 4. Bob computes $28^{10} \bmod 47$, which is 4. Alice and Bob have now independently determined that the secret key is now 4. To find the key, Trudy now has to solve the equation $3^x \bmod 47 = 28$, which can be done by exhaustive search for small numbers like this, but not when all the numbers are hundreds of bits long. All currently known algorithms simply take far too long, even on massively parallel, lightning fast supercomputers.

Despite the elegance of the Diffie-Hellman algorithm, there is a problem: when Bob gets the triple (47, 3, 28), how does he know it is from Alice and not from Trudy? There is no way he can know. Unfortunately, Trudy can exploit this fact to deceive both Alice and Bob, as illustrated in Fig. 8-38. Here, while Alice and Bob are choosing x and y, respectively, Trudy picks her own random number, z. Alice sends message 1, intended for Bob. Trudy intercepts it and sends message 2 to Bob, using the correct g and n (which are public anyway) but with her own z instead of x. She also sends message 3 back to Alice. Later Bob sends message 4 to Alice, which Trudy again intercepts and keeps.

Now everybody does the modular arithmetic. Alice computes the secret key as $g^{xz} \bmod n$, and so does Trudy (for messages to Alice). Bob computes $g^{yz} \bmod n$ and so does Trudy (for messages to Bob). Alice thinks she is talking to Bob, so she establishes a session key (with Trudy). So does Bob. Every message that Alice sends on the encrypted session is captured by Trudy, stored, modified if desired, and then (optionally) passed on to Bob. Similarly, in the other direction, Trudy sees everything and can modify all messages at will, while both Alice and Bob are under the illusion that they have a secure channel to one another. For this

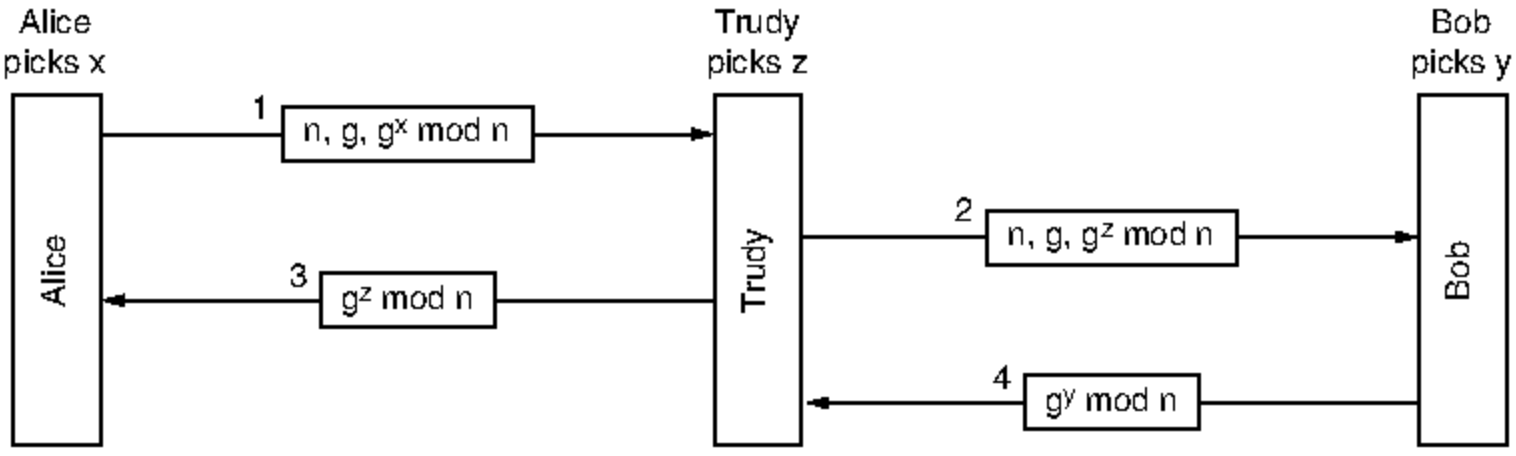


Figure 8-38. The man-in-the-middle attack.

reason, the attack is known as the **man-in-the-middle attack**. It is also called the **bucket brigade attack**, because it vaguely resembles an old-time volunteer fire department passing buckets along the line from the fire truck to the fire.

8.7.3 Authentication Using a Key Distribution Center

Setting up a shared secret with a stranger almost worked, but not quite. On the other hand, it probably was not worth doing in the first place (sour grapes attack). To talk to n people this way, you would need n keys. For popular people, key management would become a real burden, especially if each key had to be stored on a separate plastic chip card.

A different approach is to introduce a trusted key distribution center. In this model, each user has a single key shared with the KDC. Authentication and session key management now go through the KDC. The simplest known KDC authentication protocol involving two parties and a trusted KDC is depicted in Fig. 8-39.

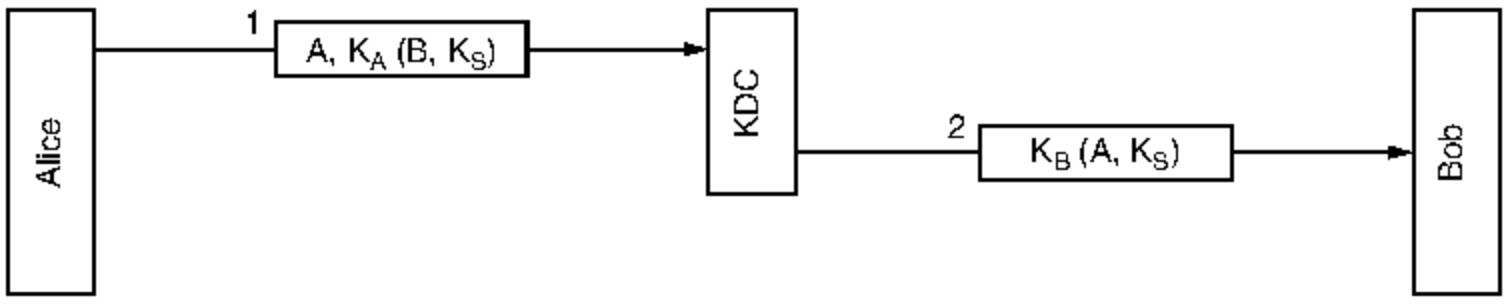


Figure 8-39. A first attempt at an authentication protocol using a KDC.

The idea behind this protocol is simple: Alice picks a session key, K_S, and tells the KDC that she wants to talk to Bob using K_S. This message is encrypted

with the secret key Alice shares (only) with the KDC, K_A. The KDC decrypts this message, extracting Bob's identity and the session key. It then constructs a new message containing Alice's identity and the session key and sends this message to Bob. This encryption is done with K_B, the secret key Bob shares with the KDC. When Bob decrypts the message, he learns that Alice wants to talk to him and which key she wants to use.

The authentication here happens for free. The KDC knows that message 1 must have come from Alice, since no one else would have been able to encrypt it with Alice's secret key. Similarly, Bob knows that message 2 must have come from the KDC, whom he trusts, since no one else knows his secret key.

Unfortunately, this protocol has a serious flaw. Trudy needs some money, so she figures out some legitimate service she can perform for Alice, makes an attractive offer, and gets the job. After doing the work, Trudy then politely requests Alice to pay by bank transfer. Alice then establishes a session key with her banker, Bob. Then she sends Bob a message requesting money to be transferred to Trudy's account.

Meanwhile, Trudy is back to her old ways, snooping on the network. She copies both message 2 in Fig. 8-39 and the money-transfer request that follows it. Later, she replays both of them to Bob who thinks: "Alice must have hired Trudy again. She clearly does good work." Bob then transfers an equal amount of money from Alice's account to Trudy's. Some time after the 50th message pair, Bob runs out of the office to find Trudy to offer her a big loan so she can expand her obviously successful business. This problem is called the **replay attack**.

Several solutions to the replay attack are possible. The first one is to include a timestamp in each message. Then, if anyone receives an obsolete message, it can be discarded. The trouble with this approach is that clocks are never exactly synchronized over a network, so there has to be some interval during which a timestamp is valid. Trudy can replay the message during this interval and get away with it.

The second solution is to put a nonce in each message. Each party then has to remember all previous nonces and reject any message containing a previously used nonce. But nonces have to be remembered forever, lest Trudy try replaying a 5-year-old message. Also, if some machine crashes and it loses its nonce list, it is again vulnerable to a replay attack. Timestamps and nonces can be combined to limit how long nonces have to be remembered, but clearly the protocol is going to get a lot more complicated.

A more sophisticated approach to mutual authentication is to use a multiway challenge-response protocol. A well-known example of such a protocol is the **Needham-Schroeder authentication** protocol (Needham and Schroeder, 1978), one variant of which is shown in Fig. 8-40.

The protocol begins with Alice telling the KDC that she wants to talk to Bob. This message contains a large random number, R_A, as a nonce. The KDC sends back message 2 containing Alice's random number, a session key, and a ticket

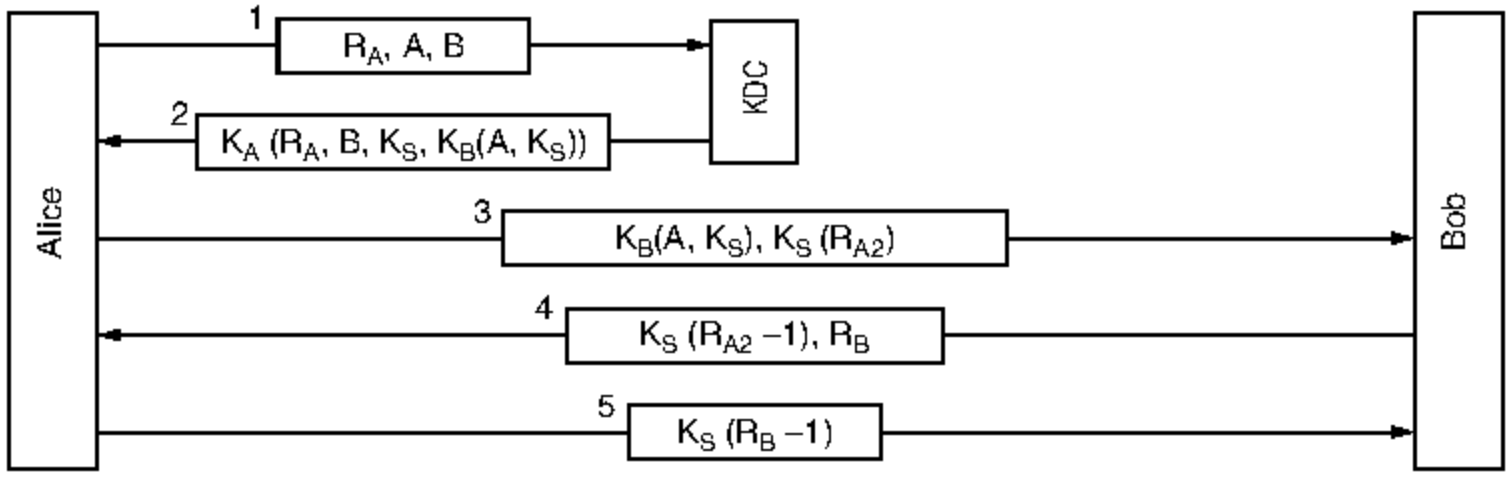


Figure 8-40. The Needham-Schroeder authentication protocol.

that she can send to Bob. The point of the random number, R_A, is to assure Alice that message 2 is fresh, and not a replay. Bob's identity is also enclosed in case Trudy gets any funny ideas about replacing B in message 1 with her own identity so the KDC will encrypt the ticket at the end of message 2 with K_T instead of K_B. The ticket encrypted with K_B is included inside the encrypted message to prevent Trudy from replacing it with something else on the way back to Alice.

Alice now sends the ticket to Bob, along with a new random number, R_{A2}, encrypted with the session key, K_S. In message 4, Bob sends back $K_S(R_{A2} - 1)$ to prove to Alice that she is talking to the real Bob. Sending back $K_S(R_{A2})$ would not have worked, since Trudy could just have stolen it from message 3.

After receiving message 4, Alice is now convinced that she is talking to Bob and that no replays could have been used so far. After all, she just generated R_{A2} a few milliseconds ago. The purpose of message 5 is to convince Bob that it is indeed Alice he is talking to, and no replays are being used here either. By having each party both generate a challenge and respond to one, the possibility of any kind of replay attack is eliminated.

Although this protocol seems pretty solid, it does have a slight weakness. If Trudy ever manages to obtain an old session key in plaintext, she can initiate a new session with Bob by replaying the message 3 that corresponds to the compromised key and convince him that she is Alice (Denning and Sacco, 1981). This time she can plunder Alice's bank account without having to perform the legitimate service even once.

Needham and Schroeder (1987) later published a protocol that corrects this problem. In the same issue of the same journal, Otway and Rees (1987) also published a protocol that solves the problem in a shorter way. Figure 8-41 shows a slightly modified Otway-Rees protocol.

In the Otway-Rees protocol, Alice starts out by generating a pair of random numbers: R, which will be used as a common identifier, and R_A, which Alice will use to challenge Bob. When Bob gets this message, he constructs a new message from the encrypted part of Alice's message and an analogous one of his own.

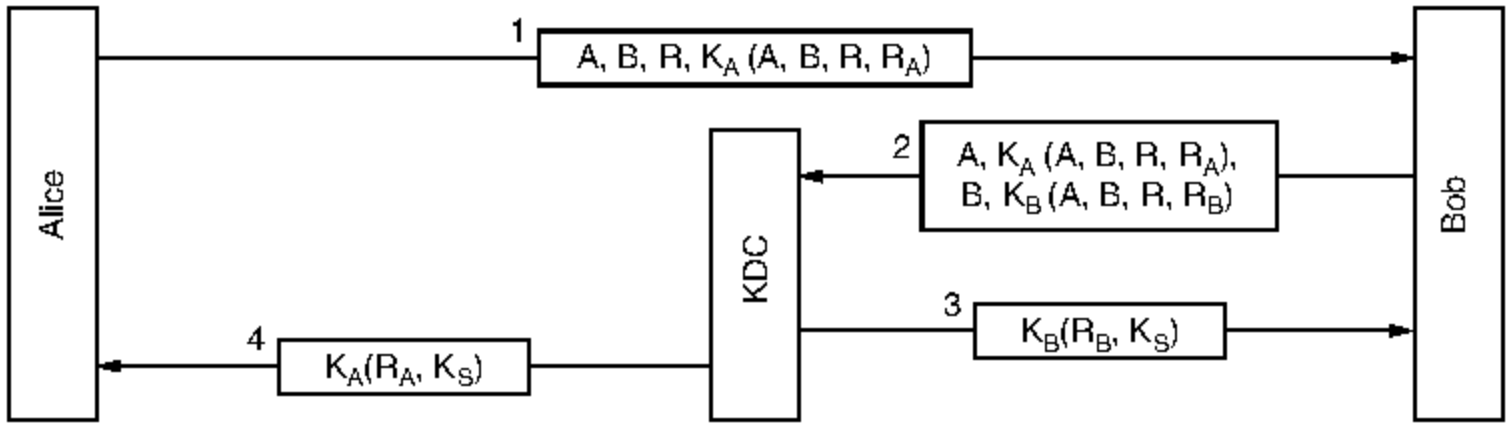


Figure 8-41. The Otway-Rees authentication protocol (slightly simplified).

Both the parts encrypted with K_A and K_B identify Alice and Bob, contain the common identifier, and contain a challenge.

The KDC checks to see if the R in both parts is the same. It might not be if Trudy has tampered with R in message 1 or replaced part of message 2. If the two Rs match, the KDC believes that the request message from Bob is valid. It then generates a session key and encrypts it twice, once for Alice and once for Bob. Each message contains the receiver's random number, as proof that the KDC, and not Trudy, generated the message. At this point, both Alice and Bob are in possession of the same session key and can start communicating. The first time they exchange data messages, each one can see that the other one has an identical copy of K_S, so the authentication is then complete.

8.7.4 Authentication Using Kerberos

An authentication protocol used in many real systems (including Windows 2000 and later versions) is **Kerberos**, which is based on a variant of Needham-Schroeder. It is named for a multiheaded dog in Greek mythology that used to guard the entrance to Hades (presumably to keep undesirables out). Kerberos was designed at M.I.T. to allow workstation users to access network resources in a secure way. Its biggest difference from Needham-Schroeder is its assumption that all clocks are fairly well synchronized. The protocol has gone through several iterations. V5 is the one that is widely used in industry and defined in RFC 4120. The earlier version, V4, was finally retired after serious flaws were found (Yu et al., 2004). V5 improves on V4 with many small changes to the protocol and some improved features, such as the fact that it no longer relies on the now-dated DES. For more information, see Neuman and Ts'o (1994).

Kerberos involves three servers in addition to Alice (a client workstation):

1. Authentication Server (AS): Verifies users during login.
2. Ticket-Granting Server (TGS): Issues "proof of identity tickets."
3. Bob the server: Actually does the work Alice wants performed.

AS is similar to a KDC in that it shares a secret password with every user. The TGS's job is to issue tickets that can convince the real servers that the bearer of a TGS ticket really is who he or she claims to be.

To start a session, Alice sits down at an arbitrary public workstation and types her name. The workstation sends her name and the name of the TGS to the AS in plaintext, as shown in message 1 of Fig. 8-42. What comes back is a session key and a ticket, $K_{TGS}(A, K_S, t)$, intended for the TGS. The session key is encrypted using Alice's secret key, so that only Alice can decrypt it. Only when message 2 arrives does the workstation ask for Alice's password—not before then. The password is then used to generate K_A in order to decrypt message 2 and obtain the session key.

At this point, the workstation overwrites Alice's password to make sure that it is only inside the workstation for a few milliseconds at most. If Trudy tries logging in as Alice, the password she types will be wrong and the workstation will detect this because the standard part of message 2 will be incorrect.

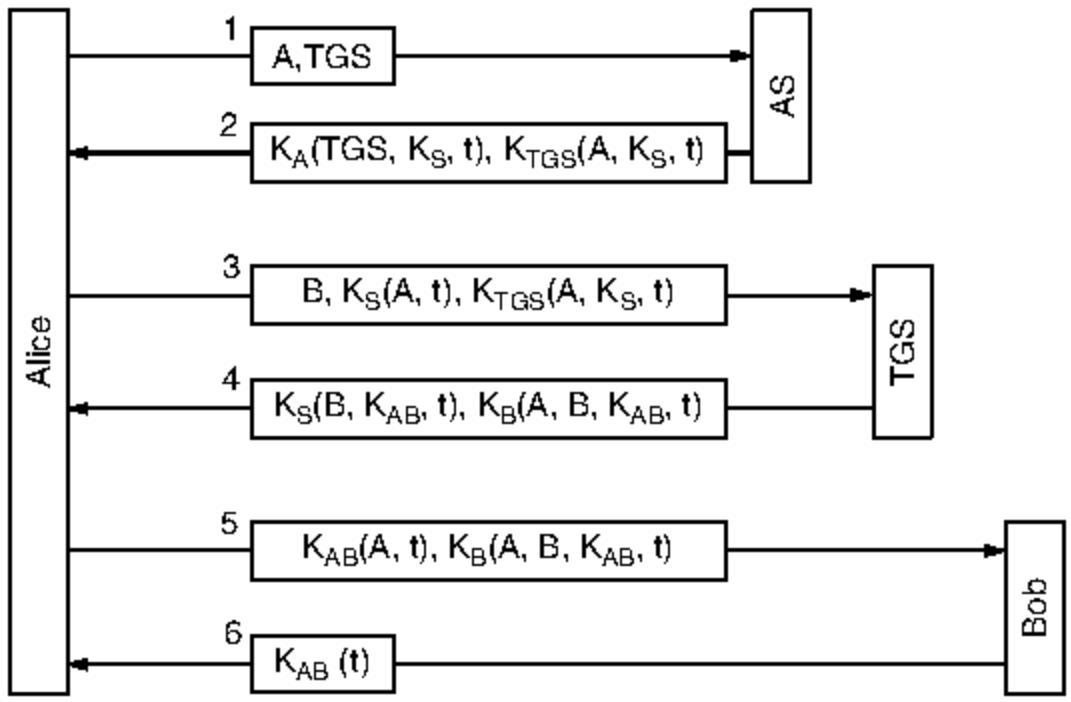


Figure 8-42. The operation of Kerberos V5.

After she logs in, Alice may tell the workstation that she wants to contact Bob the file server. The workstation then sends message 3 to the TGS asking for a ticket to use with Bob. The key element in this request is the ticket $K_{TGS}(A, K_S, t)$, which is encrypted with the TGS's secret key and used as proof that the sender really is Alice. The TGS responds in message 4 by creating a session key, K_{AB}, for Alice to use with Bob. Two versions of it are sent back. The first is encrypted with only K_S, so Alice can read it. The second is another ticket, encrypted with Bob's key, K_B, so Bob can read it.

Trudy can copy message 3 and try to use it again, but she will be foiled by the encrypted timestamp, t, sent along with it. Trudy cannot replace the timestamp with a more recent one, because she does not know K_S, the session key Alice uses to talk to the TGS. Even if Trudy replays message 3 quickly, all she will get is another copy of message 4, which she could not decrypt the first time and will not be able to decrypt the second time either.

Now Alice can send K_{AB} to Bob via the new ticket to establish a session with him (message 5). This exchange is also timestamped. The optional response (message 6) is proof to Alice that she is actually talking to Bob, not to Trudy.

After this series of exchanges, Alice can communicate with Bob under cover of K_{AB}. If she later decides she needs to talk to another server, Carol, she just repeats message 3 to the TGS, only now specifying C instead of B. The TGS will promptly respond with a ticket encrypted with K_C that Alice can send to Carol and that Carol will accept as proof that it came from Alice.

The point of all this work is that now Alice can access servers all over the network in a secure way and her password never has to go over the network. In fact, it only had to be in her own workstation for a few milliseconds. However, note that each server does its own authorization. When Alice presents her ticket to Bob, this merely proves to Bob who sent it. Precisely what Alice is allowed to do is up to Bob.

Since the Kerberos designers did not expect the entire world to trust a single authentication server, they made provision for having multiple **realms**, each with its own AS and TGS. To get a ticket for a server in a distant realm, Alice would ask her own TGS for a ticket accepted by the TGS in the distant realm. If the distant TGS has registered with the local TGS (the same way local servers do), the local TGS will give Alice a ticket valid at the distant TGS. She can then do business over there, such as getting tickets for servers in that realm. Note, however, that for parties in two realms to do business, each one must trust the other's TGS. Otherwise, they cannot do business.

8.7.5 Authentication Using Public-Key Cryptography

Mutual authentication can also be done using public-key cryptography. To start with, Alice needs to get Bob's public key. If a PKI exists with a directory server that hands out certificates for public keys, Alice can ask for Bob's, as shown in Fig. 8-43 as message 1. The reply, in message 2, is an X.509 certificate containing Bob's public key. When Alice verifies that the signature is correct, she sends Bob a message containing her identity and a nonce.

When Bob receives this message, he has no idea whether it came from Alice or from Trudy, but he plays along and asks the directory server for Alice's public key (message 4), which he soon gets (message 5). He then sends Alice message 6, containing Alice's R_A, his own nonce, R_B, and a proposed session key, K_S.

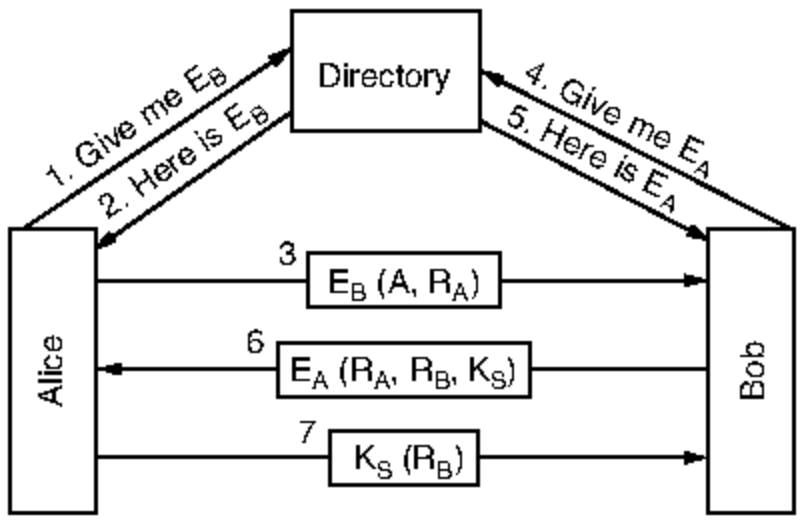


Figure 8-43. Mutual authentication using public-key cryptography.

When Alice gets message 6, she decrypts it using her private key. She sees R_A in it, which gives her a warm feeling inside. The message must have come from Bob, since Trudy has no way of determining R_A. Furthermore, it must be fresh and not a replay, since she just sent Bob R_A. Alice agrees to the session by sending back message 7. When Bob sees R_B encrypted with the session key he just generated, he knows Alice got message 6 and verified R_A. Bob is now a happy camper.

What can Trudy do to try to subvert this protocol? She can fabricate message 3 and trick Bob into probing Alice, but Alice will see an R_A that she did not send and will not proceed further. Trudy cannot forge message 7 back to Bob because she does not know R_B or K_S and cannot determine them without Alice's private key. She is out of luck.

8.8 EMAIL SECURITY

When an email message is sent between two distant sites, it will generally transit dozens of machines on the way. Any of these can read and record the message for future use. In practice, privacy is nonexistent, despite what many people think. Nevertheless, many people would like to be able to send email that can be read by the intended recipient and no one else: not their boss and not even their government. This desire has stimulated several people and groups to apply the cryptographic principles we studied earlier to email to produce secure email. In the following sections we will study a widely used secure email system, PGP, and then briefly mention one other, S/MIME. For additional information about secure email, see Kaufman et al. (2002) and Schneier (1995).

8.8.1 PGP—Pretty Good Privacy

Our first example, **PGP** (**Pretty Good Privacy**) is essentially the brainchild of one person, Phil Zimmermann (1995a, 1995b). Zimmermann is a privacy advocate whose motto is: "If privacy is outlawed, only outlaws will have privacy." Released in 1991, PGP is a complete email security package that provides privacy, authentication, digital signatures, and compression, all in an easy-to-use form. Furthermore, the complete package, including all the source code, is distributed free of charge via the Internet. Due to its quality, price (zero), and easy availability on UNIX, Linux, Windows, and Mac OS platforms, it is widely used today.

PGP encrypts data by using a block cipher called **IDEA** (**International Data Encryption Algorithm**), which uses 128-bit keys. It was devised in Switzerland at a time when DES was seen as tainted and AES had not yet been invented. Conceptually, IDEA is similar to DES and AES: it mixes up the bits in a series of rounds, but the details of the mixing functions are different from DES and AES. Key management uses RSA and data integrity uses MD5, topics that we have already discussed.

PGP has also been embroiled in controversy since day 1 (Levy, 1993). Because Zimmermann did nothing to stop other people from placing PGP on the Internet, where people all over the world could get it, the U.S. Government claimed that Zimmermann had violated U.S. laws prohibiting the export of munitions. The U.S. Government's investigation of Zimmermann went on for 5 years but was eventually dropped, probably for two reasons. First, Zimmermann did not place PGP on the Internet himself, so his lawyer claimed that *he* never exported anything (and then there is the little matter of whether creating a Web site constitutes export at all). Second, the government eventually came to realize that winning a trial meant convincing a jury that a Web site containing a downloadable privacy program was covered by the arms-trafficking law prohibiting the export of war materiel such as tanks, submarines, military aircraft, and nuclear weapons. Years of negative publicity probably did not help much, either.

As an aside, the export rules are bizarre, to put it mildly. The government considered putting code on a Web site to be an illegal export and harassed Zimmermann about it for 5 years. On the other hand, when someone published the complete PGP source code, in C, as a book (in a large font with a checksum on each page to make scanning it in easy) and then exported the book, that was fine with the government because books are not classified as munitions. The sword is mightier than the pen, at least for Uncle Sam.

Another problem PGP ran into involved patent infringement. The company holding the RSA patent, RSA Security, Inc., alleged that PGP's use of the RSA algorithm infringed on its patent, but that problem was settled with releases starting at 2.6. Furthermore, PGP uses another patented encryption algorithm, IDEA, whose use caused some problems at first.

Since PGP is open source, various people and groups have modified it and produced a number of versions. Some of these were designed to get around the munitions laws, others were focused on avoiding the use of patented algorithms, and still others wanted to turn it into a closed-source commercial product. Although the munitions laws have now been slightly liberalized (otherwise, products using AES would not have been exportable from the U.S.), and the RSA patent expired in September 2000, the legacy of all these problems is that several incompatible versions of PGP are in circulation, under various names. The discussion below focuses on classic PGP, which is the oldest and simplest version. Another popular version, Open PGP, is described in RFC 2440. Yet another is the GNU Privacy Guard.

PGP intentionally uses existing cryptographic algorithms rather than inventing new ones. It is largely based on algorithms that have withstood extensive peer review and were not designed or influenced by any government agency trying to weaken them. For people who distrust government, this property is a big plus.

PGP supports text compression, secrecy, and digital signatures and also provides extensive key management facilities, but, oddly enough, not email facilities. It is like a preprocessor that takes plaintext as input and produces signed ciphertext in base64 as output. This output can then be emailed, of course. Some PGP implementations call a user agent as the final step to actually send the message.

To see how PGP works, let us consider the example of Fig. 8-44. Here, Alice wants to send a signed plaintext message, P, to Bob in a secure way. Both Alice and Bob have private (D_X) and public (E_X) RSA keys. Let us assume that each one knows the other's public key; we will cover PGP key management shortly.

Alice starts out by invoking the PGP program on her computer. PGP first hashes her message, P, using MD5, and then encrypts the resulting hash using her private RSA key, D_A. When Bob eventually gets the message, he can decrypt the hash with Alice's public key and verify that the hash is correct. Even if someone else (e.g., Trudy) could acquire the hash at this stage and decrypt it with Alice's known public key, the strength of MD5 guarantees that it would be computationally infeasible to produce another message with the same MD5 hash.

The encrypted hash and the original message are now concatenated into a single message, *P1*, and compressed using the ZIP program, which uses the Ziv-Lempel algorithm (Ziv and Lempel, 1977). Call the output of this step *P1.Z*.

Next, PGP prompts Alice for some random input. Both the content and the typing speed are used to generate a 128-bit IDEA message key, K_M (called a session key in the PGP literature, but this is really a misnomer since there is no session). K_M is now used to encrypt *P1.Z* with IDEA in cipher feedback mode. In addition, K_M is encrypted with Bob's public key, E_B. These two components are then concatenated and converted to base64, as we discussed in the section on MIME in Chap. 7. The resulting message contains only letters, digits, and the symbols +, /, and =, which means it can be put into an RFC 822 body and be expected to arrive unmodified.

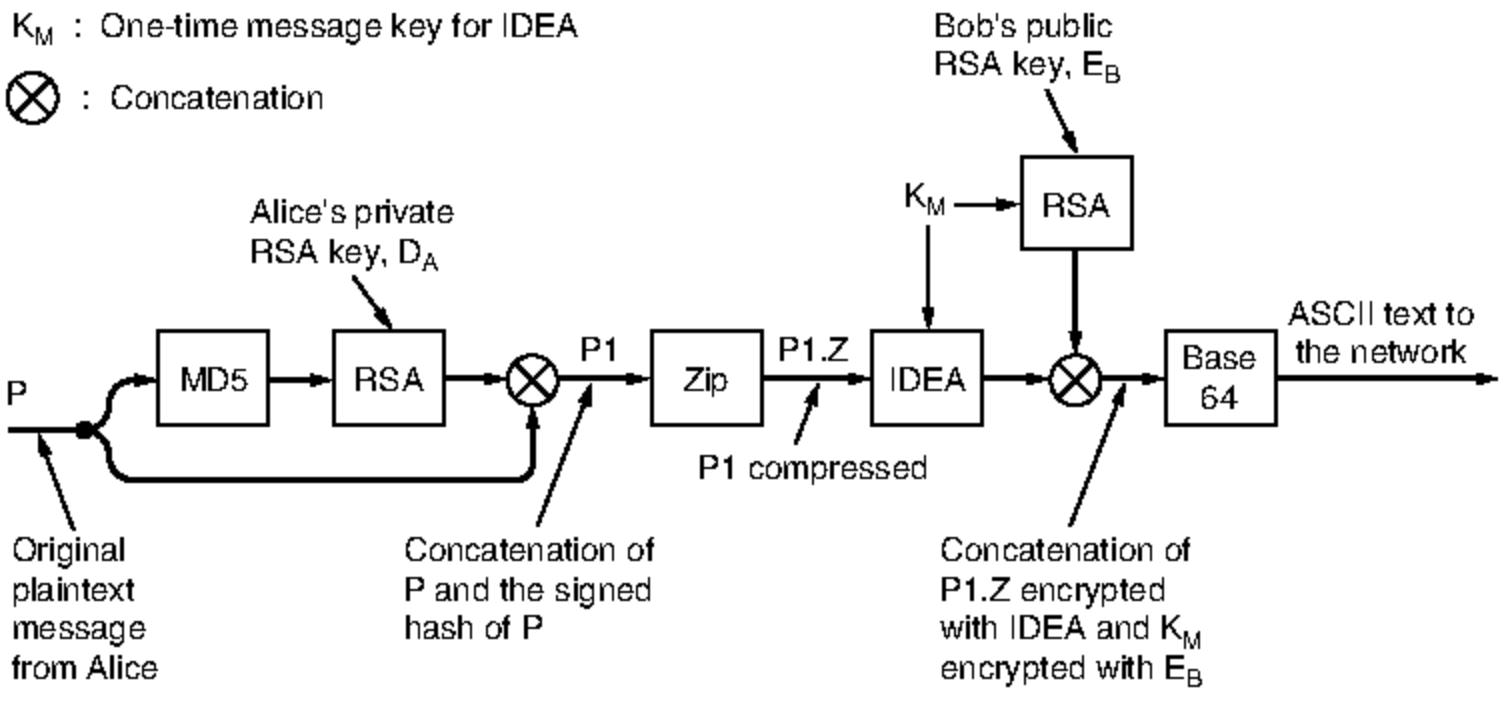


Figure 8-44. PGP in operation for sending a message.

When Bob gets the message, he reverses the base64 encoding and decrypts the IDEA key using his private RSA key. Using this key, he decrypts the message to get *P1.Z*. After decompressing it, Bob separates the plaintext from the encrypted hash and decrypts the hash using Alice's public key. If the plaintext hash agrees with his own MD5 computation, he knows that *P* is the correct message and that it came from Alice.

It is worth noting that RSA is only used in two places here: to encrypt the 128-bit MD5 hash and to encrypt the 128-bit IDEA key. Although RSA is slow, it has to encrypt only 256 bits, not a large volume of data. Furthermore, all 256 plaintext bits are exceedingly random, so a considerable amount of work will be required on Trudy's part just to determine if a guessed key is correct. The heavy-duty encryption is done by IDEA, which is orders of magnitude faster than RSA. Thus, PGP provides security, compression, and a digital signature and does so in a much more efficient way than the scheme illustrated in Fig. 8-19.

PGP supports four RSA key lengths. It is up to the user to select the one that is most appropriate. The lengths are:

1. Casual (384 bits): Can be broken easily today.
2. Commercial (512 bits): Breakable by three-letter organizations.
3. Military (1024 bits): Not breakable by anyone on earth.
4. Alien (2048 bits): Not breakable by anyone on other planets, either.

Since RSA is only used for two small computations, everyone should use alien-strength keys all the time.

The format of a classic PGP message is shown in Fig. 8-45. Numerous other formats are also in use. The message has three parts, containing the IDEA key, the signature, and the message, respectively. The key part contains not only the key, but also a key identifier, since users are permitted to have multiple public keys.

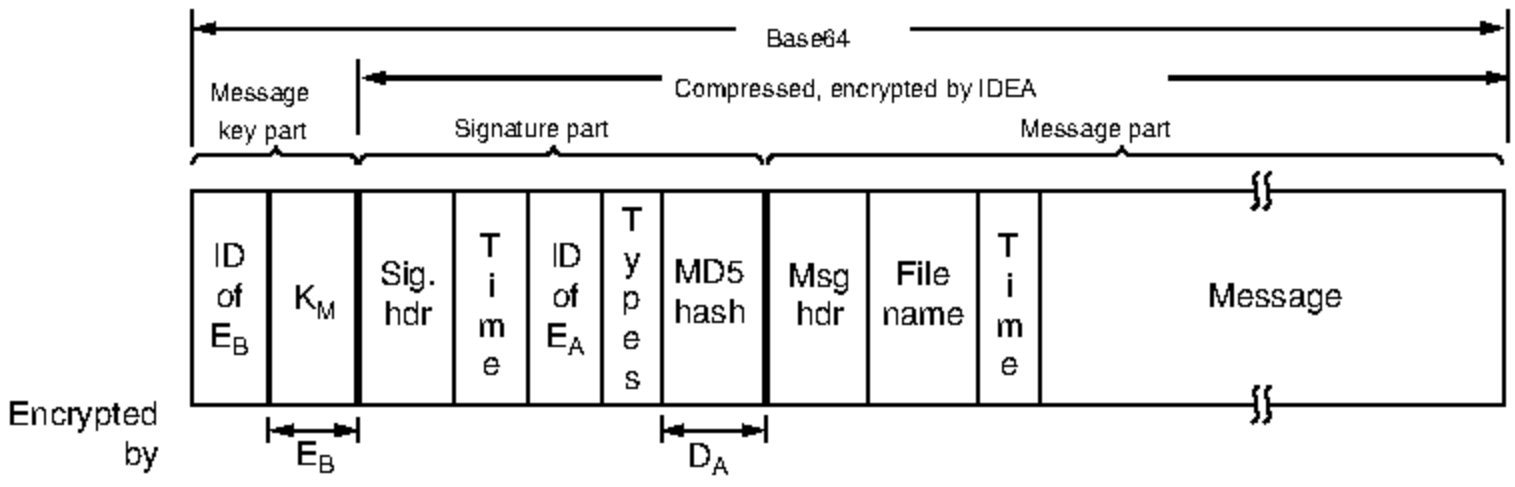


Figure 8-45. A PGP message.

The signature part contains a header, which will not concern us here. The header is followed by a timestamp, the identifier for the sender's public key that can be used to decrypt the signature hash, some type information that identifies the algorithms used (to allow MD6 and RSA2 to be used when they are invented), and the encrypted hash itself.

The message part also contains a header, the default name of the file to be used if the receiver writes the file to the disk, a message creation timestamp, and, finally, the message itself.

Key management has received a large amount of attention in PGP as it is the Achilles' heel of all security systems. Key management works as follows. Each user maintains two data structures locally: a private key ring and a public key ring. The **private key ring** contains one or more personal private/public key pairs. The reason for supporting multiple pairs per user is to permit users to change their public keys periodically or when one is thought to have been compromised, without invalidating messages currently in preparation or in transit. Each pair has an identifier associated with it so that a message sender can tell the recipient which public key was used to encrypt it. Message identifiers consist of the low-order 64 bits of the public key. Users are themselves responsible for avoiding conflicts in their public-key identifiers. The private keys on disk are encrypted using a special (arbitrarily long) password to protect them against sneak attacks.

The **public key ring** contains public keys of the user's correspondents. These are needed to encrypt the message keys associated with each message. Each entry

on the public key ring contains not only the public key, but also its 64-bit identifier and an indication of how strongly the user trusts the key.

The problem being tackled here is the following. Suppose that public keys are maintained on bulletin boards. One way for Trudy to read Bob's secret email is to attack the bulletin board and replace Bob's public key with one of her choice. When Alice later fetches the key allegedly belonging to Bob, Trudy can mount a bucket brigade attack on Bob.

To prevent such attacks, or at least minimize the consequences of them, Alice needs to know how much to trust the item called "Bob's key" on her public key ring. If she knows that Bob personally handed her a CD-ROM containing the key, she can set the trust value to the highest value. It is this decentralized, user-controlled approach to public-key management that sets PGP apart from centralized PKI schemes.

Nevertheless, people do sometimes obtain public keys by querying a trusted key server. For this reason, after X.509 was standardized, PGP supported these certificates as well as the traditional PGP public key ring mechanism. All current versions of PGP have X.509 support.

8.8.2 S/MIME

IETF's venture into email security, called **S/MIME** (**Secure/MIME**), is described in RFCs 2632 through 2643. It provides authentication, data integrity, secrecy, and nonrepudiation. It also is quite flexible, supporting a variety of cryptographic algorithms. Not surprisingly, given the name, S/MIME integrates well with MIME, allowing all kinds of messages to be protected. A variety of new MIME headers are defined, for example, for holding digital signatures.

S/MIME does not have a rigid certificate hierarchy beginning at a single root, which had been one of the political problems that doomed an earlier system called PEM (Privacy Enhanced Mail). Instead, users can have multiple trust anchors. As long as a certificate can be traced back to some trust anchor the user believes in, it is considered valid. S/MIME uses the standard algorithms and protocols we have been examining so far, so we will not discuss it any further here. For the details, please consult the RFCs.

8.9 WEB SECURITY

We have just studied two important areas where security is needed: communications and email. You can think of these as the soup and appetizer. Now it is time for the main course: Web security. The Web is where most of the Trudies hang out nowadays and do their dirty work. In the following sections, we will look at some of the problems and issues relating to Web security.

Web security can be roughly divided into three parts. First, how are objects and resources named securely? Second, how can secure, authenticated connections be established? Third, what happens when a Web site sends a client a piece of executable code? After looking at some threats, we will examine all these issues.

8.9.1 Threats

One reads about Web site security problems in the newspaper almost weekly. The situation is really pretty grim. Let us look at a few examples of what has already happened. First, the home pages of numerous organizations have been attacked and replaced by new home pages of the crackers' choosing. (The popular press calls people who break into computers "hackers," but many programmers reserve that term for great programmers. We prefer to call these people "crackers.") Sites that have been cracked include those belonging to Yahoo!, the U.S. Army, the CIA, NASA, and the *New York Times*. In most cases, the crackers just put up some funny text and the sites were repaired within a few hours.

Now let us look at some much more serious cases. Numerous sites have been brought down by denial-of-service attacks, in which the cracker floods the site with traffic, rendering it unable to respond to legitimate queries. Often, the attack is mounted from a large number of machines that the cracker has already broken into (DDoS attacks). These attacks are so common that they do not even make the news any more, but they can cost the attacked sites thousands of dollars in lost business.

In 1999, a Swedish cracker broke into Microsoft's Hotmail Web site and created a mirror site that allowed anyone to type in the name of a Hotmail user and then read all of the person's current and archived email.

In another case, a 19-year-old Russian cracker named Maxim broke into an e-commerce Web site and stole 300,000 credit card numbers. Then he approached the site owners and told them that if they did not pay him $100,000, he would post all the credit card numbers to the Internet. They did not give in to his blackmail, and he indeed posted the credit card numbers, inflicting great damage on many innocent victims.

In a different vein, a 23-year-old California student emailed a press release to a news agency falsely stating that the Emulex Corporation was going to post a large quarterly loss and that the C.E.O. was resigning immediately. Within hours, the company's stock dropped by 60%, causing stockholders to lose over $2 billion. The perpetrator made a quarter of a million dollars by selling the stock short just before sending the announcement. While this event was not a Web site break-in, it is clear that putting such an announcement on the home page of any big corporation would have a similar effect.

We could (unfortunately) go on like this for many more pages. But it is now time to examine some of the technical issues related to Web security. For more

information about security problems of all kinds, see Anderson (2008a); Stuttard and Pinto (2007); and Schneier (2004). Searching the Internet will also turn up vast numbers of specific cases.

8.9.2 Secure Naming

Let us start with something very basic: Alice wants to visit Bob's Web site. She types Bob's URL into her browser and a few seconds later, a Web page appears. But is it Bob's? Maybe yes and maybe no. Trudy might be up to her old tricks again. For example, she might be intercepting all of Alice's outgoing packets and examining them. When she captures an HTTP *GET* request headed to Bob's Web site, she could go to Bob's Web site herself to get the page, modify it as she wishes, and return the fake page to Alice. Alice would be none the wiser. Worse yet, Trudy could slash the prices at Bob's e-store to make his goods look very attractive, thereby tricking Alice into sending her credit card number to "Bob" to buy some merchandise.

One disadvantage of this classic man-in-the-middle attack is that Trudy has to be in a position to intercept Alice's outgoing traffic and forge her incoming traffic. In practice, she has to tap either Alice's phone line or Bob's, since tapping the fiber backbone is fairly difficult. While active wiretapping is certainly possible, it is a fair amount of work, and while Trudy is clever, she is also lazy. Besides, there are easier ways to trick Alice.

DNS Spoofing

One way would be for Trudy to crack the DNS system or maybe just the DNS cache at Alice's ISP, and replace Bob's IP address (say, 36.1.2.3) with her (Trudy's) IP address (say, 42.9.9.9). That leads to the following attack. The way it is supposed to work is illustrated in Fig. 8-46(a). Here, Alice (1) asks DNS for Bob's IP address, (2) gets it, (3) asks Bob for his home page, and (4) gets that, too. After Trudy has modified Bob's DNS record to contain her own IP address instead of Bob's, we get the situation in Fig. 8-46(b). Here, when Alice looks up Bob's IP address, she gets Trudy's, so all her traffic intended for Bob goes to Trudy. Trudy can now mount a man-in-the-middle attack without having to go to the trouble of tapping any phone lines. Instead, she has to break into a DNS server and change one record, a much easier proposition.

How might Trudy fool DNS? It turns out to be relatively easy. Briefly summarized, Trudy can trick the DNS server at Alice's ISP into sending out a query to look up Bob's address. Unfortunately, since DNS uses UDP, the DNS server has no real way of checking who supplied the answer. Trudy can exploit this property by forging the expected reply and thus injecting a false IP address into the DNS server's cache. For simplicity, we will assume that Alice's ISP does not initially have an entry for Bob's Web site, *bob.com*. If it does, Trudy can wait until it times out and try later (or use other tricks).

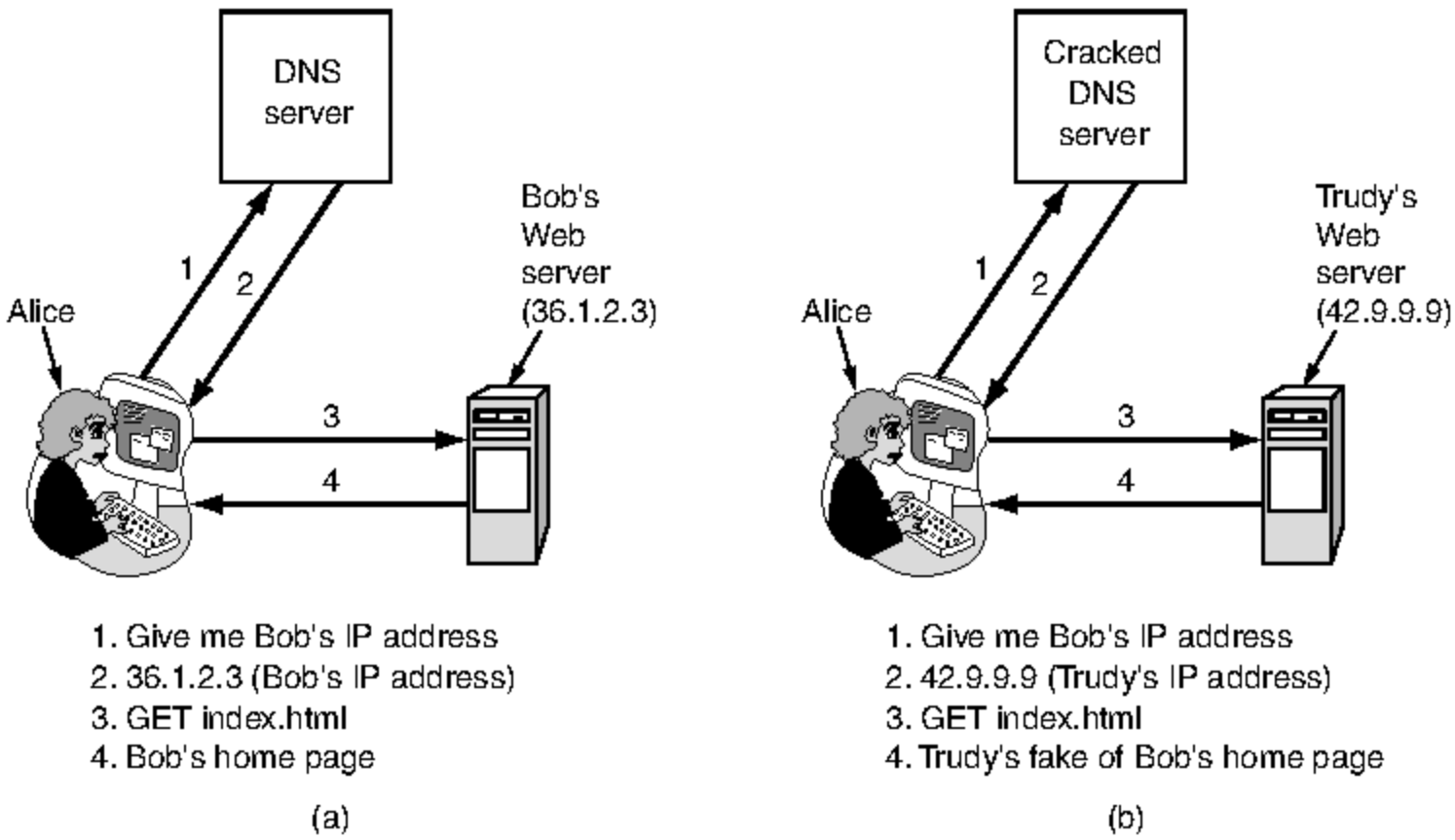


Figure 8-46. (a) Normal situation. (b) An attack based on breaking into a DNS server and modifying Bob's record.

Trudy starts the attack by sending a lookup request to Alice's ISP asking for the IP address of *bob.com*. Since there is no entry for this DNS name, the cache server queries the top-level server for the *com* domain to get one. However, Trudy beats the *com* server to the punch and sends back a false reply saying: "*bob.com* is 42.9.9.9," where that IP address is hers. If her false reply gets back to Alice's ISP first, that one will be cached and the real reply will be rejected as an unsolicited reply to a query no longer outstanding. Tricking a DNS server into installing a false IP address is called **DNS spoofing**. A cache that holds an intentionally false IP address like this is called a **poisoned cache**.

Actually, things are not quite that simple. First, Alice's ISP checks to see that the reply bears the correct IP source address of the top-level server. But since Trudy can put anything she wants in that IP field, she can defeat that test easily since the IP addresses of the top-level servers have to be public.

Second, to allow DNS servers to tell which reply goes with which request, all requests carry a sequence number. To spoof Alice's ISP, Trudy has to know its current sequence number. The easiest way to learn the current sequence number is for Trudy to register a domain herself, say, *trudy-the-intruder.com*. Let us assume its IP address is also 42.9.9.9. She also creates a DNS server for her newly hatched domain, *dns.trudy-the-intruder.com*. It, too, uses Trudy's 42.9.9.9 IP address, since Trudy has only one computer. Now she has to make Alice's ISP aware of her DNS server. That is easy to do. All she has to do is ask Alice's ISP for *foobar.trudy-the-intruder.com*, which will cause Alice's ISP to find out who serves Trudy's new domain by asking the top-level *com* server.

With *dns.trudy-the-intruder.com* safely in the cache at Alice's ISP, the real attack can start. Trudy now queries Alice's ISP for *www.trudy-the-intruder.com*. The ISP naturally sends Trudy's DNS server a query asking for it. This query bears the sequence number that Trudy is looking for. Quick like a bunny, Trudy asks Alice's ISP to look up Bob. She immediately answers her own question by sending the ISP a forged reply, allegedly from the top-level *com* server, saying: "*bob.com* is 42.9.9.9". This forged reply carries a sequence number one higher than the one she just received. While she is at it, she can also send a second forgery with a sequence number two higher, and maybe a dozen more with increasing sequence numbers. One of them is bound to match. The rest will just be thrown out. When Alice's forged reply arrives, it is cached; when the real reply comes in later, it is rejected since no query is then outstanding.

Now when Alice looks up *bob.com*, she is told to use 42.9.9.9, Trudy's address. Trudy has mounted a successful man-in-the-middle attack from the comfort of her own living room. The various steps to this attack are illustrated in Fig. 8-47. This one specific attack can be foiled by having DNS servers use random IDs in their queries rather than just counting, but it seems that every time one hole is plugged, another one turns up. In particular, the IDs are only 16 bits, so working through all of them is easy when it is a computer that is doing the guessing.

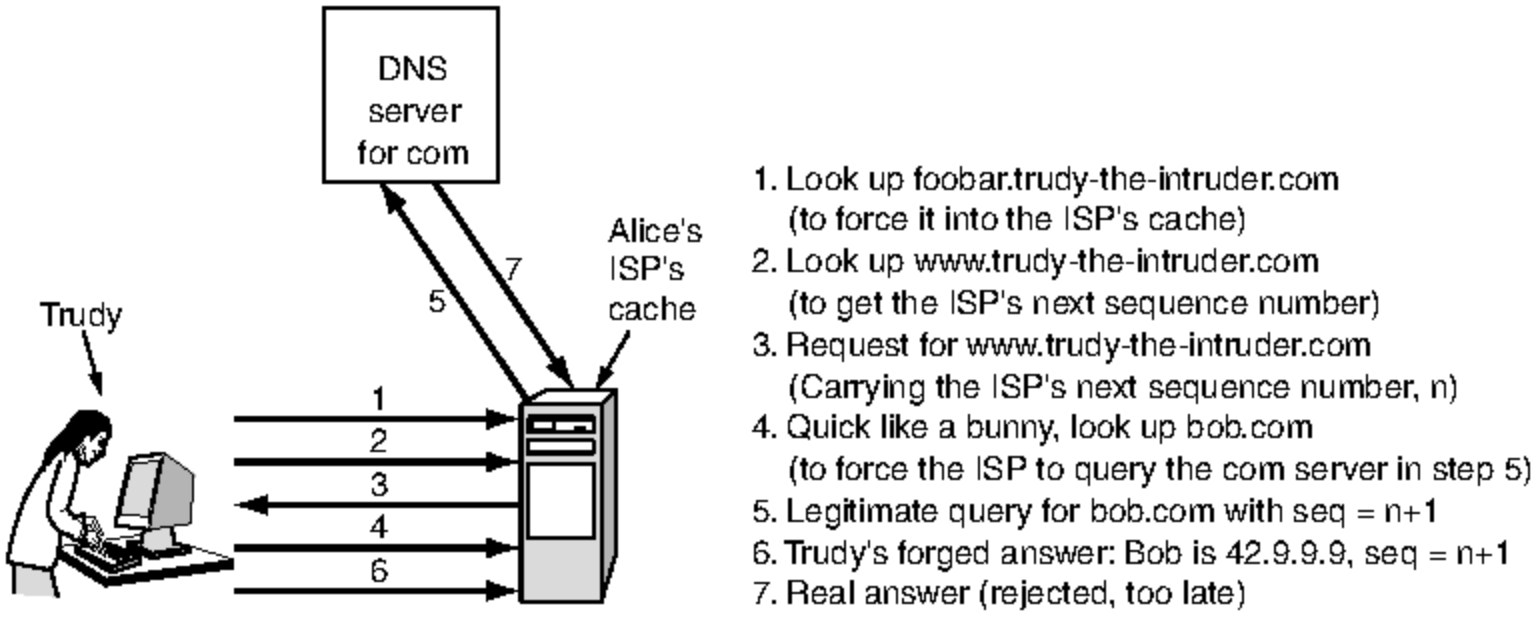


Figure 8-47. How Trudy spoofs Alice's ISP.

Secure DNS

The real problem is that DNS was designed at a time when the Internet was a research facility for a few hundred universities, and neither Alice, nor Bob, nor Trudy was invited to the party. Security was not an issue then; making the Internet work at all was the issue. The environment has changed radically over the

years, so in 1994 IETF set up a working group to make DNS fundamentally secure. This (ongoing) project is known as **DNSsec** (**DNS security**); its first output was presented in RFC 2535. Unfortunately, DNSsec has not been fully deployed yet, so numerous DNS servers are still vulnerable to spoofing attacks.

DNSsec is conceptually extremely simple. It is based on public-key cryptography. Every DNS zone (in the sense of Fig. 7-5) has a public/private key pair. All information sent by a DNS server is signed with the originating zone's private key, so the receiver can verify its authenticity.

DNSsec offers three fundamental services:

1. Proof of where the data originated.
2. Public key distribution.
3. Transaction and request authentication.

The main service is the first one, which verifies that the data being returned has been approved by the zone's owner. The second one is useful for storing and retrieving public keys securely. The third one is needed to guard against playback and spoofing attacks. Note that secrecy is not an offered service since all the information in DNS is considered public. Since phasing in DNSsec is expected to take several years, the ability for security-aware servers to interwork with security-ignorant servers is essential, which implies that the protocol cannot be changed. Let us now look at some of the details.

DNS records are grouped into sets called **RRSets** (**Resource Record Sets**), with all the records having the same name, class, and type being lumped together in a set. An RRSet may contain multiple *A* records, for example, if a DNS name resolves to a primary IP address and a secondary IP address. The RRSets are extended with several new record types (discussed below). Each RRSet is cryptographically hashed (e.g., using SHA-1). The hash is signed by the zone's private key (e.g., using RSA). The unit of transmission to clients is the signed RRSet. Upon receipt of a signed RRSet, the client can verify whether it was signed by the private key of the originating zone. If the signature agrees, the data are accepted. Since each RRSet contains its own signature, RRSets can be cached anywhere, even at untrustworthy servers, without endangering the security.

DNSsec introduces several new record types. The first of these is the *KEY* record. This records holds the public key of a zone, user, host, or other principal, the cryptographic algorithm used for signing, the protocol used for transmission, and a few other bits. The public key is stored naked. X.509 certificates are not used due to their bulk. The algorithm field holds a 1 for MD5/RSA signatures (the preferred choice), and other values for other combinations. The protocol field can indicate the use of IPsec or other security protocols, if any.

The second new record type is the *SIG* record. It holds the signed hash according to the algorithm specified in the *KEY* record. The signature applies to all the records in the RRSet, including any *KEY* records present, but excluding

itself. It also holds the times when the signature begins its period of validity and when it expires, as well as the signer's name and a few other items.

The DNSsec design is such that a zone's private key can be kept offline. Once or twice a day, the contents of a zone's database can be manually transported (e.g., on CD-ROM) to a disconnected machine on which the private key is located. All the RRSets can be signed there and the *SIG* records thus produced can be conveyed back to the zone's primary server on CD-ROM. In this way, the private key can be stored on a CD-ROM locked in a safe except when it is inserted into the disconnected machine for signing the day's new RRSets. After signing is completed, all copies of the key are erased from memory and the disk and the CD-ROM are returned to the safe. This procedure reduces electronic security to physical security, something people understand how to deal with.

This method of presigning RRSets greatly speeds up the process of answering queries since no cryptography has to be done on the fly. The trade-off is that a large amount of disk space is needed to store all the keys and signatures in the DNS databases. Some records will increase tenfold in size due to the signature.

When a client process gets a signed RRSet, it must apply the originating zone's public key to decrypt the hash, compute the hash itself, and compare the two values. If they agree, the data are considered valid. However, this procedure begs the question of how the client gets the zone's public key. One way is to acquire it from a trusted server, using a secure connection (e.g., using IPsec).

However, in practice, it is expected that clients will be preconfigured with the public keys of all the top-level domains. If Alice now wants to visit Bob's Web site, she can ask DNS for the RRSet of *bob.com*, which will contain his IP address and a *KEY* record containing Bob's public key. This RRSet will be signed by the top-level *com* domain, so Alice can easily verify its validity. An example of what this RRSet might contain is shown in Fig. 8-48.

Domain name	Time to live	Class	Type	Value
bob.com.	86400	IN	A	36.1.2.3
bob.com.	86400	IN	KEY	3682793A7B73F731029CE2737D...
bob.com.	86400	IN	SIG	86947503A8B848F5272E53930C...

Figure 8-48. An example RRSet for *bob.com*. The *KEY* record is Bob's public key. The *SIG* record is the top-level *com* server's signed hash of the *A* and *KEY* records to verify their authenticity.

Now armed with a verified copy of Bob's public key, Alice can ask Bob's DNS server (run by Bob) for the IP address of *www.bob.com*. This RRSet will be signed by Bob's private key, so Alice can verify the signature on the RRSet Bob returns. If Trudy somehow manages to inject a false RRSet into any of the caches, Alice can easily detect its lack of authenticity because the *SIG* record contained in it will be incorrect.

However, DNSsec also provides a cryptographic mechanism to bind a response to a specific query, to prevent the kind of spoof Trudy managed to pull off in Fig. 8-47. This (optional) antispoofing measure adds to the response a hash of the query message signed with the respondent's private key. Since Trudy does not know the private key of the top-level *com* server, she cannot forge a response to a query Alice's ISP sent there. She can certainly get her response back first, but it will be rejected due to its invalid signature over the hashed query.

DNSsec also supports a few other record types. For example, the *CERT* record can be used for storing (e.g., X.509) certificates. This record has been provided because some people want to turn DNS into a PKI. Whether this will actually happen remains to be seen. We will stop our discussion of DNSsec here. For more details, please consult RFC 2535.

8.9.3 SSL—The Secure Sockets Layer

Secure naming is a good start, but there is much more to Web security. The next step is secure connections. We will now look at how secure connections can be achieved. Nothing involving security is simple and this is not either.

When the Web burst into public view, it was initially used for just distributing static pages. However, before long, some companies got the idea of using it for financial transactions, such as purchasing merchandise by credit card, online banking, and electronic stock trading. These applications created a demand for secure connections. In 1995, Netscape Communications Corp., the then-dominant browser vendor, responded by introducing a security package called **SSL** (**Secure Sockets Layer**) to meet this demand. This software and its protocol are now widely used, for example, by Firefox, Safari, and Internet Explorer, so it is worth examining in some detail.

SSL builds a secure connection between two sockets, including

1. Parameter negotiation between client and server.
2. Authentication of the server by the client.
3. Secret communication.
4. Data integrity protection.

We have seen these items before, so there is no need to elaborate on them.

The positioning of SSL in the usual protocol stack is illustrated in Fig. 8-49. Effectively, it is a new layer interposed between the application layer and the transport layer, accepting requests from the browser and sending them down to TCP for transmission to the server. Once the secure connection has been established, SSL's main job is handling compression and encryption. When HTTP is used over SSL, it is called **HTTPS** (**Secure HTTP**), even though it is the standard HTTP protocol. Sometimes it is available at a new port (443) instead of port 80.

As an aside, SSL is not restricted to Web browsers, but that is its most common application. It can also provide mutual authentication.

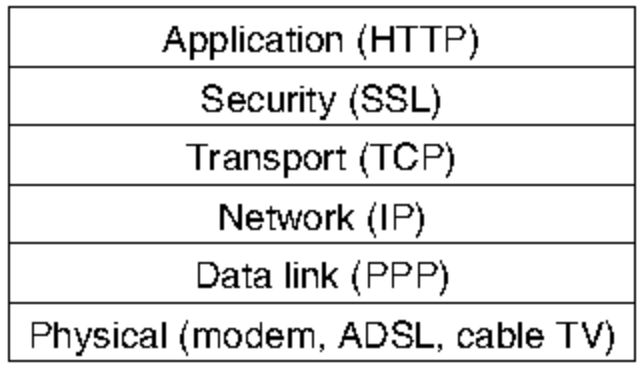


Figure 8-49. Layers (and protocols) for a home user browsing with SSL.

The SSL protocol has gone through several versions. Below we will discuss only version 3, which is the most widely used version. SSL supports a variety of different options. These options include the presence or absence of compression, the cryptographic algorithms to be used, and some matters relating to export restrictions on cryptography. The last is mainly intended to make sure that serious cryptography is used only when both ends of the connection are in the United States. In other cases, keys are limited to 40 bits, which cryptographers regard as something of a joke. Netscape was forced to put in this restriction in order to get an export license from the U.S. Government.

SSL consists of two subprotocols, one for establishing a secure connection and one for using it. Let us start out by seeing how secure connections are established. The connection establishment subprotocol is shown in Fig. 8-50. It starts out with message 1 when Alice sends a request to Bob to establish a connection. The request specifies the SSL version Alice has and her preferences with respect to compression and cryptographic algorithms. It also contains a nonce, R_A, to be used later.

Now it is Bob's turn. In message 2, Bob makes a choice among the various algorithms that Alice can support and sends his own nonce, R_B. Then, in message 3, he sends a certificate containing his public key. If this certificate is not signed by some well-known authority, he also sends a chain of certificates that can be followed back to one. All browsers, including Alice's, come preloaded with about 100 public keys, so if Bob can establish a chain anchored to one of these, Alice will be able to verify Bob's public key. At this point, Bob may send some other messages (such as a request for Alice's public-key certificate). When Bob is done, he sends message 4 to tell Alice it is her turn.

Alice responds by choosing a random 384-bit **premaster key** and sending it to Bob encrypted with his public key (message 5). The actual session key used for encrypting data is derived from the premaster key combined with both nonces in a complex way. After message 5 has been received, both Alice and Bob are able to compute the session key. For this reason, Alice tells Bob to switch to the

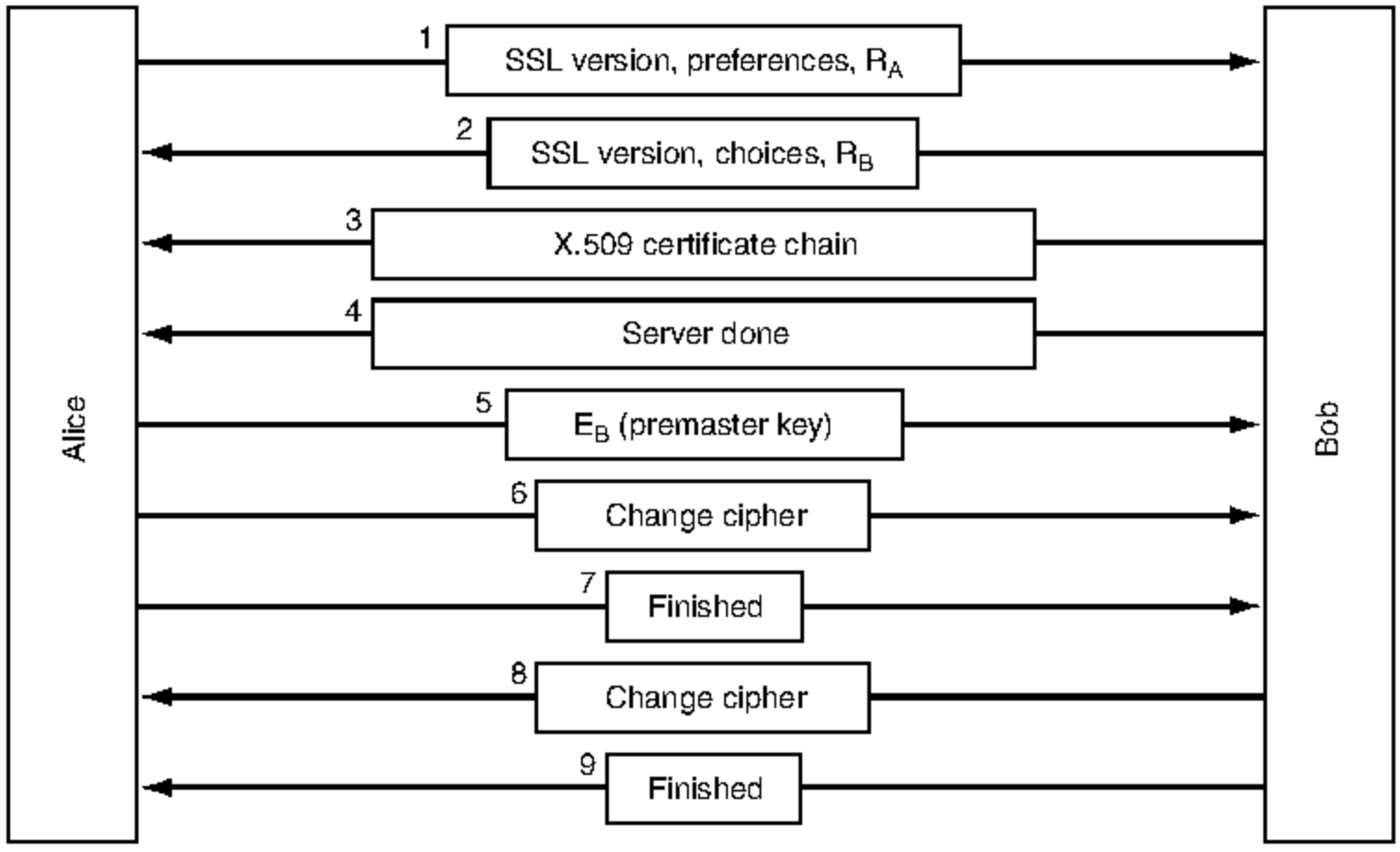


Figure 8-50. A simplified version of the SSL connection establishment subprotocol.

new cipher (message 6) and also that she is finished with the establishment subprotocol (message 7). Bob then acknowledges her (messages 8 and 9).

However, although Alice knows who Bob is, Bob does not know who Alice is (unless Alice has a public key and a corresponding certificate for it, an unlikely situation for an individual). Therefore, Bob's first message may well be a request for Alice to log in using a previously established login name and password. The login protocol, however, is outside the scope of SSL. Once it has been accomplished, by whatever means, data transport can begin.

As mentioned above, SSL supports multiple cryptographic algorithms. The strongest one uses triple DES with three separate keys for encryption and SHA-1 for message integrity. This combination is relatively slow, so it is mostly used for banking and other applications in which the highest security is required. For ordinary e-commerce applications, RC4 is used with a 128-bit key for encryption and MD5 is used for message authentication. RC4 takes the 128-bit key as a seed and expands it to a much larger number for internal use. Then it uses this internal number to generate a keystream. The keystream is XORed with the plaintext to provide a classical stream cipher, as we saw in Fig. 8-14. The export versions also use RC4 with 128-bit keys, but 88 of the bits are made public to make the cipher easy to break.

For actual transport, a second subprotocol is used, as shown in Fig. 8-51. Messages from the browser are first broken into units of up to 16 KB. If data

compression is enabled, each unit is then separately compressed. After that, a secret key derived from the two nonces and premaster key is concatenated with the compressed text and the result is hashed with the agreed-on hashing algorithm (usually MD5). This hash is appended to each fragment as the MAC. The compressed fragment plus MAC is then encrypted with the agreed-on symmetric encryption algorithm (usually by XORing it with the RC4 keystream). Finally, a fragment header is attached and the fragment is transmitted over the TCP connection.

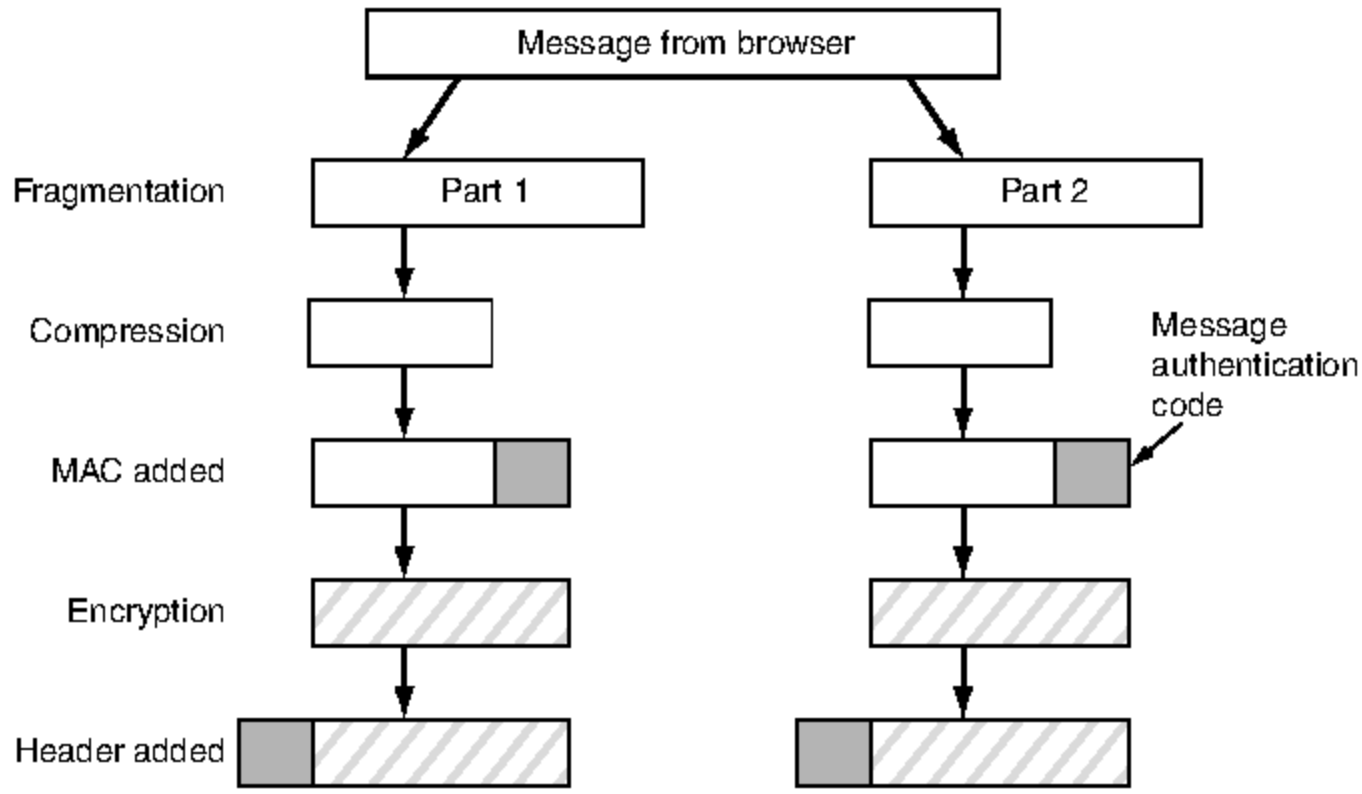


Figure 8-51. Data transmission using SSL.

A word of caution is in order, however. Since it has been shown that RC4 has some weak keys that can be easily cryptanalyzed, the security of SSL using RC4 is on shaky ground (Fluhrer et al., 2001). Browsers that allow the user to choose the cipher suite should be configured to use triple DES with 168-bit keys and SHA-1 all the time, even though this combination is slower than RC4 and MD5. Or, better yet, users should upgrade to browsers that support the successor to SSL that we describe shortly.

A problem with SSL is that the principals may not have certificates, and even if they do, they do not always verify that the keys being used match them.

In 1996, Netscape Communications Corp. turned SSL over to IETF for standardization. The result was **TLS** (**Transport Layer Security**). It is described in RFC 5246.

TLS was built on SSL version 3. The changes made to SSL were relatively small, but just enough that SSL version 3 and TLS cannot interoperate. For example, the way the session key is derived from the premaster key and nonces was

changed to make the key stronger (i.e., harder to cryptanalyze). Because of this incompatibility, most browsers implement both protocols, with TLS falling back to SSL during negotiation if necessary. This is referred to as SSL/TLS. The first TLS implementation appeared in 1999 with version 1.2 defined in August 2008. It includes support for stronger cipher suites (notably AES). SSL has remained strong in the marketplace although TLS will probably gradually replace it.

8.9.4 Mobile Code Security

Naming and connections are two areas of concern related to Web security. But there are more. In the early days, when Web pages were just static HTML files, they did not contain executable code. Now they often contain small programs, including Java applets, ActiveX controls, and JavaScripts. Downloading and executing such **mobile code** is obviously a massive security risk, so various methods have been devised to minimize it. We will now take a quick peek at some of the issues raised by mobile code and some approaches to dealing with it.

Java Applet Security

Java applets are small Java programs compiled to a stack-oriented machine language called **JVM** (**Java Virtual Machine**). They can be placed on a Web page for downloading along with the page. After the page is loaded, the applets are inserted into a JVM interpreter inside the browser, as illustrated in Fig. 8-52.

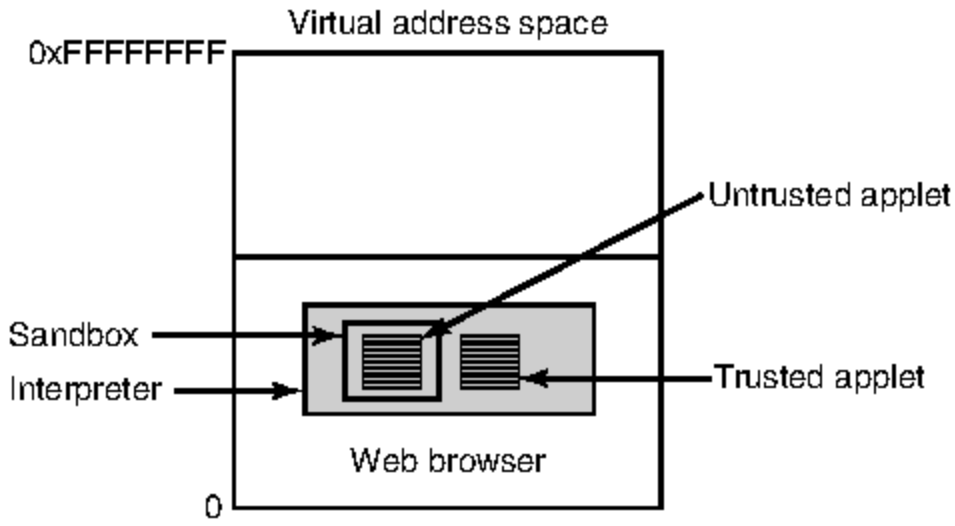


Figure 8-52. Applets can be interpreted by a Web browser.

The advantage of running interpreted code over compiled code is that every instruction is examined by the interpreter before being executed. This gives the interpreter the opportunity to check whether the instruction's address is valid. In addition, system calls are also caught and interpreted. How these calls are handled is a matter of the security policy. For example, if an applet is trusted (e.g., it

came from the local disk), its system calls could be carried out without question. However, if an applet is not trusted (e.g., it came in over the Internet), it could be encapsulated in what is called a **sandbox** to restrict its behavior and trap its attempts to use system resources.

When an applet tries to use a system resource, its call is passed to a security monitor for approval. The monitor examines the call in light of the local security policy and then makes a decision to allow or reject it. In this way, it is possible to give applets access to some resources but not all. Unfortunately, the reality is that the security model works badly and that bugs in it crop up all the time.

ActiveX

ActiveX controls are x86 binary programs that can be embedded in Web pages. When one of them is encountered, a check is made to see if it should be executed, and it if passes the test, it is executed. It is not interpreted or sandboxed in any way, so it has as much power as any other user program and can potentially do great harm. Thus, all the security is in the decision whether to run the ActiveX control. In retrospect, the whole idea is a gigantic security hole.

The method that Microsoft chose for making this decision is based on the idea of **code signing**. Each ActiveX control is accompanied by a digital signature—a hash of the code that is signed by its creator using public-key cryptography. When an ActiveX control shows up, the browser first verifies the signature to make sure it has not been tampered with in transit. If the signature is correct, the browser then checks its internal tables to see if the program's creator is trusted or there is a chain of trust back to a trusted creator. If the creator is trusted, the program is executed; otherwise, it is not. The Microsoft system for verifying ActiveX controls is called **Authenticode**.

It is useful to contrast the Java and ActiveX approaches. With the Java approach, no attempt is made to determine who wrote the applet. Instead, a run-time interpreter makes sure it does not do things the machine owner has said applets may not do. In contrast, with code signing, there is no attempt to monitor the mobile code's run-time behavior. If it came from a trusted source and has not been modified in transit, it just runs. No attempt is made to see whether the code is malicious or not. If the original programmer *intended* the code to format the hard disk and then erase the flash ROM so the computer can never again be booted, and if the programmer has been certified as trusted, the code will be run and destroy the computer (unless ActiveX controls have been disabled in the browser).

Many people feel that trusting an unknown software company is scary. To demonstrate the problem, a programmer in Seattle formed a software company and got it certified as trustworthy, which is easy to do. He then wrote an ActiveX control that did a clean shutdown of the machine and distributed his ActiveX control widely. It shut down many machines, but they could just be rebooted, so no

harm was done. He was just trying to expose the problem to the world. The official response was to revoke the certificate for this specific ActiveX control, which ended a short episode of acute embarrassment, but the underlying problem is still there for an evil programmer to exploit (Garfinkel with Spafford, 2002). Since there is no way to police the thousands of software companies that might write mobile code, the technique of code signing is a disaster waiting to happen.

JavaScript

JavaScript does not have any formal security model, but it does have a long history of leaky implementations. Each vendor handles security in a different way. For example, Netscape Navigator version 2 used something akin to the Java model, but by version 4 that had been abandoned for a code-signing model.

The fundamental problem is that letting foreign code run on your machine is asking for trouble. From a security standpoint, it is like inviting a burglar into your house and then trying to watch him carefully so he cannot escape from the kitchen into the living room. If something unexpected happens and you are distracted for a moment, bad things can happen. The tension here is that mobile code allows flashy graphics and fast interaction, and many Web site designers think that this is much more important than security, especially when it is somebody else's machine at risk.

Browser Extensions

As well as extending Web pages with code, there is a booming marketplace in **browser extensions**, **add-ons**, and **plug-ins**. They are computer programs that extend the functionality of Web browsers. Plug-ins often provide the capability to interpret or display a certain type of content, such as PDFs or Flash animations. Extensions and add-ons provide new browser features, such as better password management, or ways to interact with pages by, for example, marking them up or enabling easy shopping for related items.

Installing an extension, add-on, or plug-in is as simple as coming across something you want when browsing and following the link to install the program. This action will cause code to be downloaded across the Internet and installed into the browser. All of these programs are written to frameworks that differ depending on the browser that is being enhanced. However, to a first approximation, they become part of the trusted computing base of the browser. That is, if the code that is installed is buggy, the entire browser can be compromised.

There are two other obvious failure modes as well. The first is that the program may behave maliciously, for example, by gathering personal information and sending it to a remote server. For all the browser knows, the user installed the extension for precisely this purpose. The second problem is that plug-ins give the browser the ability to interpret new types of content. Often this content is a full

blown programming language itself. PDF and Flash are good examples. When users view pages with PDF and Flash content, the plug-ins in their browser are executing the PDF and Flash code. That code had better be safe; often there are vulnerabilities that it can exploit. For all of these reasons, add-ons and plug-ins should only be installed as needed and only from trusted vendors.

Viruses

Viruses are another form of mobile code. Only, unlike the examples above, viruses are not invited in at all. The difference between a virus and ordinary mobile code is that viruses are written to reproduce themselves. When a virus arrives, either via a Web page, an email attachment, or some other way, it usually starts out by infecting executable programs on the disk. When one of these programs is run, control is transferred to the virus, which usually tries to spread itself to other machines, for example, by emailing copies of itself to everyone in the victim's email address book. Some viruses infect the boot sector of the hard disk, so when the machine is booted, the virus gets to run. Viruses have become a huge problem on the Internet and have caused billions of dollars' worth of damage. There is no obvious solution. Perhaps a whole new generation of operating systems based on secure microkernels and tight compartmentalization of users, processes, and resources might help.

8.10 SOCIAL ISSUES

The Internet and its security technology is an area where social issues, public policy, and technology meet head on, often with huge consequences. Below we will just briefly examine three areas: privacy, freedom of speech, and copyright. Needless to say, we can only scratch the surface. For additional reading, see Anderson (2008a), Garfinkel with Spafford (2002), and Schneier (2004). The Internet is also full of material. Just type words such as "privacy," "censorship," and "copyright" into any search engine. Also, see this book's Web site for some links. It is at *http://www.pearsonhighered.com/tanenbaum*.

8.10.1 Privacy

Do people have a right to privacy? Good question. The Fourth Amendment to the U.S. Constitution prohibits the government from searching people's houses, papers, and effects without good reason, and goes on to restrict the circumstances under which search warrants shall be issued. Thus, privacy has been on the public agenda for over 200 years, at least in the U.S.

What has changed in the past decade is both the ease with which governments can spy on their citizens and the ease with which the citizens can prevent such

spying. In the 18th century, for the government to search a citizen's papers, it had to send out a policeman on a horse to go to the citizen's farm demanding to see certain documents. It was a cumbersome procedure. Nowadays, telephone companies and Internet providers readily provide wiretaps when presented with search warrants. It makes life much easier for the policeman and there is no danger of falling off a horse.

Cryptography changes all that. Anybody who goes to the trouble of downloading and installing PGP and who uses a well-guarded alien-strength key can be fairly sure that nobody in the known universe can read his email, search warrant or no search warrant. Governments well understand this and do not like it. Real privacy means it is much harder for them to spy on criminals of all stripes, but it is also much harder to spy on journalists and political opponents. Consequently, some governments restrict or forbid the use or export of cryptography. In France, for example, prior to 1999, all cryptography was banned unless the government was given the keys.

France was not alone. In April 1993, the U.S. Government announced its intention to make a hardware cryptoprocessor, the **clipper chip**, the standard for all networked communication. It was said that this would guarantee citizens' privacy. It also mentioned that the chip provided the government with the ability to decrypt all traffic via a scheme called **key escrow**, which allowed the government access to all the keys. However, the government promised only to snoop when it had a valid search warrant. Needless to say, a huge furor ensued, with privacy advocates denouncing the whole plan and law enforcement officials praising it. Eventually, the government backed down and dropped the idea.

A large amount of information about electronic privacy is available at the Electronic Frontier Foundation's Web site, *www.eff.org*.

Anonymous Remailers

PGP, SSL, and other technologies make it possible for two parties to establish secure, authenticated communication, free from third-party surveillance and interference. However, sometimes privacy is best served by *not* having authentication, in fact, by making communication anonymous. The anonymity may be desired for point-to-point messages, newsgroups, or both.

Let us consider some examples. First, political dissidents living under authoritarian regimes often wish to communicate anonymously to escape being jailed or killed. Second, wrongdoing in many corporate, educational, governmental, and other organizations has often been exposed by whistleblowers, who frequently prefer to remain anonymous to avoid retribution. Third, people with unpopular social, political, or religious views may wish to communicate with each other via email or newsgroups without exposing themselves. Fourth, people may wish to discuss alcoholism, mental illness, sexual harassment, child abuse, or being a

member of a persecuted minority in a newsgroup without having to go public. Numerous other examples exist, of course.

Let us consider a specific example. In the 1990s, some critics of a nontraditional religious group posted their views to a USENET newsgroup via an **anonymous remailer**. This server allowed users to create pseudonyms and send email to the server, which then remailed or re-posted them using the pseudonyms, so no one could tell where the messages really came from. Some postings revealed what the religious group claimed were trade secrets and copyrighted documents. The religious group responded by telling local authorities that its trade secrets had been disclosed and its copyright infringed, both of which were crimes where the server was located. A court case followed and the server operator was compelled to turn over the mapping information that revealed the true identities of the persons who had made the postings. (Incidentally, this was not the first time that a religious group was unhappy when someone leaked its trade secrets: William Tyndale was burned at the stake in 1536 for translating the Bible into English).

A substantial segment of the Internet community was completely outraged by this breach of confidentiality. The conclusion that everyone drew is that an anonymous remailer that stores a mapping between real email addresses and pseudonyms (now called a type 1 remailer) is not worth much. This case stimulated various people into designing anonymous remailers that could withstand subpoena attacks.

These new remailers, often called **cypherpunk remailers**, work as follows. The user produces an email message, complete with RFC 822 headers (except *From:*, of course), encrypts it with the remailer's public key, and sends it to the remailer. There the outer RFC 822 headers are stripped off, the content is decrypted and the message is remailed. The remailer has no accounts and maintains no logs, so even if the server is later confiscated, it retains no trace of messages that have passed through it.

Many users who wish anonymity chain their requests through multiple anonymous remailers, as shown in Fig. 8-53. Here, Alice wants to send Bob a really, really, really anonymous Valentine's Day card, so she uses three remailers. She composes the message, M, and puts a header on it containing Bob's email address. Then she encrypts the whole thing with remailer 3's public key, E_3 (indicated by horizontal hatching). To this she prepends a header with remailer 3's email address in plaintext. This is the message shown between remailers 2 and 3 in the figure.

Then she encrypts this message with remailer 2's public key, E_2 (indicated by vertical hatching) and prepends a plaintext header containing remailer 2's email address. This message is shown between 1 and 2 in Fig. 8-53. Finally, she encrypts the entire message with remailer 1's public key, E_1, and prepends a plaintext header with remailer 1's email address. This is the message shown to the right of Alice in the figure and this is the message she actually transmits.

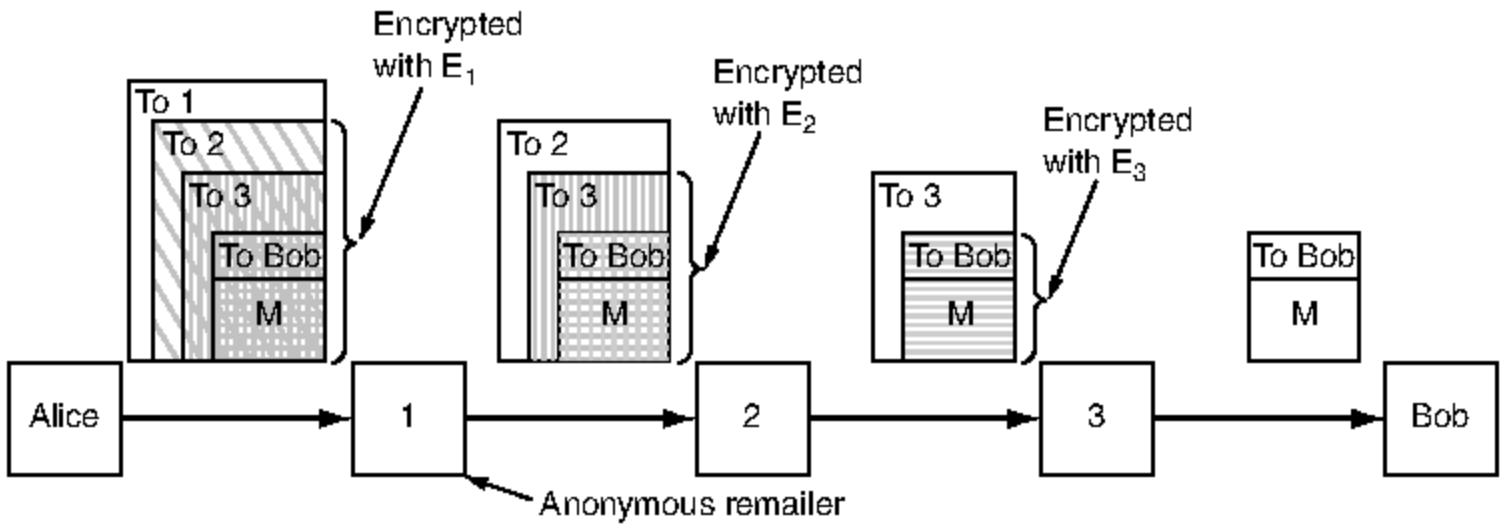


Figure 8-53. How Alice uses three remailers to send Bob a message.

When the message hits remailer 1, the outer header is stripped off. The body is decrypted and then emailed to remailer 2. Similar steps occur at the other two remailers.

Although it is extremely difficult for anyone to trace the final message back to Alice, many remailers take additional safety precautions. For example, they may hold messages for a random time, add or remove junk at the end of a message, and reorder messages, all to make it harder for anyone to tell which message output by a remailer corresponds to which input, in order to thwart traffic analysis. For a description of this kind of remailer, see Mazières and Kaashoek (1998).

Anonymity is not restricted to email. Services also exist that allow anonymous Web surfing using the same form of layered path in which one node only knows the next node in the chain. This method is called **onion routing** because each node peels off another layer of the onion to determine where to forward the packet next. The user configures his browser to use the anonymizer service as a proxy. Tor is a well-known example of such a system (Dingledine et al., 2004). Henceforth, all HTTP requests go through the anonymizer network, which requests the page and sends it back. The Web site sees an exit node of the anonymizer network as the source of the request, not the user. As long as the anonymizer network refrains from keeping a log, after the fact no one can determine who requested which page.

8.10.2 Freedom of Speech

Privacy relates to individuals wanting to restrict what other people can see about them. A second key social issue is freedom of speech, and its opposite, censorship, which is about governments wanting to restrict what individuals can read and publish. With the Web containing millions and millions of pages, it has become a censor's paradise. Depending on the nature and ideology of the regime, banned material may include Web sites containing any of the following:

1. Material inappropriate for children or teenagers.
2. Hate aimed at various ethnic, religious, sexual or other groups.
3. Information about democracy and democratic values.
4. Accounts of historical events contradicting the government's version.
5. Manuals for picking locks, building weapons, encrypting messages, etc.

The usual response is to ban the "bad" sites.

Sometimes the results are unexpected. For example, some public libraries have installed Web filters on their computers to make them child friendly by blocking pornography sites. The filters veto sites on their blacklists but also check pages for dirty words before displaying them. In one case in Loudoun County, Virginia, the filter blocked a patron's search for information on breast cancer because the filter saw the word "breast." The library patron sued Loudoun County. However, in Livermore, California, a parent sued the public library for *not* installing a filter after her 12-year-old son was caught viewing pornography there. What's a library to do?

It has escaped many people that the World Wide Web is a worldwide Web. It covers the whole world. Not all countries agree on what should be allowed on the Web. For example, in November 2000, a French court ordered Yahoo!, a California Corporation, to block French users from viewing auctions of Nazi memorabilia on Yahoo!'s Web site because owning such material violates French law. Yahoo! appealed to a U.S. court, which sided with it, but the issue of whose laws apply where is far from settled.

Just imagine. What would happen if some court in Utah instructed France to block Web sites dealing with wine because they do not comply with Utah's much stricter laws about alcohol? Suppose that China demanded that all Web sites dealing with democracy be banned as not in the interest of the State. Do Iranian laws on religion apply to more liberal Sweden? Can Saudi Arabia block Web sites dealing with women's rights? The whole issue is a veritable Pandora's box.

A relevant comment from John Gilmore is: "The net interprets censorship as damage and routes around it." For a concrete implementation, consider the **eternity service** (Anderson, 1996). Its goal is to make sure published information cannot be depublished or rewritten, as was common in the Soviet Union during Josef Stalin's reign. To use the eternity service, the user specifies how long the material is to be preserved, pays a fee proportional to its duration and size, and uploads it. Thereafter, no one can remove or edit it, not even the uploader.

How could such a service be implemented? The simplest model is to use a peer-to-peer system in which stored documents would be placed on dozens of participating servers, each of which gets a fraction of the fee, and thus an incentive to join the system. The servers should be spread over many legal jurisdictions for maximum resilience. Lists of 10 randomly selected servers would be stored

securely in multiple places, so that if some were compromised, others would still exist. An authority bent on destroying the document could never be sure it had found all copies. The system could also be made self-repairing in the sense that if it became known that some copies had been destroyed, the remaining sites would attempt to find new repositories to replace them.

The eternity service was the first proposal for a censorship-resistant system. Since then, others have been proposed and, in some cases, implemented. Various new features have been added, such as encryption, anonymity, and fault tolerance. Often the files to be stored are broken up into multiple fragments, with each fragment stored on many servers. Some of these systems are Freenet (Clarke et al., 2002), PASIS (Wylie et al., 2000), and Publius (Waldman et al., 2000). Other work is reported by Serjantov (2002).

Increasingly, many countries are trying to regulate the export of intangibles, which often include Web sites, software, scientific papers, email, telephone helpdesks, and more. Even the U.K., which has a centuries-long tradition of freedom of speech, is now seriously considering highly restrictive laws, that would, for example, define technical discussions between a British professor and his foreign Ph.D. student, both located at the University of Cambridge, as regulated export needing a government license (Anderson, 2002). Needless to say, many people consider such a policy to be outrageous.

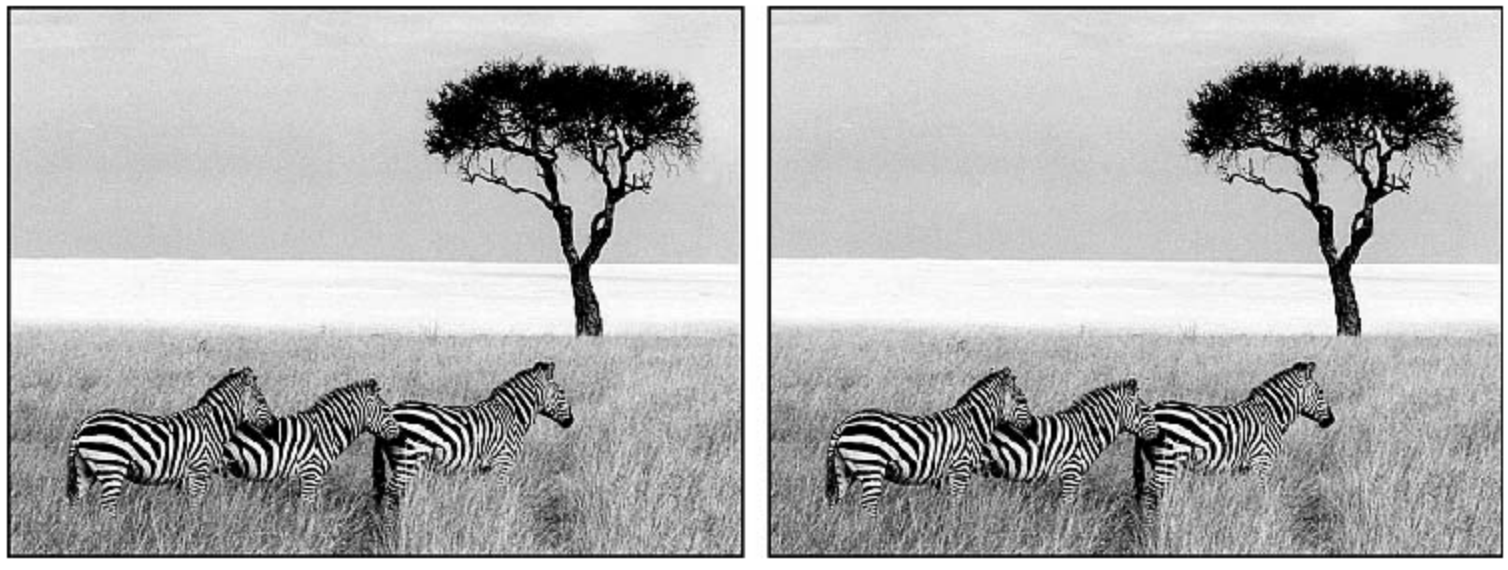
Steganography

In countries where censorship abounds, dissidents often try to use technology to evade it. Cryptography allows secret messages to be sent (although possibly not lawfully), but if the government thinks that Alice is a Bad Person, the mere fact that she is communicating with Bob may get him put in this category, too, as repressive governments understand the concept of transitive closure, even if they are short on mathematicians. Anonymous remailers can help, but if they are banned domestically and messages to foreign ones require a government export license, they cannot help much. But the Web can.

People who want to communicate secretly often try to hide the fact that any communication at all is taking place. The science of hiding messages is called **steganography**, from the Greek words for "covered writing." In fact, the ancient Greeks used it themselves. Herodotus wrote of a general who shaved the head of a messenger, tattooed a message on his scalp, and let the hair grow back before sending him off. Modern techniques are conceptually the same, only they have a higher bandwidth, lower latency, and do not require the services of a barber.

As a case in point, consider Fig. 8-54(a). This photograph, taken by one of the authors (AST) in Kenya, contains three zebras contemplating an acacia tree. Fig. 8-54(b) appears to be the same three zebras and acacia tree, but it has an extra added attraction. It contains the complete, unabridged text of five of

Shakespeare's plays embedded in it: *Hamlet*, *King Lear*, *Macbeth*, *The Merchant of Venice*, and *Julius Caesar*. Together, these plays total over 700 KB of text.



(a) (b)

Figure 8-54. (a) Three zebras and a tree. (b) Three zebras, a tree, and the complete text of five plays by William Shakespeare.

How does this steganographic channel work? The original color image is 1024×768 pixels. Each pixel consists of three 8-bit numbers, one each for the red, green, and blue intensity of that pixel. The pixel's color is formed by the linear superposition of the three colors. The steganographic encoding method uses the low-order bit of each RGB color value as a covert channel. Thus, each pixel has room for 3 bits of secret information, 1 in the red value, 1 in the green value, and 1 in the blue value. With an image of this size, up to $1024 \times 768 \times 3$ bits or 294,912 bytes of secret information can be stored in it.

The full text of the five plays and a short notice add up to 734,891 bytes. This text was first compressed to about 274 KB using a standard compression algorithm. The compressed output was then encrypted using IDEA and inserted into the low-order bits of each color value. As can be seen (or actually, cannot be seen), the existence of the information is completely invisible. It is equally invisible in the large, full-color version of the photo. The eye cannot easily distinguish 21-bit color from 24-bit color.

Viewing the two images in black and white with low resolution does not do justice to how powerful the technique is. To get a better feel for how steganography works, we have prepared a demonstration, including the full-color high-resolution image of Fig. 8-54(b) with the five plays embedded in it. The demonstration, including tools for inserting and extracting text into images, can be found at the book's Web site.

To use steganography for undetected communication, dissidents could create a Web site bursting with politically correct pictures, such as photographs of the Great Leader, local sports, movie, and television stars, etc. Of course, the pictures would be riddled with steganographic messages. If the messages were first

compressed and then encrypted, even someone who suspected their presence would have immense difficulty in distinguishing the messages from white noise. Of course, the images should be fresh scans; copying a picture from the Internet and changing some of the bits is a dead giveaway.

Images are by no means the only carrier for steganographic messages. Audio files also work fine. Hidden information can be carried in a voice-over-IP call by manipulating the packet delays, distorting the audio, or even in the header fields of packets (Lubacz et al., 2010). Even the layout and ordering of tags in an HTML file can carry information.

Although we have examined steganography in the context of free speech, it has numerous other uses. One common use is for the owners of images to encode secret messages in them stating their ownership rights. If such an image is stolen and placed on a Web site, the lawful owner can reveal the steganographic message in court to prove whose image it is. This technique is called **watermarking**. It is discussed in Piva et al. (2002).

For more on steganography, see Wayner (2008).

8.10.3 Copyright

Privacy and censorship are just two areas where technology meets public policy. A third one is the copyright law. **Copyright** is granting to the creators of **IP** (**Intellectual Property**), including writers, poets, artists, composers, musicians, photographers, cinematographers, choreographers, and others, the exclusive right to exploit their IP for some period of time, typically the life of the author plus 50 years or 75 years in the case of corporate ownership. After the copyright of a work expires, it passes into the public domain and anyone can use or sell it as they wish. The Gutenberg Project (*www.promo.net/pg*), for example, has placed thousands of public-domain works (e.g., by Shakespeare, Twain, and Dickens) on the Web. In 1998, the U.S. Congress extended copyright in the U.S. by another 20 years at the request of Hollywood, which claimed that without an extension nobody would create anything any more. By way of contrast, patents last for only 20 years and people still invent things.

Copyright came to the forefront when Napster, a music-swapping service, had 50 million members. Although Napster did not actually copy any music, the courts held that its holding a central database of who had which song was contributory infringement, that is, it was helping other people infringe. While nobody seriously claims copyright is a bad idea (although many claim that the term is far too long, favoring big corporations over the public), the next generation of music sharing is already raising major ethical issues.

For example, consider a peer-to-peer network in which people share legal files (public-domain music, home videos, religious tracts that are not trade secrets, etc.) and perhaps a few that are copyrighted. Assume that everyone is online all the time via ADSL or cable. Each machine has an index of what is on the hard

disk, plus a list of other members. Someone looking for a specific item can pick a random member and see if he has it. If not, he can check out all the members in that person's list, and all the members in their lists, and so on. Computers are very good at this kind of work. Having found the item, the requester just copies it.

If the work is copyrighted, chances are the requester is infringing (although for international transfers, the question of whose law applies matters because in some countries uploading is illegal but downloading is not). But what about the supplier? Is it a crime to keep music you have paid for and legally downloaded on your hard disk where others might find it? If you have an unlocked cabin in the country and an IP thief sneaks in carrying a notebook computer and scanner, scans a copyrighted book to the notebook's hard disk, and sneaks out, are *you* guilty of the crime of failing to protect someone else's copyright?

But there is more trouble brewing on the copyright front. There is a huge battle going on now between Hollywood and the computer industry. The former wants stringent protection of all intellectual property but the latter does not want to be Hollywood's policeman. In October 1998, Congress passed the **DMCA** (**Digital Millennium Copyright Act**), which makes it a crime to circumvent any protection mechanism present in a copyrighted work or to tell others how to circumvent it. Similar legislation has been enacted in the European Union. While virtually no one thinks that pirates in the Far East should be allowed to duplicate copyrighted works, many people think that the DMCA completely shifts the balance between the copyright owner's interest and the public interest.

A case in point: in September 2000, a music industry consortium charged with building an unbreakable system for selling music online sponsored a contest inviting people to try to break the system (which is precisely the right thing to do with any new security system). A team of security researchers from several universities, led by Prof. Edward Felten of Princeton, took up the challenge and broke the system. They then wrote a paper about their findings and submitted it to a USENIX security conference, where it underwent peer review and was accepted. Before the paper was to be presented, Felten received a letter from the Recording Industry Association of America that threatened to sue the authors under the DMCA if they published the paper.

Their response was to file a lawsuit asking a federal court to rule on whether publishing scientific papers on security research was still legal. Fearing a definitive court ruling against it, the industry withdrew its threat and the court dismissed Felten's suit. No doubt the industry was motivated by the weakness of its case: it had invited people to try to break its system and then threatened to sue some of them for accepting its own challenge. With the threat withdrawn, the paper was published (Craver et al., 2001). A new confrontation is virtually certain.

Meanwhile, pirated music and movies have fueled the massive growth of peer-to-peer networks. This has not pleased the copyright holders, who have used the DMCA to take action. There are now automated systems that search peer-to-peer networks and then fire off warnings to network operators and users who are

suspected of infringing copyright. In the United States, these warnings are known as **DMCA takedown notices**. This search is an arms' race because it is hard to reliably catch copyright infringers. Even your printer might be mistaken for a culprit (Piatek et al., 2008).

A related issue is the extent of the **fair use doctrine**, which has been established by court rulings in various countries. This doctrine says that purchasers of a copyrighted work have certain limited rights to copy the work, including the right to quote parts of it for scientific purposes, use it as teaching material in schools or colleges, and in some cases make backup copies for personal use in case the original medium fails. The tests for what constitutes fair use include (1) whether the use is commercial, (2) what percentage of the whole is being copied, and (3) the effect of the copying on sales of the work. Since the DMCA and similar laws within the European Union prohibit circumvention of copy protection schemes, these laws also prohibit legal fair use. In effect, the DMCA takes away historical rights from users to give content sellers more power. A major showdown is inevitable.

Another development in the works that dwarfs even the DMCA in its shifting of the balance between copyright owners and users is **trusted computing** as advocated by industry bodies such as the **TCG** (**Trusted Computing Group**), led by companies like Intel and Microsoft. The idea is to provide support for carefully monitoring user behavior in various ways (e.g., playing pirated music) at a level below the operating system in order to prohibit unwanted behavior. This is accomplished with a small chip, called a **TPM** (**Trusted Platform Module**), which it is difficult to tamper with. Most PCs sold nowadays come equipped with a TPM. The system allows software written by content owners to manipulate PCs in ways that users cannot change. This raises the question of who is trusted in trusted computing. Certainly, it is not the user. Needless to say, the social consequences of this scheme are immense. It is nice that the industry is finally paying attention to security, but it is lamentable that the driver is enforcing copyright law rather than dealing with viruses, crackers, intruders, and other security issues that most people are concerned about.

In short, the lawmakers and lawyers will be busy balancing the economic interests of copyright owners with the public interest for years to come. Cyberspace is no different from meatspace: it constantly pits one group against another, resulting in power struggles, litigation, and (hopefully) eventually some kind of resolution, at least until some new disruptive technology comes along.

8.11 SUMMARY

Cryptography is a tool that can be used to keep information confidential and to ensure its integrity and authenticity. All modern cryptographic systems are based on Kerckhoff's principle of having a publicly known algorithm and a secret

key. Many cryptographic algorithms use complex transformations involving substitutions and permutations to transform the plaintext into the ciphertext. However, if quantum cryptography can be made practical, the use of one-time pads may provide truly unbreakable cryptosystems.

Cryptographic algorithms can be divided into symmetric-key algorithms and public-key algorithms. Symmetric-key algorithms mangle the bits in a series of rounds parameterized by the key to turn the plaintext into the ciphertext. AES (Rijndael) and triple DES are the most popular symmetric-key algorithms at present. These algorithms can be used in electronic code book mode, cipher block chaining mode, stream cipher mode, counter mode, and others.

Public-key algorithms have the property that different keys are used for encryption and decryption and that the decryption key cannot be derived from the encryption key. These properties make it possible to publish the public key. The main public-key algorithm is RSA, which derives its strength from the fact that it is very difficult to factor large numbers.

Legal, commercial, and other documents need to be signed. Accordingly, various schemes have been devised for digital signatures, using both symmetric-key and public-key algorithms. Commonly, messages to be signed are hashed using algorithms such as SHA-1, and then the hashes are signed rather than the original messages.

Public-key management can be done using certificates, which are documents that bind a principal to a public key. Certificates are signed by a trusted authority or by someone (recursively) approved by a trusted authority. The root of the chain has to be obtained in advance, but browsers generally have many root certificates built into them.

These cryptographic tools can be used to secure network traffic. IPsec operates in the network layer, encrypting packet flows from host to host. Firewalls can screen traffic going into or out of an organization, often based on the protocol and port used. Virtual private networks can simulate an old leased-line network to provide certain desirable security properties. Finally, wireless networks need good security lest everyone read all the messages, and protocols like 802.11i provide it.

When two parties establish a session, they have to authenticate each other and, if need be, establish a shared session key. Various authentication protocols exist, including some that use a trusted third party, Diffie-Hellman, Kerberos, and public-key cryptography.

Email security can be achieved by a combination of the techniques we have studied in this chapter. PGP, for example, compresses messages, then encrypts them with a secret key and sends the secret key encrypted with the receiver's public key. In addition, it also hashes the message and sends the signed hash to verify message integrity.

Web security is also an important topic, starting with secure naming. DNSsec provides a way to prevent DNS spoofing. Most e-commerce Web sites use

SSL/TLS to establish secure, authenticated sessions between the client and server. Various techniques are used to deal with mobile code, especially sandboxing and code signing.

The Internet raises many issues in which technology interacts strongly with public policy. Some of the areas include privacy, freedom of speech, and copyright.

PROBLEMS

1. Break the following monoalphabetic substitution cipher. The plaintext, consisting of letters only, is an excerpt from a poem by Lewis Carroll.

   mvyy bek mnyx n yvjjyr snijrh invq n muvjvdt je n idnvy
   jurhri n fehfevir pyeir oruvdq ki ndq uri jhrnqvdt ed zb jnvy
   Irr uem rntrhyb jur yeoijrhi ndq jur jkhjyri nyy nqlndpr
   Jurb nhr mnvjvdt ed jur iuvdtyr mvyy bek pezr ndq wevd jur qndpr
   mvyy bek, medj bek, mvyy bek, medj bek, mvyy bek wevd jur qndpr
   mvyy bek, medj bek, mvyy bek, medj bek, medj bek wevd jur qndpr

2. An affine cipher is a version of a monoalphabetic substitution cipher, in which the letters of an alphabet of size m are first map to the integers in the range 0 to m-1. Subsequently, the integer representing each plaintext letter is transformed to an integer representing the corresponding cipher text letter. The encryption function for a single letter is $E(x) = (ax + b) \bmod m$, where m is the size of the alphabet and a and b are the key of the cipher, and are co-prime. Trudy finds out that Bob generated a ciphertext using an affine cipher. She gets a copy of the ciphertext, and finds out that the most frequent letter of the ciphertext is 'R', and the second most frequent letter of the ciphertext is 'K'. Show how Trudy can break the code and retrieve the plaintext.

3. Break the following columnar transposition cipher. The plaintext is taken from a popular computer textbook, so "computer" is a probable word. The plaintext consists entirely of letters (no spaces). The ciphertext is broken up into blocks of five characters for readability.

   aauan cvlre rurnn dltme aeepb ytust iceat npmey iicgo gorch srsoc
   nntii imiha oofpa gsivt tpsit lbolr otoex

4. Alice used a transposition cipher to encrypt her messages to Bob. For added security, she encrypted the transposition cipher key using a substitution cipher, and kept the encrypted cipher in her computer. Trudy managed to get hold of the encrypted transposition cipher key. Can Trudy decipher Alice's messages to Bob? Why or why not?

5. Find a 77-bit one-time pad that generates the text "Hello World" from the ciphertext of Fig. 8-4.

6. You are a spy, and, conveniently, have a library with an infinite number of books at your disposal. Your operator also has such a library at his disposal. You have agreed

to use *Lord of the Rings* as a one-time pad. Explain how you could use these assets to generate an infinitely long one-time pad.

7. Quantum cryptography requires having a photon gun that can, on demand, fire a single photon carrying 1 bit. In this problem, calculate how many photons a bit carries on a 250-Gbps fiber link. Assume that the length of a photon is equal to its wavelength, which for purposes of this problem, is 1 micron. The speed of light in fiber is 20 cm/nsec.

8. If Trudy captures and regenerates photons when quantum cryptography is in use, she will get some of them wrong and cause errors to appear in Bob's one-time pad. What fraction of Bob's one-time pad bits will be in error, on average?

9. A fundamental cryptographic principle states that all messages must have redundancy. But we also know that redundancy helps an intruder tell if a guessed key is correct. Consider two forms of redundancy. First, the initial n bits of the plaintext contain a known pattern. Second, the final n bits of the message contain a hash over the message. From a security point of view, are these two equivalent? Discuss your answer.

10. In Fig. 8-6, the P-boxes and S-boxes alternate. Although this arrangement is esthetically pleasing, is it any more secure than first having all the P-boxes and then all the S-boxes? Discuss your answer.

11. Design an attack on DES based on the knowledge that the plaintext consists exclusively of uppercase ASCII letters, plus space, comma, period, semicolon, carriage return, and line feed. Nothing is known about the plaintext parity bits.

12. In the text, we computed that a cipher-breaking machine with a million processors that could analyze a key in 1 nanosecond would take 10^{16} years to break the 128-bit version of AES. Let us compute how long it will take for this time to get down to 1 year, still along time, of course. To achieve this goal, we need computers to be 10^{16} times faster. If Moore's Law (computing power doubles every 18 months) continues to hold, how many years will it take before a parallel computer can get the cipher-breaking time down to a year?

13. AES supports a 256-bit key. How many keys does AES-256 have? See if you can find some number in physics, chemistry, or astronomy of about the same size. Use the Internet to help search for big numbers. Draw a conclusion from your research.

14. Suppose that a message has been encrypted using DES in counter mode. One bit of ciphertext in block C_i is accidentally transformed from a 0 to a 1 during transmission. How much plaintext will be garbled as a result?

15. Now consider ciphertext block chaining again. Instead of a single 0 bit being transformed into a 1 bit, an extra 0 bit is inserted into the ciphertext stream after block C_i. How much plaintext will be garbled as a result?

16. Compare cipher block chaining with cipher feedback mode in terms of the number of encryption operations needed to transmit a large file. Which one is more efficient and by how much?

17. Using the RSA public key cryptosystem, with $a = 1$, $b = 2 \cdots y = 25$, $z = 26$.
(a) If $p = 5$ and $q = 13$, list five legal values for d.

(b) If $p = 5$, $q = 31$, and $d = 37$, find e.
(c) Using $p = 3$, $q = 11$, and $d = 9$, find e and encrypt "hello".

18. Alice and Bob use RSA public key encryption in order to communicate between them. Trudy finds out that Alice and Bob shared one of the primes used to determine the number n of their public key pairs. In other words, Trudy found out that $n_a = p_a \times q$ and $n_b = p_b \times q$. How can Trudy use this information to break Alice's code?

19. Consider the use of counter mode, as shown in Fig. 8-15, but with $IV = 0$. Does the use of 0 threaten the security of the cipher in general?

20. In Fig. 8-20, we see how Alice can send Bob a signed message. If Trudy replaces P, Bob can detect it. But what happens if Trudy replaces both P and the signature?

21. Digital signatures have a potential weakness due to lazy users. In e-commerce transactions, a contract might be drawn up and the user asked to sign its SHA-1 hash. If the user does not actually verify that the contract and hash correspond, the user may inadvertently sign a different contract. Suppose that the Mafia try to exploit this weakness to make some money. They set up a pay Web site (e.g., pornography, gambling, etc.) and ask new customers for a credit card number. Then they send over a contract saying that the customer wishes to use their service and pay by credit card and ask the customer to sign it, knowing that most of them will just sign without verifying that the contract and hash agree. Show how the Mafia can buy diamonds from a legitimate Internet jeweler and charge them to unsuspecting customers.

22. A math class has 25 students. Assuming that all of the students were born in the first half of the year—between January 1st and June 30th— what is the probability that at least two students have the same birthday? Assume that nobody was born on leap day, so there are 181 possible birthdays.

23. After Ellen confessed to Marilyn about tricking her in the matter of Tom's tenure, Marilyn resolved to avoid this problem by dictating the contents of future messages into a dictating machine and having her new secretary just type them in. Marilyn then planned to examine the messages on her terminal after they had been typed in to make sure they contained her exact words. Can the new secretary still use the birthday attack to falsify a message, and if so, how? *Hint*: She can.

24. Consider the failed attempt of Alice to get Bob's public key in Fig. 8-23. Suppose that Bob and Alice already share a secret key, but Alice still wants Bob's public key. Is there now a way to get it securely? If so, how?

25. Alice wants to communicate with Bob, using public-key cryptography. She establishes a connection to someone she hopes is Bob. She asks him for his public key and he sends it to her in plaintext along with an X.509 certificate signed by the root CA. Alice already has the public key of the root CA. What steps does Alice carry out to verify that she is talking to Bob? Assume that Bob does not care who he is talking to (e.g., Bob is some kind of public service).

26. Suppose that a system uses PKI based on a tree-structured hierarchy of CAs. Alice wants to communicate with Bob, and receives a certificate from Bob signed by a CA X after establishing a communication channel with Bob. Suppose Alice has never heard of X. What steps does Alice take to verify that she is talking to Bob?

27. Can IPsec using AH be used in transport mode if one of the machines is behind a NAT box? Explain your answer.

28. Alice wants to send a message to Bob using SHA-1 hashes. She consults with you regarding the appropriate signature algorithm to be used. What would you suggest?

29. Give one reason why a firewall might be configured to inspect incoming traffic. Give one reason why it might be configured to inspect outgoing traffic. Do you think the inspections are likely to be successful?

30. Suppose an organization uses VPN to securely connect its sites over the Internet. Jim, a user in the organization, uses the VPN to communicate with his boss, Mary. Describe one type of communication between Jim and Mary which would not require use of encryption or other security mechanism, and another type of communication which would require encryption or other security mechanisms. Explain your answer.

31. Change one message in the protocol of Fig. 8-34 in a minor way to make it resistant to the reflection attack. Explain why your change works.

32. The Diffie-Hellman key exchange is being used to establish a secret key between Alice and Bob. Alice sends Bob (227, 5, 82). Bob responds with (125). Alice's secret number, x, is 12, and Bob's secret number, y, is 3. Show how Alice and Bob compute the secret key.

33. Two users can establish a shared secret key using the Diffie-Hellman algorithm, even if they have never met, share no secrets, and have no certificates

(a) Explain how this algorithm is susceptible to a man-in-the-middle attack.

(b) How would this susceptibility change if n or g were secret?

34. In the protocol of Fig. 8-39, why is A sent in plaintext along with the encrypted session key?

35. In the Needham-Schroeder protocol, Alice generates two challenges, R_A and R_{A2}. This seems like overkill. Would one not have done the job?

36. Suppose an organization uses Kerberos for authentication. In terms of security and service availability, what is the effect if AS or TGS goes down?

37. Alice is using the public-key authentication protocol of Fig. 8-43 to authenticate communication with Bob. However, when sending message 7, Alice forgot to encrypt R_B. Trudy now knows the value of R_B. Do Alice and Bob need to repeat the authentication procedure with new parameters in order to ensure secure communication? Explain your answer.

38. In the public-key authentication protocol of Fig. 8-43, in message 7, R_B is encrypted with K_S. Is this encryption necessary, or would it have been adequate to send it back in plaintext? Explain your answer.

39. Point-of-sale terminals that use magnetic-stripe cards and PIN codes have a fatal flaw: a malicious merchant can modify his card reader to log all the information on the card and the PIN code in order to post additional (fake) transactions in the future. Next generation terminals will use cards with a complete CPU, keyboard, and tiny display on the card. Devise a protocol for this system that malicious merchants cannot break.

40. Is it possible to multicast a PGP message? What restrictions would apply?

41. Assuming that everyone on the Internet used PGP, could a PGP message be sent to an arbitrary Internet address and be decoded correctly by all concerned? Discuss your answer.

42. The attack shown in Fig. 8-47 leaves out one step. The step is not needed for the spoof to work, but including it might reduce potential suspicion after the fact. What is the missing step?

43. The SSL data transport protocol involves two nonces as well as a premaster key. What value, if any, does using the nonces have?

44. Consider an image of 2048×512 pixels. You want to encrypt a file sized 2.5 MB. What fraction of the file can you encrypt in this image? What fraction would you be able to encrypt if you compressed the file to a quarter of its original size? Show your calculations.

45. The image of Fig. 8-54(b) contains the ASCII text of five plays by Shakespeare. Would it be possible to hide music among the zebras instead of text? If so, how would it work and how much could you hide in this picture? If not, why not?

46. You are given a text file of size 60 MB, which is to be encrypted using steganography in the low-order bits of each color in an image file. What size image would be required in order to encrypt the entire file? What size would be needed if the file were first compressed to a third of its original size? Give your answer in pixels, and show your calculations. Assume that the images have an aspect ratio of 3:2, for example, 3000×2000 pixels.

47. Alice was a heavy user of a type 1 anonymous remailer. She would post many messages to her favorite newsgroup, *alt.fanclub.alice*, and everyone would know they all came from Alice because they all bore the same pseudonym. Assuming that the remailer worked correctly, Trudy could not impersonate Alice. After type 1 remailers were all shut down, Alice switched to a cypherpunk remailer and started a new thread in her newsgroup. Devise a way for her to prevent Trudy from posting new messages to the newsgroup, impersonating Alice.

48. Search the Internet for an interesting case involving privacy and write a one-page report on it.

49. Search the Internet for some court case involving copyright versus fair use and write a 1-page report summarizing your findings.

50. Write a program that encrypts its input by XORing it with a keystream. Find or write as good a random number generator as you can to generate the keystream. The program should act as a filter, taking plaintext on standard input and producing ciphertext on standard output (and vice versa). The program should take one parameter, the key that seeds the random number generator.

51. Write a procedure that computes the SHA-1 hash of a block of data. The procedure should have two parameters: a pointer to the input buffer and a pointer to a 20-byte output buffer. To see the exact specification of SHA-1, search the Internet for FIPS 180-1, which is the full specification.

52. Write a function that accepts a stream of ASCII characters and encrypts this input using a substitution cipher with the Cipher Block Chaining mode. The block size should be 8 bytes. The program should take plaintext from the standard input and print the ciphertext on the standard output. For this problem, you are allowed to select any reasonable system to determine that the end of the input is reached, and/or when padding should be applied to complete the block. You may select any output format, as long as it is unambiguous. The program should receive two parameters:

1. A pointer to the initializing vector; and
2. A number, k, representing the substitution cipher shift, such that each ASCII character would be encrypted by the kth character ahead of it in the alphabet.

For example, if $x = 3$, then A is encoded by D, B is encoded by E etc. Make reasonable assumptions with respect to reaching the last character in the ASCII set. Make sure to document clearly in your code any assumptions you make about the input and encryption algorithm.

53. The purpose of this problem is to give you a better understanding as to the mechanisms of RSA. Write a function that receives as its parameters primes p and q, calculates public and private RSA keys using these parameters, and outputs n, z, d and e as printouts to the standard output. The function should also accept a stream of ASCII characters and encrypt this input using the calculated RSA keys. The program should take plaintext from the standard input and print the ciphertext to the standard output. The encryption should be carried out character-wise, that is, take each character in the input and encrypt it independently of other characters in the input. For this problem, you are allowed to select any reasonable system to determine that the end of the input is reached. You may select any output format, as long as it is unambiguous. Make sure to document clearly in your code any assumptions you make about the input and encryption algorithm.

9

READING LIST AND BIBLIOGRAPHY

We have now finished our study of computer networks, but this is only the beginning. Many interesting topics have not been treated in as much detail as they deserve, and others have been omitted altogether for lack of space. In this chapter, we provide some suggestions for further reading and a bibliography, for the benefit of readers who wish to continue their study of computer networks.

9.1 SUGGESTIONS FOR FURTHER READING

There is an extensive literature on all aspects of computer networks. Two journals that publish papers in this area are *IEEE/ACM Transactions on Networking* and *IEEE Journal on Selected Areas in Communications*.

The periodicals of the ACM Special Interest Groups on Data Communications (SIGCOMM) and Mobility of Systems, Users, Data, and Computing (SIGMOBILE) publish many papers of interest, especially on emerging topics. They are *Computer Communication Review* and *Mobile Computing and Communications Review*.

IEEE also publishes three magazines—*IEEE Internet Computing*, *IEEE Network Magazine*, and *IEEE Communications Magazine*—that contain surveys, tutorials, and case studies on networking. The first two emphasize architecture, standards, and software, and the last tends toward communications technology (fiber optics, satellites, and so on).

There are a number of annual or biannual conferences that attract numerous papers on networks. In particular, look for the *SIGCOMM* conference, *NSDI* (Symposium on Networked Systems Design and Implementation), *MobiSys* (Conference on Mobile Systems, Applications, and Services), *SOSP* (Symposium on Operating Systems Principles) and *OSDI* (Symposium on Operating Systems Design and Implementation).

Below we list some suggestions for supplementary reading, keyed to the chapters of this book. Many of the suggestions are books of chapters in books, with some tutorials and surveys. Full references are in Sec. 9.2.

9.1.1 Introduction and General Works

Comer, *The Internet Book*, 4th ed.

Anyone looking for an easygoing introduction to the Internet should look here. Comer describes the history, growth, technology, protocols, and services of the Internet in terms that novices can understand, but so much material is covered that the book is also of interest to more technical readers.

Computer Communication Review, 25th Anniversary Issue, Jan. 1995

For a firsthand look at how the Internet developed, this special issue collects important papers up to 1995. Included are papers that show the development of TCP, multicast, the DNS, Ethernet, and the overall architecture.

Crovella and Krishnamurthy, *Internet Measurement*

How do we know how well the Internet works anyway? This question is not trivial to answer because no one is in charge of the Internet. This book describes the techniques that have been developed to measure the operation of the Internet, from network infrastructure to applications.

IEEE Internet Computing, Jan.–Feb. 2000

The first issue of *IEEE Internet Computing* in the new millennium did exactly what you would expect: it asked the people who helped create the Internet in the previous millennium to speculate on where it is going in the next one. The experts are Paul Baran, Lawrence Roberts, Leonard Kleinrock, Stephen Crocker, Danny Cohen, Bob Metcalfe, Bill Gates, Bill Joy, and others. See how well their predictions have fared over a decade later.

Kipnis, "Beating the System: Abuses of the Standards Adoption Process"

Standards committees try to be fair and vendor neutral in their work, but unfortunately there are companies that try to abuse the system. For example, it has happened repeatedly that a company helps develop a standard and then after it is approved, announces that the standard is based on a patent it owns and which it will license to companies that it likes and not to companies that it does not like, at

prices that it alone determines. For a look at the dark side of standardization, this article is an excellent start.

Hafner and Lyon, *Where Wizards Stay Up Late*
Naughton, *A Brief History of the Future*

Who invented the Internet, anyway? Many people have claimed credit. And rightly so, since many people had a hand in it, in different ways. There was Paul Baran, who wrote a report describing packet switching, there were the people at various universities who designed the ARPANET architecture, there were the people at BBN who programmed the first IMPs, there were Bob Kahn and Vint Cerf who invented TCP/IP, and so on. These books tell the story of the Internet, at least up to 2000, replete with many anecdotes.

9.1.2 The Physical Layer

Bellamy, *Digital Telephony*, 3rd ed.

For a look back at that other important network, the telephone network, this authoritative book contains everything you ever wanted to know and more. Particularly interesting are the chapters on transmission and multiplexing, digital switching, fiber optics, mobile telephony, and DSL.

Hu and Li, "Satellite-Based Internet: A Tutorial"

Internet access via satellite is different from using terrestrial lines. Not only is there the issue of delay, but routing and switching are also different. In this paper, the authors examine the issues related to using satellites for Internet access.

Joel, "Telecommunications and the IEEE Communications Society"

For a compact but surprisingly comprehensive history of telecommunications, starting with the telegraph and ending with 802.11, this article is the place to look. It also covers radio, telephones, analog and digital switching, submarine cables, digital transmission, television broadcasting, satellites, cable TV, optical communications, mobile phones, packet switching, the ARPANET, and the Internet.

Palais, *Fiber Optic Communication*, 5th ed.

Books on fiber optic technology tend to be aimed at the specialist, but this one is more accessible than most. It covers waveguides, light sources, light detectors, couplers, modulation, noise, and many other topics.

Su, *The UMTS Air Interface in RF Engineering*

This book provides a detailed overview of one of the main 3G cellular systems. It is focused on the air interface, or wireless protocols that are used between mobiles and the network infrastructure.

Want, *RFID Explained*

Want's book is an easy-to-read primer on how the unusual technology of the RFID physical layer works. It covers all aspects of RFID, including its potential applications. Some real-world examples of RFID deployments and the experience gained from them is also convered.

9.1.3 The Data Link Layer

Kasim, *Delivering Carrier Ethernet*

Nowadays, Ethernet is not only a local-area technology. The new fashion is to use Ethernet as a long-distance link for carrier-grade Ethernet. This book brings together essays to cover the topic in depth.

Lin and Costello, *Error Control Coding*, 2nd ed.

Codes to detect and correct errors are central to reliable computer networks. This popular textbook explains some of the most important codes, from simple linear Hamming codes to more complex low-density parity check codes. It tries to do so with the minimum algebra necessary, but that is still a lot.

Stallings, *Data and Computer Communications*, 9th ed.

Part two covers digital data transmission and a variety of links, including error detection, error control with retransmissions, and flow control.

9.1.4 The Medium Access Control Sublayer

Andrews et al., *Fundamentals of WiMAX*

This comprehensive book gives a definitive treatment of WiMAX technology, from the idea of broadband wireless, to the wireless techniques using OFDM and multiple antennas, through the multi-access system. Its tutorial style gives about the most accessible treatment you will find for this heavy material.

Gast, *802.11 Wireless Networks*, 2nd ed.

For a readable introduction to the technology and protocols of 802.11, this is a good place to start. It begins with the MAC sublayer, then introduces material on the different physical layers and also security. However, the second edition is not new enough to have much to say about 802.11n.

Perlman, *Interconnections*, 2nd ed.

For an authoritative but entertaining treatment of bridges, routers, and routing in general, Perlman's book is the place to look. The author designed the algorithms used in the IEEE 802 spanning tree bridge and she is one of the world's leading authorities on various aspects of networking.

9.1.5 The Network Layer

Comer, *Internetworking with TCP/IP*, Vol. 1, 5th ed.

Comer has written the definitive work on the TCP/IP protocol suite, now in its fifth edition. Most of the first half deals with IP and related protocols in the network layer. The other chapters deal primarily with the higher layers and are also worth reading.

Grayson et al., *IP Design for Mobile Networks*

Traditional telephone networks and the Internet are on a collision course, with mobile phone networks being implemented with IP on the inside. This book tells how to design a network using the IP protocols that supports mobile telephone service.

Huitema, *Routing in the Internet*, 2nd ed.

If you want to gain a deep understanding of routing protocols, this is a very good book. Both pronounceable algorithms (e.g., RIP, and CIDR) and unpronounceable algorithms (e.g., OSPF, IGRP, and BGP) are treated in great detail. Newer developments are not covered since this is an older book, but what is covered is explained very well.

Koodli and Perkins, *Mobile Inter-networking with IPv6*

Two important network layer developments are presented in one volume: IPv6 and Mobile IP. Both topics are covered well, and Perkins was one of the driving forces behind Mobile IP.

Nucci and Papagiannaki, *Design, Measurement and Management of Large-Scale IP Networks*

We talked a great deal about how networks work, but not how you would design, deploy and manage one if you were an ISP. This book fills that gap, looking at modern methods for traffic engineering and how ISPs provide services using networks.

Perlman, *Interconnections*, 2nd ed.

In Chaps. 12 through 15, Perlman describes many of the issues involved in unicast and multicast routing algorithm design, both for wide area networks and networks of LANs. But by far, the best part of the book is Chap. 18, in which the author distills her many years of experience with network protocols into an informative and fun chapter. It is required reading for protocol designers.

Stevens, *TCP/IP Illustrated*, Vol. 1

Chapters 3–10 provide a comprehensive treatment of IP and related protocols (ARP, RARP, and ICMP), illustrated by examples.

Varghese, *Network Algorithmics*

We have spent much time talking about how routers and other network elements interact with each other. This book is different: it is about how routers are actually designed to forward packets at prodigious speeds. For the inside scoop on that and related questions, this is the book to read. The author is an authority on clever algorithms that are used in practice to implement high-speed network elements in software and hardware.

9.1.6 The Transport Layer

Comer, *Internetworking with TCP/IP*, Vol. 1, 5th ed.

As mentioned above, Comer has written the definitive work on the TCP/IP protocol suite. The second half of the book is about UDP and TCP.

Farrell and Cahill, *Delay- and Disruption-Tolerant Networking*

This short book is the one to read for a deeper look at the architecture, protocols, and applications of "challenged networks" that must operate under harsh conditions of connectivity. The authors have participated in the development of DTNs in the IETF DTN Research Group.

Stevens, *TCP/IP Illustrated*, Vol. 1

Chapters 17–24 provide a comprehensive treatment of TCP illustrated by examples.

9.1.7 The Application Layer

Berners-Lee et al., "The World Wide Web"

Take a trip back in time for a perspective on the Web and where it is going by the person who invented it and some of his colleagues at CERN. The article focuses on the Web architecture, URLs, HTTP, and HTML, as well as future directions, and compares it to other distributed information systems.

Held, *A Practical Guide to Content Delivery Networks*, 2nd ed.

This book gives a down-to-earth exposition of how CDNs work, emphasizing the practical considerations in designing and operating a CDN that performs well.

Hunter et al., *Beginning XML*, 4th ed.

There are many, many books on HTML, XML and Web services. This 1000-page book covers most of what you are likely to want to know. It explains not only how to write XML and XHTML, but also how to develop Web services that produce and manipulate XML using Ajax, SOAP, and other techniques that are commonly used in practice.

Krishnamurthy and Rexford, *Web Protocols and Practice*

It would be hard to find a more comprehensive book about all aspects of the Web than this one. It covers clients, servers, proxies, and caching, as you might expect. But there are also chapters on Web traffic and measurements as well as chapters on current research and improving the Web.

Simpson, *Video Over IP*, 2nd ed.

The author takes a broad look at how IP technology can be used to move video across networks, both on the Internet and in private networks designed to carry video. Interestingly, this book is oriented for the video professional learning about networking, rather than the other way around.

Wittenburg, *Understanding Voice Over IP Technology*

This book covers how voice over IP works, from carrying audio data with the IP protocols and quality-of-service issues, through to the SIP and H.323 suite of protocols. It is necessarily detailed given the material, but accessible and broken up into digestible units.

9.1.8 Network Security

Anderson, *Security Engineering*, 2nd. ed.

This book presents a wonderful mix of security techniques couched in an understanding of how people use (and misuse) them. It is more technical than *Secrets and Lies*, but less technical than *Network Security* (see below). After an introduction to the basic security techniques, entire chapters are devoted to various applications, including banking, nuclear command and control, security printing, biometrics, physical security, electronic warfare, telecom security, e-commerce, and copyright protection.

Ferguson et al., *Cryptography Engineering*

Many books tell you how the popular cryptographic algorithms work. This book tells you how to use cryptography—why cryptographic protocols are designed the way they are and how to put them together into a system that will meet your security goals. It is a fairly compact book that is essential reading for anyone designing systems that depend on cryptography.

Fridrich, *Steganography in Digital Media*

Steganography goes back to ancient Greece, where the wax was melted off blank tablets so secret messages could be applied to the underlying wood before the wax was reapplied. Nowadays, videos, audio, and other content on the Internet provide different carriers for secret messages. Various modern techniques for hiding and finding information in images are discussed here.

Kaufman et al., *Network Security*, 2nd ed.

This authoritative and witty book is the first place to look for more technical information on network security algorithms and protocols. Secret and public key algorithms and protocols, message hashes, authentication, Kerberos, PKI, IPsec, SSL/TLS, and email security are all explained carefully and at considerable length, with many examples. Chapter 26, on security folklore, is a real gem. In security, the devil is in the details. Anyone planning to design a security system that will actually be used will learn a lot from the real-world advice in this chapter.

Schneier, *Secrets and Lies*

If you read *Cryptography Engineering* from cover to cover, you will know everything there is to know about cryptographic algorithms. If you then read *Secrets and Lies* cover to cover (which can be done in a lot less time), you will learn that cryptographic algorithms are not the whole story. Most security weaknesses are not due to faulty algorithms or even keys that are too short, but to flaws in the security environment. For a nontechnical and fascinating discussion of computer security in the broadest sense, this book is a very good read.

Skoudis and Liston, *Counter Hack Reloaded*, 2nd ed.

The best way to stop a hacker is to think like a hacker. This book shows how hackers see a network, and argues that security should be a function of the entire network's design, not an afterthought based on one specific technology. It covers almost all common attacks, including the "social engineering" types that take advantage of users who are not always familiar with computer security measures.

9.2 ALPHABETICAL BIBLIOGRAPHY

ABRAMSON, N.: "Internet Access Using VSATs," *IEEE Commun. Magazine*, vol. 38, pp. 60–68, July 2000.

AHMADI, S.: "An Overview of Next-Generation Mobile WiMAX Technology," *IEEE Commun. Magazine*, vol. 47, pp. 84–88, June 2009.

ALLMAN, M., and PAXSON, V.: "On Estimating End-to-End Network Path Properties," *Proc. SIGCOMM '99 Conf.*, ACM, pp. 263–274, 1999.

ANDERSON, C.: *The Long Tail: Why the Future of Business is Selling Less of More*, rev. upd. ed., New York: Hyperion, 2008a.

ANDERSON, R.J.: *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd ed., New York: John Wiley & Sons, 2008b.

ANDERSON, R.J.: "Free Speech Online and Offline," *IEEE Computer*, vol. 25, pp. 28–30, June 2002.

ANDERSON, R.J.: "The Eternity Service," *Proc. Pragocrypt Conf.*, CTU Publishing House, pp. 242–252, 1996.

ANDREWS, J., GHOSH, A., and MUHAMED, R.: *Fundamentals of WiMAX: Understanding Broadband Wireless Networking*, Upper Saddle River, NJ: Pearson Education, 2007.

ASTELY, D., DAHLMAN, E., FURUSKAR, A., JADING, Y., LINDSTROM, M., and PARKVALL, S.: "LTE: The Evolution of Mobile Broadband," *IEEE Commun. Magazine*, vol. 47, pp. 44–51, Apr. 2009.

BALLARDIE, T., FRANCIS, P., and CROWCROFT, J.: "Core Based Trees (CBT)," *Proc. SIGCOMM '93 Conf.*, ACM, pp. 85–95, 1993.

BARAN, P.: "On Distributed Communications: I. Introduction to Distributed Communication Networks," *Memorandum RM-420-PR*, Rand Corporation, Aug. 1964.

BELLAMY, J.: *Digital Telephony*, 3rd ed., New York: John Wiley & Sons, 2000.

BELLMAN, R.E.: *Dynamic Programming*, Princeton, NJ: Princeton University Press, 1957.

BELLOVIN, S.: "The Security Flag in the IPv4 Header," RFC 3514, Apr. 2003.

BELSNES, D.: "Flow Control in the Packet Switching Networks," *Communications Networks*, Uxbridge, England: Online, pp. 349–361, 1975.

BENNET, C.H., and BRASSARD, G.: "Quantum Cryptography: Public Key Distribution and Coin Tossing," *Int'l Conf. on Computer Systems and Signal Processing*, pp. 175–179, 1984.

BERESFORD, A., and STAJANO, F.: "Location Privacy in Pervasive Computing," *IEEE Pervasive Computing*, vol. 2, pp. 46–55, Jan. 2003.

BERGHEL, H.L.: "Cyber Privacy in the New Millennium," *IEEE Computer*, vol. 34, pp. 132–134, Jan. 2001.

BERNERS-LEE, T., CAILLIAU, A., LOUTONEN, A., NIELSEN, H.F., and SECRET, A.: "The World Wide Web," *Commun. of the ACM*, vol. 37, pp. 76–82, Aug. 1994.

BERTSEKAS, D., and GALLAGER, R.: *Data Networks*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1992.

BHATTI, S.N., and CROWCROFT, J.: "QoS Sensitive Flows: Issues in IP Packet Handling," *IEEE Internet Computing*, vol. 4, pp. 48–57, July–Aug. 2000.

BIHAM, E., and SHAMIR, A.: "Differential Fault Analysis of Secret Key Cryptosystems," *Proc. 17th Ann. Int'l Cryptology Conf.*, Berlin: Springer-Verlag LNCS 1294, pp. 513–525, 1997.

BIRD, R., GOPAL, I., HERZBERG, A., JANSON, P.A., KUTTEN, S., MOLVA, R., and YUNG, M.: "Systematic Design of a Family of Attack-Resistant Authentication Protocols," *IEEE J. on Selected Areas in Commun.*, vol. 11, pp. 679–693, June 1993.

BIRRELL, A.D., and NELSON, B.J.: "Implementing Remote Procedure Calls," *ACM Trans. on Computer Systems*, vol. 2, pp. 39–59, Feb. 1984.

BIRYUKOV, A., SHAMIR, A., and WAGNER, D.: "Real Time Cryptanalysis of A5/1 on a PC," *Proc. Seventh Int'l Workshop on Fast Software Encryption*, Berlin: Springer-Verlag LNCS 1978, pp. 1–8, 2000.

BLAZE, M., and BELLOVIN, S.: "Tapping on My Network Door," *Commun. of the ACM*, vol. 43, p. 136, Oct. 2000.

BOGGS, D., MOGUL, J., and KENT, C.: "Measured Capacity of an Ethernet: Myths and Reality," *Proc. SIGCOMM '88 Conf.*, ACM, pp. 222–234, 1988.

BORISOV, N., GOLDBERG, I., and WAGNER, D.: "Intercepting Mobile Communications: The Insecurity of 802.11," *Seventh Int'l Conf. on Mobile Computing and Networking*, ACM, pp. 180–188, 2001.

BRADEN, R.: "Requirements for Internet Hosts—Communication Layers," RFC 1122, Oct. 1989.

BRADEN, R., BORMAN, D., and PARTRIDGE, C.: "Computing the Internet Checksum," RFC 1071, Sept. 1988.

BRANDENBURG, K.: "MP3 and AAC Explained," *Proc. 17th Intl. Conf.: High-Quality Audio Coding*, Audio Engineering Society, pp. 99–110, Aug. 1999.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C., MALER, E., YERGEAU, F., and COWAN, J.: "Extensible Markup Language (XML) 1.1 (Second Edition)," W3C Recommendation, Sept. 2006.

BRESLAU, L., CAO, P., FAN, L., PHILLIPS, G., and SHENKER, S.: "Web Caching and Zipf-like Distributions: Evidence and Implications," *Proc. INFOCOM Conf.*, IEEE, pp. 126–134, 1999.

BURLEIGH, S., HOOKE, A., TORGERSON, L., FALL, K., CERF, V., DURST, B., SCOTT, K., and WEISS, H.: "Delay-Tolerant Networking: An Approach to Interplanetary Internet," *IEEE Commun. Magazine*, vol. 41, pp. 128–136, June 2003.

BURNETT, S., and PAINE, S.: *RSA Security's Official Guide to Cryptography*, Berkeley, CA: Osborne/McGraw-Hill, 2001.

BUSH, V.: "As We May Think," *Atlantic Monthly*, vol. 176, pp. 101–108, July 1945.

CAPETANAKIS, J.I.: "Tree Algorithms for Packet Broadcast Channels," *IEEE Trans. on Information Theory*, vol. IT–5, pp. 505–515, Sept. 1979.

CASTAGNOLI, G., BRAUER, S., and HERRMANN, M.: "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits," *IEEE Trans. on Commun.*, vol. 41, pp. 883–892, June 1993.

CERF, V., and KAHN, R.: "A Protocol for Packet Network Interconnection," *IEEE Trans. on Commun.*, vol. COM–2, pp. 637–648, May 1974.

CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W., WALLACH, D., BURROWS, M., CHANDRA, T., FIKES, A., and GRUBER, R.: "Bigtable: A Distributed Storage System for Structured Data," *Proc. OSDI 2006 Symp.*, USENIX, pp. 15–29, 2006.

CHASE, J.S., GALLATIN, A.J., and YOCUM, K.G.: "End System Optimizations for High-Speed TCP," *IEEE Commun. Magazine*, vol. 39, pp. 68–75, Apr. 2001.

CHEN, S., and NAHRSTEDT, K.: "An Overview of QoS Routing for Next-Generation Networks," *IEEE Network Magazine*, vol. 12, pp. 64–69, Nov./Dec. 1998.

CHIU, D., and JAIN, R.: "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Comput. Netw. ISDN Syst.*, vol. 17, pp. 1–4, June 1989.

CISCO: "Cisco Visual Networking Index: Forecast and Methodology, 2009–2014," Cisco Systems Inc., June 2010.

CLARK, D.D.: "The Design Philosophy of the DARPA Internet Protocols," *Proc. SIGCOMM '88 Conf.*, ACM, pp. 106–114, 1988.

CLARK, D.D.: "Window and Acknowledgement Strategy in TCP," RFC 813, July 1982.

CLARK, D.D., JACOBSON, V., ROMKEY, J., and SALWEN, H.: "An Analysis of TCP Processing Overhead," *IEEE Commun. Magazine*, vol. 27, pp. 23–29, June 1989.

CLARK, D.D., SHENKER, S., and ZHANG, L.: "Supporting Real-Time Applications in an Integrated Services Packet Network," *Proc. SIGCOMM '92 Conf.*, ACM, pp. 14–26, 1992.

CLARKE, A.C.: "Extra-Terrestrial Relays," *Wireless World*, 1945.

CLARKE, I., MILLER, S.G., HONG, T.W., SANDBERG, O., and WILEY, B.: "Protecting Free Expression Online with Freenet," *IEEE Internet Computing*, vol. 6, pp. 40–49, Jan.–Feb. 2002.

COHEN, B.: "Incentives Build Robustness in BitTorrent," *Proc. First Workshop on Economics of Peer-to-Peer Systems*, June 2003.

COMER, D.E.: *The Internet Book*, 4th ed., Englewood Cliffs, NJ: Prentice Hall, 2007.

COMER, D.E.: *Internetworking with TCP/IP*, vol. 1, 5th ed., Englewood Cliffs, NJ: Prentice Hall, 2005.

CRAVER, S.A., WU, M., LIU, B., STUBBLEFIELD, A., SWARTZLANDER, B., WALLACH, D.W., DEAN, D., and FELTEN, E.W.: "Reading Between the Lines: Lessons from the SDMI Challenge," *Proc. 10th USENIX Security Symp.*, USENIX, 2001.

CROVELLA, M., and KRISHNAMURTHY, B.: *Internet Measurement*, New York: John Wiley & Sons, 2006.

DAEMEN, J., and RIJMEN, V.: *The Design of Rijndael*, Berlin: Springer-Verlag, 2002.

DALAL, Y., and METCLFE, R.: "Reverse Path Forwarding of Broadcast Packets," *Commun. of the ACM*, vol. 21, pp. 1040–1048, Dec. 1978.

DAVIE, B., and FARREL, A.: *MPLS: Next Steps*, San Francisco: Morgan Kaufmann, 2008.

DAVIE, B., and REKHTER, Y.: *MPLS Technology and Applications*, San Francisco: Morgan Kaufmann, 2000.

DAVIES, J.: *Understanding IPv6*, 2nd ed., Redmond, WA: Microsoft Press, 2008.

DAY, J.D.: "The (Un)Revised OSI Reference Model," *Computer Commun. Rev.*, vol. 25, pp. 39–55, Oct. 1995.

DAY, J.D., and ZIMMERMANN, H.: "The OSI Reference Model," *Proc. of the IEEE*, vol. 71, pp. 1334–1340, Dec. 1983.

DECANDIA, G., HASTORIN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., and VOGELS, W.: "Dynamo: Amazon's Highly Available Key-value Store," *Proc. 19th Symp. on Operating Systems Prin.*, ACM, pp. 205–220, Dec. 2007.

DEERING, S.E.: "SIP: Simple Internet Protocol," *IEEE Network Magazine*, vol. 7, pp. 16–28, May/June 1993.

DEERING, S., and CHERITON, D.: "Multicast Routing in Datagram Networks and Extended LANs," *ACM Trans. on Computer Systems*, vol. 8, pp. 85–110, May 1990.

DEMERS, A., KESHAV, S., and SHENKER, S.: "Analysis and Simulation of a Fair Queueing Algorithm," *Internetwork: Research and Experience*, vol. 1, pp. 3–26, Sept. 1990.

DENNING, D.E., and SACCO, G.M.: "Timestamps in Key Distribution Protocols," *Commun. of the ACM*, vol. 24, pp. 533–536, Aug. 1981.

DEVARAPALLI, V., WAKIKAWA, R., PETRESCU, A., and THUBERT, P.: "Network Mobility (NEMO) Basic Support Protocol," RFC 3963, Jan. 2005.

DIFFIE, W., and HELLMAN, M.E.: "Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *IEEE Computer*, vol. 10, pp. 74–84, June 1977.

DIFFIE, W., and HELLMAN, M.E.: "New Directions in Cryptography," *IEEE Trans. on Information Theory*, vol. IT–2, pp. 644–654, Nov. 1976.

DIJKSTRA, E.W.: "A Note on Two Problems in Connexion with Graphs," *Numer. Math.*, vol. 1, pp. 269–271, Oct. 1959.

DILLEY, J., MAGGS, B., PARIKH, J., PROKOP, H., SITARAMAN, R., and WHEIL, B.: "Globally Distributed Content Delivery," *IEEE Internet Computing*, vol. 6, pp. 50–58, 2002.

DINGLEDINE, R., MATHEWSON, N., SYVERSON, P.: "Tor: The Second-Generation Onion Router," *Proc. 13th USENIX Security Symp.*, USENIX, pp. 303–320, Aug. 2004.

DONAHOO, M., and CALVERT, K.: *TCP/IP Sockets in C*, 2nd ed., San Francisco: Morgan Kaufmann, 2009.

DONAHOO, M., and CALVERT, K.: *TCP/IP Sockets in Java*, 2nd ed., San Francisco: Morgan Kaufmann, 2008.

DONALDSON, G., and JONES, D.: "Cable Television Broadband Network Architectures," *IEEE Commun. Magazine*, vol. 39, pp. 122–126, June 2001.

DORFMAN, R.: "Detection of Defective Members of a Large Population," *Annals Math. Statistics*, vol. 14, pp. 436–440, 1943.

DUTCHER, B.: *The NAT Handbook*, New York: John Wiley & Sons, 2001.

DUTTA-ROY, A.: "An Overview of Cable Modem Technology and Market Perspectives," *IEEE Commun. Magazine*, vol. 39, pp. 81–88, June 2001.

EDELMAN, B., OSTROVSKY, M., and SCHWARZ, M.: "Internet Advertising and the Generalized Second-Price Auction: Selling Billions of Dollars Worth of Keywords," *American Economic Review*, vol. 97, pp. 242–259, Mar. 2007.

EL GAMAL, T.: "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Trans. on Information Theory*, vol. IT–1, pp. 469–472, July 1985.

EPCGLOBAL: *EPC Radio-Frequency Identity Protocols Class– Generation– UHF RFID Protocol for Communication at 860-MHz to 960-MHz Version 1.2.0*, Brussels: EPCglobal Inc., Oct. 2008.

FALL, K.: "A Delay-Tolerant Network Architecture for Challenged Internets," *Proc. SIGCOMM 2003 Conf.*, ACM, pp. 27–34, Aug. 2003.

FALOUTSOS, M., FALOUTSOS, P., and FALOUTSOS, C.: "On Power-Law Relationships of the Internet Topology," *Proc. SIGCOMM '99 Conf.*, ACM, pp. 251–262, 1999.

FARRELL, S., and CAHILL, V.: *Delay- and Disruption-Tolerant Networking*, London: Artech House, 2007.

FELLOWS, D., and JONES, D.: "DOCSIS Cable Modem Technology," *IEEE Commun. Magazine*, vol. 39, pp. 202–209, Mar. 2001.

FENNER, B., HANDLEY, M., HOLBROOK, H., and KOUVELAS, I.: "Protocol Independent Multicast-Sparse Mode (PIM-SM)," RFC 4601, Aug. 2006.

FERGUSON, N., SCHNEIER, B., and KOHNO, T.: *Cryptography Engineering: Design Principles and Practical Applications*, New York: John Wiley & Sons, 2010.

FLANAGAN, D.: *JavaScript: The Definitive Guide*, 6th ed., Sebastopol, CA: O'Reilly, 2010.

FLETCHER, J.: "An Arithmetic Checksum for Serial Transmissions," *IEEE Trans. on Commun.*, vol. COM–0, pp. 247–252, Jan. 1982.

FLOYD, S., HANDLEY, M., PADHYE, J., and WIDMER, J.: "Equation-Based Congestion Control for Unicast Applications," *Proc. SIGCOMM 2000 Conf.*, ACM, pp. 43–56, Aug. 2000.

FLOYD, S., and JACOBSON, V.: "Random Early Detection for Congestion Avoidance," *IEEE/ACM Trans. on Networking*, vol. 1, pp. 397–413, Aug. 1993.

FLUHRER, S., MANTIN, I., and SHAMIR, A.: "Weakness in the Key Scheduling Algorithm of RC4," *Proc. Eighth Ann. Workshop on Selected Areas in Cryptography*, Berlin: Springer-Verlag LNCS 2259, pp. 1–24, 2001.

FORD, B.: "Structured Streams: A New Transport Abstraction," *Proc. SIGCOMM 2007 Conf.*, ACM, pp. 361–372, 2007.

FORD, L.R., Jr., and FULKERSON, D.R.: *Flows in Networks*, Princeton, NJ: Princeton University Press, 1962.

FORD, W., and BAUM, M.S.: *Secure Electronic Commerce*, Upper Saddle River, NJ: Prentice Hall, 2000.

FORNEY, G.D.: "The Viterbi Algorithm," *Proc. of the IEEE*, vol. 61, pp. 268–278, Mar. 1973.

FOULI, K., and MALER, M.: "The Road to Carrier-Grade Ethernet," *IEEE Commun. Magazine*, vol. 47, pp. S30–S38, Mar. 2009.

FOX, A., GRIBBLE, S., BREWER, E., and AMIR, E.: "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," *SIGOPS Oper. Syst. Rev.*, vol. 30, pp. 160–170, Dec. 1996.

FRANCIS, P.: "A Near-Term Architecture for Deploying Pip," *IEEE Network Magazine*, vol. 7, pp. 30–37, May/June 1993.

FRASER, A.G.: "Towards a Universal Data Transport System," *IEEE J. on Selected Areas in Commun.*, vol. 5, pp. 803–816, Nov. 1983.

FRIDRICH, J.: *Steganography in Digital Media: Principles, Algorithms, and Applications*, Cambridge: Cambridge University Press, 2009.

FULLER, V., and LI, T.: "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan," RFC 4632, Aug. 2006.

GALLAGHER, R.G.: "A Minimum Delay Routing Algorithm Using Distributed Computation," *IEEE Trans. on Commun.*, vol. COM–5, pp. 73–85, Jan. 1977.

GALLAGHER, R.G.: "Low-Density Parity Check Codes," *IRE Trans. on Information Theory*, vol. 8, pp. 21–28, Jan. 1962.

GARFINKEL, S., with SPAFFORD, G.: *Web Security, Privacy, and Commerce*, Sebastopol, CA: O'Reilly, 2002.

GAST, M.: *802.11 Wireless Networks: The Definitive Guide*, 2nd ed., Sebastopol, CA: O'Reilly, 2005.

GERSHENFELD, N., and KRIKORIAN, R., and COHEN, D.: "The Internet of Things," *Scientific American*, vol. 291, pp. 76–81, Oct. 2004.

GILDER, G.: "Metcalfe's Law and Legacy," *Forbes ASAP*, Sepy. 13, 1993.

GOODE, B.: "Voice over Internet Protocol," *Proc. of the IEEE*, vol. 90, pp. 1495–1517, Sept. 2002.

GORALSKI, W.J.: *SONET*, 2nd ed., New York: McGraw-Hill, 2002.

GRAYSON, M., SHATZKAMER, K., and WAINNER, S.: *IP Design for Mobile Networks*, Indianapolis, IN: Cisco Press, 2009.

GROBE, K., and ELBERS, J.: "PON in Adolescence: From TDMA to WDM-PON," *IEEE Commun. Magazine*, vol. 46, pp. 26–34, Jan. 2008.

GROSS, G., KAYCEE, M., LIN, A., MALIS, A., and STEPHENS, J.: "The PPP Over AAL5," RFC 2364, July 1998.

HA, S., RHEE, I., and LISONG, X.: "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 64–74, June 2008.

HAFNER, K., and LYON, M.: *Where Wizards Stay Up Late*, New York: Simon & Schuster, 1998.

HALPERIN, D., HEYDT-BENJAMIN, T., RANSFORD, B., CLARK, S., DEFEND, B., MORGAN, W., FU, K., KOHNO, T., and MAISEL, W.: "Pacemakers and Implantable Cardi-

ac Defibrillators: Software Radio Attacks and Zero-Power Defenses," *IEEE Symp. on Security and Privacy*, pp. 129–142, May 2008.

HALPERIN, D., HU, W., SHETH, A., and WETHERALL, D.: "802.11 with Multiple Antennas for Dummies," *Computer Commun. Rev.*, vol. 40, pp. 19–25, Jan. 2010.

HAMMING, R.W.: "Error Detecting and Error Correcting Codes," *Bell System Tech. J.*, vol. 29, pp. 147–160, Apr. 1950.

HARTE, L., KELLOGG, S., DREHER, R., and SCHAFFNIT, T.: *The Comprehensive Guide to Wireless Technology*, Fuquay-Varina, NC: APDG Publishing, 2000.

HAWLEY, G.T.: "Historical Perspectives on the U.S. Telephone Loop," *IEEE Commun. Magazine*, vol. 29, pp. 24–28, Mar. 1991.

HECHT, J.: *Understanding Fiber Optics*, Upper Saddle River, NJ: Prentice Hall, 2005.

HELD, G.: *A Practical Guide to Content Delivery Networks*, 2nd ed., Boca Raton, FL: CRC Press, 2010.

HEUSSE, M., ROUSSEAU, F., BERGER-SABBATEL, G., DUDA, A.: "Performance Anomaly of 802.11b," *Proc. INFOCOM Conf.*, IEEE, pp. 836–843, 2003.

HIERTZ, G., DENTENEER, D., STIBOR, L., ZANG, Y., COSTA, X., and WALKE, B.: "The IEEE 802.11 Universe," *IEEE Commun. Magazine*, vol. 48, pp. 62–70, Jan. 2010.

HOE, J.: "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," *Proc. SIGCOMM '96 Conf.*, ACM, pp. 270–280, 1996.

HU, Y., and LI, V.O.K.:"Satellite-Based Internet: A Tutorial," *IEEE Commun. Magazine*, vol. 30, pp. 154–162, Mar. 2001.

HUITEMA, C.: *Routing in the Internet*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1999.

HULL, B., BYCHKOVSKY, V., CHEN, K., GORACZKO, M., MIU, A., SHIH, E., ZHANG, Y., BALAKRISHNAN, H., and MADDEN, S.: "CarTel: A Distributed Mobile Sensor Computing System," *Proc. Sensys 2006 Conf.*, ACM, pp. 125–138, Nov. 2006.

HUNTER, D., RAFTER, J., FAWCETT, J., VAN DER LIST, E., AYERS, D., DUCKETT, J., WATT, A., and MCKINNON, L.: *Beginning XML*, 4th ed., New Jersey: Wrox, 2007.

IRMER, T.: "Shaping Future Telecommunications: The Challenge of Global Standardization," *IEEE Commun. Magazine*, vol. 32, pp. 20–28, Jan. 1994.

ITU (INTERNATIONAL TELECOMMUNICATION UNION): *ITU Internet Reports 2005: The Internet of Things*, Geneva: ITU, Nov. 2005.

ITU (INTERNATIONAL TELECOMMUNICATION UNION): *Measuring the Information Society: The ICT Development Index*, Geneva: ITU, Mar. 2009.

JACOBSON, V.: "Compressing TCP/IP Headers for Low-Speed Serial Links," RFC 1144, Feb. 1990.

JACOBSON, V.: "Congestion Avoidance and Control," *Proc. SIGCOMM '88 Conf.*, ACM, pp. 314–329, 1988.

JAIN, R., and ROUTHIER, S.: "Packet Trains—Measurements and a New Model for Computer Network Traffic," *IEEE J. on Selected Areas in Commun.*, vol. 6, pp. 986–995, Sept. 1986.

JAKOBSSON, M., and WETZEL, S.: "Security Weaknesses in Bluetooth," *Topics in Cryptology: CT-RSA 2001*, Berlin: Springer-Verlag LNCS 2020, pp. 176–191, 2001.

JOEL, A.: "Telecommunications and the IEEE Communications Society," *IEEE Commun. Magazine*, 50th Anniversary Issue, pp. 6–14 and 162–167, May 2002.

JOHNSON, D., PERKINS, C., and ARKKO, J.: "Mobility Support in IPv6," RFC 3775, June 2004.

JOHNSON, D.B., MALTZ, D., and BROCH, J.: "DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks," *Ad Hoc Networking*, Boston: Addison-Wesley, pp. 139–172, 2001.

JUANG, P., OKI, H., WANG, Y., MARTONOSI, M., PEH, L., and RUBENSTEIN, D.: "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 96–107, Oct. 2002.

KAHN, D.: *The Codebreakers*, 2nd ed., New York: Macmillan, 1995.

KAMOUN, F., and KLEINROCK, L.: "Stochastic Performance Evaluation of Hierarchical Routing for Large Networks," *Computer Networks*, vol. 3, pp. 337–353, Nov. 1979.

KARN, P.: "MACA—A New Channel Access Protocol for Packet Radio," *ARRL/CRRL Amateur Radio Ninth Computer Networking Conf.*, pp. 134–140, 1990.

KARN, P., and PARTRIDGE, C.: "Improving Round-Trip Estimates in Reliable Transport Protocols," *Proc. SIGCOMM '87 Conf.*, ACM, pp. 2–7, 1987.

KARP, B., and KUNG, H.T.: "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," *Proc. MOBICOM 2000 Conf.*, ACM, pp. 243–254, 2000.

KASIM, A.: *Delivering Carrier Ethernet*, New York: McGraw-Hill, 2007.

KATABI, D., HANDLEY, M., and ROHRS, C.: "Internet Congestion Control for Future High Bandwidth-Delay Product Environments," *Proc. SIGCOMM 2002 Conf.*, ACM, pp. 89–102, 2002.

KATZ, D., and FORD, P.S.: "TUBA: Replacing IP with CLNP," *IEEE Network Magazine*, vol. 7, pp. 38–47, May/June 1993.

KAUFMAN, C., PERLMAN, R., and SPECINER, M.: *Network Security*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 2002.

KENT, C., and MOGUL, J.: "Fragmentation Considered Harmful," *Proc. SIGCOMM '87 Conf.*, ACM, pp. 390–401, 1987.

KERCKHOFF, A.: "La Cryptographie Militaire," *J. des Sciences Militaires*, vol. 9, pp. 5–38, Jan. 1883 and pp. 161–191, Feb. 1883.

KHANNA, A., and ZINKY, J.: "The Revised ARPANET Routing Metric," *Proc. SIGCOMM '89 Conf.*, ACM, pp. 45–56, 1989.

KIPNIS, J.: "Beating the System: Abuses of the Standards Adoption Process," *IEEE Commun. Magazine*, vol. 38, pp. 102–105, July 2000.

KLEINROCK, L.: "Power and Other Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications," *Proc. Intl. Conf. on Commun.*, pp. 43.1.1–43.1.10, June 1979.

KLEINROCK, L., and TOBAGI, F.: "Random Access Techniques for Data Transmission over Packet-Switched Radio Channels," *Proc. Nat. Computer Conf.*, pp. 187–201, 1975.

KOHLER, E., HANDLEY, H., and FLOYD, S.: "Designing DCCP: Congestion Control without Reliability," *Proc. SIGCOMM 2006 Conf.*, ACM, pp. 27–38, 2006.

KOODLI, R., and PERKINS, C.E.: *Mobile Inter-networking with IPv6*, New York: John Wiley & Sons, 2007.

KOOPMAN, P.: "32-Bit Cyclic Redundancy Codes for Internet Applications," *Proc. Intl. Conf. on Dependable Systems and Networks.*, IEEE, pp. 459–472, 2002.

KRISHNAMURTHY, B., and REXFORD, J.: *Web Protocols and Practice*, Boston: Addison-Wesley, 2001.

KUMAR, S., PAAR, C., PELZL, J., PFEIFFER, G., and SCHIMMLER, M.: "Breaking Ciphers with COPACOBANA: A Cost-Optimized Parallel Code Breaker," *Proc. 8th Cryptographic Hardware and Embedded Systems Wksp.*, IACR, pp. 101–118, Oct. 2006.

LABOVITZ, C., AHUJA, A., BOSE, A., and JAHANIAN, F.: "Delayed Internet Routing Convergence," *IEEE/ACM Trans. on Networking*, vol. 9, pp. 293–306, June 2001.

LAM, C.K.M., and TAN, B.C.Y.: "The Internet Is Changing the Music Industry," *Commun. of the ACM*, vol. 44, pp. 62–66, Aug. 2001.

LAOUTARIS, N., SMARAGDAKIS, G., RODRIGUEZ, P., and SUNDARAM, R.: "Delay Tolerant Bulk Data Transfers on the Internet," *Proc. SIGMETRICS 2009 Conf.*, ACM, pp. 229–238, June 2009.

LARMO, A., LINDSTROM, M., MEYER, M., PELLETIER, G., TORSNER, J., and WIEMANN, H.: "The LTE Link-Layer Design," *IEEE Commun. Magazine*, vol. 47, pp. 52–59, Apr. 2009.

LEE, J.S., and MILLER, L.E.: *CDMA Systems Engineering Handbook*, London: Artech House, 1998.

LELAND, W., TAQQU, M., WILLINGER, W., and WILSON, D.: "On the Self-Similar Nature of Ethernet Traffic," *IEEE/ACM Trans. on Networking*, vol. 2, pp. 1–15, Feb. 1994.

LEMON, J.: "Resisting SYN Flood DOS Attacks with a SYN Cache," *Proc. BSDCon Conf.*, USENIX, pp. 88–98, 2002.

LEVY, S.: "Crypto Rebels," *Wired*, pp. 54–61, May/June 1993.

LEWIS, M.: *Comparing, Designing, and Deploying VPNs*, Indianapolis, IN: Cisco Press, 2006.

LI, M., AGRAWAL, D., GANESAN, D., and VENKATARAMANI, A.: "Block-Switched Networks: A New Paradigm for Wireless Transport," *Proc. NSDI 2009 Conf.*, USENIX, pp. 423–436, 2009.

LIN, S., and COSTELLO, D.: *Error Control Coding*, 2nd ed., Upper Saddle River, NJ: Pearson Education, 2004.

LUBACZ, J., MAZURCZYK, W., and SZCZYPIORSKI, K.: "Vice over IP," *IEEE Spectrum*, pp. 42–47, Feb. 2010.

MACEDONIA, M.R.: "Distributed File Sharing," *IEEE Computer*, vol. 33, pp. 99–101, 2000.

MADHAVAN, J., KO, D., LOT, L., GANGPATHY, V., RASMUSSEN, A., and HALEVY, A.: "Google's Deep Web Crawl," *Proc. VLDB 2008 Conf.*, VLDB Endowment, pp. 1241–1252, 2008.

MAHAJAN, R., RODRIG, M., WETHERALL, D., and ZAHORJAN, J.: "Analyzing the MAC-Level Behavior of Wireless Networks in the Wild," *Proc. SIGCOMM 2006 Conf.*, ACM, pp. 75–86, 2006.

MALIS, A., and SIMPSON, W.: "PPP over SONET/SDH," RFC 2615, June 1999.

MASSEY, J.L.: "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. on Information Theory*, vol. IT–5, pp. 122–127, Jan. 1969.

MATSUI, M.: "Linear Cryptanalysis Method for DES Cipher," *Advances in Cryptology—Eurocrypt 1993 Proceedings*, Berlin: Springer-Verlag LNCS 765, pp. 386–397, 1994.

MAUFER, T.A.: *IP Fundamentals*, Upper Saddle River, NJ: Prentice Hall, 1999.

MAYMOUNKOV, P., and MAZIERES, D.: "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *Proc. First Intl. Wksp. on Peer-to-Peer Systems*, Berlin: Springer-Verlag LNCS 2429, pp. 53–65, 2002.

MAZIERES, D., and KAASHOEK, M.F.: "The Design, Implementation, and Operation of an Email Pseudonym Server," *Proc. Fifth Conf. on Computer and Commun. Security*, ACM, pp. 27–36, 1998.

MCAFEE LABS: *McAfee Threat Reports: First Quarter 2010*, McAfee Inc., 2010.

MENEZES, A.J., and VANSTONE, S.A.: "Elliptic Curve Cryptosystems and Their Implementation," *Journal of Cryptology*, vol. 6, pp. 209–224, 1993.

MERKLE, R.C., and HELLMAN, M.: "Hiding and Signatures in Trapdoor Knapsacks," *IEEE Trans. on Information Theory*, vol. IT–4, pp. 525–530, Sept. 1978.

METCALFE, R.M.: "Computer/Network Interface Design: Lessons from Arpanet and Ethernet," *IEEE J. on Selected Areas in Commun.*, vol. 11, pp. 173–179, Feb. 1993.

METCALFE, R.M., and BOGGS, D.R.: "Ethernet: Distributed Packet Switching for Local Computer Networks," *Commun. of the ACM*, vol. 19, pp. 395–404, July 1976.

METZ, C: "Interconnecting ISP Networks," *IEEE Internet Computing*, vol. 5, pp. 74–80, Mar.–Apr. 2001.

MISHRA, P.P., KANAKIA, H., and TRIPATHI, S.: "On Hop by Hop Rate-Based Congestion Control," *IEEE/ACM Trans. on Networking*, vol. 4, pp. 224–239, Apr. 1996.

MOGUL, J.C.: "IP Network Performance," in *Internet System Handbook*, D.C. Lynch and M.Y. Rose (eds.), Boston: Addison-Wesley, pp. 575–575, 1993.

MOGUL, J., and DEERING, S.: "Path MTU Discovery," RFC 1191, Nov. 1990.

MOGUL, J., and MINSHALL, G.: "Rethinking the Nagle Algorithm," *Comput. Commun. Rev.*, vol. 31, pp. 6–20, Jan. 2001.

MOY, J.: "Multicast Routing Extensions for OSPF," *Commun. of the ACM*, vol. 37, pp. 61–66, Aug. 1994.

MULLINS, J.: "Making Unbreakable Code," *IEEE Spectrum*, pp. 40–45, May 2002.

NAGLE, J.: "On Packet Switches with Infinite Storage," *IEEE Trans. on Commun.*, vol. COM–5, pp. 435–438, Apr. 1987.

NAGLE, J.: "Congestion Control in TCP/IP Internetworks," *Computer Commun. Rev.*, vol. 14, pp. 11–17, Oct. 1984.

NAUGHTON, J.: *A Brief History of the Future*, Woodstock, NY: Overlook Press, 2000.

NEEDHAM, R.M., and SCHROEDER, M.D.: "Using Encryption for Authentication in Large Networks of Computers," *Commun. of the ACM*, vol. 21, pp. 993–999, Dec. 1978.

NEEDHAM, R.M., and SCHROEDER, M.D.: "Authentication Revisited," *Operating Systems Rev.*, vol. 21, p. 7, Jan. 1987.

NELAKUDITI, S., and ZHANG, Z.-L.: "A Localized Adaptive Proportioning Approach to QoS Routing," *IEEE Commun. Magazine* vol. 40, pp. 66–71, June 2002.

NEUMAN, C., and TS'O, T.: "Kerberos: An Authentication Service for Computer Networks," *IEEE Commun. Mag.*, vol. 32, pp. 33–38, Sept. 1994.

NICHOLS, R.K., and LEKKAS, P.C.: *Wireless Security*, New York: McGraw-Hill, 2002.

NIST: "Secure Hash Algorithm," U.S. Government Federal Information Processing Standard 180, 1993.

NONNENMACHER, J., BIERSACK, E., and TOWSLEY, D.: "Parity-Based Loss Recovery for Reliable Multicast Transmission," *Proc. SIGCOMM '97 Conf.*, ACM, pp. 289–300, 1997.

NUCCI, A., and PAPAGIANNAKI, D.: *Design, Measurement and Management of Large-Scale IP Networks*, Cambridge: Cambridge University Press, 2008.

NUGENT, R., MUNAKANA, R., CHIN, A., COELHO, R., and PUIG-SUARI, J.: "The CubeSat: The PicoSatellite Standard for Research and Education," *Proc. SPACE 2008 Conf.*, AIAA, 2008.

ORAN, D.: "OSI IS-IS Intra-domain Routing Protocol," RFC 1142, Feb. 1990.

OTWAY, D., and REES, O.: "Efficient and Timely Mutual Authentication," *Operating Systems Rev.*, pp. 8–10, Jan. 1987.

PADHYE, J., FIROIU, V., TOWSLEY, D., and KUROSE, J.: "Modeling TCP Throughput: A Simple Model and Its Empirical Validation," *Proc. SIGCOMM '98 Conf.*, ACM, pp. 303–314, 1998.

PALAIS, J.C.: *Fiber Optic Commun.*, 5th ed., Englewood Cliffs, NJ: Prentice Hall, 2004.

PARAMESWARAN, M., SUSARLA, A., and WHINSTON, A.B.: "P2P Networking: An Information-Sharing Alternative," *IEEE Computer*, vol. 34, pp. 31–38, July 2001.

PAREKH, A., and GALLAGHER, R.: "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple-Node Case," *IEEE/ACM Trans. on Networking*, vol. 2, pp. 137–150, Apr. 1994.

PAREKH, A., and GALLAGHER, R.: "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Trans. on Networking*, vol. 1, pp. 344–357, June 1993.

PARTRIDGE, C., HUGHES, J., and STONE, J.: "Performance of Checksums and CRCs over Real Data," *Proc. SIGCOMM '95 Conf.*, ACM, pp. 68–76, 1995.

PARTRIDGE, C., MENDEZ, T., and MILLIKEN, W.: "Host Anycasting Service," RFC 1546, Nov. 1993.

PAXSON, V., and FLOYD, S.: "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Trans. on Networking*, vol. 3, pp. 226–244, June 1995.

PERKINS, C.: "IP Mobility Support for IPv4," RFC 3344, Aug. 2002.

PERKINS, C.E.: *RTP: Audio and Video for the Internet*, Boston: Addison-Wesley, 2003.

PERKINS, C.E. (ed.): *Ad Hoc Networking*, Boston: Addison-Wesley, 2001.

PERKINS, C.E.: *Mobile IP Design Principles and Practices*, Upper Saddle River, NJ: Prentice Hall, 1998.

PERKINS, C.E., and ROYER, E.: "The Ad Hoc On-Demand Distance-Vector Protocol," in *Ad Hoc Networking*, edited by C. Perkins, Boston: Addison-Wesley, 2001.

PERLMAN, R.: *Interconnections*, 2nd ed., Boston: Addison-Wesley, 2000.

PERLMAN, R.: *Network Layer Protocols with Byzantine Robustness*, Ph.D. thesis, M.I.T., 1988.

PERLMAN, R.: "An Algorithm for the Distributed Computation of a Spanning Tree in an Extended LAN," *Proc. SIGCOMM '85 Conf.*, ACM, pp. 44–53, 1985.

PERLMAN, R., and KAUFMAN, C.: "Key Exchange in IPsec," *IEEE Internet Computing*, vol. 4, pp. 50–56, Nov.–Dec. 2000.

PETERSON, W.W., and BROWN, D.T.: "Cyclic Codes for Error Detection," *Proc. IRE*, vol. 49, pp. 228–235, Jan. 1961.

PIATEK, M., KOHNO, T., and KRISHNAMURTHY, A.: "Challenges and Directions for Monitoring P2P File Sharing Networks—or Why My Printer Received a DMCA Takedown Notice," *3rd Workshop on Hot Topics in Security*, USENIX, July 2008.

PIATEK, M., ISDAL, T., ANDERSON, T., KRISHNAMURTHY, A., and VENKATARAMANI, V.: "Do Incentives Build Robustness in BitTorrent?," *Proc. NSDI 2007 Conf.*, USENIX, pp. 1–14, 2007.

PISCITELLO, D.M., and CHAPIN, A.L.: *Open Systems Networking: TCP/IP and OSI*, Boston: Addison-Wesley, 1993.

PIVA, A., BARTOLINI, F., and BARNI, M.: "Managing Copyrights in Open Networks," *IEEE Internet Computing*, vol. 6, pp. 18–26, May– 2002.

POSTEL, J.: "Internet Control Message Protocols," RFC 792, Sept. 1981.

RABIN, J., and McCATHIENEVILE, C.: "Mobile Web Best Practices 1.0," W3C Recommendation, July 2008.

RAMAKRISHNAM, K.K., FLOYD, S., and BLACK, D.: "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sept. 2001.

RAMAKRISHNAN, K.K., and JAIN, R.: "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer," *Proc. SIGCOMM '88 Conf.*, ACM, pp. 303–313, 1988.

RAMASWAMI, R., KUMAR, S., and SASAKI, G.: *Optical Networks: A Practical Perspective*, 3rd ed., San Francisco: Morgan Kaufmann, 2009.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., and SHENKER, S.: "A Scalable Content-Addressable Network," *Proc. SIGCOMM 2001 Conf.*, ACM, pp. 161–172, 2001.

RIEBACK, M., CRISPO, B., and TANENBAUM, A.: "Is Your Cat Infected with a Computer Virus?," *Proc. IEEE Percom*, pp. 169–179, Mar. 2006.

RIVEST, R.L.: "The MD5 Message-Digest Algorithm," RFC 1320, Apr. 1992.

RIVEST, R.L., SHAMIR, A., and ADLEMAN, L.: "On a Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Commun. of the ACM*, vol. 21, pp. 120–126, Feb. 1978.

ROBERTS, L.G.: "Extensions of Packet Communication Technology to a Hand Held Personal Terminal," *Proc. Spring Joint Computer Conf.*, AFIPS, pp. 295–298, 1972.

ROBERTS, L.G.: "Multiple Computer Networks and Intercomputer Communication," *Proc. First Symp. on Operating Systems Prin.*, ACM, pp. 3.1–3.6, 1967.

ROSE, M.T.: *The Simple Book*, Englewood Cliffs, NJ: Prentice Hall, 1994.

ROSE, M.T.: *The Internet Message*, Englewood Cliffs, NJ: Prentice Hall, 1993.

ROWSTRON, A., and DRUSCHEL, P.: "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Storage Utility," *Proc. 18th Int'l Conf. on Distributed Systems Platforms*, London: Springer-Verlag LNCS 2218, pp. 329–350, 2001.

RUIZ-SANCHEZ, M.A., BIERSACK, E.W., and DABBOUS, W.: "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network Magazine*, vol. 15, pp. 8–23, Mar.–Apr. 2001.

SALTZER, J.H., REED, D.P., and CLARK, D.D.: "End-to-End Arguments in System Design," *ACM Trans. on Computer Systems*, vol. 2, pp. 277–288, Nov. 1984.

SAMPLE, A., YEAGER, D., POWLEDGE, P., MAMISHEV, A., and SMITH, J.: "Design of an RFID-Based Battery-Free Programmable Sensing Platform," *IEEE Trans. on Instrumentation and Measurement*, vol. 57, pp. 2608–2615, Nov. 2008.

SAROIU, S., GUMMADI, K., and GRIBBLE, S.: "Measuring and Analyzing the Characteristics of Napster & Gnutella Hosts," *Multim. Syst.*, vol. 9,, pp. 170–184, Aug. 2003.

SCHALLER, R.: "Moore's Law: Past, Present and Future," *IEEE Spectrum*, vol. 34, pp. 52–59, June 1997.

SCHNEIER, B.: *Secrets and Lies*, New York: John Wiley & Sons, 2004.

SCHNEIER, B.: *E-Mail Security*, New York: John Wiley & Sons, 1995.

SCHNORR, C.P.: "Efficient Signature Generation for Smart Cards," *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.

SCHOLTZ, R.A.: "The Origins of Spread-Spectrum Communications," *IEEE Trans. on Commun.*, vol. COM–0, pp. 822–854, May 1982.

SCHWARTZ, M., and ABRAMSON, N.: "The AlohaNet: Surfing for Wireless Data," *IEEE Commun. Magazine*, vol. 47, pp. 21–25, Dec. 2009.

SEIFERT, R., and EDWARDS, J.: *The All-New Switch Book*, NY: John Wiley, 2008.

SENN, J.A.: "The Emergence of M-Commerce," *IEEE Computer*, vol. 33, pp. 148–150, Dec. 2000.

SERJANTOV, A.: "Anonymizing Censorship Resistant Systems," *Proc. First Int'l Workshop on Peer-to-Peer Systems*, London: Springer-Verlag LNCS 2429, pp. 111–120, 2002.

SHACHAM, N., and MCKENNY, P.: "Packet Recovery in High-Speed Networks Using Coding and Buffer Management," *Proc. INFOCOM Conf.*, IEEE, pp. 124–131, June 1990.

SHAIKH, A., REXFORD, J., and SHIN, K.: "Load-Sensitive Routing of Long-Lived IP Flows," *Proc. SIGCOMM '99 Conf.*, ACM, pp. 215–226, Sept. 1999.

SHALUNOV, S., and CARLSON, R.: "Detecting Duplex Mismatch on Ethernet," *Passive and Active Network Measurement*, Berlin: Springer-Verlag LNCS 3431, pp. 3135–3148, 2005.

SHANNON, C.: "A Mathematical Theory of Communication," *Bell System Tech. J.*, vol. 27, pp. 379–423, July 1948; and pp. 623–656, Oct. 1948.

SHEPARD, S.: *SONET/SDH Demystified*, New York: McGraw-Hill, 2001.

SHREEDHAR, M., and VARGHESE, G.: "Efficient Fair Queueing Using Deficit Round Robin," *Proc. SIGCOMM '95 Conf.*, ACM, pp. 231–243, 1995.

SIMPSON, W.: *Video Over IP*, 2nd ed., Burlington, MA: Focal Press, 2008.

SIMPSON, W.: "PPP in HDLC-like Framing," RFC 1662, July 1994b.

SIMPSON, W.: "The Point-to-Point Protocol (PPP)," RFC 1661, July 1994a.

SIU, K., and JAIN, R.: "A Brief Overview of ATM: Protocol Layers, LAN Emulation, and Traffic," *ACM Computer Communications Review*, vol. 25, pp. 6–20, Apr. 1995.

SKOUDIS, E., and LISTON, T.: *Counter Hack Reloaded*, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2006.

SMITH, D.K., and ALEXANDER, R.C.: *Fumbling the Future*, New York: William Morrow, 1988.

SNOEREN, A.C., and BALAKRISHNAN, H.: "An End-to-End Approach to Host Mobility," *Int'l Conf. on Mobile Computing and Networking* , ACM, pp. 155–166, 2000.

SOBEL, D.L.: "Will Carnivore Devour Online Privacy," *IEEE Computer*, vol. 34, pp. 87–88, May 2001.

SOTIROV, A., STEVENS, M., APPELBAUM, J., LENSTRA, A., MOLNAR, D., OSVIK, D., and DE WEGER, B.: "MD5 Considered Harmful Today," *Proc. 25th Chaos Communication Congress*, Verlag Art d'Ameublement, 2008.

SOUTHEY, R.: *The Doctors*, London: Longman, Brown, Green and Longmans, 1848.

SPURGEON, C.E.: *Ethernet: The Definitive Guide*, Sebastopol, CA: O'Reilly, 2000.

STALLINGS, W.: *Data and Computer Communications*, 9th ed., Upper Saddle River, NJ: Pearson Education, 2010.

STARR, T., SORBARA, M., COIFFI, J., and SILVERMAN, P.: "DSL Advances," Upper Saddle River, NJ: Prentice Hall, 2003.

STEVENS, W.R.: *TCP/IP Illustrated: The Protocols*, Boston: Addison Wesley, 1994.

STINSON, D.R.: *Cryptography Theory and Practice*, 2nd ed., Boca Raton, FL: CRC Press, 2002.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M.F., and BALAKRISHNAN, H.: "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. SIGCOMM 2001 Conf.*, ACM, pp. 149–160, 2001.

STUBBLEFIELD, A., IOANNIDIS, J., and RUBIN, A.D.: "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP," *Proc. Network and Distributed Systems Security Symp.*, ISOC, pp. 1–11, 2002.

STUTTARD, D., and PINTO, M.: *The Web Application Hacker's Handbook*, New York: John Wiley & Sons, 2007.

SU, S.: *The UMTS Air Interface in RF Engineering*, New York: McGraw-Hill, 2007.

SULLIVAN, G., and WIEGAND, T.: "Tree Algorithms for Packet Broadcast Channels," *Proc. of the IEEE*, vol. 93, pp. 18–31, Jan. 2005.

SUNSHINE, C.A., and DALAL, Y.K.: "Connection Management in Transport Protocols," *Computer Networks*, vol. 2, pp. 454–473, 1978.

TAN, K., SONG, J., ZHANG, Q., and SRIDHARN, M.: "A Compound TCP Approach for High-Speed and Long Distance Networks," *Proc. INFOCOM Conf.*, IEEE, pp. 1–12, 2006.

TANENBAUM, A.S.: *Modern Operating Systems*, 3rd ed., Upper Saddle River, NJ: Prentice Hall, 2007.

TANENBAUM, A.S., and VAN STEEN, M.: *Distributed Systems: Principles and Paradigms*, Upper Saddle River, NJ: Prentice Hall, 2007.

TOMLINSON, R.S.: "Selecting Sequence Numbers," *Proc. SIGCOMM/SIGOPS Interprocess Commun. Workshop*, ACM, pp. 11–23, 1975.

TUCHMAN, W.: "Hellman Presents No Shortcut Solutions to DES," *IEEE Spectrum*, vol. 16, pp. 40–41, July 1979.

TURNER, J.S.: "New Directions in Communications (or Which Way to the Information Age)," *IEEE Commun. Magazine*, vol. 24, pp. 8–15, Oct. 1986.

UNGERBOECK, G.: "Trellis-Coded Modulation with Redundant Signal Sets Part I: Introduction," *IEEE Commun. Magazine*, vol. 25, pp. 5–11, Feb. 1987.

VALADE, J.: *PHP & MySQL for Dummies*, 5th ed., New York: John Wiley & Sons, 2009.

VARGHESE, G.: *Network Algorithmics*, San Francisco: Morgan Kaufmann, 2004.

VARGHESE, G., and LAUCK, T.: "Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility," *Proc. 11th Symp. on Operating Systems Prin.*, ACM, pp. 25–38, 1987.

VERIZON BUSINESS: *2009 Data Breach Investigations Report*, Verizon, 2009.

VITERBI, A.: *CDMA: Principles of Spread Spectrum Communication*, Englewood Cliffs, NJ: Prentice Hall, 1995.

VON AHN, L., BLUM, B., and LANGFORD, J.: "Telling Humans and Computers Apart Automatically," *Commun. of the ACM*, vol. 47, pp. 56–60, Feb. 2004.

WAITZMAN, D., PARTRIDGE, C., and DEERING, S.: "Distance Vector Multicast Routing Protocol," RFC 1075, Nov. 1988.

WALDMAN, M., RUBIN, A.D., and CRANOR, L.F.: "Publius: A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System," *Proc. Ninth USENIX Security Symp.*, USENIX, pp. 59–72, 2000.

WANG, Z., and CROWCROFT, J.: "SEAL Detects Cell Misordering," *IEEE Network Magazine*, vol. 6, pp. 8–9, July 1992.

WANT, R.: *RFID Explained*, San Rafael, CA: Morgan Claypool, 2006.

WARNEKE, B., LAST, M., LIEBOWITZ, B., and PISTER, K.S.J.: "Smart Dust: Communicating with a Cubic Millimeter Computer," *IEEE Computer*, vol. 34, pp. 44–51, Jan. 2001.

WAYNER, P.: *Disappearing Cryptography: Information Hiding, Steganography, and Watermarking*, 3rd ed., San Francisco: Morgan Kaufmann, 2008.

WEI, D., CHENG, J., LOW, S., and HEGDE, S.: "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM Trans. on Networking*, vol. 14, pp. 1246–1259, Dec. 2006.

WEISER, M.: "The Computer for the Twenty-First Century," *Scientific American*, vol. 265, pp. 94–104, Sept. 1991.

WELBOURNE, E., BATTLE, L., COLE, G., GOULD, K., RECTOR, K., RAYMER, S., BALAZINSKA, M., and BORRIELLO, G.: "Building the Internet of Things Using RFID," *IEEE Internet Computing*, vol. 13, pp. 48–55, May 2009.

WITTENBURG, N.: *Understanding Voice Over IP Technology*, Clifton Park, NY: Delmar Cengage Learning, 2009.

WOLMAN, A., VOELKER, G., SHARMA, N., CARDWELL, N., KARLIN, A., and LEVY, H.: "On the Scale and Performance of Cooperative Web Proxy Caching," *Proc. 17th Symp. on Operating Systems Prin.*, ACM, pp. 16–31, 1999.

WOOD, L., IVANCIC, W., EDDY, W., STEWART, D., NORTHAM, J., JACKSON, C., and DA SILVA CURIEL, A.: "Use of the Delay-Tolerant Networking Bundle Protocol from Space," *Proc. 59th Int'l Astronautical Congress*, Int'l Astronautical Federation, pp. 3123–3133, 2008.

WU, T.: "Network Neutrality, Broadband Discrimination," *Journal on Telecom. and High-Tech. Law*, vol. 2, pp. 141–179, 2003.

WYLIE, J., BIGRIGG, M.W., STRUNK, J.D., GANGER, G.R., KILICCOTE, H., and KHOSLA, P.K.: "Survivable Information Storage Systems," *IEEE Computer*, vol. 33, pp. 61–68, Aug. 2000.

YU, T., HARTMAN, S., and RAEBURN, K.: "The Perils of Unauthenticated Encryption: Kerberos Version 4," *Proc. NDSS Symposium*, Internet Society, Feb. 2004.

YUVAL, G.: "How to Swindle Rabin," *Cryptologia*, vol. 3, pp. 187–190, July 1979.

ZACKS, M.: "Antiterrorist Legislation Expands Electronic Snooping," *IEEE Internet Computing*, vol. 5, pp. 8–9, Nov.–Dec. 2001.

ZHANG, Y., BRESLAU, L., PAXSON, V., and SHENKER, S.: "On the Characteristics and Origins of Internet Flow Rates," *Proc. SIGCOMM 2002 Conf.*, ACM, pp. 309–322, 2002.

ZHAO, B., LING, H., STRIBLING, J., RHEA, S., JOSEPH, A., and KUBIATOWICZ, J.: "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE J. on Selected Areas in Commun.*, vol. 22, pp. 41–53, Jan. 2004.

ZIMMERMANN, P.R.: *The Official PGP User's Guide*, Cambridge, MA: M.I.T. Press, 1995a.

ZIMMERMANN, P.R.: *PGP: Source Code and Internals*, Cambridge, MA: M.I.T. Press, 1995b.

ZIPF, G.K.: *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*, Boston: Addison-Wesley, 1949.

ZIV, J., and LEMPEL, Z.: "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, vol. IT–3, pp. 337–343, May 1977.

This page intentionally left blank

INDEX

This page intentionally left blank

INDEX

Numbers

1-persistent CSMA, 266
3GPP (*see* Third Generation Partnership Project)
4B/5B encoding, 128, 292
8B/10B encoding, 129, 295
10-Gigabit Ethernet, 296–297
64B/66B encoding, 297
100Base-FX Ethernet, 292
100Base-T4 Ethernet, 291–292
802.11 (*see* IEEE 802.11)
1000Base-T Ethernet, 295–296

A

A-law, 153, 700
AAL5 (*see* ATM Adaptation Layer 5)
Abstract Syntax Notation 1, 809
Access point, 19, 70, 299
  transport layer, 509
Acknowledged datagram, 37
Acknowledgement
  cumulative, 238, 558
  duplicate, 577
  selective, 560, 580
Acknowledgement clock, TCP, 574
Acknowledgement frame, 43
ACL (*see* Asynchronous Connectionless link)
Active server page, 676
ActiveX, 858–859
ActiveX control, 678
Ad hoc network, 70, 299, 389–392
  routing, 389–392
Ad hoc on-demand distance vector, 389
Adaptation, rate, 301
Adaptive frequency hopping, Bluetooth, 324
Adaptive routing algorithm, 364
Adaptive tree walk protocol, 275–277
ADC (*see* Analog-to-Digital Converter)
Additive increase multiplicative decrease law, 537
Address resolution protocol, 467–469
  gratuitous, 469
Address resolution protocol proxy, 469
Addressing, 34
  classful IP, 449–451
  transport layer, 509–512

Adjacent router, 478
Admission control, 395, 397–398, 415–418
ADSL (*see* Asymmetric Digital Subscriber Line)
Advanced audio coding, 702
Advanced Encryption Standard, 312, 783–787
Advanced Mobile Phone System, 65, 167–170
Advanced Research Projects Agency, 56
Advanced video coding, 710
AES (*see* Advanced Encryption Standard)
Aggregation, route, 447
AH (*see* Authentication Header)
AIFS (*see* Arbitration InterFrame Space)
AIMD (*see* Additive Increase Multiplicative Decrease law)
Air interface, 66, 171
AJAX (*see* Asynchronous JavaScript and XML)
Akamai, 745–746
Algorithm
  adaptive routing, 364
  backward learning, 333, 335
  Bellman-Ford, 370
  binary exponential backoff, 285–286
  congestion control, 392–404
  Dijkstra's, 369
  encoding, 550
  forwarding, 27
  international data encryption, 842
  Karn's, 571
  leaky bucket, 397, 407–411
  longest matching prefix, 448
  lottery, 112
  Nagle's, 566
  network layer routing, 362–392
  nonadaptive, 363–364
  reverse path forwarding, 381, 419
  Rivest Shamir Adleman, 794–796
  routing, 27, 362–392
  token bucket, 408–411
Alias, 617, 619, 630
Allocation, channel, 258–261
ALOHA, 72
  pure, 262–264
  slotted, 264–266
Alternate mark inversion, 129
AMI (*see* Alternate Mark Inversion)
Amplitude shift keying, 130
AMPS (*see* Advanced Mobile Phone System)
Analog-to-digital converter, 699
Andreessen, Marc, 646–647
Anomaly, rate, 309
Anonymous remailer, 861–863
ANSNET, 60
Antenna, sectored, 178
Antheil, George, 108
Anycast routing, 385–386
AODV (*see* Ad-hoc On-demand Distance Vector routing)
AP (*see* Access Point)
Apocalypse of the two elephants, 51–52
Applet, 678
Application
  business, 3
  Web, 4
Application layer, 45, 47–48
  content-delivery network, 734–757
  distributed hash table, 753–757
  Domain Name System, 611–623
  email, 623–646
  multimedia, 607–734
  world Wide Web, 646–697
Application-level gateway, 819
APSD (*see* Automatic Power Save Delivery)
Arbitration interframe space, 308
Architectural overview, Web, 647–649
Architecture and services, email, 624–626
Area, autonomous system
  backbone, 476
  stub, 477
Area border router, 476
ARP (*see* Address Resolution Protocol)
ARPA (*see* Advanced Research Projects Agency)
ARPANET, 55–59
ARQ (*see* Automatic Repeat reQuest)
AS (*see* Autonomous System)
ASK (*see* Amplitude Shift Keying)
ASN.1 (*see* Abstract Syntax Notation 1)
ASP (*see* Active Server Pages)
Aspect ratio, video, 705
Association, IEEE 802.11, 311
Assured forwarding, 423–424
Asymmetric digital subscriber line, 94, 124, 147, 248–250
  vs. cable, 185
Asynchronous connectionless link, 325
Asynchronous I/O, 682
Asynchronous Javascript and XML, 679–683
Asynchronous transfer mode, 249
AT&T, 55, 110
ATM (*see* Asynchronous Transfer Mode)
ATM adaptation layer 5, 250

Attack
  birthday, 804–806
  bucket brigade, 835
  chosen plaintext, 769
  ciphertext-only, 769
  denial of service, 820
  keystream reuse, 791
  known plaintext, 769
  man-in-the-middle, 835
  reflection, 829
  replay, 836
Attenuation, 102, 109
Attribute
  cryptographic certificate, 808
  HTML, 664
Auction, spectrum, 112
Audio
  digital, 699–704
  streaming, 697–704
Audio compression, 701–704
Authentication, 35
  IEEE 802.11, 311
  Needham-Schroeder, 836–838
  using key distribution center, 835–838
Authentication header, 815–816
Authentication protocol, 827–841
Authentication using a shared secret, 828–833
Authentication using Kerberos, 838–840
Authentication using public keys, 840–841
Authenticode, 858
Authoritative record, 620
Autocorrelation, 176
Autonegotiation, 293
Automatic power save delivery, 307
Automatic repeat request, 225, 522
Autonomous system, 432, 437, 472–476, 474
Autoresponder, 629
AVC (*see* Advanced Video Coding)

B

B-frame, 712
Backbone, Internet, 63
Backbone area, 476
Backbone router, 476
Backpressure, hop-by-hop, 400–401
Backscatter, RFID, 74, 329
Backward learning algorithm, 333, 335
Backward learning bridge, 335–336
Balanced signal, 129–130
Bandwidth, 91
Bandwidth allocation, 531–535
Bandwidth efficiency, 126–127
Bandwidth-delay product, 233, 267, 597
Baran, Paul, 55
Barker sequence, 302
Base station, 19, 70
Base station controller, 171
Base-T Ethernet, 295–296
Base64 encoding, 634
Baseband signal, 91, 130
Baseband transmission, 125–130
Basic bit-map method, 270
Baud rate, 127, 146
BB84 protocol, 773
Beacon frame, 307
Beauty contest, for allocating spectrum, 112
Bell, Alexander Graham, 139
Bell Operating Company, 142
Bellman-Ford routing algorithm, 370
Bent-pipe transponder, 116
Berkeley socket, 500–507
Best-effort service, 318–319
BGP (*see* Border Gateway Protocol)
Big-endian computer, 439
Binary countdown protocol, 272–273
Binary exponential backoff algorithm, 285–286
Binary phase shift keying, 130
Bipolar encoding, 129
Birthday attack, 804–806
Bit rate, 127
Bit stuffing, 199
Bit-map protocol, 270–271
BitTorrent, 750–753
  choked peer, 752
  chunk, 751
  free-rider, 752
  leecher, 752
  seeder, 751
  swarm, 751
  tit-for-tat strategy, 752
  torrent, 750
  tracker, 751
  unchoked peer, 752
Blaatand, Harald, 320
Block cipher, 779
Block code, 204

Bluetooth, 18, 320–327
  adaptive frequency hopping, 324
  applications, 321–322
  architecture, 320–321
  frame structure, 325–327
  link, 324
  link layer, 324–325
  pairing, 320
  piconet, 320
  profile, 321
  protocol stack, 322–323
  radio layer, 324
  scatternet, 320
  secure pairing, 325
  security, 826–827
Bluetooth SIG, 321
BOC (*see* Bell Operating Company)
Body, HTML tag, 625
Border gateway protocol, 432, 479–484
Botnet, 16, 628
Boundary router, 477
BPSK (*see* Binary Phase Shift Keying)
Bridge, 332–342
  backward learning, 335–336
  compared to other devices, 340–342
  learning, 334–337
  spanning tree, 337–340
  use, 332–333
Broadband, 63, 147–151
Broadband wireless, 312–320
Broadcast control channel, 173
Broadcast network, 17
Broadcast routing, 380–382
Broadcast storm, 344, 583
Broadcasting, 17, 283
Browser, 648
  extension, 859–860
  helper application, 654
  plug-in, 653–654, 859
BSC (*see* Base Station Controller)
Bucket, leaky, 397
Bucket brigade attack, 835
Buffering, 222, 238, 290, 341
Bundle, delay-tolerant network, 601
Bundle protocol, 603–605
Bursty traffic, 407
Bush, Vannevar, 647
Business application, 3
Byte stream, reliable, 502
Byte stuffing, 198

C

CA (*see* Certification Authority)
Cable headend, 23, 63, 179
Cable Internet, 180–182
Cable modem, 63, 183–185, 183–195
Cable modem termination system, 63, 183
Cable television, 179–186
Cache
  ARP 468–469
  DNS, 620–622, 848–850
  poisoned, 849
  Web, 656, 690–692
Caesar cipher, 769
Call management, 169
Capacitive coupling, 129
Capacity, channel, 94
CAPTCHA, 16
Care-of address, 387
Carnivore, 15
Carrier extension, Ethernet, 294
Carrier sense multiple access, 72, 266–269
  1-persistent, 266
  collision detection, 268–269
  nonpersistent, 267
  p-persistent, 267
Carrier sensing, 260
Carrier-grade Ethernet, 299
Cascading style sheet, 670–672
Category 3 wiring, 96
Category 5 wiring, 96
Category 6 wiring, 96
Category 7 wiring, 96
CCITT (*see* International Telecommunication Union)
CCK (*see* Complementary Code Keying)
CD (*see* Committee Draft)
CDM (*see* Code Division Multiplexing)
CDMA (*see* Code Division Multiple Access)
CDMA2000, 175
CDN (*see* Content Delivery Network)
Cell, mobile phone, 167, 249
Cell phone, 165
  first generation, 166–170
  second generation, 170–174
  third generation, 65–69, 174–179
Cellular base station, 66
Cellular network, 65
Certificate
  cryptographic, 807–809
  X.509, 809–810

Certificate revocation list, 813
Certification authority, 807
Certification path, 812
CGI (*see* Common Gateway Interface)
Chain of trust, 812
Challenge-response protocol, 828
Channel
  access grant, 174
  broadcast control, 173
  common control, 174
  dedicated control, 174
  erasure, 203
  multiaccess, 257
  paging, 174
  random access, 174, 257
Channel allocation, 258–261
  dynamic, 260–261
Channel capacity, 94
Channel-associated signaling, 155
Checksum, 211
  CRC, 210
  Fletcher's, 212
Chip sequence, CDMA, 136
Choke packet, 399–400
Choked peer, BitTorrent, 752
Chord, 754–757
  finger table, 756
  key, 754
Chosen plaintext attack, 769
Chromatic dispersion, 103
Chrominance, video, 706
Chunk, BitTorrent, 751
CIDR (*see* Classless InterDomain Routing)
Cintent delivery network, 743–748
Cipher, 766
  AES, 783–787
  Caesar, 769
  monoalphabetic substitution, 770
  Rijndael, 784–787
  substitution, 767–770
  symmetric-key, 778–787
  transposition, 771–772
Cipher block chaining mode, 788–789
Cipher feedback mode, 789–790
Cipher modes, 787–792
Ciphertext, 767
Ciphertext-only attack, 769
Circuit, 35
  virtual, 249
Circuit switching, 161–162
Clark, David, 51, 81
Class A network, 450
Class B network, 450
Class C network, 450
Class-based routing, 421
Classful addressing, IP, 449–451
Classic Ethernet, 21, 280, 281–288
Classless interdomain routing, 447–449
Clear to send, 279
Click fraud, 697
Client, 4
Client side on the Web, 649–652
Client side dynamic Web page generation, 676–678
Client side on the Web, 649–652
Client stub, 544
Client-server model, 4
Clipper chip, 861
Clock recovery, 127–129
Cloud computing, 672
CMTS (*see* Cable Modem Termination System)
Coaxial cable, 97–98
Code, cryptographic, 766
Code division multiple access, 66, 108, 135, 170
Code division multiplexing, 135–138
Code rate, 204
Code signing, 858
Codec, 153
Codeword, 204
Collision, 260
Collision detection, CSMA, 268–269
Collision domain, 289
Collision-free protocol, 269–273
Combing, visual artifact, 705
Committee draft, 79
Common control channel, 174
Common gateway interface, 674
Common-channel signaling, 155
Communication medium, 5
Communication satellite, 116–125
Communication security, 813–827
Communication subnet, 24
Community antenna television, 179–180
Companding, 154
Comparison of the OSI and TCP/IP
    models, 49–51
Complementary code keying, 302
Compression
  audio, 701–704
  header, 593–595
  video, 706–712

Computer, wearable, 13
Computer network (*see* Network)
Conditional GET, HTTP, 691
Confidentiality, 35
Congestion, network, 35, 392–404
Congestion avoidance, 398
Congestion collapse, 393
  TCP, 572
Congestion control
  convergence, 534–535
  network layer, 392–404
  provisioning, 395
  TCP, 571–581
Congestion window, TCP, 571
Connection, HTTP, 684–686
Connection establishment, 512–517
  TCP, 560–562
Connection management, TCP, 562–565
Connection release, 517–522
  TCP, 562
Connection reuse, HTTP, 684
Connection-oriented service, 35–38, 359–361
  implementation, 359–361
Connectionless service, 35–38, 358–359
  implementation, 358–359
Connectivity, 6, 11
Constellation, 146
Constellation diagram, 131
Constraint length, 207
Contact, delay-tolerant network, 601
Content and Internet traffic, 7360738
Content delivery network, 743–748
Content distribution, 734–757
Content transformation, 694
Contention system, 262
Continuous media, 699
Control channel, broadcast, 173
Control law, 536
Convergence, 372
  congestion control, 534–535
Convolutional code, 207
Cookie, 15
  SYN, 561
  Web, 658–662
Copyright, 867–869
Cordless telephone, 165
Core network, 66
Core-based tree, 384
Count-to-infinity problem, 372–373
Counter mode, 791–792
Crash recovery, 527–530
CRC (*see* Cyclic Redundancy Check)
Critique of OSI and TCP/IP, 51–53
CRL (*see* Certificate Revocation List)
Cross-correlation, 176
Cryptanalysis, 768, 792–793
  differential, 792–793
  linear, 793
Cryptographic certificate, 807–809
Cryptographic key, 767
Cryptographic principles, 776–778
Cryptographic round, 780
Cryptography, 766–797
  AES, 312
  certificate, 807–809
  ciphertext, 767
  DES, 780–784
  Kerckhoff's principle, 768
  key, 767
  one-time pad, 772–773
  P-box, 779
  plaintext, 767
  public-key, 793–797
  quantum, 773–776
  Rijndael, 312
  S-box, 779
  security by obscurity, 768
  symmetric-key, 778–793
  triple DES, 782–783
  vs. code, 766
  work factor, 768
Cryptology, 768
CSMA (*see* Carrier Sense Multiple Access)
CSMA with collision avoidance, 303
CSMA with collision detection, 268
CSMA/CA (*see* CSMA with Collision Avoidance)
CSMA/CD (*see* CSMA with Collision Detection)
CSNET, 59
CSS (*see* Cascading Style Sheet)
CTS (*see* Clear To Send)
CubeSat, 123
Cumulative acknowledgement, 238, 558
  TCP, 568
Custody transfer, delay-tolerant network, 604
Cut-through switching, 36, 336
Cybersquatting, 614
Cyclic redundancy check, 212
Cypherpunk remailer, 862

D

D-AMPS (*see* Digital Advanced Mobile Phone System)
DAC (*see* Digital-to-Analog Converter)
Daemen, Joan, 784
Daemon, 554
DAG (*see* Directed Acyclic Graph)
Data center, 64
Data delivery service, IEEE 802.11, 312
Data encryption standard, 780–784
Data frame, 43
Data link layer, 43, 193–251
  bit stuffing, 199
  byte stuffing, 197
  design issues, 194–215
  elementary protocols, 215–244
  example protocols, 244–250
  sliding window protocols, 226–244
  stop-and-wait protocol, 222–223
Data link layer switching, 332–349
Data link protocol, 215–250
  ADSL, 248–250
  elementary, 215–244
  examples, 215–250
  packet over SONET, 245–248
  sliding window, 226–244
  stop-and-wait, 222–223
Data over cable service interface specification, 183
Datagram, 37, 358
Datagram congestion control protocol, 503
Datagram network, 358
Datagram service, comparison with VCs, 361–362
Davies, Donald, 56
DB (*see* Decibel)
DCCP (*see* Datagram Congestion Controlled Protocol)
DCF (*see* Distributed Coordination Function)
DCF interframe spacing, 308
DCT (*see* Discrete Cosine Transformation)
DDoS attack (*see* Distributed Denial of Service attack)
De facto standard, 76
De jure standard, 76
Decibel, 94, 699
Decoding, audio, 701
Dedicated control channel, 174
Deep Web, 696
Default gateway, 469
Default-free zone, 446
Deficit round robin, 414
Delay, queueing, 164
Delay-tolerant network, 599–605
  architecture, 600–603
  custody transfer, 604
  protocol, 603–605
Delayed acknowledgement, TCP, 566
Demilitarized zone, 819
Denial of service attack, 820
  distributed, 821–822
Dense wavelength division multiplexing, 160
DES (*see* Data Encryption Standard)
Design issues
  data link layer, 194–202
  fast networks, 586–590
  network, 33–35
  network layer, 355–362
  transport layer, 507–530
Designated router, 375, 478
Desktop sharing, 5
Destination port, 453–454
Device driver, 215
DHCP (*see* Dynamic Host Configuration Protocol)
DHT (*see* Distributed Hash Table)
Diagonal basis, in quantum cryptography, 774
Dial-up modem, 62
Dialog control, 44
Differential cryptanalysis, 792–793
Differentiated service, 421–424, 440, 458
Diffie-Hellman protocol, 833–835
DIFS (*see* DCF InterFrame Spacing)
Digital advanced mobile phone system, 170
Digital audio, 699–704
Digital Millennium Copyright Act, 14, 868
Digital modulation, 125
Digital signature, 797–806
Digital signature standard, 800
Digital subscriber line, 62, 147–151
Digital subscriber line access multiplexer, 62, 150
Digital video, 704–712
Digital-to-analog converter, 700
Digitizing voice signals, 153–154
Digram, 770
Dijkstra, Edsger, 367
Dijkstra's algorithm, 369
Direct acyclic graph, 365
Direct sequence spread spectrum, 108
Directed acyclic graph, 365
Directive, HTML, 663
Directory, PKI, 812

DIS (*see* Draft International Standard)
Disassociation, IEEE 802.11, 311
Discovery, path MTU, 556
Discrete cosine transformation, MPEG, 707
Discrete multitone, 148
Dispersion, chromatic, 103
Disposition, message, 628
Disruption-tolerant network, 600
Distance vector multicast routing protocol, 383
Distance vector routing, 370–378
Distortion, 700
Distributed coordination function, 304
Distributed denial of service attack, 821–822
Distributed Hash Table, 753–757
Distributed system, 2
Distribution service, IEEE 802.11, 311
Distribution system, 299
DIX Ethernet standard, 281, 283
DMCA (*see* Digital Millennium Copyright Act)
DMCA takedown notice, 14
DMT (*see* Discrete MultiTone)
DMZ (*see* DeMilitarized Zone)
DNS (*see* Domain Name System)
DNS Name Space, 612–616
DNS spoofing, 848–850
DNSsec (*see* Domain Name System security)
DOCSIS (*see* Data Over Cable Service Interface Specification)
Document object model, 679
DOM (*see* Document Object Model)
Domain
  collision, 289
  frequency, 133
Domain Name System, 59, 611–623
  authoritative record, 620–622
  cybersquatting, 614
  domain resource record, 616–619
  name server, 619–623
  name space, 613
  registrar, 613
  resource record, 616–619
  reverse lookup, 617
  spoofing, 851
  top-level domain, 613
  zone, 851–852
DoS attack (*see* Denial of Service attack)
Dot com era, 647
Dotted decimal notation, 443
Downstream proxy, 742
Draft International Standard, 79
Draft standard, 82
DSL (*see* Digital Subscriber Line)
DSLAM (*see* Digital Subscriber Line Access Multiplexer)
DTN (*see* Delay-Tolerant Network)
Duplicate acknowledgement, TCP, 577
DVMRP (*see* Distance Vector Multicast Routing Protocol)
DWDM (*see* Dense Wavelength Division Multiplexing)
Dwell time, 324
Dynamic channel allocation, 260–261
Dynamic frequency selection, 312
Dynamic host configuration protocol, 470
Dynamic HTML, 676
Dynamic page, Web, 649
Dynamic routing, 364
Dynamic Web page, 649, 672–673
Dynamic Web page generation
  client side, 676–678
  server side, 673–676

E

E-commerce, 6, 9
E-mail (*see* Email)
E1 line, 155
EAP (*see* Extensible Authentication Protocol)
Early exit, 484
ECB (*see* Electronic Code Book mode)
ECMP (*see* Equal Cost MultiPath)
ECN (*see* Explicit Congestion Notification)
EDE (*see* Encrypt Decrypt Encrypt mode)
EDGE (*see* Enhanced Data rates for GSM Evolution)
EEE (*see* Encrypt Encrypt Encrypt mode)
EIFS (*see* Extended InterFrame Spacing)
Eisenhower, Dwight, 56
Electromagnetic spectrum, 105–109, 111–114
Electronic code book mode, 787–788
Electronic commerce, 6
Electronic mail (*see* Email)
Electronic product code, 327
Elephant flow, 737
Email, 5, 623–646
  architecture and services, 624–626
  authoritative record, 620
  base64 encoding, 634
  body, 625
  cached record, 620

Email (*continued*)
envelope, 625
final delivery, 643
IMAP, 644–645
mail server, 624
mail submission, 641
mailbox, 625
message format, 630
message transfer, 624, 637–643, 642
MIME, 633
name resolution, 620
open mail relay, 642
POP3, 644
quoted-printable encoding, 634
signature block, 629
simple mail transfer protocol, 625
transfer agent, 624–625
user agent, 624, 626
vacation agent, 629
Webmail, 645–646
X400, 629
Email header, 625
Email reader, 626
Email security, 841–846
Emoticon, 623
Encapsulating security payload, 817
Encapsulation, packet, 387
Encoding, 4B/5B, 292
audio, 701–704
video, 706–712
Ethernet 4B/5B, 292
Ethernet 8B/10B, 295
Ethernet 64B/66B, 297
Encrypt decrypt encrypt mode, 782
Encrypt encrypt encrypt mode, 782
Encryption, link, 765
End office, 140
End-to-end argument, 357, 523
Endpoint, multiplexing, 527
Enhanced data rates for GSM evolution, 178
Enterprise network, 19
Entity, transport, 496
Envelope, 625
EPC (*see* Electronic Product Code)
EPON (*see* Ethernet PON)
Equal cost multipath, 476
Erasure, 716
Erasure channel, 203
Error control, 200–201
transport layer, 522–527
Error correction, 34
Error detection, 33
Error syndrome, 207
Error-correcting code, 204–209
Error-detecting code, 209–215
ESMTP (*see* Extended SMTP)
ESP (*see* Encapsulating Security Payload)
Eternity service, 864
Ethernet, 20, 280–299
10-gigabit, 296–297
100Base-FX, 292
100Base-T4, 292
1000Base-T, 295–296
Base-T, 295–296
carrier-grade, 299
classic, 21, 281–288
DIX, 281, 283
fast, 290–293
gigabit, 293–296
MAC sublayer, 280–299
promiscuous mode, 290
retrospective, 298–299
switched, 20, 288–290
Ethernet carrier extension, 294
Ethernet encoding
4B/5B, 292
64B/66B, 297
8B/10B, 295
Ethernet frame bursting, 294
Ethernet header, 282
Ethernet hub, 288
Ethernet jumbo frame, 296
Ethernet performance, 286–288
Ethernet PON, 151–152
Ethernet port, 20
Ethernet switch, 20, 289
EuroDOCSIS, 183
EWMA (*see* Exponentially Weighted Moving Average)
Expedited forwarding, 422–423
Explicit congestion notification, 400
Exponential decay, 738
Exponentially weighted moving average, 399, 570
Exposed terminal problem, 278–280
Extended hypertext markup language, 681
Extended interframe spacing, 309
Extended SMTP, 639
Extended superframe, 154
Extensible authentication protocol, 824

Extensible markup language, 680
Extensible stylesheet language transformation, 681
Extension header, IPv6, 461–463
Exterior gateway protocol, 431, 474

F

Facebook, 8
Fair queueing, 412
Fair use doctrine, 869
Fast Ethernet, 290–293
Fast network, design, 586–590
Fast recovery, TCP, 578
Fast retransmission, TCP, 577
Fast segment processing, 590–593
FCFS (*see* First-Come First-Served packet scheduling)
FDD (*see* Frequency Division Duplex)
FDDI (*see* Fiber Distributed Data Interface)
FDM (*see* Frequency Division Multiplexing)
FEC (*see* Forwarding Equivalence Class)
FEC (*see* Forward Error Correction)
Fiber distributed data interface, 272
Fiber node, 180
Fiber optics, 99–105
  compared to copper wire, 104–105
Fiber to the home, 63, 100, 151
Fibre channel, 298
Field, video, 705
FIFO (*see* First-In First-Out packet scheduling)
File transfer protocol, 455, 623
Filtering, ingress, 487
Final delivery, email, 643
Finger table, Chord, 756
Firewall, 818–821
  stateful, 819
First-come first-served packet scheduling, 412
First-generation mobile phone network, 166–170
First-in first-out packet scheduling, 412
Fixed wireless, 11
Flag byte, 198
Flash crowd, 747, 748
Fletcher's checksum, 212
Flooding algorithm, 368–370
Flow control, 35, 201–202
  transport layer, 522–527
Flow specification, 416
Footprint, satellite, 119
Forbidden region, 514–515
Foreign agent, 388, 487
Form, Web, 667–670
Forward error correction, 203, 715
Forwarding, 363
  assured, 423–424
  expedited, 422–423
Forwarding algorithm, 27
Forwarding equivalence class, 472
Fourier analysis, 90
Fourier series, 90
Fourier transform, 135, 702
Fragment
  frame, 307
  packet, 433
Fragmentation, packet, 432–436
Frame, 194
  acknowledgement, 43
  beacon, 307
  data, 43
Frame bursting, Ethernet, 294
Frame fragment, 307
Frame header, 218
Frame structure
  Bluetooth, 325–327
  IEEE 802.11, 309–310
  IEEE 802.16, 319–320
Framing, 197–201
Free-rider, BitTorrent, 752
Free-space optics, 114
Freedom of speech, 863–865
Frequency, electromagnetic spectrum, 106
Frequency division duplex, 169, 317
Frequency division multiplexing, 132–135
Frequency hopping, Bluetooth, 324
Frequency hopping spread spectrum, 107
Frequency masking, psychoacoustics, 703
Frequency reuse, 65
Frequency selection, dynamic, 312
Frequency shift keying, 130
Freshness of messages, 778
Front end, Web server, 739
FSK (*see* Frequency Shift Keying)
FTP (*see* File Transfer Program)
FttH (*see* Fiber to the Home)
Full-duplex link, 97
Future of TCP, 581–582
Fuzzball, 59

G

G.711 standard, 728
G.dmt, 149
G.lite, 150
Gatekeeper, multimedia, 728
Gateway, 28
  application level, 819
  default, 469
  media, 68
  multimedia, 728
Gateway GPRS support node, 68
Gateway mobile switching center, 68
Gen 2 RFID, 327–331
General packet radio service, 68
Generator polynomial, 213
GEO (*see* Geostationary Earth Orbit)
Geostationary earth orbit, 117
Geostationary satellite, 117
GGSN (*see* Gateway GPRS Support Node)
Gigabit Ethernet, 293–296
Gigabit-capable PON, 151
Global Positioning System, 12, 121
Global system for mobile communication, 65, 170–174
Globalstar, 122
Gmail, 15
GMSC (*see* Gateway Mobile Switching Center)
Go-back-n protocol, 232–239
Goodput, 393, 531
GPON (*see* Gigabit-capable PON)
GPRS (*see* General Packet Radio Service)
GPS (*see* Global Positioning System)
Gratuitous ARP, 469, 487
Gray, Elisha, 139
Gray code, 132 Group, 153
Group, telephone hierarchy, 153
GSM (*see* Global System for Mobile communication)
Guard band, 133
Guard time, 135
Guided transmission media, 85–105, 95–105

H

H.225 standard, 729
H.245 standard, 729
H.264 standard, 710
H.323
  compared to SIP, 733–734
  standard, 728–731
Half-duplex link, 97
Hamming code, 206–207
Hamming distance, 205
Handoff, 68–69, 168
  hard, 178
  soft, 178
Handover (*see* Handoff)
Hard-decision decoding, 208
Harmonic, 90
Hashed message authentication code, 817
HDLC (*see* High-level Data Link Control)
HDTV (*see* High-Definition TeleVision)
Headend, cable, 63, 179
Header, 625
  email, 625
  Ethernet, 282
  IPv4, 439–442
  IPv6, 458–463
  IPv6 extension, 461–463
  packet, 31
  TCP segment, 557–560
Header compression, 593–595
  robust, 594
Header prediction, 592
Helper application, browser, 654
Hertz, 106
Hertz, Heinrich, 106
HF RFID, 74
HFC (*see* Hybrid Fiber Coax)
Hidden terminal problem, 278
Hierarchical routing, 378–380
High-definition television, 705
High-level data link control, 199, 246
High-water mark, 719
History of the Internet, 54–61
HLR (*see* Home Location Register)
HMAC (*see* Hashed Message Authentication Code)
Home agent, 387, 486
Home location, 386
Home location register, 171
Home network, 6–10
Home subscriber server, 69
Hop-by-hop backpressure, 400–401
Host, 23
  mobile, 386
Host design for fast networks, 586–590
Hosting, 64

Hot-potato routing, 484
Hotspot, 11
HSS (*see* Home Subscriber Server)
HTML(*see* HyperText Markup Language)
HTTP (*see* HyperText Transfer Protocol)
HTTPS (*see* Secure HyperText Transfer Protocol)
Hub, 340–342
  compared to bridge and switch, 340–342
  Ethernet, 288
  satellite, 119
Hybrid fiber coax, 180
Hyperlink, 648
Hypertext, 647
Hypertext markup language, 663–670
  attribute, 664
  directive, 663
  tag, 663–666
Hypertext transfer protocol, 45, 649, 651, 683–693
  conditional get, 691
  connection, 684–686
  connection reuse, 684
  method, 686–688
  parallel connection, 686
  persistent connection, 684
  scheme, 650
  secure 853
Hz (*see* Hertz)

I

IAB (*see* Internet Activities Board)
ICANN (*see* Internet Corporation for Assigned Names and Number)
ICMP (*see* Internet Control Message Protocol)
IDEA (*see* International Data Encryption Algorithm)
IEEE 802.11, 19, 299–312
  architecture, 299–301
  association, 311
  authentication, 311
  comparison with IEEE 802.16, 313–314
  data delivery service, 312
  disassociation, 311
  distribution service, 311
  frame structure, 309–310
  integration service, 312
  MAC sublayer, 299–312
  MAC sublayer protocol, 303–309
  physical layer, 301–303
  privacy service, 312
  procotol stack, 299–301
  reassociation, 311
  security, 823–826
  services, 311–312
  transmit power control, 312
IEEE 802.11a, 302
IEEE 802.11b, 17, 301–302
IEEE 802.11g, 203
IEEE 802.11i, 823
IEEE 802.11n, 302–303
IEEE 802.16, 179, 313–320
  architecture, 314–315
  comparison with IEEE 802.11, 313–314
  frame structure, 319–320
  MAC sublayer protocol, 317–319
  physical layer, 316–317
  protocol stack, 314–315
  ranging, 317
IEEE 802.1Q, 346–349
IEEE 802.1X, 824
IEEE(*see* Institute of Electrical and Electronics Engineers)
IETF (*see* Internet Engineering Task Force)
IGMP (*see* Internet Group Management Protocol)
IKE (*see* Internet Key Exchange)
IMAP (*see* Internet Message Access Protocol
IMP (*see* Interface Message Processor)
Improved mobile telephone system, 166
IMT-2000, 174–175
IMTS (*see* Improved Mobile Telephone System)
In-band signaling, 155
Industrial, scientific, medical bands, 70, 112
Inetd, 554
Infrared Data Association, 114
Infrared transmission, 114
Ingress filtering, 487
Initial connection protocol, 511
Initialization vector, 788
Input form, Web, 667–670
Instant messaging, 8
Institute of Electrical and Electronics Engineers, 79
Integrated services, 418–421
Integration service, IEEE 802.11, 312
Intellectual property, 867
Interdomain protocol, 431
Interdomain routing, 474
Interexchange carrier, 143

Interface, 30
  air, 66, 171
Interface message processor, 56–57
Interior gateway protocol, 431, 474
Interlacing, video, 705
Interleaving, 716
Internal router, 476
International data encryption algorithm, 842
International Mobile Telecommunication-2000, 174–175
International Standard, 78–79
International Standard IS-95, 170
International Standards Organization, 78
International Telecommunication Union, 77
Internet, 2, 28
  architecture, 61–64
  backbone, 63
  cable, 180–182
  control protocols, 465–470
  daemon, 554
  history, 54–61
  interplanetary, 18
  key exchange, 815
  message access protocol, 644–645
  multicasting, 484–488
  protocol version 4, 439–455
  protocol version 6, 455–465
  radio, 721
  TCP/IP layer, 46–47
Internet Activities Board, 81
Internet Architecture Board, 81
Internet control message protocol, 47
Internet Corporation for Assigned Names and Numbers, 444, 612
Internet Engineering Task Force, 81
Internet exchange point, 63, 480
Internet group management protocol, 485
Internet over cable, 180–182
Internet protocol (IP), 47, 438–488
  CIDR, 447–449
  classful addressing, 449–451
  control, 465–470
  control message, 47
  control protocols, 465–470
  dotted decimal notation, 443
  group management, 485
  IP addresses, 442–455
  message access, 644–645
  mobile, 485–488
  subnet, 444–446
Internet protocol (*continued*)
  version 4, 439–442
  version 5, 439
  version 6, 455–465
  version 6 controversies, 463–465
  version 6 extension headers, 461–463
  version 6 main header, 458–461
Internet protocol version 4, 439–455
Internet protocol version 6, 455–465
Internet Research Task Force, 81
Internet service provider, 26, 62
Internet Society, 81
Internet standard, 82
Internet telephony, 698, 725
Internetwork, 25, 28–29, 424–436
Internetwork routing, 431–432
Internetworking, 34, 424–436
  network layer, 19, 424–436
Internetworking network layer, 424–436
Interoffice trunk, 141
Interplanetary Internet, 18
Intradomain protocol, 431
Intradomain routing, 474
Intranet, 64
Intruder, security, 767
Inverse multiplexing, 527
IP (*see* Internet protocol)
IP address, 442–455
  CIDR, 447–449
  classful, 449–451
  NAT, 451–455
  prefix, 443–444
IP security, 814–818
  transport mode, 815
  tunnel mode, 815
IP telephony, 5
IP television, 9, 721
IPsec (*see* IP security)
IPTV (*see* IP TeleVision)
IPv4 (*see* Internet Protocol, version 4)
IPv5 (*see* Internet Protocol, version 5)
IPv6 (*see* Internet Protocol, version 6)
IrDA (*see* Infrared Data Association)
Iridium, 121
IRTF (*see* Internet Research Task Force)
IS (*see* International Standard)
IS-95, 170
IS-IS, 378, 474
ISAKMP (*see* Internet Security Association and Key Management Protocol)

ISM (*see* Industrial, Scientific, Medical bands)
ISO (*see* International Standards Organization)
ISP (*see* Internet Service Provider)
ISP network, 26
Iterative query, 622
ITU (*see* International Telecommunication Union)
IV (*see* Initialization Vector)
IXC (*see* IntereXchange Carrier)
IXP (*see* Internet eXchange Point)

J

Java applet security, 857–858
Java virtual machine, 678
Java Virtual Machine, 857
JavaScript, 676, 859
Javaserver page, 675
Jitter, 406, 698
Jitter control, 550–552
Joint photographic experts group, 706
JPEG (*see* Joint Photographic Experts Group)
JPEG standard, 706–709
JSP (*see* JavaServer Page)
Jumbo frame, Ethernet, 296
Jumbogram, 462
JVM (*see* Java Virtual Machine)

K

Karn's algorithm, 571
KDC (*see* Key Distribution Center)
Keepalive timer, TCP, 571
Kepler's Law, 116
Kerberos, 838–840
  realm, 840
Kerckhoff's principle, 768
Key
  Chord, 754
  cryptographic, 767
Key distribution center, 807, 828
  authentication using, 835–836
Key escrow, 861
Keystream, 790
Keystream reuse attack, 791
Known plaintext attack, 769

L

L2CAP (*see* Logical Link Control Adaptation Protocol)
Label edge router, 472
Label switched router, 472
Label switching, 360, 470–474
Lamarr, Hedy, 107–108
LAN (*see* Local Area Network)
LAN, virtual, 21
LATA (*see* Local Access and Transport Area)
Layer
  application, 45, 47–48, 611–758
  data link, 193–251
  IEEE 802.11 physical, 301–303
  IEEE 802.16 physical, 316–317
  network, 29, 355–489
  physical, 89–187
  session, 44–45
  transport, 44, 495–606
LCP (*see* Link Control Protocol)
LDPC (*see* Low-Density Parity Check)
Leaky bucket algorithm, 397, 407–411
Learning bridge, 334–337
LEC (*see* Local Exchange Carrier)
Leecher, BitTorrent, 752
LEO (*see* Low-Earth Orbit satellite)
LER (*see* Label Edge Router)
Level, network, 29
Light transmission, 114–116
Limited-contention protocol, 274–277
Line code, 126
Linear code, 204
Linear cryptanalysis, 793
Link
  asynchronous connectionless, 325
  Bluetooth, 324
  full-duplex, 97
  half-duplex, 97
  synchronous connection-oriented, 325
Link control protocol, 245
Link encryption, 765
Link layer
  Bluetooth, 324–325
  TCP/IP. 46
Link state routing, 373–378
Little-endian computer, 429
LLC (*see* Logical Link Control)
Load balancing, Web server, 740
Load shedding, 395, 401–403

Load-shedding policy
  milk, 401
  wine, 401
Local access and transport area, 142
Local area network, 19
  virtual, 342–349
Local central office, 21
Local exchange carrier, 142
Local loop, 140, 144–152
Local number portability, 144
Logical link control, 283, 310
Logical link control adaptation protocol, 323
Long fat network, 595–599
Long-term evolution, 69, 179, 314
Longest matching prefix algorithm, 448
Lossless audio compression, 701
Lossy audio compression, 701
Lottery algorithm, 112
Low-density parity check, 209
Low-earth orbit satellite, 121, 121–123
Low-water mark, 718
LSR (*see* Label Switched Router)
LTE (*see* Long Term Evolution)
Luminance, video, 706

M

M-commerce, 12
MAC (*see* Medium Access Control)
MAC sublayer protocol, 303–309, 317–319
  IEEE 802.11, 303–309
  IEEE 802.16, 317–319
MACA (*see* Multiple Access with Collision Avoidance)
Macroblock, MPEG, 711
Magnetic media, 95–96
MAHO (*see* Mobile Assisted HandOff)
Mail server, 624
Mail submission, 624, 637, 641
Mailbox, 625
Mailing list, 625
Maintenance, route, 391–392
MAN (*see* Metropolitan Area Network)
Man-in-the-middle attack, 835
Manchester encoding, 127
MANET (*see* Mobile Ad hoc NETwork)
Markup language, 663, 680
Marshaling parameters, 544
Max-min fairness, 532–534
Maximum segment size, TCP, 559
Maximum transfer unit, 433, 556
Maximum transmission unit path, 433
Maxwell, James Clerk, 105
MCI, 110
MD5, 804–807
Measurements of network performance, 584–586
Media gateway, 68
Medium access control sublayer, 43, 257–350, 320–327
  Bluetooth, 320–327
  broadband wireless, 312–320
  channel allocation, 258–261
  Ethernet, 280–299
  IEEE 802.11, 299–312
  multiple access protocols, 261–280
  wireless LANs, 299–312
Medium-earth orbit satellite, 121
MEO (*see* Medium-Earth Orbit Satellite)
Merkle, Ralph, 796
Message digest, 800–801
Message disposition, 628
Message format, 630
Message header, 688–690
Message integrity check, 825
Message switching, 600
Message transfer, 637–643, 642
Message transfer agent, 624
Metafile, 714
Metcalfe, Robert, 6
Method, HTTP, 686–688
Metric units, 82–83
Metropolitan area network, 23
MFJ (*see* Modified Final Judgment)
MGW (*see* Media GateWay)
MIC (*see* Message Integrity Check)
Michelson-Morley experiment, 281
Microcell, 168
Microwave transmission, 110–114
Middlebox, 740
Middleware, 2
Milk, load-shedding policy, 401
MIME (*see* Multipurpose Internet Mail Extension)
MIME type, 652–655
MIMO (*see* Multiple Input Multiple Output)
Minislot, 184
Mirroring a Web site, 744
Mobile ad hoc network, 389–392
Mobile assisted handoff, 174

Mobile code security, 857–860
Mobile commerce, 12
Mobile host, 386
  routing, 386–389
Mobile Internet protocol, 485–488
Mobile IP protocol, 485–488
Mobile phone, 164–179
Mobile phone system
  first-generation, 166–170
  second-generation, 170–174
  third-generation, 65–69, 174–179
Mobile switching center, 68, 168
Mobile telephone, 164–179
Mobile telephone switching office, 168
Mobile telephone system, 164–179
Mobile user, 10–13
Mobile Web, 693–695
Mobile wireless, 11
Mockapetris, Paul, 52
Modem, 145
  cable, 63, 183–195
  dial-up, 62
  V.32, 146
  V.34, 146
  V.90, 147
Modified final judgment, 142
Modulation, 130–132
  amplitude shift keying, 130–131
  BPSK, 302
  digital, 125
  discrete multitone, 149
  frequency shift keying, 130–131
  phase shift keying, 130–131
  pulse code, 153
  quadrature phase shift keying, 131
  trellis coded, 146
Monoalphabetic substitution cipher, 770
MOSPF (*see* Multicast OSPF)
Motion picture experts group, 709
Mouse, 737
Mouse flow, 737
MP3, 702
MP4, 702
MPEG (*see* Motion Picture Experts Group)
MPEG compression, 709–712
  frame types, 712
  MPEG-1, 710
  MPEG-2, 710
  MPEG-4, 710
MPEG standards, 709–710
MPLS (*see* MultiProtocol Label Switching)
MSC (*see* Mobile Switching Center)
MSS (*see* Maximum Segment Size)
MTSO (*see* Mobile Telephone Switching Office)
MTU (*see* Maximum Transfer Unit)
Mu law, 700
Mu-law, 153
Multiaccess channel, 257
Multiaccess network, 475
Multicast OSPF, 383
Multicast routing, 382, 382–385
Multicasting, 17, 283, 382
  Internet, 484–488
Multidestination routing, 380
Multihoming, 481
Multihop network, 75
Multimedia, 697–734, 699
  Internet telephony, 728–734
  jitter control, 550–552
  live streaming, 721–724
  MP3, 702–704
  RTSP, 715
  streaming audio, 699–704
  video on demand, 713–720
  videoconferencing, 725–728
  voice over IP, 728–734
Multimode fiber, 101
Multipath fading, 70, 111
Multiple access protocol, 261–280
Multiple access with collision avoidance, 279
Multiple input multiple output, 303
Multiplexing, 125, 152–160
  endpoint, 527
  inverse, 527
  statistical, 34
Multiprotocol label switching, 357, 360, 471–474
Multiprotocol router, 429
Multipurpose internet mail extension, 632–637
Multithreaded Web server, 656
Multitone, discrete, 148
Mutlimedia, streaming video, 704–712

N

Nagle's algorithm, 566
Name resolution, 620
Name server, 619–623
Naming (*see* Addressing)

Naming, secure, 848–853
NAP (*see* Network Access Point)
NAT (*see* Network Address Translation)
NAT box, 453
National Institute of Standards and Technology, 79, 783
National Security Agency, 782
NAV (*see* Network Allocation Vector)
NCP (*see* Network Control Protocol)
Near field communication, 12
Needham-Schroeder authentication, 836–838
Negotiation protocol, 35
Network
  ad hoc, 70, 299, 389–392
  ALOHA, 72
  broadcast, 17
  cellular, 65
  delay-tolerant, 599–605
  enterprise, 19
  first-generation mobile phone, 166–170
  home, 6–10
  local area, 19
  metropolitan area, 23
  multiaccess, 475
  multihop, 75
  passive optical, 151
  peer-to-peer, 7
  performance, 582–599
  personal area, 18
  point-to-point, 17
  power-line, 10, 22
  public switched telephone, 68, 139
  scalable, 34
  second-generation mobile phone, 170–174
  sensor, 13, 73–75
  social, 8
  stub, 481
  third-generation mobile phone, 65–69, 174–179
  uses, 3–16
  virtual circuit, 358
  virtual private, 4, 26
  wide area, 23
Network accelerator, 215
Network access point, 60–61
Network address translation, 452–455
Network allocation vector, 305
Network architecture, 31
Network control protocol, 245
Network design issues, 33–35
Network hardware, 17–29
Network interface card, 202, 215
Network interface device, 149
Network layer, 43–44, 355–489
  congestion control, 392–404
  design issues, 355–362
  Internet, 436–488
  internetworking, 424–436
  quality of service, 404–424
  routing algorithms, 362–392
Network neutrality, 14
Network overlay, 430
Network performance measurement, 584–586
Network protocol (*see* Protocol)
Network service access point, 509
Network service provider, 26
Network software, 29–41
Network standardization, 75–82
NFC (*see* Near Field Communication)
NIC (*see* Network Interface Card)
NID (*see* Network Interface Device)
NIST (*see* National Institute of Standards and Technology)
Node B, 66
Node identifier, 754
Non-return-to-zero inverted encoding, 127
Non-return-to-zero modulation, 125
Nonadaptive algorithm, 363–364
Nonce, 824
Nonpersistent CSMA, 267
Nonpersistent Web cookie, 659
Nonrepudiation, 798
Notification, explicit congestion, 400
NRZ (*see* Non-Return-to-Zero modulation)
NRZI (*see* Non-Return-to-Zero Inverted encoding)
NSA (*see* National Security Agency)
NSAP (*see* Network Service Access Point)
NSFNET, 59–61
Nyquist frequency, 94, 146, 153

O

OFDM (*see* Orthogonal Frequency Division Multiplexing)
OFDMA (*see* Orthogonal Frequency Division Multiple Access)
One-time pad, 772–773
Onion routing, 863
Open mail relay, 642

Open shortest path first, 378
Open shortest path first routing, 474–479
Open Systems Interconnection, 41–45
  comparison with TCP/IP, 49–51
Open Systems Interconnection,
  application layer, 45
  critique, 51–53
  data link layer, 43
  network layer, 43–44
  physical layer, 43
  presentation layer, 45
  reference model, 41–45
  session layer, 44–45
  transport layer, 44
Optimality principle, 364–365
Organizationally unique identifier, 283
Orthogonal frequency division multiple access, 316
Orthogonal frequency division multiplexing, 71, 133–134, 302
Orthogonal sequence, 136
OSI (*see* Open Systems Interconnection)
OSPF (*see* Open Shortest Path First routing)
OSPF(*see* Open Shortest Path First routing)
OUI (*see* Organizationally Unique Identifier)
Out-of-band signaling, 155
Overlay, 430
  network, 430
Overprovisioning, 404

P

P-box, cryptographic, 779
P-frame, 611–712
P-persistent CSMA, 267
P2P (*see* Peer-to-peer network)
Packet, 17, 36
Packet encapsulation, 387
Packet filter, 819
Packet fragmentation, 432–436, 433
Packet header, 31
Packet over SONET, 245–248
Packet scheduling, 411–414
Packet switching, 162–164
  store-and-forward, 356
Packet train, 736
Page, Web, 647
Paging channel, 174
Pairing, Bluetooth, 320
PAN (*see* Personal Area Network)
PAR (*see* Positive Acknowledgement with Retransmission protocol)
Parallel connection, HTTP, 686
Parameters, marshaling, 544
Parity bit, 210
Parity packet, 716
Passband, 91
Passband signal, 130
Passband transmission, 125, 130–132
Passive optical network, 151
Passive RFID, 73
Passkey, 826
Path,
  autonomous system, 481
  certification, 812
  maximum transmission unit, 433
Path diversity, 70
Path loss, 109
Path maximum transmission unit discovery, 433
Path MTU discovery, 556
Path vector protocol, 481
PAWS (*see* Protection Against Wrapped Sequence numbers)
PCF (*see* Point Coordination Function)
PCM (*see* Pulse Code Modulation)
PCS (*see* Personal Communications Service)
Peer, 29, 63
Peer-to-peer network, 7, 14, 735, 748–753
  BitTorrent, 750–753
  content distribution, 750–753
Peering, 481
Per hop behavior, 421
Perceptual coding, 702
Performance, TCP, 582–599
Performance issues in networks, 582–599
Performance measurements, network, 584–586
Perlman, Radia, 339
Persistence timer, TCP, 571
Persistent and nonpersistent CSMA, 266–268
Persistent connection, HTTP, 684
Persistent cookie, Web, 659
Personal area network, 18
Personal communications service, 170
PGP (*see* Pretty Good Privacy)
Phase shift keying, 130
Phishing, 16
Phone (*see* Telephone)
Photon, 774–776, 872
PHP, 674

Physical layer, 89–187
  cable television, 179–186
  code division multiplexing, 135–138
  communication satellites, 116–125
  fiber optics, 99–104
  frequency division multiplexing, 132–135
  IEEE 802.11, 301–303
  IEEE 802.16, 316–317
  mobile telephones, 164–179
  modulation, 125–135
  Open Systems Interconnection, 43
  telephone system, 138–164
  time division multiplexing, 135
  twisted pairs, 96–98
  wireless transmission, 105–116
Physical medium, 30
Piconet, Bluetooth, 320
Piggybacking, 226
PIM (*see* Protocol Independent Multicast)
Ping, 467
Pipelining, 233
Pixel, 704
PKI (*see* Public Key Infrastructure)
Plain old telephone service, 148
Plaintext, 767
Playback point, 551
Plug-in, browser, 653, 859
Podcast, 721
Poem, spanning tree, 339
Point coordination function, 304
Point of presence, 63, 143
Point-to-point network, 17
Point-to-point protocol, 198, 245
Poisoned cache, 849
Poisson model, 260
Polynomial generator, 213
Polynomial code, 212
PON (*see* Passive Optical Network)
POP (*see* Point of Presence)
POP3 (*see* Post Office Protocol)
Port
  destination, 453–454
  Ethernet, 20
  source, 453
  TCP, 553
  transport layer, 509
  UDP, 542
Portmapper, 510
Positive acknowledgement with retransmision protocol, 225
Post, Telegraph & Telephone administration, 77
Post office protocol, version 3, 644
POTS (*see* Plain Old Telephone Service)
Power, 532
Power law, 737
Power-line network, 10, 22, 98–99
Power-save mode, 307
Power-line network, 10, 22
PPP (*see* Point-to-Point Protocol)
PPP over ATM, 250
PPPoA (*see* PPP over ATM)
Preamble, 200
Prediction, header,592
Predictive encoding, 548
Prefix, IP address, 443–444
Premaster key, 854
Presentation layer, 45
Pretty good privacy, 842–846
Primitive, service, 38–40
Principal, security, 773
Privacy, 860–861
Privacy amplification, 776
Privacy service, IEEE 802.11, 312
Private key ring, 845
Private network, virtual, 26, 821
Process server, 511
Procotol stack, IEEE 802.11, 299–301
Product cipher, 780
Product code, electronic, 327
Profile, Bluetooth, 321
Progressive video, 705
Promiscuous mode Ethernet, 290
Proposed standard, 81
Protection against wrapped sequence numbers, 516, 560
Protocol, 29
  1-bit sliding window, 229–232
  adaptive tree walk, 275–277
  address resolution, 467–469
  address resolution gratuitous, 469
  address resolution protocol proxy, 469
  authentication protocols, 827–841
  BB84, 773
  binary countdown, 272–273
  bit-map, 270–271
  Bluetooth, 322–323
  Bluetooth protocol stack, 322–323
  border gateway, 432, 479–484
  bundle, 603–605
  carrier sense multiple access, 266–269
  challenge-response, 828

Protocol (*continued*)
collision-free, 269–273
CSMA, 266–269
data link, 215–250
datagram congestion control, 503
delay-tolerant network, 603–605
Diffie-Hellman, 833–835
distance vector multicast routing, 383
dotted decimal notation Internet, 443
DVMR, 383
dynamic host configuration, 470
extensible authentication, 824
exterior gateway, 431, 474
file transfer, 455, 623
go-back-n, 232–239
hypertext transfer, 45, 649, 651, 683–693
IEEE 802.11, 299–312
IEEE 802.16, 312–320
initial connection, 511
interdomain, 431
interior gateway, 431, 474
IP, 438–488
intradomain, 431
limited-contention, 274–277
link control, 245
logical link control adaptation, 323
long fat network, 595–599
MAC, 261–280
mobile IP, 485–488
multiple access, 261–280
multiprotocol label switching, 360, 471–474
multiprotocol router, 429
negotiation, 35
network, 29
network control, 245
packet over SONET, 245–248
path vector, 481
point-to-point, 198, 245
POP3, 644
positive acknowledgement with retransmission, 225
real time streaming, 715
real-time, 606
real-time transport, 546–550
relation to services, 40
request-reply, 37
reservation, 271
resource reservation, 418
selective repeat, 239–244
session initiation, 731–735
Protocol (*continued*)
simple Internet plus, 457
simple mail transfer, 625, 638–641
sliding-window, 226–244, 229–232, 522
SLIP, 245
SOAP, 682
stop-and-wait, 221–226, 522
stream, 503, 527
stream control transmission, 503, 527
subnet Internet, 444–446
TCP (*see* Transmission Control Protocol)
temporary key integrity, 825
TKIP, 825
token passing, 271–272
transport, 507–530, 541–582
tree walk, 275–277
UDP, 541–552
utopia, 220–222
wireless application, 693
wireless LAN, 277–280
Protocol 1 (utopia), 220–222
Protocol 2 (stop-and-wait), 221–222
Protocol 3 (PAR), 222–226
Protocol 4 (sliding window), 229–232
Protocol 5 (go-back-n), 232–239
Protocol 6 (selective repeat), 239–244
Protocol hierarchy, 29–33
Protocol independent multicast, 385, 485
Protocol layering, 34
Protocol stack, 31–32
Bluetooth, 322–323
H.323, 728–731
IEEE 802.11, 299–301
IEEE 802.16, 314–315
OSI, 41–45
TCP/IP, 45–48
Proxy ARP, 469
Proxy caching, Web, 692
PSK (*see* Phase Shift Keying)
PSTN (*see* Public Switched Telephone Network)
Psychoacoustic audio encoding, 702–703
PTT (*see* Post, Telegraph & Telephone administration)
Public switched telephone network, 68, 138–164, 139
Public-key authentication using, 840–841
Public-key cryptography, 793–797
other algoirhtms, 796–797
RSA, 794–796
Public-key infrastructure, 810–813
directory, 812
Public-key ring, 845

Public-key signature, 799–800
Pulse code modulation, 153
Pure ALOHA, 262–265
Push-to-talk system, 166

Q

Q.931 standard, 729
QAM (*see* Quadrature Amplitude Modulation)
QoS (*see* Quality of Service)
QoS traffic scheduling (*see* Transmit power control)
QPSK (*see* Quadrature Phase Shift Keying)
Quadrature amplitude modulation, 132
Quadrature phase shift keying, 131
Quality of service, 35, 404–424, 411–414
  admission control, 415–418
  application requirements, 405–406
  differentiated services, 421–424
  integrated services, 418–421
  network layer, 404–424
  requirements, 405–406
  traffic shaping, 407–411
Quality of service routing, 415
Quantization, MPEG, 707
Quantization noise, 700
Quantum cryptography, 773–776
Qubit, 774
Queueing delay, 164
Queueing theory, 259
Quoted-printable encoding, 634

R

RA (*see* Regional Authority)
Radio access network, 66
Radio frequency identification, 10, 327–332
  active, 74
  backscatter, 329
  generation 2, 327–331
  HF, 74
  LF, 74
  passive 74
  UHF, 73–74
Radio network controller, 66
Radio transmission, 109–110
Random access channel, 174, 257
Random early detection, 403–404
Ranging, 184
  IEEE 802.16, 317
RAS (*see* Registration/Admission/Status)
Rate adaptation, 301
Rate anomaly, 309
Rate regulation, sending, 535–539
Real-time audio, 697
Real-time conferencing, 724–734
Real-time protocol, 606
Real-time streaming protocol, 715
Real-time transport protocols, 546–550
Real-time video, 697
Realm, Kerberos, 840
Reassociation, IEEE 802.11, 311
Receiving window, 228
Recovery
  clock, 127–129
  crash, 527–530
  fast, 578
Rectilinear basis, in quantum cryptography, 774
Recursive query, 621
RED (*see* Random Early Detection)
Redundancy, in quantum cryptography, 777–778
Reed-Solomon code, 208
Reference model, 41–54,
  Open Systems Interconnection. 41–45
  TCP/IP, 45–51
Reflection attack, 829
Region, in a network, 379
Regional Authority, 811
Registrar, 613
Registration/admission/status, 729
Relation of protocols to services, 40
Relation of services to protocols, 40
Reliable byte stream, 502
Remailer
  anonymous, 861–863
  cypherpunk, 862
Remote login, 61, 405–406
Remote procedure call, 543–546
  marshaling parameters, 544
  stubs, 544
Rendezvous point, 384
Repeater, 281, 340–342
Replay attack, 836
Request for comments, 81
Request header, 688
Request to send, 279
Request-reply protocol, 37

Request-reply service, 37
Reservation protocol, 271
Resilient packet ring, 271, 272
Resolver, 612
Resource record, 616
Resource record set, 851
Resource reservation protocol, 418
Resource sharing, 3
Response header, 688
Retransmission, fast, 577
Retransmission timeout, TCP, 568
Retransmission timer, 570–571
Retrospective on Ethernet, 298–299
Reverse lookup, 617
Reverse path forwarding algorithm, 381, 419
Revocation certificate, 812–813
RFC (*see* Request For Comments)
RFC 768, 541
RFC 793, 552
RFC 821, 625
RFC 822, 625, 630, 632, 633, 635, 636, 843, 862
RFC 1058, 373
RFC 1122, 553
RFC 1191, 556
RFC 1323, 516, 596
RFC 1521, 634
RFC 1663, 246
RFC 1700, 441
RFC 1939, 644
RFC 1958, 436
RFC 2018, 553
RFC 2109, 659
RFC 2326, 719
RFC 2364, 250
RFC 2410, 814
RFC 2440, 843
RFC 2459, 809
RFC 2535, 851, 853
RFC 2581, 553
RFC 2597, 423
RFC 2615, 247
RFC 2616, 683, 689
RFC 2854, 635
RFC 2883, 560, 580
RFC 2965, 689
RFC 2988, 553, 570
RFC 2993, 455
RFC 3003, 635
RFC 3022, 452
RFC 3023, 635
RFC 3119, 717
RFC 3168, 553, 558, 581
RFC 3174, 804
RFC 3194, 460
RFC 3246, 422
RFC 3261, 731
RFC 3344, 487
RFC 3376, 485
RFC 3390, 574
RFC 3501, 644
RFC 3517, 580
RFC 3550, 549
RFC 3748, 824
RFC 3775, 488
RFC 3782, 580
RFC 3875, 674
RFC 3963, 488
RFC 3986, 652
RFC 4120, 838
RFC 4306, 815
RFC 4409, 642
RFC 4614, 553
RFC 4632, 447
RFC 4838, 601
RFC 4960, 582
RFC 4987, 562
RFC 5050, 603
RFC 5246, 856
RFC 5280, 809
RFC 5321, 625, 633, 639
RFC 5322, 625, 630, 631, 632, 633, 760
RFC 5681, 581
RFC 5795, 595
RFID (*see* Radio Frequency IDentification)
RFID backscatter, 74
RFID network, 73–75
Rijmen, Vincent, 784
Rijndael, 784–787
Rijndael cipher, 312
Ring
  resilient packet, 271
  token, 271
Rivest, Ron, 773, 792, 795, 797, 804
Rivest Shamir Adleman algorithm, 794–796
RNC (*see* Radio Network Controller)
Robbed-bit signaling, 155
Roberts, Larry, 56
Robust header compression, 594
ROHC (*see* RObust Header Compression)
Root name server, 620

Routing algorithm, 27, 34, 362, 362–392
  ad hoc network, 389–392
  adaptive, 364
  anycast, 385–386
  AODV, 389
  Bellman-Ford, 370
  broadcast, 380–382
  class-based, 421
  classless interdomain, 447–449
  distance vector multicast protocol, 383
  dynamic, 364
  hierarchical, 378–380
  hot-potato, 484
  interdomain, 474
  internetwork, 431–432
  intradomain, 474
  link state, 373–378
  mobile host, 386–389
  multicast, 382–385
  multidestination, 380
  network layer, 362–392
  onion, 863
  OSPF, 464–479
  distance vector multicast, 383
  quality of service, 415
  session, 362
  shortest path, 366–368
  static, 364
  traffic-aware, 395–396
  triangle, 388
  wormhole, 336
Routing policy, 432
RPC (*see* Remote Procedure Call)
RPR (*see* Resilient Packet Ring)
RRSet (*see* Resource Record Set)
RSA (*see* Rivest Shamir Adleman algorithm)
RSVP (*see* Resource reSerVation Protocol)
RTCP (*see* Real-time Transport Control Protocol)
RTO (*see* Retransmission TimeOut, TCP)
RTP (*see* Real-time Transport Protocol)
RTS (*see* Request To Send)
RTSP (*see* Real Time Streaming Protocol)
Run-length encoding, 709

S

S-box, cryptographic, 779
S/MIME, 846
SA (*see* Security Association)
SACK (*see* Selective ACKnowledgement)
Sandbox, 858
Satellite
  communication, 116–125
  geostationary, 117
  low-earth orbit, 121–123
  medium-earth orbit, 121
Satellite footprint, 119
Satellite hub, 119
Sawtooth, TCP, 579
Scalable network, 34
Scatternet, Bluetooth, 320
Scheduling, packet, 411–414
Scheme, HTTP, 650
SCO (*see* Synchronous Connection Oriented link)
Scrambler, 128
SCTP (*see* Stream Control Transmission Protocol)
SDH (*see* Synchronous Digital Hierarchy)
Second-generation mobile phone network, 170–174
Sectored antenna, 178
Secure DNS, 850–853
Secure HTTP, 853
Secure naming, 848–853
Secure pairing bluetooth, 325
Secure simple pairing, Bluetooth, 325
Secure sockets layer, 853–857
Secure/MIME, 846
Security
  Bluetooth, 826–827
  communication, 813–827
  email, 841–846
  IEEE 802.11, 823–826
  IP, 814–818
  Java applet, 857–858
  mobile code, 857–860
  social issues, 860–869
  transport layer, 856
  Web, 856–860
  wireless, 822–827
Security association, 815
Security by obscurity, 768
Security principal, 773
Security threats, 847–848
Seeder, BitTorrent, 751
Segment, 499, 542
Segment header, TCP, 557–560
Selective acknowledgement, TCP, 560, 580
Selective repeat protocol, 239–244
Self-similarity, 737

Sending rate, regulation, 535–539
Sending window, 228
Sensor network, 13, 73–75
Serial line Internet protocol, 245
Server, 4
Server farm, 64, 738–741
Server side on the Web, 655–658
Server side Web page generation, 673–676
Server stub, 544
Service
  connection-oriented, 35–38, 359–361
  connectionless, 35–38, 358–359
Service level agreement, 407
Service primitive, 38–40
Service
  IEEE 802.11, 311–312
  integrated, 418–421
  provided by transport layer, 495–507
  provided to the transport layer, 356–357
  relation to protocols, 40
Service user, transport, 497
Serving GPRS support node, 68
Session, 44
Session initiation protocol, 731, 731–735
  compared to H.323, 733–734
Session key, 828
Session layer, 44–45
Session routing, 362
Set-top box, 4, 723
SGSN (*see* Serving GPRS Support Node)
SHA (*see* Secure Hash Algorithm)
Shannon, Claude, 94–95
Shannon limit, 100, 106, 146
Shared secret, authentication using, 828–833
Short interframe spacing, 308
Short message service, 12
Shortest path routing, 366–368
SIFS (*see* Short InterFrame Spacing)
Signal, balanced, 129–130
Signal-to-noise ratio, 94
Signaling
  common-channel, 155
  in-band, 155
  robbed-bit 155
Signature block, 629
Signatures, digital, 797–806
Signing, code, 858
Silly window syndrome, 567
SIM card, 69, 171
Simple Internet protocol, plus, 457
Simple mail transfer protocol, 625, 638–641
Simple object access protocol, 682
Simplex link, 97
Single-mode fiber, 101
Sink tree, 365
SIP (*see* Session Initiation Protocol
SIPP (*see* Simple Internet protocol Plus)
Skin, player, 715
SLA (*see* Service Level Agreement)
Sliding window, TCP, 565–568
Sliding window protocol, 1-bit, 229–232
  226–244, 229–232, 522
SLIP (*see* Serial Line Internet protocol)
Slot, 264
Slotted ALOHA, 264–266, 265
Slow start, TCP, 574
  threshold, 576
Smart phone, 12
Smiley, 623
SMS (*see* Short Message Service)
SMTP (*see* Simple Mail Transfer Protocol)
Snail mail, 623
SNR (*see* Signal-to-Noise Ratio)
SOAP (*see* Simple Object Access Protocol)
Social issues, 14–16
  security, 860–869
Social network, 8
Socket, 59
  Berkeley, 500–507
  TCP, 553
Socket programming, 503–507
Soft handoff, 178
Soft-decision decoding, 208
Soliton, 103
SONET (*see* Synchronous Optical NETwork)
Source port, 453
Spam, 623
Spanning tree, 382
Spanning tree bridge, 337–340
Spanning tree poem, 339
SPE (*see* Synchronous Payload Envelope)
Spectrum allocation, 182–183
Spectrum auction, 112
Speed of light, 106
Splitter, 149
Spoofing, DNS, 848–850
Spot beam, 119
Spread spectrum, 135
  direct sequence, 108
  frequency hopping, 107

Sprint, 111
Spyware, 662
SSL (*see* Secure Sockets Layer)
SST (*see* Structured Stream Transport)
Stack, protocol, 31–32
Standard
  de facto, 76
  de jure, 76
Stateful firewall, 819
Static channel allocation, 258–261
Static page, Web, 649
Static routing, 364
Static Web page, 649, 662–663
Station keeping, 118
Statistical multiplexing, 34
Statistical time division multiplexing, 135
STDM (*see* Statistical Time Division Multiplexing)
Steganography, 865–867
Stop-and-wait protocol, 221–226, 522
Store-and-forward packet switching, 36, 356
Stream cipher mode, 790–791
Stream control transmission protocol, 503, 527
Streaming audio and video, 697–734
Streaming live media, 721–724
Streaming media, 699
Streaming stored media, 713–720
Strowger gear, 161
Structured P2P network, 754
Structured stream transport, 503
STS-1 (*see* Synchronous Transport Signal-1)
Stub
  client, 544
  server, 544
Stub area, 477
Stub network, 481
Style sheet, 670–671
Sublayer, medium access control, 257–350
Subnet, 24, 444–446
Subnet Internet protocol, 444–446
Subnet mask, 443
Subnetting, 444
Subscriber identity module, 69, 171
Substitution cipher, 769–770
Superframe, extended, 154
Supergroup, 153
Supernet, 447
Swarm, BitTorrent, 751
Switch, 24
  compared to bridge and hub, 340–342
  Ethernet, 20, 289
Switched Ethernet, 20, 280, 288–290
Switching, 161–164
  circuit, 161–162
  cut-through, 36, 336
  data link layer, 332–349
  label, 360, 470–474
  message, 600
  packet, 162–164
  store-and-forward, 36
Switching element, 24
Symbol, 126
Symbol rate, 127
Symmetric-key cryptography, 778–793
  AES, 783–787
  cipher feedback mode, 789–790
  counter mode, 791–792
  DES, 780–782
  electronic code book mode, 787–788
  Rijndael, 784–787
  stream cipher mode, 790–791
  triple DES, 782–783
Symmetric-key signature, 798–799
SYN cookie, TCP, 561
SYN flood attack, 561
Synchronization, 44
Synchronous connection-oriented link, 325
Synchronous digital hierarchy, 156–159
Synchronous optical network, 156–159
Synchronous payload envelope, 157
Synchronous transport signal-1, 157
System, distributed, 2
Systematic code, 204

T

T1 carrier, 154–155
T1 line, 128, 154
Tag, HTML, 663–666
Tag switching, 471
Tail drop, 412
Talkspurt, 551
Tandem office, 141
TCG (*see* Trusted Computing Group)
TCM (*see* Trellis Coded Modulation)
TCP (*see* Transmission Control Protocol)
TDD (*see* Time Division Duplex)
TDM (*see* Time Division Multiplexing)
Telco, 61

Telephone
  cordless, 165
  mobile, 164–179
  smart, 12
Telephone system, 138–164
  end office, 140
  guard band, 133
  guard time, 135
  local loop, 144–152
  mobile, 164–179
  modem, 145
  modulation, 130–132
  point of presence, 143
  politics, 142–144
  structure, 139–142
  switching, 161–164
  tandem office, 141
  toll office, 141
  trunk, 152–160
Telephone trunk, 152–160
Television
  cable, 179–186
  community antenna, 179–180
Temporal masking, 703
Temporary key integrity protocol, 825
Terminal, VoIP, 728
Text messaging, 12
Texting, 12
Third Generation Partnership Project, 76
Third-generation mobile phone network, 65–69, 174–179
Third-party Web cookie, 662
Threats, security, 847–848
Three bears problem, 450
Three-way handshake, 516
Tier 1 ISP, 64
Tier 1 network, 437
Time division duplex, 316–317
Time division multiplexing, 135, 154–156
Time slot, 135
Timer management, TCP, 568–571
Timestamp, TCP, 560
Timing wheel, 593
Tit-for-tat strategy, BitTorrent, 752
TKIP (*see* Temporary Key Integrity Protocol)
TLS (*see* Transport Layer Security)
Token, 271
Token bucket algorithm, 397, 408–411
Token bus, 272
Token management, 44
Token passing protocol, 271–272
Token ring, 271
Toll connecting trunk, 141
Toll office, 141
Top-level domain, 613
Torrent, BitTorrent, 750
TPDU (*see* Transport Protocol Data Unit)
TPM (*see* Trusted Platform Module)
Traceroute, 466
Tracker, BitTorrent, 751
Traffic analysis, 815
Traffic engineering, 396
Traffic policing, 407
Traffic shaping, 407, 407–411
Traffic throttling, 398–401
Traffic-aware routing, 395–396
Trailer, 32, 194, 216, 250, 326
Transcoding, 694
Transfer agent, 624–625, 630–631
Transit service, 480
Transmission
  baseband, 125
  light, 14–116
  passband, 125
  wireless, 105–116
Transmission control protocol (TCP), 47, 552–582
  acknowledgement clock, 574
  application layer, 47–48
  comparison with OSI, 49–51
  congestion collapse, 572
  congestion control, 571–581
  congestion window, 571
  connection establishment, 560–562
  connection management, 562–565
  connection release, 562
  critique, 53–54
  cumulative acknowledgement, 558, 568
  delayed acknowledgement, 566
  duplicate acknowledgement, 577
  fast recovery, 578
  fast retransmission, 577
  future, 581–582
  introduction, 552–553
  Karn's algorithm, 571
  keepalive timer, 571
  link layer, 46
  maximum segment size, 559
  maximum transfer unit, 556
  Nagle's algorithm, 566
  performance, 582–599

Transmission control protocol (*continued*)
persistence timer, 571
port, 553
reference model, 45–51
retransmission timeout, 568
sawtooth, 579
segment header, 557–560
selective acknowledgement, 580
silly window syndrome, 567
sliding window, 565–568
slow start, 574
slow start threshold, 576
socket, 553
speeding up, 582–599
SYN cookie, 561
SYN flood attack, 561
timer management, 568–571
timestamp option, 560
transport layer, 47
urgent data, 555
well-known port, 553
window probe, 566
window scale, 560
Transmission line, 24
Transmission media, guided, 85–105
Transmission opportunity, 309
Transmit power control, IEEE 802.11, 312
Transponder, 116
Transport, structured stream, 503
Transport entity, 496
Transport layer, 44
addressing, 509–512
congestion control, 530–541
delay-tolerant networking, 599–605
error control, 522–527
flow control, 522–527
performance, 582–599
port, 509
security, 856
TCP, 552–582
TCP/IP, 47
Transport protocols, 507–530
transport service, 495–507
UDP, 541–552
Transport mode, IP security, 815
Transport protocol, 507–530, 541–582
design issues, 507–530
Transport protocol data unit, 499
Transport service access point, 509
Transport service primitive, 498–500
Transport service provider, 497
Transport service user, 497
Transposition cipher, 771–772
Tree walk protocol, adaptive, 275–277
Trellis-coded modulation, 146
Triangle routing, 388
Trigram, 770
Triple DES, 782–783
Trunk, telephone, 152–160
Trust anchor, 812
Trusted computing, 869
Trusted platform module, 869
TSAP (*see* Transport Service Access Point)
Tunnel mode, IPSec, 815
Tunneling, 387, 429–431
Twisted pair, 96–97
unshielded, 97
Twitter, 8
Two-army problem, 518–519
TXOP (*see* Transmission opportunity)

U

U-NII (*see* Unlicensed National Information Infrastructure)
Ubiquitous computing, 10
UDP (*see* User Datagram Protocol)
UHF RFID, 73–74
Ultra-wideband, 108
UMTS (*see* Universal Mobile Telecommunications System)
Unchoked peer, BitTorrent, 752
Unicast, 385
Unicasting, 17, 385
Uniform resource identifier, 652
Uniform resource locator, 650
Uniform resource name, 652,
Universal mobile telecommunications system, 65, 175
Universal serial bus, 128
Unlicensed national information infrastructure, 113
Unshielded twisted pair, 97
Unstructured P2P network, 754
Upstream proxy, 742
Urgent data, 555
URI (*see* Uniform Resource Identifier)
URL (*see* Scheme)
URN (*see* Uniform Resource Name)
USB (*see* Universal Serial Bus)

User, mobile, 10–13
User agent, 624, 626
User datagram protocol, 47, 541–552, 542, 549–550
  port, 542
  real-time transmission, 546–552
  remote procedure call, 541–543
  RTP, 547–549
Utopia protocol, 220–222
UTP (*see* Unshielded Twisted Pair)
UWB (*see* Ultra-WideBand)

V

V.32 modem, 146
V.34 modem, 146
V.90 modem, 147
V.92 modem, 147
Vacation agent, 629
Vampire tap, 291
Van Allen belt, 117
VC (*see* Virtual Circuit)
Very small aperture terminal, 119
Video
  interlaced, 705
  progressive, 705
  streaming, 704–712
Video compression, 706
Video field, 705
Video on demand, 713
Video server, 414, 416
Virtual circuit, 249, 358–361
Virtual circuit network, 358
  comparison with datagram network, 361–362
Virtual LAN, 21, 332, 342–349
Virtual private network, 4, 26, 431, 821–822
Virtual-circuit network, 358
Virus, 860
Visitor location register, 171
VLAN (*see* Virtual LAN)
VLR (*see* Visitor Location Register)
Vocal tract, 702
Vocoder, 701
VOD (*see* Video on Demand)
Voice over IP, 5, 36, 698, 725, 728–734
Voice signals, digitizing, 153–154
Voice-grade line, 93
VoIP (*see* Voice over IP)
VPN (*see* Virtual Private Network)
VSAT (*see* Very Small Aperture Terminal)

W

W3C (*see* World Wide Web Consortium)
Walled garden, 723
Walsh code, 136
WAN (*see* Wide Area Network)
WAP (*see* Wireless Application Protocol)
Watermarking, 867
Waveform coding, 702
Wavelength, 106
Wavelength division multiplexing, 159–160
WCDMA (*see* Wideband Code Division Multiple Access)
WDM (*see* Wavelength Division Multiplexing)
Wearable computer, 13
Web (*see* World Wide Web)
Web application, 4
Web browser, 648
  extension, 859–860
  helper application, 654
  plug-in, 653–654, 859
  proxy, 741–742
Webmail, 645, 645–646
Weighted fair queueing, 414
Well-known port, TCP, 553
WEP (*see* Wired Equivalent Privacy)
WFQ (*see* Weighted Fair Queueing)
White space, 113
Whitening, 781
Wide area network, 23, 23–27
Wideband code division multiple access, 65, 175
WiFi (*see* IEEE 802.11)
WiFi Alliance, 76
WiFi protected access, 73, 311
WiFi protected access 2 312, 823
Wiki, 8
Wikipedia, 8
WiMAX (*see* IEEE 802.16)
WiMAX Forum, 313
Window probe, TCP, 566
Window scale, TCP, 560
Wine, load-shedding policy, 401
Wired equivalent privacy, 73, 311, 823

Wireless
  broadband, 312–320
  fixed, 11
Wireless application protocol, 693
Wireless issues, 539–541
Wireless LAN, 39, 277–280, 299–312
Wireless LAN (*see* IEEE 802.11)
Wireless LAN protocol, 277–280
Wireless LAN protocols, 277–280
Wireless router, 19
Wireless security, 822–827
Wireless transmission, 105–116
Work factor, cryptographic, 768
World Wide Web, 2, 646–697
  AJAX, 679–683
  architectural overview, 647–649
  caching, 690–692
  cascading style sheets, 670–672
  client side, 649–652
  client-side page generation, 676–678
  connections, 684–686
  cookies, 658–662
  crawling, 696
  dynamic pages, 672
  forms, 667–680
  HTML, 663–667
  HTTP, 683–684
  message headers, 688–690
  methods, 686–688
  MIME types, 652–655
  mobile web, 693–695
  page, 647
  proxy, 692, 741–743
  search, 695–697
  security, 856–860
  tracking, 661
  server side, 655–658
  server-side page generation, 673–676
  static web pages, 662–672
World Wide Web Consortium, 82, 647
Wormhole routing, 336
WPA (*see* WiFi Protected Access)
WPA2 (*see* WiFi Protected Access 2)

X

X.400 standard, 629
X.509, 809–810
XHTML (*see* eXtended HyperText Markup Language)
XML (*see* eXtensible Markup Language)
XSLT (*see* eXtensible Stylesheet Language Transformation)

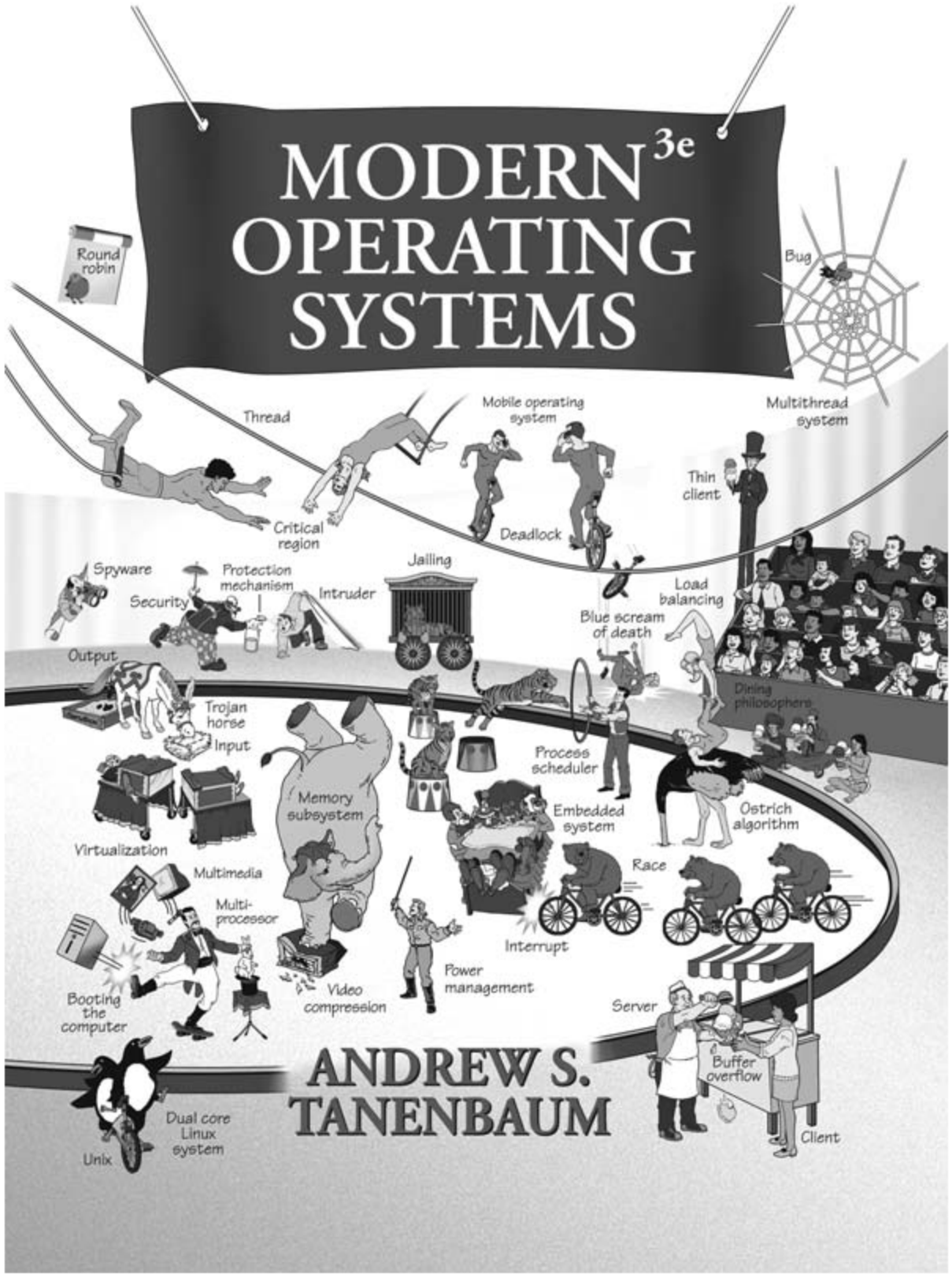
Z

Zipf's law, 737
Zone
  DNS, 619–620
  multimedia, 728

Also by Andrew S. Tanenbaum

Modern Operating Systems, 3rd ed.

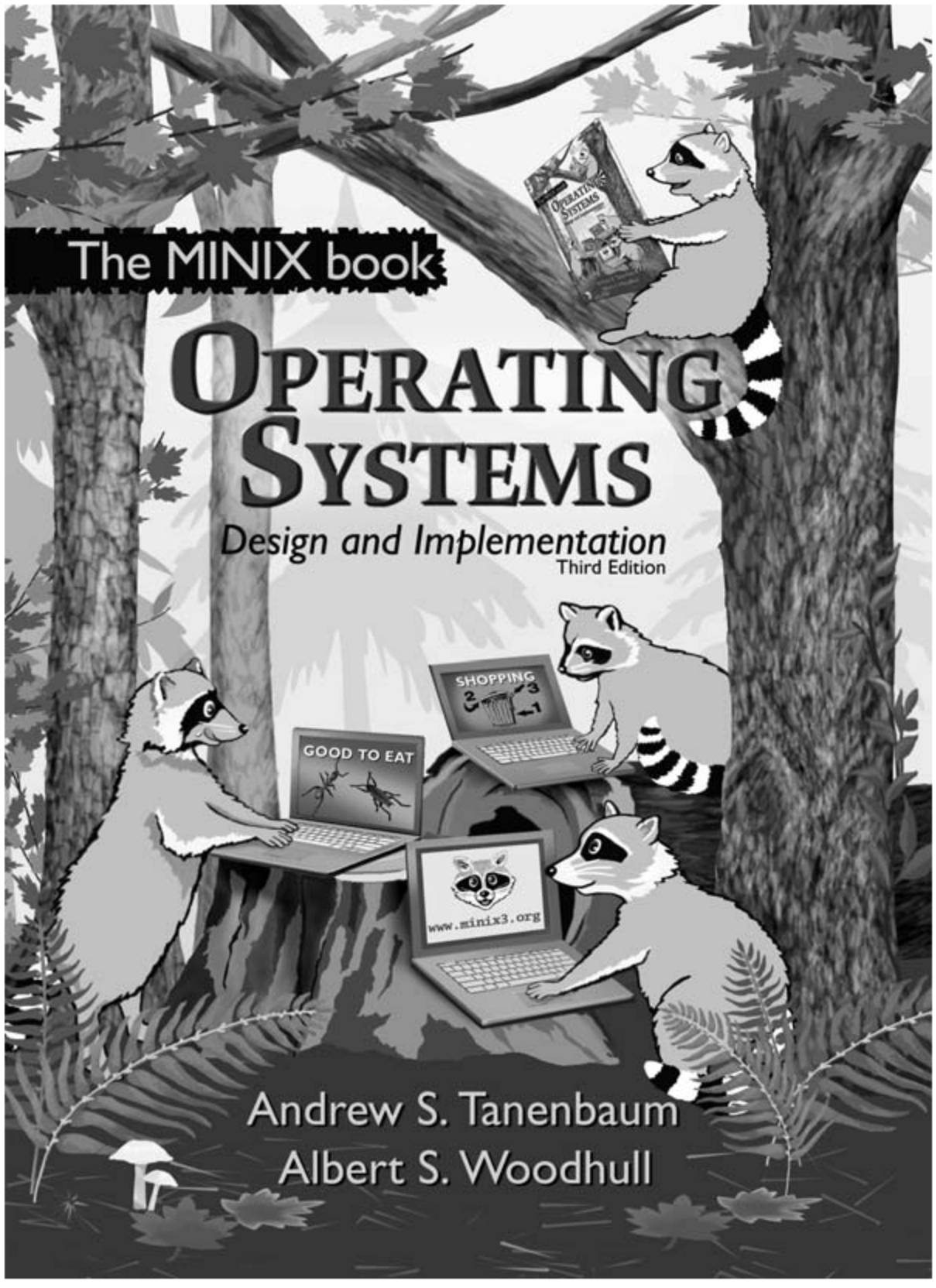
This worldwide best-seller incorporates the latest developments in operating systems. The book starts with chapters on the principles, including processes, memory management, file systems, I/O, and so on. Then it goes into three chapter-long case studies: Linux, Windows, and Symbian. Tanenbaum's experience as the designer of three operating systems (Amoeba, Globe, and MINIX) gives him a background few other authors can match, so the final chapter distills his long experience into advice for operating system designers.



Also by Andrew S. Tanenbaum and Albert S. Woodhull

Operating Systems: Design and Implementation, 3rd ed.

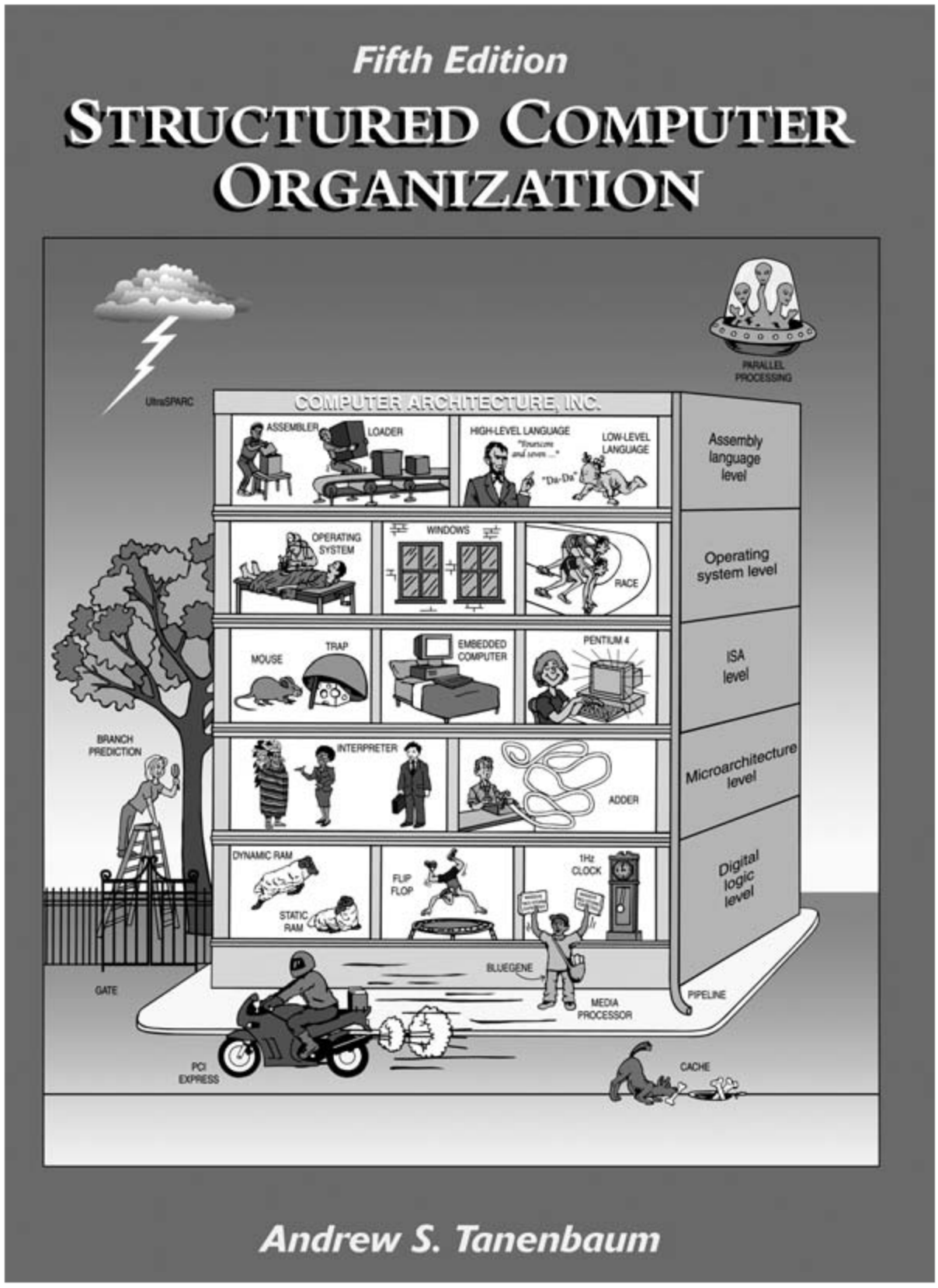
All other textbooks on operating systems are long on theory and short on practice. This one is different. In addition to the usual material on processes, memory management, file systems, I/O, and so on, it contains a CD-ROM with the source code (in C) of a small, but complete, POSIX-conformant operating system called MINIX 3 (see *www.minix3.org*). All the principles are illustrated by showing how they apply to MINIX 3. The reader can also compile, test, and experiment with MINIX 3, leading to in-depth knowledge of how an operating system really works.



Also by Andrew S. Tanenbaum

Structured Computer Organization, 5th ed.

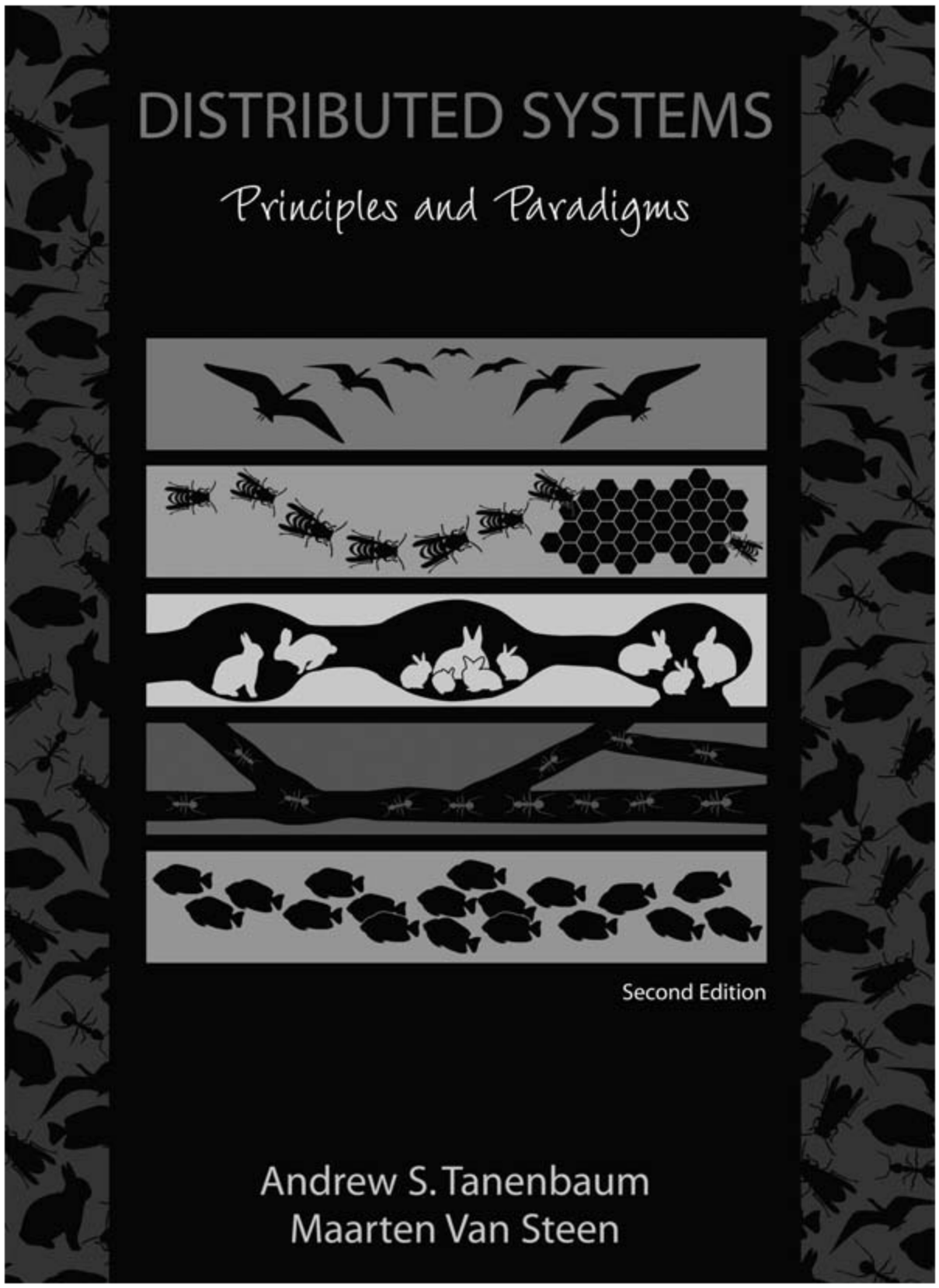
A computer can be structured as a hierarchy of levels, from the hardware up through the operating system. This book treats all of them, starting with how a transistor works and ending with operating system design. No previous experience with either hardware or software is needed to follow this book, however, as all the topics are self contained and explained in simple terms starting right at the beginning. The running examples used throughout the book range from the high-end UltraSPARC III, through the ever-popular x86 (Pentium) to the tiny Intel 8051 used in small embedded systems.



Also by Andrew S. Tanenbaum and Maarten van Steen

Distributed Systems: Principles and Paradigms, 2nd ed.

Distributed systems are becoming ever-more important in the world and this book explains their principles and illustrates them with numerous examples. Among the topics covered are architectures, processes, communication, naming, synchronization, consistency, fault tolerance, and security. Examples are taken from distributed object-based, file, Web-based, and coordination-based systems.



ABOUT THE AUTHORS

Andrew S. Tanenbaum has an S.B. degree from M.I.T. and a Ph.D. from the University of California at Berkeley. He is currently a Professor of Computer Science at the Vrije Universiteit where he has taught operating systems, networks, and related topics for over 30 years. His current research is on highly reliable operating systems although he has worked on compilers, distributed systems, security, and other topics over the years. These research projects have led to over 150 refereed papers in journals and conferences.

Prof. Tanenbaum has also (co)authored five books which have now appeared in 19 editions. The books have been translated into 21 languages, ranging from Basque to Thai and are used at universities all over the world. In all, there are 159 versions (language/edition combinations), which are listed at *www.cs.vu.nl/~ast/publications*.

Prof. Tanenbaum has also produced a considerable volume of software, including the Amsterdam Compiler Kit (a retargetable portable compiler), Amoeba (an early distributed system used on LANs), and Globe (a wide-area distributed system).

He is also the author of MINIX, a small UNIX clone initially intended for use in student programming labs. It was the direct inspiration for Linux and the platform on which Linux was initially developed. The current version of MINIX, called MINIX 3, is now focused on being an extremely reliable and secure operating system. Prof. Tanenbaum will consider his work done when no computer is equipped with a reset button and no living person has ever experienced a system crash. MINIX 3 is an on-going open-source project to which you are invited to contribute. Go to *www.minix3.org* to download a free copy and find out what is happening.

Tanenbaum is a Fellow of the ACM, a Fellow of the the IEEE, and a member of the Royal Netherlands Academy of Arts and Sciences. He has also won numerous scientific prizes, including:

- 2010 TAA McGuffey Award for Computer Science and Engineering books
- 2007 IEEE James H. Mulligan, Jr. Education Medal
- 2002 TAA Texty Award for Computer Science and Engineering books
- 1997 ACM/SIGCSE Award for Outstanding Contributions to Computer Science Education
- 1994 ACM Karl V. Karlstrom Outstanding Educator Award

His home page on the World Wide Web can be found at *http://www.cs.vu.nl/~ast/*.

David J. Wetherall is an Associate Professor of Computer Science and Engineering at the University of Washington in Seattle, and advisor to Intel Labs in Seattle. He hails from Australia, where he received his B.E. in electrical enginering from the University of Western Australia and his Ph.D. in computer science from M.I.T.

Prof. Wetherall has worked in the area of networking for the past two decades. His research is focused on network systems, especially wireless networks and mobile computing, the design of Internet protocols, and network measurement.

He received the ACM SIGCOMM Test-of-Time award for research that pioneered active networks, an architecture for rapidly introducing new network services. He received the IEEE William Bennett Prize for breakthroughs in Internet mapping. His research was recognized with an NSF CAREER award in 2002, and he became a Sloan Fellow in 2004.

As well as teaching networking, Prof. Wetherall participates in the networking research community. He has co-chaired the program committees of SIGCOMM, NSDI and MobiSys, and cofounded the ACM HotNets workshops. He has served on numerous program committees for networking conferences, and is an editor for ACM Computer Communication Review.

His home page on the World Wide Web can be found at *http://djw.cs.washington.edu*.