

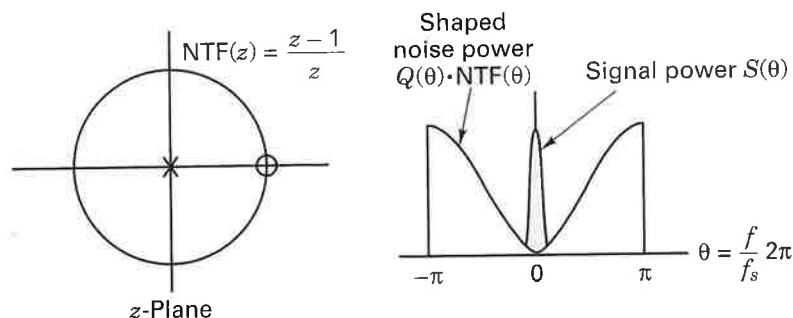


Figure 13.26 Noise transfer function in z-plane and power spectrum of signal and shaped noise of Σ - Δ modulator.

13.3.4.1 Noise Performance of Σ - Δ Modulator

In the previous section, we alluded to the ability of a Σ - Δ modulator to exchange excess sample rate for improvement in SNR of quantized data. We first consider how this trade occurs with oversampled and filtered data plus AWGN, and we will then examine the same process with shaped noise. If the quantization noise is white, and the signal is sampled at a rate that exceeds the Nyquist rate, the white noise is uniformly distributed over a spectral interval equal to the sample rate. This interval is denoted the *first Nyquist zone* or the *primary strip*. Since the quantizing noise power is fixed at $q^2/12$ (see Equation 13.12), the power spectral density of the quantizing noise for a signal sampled at the rate f_s must be $q^2/(12f_s)$ watts/Hz. Operating the quantizer at elevated sample rates reduces the power spectral density of the quantizing noise in the signal bandwidth. The oversampled data can be digitally filtered to reject the out-of-band quantizing noise and then down sampled to the signal's Nyquist rate. If the signal is oversampled by a factor of two, the filtering will reject half the noise power. Rejecting half the noise power reduces the rms value of the quantized noise amplitude by $\sqrt{2}$, or the power by 3 dB. To reduce the noise power by 6 dB, and thus improve quantization noise by 1-bit (see Equation 13.24), one must oversample by a factor of 4 and reject all but one-fourth of the quantizing noise with a filter. We conclude that we improve the SNR of an oversampled white noise quantizer, at the rate of 3-dB (or half a bit) per doubling of the sample rate relative to Nyquist rate.

We now address the rate at which we can improve SNR with an over sampled shaped noise quantizer. The NTF of the Σ - Δ shaping filter has a zero at dc which results in a double zero in the filter's power spectral response. If we expand the spectral response of the filter in a Taylor series, and truncate after the first nonzero term we obtain the following simple approximation to the filter response valid in the neighborhood of the signal spectrum:

$$H^2(\omega) = \left[2 \sin \left(\frac{\omega}{2\omega_s} \right) \right]^2$$

$$= 2 \left[1 - \cos \left(\frac{\omega}{\omega_s} \right) \right] \quad (13.64)$$

$$= 2 \left\{ 1 - \left[1 - \frac{1}{2!} \left(\frac{\omega}{\omega_s} \right)^2 + \cdots \right] \right\} \approx \left(\frac{\omega}{\omega_s} \right)^2 = \left(\frac{f}{f_s} \right)^2$$

Here, f_s is the sample rate of the modulator.

The shaped noise power that survives the filtering action of the low-pass filter following the Σ - Δ modulator is of the form

$$N(\omega) = \frac{N_0}{2} \int_{-f_{BW}}^{f_{BW}} \left(\frac{f}{f_s} \right)^2 df = \frac{N_0}{2} \frac{1}{3} \left(\frac{f}{f_s} \right)^3 \bigg|_{f=-f_{BW}}^{f=f_{BW}} = \frac{N_0}{3} \left(\frac{f_{BW}}{f_s} \right)^3 \quad (13.65)$$

The ratio of the noise term for a signal sampled at f_s , and one sampled at $2f_s$ and then filtered down to the same output signal bandwidth f_{BW} , is found to be 8 or 9 dB. Thus a Σ - Δ modulator with a single zero in its NTF improves its SNR by 9 dB or by 1.5 bits per doubling of the sample rate. Sigma-delta modulators built formed with multiple digital integrators and feedback loops have an increased number of transmission zeros in the NTF. By a similar derivation, we would find that the NTF of a Σ - Δ modulator with 2 and 3 zeros improves SNR by 15 and 21 dB (or 2.5 and 3.5 bits), respectively. Thus, a two-zero Σ - Δ operating at 64 times Nyquist or with sample rate doubled 6 times, exhibits a SNR improvement of 90 dB. The spectrum illustrated in Figure 13.27 was formed with a two-zero Σ - Δ modulator, and if we include the 6-dB loss due to the spectral decomposition of a real signal, 2-dB reduction due to amplitude back-off from full scale, and 3-dB loss due to the windowing of the DFT, the noise level is at the expected level of 79 dB below the spectral peak.

13.3.5 Sigma-Delta A-to-D Converter (ADC)

A Σ - Δ analog-to-digital (A-to-D) converter or ADC is usually implemented as an integrated circuit built around the Σ - Δ modulator. The circuit must contain auxiliary subsystems to form a complete system. These include an analog anti-alias filter, a sample-and-hold circuit, a switched-capacitor integrator for the modulator, a feedback digital-to-analog (D-to-A) converter or DAC, and a digital resampling filter. Due to the high oversampling rate of the process, the analog anti-alias filter can be simple RC or dual RC stage, with a large transition bandwidth extending over many octaves. The DAC converter is required to deliver an analog feedback signal from the output of the A-to-D quantizer. Since the DAC is in the feedback loop, it does not benefit from the loop gain, and consequently its linearity and precision must match the design performance levels of the entire system. The Σ - Δ modulator preserves the signal fidelity in a restricted segment of the sampled spectrum. To access this high-fidelity segment of the spectrum, the output of the modulator must be filtered and down sampled. The post-processing filter following the modulation process rejects the out-of-band noise located in the excess bandwidth that exists due to the oversampling. This is usually a linear-phase, finite-impulse response (FIR), resampling digital filter.

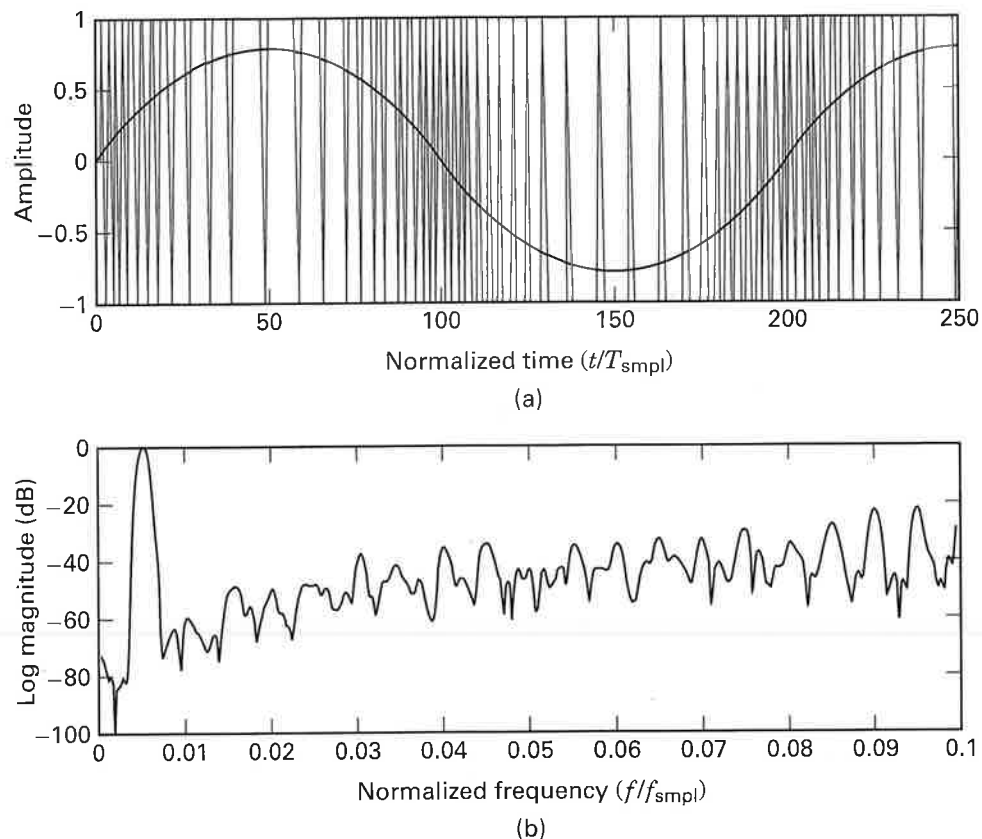
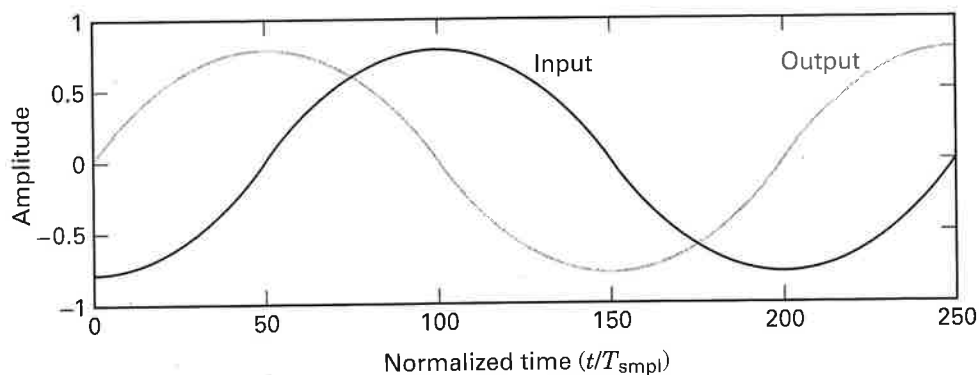


Figure 13.27 One bit Σ - Δ modulator: (a) input and output time series (b) spectral response.

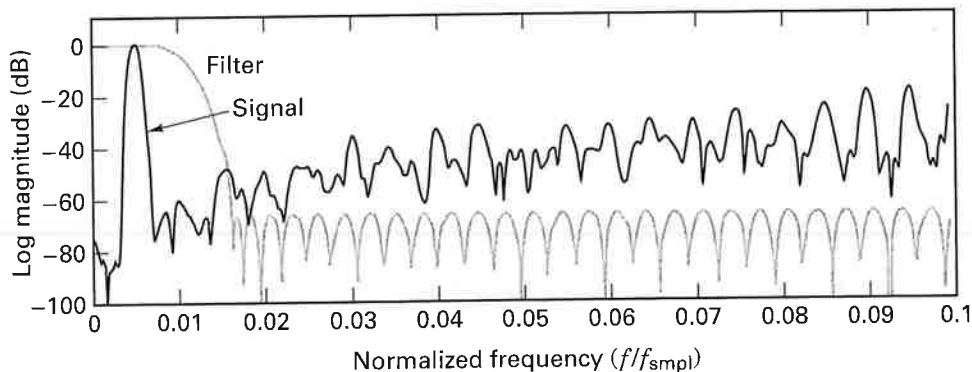
Figure 13.27a presents an oversampled input sinusoidal signal and the corresponding output signal of a one-bit, dual-zero, Σ - Δ modulator. Figure 13.27b shows the spectral response of the output series. Note that the shaped noise spectrum in the neighborhood of the signal is approximately 80 dB below the peak spectrum of the input sinusoid. Note also that the output signal is restricted to ± 1 , and that the loop is essentially duty cycle modulating the square wave in proportion to the amplitude of the input signal. Figure 13.28 presents the time series and the spectrum obtained from the output of the down-sampling filter following the modulator.

13.3.6 Sigma-Delta D-to-A Converter (DAC)

The Σ - Δ modulator, originally designed to be embedded in an ADC, has become an essential building block of the reverse task—the digital-to-analog converter (DAC). Nearly all high-end audio equipment and most communication system



(a)



(b)

Figure 13.28 Post processing filter following Σ - Δ modulator: (a) input and output time series, (b) spectral response.

DACs are implemented with Σ - Δ converters. The process uses the Σ - Δ modulator as a digital-to-digital (D-to-D) transformation, which converts a high-precision (say, 16-bit) representation of oversampled digital data to a low-precision (say, 1-bit) representation of the same data. The oversampled one-bit data stream is then delivered to a 1-bit DAC with two analog output levels defined with the same precision as a 16-bit converter. Here, the advantage of a one-bit DAC operating at high speed but with only two levels is that speed is less expensive than precision. The 2-level, high-speed DAC replaces a low-speed DAC that would have to resolve 65,536 distinct levels. Very simple analog low-pass filtering following the 1-bit DAC suppresses the out-of-band noise spectrum and delivers the reduced bandwidth, high-fidelity version of the original digital data. The requantizing of the oversampled data is a straightforward signal processing task, using an all-digital Σ - Δ modulator. The only additional task to be performed when using a Σ - Δ DAC is the requirement to raise the sample rate of the data to 64 times its Nyquist rate.

This task is performed by a DSP-based interpolating filter, which is a standard signal-processing block found in most systems that use a DAC to transition between a digital signal source and an analog output [12].

As a standard illustration of the process, a CD player uses an interpolating filter to realize a 1-to-4 up-sampling, resulting in a separation of the periodic spectra associated with sampled data. This increased separation permits the smoothing filter following the DAC to have a wider transition bandwidth, and hence, a reduced component count and reduced implementation cost. The CD specification uses terms such as “four-to-one oversampled” to reflect the presence of the interpolating filters. Starting with the CD 1-to-4 interpolator, it is a simple task to again raise the sample rate by another factor of 1-to-16, with a second inexpensive interpolating filter. The data, now 64 times oversampled, is presented to the all-digital Σ - Δ modulator and one-bit DAC to complete the analog conversion process. This structure is shown in Figure 13.29.

There are many signals that are significantly oversampled with respect to the signal’s bandwidth. These signals can easily be converted to analog representations by the use of a Σ - Δ modulator and a 1-bit DAC. Examples are control signals used in circuits such as AGC, carrier VCO, and timing VCO. Many systems employ the Σ - Δ modulator and 1-bit DAC to generate and deliver analog control signals throughout the system.

13.4 ADAPTIVE PREDICTION

The prediction gain to be had in classical predictive coders is proportional to the ratio of the signal variance to prediction-error variance. This is because for a fixed quantizing noise level, fewer bits are required to describe a signal with smaller energy. The utility of the predictive coder is limited by possible mismatches between the source signal and the predictor filter. The sources of mismatch are related to the time-varying behavior (i.e., nonstationarity) of the amplitude distribution and of the spectral or correlation properties of the signal. Adaptive encoders incorporate (slow-time) auxiliary loops to estimate the parameters required to obtain locally optimal performance. These auxiliary loops periodically schedule modifications to the prediction loop parameters and thus avoid predictor mismatch. The International Telegraph and Telephone Consultative Committee (CCITT) has selected a 32-kbits/s adaptive differential pulse-code modulation (ADPCM) coder

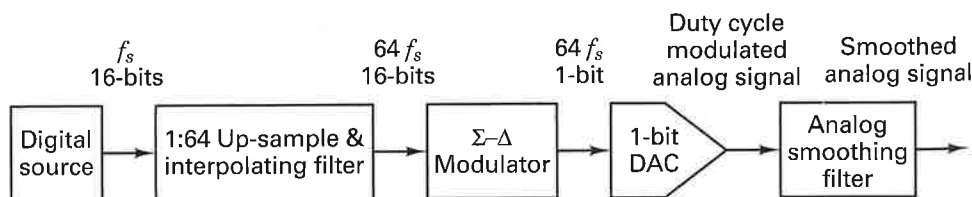


Figure 13.29 Signal flow graph of Σ - Δ digital-to-analog converter.

a standard
transition be-

polating fil-
odic spectra
oothing fil-
e, a reduced
cation uses
interpolat-
again raise
interpolat-
-digital $\Sigma\Delta$
. This struc-

spect to the
resentations
signals used
employ the
ontrol signals

ional to the
e for a fixed
smaller en-
hes between
e related to
tribution and
ers incorpo-
d to obtain
le modifica-
smatch. The
CCITT) has
PCM) coder

Smoothed
analog signal
analog
oothing
filter

as a standard for toll-quality speech. This achieves a 2:1 savings in bit rate relative to 64-kbits/s logarithmic compressed PCM.

13.4.1 Forward Adaptation

In forward adaptation algorithms, the input data to be encoded are buffered and processed in order to estimate the local statistics, such as the first N samples of the autocorrelation function. The zero-delay correlation sample $R_x(0)$ is a short-term estimate of the variance. This estimate is used to adjust the automatic gain control (AGC) in order to obtain an optimal match of the scaled input signal to that of the quantizer dynamic range. This is denoted AQF, for *adaptive quantization forward control*. The remaining $N - 1$ correlation estimates are used to form new filter coefficients for the prediction filter. This adaptation is called adaptive prediction forward (APF) control. Figure 13.30 shows this form of the adaptive algorithm. This is an extension of the structure presented in Figure 13.20. Here the predictor coefficients are derived from the input data, now called *side information*, and must be transmitted along with the prediction errors from the encoder to the decoder. The update rate of these adaptive coefficients is related to the length of time the input signal can be considered locally stationary. For example, speech caused by mechanical displacement of the speech articulators (tongue, lips, teeth, etc.) cannot change characteristics more rapidly than 10 or 20 times per second. This suggests an update interval of 50 to 100 ms. Using arithmetically simple but sub-optimal estimating algorithms to compute the local filter parameters makes a higher update rate necessary. An update every 20 ms to compute the parameters of a 10- to 12-tap filter has become a common rate. Prediction gains of 10 to 16 dB can be had with 10-tap filters, when feed-forward adaptation is used with predictive coders [13].

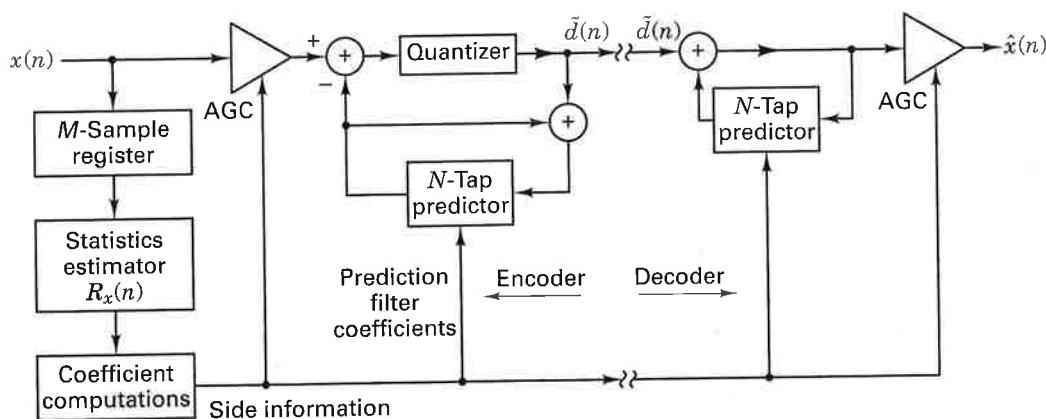


Figure 13.30 Forward adaptive prediction and quantization coding.

13.4.2 Synthesis/Analysis Coding

The encoding schemes we have examined until now can be classed as *waveform encoders*. They construct approximations to the input signals, which minimizes some distance measure between the signal and the approximation. These techniques are very general and can be applied to any signal source. Synthesis/analysis coders, on the other hand, are very signal specific; in particular, they are designed primarily for speech signals. These encoders take advantage of the fact that while the hearing mechanism responds to the amplitude content of a signal's short-term spectrum, it is fairly insensitive to its phase structure. Thus, this class of encoder forms a reconstructed signal that approximates the magnitudes and time-varying characteristic of a sequence of the signal's short-term spectra, but it makes no attempt to preserve its relative phase.

The spectral characteristic of speech appears to be stationary over periods between 20 and 50 ms. A number of techniques have evolved which analyze the spectral characteristics of voice every 20 ms and that use the results of that analysis to synthesize a waveform exhibiting the same short-term power spectrum. Some techniques employ a model of the speech-generation mechanism for which model parameters have to be estimated at the update rate. This type of encoder is best represented, in its various forms, as the linear predictive coder (LPC). Variations of LPC encoding manipulate the signal by combinations of spectral modifications and time partitions, which, with side information, reduce the number of time samples required to faithfully reconstruct the original spectrum. The common thread that runs through all synthesis/analysis encoders used for speech signals is that the voice signal is not required to "look" like the original signal, but rather, to "sound" like it.

13.4.2.1 Linear Predictive Coding

The adaptive predictors, described in Section 13.3.2, were designed to predict or form good estimates of an input signal. In the adaptive form, the prediction coefficients are recomputed as side information from periodic examination of the input data. Then, the difference between the input and the prediction is transmitted to the receiver to resolve the prediction error. *Linear predictive coders* (LPCs) are the natural extension of N -tap predictive coders. When the filter coefficients are periodically computed with an optimal algorithm, the prediction is so good that there is (essentially) no prediction error information worth transmitting to the receiver. Rather than transmit these low-level prediction errors, the LPC system transmits the filter coefficients and the voiced/unvoiced excitation decision for the model. Thus, the only data sent in LPC is the high-quality side information of the classic adaptive algorithm. An LPC model for voice synthesis is shown in Figure 13.31. The LPC encoders represent the core of hybrid coders that embed the coder and excitation generator in an analysis-by-synthesis loop that searches through excitation options to minimize the difference between the input signal and synthesized signal. The Regular-Pulse Excited (RPE), and the Codebook-Excited Linear Predictive (CELP) encoder are used in cellular phones to obtain toll-quality speech at

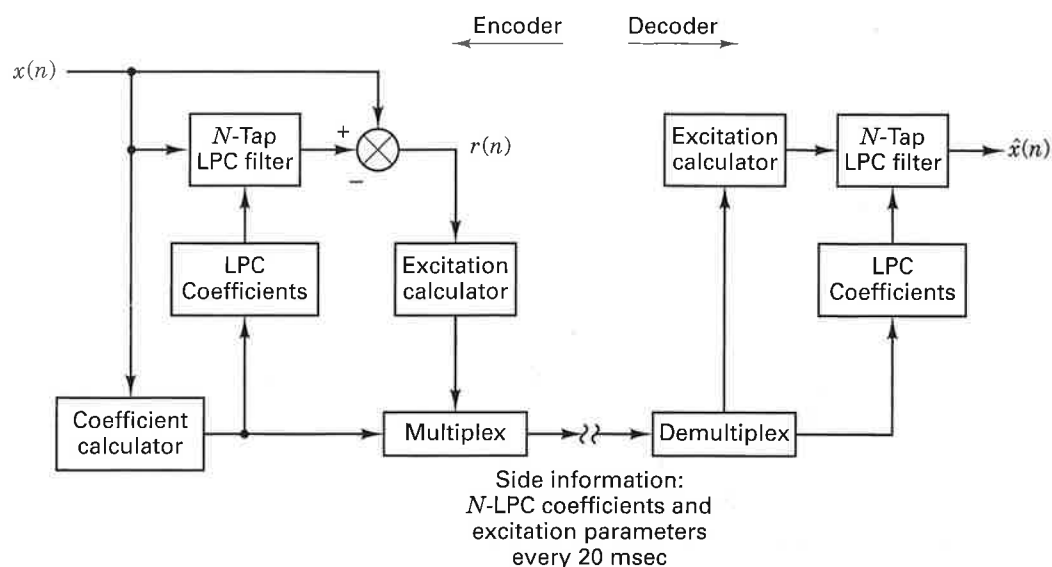


Figure 13.31 Block diagram: Linear predictive coefficient (LPC) speech modeling.

data rates below 9.6 kbits/s. The Global System for Mobile (GSM) Communications uses RPE compression, while mobile phone systems, designed to meet code division multiple access (CDMA) standard IS-95, use a variant of CELP. Additional material on CELP is presented in section 13.8.1.3.

An application of this model employing a 12-tap LPC speech synthesizer is found in children's speaking games. Further treatment of LPC techniques used for speech are covered in Section 13.8.1.

13.5 BLOCK CODING

The quantizers we have examined up to now have been *scalar* in nature, since they form a *single output sample* based on the present input sample and (possibly) the N previous output samples. Block coders, on the other hand, form a *vector of output samples* based on the present and the N previous input samples. The *coding gain* of a waveform coder is the ratio of the input SNR to the output SNR. When the noise variances of the input and output are equal, this gain is simply the ratio of input-to-output signal variances. The ratio converts directly to 6 dB per bit for the difference between the number of input bits per sample and the average number of output bits per sample. Block coders can achieve impressive coding gains. On average, they can represent sequences quantized to 8 bits with only 1 or 2 bits per sample [8]. Block-coding techniques are varied, but a common thread that runs through block-coding techniques is the mapping of an input sequence to an alternative coordinate system. This mapping may be to a subspace of a larger space, so

that the mapping may not be reversible [8]. Alternatively, a data-dependent editing scheme may be used to identify the subspace of the mapping from which the quantized data are extracted. Block-coding techniques are often classified by their mapping techniques, which include, for example, vector quantizers, various orthogonal transform coders, and channelized coders, such as subband coders. Block coders are further described by their algorithmic structures, such as codebook, tree, trellis, discrete Fourier transform, discrete cosine transform, discrete Walsh-Hadamard transform, discrete Karhunen-Loeve transform, and quadrature mirror filter-bank coders. We now examine examples of the various block-coding schemes.

13.5.1 Vector Quantizing

Vector quantizers represent an extension of conventional scalar quantization. In scalar quantization, a scalar value is selected from a finite list of possible values to represent an input sample. The value is selected to be close (in some sense) to the sample it is representing. The fidelity measures are various weighted mean-square measures that preserve our intuitive concept of distance in terms of ordinary vector lengths. By extension, in vector quantization, a vector is selected from a finite list of possible vectors to represent an input vector of samples. The selected vector is chosen to be close (in some sense) to the vector it is representing.

Each input vector can be visualized as a point in an N -dimensional space. The quantizer is defined by a partition of this space into a set of nonoverlapping volumes [14]. These volumes are called intervals, polygons, and polytopes, respectively, for one-, two-, and N -dimensional vector spaces. The task of the vector quantizer is to determine the volume in which an input vector is located; The output of the optimal quantizer is the vector identifying the centroid of that volume. As in the one-dimensional quantizer, the mean-square error is a function of the boundary locations for the partition and the multidimensional pdf of the input vector.

The description of a vector quantizer can be cast as two distinct tasks. The first is the code-design task, which deals with the problem of performing the multidimensional volume quantization (or partition) and selecting the allowable output sequences. The second task is that of using the code, and deals with searching for the particular volume with this partition that corresponds (according to some fidelity criterion) to the best description of the source. The form of the algorithm selected to control the complexity of encoding and decoding may couple the two tasks—the partition and the search. The standard vector coding methods are codebook-, tree-, and trellis-coding algorithms [15, 16].

13.5.1.1 Codebook, Tree, and Trellis Coders

The codebook coders are essentially table look-up algorithms; A list of candidate patterns (codewords) is stored in the codebook memory. Each pattern is identified by an address or pointer index. The coding routine searches through the list of patterns for the one that is closest to the input pattern and transmits to the receiver the address where that pattern can be found in its codebook. The tree and

trellis coders are sequential coders. As such, the allowable codewords of the code cannot be selected independently but must exhibit a node-steering structure. This is similar to the structure of the sequential error-detection-and-correction algorithms, which traverse the branches of a graph while forming the branch weight approximation to the input sequence. (See Section 6.5.1.) A tree graph suffers from exponential memory growth as the dimension or depth of the tree increases. The trellis graph reduces the dimensionality problem by the simultaneous tracking of contender paths with an associated path-weight metric called *intensity*, with the use of a finite state trellis. (See Section 6.3.3.)

13.5.1.2 Code Population

The code vectors stored in the codebook, tree, or trellis are the likely or typical vectors. The first task, that of code design, in which the likely code vectors are identified, is called *populating* the code. The methods of determining the code population are classically *deterministic*, *stochastic*, and *iterative*. The deterministic population is a list of preassigned possible outputs based on a simple suboptimal or user-perception fidelity criterion or based on a simple decoding algorithm. An example of the former is the coding of the samples in 3-space of the red, green, and blue (RGB) components of a color TV signal. The eye does not have the same resolution to each color and it would appear that the coding could be applied independently to each color to reflect this different sensitivity. The resulting quantizing volumes would be rectangular parallelepipeds. The problem with independent quantizing is that we do not see images in this coordinate system; rather, we see images in the coordinates of luminance, hue, and saturation. A black-and-white photo, for example, uses only the luminance coordinate. Thus, quantizing RGB coordinates independently does not result in the smallest amount of user-perceived distortion for a given number of bits. To obtain improved distortion performance, the RGB quantizer should partition its space into regions that reflect the partitions in the alternate space. Alternatively, the quantization could be performed independently in the alternative space by the use of transform coding, treated in Section 13.6. Deterministic coding is the easiest to implement but leads to the smallest coding gain (smallest reduction in bit rate for a given SNR).

The stochastic population would be chosen based on an assumed underlying pdf of the input samples. Iterative solutions to the optimal partitions exist and can be determined for any assumed pdf. The overall samples are modeled by the assumed pdf. In the absence of an underlying pdf, iterative techniques based on a large population of training sequences can be used to form the partition and the output population. Training sequences may involve tens of thousands of representative input samples.

13.5.1.3 Searching

Given an input vector and a populated codebook, tree, or trellis, the coder algorithm must conduct a search to determine the best matching contender vector. An exhaustive search over all possible contenders will assure the best match. Coder performance improves for larger dimensional spaces, but so does complexity. An

exhaustive search over a large dimension may be prohibitively time consuming. An alternative is to conduct a nonexhaustive, suboptimal search scheme, with acceptably small degradations from the optimal path. Memory requirements and computational complexity often are a driving consideration in the selection of search algorithms. Examples of search algorithms include single-path (best leaving branch) algorithms, multiple-path algorithms, and binary (successive approximation) codebook algorithms. Most of the search algorithms attempt to identify and discard unlikely patterns without having to test the entire pattern.

13.6 TRANSFORM CODING

In Section 13.5.1 we examined vector quantizers in terms of a set of likely patterns and techniques to determine the one pattern in the set closest to the input pattern. One measure of goodness of approximation is the weighted mean-square error of the form

$$d(\mathbf{X}, \hat{\mathbf{X}}) = (\mathbf{X} - \hat{\mathbf{X}})\mathbf{B}(\mathbf{X})(\mathbf{X} - \hat{\mathbf{X}})^T \quad (13.76)$$

where $\mathbf{B}(\mathbf{X})$ is a weight matrix and $\mathbf{X}^T$ is the transpose of the vector $\mathbf{X}$. The minimization may be computationally simpler if the weighting matrix is a diagonal matrix. A diagonal weighting matrix implies a decoupled (or uncorrelated) coordinate set so that the error minimization due to quantization can be performed independently over each coordinate.

Thus, transform coding entails the following set of operations, which are shown in Figure 13.32:

1. An invertible transform is applied to the input vector.
2. The coefficients of the transform are quantized.
3. The quantized coefficients are transmitted and received.
4. The transform is inverted with quantized coefficients.

Note that the transform does not perform any source encoding, it merely allows for a more convenient description of the signal vector to permit ease of source encoding. The task of the transform is to map a correlated input sequence into a different coordinate system in which the coordinates have reduced correlation. Recall that this is precisely the task performed by predictive coders. The source encoding occurs with the bit assignment to the various coefficients of the transform. As part of this assignment, the coefficients may be partitioned into subsets that are quantized with a different number of bits but *not* with different quantizing step sizes. This assignment reflects the dynamic range (variance) of each coefficient and may be weighted by a measure that reflects the importance, relative to the human perception [17], of the basis element carried by each coefficient. A subset of the coefficients, for instance, may be set to zero amplitude, or may be quantized with 1 or 2 bits.

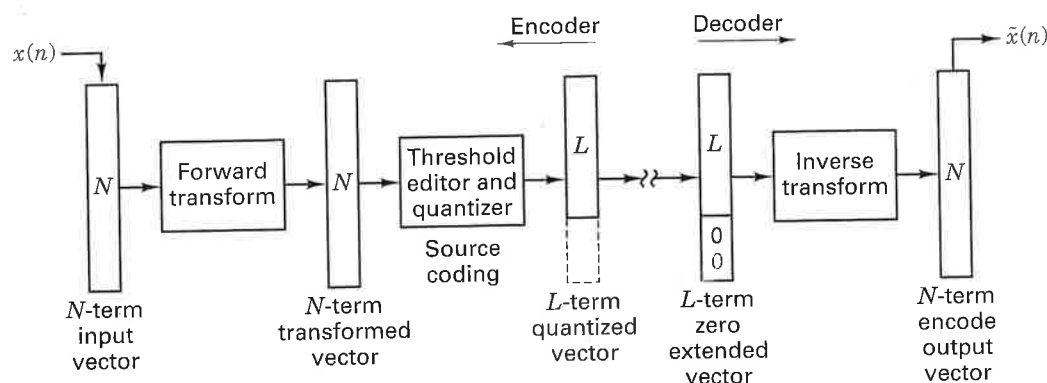


Figure 13.32 Block diagram: Transform coding.

The transformation can be chosen to be independent of the data vector. Examples of such transforms are the discrete Fourier transform (DFT), discrete Walsh-Hadamard transform (DWHT), discrete cosine transform (DCT), and the discrete slant transform (DST). The transformation can also be derived from the data vector, as is done in the discrete Karhunen-Loeve transform (DKLT), sometimes called the *principal component transform* (PCT) [18]. The data-independent transforms are easiest to implement but do not perform as well as the data-dependent transforms. Often, the attraction of computational simplicity is sufficient justification for using the data-independent transformations. The coding gain penalty for using a good suboptimal transformation is small (typically less than 2 dB), and the degradation is usually cited when demonstrating performance characteristics.

13.6.1 Quantization for Transform Coding

Transform coders are called spectral encoders because the signal is described in terms of a spectral decomposition (in a selected basis set). The spectral terms are computed for nonoverlapped successive blocks of input data. Thus, the output of a transform coder can be viewed as a set of time series, one series for each spectral term. The variance of each series can be determined and each can be quantized with a different number of bits. By permitting independent quantization of each transform coefficient, we have the option of allocating a fixed number of bits among the transform coefficients to obtain a minimum quantizing error.

13.6.2 Subband Coding

The transform coders of Section 13.6 were described as a partition of an input signal into a collection of slowly varying time series, each of which is associated with a particular basis vector of the transform. The spectral terms, the inner product of the data with the basis vectors, are computed by a set of inner products. The set of inner prod-

ucts can be computed by a set of *finite impulse response* (FIR) filters [19]. With this perspective, the transform coder can be considered to be performing a channelization of the input data. By extension, a *subband coder*, which performs a spectral channelization by a bank of contiguous narrowband filters, can be considered a special case of a transform coder. (A typical subband coder is shown in Figure 13.33.)

Casting the spectral decomposition of the data as a filtering task affords us the option of forming a class of *custom* basis sets (i.e., spectral filters)—in particular, basis sets that reflect out user-perception preferences and our source models. For example, the quantizing noise generated in a band with large variance will be confined to that band, not spilling into a nearby band with low variance and hence susceptible to low-level signals being masked by noise. We also have the option of forming filters with equal or unequal bandwidths. (See Figure 13.33.) Thus, we can independently assign to each subband the sample rate appropriate to its bandwidth and a number of quantizing bits appropriate to its variance. By comparison, in conventional transform coding, each basis vector amplitude is sampled at the same rate.

The subband coder can be designed as a transmultiplexer. Here the input signal is considered to be composed of a number of basis functions modeled as independent narrow-bandwidth subchannels. The encoder separates or channelizes the input signal into a set of low-data-rate, time-division multiplexed (TDM) channels.

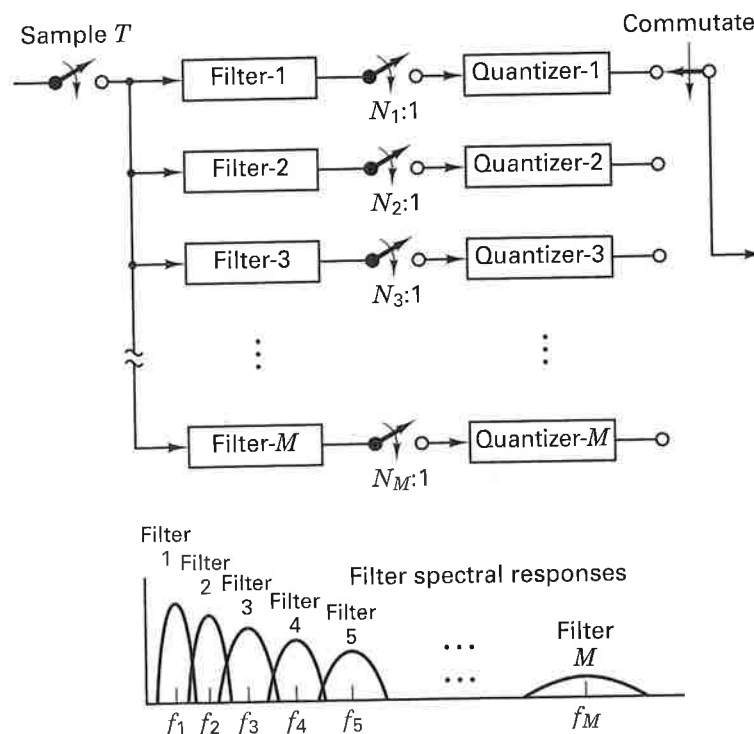


Figure 13.33 Subband coding performed by channelized spectral decomposition.

With this channelization process, a special 33.) words us the particular, els. For ex- e confined usceptible ning filters pendently number of transform

input sig- d as inde- nelizes the) channels.

After quantization and transmission, the decoder reverses the filtering and resampling process, converting the TDM channels back to the original signal. In the classic approach to this process, one can implement a bank of narrow-band filters with the steps of heterodyning, low-pass filtering, and downsampling (often called *decimation*). This filtering operation reduces the input bandwidth to the selected channel bandwidth and resamples the signal to the lowest rate that avoids aliasing of the reduced bandwidth channelized data. At the receiver, the reverse process is performed. The channelized signals are passed through interpolating filters to increase their sample rate to the desired output sample rate are heterodyned back to their proper spectral position, and they are combined to form the original composite signal. For speech encoding—or, more generally, for signals that are related to mechanical resonance—filter banks with nonequal center frequencies and nonequal bandwidths are desirable. Such filters are called constant- Q (or proportional) filter banks. These filters have logarithmically spaced center frequencies with bandwidths proportional to the center frequencies. This proportional spacing appears as uniform spacing and bandwidth when viewed on a log scale, and it reflects the spectral properties of many physical acoustic sources.

13.7 SOURCE CODING FOR DIGITAL DATA

Coding to reduce the redundancy of a data source entails the selection of an (usually) efficient binary representation of that source. Often this requires the substitution of one binary representation of the source symbols with an alternative representation. The substitution is usually temporary and is performed to achieve an economy of storage or transmission of the discrete source symbols. The binary code assigned to each source symbol must satisfy certain constraints to permit reversal of the substitution. In addition, the code may be further constrained by system considerations, such as memory limits or implementation ease.

We are so used to assigning binary codes to represent source symbols that we may lose sight of the arbitrariness of this assignment. The most common example of this is the binary assignments to the cardinal numbers (let's not even consider the negative numbers). We can count in straight binary, binary-coded octal, binary-coded decimal, binary-coded hexadecimal, two-out-of-five decimal, excess-three decimal, and so forth. In this example, ease of computation, error detection, ease of display, or convenience of coding are the considerations for selecting the assignment. For the specific task of data compression, the primary goal is to *reduce the number of bits*.

Finite discrete sources are characterized by a set of distinct symbols. $X(n)$, where $n = 1, 2, \dots, N$ is called the *source alphabet*, and where n is a data index. A complete characterization requires the probability of each symbol and the joint probabilities of the symbols taken two at a time, three at a time, and so on. The symbols may represent a two-level (binary) source, such as the black-and-white levels of a facsimile scan, or a many-symbol source, such as the 40 common characters of Sanskrit. Another common many-symbol alphabet is the keyboard of a computer terminal. These nonbinary symbols are mapped, via a dictionary called a character code, to a binary alphabet description. (See Figure 2.3 for the ASCII

tral

code and Figure 2.4 for the EBCDIC code.) The standard character codes are of fixed length, such as 5, 6, or 7 bits. The length is usually chosen so that there are enough binary characters to assign a unique binary sequence to each input alphabet character. These may include the upper and lowercase letters of the alphabet, numerals, punctuation, special characters, and control characters, such as backspace, return, and so forth. Fixed-length codes have the property that character boundaries are separated by a fixed bit count. This allows the conversion of a serial data stream to a parallel data stream by a simple bit counter.

Two-code standards may define the same symbol in different ways. For example, the ASCII (7-bit) code has enough bits to assign different binary sequences to the upper and lowercase versions of each letter. On the other hand, the *Baudot* (5-bit) code, with only 32 binary sequences, cannot do the same. To account for the full character set, the Baudot code defines two control characters, called *letter shift* (*LS*) and *figure shift* (*FS*), to be used as prefixes. When used, these control characters reassign the binary-to-symbol mapping. This works very much like the *shift* key on a typewriter—the shift key reassigns a completely new character set to the keyboard. In a similar fashion, the keyboards on some calculators have two prefix character keys, so that each keystroke can have three possible meanings. Also, some word processor instruction codes use double- and triple-stroke command functions. In a very real sense, these two- and three-word instructions represent a variable-length code assignment. These longer code words are assigned to characters (or instructions) that do not occur as often as those assigned single codewords. What we receive in exchange for using the occasional longer words is more efficient storage (smaller keyboard) or transmission of the source.

Data compression codes are often variable-length codes. Intuitively, we would expect the length of a binary sequence assigned to each alphabet symbol to be inversely related to the probability of that symbol. After all, if a symbol occurs with high probability, it contains little information and should not be assigned much of the system resources. In a similar manner, it would not seem unreasonable to find that when all symbols are equally likely, the code should be of fixed length. Perhaps the best known variable-length code is the Morse code. Samuel Morse counted the quantity of letters in a printer's font drawer to determine the relative frequency of letters in normal text. The variable-length code assignment reflects this relative frequency.

A significant amount of *data compression* can be realized when there is a wide difference in the probabilities of the symbols. To achieve this compression, there must also be a sufficiently large number of symbols. Sometimes, in order to have a large enough set of symbols, we form a new set of symbols derived from the original set, called an *extension code*. We have already seen this trick in Example 13.3, and we will examine the general technique in the next section.

13.7.1 Properties of Codes

Earlier we alluded to properties that a code must satisfy for it to be useful. Some of these properties are obvious, and some are not. It is worth listing and demonstrating the *desired properties*. Consider the following three-symbol alphabet with the probability assignments shown:

codes are of
 hat there are
 input alpha-
 the alphabet,
 such as back-
 hat character
 ion of a serial

ys. For exam-
 sequences to
 1, the *Baudot*
 account for the
 lled *letter shift*
 ontrol charac-
 e the *shift key*
 set to the key-
 ve two prefix
 eanings. Also,
 ke command
 ns represent a
 ned to charac-
 gle codewords.
 s in more effi-

vely, we would
 mbol to be in-
 ol occurs with
 ed much of the
 ole to find that
 h. Perhaps the
 nted the quan-
 yency of letters
 ive frequency.
 hen there is a
 s compression,
 es, in order to
 rived from the
 ick in Example

useful. Some of
 nd demonstrat-
 habet with the

X_i	$P(X_i)$
a	0.73
b	0.25
c	0.02

Accompanying the input alphabet are the following six binary code assignments, where the rightmost bit is the earliest bit:

Symbol	Code 1	Code 2	Code 3	Code 4	Code 5	Code 6
a	00	00	0	1	1	1
b	00	01	1	10	00	01
c	11	10	11	100	01	11

Scan these for a moment and try to determine which codes are practical.

Uniquely Decodable Property. Uniquely decodable codes are those that allow us to invert the mapping to the original symbol alphabet. Obviously, code 1 in the preceding example is not uniquely decodable because the symbols a and b are assigned the same binary sequence. Thus, the first requirement of a useful code is that each symbol be assigned a unique binary sequence. By this condition, all the other codes appear satisfactory until we examine codes 3 and 6 carefully. These codes indeed have unique binary sequences assigned to each symbol. The problem occurs when these code sequences are strung together. For instance, try to decode the binary pattern 1 0 1 1 in code 3. Is it b , a , b , b or b , a , b , c or b , a , c , b ? Trying to decode the same sequence in code 6 gives similar difficulties. These codes are not uniquely decodable, even though the individual characters have unique code assignments.

Prefix-Free Property. A sufficient (but not necessary) condition to assure that a code is uniquely decodeable is that no codeword be the prefix of any other code word. Codes that satisfy this condition are called prefix-free codes. Note that code 4 is not prefix-free, but it is uniquely decodable. Prefix-free codes also have the property that they are instantaneously decodable. Code 4 has a property that may be undesirable; it is not instantaneously decodable. An instantaneously decodable code is one for which the boundary of the present codeword can be identified by the end of the present codeword, rather than by the beginning of the next codeword. For instance, in transmitting the symbol b with the binary sequence 1 0 in code 4, the receiver cannot determine if this is the whole codeword for symbol b or the partial codeword for symbol c . By contrast, codes 2 and 5 are prefix free.

13.7.1.1 Code Length and Source Entropy

At the beginning of the chapter, we described the formal concept of information content and source entropy. We identified the self-information $I(X_n)$, in bits, about the symbol X_n denoted as $I(X_n) = \log[1/P(X_n)]$. From the perspective that information resolves uncertainty we recognize that the information content of a sym-

bol goes to zero as the probability of that symbol goes to unity. We also defined the *entropy* of a finite discrete source as the average information of that source. From the perspective that information resolves uncertainty, the entropy is the average amount of uncertainty resolved per use of the alphabet. It also represents the average number of bits per symbol required to describe the source. In this sense, it is also the lower bound of what can be achieved with some variable length data compression codes. A number of considerations may prevent an actual code from achieving the entropy bound of the input alphabet. These include uncertainty in probability assignment and buffering constraints. The average bit length achieved by a given code is denoted by $\bar{n}$. This average length is computed as the sum of the binary code lengths n_i weighted by the probability of that code symbol $P(X_i)$.

$$\bar{n} = \sum_i n_i P(X_i)$$

A great deal is implied about the performance of a variable-length code when we say *average number of bits*. In a variable-length code assignment, some symbols will have code lengths which exceed the average length, while some will have code lengths which are smaller than the average. It may occur that a long pattern of symbols with long codewords is delivered to the coder. The short-term bit rate required to transmit these symbols will exceed the average bit rate of the code. If a channel is expecting data at the average rate, the local excess rate must be buffered in a memory. By the same token, a long pattern of symbols with short code words may be delivered to the coder. The short-term bit rate required to transmit these symbols will fall short of the average rate of the code. Here the channel will find itself waiting for bits that are not to be had. For this reason, data *buffering* is required to smooth the local statistical variations associated with the input alphabet.

The last caveat is that variable-length codes are designed to operate with a specified list of symbols and probabilities. If the data presented to the coder have a significantly different list of probabilities, the coder buffers may not be able to support the mismatch and buffer underflow or overflow will occur.

13.7.2 Huffman Code

The Huffman code [20] is a prefix-free, variable-length code that can achieve the shortest average code length $\bar{n}$ for a given input alphabet. The shortest average code length for a particular alphabet may be significantly greater than the entropy of the source alphabet. This inability to exploit the promised data compression is related to the alphabet, not to the coding technique. Often the alphabet can be modified to form an extension code, and the same coding technique is then reapplied to achieve better compression performance. Compression performance is measured by the *compression ratio*. This measure is equal to the ratio of the average number of bits per sample before compression to the average number of bits per sample after compression.

The Huffman coding procedure can be applied for transforming between any two alphabets. We will demonstrate the application of the procedure between an arbitrary input alphabet and a binary output alphabet. The Huffman code is gener-

defined the source. From the average its the aver- sense, it is h data com- code from certainty in th achieved e sum of the $P(X_i)$.

de when we symbols will l have code tern of sym- ate required If a channel offered in a e words may t these sym- ill find itself s required to t. erate with a coder have a able to sup-

achieve the test average the entropy mpresion is abet can be is then reap- rformance is of the aver- mber of bits

between any e between an ode is gener-

ated as part of a tree-forming process. The process starts by listing the input alphabet symbols, along with their probabilities (or relative frequencies), in descending order of occurrence. These tabular entries correspond to the branch ends of a tree, as shown in Figure 13.34. Each branch is assigned a branch weight equal to the probability of that branch. The process now forms the tree that supports these branches. The two entries with the lowest relative frequency are merged (at a branch node) to form a new branch with their composite probability. After every merging, the new branch and the remaining branches are reordered (if necessary) to assure that the reduced table preserves the descending probability of occurrence. We call this reordering *bubbling* [21]. During the rearrangement after each merging, the new branch rises through the table until it can rise no further. Thus, if we form a branch with a weight of 0.2 and during the bubbling process find two other branches already with the 0.2 weight, the new branch is bubbled to the top of the 0.2 group, as opposed to simply joining it. The bubbling to the top of the group results in a code with reduced code length variance but otherwise a code with the same average length as that obtained by simply joining the group. This reduced code length variance lowers the chance of buffer overflow.

As an example of this part of the code process, we will apply the Huffman procedure to the input alphabet shown in Figure 13.34. The tabulated alphabet and the associated probabilities are shown on the figure. After forming the tree each branch node is labeled with a binary 1/0 decision to distinguish the two branches. The labeling is arbitrary, but for consistency, at each node we will label the branch going up with a "1" and the branch going down with a "0". After labeling the branch nodes, we trace the tree path from the base of the tree (far right) to each output branch (far left). The path contains the binary sequence to reach that branch. In the following table, we have listed at each end branch the path sequence corresponding to each path where $i = 1, \dots, 6$:

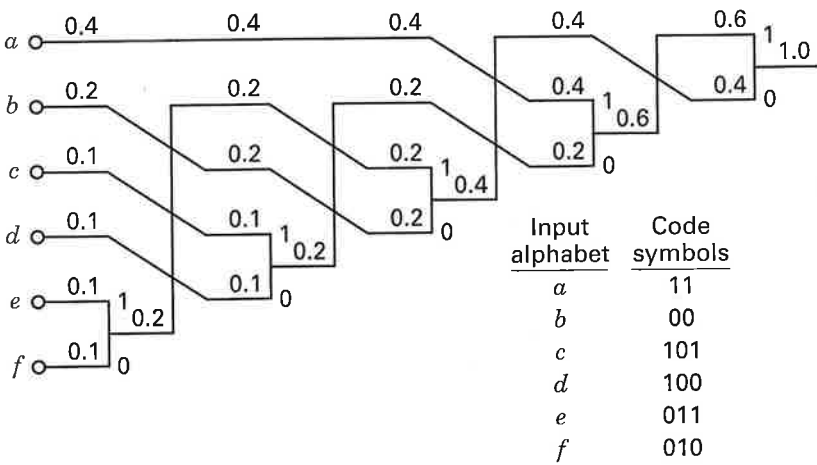


Figure 13.34 Huffman coding tree for a six-character set.

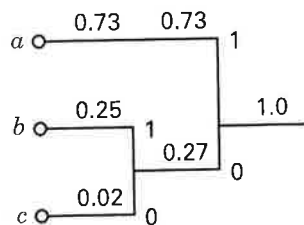
X_i	$P(X_i)$	Code	n_i	$niP(X_i)$
a	0.4	11	2	0.8
b	0.2	00	2	0.4
c	0.1	101	3	0.3
d	0.1	100	3	0.3
e	0.1	011	3	0.3
f	0.1	010	3	0.3
				$\bar{n} = 2.4$

We find that the average code length $\bar{n}$ for this alphabet is 2.4 bits per character. It does not mean that we have to find a way to transmit a noninteger number of bits. Rather, it means that, on average, 240 bits will have to be moved through the communication channel when transmitting 100 input symbols. For comparison, a fixed-length code required to span the six-character input alphabet would be of length 3 bits, and the entropy of the input alphabet, using Equation (13.2), is 2.32 bits. Thus, this code offers a compression ratio of 1.25 (3.0/2.4) and achieves 96.7% (2.32/2.40) of the possible compression ratio. As another example, one for which we can demonstrate the use of code extension, let us examine the three-character alphabet presented in Section 13.6.1:

X_j	$P(X_j)$
a	0.73
b	0.25
c	0.02

The Huffman code tree for this alphabet is shown in Figure 13.35, and the details are tabulated as

X_i	$P(X_i)$	Code	n_i	$n_iP(X_i)$
a	0.73	1	1	0.73
b	0.27	01	2	0.54
c	0.02	00	2	0.04
				$\bar{n} = 1.31$



Input alphabet	Code symbols
a	1
b	01
c	00

Figure 13.35 Huffman coding tree for three-character set.

where $i = 1, \dots, 3$. The average code length for this Huffman code is 1.31 bits; it would be 2 bits for a fixed-length code. The compression ratio for this code is 1.53. Again, using Equation (13.2), the entropy for the alphabet is 0.9443 bit, so that the efficiency ($0.944/1.31 = 72\%$) of the code is significantly smaller than for the preceding example.

To improve coding efficiency or to achieve greater compression gain, we have to redefine the source alphabet. A larger source alphabet holds the promise of increased variability, which is one requirement to realize a reduction in average code length, and an increased number of tree branches for assigning the variable-length code. We do this by selecting characters two at a time from the source alphabet to be new characters in the extension alphabet. If we assume that the symbols are independent, the probability of each new element is the product of the individual probabilities. The extension alphabet is

X_i	$P(X_i)$	Code	n_i	$n_i P(X_i)$
aa	0.5329	1	1	0.5329
ab	0.1825	00	2	0.3650
ba	0.1825	011	3	0.5475
bb	0.0625	0101	4	0.2500
ac	0.0146	01000	5	0.0730
ca	0.0146	010011	6	0.0876
bc	0.0050	0100100	7	0.0350
cb	0.0050	01001011	8	0.0400
cc	0.0002	01001010	8	0.0016

$$\bar{n} = 1.9326 \text{ bits/two symbols} \\ = 0.9663 \text{ bit/symbol}$$

where $i = 1, \dots, 9$, and the code sequence for each X_i has been found by the use of the Huffman procedure above. The compression ratio of this extension code is 2.07 and the coding efficiency is 97.7%.

Extension codes offer a very powerful technique to include the effects of non-independent symbol sets. For example, in English text, adjacent letters are highly correlated. Very common pairs include

th	re	in
sh	he	e_
de	ed	s_
ng	at	r_
te	es	d_

where the underscore represents a space. Similarly, common English three-tuples include

the	and	for
ing	ion	ess

Thus, rather than perform Huffman coding on the individual letters, it is more efficient to extend the alphabet to include all 1-tuples plus common 2-tuples and 3-tuples, and then to perform the coding on the extension code.

13.7.3 Run-Length Codes

In many applications, lengthy runs of specific symbols characterize a sequence of symbols to be transmitted or stored. Rather than code each symbol of a lengthy run, it makes sense to describe the run with an efficient substitution code. As an example, runs of spaces (the most common symbol in text) are encoded in many communication protocols by a control character followed by the character count. The IBM 3780 BISYNC protocol has an option to replace runs of spaces with an "IGS" character (if EBCDIC; or "GS" if ASCII), followed by a count of 2 to 63. Longer runs are partitioned into successive runs of 63 characters.

The run-length substitution coding can be applied to the original symbol alphabet or the binary representation of that alphabet. Run-length coding is particularly attractive for binary alphabets derived from specific sources. The most important commercial example is facsimile coding, used for transmitting documents by instant electronic mail [22].

13.7.3.1 Huffman Coding for Facsimile Transmission

Facsimile transmission is the process of transmitting a two-dimensional image as a sequence of successive line scans. The most common images are, in fact, documents containing text and figures. The position of the scan lines and the position along a scan line are quantized into spatial locations that define a two dimensional grid of picture elements, called *pixels*. The standard CCITT document is defined to be of width 8.27 in. (20.7 cm) and to have length 11.7 in. (29.2 cm), almost 8.5 in. by 11.0 in. The spatial quantization for normal resolution is 1728 pixels/line and 1188 lines/document. The standard also defines a high-resolution quantization with the same 1728 pixels/line but with 2376 lines/document. The total number of individual pixels for a normal-resolution facsimile transmission is 2,052,864, and it is doubled for high resolution. For comparison, the number of pixels in NTSC (National Television Standards Committee) standard commercial television is 480×640 or 307,200. Thus, facsimile has 6.7 or 13.4 times the resolution of a standard TV image.

The relative brightness or darkness of the scanned image at each position in the scan is quantized into two levels: B (for black) and W (for white). Thus, the signal observed during a scan line is a two-level pattern representing the B and W image intensity under the scan. It is easy to see that a horizontal scan line across this sheet of paper will exhibit a pattern consisting of long runs of B and W levels. The standard CCITT run-length coding scheme to compress the run of B and W levels is based on a modified variable-length Huffman code; it is listed in Table 13.1. Two types of patterns are identified, runs of W and runs of B. Each run length is described by a *partitioned codeword*. The first partition, called the *makeup codeword* or most significant bits (MSB), identifies runs with lengths that are multiples of 64. The second partition, called the *terminating codeword* of least significant bits (LSB), identifies the length of the remaining run. Each run of B (or W) of length from 0 through 63 is assigned a unique Huffman code word, as is each run of length $64 \times K$, where $K = 1, 2, \dots, 27$. A unique END OF LINE (EOL) is also defined in

TABLE

Run
Length

64
128
192
256
320
384
448
512
576
640
704
768
832
896

Run
Length

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

TABLE 13.1 Modified Huffman Code for CCITT Facsimile Standard

Run Length	White	Black	Run Length	White	Black
Makeup Code Words					
64	11011	0000001111	960	011010100	0000001110011
128	10010	000011001000	1024	011010101	0000001110100
192	010111	000011001001	1088	011010110	0000001110101
256	0110111	000001011011	1152	011010111	0000001110110
320	00110110	000000110011	1216	011011000	0000001110111
384	00110111	000000110100	1280	011011001	0000001010010
448	01100100	000000110101	1344	011011010	0000001010011
512	01100101	0000001101100	1408	011011011	0000001010100
576	01101000	0000001101101	1472	010011000	0000001010101
640	01100111	0000001001010	1536	010011001	0000001011010
704	011001100	0000001001011	1600	010011010	0000001011011
768	011001101	0000001001100	1664	011000	0000001100100
832	011010010	0000001001101	1728	010011011	0000001100101
896	011010011	0000001110010	EOL	000000000001	000000000001
Run Length	White	Black	Run Length	White	Black
Terminating Codewords					
0	00110101	000110111	32	00011011	000001101010
1	000111	010	33	00010010	000001101011
2	0111	11	34	00010011	000011010010
3	1000	10	35	00010100	000011010011
4	1011	011	36	00010101	000011010100
5	1100	0011	37	00010110	000011010101
6	1110	0010	38	00010111	000011010110
7	1111	00011	39	00101000	000011010111
8	10011	000101	40	00101001	000001101100
9	10100	000100	41	00101010	000001101101
10	00111	0000100	42	00101011	000011011010
11	01000	0000101	43	00101100	000011011011
12	001000	0000111	44	00101101	000001010100
13	000011	00000100	45	00000100	000001010101
14	110100	00000111	46	00000101	000001010110
15	110101	000011000	47	00001010	000001010111
16	101010	0000010111	48	00001011	000001100100
17	101011	0000011000	49	01010010	000001100101
18	0100111	0000001000	50	01010011	000001010010
19	0001100	00001100111	51	01010100	000001010011
20	0001000	00001101000	52	01010101	000001000100
21	0010111	00001101100	53	00100100	000000110111
22	0000011	00000110111	54	00100101	000000111000
23	0000100	00000101000	55	01011000	000000100111
24	0101000	00000010111	56	01011001	000000101000
25	0101011	00000011000	57	01011010	000001011000
26	0010011	000011001010	58	01011011	000001011001
27	0100100	000011001011	59	01001010	000000101011
28	0011000	000011001100	60	01001011	000000101100
29	00000010	000011001101	61	00110010	000001011010
30	00000011	000001101000	62	00110011	000001100110
31	00011010	000001101001	63	00110100	000001100111

sequence of
of a lengthy
e. As an ex-
many com-
count. The
th an "IGS"
63. Longer

l symbol al-
g is particu-
. The most
itting docu-

sional image
in fact, docu-
the position
dimensional
is defined to
most 8.5 in. by
line and 1188
tion with the
of individual
it is doubled
National Tele-
80 × 640 or
standard TV

ch position in
Thus, the sig-
the B and W
an line across
and W levels.
n of B and W
isted in Table
ach run length
makeup code-
t are multiples
significant bits
r W) of length
h run of length
also defined in

the code, which indicates that no black pixels follow; hence, the next line should be started, which is akin to a carriage return on a typewriter [23].

Enco

Example 13.8 Run-Length Code

Use the modified Huffman code to compress the line

200 W, 10 B, 10 W, 84 B, 1424 W

consisting of 1728 pixel elements.

Solution

Using Table 13.1, we determine the coding for this pattern to be (the spaces are for the ease of reading)

010111	10011	0000100	00111	0000001111	00001101000	000000000001
192W	8W	10B	10W	64B	20B	EOL

Only 56 bits are required to send this line containing a sequence of 1728 bits.

13.7.3.2 Lempel-Ziv (ZIP) Codes

A major difficulty in using the Huffman code is that the symbol probabilities must be known or estimated, and both the encoder and decoder must know the coding tree. If a tree is constructed from an unusual alphabet at the encoder, the channel connecting the coder and encoder must also deliver the coding tree as a header for the compressed file. This overhead would reduce the compression efficiency realized by building and applying the tree to the source alphabet. The Lempel-Ziv algorithm and its numerous variants use the text itself to iteratively construct a parsed sequence of variable length codewords that form a code dictionary.

The code assumes that a dictionary exists containing already-coded segments of a sequence of alphabet symbols. Data is encoded by looking through the existing dictionary for a match to the next short segment in the sequence being coded. If a match is found, the code operates under the following philosophy: since the receiver already has this code segment in its memory, there is no need to resend it, we only have to identify the address to retrieve the segment. The code references the location of the segment sequence and then appends the next symbol in the sequence to form a new entry in the code dictionary. The code starts with an empty dictionary, so the first elements are entries that do not refer to earlier entries. In one form of dictionary, we recursively form a running sequence of addresses and the segment of alphabet symbols contained therein. Encoded data is made up of a two-word packet of the form < dictionary address, next data character >, with each new data-character entry in the dictionary formed as a packet containing the address of that character, followed by the next character. An example of this coding technique follows:

Enco

Encode the sequence of symbols [a b a a b b b b b b a b b b b a]

Encoded packets:	<0,a>	<0,b>	<1,a>	<2,a>	<2,b>	<5,b>	<5,a>	<6,b>	<4,->
Address:	1	2	3	4	5	6	7	8	
Content:	a	b	aa	ba	bb	bbb	bba	bbbb	

The initial packet <0,a> shows a zero address because there are no entries in the dictionary yet. In this packet, the character a is the first in the data sequence and it is assigned to address 1. The next packet <0,b> contains the second data character b which had not been in the dictionary yet (hence the address portion is 0); b is assigned to address 2. The next packet <1,a> represents the encoding of the next two data characters aa by calling out address 1 for the first a, and appending to that address the next character a. The pair of data characters aa is assigned to address 3. The next packet <2,a> represents the encoding of the next two data characters ba by calling out address 2 for the character b, and appending to that address the next character a. The pair of data characters ba is assigned to address 4, and so forth. Notice how the run-length coding is accomplished. The eighth packet is made up of address 6, containing three b characters followed by another b. For this example, the encoded data can be described with a three-bit address and a 0/1 to identify the appended character. There is a sequence of 9 symbols in the coded sequence, for a total of 36 bits to encode the data consisting of 20 characters. As in many compression schemes, coding gain is not realized for short sequences, as in this example, and is only seen for long sequences.

In another form of the Lempel-Ziv algorithm, the encoded data is represented as three word packets of the form <number of characters back, length, next character>. Here, the concept of an address is not used; instead, previous data sequences are referenced. Also, recursive references are permitted in the length parameter. This is demonstrated in the following example, presented as entry <1, 7, a>:

Encode the sequence of symbols [a b a a b b b b b b a b b b b a]

Encoded packets:	<0,0,a>	<0,0,b>	<2,1,a>	<3,2,b>	<1,7,a>	<6,5,a>
Content:	a	b	aa	bab	bbbbbbba	bbbbba
Running text:	a	ab	abaa	abaabab	abaababbbbbbbba	whole seq.

Here too the coding gain of the process has not been demonstrated for the short run of data. Variations of the code limit the size of the back-pointer such as 12-bits for a maximum of a 4096 point back reference. This limitation constrains the size of memory required for the dictionary and reduces the likelihood of memory overflow. Variations may also constrain the length of the prefix or phrase defined by the first two arguments <back *n*1, forward *n*2, xxx> to be less than some value such as 16 to limit complexity of back search during encoding. The Lempel-Ziv algorithm exists in many commercial and share-ware forms, which include LZ77, Gzip, LZ78, LZW, and UNIX *compress*.

13.8 EXAMPLES OF SOURCE CODING

Source coding has become an essential subsystem in modern communication systems. High demand for bandwidth and storage capacity has been the motivator, while the integrated-circuit and signal-processing techniques have been the enablers. A secondary enabler for wide acceptance of a process in a communication system is the definition of industry-wide *standards* that permit multiple suppliers to field cost-effective and competitive realizations of the coding process. CCITT standards can be found for source coding or compression algorithms for speech, for audio, for still images, and for moving images. In this section, we will examine a number of standard-based source-coding algorithms to demonstrate the wide dissemination of source coding in communication systems as well as to illustrate typical levels of performance.

13.8.1 Audio Compression

Audio compression has become well entrenched in consumer and professional digital audio products such as the compact disc (CD), digital audio tape (DAT), the mini-disc (MD), the digital compact cassette (DCC), digital versatile disc (DVD), digital audio broadcasting (DAB), and motion picture experts group (MPEG) audio layer 3 (MP3) distribution on the internet. In addition, speech compression for telephony, and in particular, cellular telephony, required to preserve bandwidth and battery life has spawned a large number of speech compression standards. Different algorithms are applied to speech signals and wider bandwidth consumer entertainment signals. Audio and speech compression schemes can be conveniently partitioned into applications reflecting some measure of acceptable quality. The parameters describing this partition follow [24, 25]:

Typical parameters values for three classes of audio signals

	Frequency Range	Sampling Rate	PCM bits/ Sample	PCM bit-rate
Telephone Speech	300–3,400 Hz	8 kHz	8	64 kb/s
Wideband Speech	60–7,000 Hz	16 kHz	14	224 kb/s
Wideband Audio	10–20,000 Hz	48 kHz	16	768 kb/s

A/ μ -law
input

13.8.1.1 Adaptive Differential Pulse-Code Modulation

We start this discussion with telephone speech processing. One standard that addresses this area is the CCITT, G. 726 adaptive differential pulse-code modulation (ADPCM) codec. This standard encodes sample by sample, predicting the value of each sample from reconstructed speech of previous samples using an adaptive feedback predictor. It accepts toll-quality 8-bit linear, A-law, or μ -law sampled speech at 64-kbits/sec, and it outputs compressed speech at rates of 16, 24, 32, and 40-kbits/sec. The encoder uses a decoder in its feedback path to analyze and modify algorithm parameters in order to minimize the reconstruction errors. The pre-

communication system the motivator, have been the encoder communication multiple suppliers to assess. CCITT standards for speech, for we will examine a state the wide distortion to illustrate typical

professional digital tape (DAT), the compact disc (DVD), the group (MPEG) speech compression preserve bandwidth on standards. Different consumer encoder be conveniently able quality. The

PCM bit-rate
64 kb/s
224 kb/s
768 kb/s

One standard that uses pulse-code modulation, predicting the samples using an adaptive or μ -law sampled rates of 16, 24, 32, and analyze and modern errors. The pre-

dictor uses a sixth-order filter to model zeros and a second-order filter to model poles of the input signal source. A block diagram of the encoder is seen in Figure 13.36.

13.8.1.2 Subband-Partitioned, Adaptive Differential Pulse-Code Modulation

The CCITT G.722 is a wideband speech-coding standard. The wideband compression offers significant improvement in quality over that of telephone quality speech and is closer to broadcast quality speech and music signals. This coder uses complementary low-pass and high-pass filters to separate the 7-kHz input bandwidth sampled at 16-kHz into a higher and lower subband, each downsampled to 8-kHz. Both low-pass and high-pass filters, as well as the resampling operation, are implemented in a digital filter known as a *quadrature mirror filter*. Independent ADPCM encoders process the reduced bandwidth time series from the two filters and form 48-kbit/s and 16-kbit/s data rates, respectively, as the low and high band outputs. These coders are a modified version of CCITT B.721 ADPCM speech coders that use backward prediction filters based on the coded difference signal. Discarding the least significant bit of the prediction filter coefficients permits this encoder to operate at 56 and 48 kbit/s, as well as the nominal 64 kbit/s. The reduced bit rate allows the communication system to assign the unused bits to an auxiliary data stream operating at 8 and 16-kbit/s when the channel supports a fixed output rate of 64 kbit/s. The predictor uses a pole-zero structure with 6-zeros and 2-poles. A block diagram of the wideband 64-kbit/s audio coder is shown in Figure 13.37.

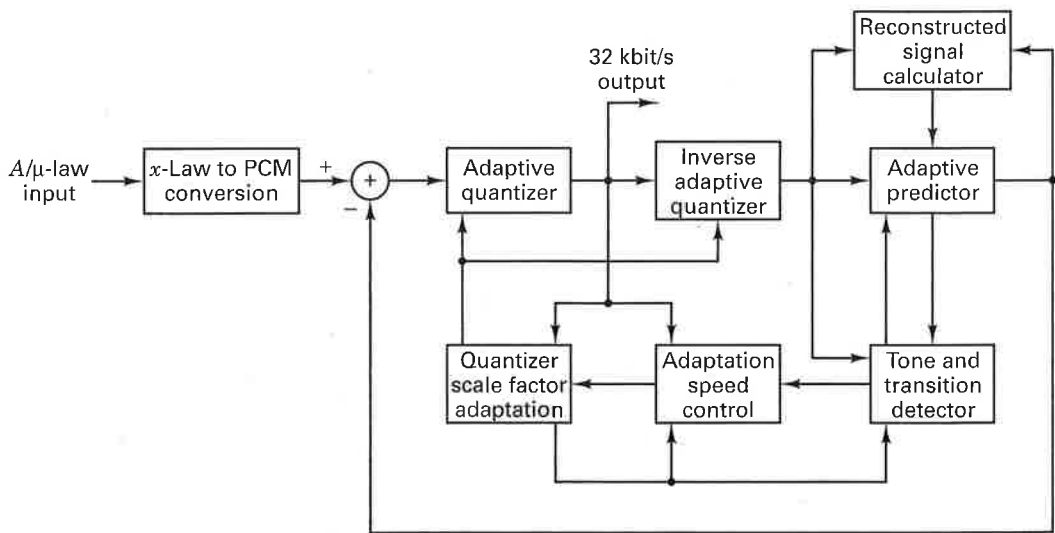


Figure 13.36 Adaptive differential pulse code modulator speech codec (G.726).

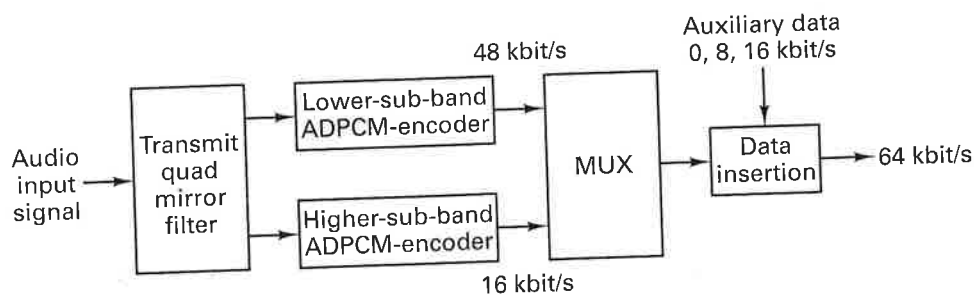


Figure 13.37 Wideband QMF-ADPCM (64-kbit/s) codec (G.722).

13.8.1.3 Codebook-Excited Linear-Predictive Coding (CELP)

Speech coders using linear predictive filters can provide high quality encoded speech at rates above 16 kbits/sec, but they degrade quickly at lower rates. The LPC encoders can be modified to obtain high-quality speech compression at rates on the order of 4.8 to 9.6 kbits/s by recasting the synthesis problem as a two-step procedure called *synthesis by analysis*. In the first step, we form a 10-th order LPC model of the signal that is valid over a short interval—say, every 20 ms. In the second step, we find the waveform that when applied to the LPC model forms an output signal as close as possible to the original signal being synthesized. We accomplish this task by sequentially applying candidate excitation waveforms to the model and comparing each synthesized wave-shape with the original signal and then selecting the one that minimizes the error between the original signal and the output of the driven model.

From the physiology of the speech-generation process we know that the excitation for the speech process often consists of periodic pulses (formed by the vibrating vocal chords). The period P of the periodic pulses is related to the speakers pitch. A one-tap recursive filter is defined by two parameters; P for the number of delay intervals in the feedback path and g , the feedback gain. The impulse response of this filter is a decaying sequence with P zero-value output samples between successive non-zero output samples. The output of this filter generates the periodic excitation signal applied to the input of the LPC model (see Section 13.3.2). The synthesis algorithm has to test possible values of P from a list of candidate values. The two pitch parameters are estimated every 5-milliseconds. The input to the pitch filter is extracted from a table of candidate excitation sequences. The output of the pitch filter in turn drives the LPC model. The table, typically containing 1024 entries, is called a codebook. The codebook is accessed every 2.5 ms. When the best combination of codebook entries and pitch period is determined by exhaustive search, a frame is assembled containing the sequence of pitch parameters, the sequence of codebook addresses, and information about the LPC coefficients.

The encoder must deliver parameters describing the LPC model to the decoder. The LPC filter spectral response is very sensitive to coefficient quantization and as such would have to be represented with an unacceptably large number of

bits. The LPC coefficients are transformed to a different set of parameters called *Line Spectral Pairs* [10] that are insensitive to quantization

Systems designed to the IS-95 standard use the following LPC frame format. The frame required to describe 20 ms of data contains 192 bits assigned to represent encoded parameters in the following manner:

10 LPC Coefficients	40 bits
4 Lag and Gain Parameters	40 bits
8 Codebook Addresses	80 bits
Parity, Check bits, and Overhead	32 bits

The overall bit rate for this system is 192 bits per 20 ms, or 9600 bits/s. The bit rate can be reduced if the encoder detects speech pauses.

13.8.1.4 MPEG Layers I, II and III

The International Standards Organizations (ISO) and Motion Picture Experts Group (MPEG) audio coding standard describes audio compression for synchronized audio to accompany the compressed video known as MPEG. It combines features of MUSICAM (Masking pattern adapted Universal Subband Integrated Coding And Multiplexing) and ASPEC (Adaptive Spectral Perceptual Entropy Coding). It consists of three layers (codes) of increasing complexity and improving subjective performance, and it operates with input sampling rates of 32, 44.1 and 48 kHz, and it outputs bit rates per monophonic channel between 32 and 192 kbit/s, or per stereophonic channel between 64 and 384 kbit/s. The standard supports single channel mode, stereo mode, dual channel mode (for bilingual audio programs), and an optional joint stereo mode. In this later mode, the two coders for the left and right channel can support each other by exploiting common statistics between these channels in order to compress the audio bit rate to an even higher degree than is possible in monophonic transmission [26].

The encoder operates in conjunction with a real-time model of the *human spectral perception threshold*. This threshold is a frequency-dependent boundary or threshold that marks sound pressure levels (SPL) below which the human ear cannot detect sounds. This curve, called the *acuity threshold* is generated during a hearing test. The acuity threshold is normally presented as amplitude levels as a function of spectral position, in much the same way as a power-spectrum curve. This threshold is a time-varying function of the short-time power spectral density and is locally peaked in response to high-level tones and tone-like signals (called tonals). The lifting of the threshold due to the presence of strong tonals results in a local masking of spectral components below the new threshold level. Signal spectral components below the threshold level that cannot be heard are declared irrelevant, and they are not encoded in the compression process. Signals above the frequency-dependent threshold are encoded with sufficient fidelity to keep the approximation error below the acuity threshold. This is accomplished by partitioning the spectrum with a set of narrow-band filters and assigning a sufficient number of bits to describe each filter output in proportion to its amplitude above the

threshold. Thus, a signal that is 30 dB above threshold in a particular band will be assigned 5-bits for its quantization, for which case the quantization noise falls below the threshold, since the quantization noise-to-signal ratio is reduced by 6 dB per bit. A typical threshold acuity curve is presented in Figure 13.38.

The encoder operates in this manner. The standard 16-bit PCM audio signal is windowed and converted into spectral subband components by a polyphase filter bank consisting of 32 equally spaced bandpass filters. The filter bank is designed with adjacent channel attenuation exceeding 96 dB, a level required to suppress perceptual distortion caused by quantization noise. The filtered output signals are downsampled to the Nyquist rate for each bandpass bandwidth. In the decoder, this process is reversed. The sampling rate of each subband filter is increased to that of the original source signal by interpolating the subband signals formed at the bandpass outputs of the synthesis filter bank. Figure 13.39 presents a block diagram of the Layer I and II MPEG Audio Encoder and Decoder.

In Layer III of the MPEG/ISO (MP3) standard, a higher frequency resolution is achieved that closely matches critical frequency-resolution bandwidths of the human auditory process. This enhanced partition is achieved by post-processing the 32 subband signals with an overlapped and windowed 6-point or 18-point modified Discrete Cosine Transform (MDCT). (A short description of the DCT is presented in the next section on image compression.) The resulting number of frequency bands that can be resolved in Layer 3 is 32×18 or 576, which each filter representing a bandwidth of 24,000/576 or 41.67 Hz. Layer III differs from layer I and II by the addition of a modified DCT to the analysis bank, a Huffman coder to the output of the quantizer, and a side information data path.

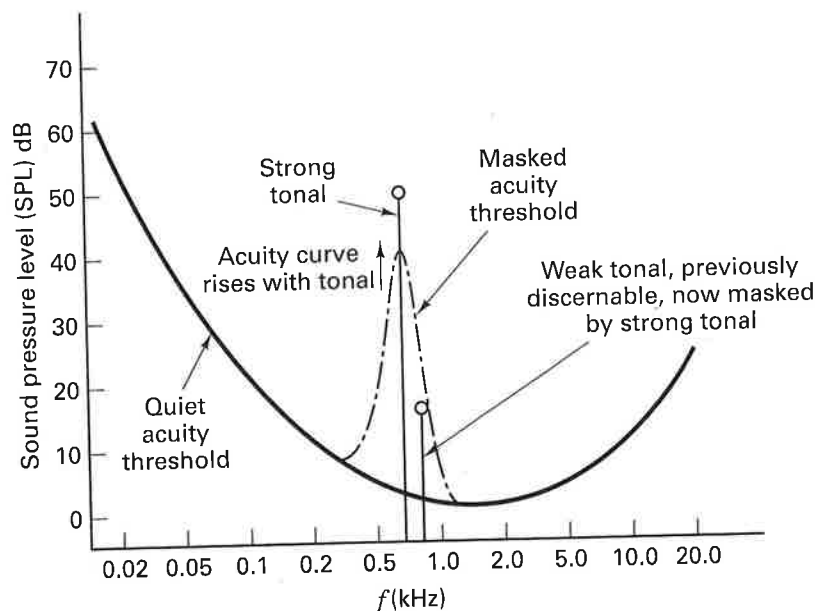


Figure 13.38 Acuity threshold and auditory masking.

r band will be
on noise falls
duced by 6 dB

M audio signal
polyphase filter
bank is designed
to suppress per-
output signals are
in the decoder,
is increased to
s formed at the
block diagram

ency resolution
widths of the
post-processing
18-point modi-
the DCT is pre-
ing number of
which each filter
ers from layer I
uffman coder to

Previously
now masked
tonal

0.0

9.

oding Chap. 13

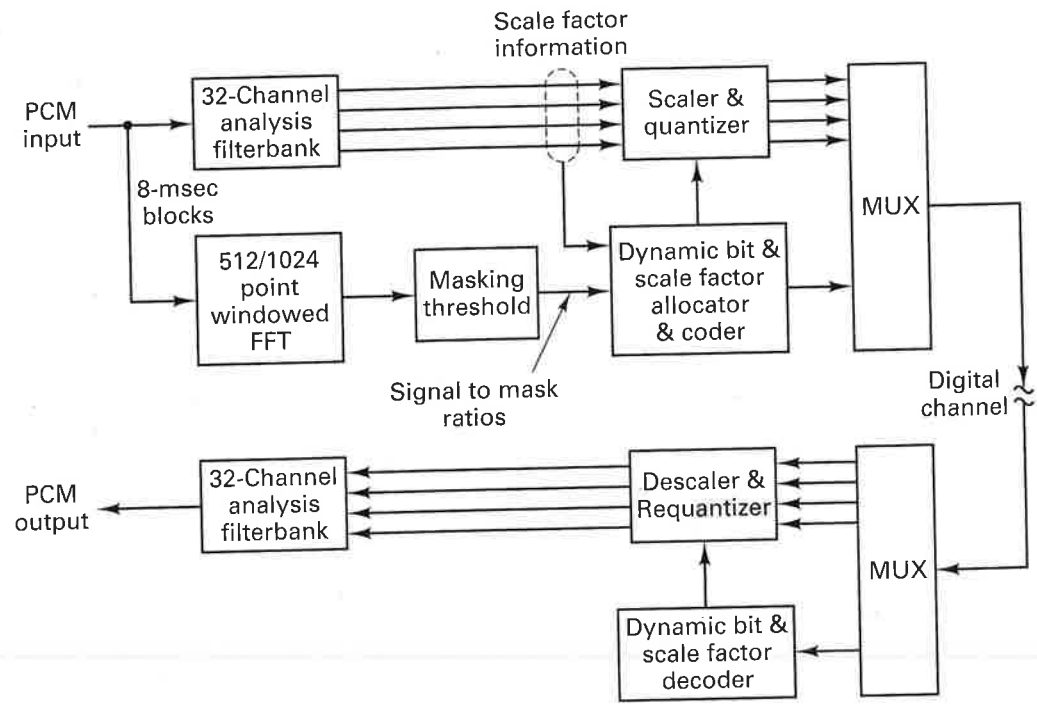


Figure 13.39 Block diagram of ISO/MPEG audio encoder and decoder, layers I and II.

13.8.2 Image Compression

An old saying we have heard many times is that, *A picture is worth a thousand words*. Is this saying true? A 1000 words contain 6000 characters, which, when coded as 7-bit ASCII symbols, require a total of 42,000 bits. What size image (or picture) can we describe with 42,000 bits? If we use a monochrome (i.e. black and white) image with a standard 8-bit grey scale the image would be restricted to 5250 pixels (or picture elements). This image would be of dimension 70 by 75 pixels, and if we assume a medium quality image is formed with 300 pixels per inch, we find that our image is approximately $\frac{1}{4}$ inch by $\frac{1}{4}$ inch. We certainly have need for some image coding.

Let us approach the problem from the opposite direction. How big is an image? Selecting an 8.5 by 11.0 inch sheet of paper containing an image resolved at 300 pixels per inch, we find the image contains $8.5 \times 300 \times 11.0 \times 300$ or 8.4×10^6 picture elements. If the picture is full color with three colors per element, each described by 8-bit words, we find the image contains 2×10^8 bits which is equivalent to 4.8×10^6 6-character ASCII words. Perhaps the old saying should be updated to reflect that *A picture is worth about five million words*. For comparison with other image formats, we note that a single frame of a high definition television image con-

tains approximately 1.8×10^6 pixels, a standard television image contains approximately 0.33×10^6 pixels, and high-end computer monitors contain 1.2 -to- 3.1×10^6 picture elements.

Technology has presented us with low-cost, high-resolution color printers, scanners, cameras, and monitors, enabling us to capture and present images for commerce and entertainment. The storage and transport of these images rely quite heavily on source coding to reduce the demands on bandwidth and memory. There are numerous standards that have been developed to compress images. In the next section, we examine elements of two primary compression schemes [26, 27].

13.8.2.1 Joint Photographic Experts Group (JPEG)

JPEG is the common name given to the ISO/JPEG international standard 10918-1 or ITU-T Recommendation T.81 standard for "Digital Compression of Continuous-Tone Still Images." JPEG is primarily known as a transform-based lossy compression scheme, and this is the mode that is examined here. Lossy compression permits errors in the signal construction. The error levels are restricted to be below the perception threshold of a human observer. JPEG supports three modes of operation related to the Discrete Cosine Transform (DCT): Sequential DCT, Progressive DCT, and Hierarchical, as well as a lossless mode using differential prediction and entropy coding of the prediction error. The DCT is a numerical transform related to the Discrete Fourier transform (DFT) for obtaining the spectral decomposition of even symmetric sequences. When the input sequence is even symmetric, there is no need for the sine components of the transform; hence, a DCT can replace the FFT.

Let us start with an introduction to the 2-D 8-by-8 DCT. We first comment on the use of the DCT to form a spectral description of an 8-by-8 pixel block. The 2-D DCT is a separable transform that can be written as a double sum over the two dimensions. The separable DCT performs eight 8-point DCTs in each direction; hence, the basic building block is a single 8-point DCT. The first question to address is, Why use a DCT rather than some other transform, such as the DFT? The answer is related to the interaction between the sampling theorem and Fourier transforms. Sampling in one domain induces periodicity in the other. When we sample a time series, its spectrum becomes periodic. On the other hand, when we sample the spectrum of a time series, the time series is periodically extended. We acknowledge this effect by labeling this process the *periodic extension* and by labeling the resulting processing a *periodogram*. The periodic extension of the original data, (shown in Figure 13.40) exhibits discontinuities at the boundaries that limit the rate of spectral decay in the spectrum to $1/f$. We can form the even-extension of the data by reflecting the data about one of its boundaries. When this data is periodically extended, as indicated in Figure 13.40, discontinuities no longer reside in the amplitude of the data but rather in its first derivative, so that the rate of spectral decay increases to $1/f^2$. The faster rate of spectral decay means that there will be a reduced number of significant spectral terms. A second advantage to the DCT is that since the data is even symmetric, its transform is also real and symmetric and hence has no need for the odd symmetric basis terms, the sine functions.

tains approxi-
2-to- 3.1×10^6

color printers,
nt images for
ges rely quite
emory. There
es. In the next
[26, 27].

onal standard
ompression of
ansform-based
e. Lossy com-
e restricted to
upports three
T): Sequential
using differen-
is a numerical
ining the spec-
sequence is even
form; hence, a

st comment on
block. The 2-D
ver the two di-
each direction;
question to ad-
the DFT? The
m and Fourier
ther. When we
er hand, when
cally extended.
ctension and by
sion of the orig-
boundaries that
form the even-
uries. When this
uities no longer
so that the rate
decay means that
nd advantage to
so real and sym-
e sine functions.

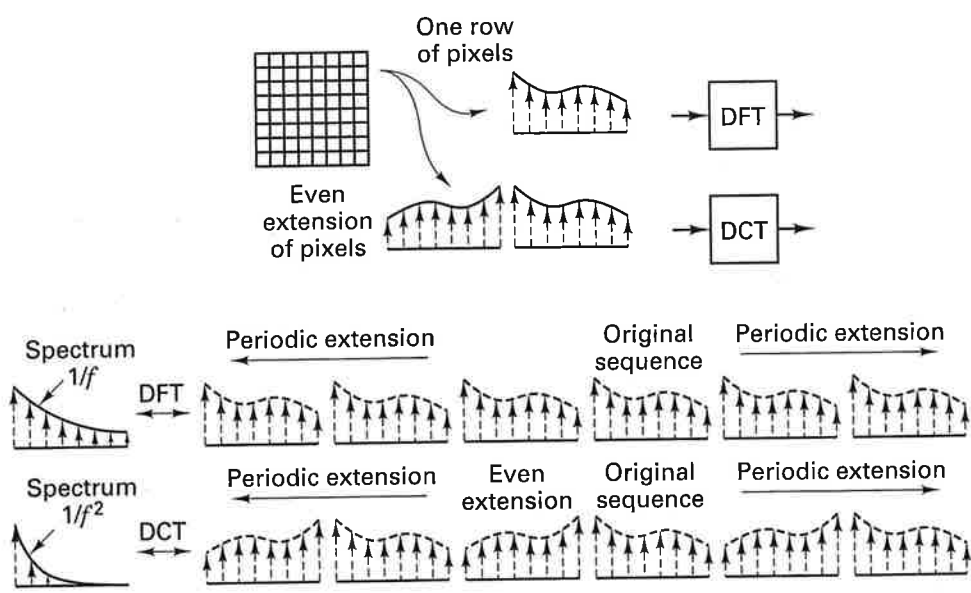


Figure 13.40 Spectral decay & periodic extension of time series by DFT & DCT.

Since the amplitude of an image is highly correlated over short spatial intervals, the DCT of an 8-by-8 block of pixels is dominated by the dc term and further contains relatively few significant terms. A typical set of amplitudes and their DCT is suggested in Figure 13.41. Note that the spectral terms fall at least as fast as $1/f^2$, and that most of the high frequency terms are essentially zero. The spectrum is passed to a quantizer that uses standard quantization tables to assign bits to the spectral terms in proportion to their amplitude, as well as their psychovisual contribution. Different quantization tables are used for the luminance and chrominance components.

To take advantage of the large number of zero-valued entries in the quantized DCT, the spectral addresses of the DCT are scanned in a zigzag pattern, as shown in Figure 13.42. The zigzag pattern assures a long run of zeros. This improves the coding efficiency of the run-length Huffman code describing the spectral samples. Figure 13.43 presents the block diagram of a JPEG encoder. The signal delivered to the encoder is normally presented as raster-scanned and sampled primary additive colors, red, green and blue, ($R-G-B$). The color planes are transformed to luminance (Y) and chrominance components $0.564 \times (B-Y)$ (denoted C_B) and $0.713 \times (R-Y)$ (denoted C_R), using a variation of the color difference transformation developed for color TV. This mapping is

$$\begin{bmatrix} Y \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

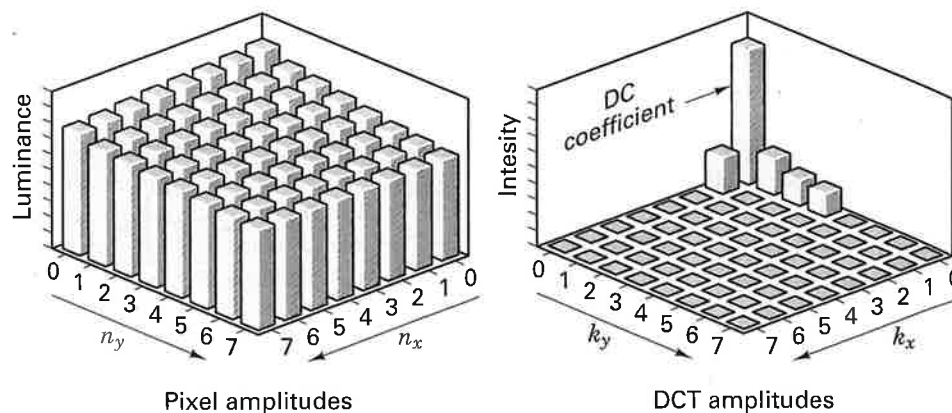


Figure 13.41 Pixel and DCT amplitudes describing same 8-by-8 block of pixels.

where the Y component is formed to reflect the human-eye sensitivity to the primary colors.

The human eye exhibits a different acuity for the color components of an image than it does for the luminance (black and white) component. This difference in resolution capability is the result of the distribution of the color receptors (rods) and luminance receptors (cones) on the retina. The human eye can resolve 1-inch alternating black and white stripes at 180 ft (1/40-th degree). By comparison, 1-inch color stripes of blue-red or blue-green are not resolvable at distances greater than 40 feet (1/8-th degree). Hence, three-color images require about 1/25-th (1/5-th in each direction) more data than is required to form a black-and-white image. Early photographers knew the eye required very little color detail to color an image, and a lively industry existed in which artists hand-colored black and white photographs and postal-cards. Both analog and digital color TV takes advantage of this acuity difference to deliver the additional color components in a significantly reduced bandwidth. NTSC delivers all three colors in a 0.5 MHz bandwidth, rather than the

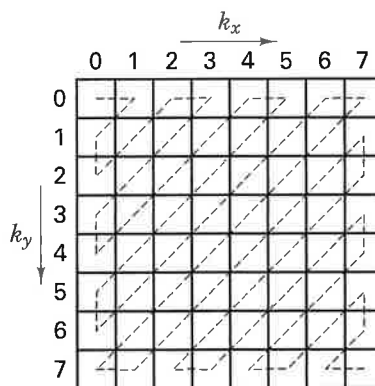


Figure 13.42 Zigzag scan pattern for DCT spectral terms.

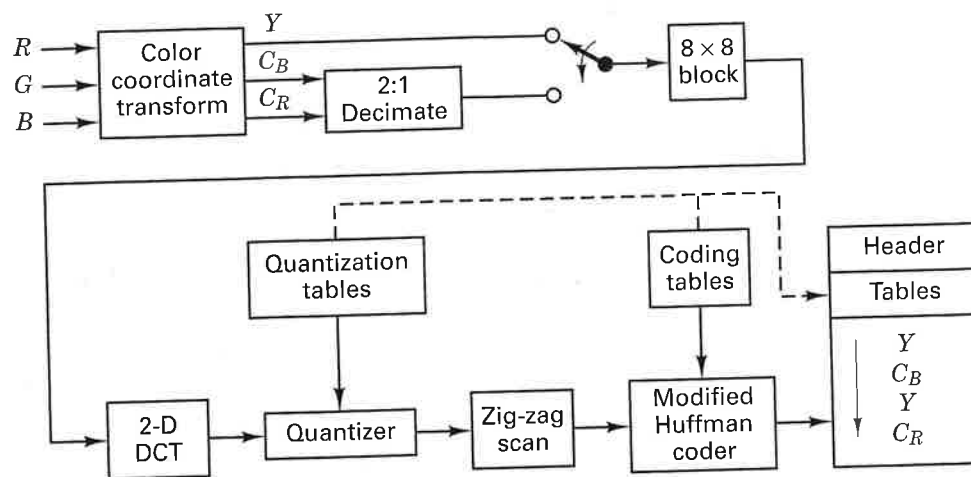


Figure 13.43 Block diagram of JPEG encoder.

4.2-MHz bandwidth required for the luminance component. JPEG similarly takes advantage of the color acuity difference and down samples the color difference components 2-to-1 in the scan direction (x), but not in the successive line (y) direction.

The luminance and down-sampled color difference signals are sequentially presented as 8-by-8 blocks to the 2-D DCT. The outputs of the DCT are quantized by the appropriate table and then zigzag scanned for presentation to the Huffman coder. JPEG uses the Huffman coder to encode the ac-coefficients of the 2D-DCT, but since the DC components are highly correlated over adjacent blocks, uses differential encoding for the DC component. The decoder, of course, reverses these operations to form an image.

13.8.2.1.1 Variations of JPEG Decoding. During reconstruction of the image, the decoder can operate sequentially, starting in the upper left corner of the image and forming 8-by-8 pixel blocks as they arrive. This is the sequential mode of JPEG. In the progressive coding mode, the image is first assembled in 8-by-8 blocks formed by only the dc component in each block. This is a very quick process that presents a coarse but recognizable preview image, a process often seen on the internet when downloading GIF (Graphic Interchange Format) files which has delivered the only dc components at the beginning of the data transfer. The image is then updated with each 8-by-8 block formed from the dc component and the first two dc components which are the next set of data delivered to the decoder. Finally, the image is updated at full resolution formed by the full set of coefficients associated with each 8-by-8 block.

In Hierarchical Coding, the image is encoded and decoded as overlapped frames. A low resolution, down sampled 4-to-1 in each direction image is encoded using the DCT and quantized coefficient processing to form the first frame. The image formed by this frame is up sampled and compared with a higher resolution,

down-sampled (2-to-1 in each direction) version of the original image and the difference, representing an error in the formation of the image, is again coded as an MPEG image. The two frames formed by the two layers of coding are used to form a composite image that is up-sampled and compared with the original image. The difference between the original image and two lower level resolution reconstructions is formed at the highest resolution and JPEG-encoded once again. This process is useful for delivering images with successively higher quality reconstruction in a way similar to the progressive coding. The difference here is that the additional resolution is available but may not be sent unless requested. An example might be a user scanning an image library and requesting the final quality image after reviewing a set of images. Another application might be the delivery of one level of quality to a display on a personal computer but a higher level of quality to a display on a high-resolution workstation.

As a final comment on JPEG, JPEG-2000 is a JPEG initiative to define a *New Image Coding System*, addressing internet and mobile applications. This system offers low bandwidth, multiple resolution, error resilience, image security, and low complexity. It is based on wavelet compression algorithms, and relative to JPEG, it offers improved compression efficiency with multiple resolution capabilities [28].

13.8.2.2 Motion Picture Experts Group (MPEG)

MPEG is a set of standards designed to support *Coding of Moving Pictures and Associated Audio—for digital storage media at up to 1.5 Mbits/s*. MPEG-1, ISO standard 11172 approved in November 1992, was designed to permit full motion video recordings on CD players originally designed for stereo audio playback. MPEG-2, ISO standard 13818 or ITU T-recommendation H.262, *Generic Coding of Moving Pictures and Associated Audio*, approved in November 1994, addressed greater input-output format flexibility, data rates, and system considerations such as transport and synchronization, topics neglected in MPEG-1. MPEG-2 supports variations of Digital TV covering digitized video that matches existing analog formats with selectable quality through DVD (digital video disc) and HDTV (high definition television) with different aspect ratio, line rate, pixel span, scan conversion options, and various resampling options for the color difference components. The following section describes the basic theory of operation of the simplest version of MPEG-2.

MPEG-2. MPEG compresses a sequence of moving images by taking advantage of the high correlation that exists between successive pictures of a moving picture. MPEG constructs three types of pictures: Intra pictures (*I*-pictures), Predicted pictures (*P*-pictures), and Bi-directional Prediction pictures (*B*-pictures). In MPEG, every *M*-th picture in a sequence can be fully compressed using a standard JPEG algorithm; these are the *I*-pictures. The process then compares successive *I*-pictures and identifies portions of the image that have moved. The image sections that didn't move are carried forward in time to intermediate pictures by the decoder memory. The process then selects a subset of intermediate pictures, then pre-

and the dif-
 coded as an
 sed to form
 image. The
 reconstruc-
 again. This
 reconstruc-
 at the addi-
 An example
 ality image
 ivery of one
 quality to a

to define a
 ns. This sys-
 security, and
 l relative to
 ion capabili-

ving Pictures
 IPEG-1, ISO
 t full motion
 io playback.
 neric Coding
 4, addressed
 erations such
 G-2 supports
 g analog for-
 HDTV (high
 scan conver-
 components.
 simplest ver-

by taking ad-
 s of a moving
 ictures), Pre-
 3-pictures). In
 ng a standard
 res successive
 mage sections
 es by the de-
 res, then pre-

dicts (via linear interpolation between *I*-pictures) and corrects the location of the image sections that were found to have moved. These predicted and corrected images are the *P*-pictures. The pictures between the *I*- and *P*-pictures are the *B*-pictures that incorporate the stationary image sections uncovered by moving sections. The relative position of these picture is shown in Figure 13.44. The relative position of these pictures is shown in Figure 13.44. Note that *P* and *B*-pictures are permitted but not required, and their quantity is each variable. A sequence can be formed without any *P* or *B* pictures, but a sequence containing only *P* or *B*-pictures cannot exist.

The *I*-pictures are compressed as if they were JPEG images. This compression is applied to four contiguous 8-by-8 blocks called a macro block. The macro blocks can be down sampled for subsequent compression of the chrominance components. The macro blocks and their down sampling options are shown in Figure 13.45. Compression of the *I*-frame is performed without reference to any earlier or later pictures in the frame sequence. The distance in sequence count between *I*-pictures is adjustable, and it can be as small as 1 so that the *I*-pictures are adjacent, or as large as reconstruction memory would allow. Editing cuts in a sequence of pictures and local program insertion can only occur at *I*-pictures. Since one-half second is acceptable time accuracy for executing such editing, then the distance between *I*-pictures is usually limited to approximately 15 image pictures for the NTSC standard of 30 pictures per second, or 12 image pictures for the British standard PAL (phase alternating lines) of 25 pictures per second.

The first processing step performed by MPEG is the task of determining which macro blocks have moved between *I*-pictures. This is determined by carrying each macro block from one *I*-frame forward to the next and performing a two-dimensional cross correlation in the neighborhood of its original location. Motion vectors are determined for each macro block that identifies the direction and the amount of movement of each shifted macro block. Macro blocks that have not shifted are stationary in the image pictures between the *I*-pictures and can be carried forward in the intermediate image pictures.

The next processing step in MPEG is to form the *P*-frame between the *I*-pictures. We first assume that the shifted macro blocks have moved linearly in time between the two positions identified in the first processing step. Each macro block is placed at its predicted location on the *P*-frame and is cross-correlated in its neighborhood to determine the true location of the macro block in the *P*-frame. The difference between the predicted and true position of the macro block is an error in the prediction and this error is compressed using the DCT and is used to correct the *P*-frame. The same information is forwarded to the decoder so that it

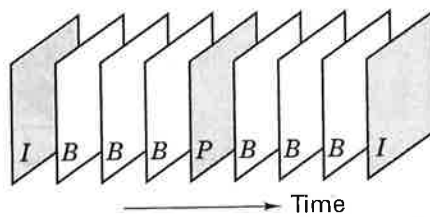


Figure 13.44 Sequence of pictures in MPEG compression.

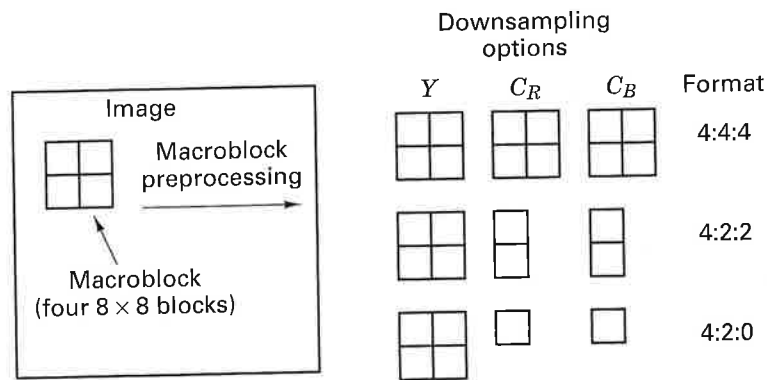


Figure 13.45 Preprocessing of macro block for chrominance down sampling.

can correct its predictions. Figure 13.46 presents the macro block shift between *I*-pictures and an intermediate *P*-picture.

B-pictures are image pictures located between *I*-pictures and *P*-pictures. In these pictures the motion vectors move the shifted macro blocks linearly through time to their bi-directional-interpolated positions at each successive *B*-frame in the sequence. *I*-pictures require the maximum amount of data to describe their DCT compressed contents. *P*-pictures require less data, having only to describe the error pixels in the motion predicted macro blocks in the frame. The rest of the pixels in the frame are carried forward in memory from the previous *I*-frame. *B*-pictures are the most efficient pictures in the set. They only have to *linearly shift and correct* the pixels newly covered and uncovered due to the motion of macro blocks on that image frame.

Reconstruction of the images at the decoder requires that the sequence of images be delivered in the order necessary for appropriate processing. For instance, since the computation of *B*-pictures require information from the *I* and *P* or between the *P*- and *P*-pictures on either side, the *I* and *P* pictures must be delivered first. The following is an example of the required ordering of images at the input and output of the encoder and decoder.

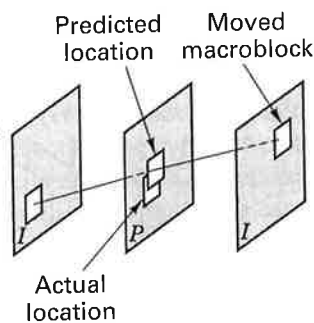


Figure 13.46 Macro block movement between *I*-pictures and *P*-picture.

Picture Order at Encoder Input												
1	2	3	4	5	6	7	8	9	10	11	12	13
I_n	B_1	B_2	P_1	B_3	B_4	P_2	B_5	B_6	I_{n+1}	B_1	B_2	P_1

Encoded Picture Order at Encoder Output and Decoder Input												
1	2	3	4	5	6	7	8	9	10	11	12	13
I_n	P_1	B_1	B_2	P_2	B_3	B_4	I_{n+1}	B_5	B_6	P_1	B_1	B_2

Picture Order at Decoder Output												
1	2	3	4	5	6	7	8	9	10	11	12	13
I_n	B_1	B_2	P_1	B_3	B_4	P_2	B_5	B_6	I_{n+1}	B_1	B_2	P_1

Figure 13.47 presents a block diagram of an MPEG encoder. Notice its structure is the standard predictor-corrector model. An interesting relationship exists between the eye-brain measure of image quality and image activity. On one hand, when the image contains considerable movement, the eye will accept lower quality images; and on the other hand, when the image contains little movement, the eye is very sensitive to artifacts. In the coder, absence of movement affects coding activity and results in lower data rate delivered to the output buffer. The buffer recognizes this as an indicator of stationary images and controls the image by permitting higher quality quantization of the DCT. The output rate of the output buffer is fixed by communication link considerations. Flow control is used to match the average input rate to the fixed output rate. The flow control detects low encoder activity by noting that its buffer is emptying faster than it is filling. A simple indicator of the difference between input and output rates is the location of the output-address pointer. If this pointer is moving towards the beginning of buffer memory, a precursor of memory underflow, the system increases the input rate by selecting a quantization table that delivers a larger

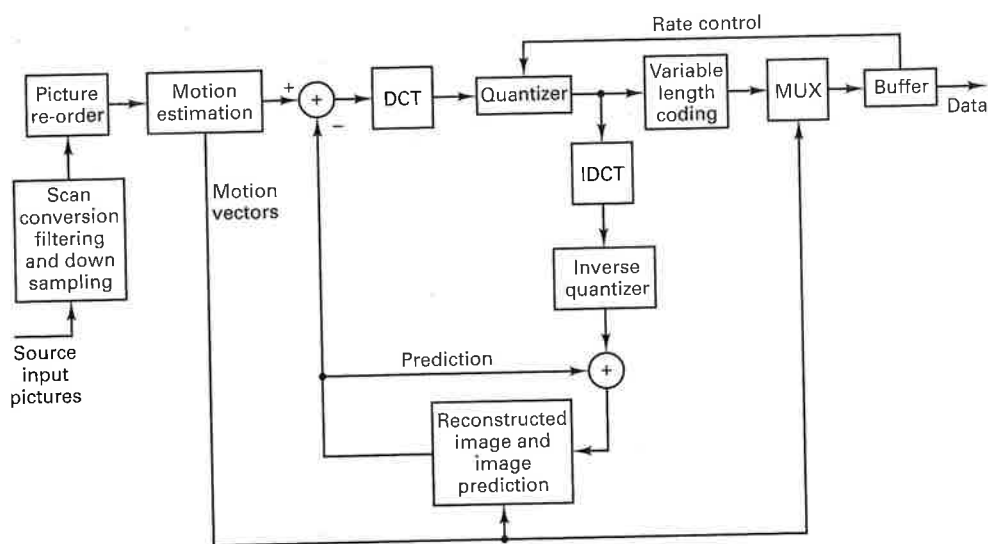


Figure 13.47 Block diagram of MPEG encoder with embedded decoder.

number of bits per DCT. Similarly, if the pointer is moving towards the end of buffer memory, a precursor of over-flow, the system decreases the input rate by selecting a quantization table that delivers a smaller number of bits per DCT. This process matches the image quality to the eye-brain quality threshold while sustaining the average output rate to the channel.

13.9 CONCLUSION

In this chapter, we have presented some of the highlights of source coding. We saw that source coding can be applied to digital data and to waveform signals. Digital data can be reconstructed exactly from a reduced data description of a source, when the source exhibits correlation between alphabet elements, or if the elements are not equally likely. Waveform signals, in general, experience distortion when represented by a digital description. This distortion can be made arbitrarily small by an appropriate increase in bit rate required to describe the source. Source coding can also be applied to waveform sources to obtain reduced-data-rate descriptions, when the source exhibits a long correlation interval or if the possible amplitudes are not equally probable.

The system advantage of source coding is the reduced need for the system resources of bandwidth and/or energy per bit required to deliver a description of the source. This advantage is available in exchange for a third system resource computation and memory. With the cost of these latter resources continuing to fall as they have over the past decades, source coding promises to fill an ever-increasing role in future communication and storage systems. The interested reader is encouraged to examine References [8, 17, 24-26] dealing with source coding.

REFERENCES

1. Papoulis, A., *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill Book Company, New York, 1965.
2. Harris, F. J., "Windows, Harmonic Analysis, and the Discrete Fourier Transform," *Proc. IEEE*, vol. 67, no., Jan. 1979.
3. Martin, G., "Gyroscopes May Cease Spinning," *IEEE Spectrum*, vol. 23, no. 2, Feb. 1986, pp. 48-53.
4. Vanderkooy, J., and Lipshitz, S. T., "Resolution beyond the Least Significant Bit with Dither," *J. Audio Eng. Soc.*, no. 3, Mar. 1984, pp. 106-112.
5. Blesser, B. A., "Digitization of Audio: A Comprehensive Examination of Theory, Implementation, and Current Practice," *J. Audio Eng. Soc.*, vol. 26, no. 10, Oct. 1978, pp. 739-771.
6. Sluyter, R. J., "Digitization of Speech," *Phillips Tech. Rev.*, vol. 41, no. 7-8, 1983-84, pp. 201-221.
7. Bell Telephone Laboratories Staff, *Transmission Systems for Communications*, Western Electric Co. Technical Publications, Winston-Salem, N.C., 1971.
8. Jayant, N. S., and Noll, P., *Digital Coding of Waveforms*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1984.

9. Markel, J. D., and Gray, A. H., Jr., *Linear Prediction of Speech*, Springer-Verlag, New York, 1976.
10. Deller, J., Proakis, J., and Hansen, J., *Discrete-Time Processing of Speech Signals*, Macmillan, New York, 1993.
11. Candy, J., and Temes, G., *Oversampling Delta-Sigma Data Converters*, IEEE Press, 1991.
12. Dick, C., and Harris, F., *FPGA Signal Processing Sigma-Delta Modulation*, IEEE Signal Proc. Mag., Vol. 17, No. 1, Jan 2000, pp 20–35.
13. Cummisky, P., Jayant, N., and Flanagan, J., *Adaptive Quantization in Differential PCM Coding of Speech*, Bell Syst. Tech J., Vol. 52, 1973, pp. 115–119.
14. Gersho, A., “Asymptotically Optimal Block Quantization,” *IEEE Trans. Inf. Theory*, vol. IT25, no. 4, July 1979, pp. 373–380.
15. Gersho, A., “On the Structure of Vector Quantizers,” *IEEE Trans. Inf. Theory*, vol. IT28, no. 2, Mar. 1982, pp. 157–166.
16. Abut, H., *Vector Quantization*, IEEE Press, 1990.
17. Jeffress, L., “Masking,” in J. Tobias, ed., *Foundations of Modern Auditory Theory*, Academic Press, Inc., New York, 1970.
18. Lynch, T. J., *Data Compression Techniques and Applications*, Lifetime Learning Publications, New York, 1985.
19. Schafer, R. W., and Rabiner, L. R., “Design of Digital Filter Banks for Speech Analysis,” *Bell Syst. Tech. J.*, vol. 50, no. 10, Dec. 1971, pp. 3097–3115.
20. Huffman, D. A., “A Method for the Construction of Minimum Redundancy Codes,” *Proc. IRE*, vol. 40, Sept. 1952, pp. 1098–1101.
21. Hamming, R. W., *Coding and Information Theory*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1980.
22. Hunter, R., and Robinson, A., *International Digital Facsimile Coding Standard*, Proc. IEEE, Vol. 68, No. 7, July 1980, pp. 854–867.
23. McConnel, K., Bodson, D., and Urban, S., *FAX: Facsimile Technology and Systems*, Artech House, 1999.
24. Cox, R., *Three New Speech Coders From the ITU Cover a Range of Applications*, IEEE Comm. Mag., Vol. 35, NO. 9, Sept 1997, pp. 40–47.
25. Noll, P., *Wideband Speech and Audio Coding*, IEEE Comm. Mag., Vol. 31, No. 11, Nov. 1993, pp. 34–44.
26. Solari, S., *Digital Video and Audio Compression*, McGraw-Hill, New York, 1997
27. Rzeszewski, T., *Digital Video: Concepts and Applications Across Industries*, IEEE Press, 1995
28. Ebrahimi, T., Santa Cruz, D., Christopoulos, C., Askelöf, J., Larsson, M. “JPEG 2000 Still Image Coding Versus Other Standards”, *SPIE International Symposium*, 30 July–4 August 2000, Special Session on JPEG2000, San Diego, CA.

PROBLEMS

- 13.1. A discrete source generates three independent symbols A , B , and C with probabilities 0.9, 0.08, and 0.02, respectively. Determine the entropy of the source.
- 13.2. A discrete source generates two dependent symbols A and B with conditional probabilities

$$P(A|A) = 0.8 \quad P(B|A) = 0.2$$

$$P(A|B) = 0.6 \quad P(B|B) = 0.4$$

- (a) Determine the probabilities of symbols A and B .
 - (b) Determine the entropy of the source.
 - (c) Determine the entropy of the source if the symbols were independent with the same probabilities.
- 13.3. A 16-bit linear analog-to-digital converter operates over an input range of ± 5.0 V.
- (a) Determine the size of a quantile.
 - (b) Determine the rms quantizing noise voltage.
 - (c) Determine the average SNR (due to quantizing) for a full-scale sinusoidal input signal.
 - (d) Consider that the distance traveled on a 100-mile automobile trip is measured to the same accuracy as that of the 16-bit converter. What is the rms error in feet?
- 13.4. A 10-bit A-to-D converter (ADC) is designed to operate over a full-scale range of ± 5.0 V.
- (a) Determine the size of a single quantile step.
 - (b) For a 5.0-V (full-scale) sinusoid, determine the output signal-to-quantizing noise ratio.
 - (c) For a 0.050-V ($\frac{1}{100}$ of full-scale) sinusoid, determine the output signal-to-quantizing noise ratio.
 - (d) For an input signal with a Gaussian-distributed amplitude, the probability of saturation is controlled by adjusting the input attenuator so that the saturation level corresponds to four standard deviations. Determine the output signal-to-quantizing noise ratio for this case.
 - (e) Determine the probability of signal saturation for the signal described in part (d).
- 13.5. Determine the optimal compression characteristic for the input density function shown in Figure P13.1 (an approximation to a continuous density function).
- 13.6. A 10-bit μ -law converter is designed to operate over a full-scale range of ± 5.0 V.
- (a) If $\mu = 100$, determine the output signal-to-quantizing noise ratio for a 5.0-V (full-scale) sinusoid.
 - (b) If $\mu = 100$, determine the output signal-to-quantizing noise ratio for a 0.050-V ($\frac{1}{100}$ -th of full scale) sinusoid.
 - (c) Repeat parts (a) and (b) for $\mu = 250$.

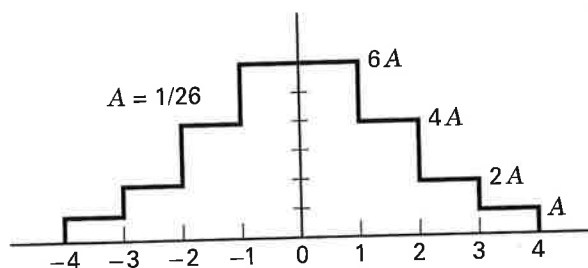


Figure P13.1

- 13.7. A compact disc (CD) recording system samples each of two stereo signals with a 16-bit A-to-D converter at 44.1 kilosamples/s.
- Determine the output signal-to-noise ratio for a full-scale sinusoid.
 - If the recorded music is designed to have a crest factor (peak-to-rms ratio) of 20, determine the average output signal-to-quantizing noise ratio.
 - The bit stream of digitized data is augmented by the addition of error-correcting bits, substitution bits to aid the clock extraction by a phase-locked loop (PLL), and display and control bit fields. These additional bits represent 100% overhead; that is, 2 bits are stored for each bit generated by the ADC. Determine the output bit rate of the CD recorder system.
 - The CD can record an hour's worth of music. Determine the number of bits recorded on a CD.
 - For a comparison, a good collegiate dictionary may contain 1500 pages, 2 columns/page, 100 lines/column, 7 words/line, 6 letters/word, and 6 bits/letter. Determine the number of bits required to describe the dictionary and estimate the number of comparable books that can be stored on a CD.
- 13.8. A 1-bit quantizer is being designed to sample an input sinusoid of amplitude A with uniformly distributed phase. Determine the amplitude x_0 , the output level of the 1-bit quantizer, which minimizes the mean-square quantization error.
- 13.9. A one-step linear predictive filter is to be used to sample a constant-amplitude sinusoid. The ratio of sample frequency to sinusoid frequency is 10.0. Determine the prediction coefficient of the filter. Determine the ratio of output power to input power for the one-tap predictor.
- 13.10. A two-tap linear predictor filter is being designed to operate in a DPCM system. The predictor is of the form $\hat{x}(n) = a_1x(n-1) + a_2x(n-2)$
- Determine the values a_1^{opt} and a_2^{opt} which minimize the mean-square prediction error.
 - Determine the expression for the mean-square prediction error.
 - Determine the prediction error power if the correlation coefficient of the input signal is of the form
- $$c(n) = \begin{cases} 1 - \frac{|n|}{4} & \text{for } n = -4, -3, -2, -1, 0, 1, 2, 3, 4 \\ 0 & \text{otherwise} \end{cases}$$
- Determine the prediction error power if the correlation coefficient of the input signal is of the form $C(n) = \cos \theta_0 n$
- 13.11. A single loop sigma-delta modulator is designed to operate at 20 times the Nyquist rate for a signal with a 10-kHz bandwidth. The converter is a 1-bit ADC.
- Determine the maximum SNR for a 8.0-kHz input signal.
 - Determine the maximum SNR for the same signal if the modulator is operated at 50 times the Nyquist rate.
 - Determine the maximum SNR for the same signal if the modulator is replaced with a 2-zero modulator operating at 20 times the Nyquist rate.
- 13.12. Design a binary Huffman code for a discrete source of three independent symbols A , B , and C with probabilities 0.9, 0.08, and 0.02, respectively. Determine the average code length for the code.

- 13.13.** Design a binary first-order extension code (two symbols at a time) for the discrete source described in Problem 13.12. Determine the average code length per symbol for this code.
- 13.14.** An input alphabet (a keyboard on a word processor) consists of 100 characters.
- (a) If the keystrokes are encoded by a fixed-length code, determine the required number of bits for the encoding.
 - (b) We make the simplifying assumption that 10 of the keystrokes are equally likely and that each occurs with probability 0.05. We also assume that the remaining 90 keystrokes are equally likely. Determine the average number of bits required to encode this alphabet using a variable-length Huffman code.
- 13.15.** Use the CCITT-modified Huffman facsimile code to encode the following single-line sequence of 2047 black-and-white pixels. Determine the ratio of coded bits to input bits.

1W 1B 2W 2B 4W 4B 8W 8B 16W 16B 32W 32B

64W 64B 128W 128B 256W 256B 512W 512B 1W

- 13.16.** JPEG quantizes the spectral terms formed by the DCT of the even extension of its processed data. To demonstrate the relative losses in a DCT and an FFT, form the even extension and the replicate extension of the series {10 12 14 16 18 20 22 24} to obtain {10 12 14 16 18 20 22 24 10 12 14 16 18 20 22 24} and {10 12 14 16 18 20 22 24 22 20 18 16 14 12 10} respectively. FFT the two time series and compare the relative sizes of the spectral components (other than the dc term). Now edit the spectra, setting to zero all but 5 spectral bins: In the even extension we keep bins [1 2 3 15 16] while in the periodic extension we keep bins [1 3 5 13 15]. Take the inverse FFT of each and compare the relative size of the reconstruction error for the two transforms.
- 13.17.** JPEG uses a zigzag scan pattern to address the DCT spectral terms delivered by its quantizer. An alternate scan pattern would be a raster scan, scanning successive rows commonly done in an image scan. Compare the coding efficiency of the zigzag scan to the raster scan when the non-zero spectral terms are $S(0, 0) = 11001100$, $S(1, 0) = 10101$, and $S(0, 1) = 110001$. Use the modified Huffman code of Table 13.2 to identify run lengths of zeros and assume the following table defines the bit assignment per spectral bin

8	6	5	4	3	2	2	2
6	5	4	3	2	2	1	1
5	4	3	2	2	1	1	1
4	3	2	2	1	1	1	1
3	2	2	1	1	1	1	1
2	2	1	1	1	1	1	1
2	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1

- 13.18.** The DCT transforms a block of 8×8 pixels containing 8-bit words into a block of 8×8 spectral samples containing the number of bits indicated in the quantization table of problem 13.17. Assuming there is no run length of zeros, so that every bit is present in the output of the DCT, compute the compression ratio (input/output bits) attributed to the DCT. Then compute the compression ratio assuming that the number of significant DCT coefficients is limited to the upper triangle of the quantization

table consisting of one 8-bit word, two 6-bit words, and three 5-bit words with the remaining bits carried by the run length code for 95 zeros.

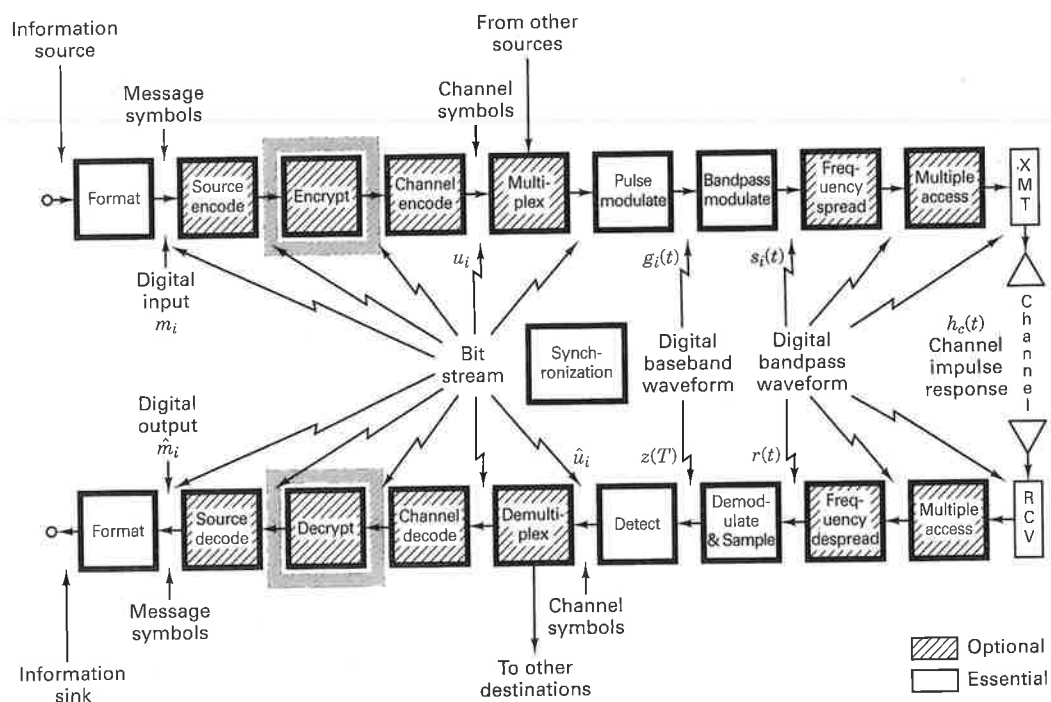
QUESTIONS

- 13.1. Prior to transmission or storage, why are signals subjected to *source coding* operations? (See Sections 13.1, and 13.7.)
- 13.2. What properties of a *continuous signal* permit the signal to be represented with a reduced number of bits per sample? (See Sections 13.1, 13.3, and 13.7.)
- 13.3. What properties of a *discrete signal* permit the signal to be represented with a reduced number of bits per symbol? (See Sections 13.1, and 13.7.)
- 13.4. Most quantizers are *uniform* in step size. Applications exist for which the quantizers have *non-uniform* step sizes. These are sometimes called *companded* quantizers. Why would we want a quantizer with unequal step sizes? (See Section 13.2.5.)
- 13.5. An analog-to-digital converter (ADC) represents the levels of a sampled data signal with a number of bits per sample selected to satisfy a required fidelity requirement. Most ADCs are *memoryless*, which means each quantization (conversion) is performed independently of other conversions. How can memory be used to reduce the number of bits per sample? (See Section 13.3.)
- 13.6. Source coding reduces *redundancy* and discards *irrelevant* content? What is the difference between redundancy and irrelevancy? (See Section 13.7.)
- 13.7. An often-heard saying is that "A picture is worth a thousand words." Is a picture really worth a thousand words? (See Section 13.8.2.)

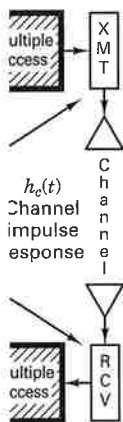
EXERCISES

Using the Companion CD, run the exercises associated with Chapter 13.

Encryption and Decryption



14.1



 Optional
 Essential

14.1 MODELS, GOALS, AND EARLY CIPHER SYSTEMS

14.1.1 A Model of the Encryption and Decryption Process

The desire to communicate privately is a human trait that dates back to earliest times. Hence the history of secret communications is rich with unique inventions and colorful anecdotes [1]. The study of ways to disguise messages so as to avert unauthorized interception is called *cryptography*. The terms *encipher* and *encrypt* refer to the message transformation performed at the transmitter, and the terms *decipher* and *decrypt* refer to the inverse transformation performed at the receiver. The two primary reasons for using cryptosystems in communications are (1) *privacy*, to prevent unauthorized persons from extracting information from the channel (eavesdropping); and (2) *authentication*, to prevent unauthorized persons from injecting information into the channel (spoofing). Sometimes, as in the case of electronic funds transfer or contract negotiations, it is important to provide the electronic equivalent of a *written signature* in order to avoid or settle any dispute between the sender and receiver as to what message, if any, was sent.

Figure 14.1 illustrates a model of a cryptographic channel. A message, or plaintext, M , is encrypted by the use of an invertible transformation, E_K , that produces a ciphertext, $C = E_K(M)$. The ciphertext is transmitted over an insecure or *public channel*. When an authorized receiver obtains C , he decrypts it with the inverse transformation, $D_K = E_K^{-1}$, to obtain the original plaintext message, as follows:

$$D_K(C) = E_K^{-1}[E_K(M)] = M \quad (14.1)$$

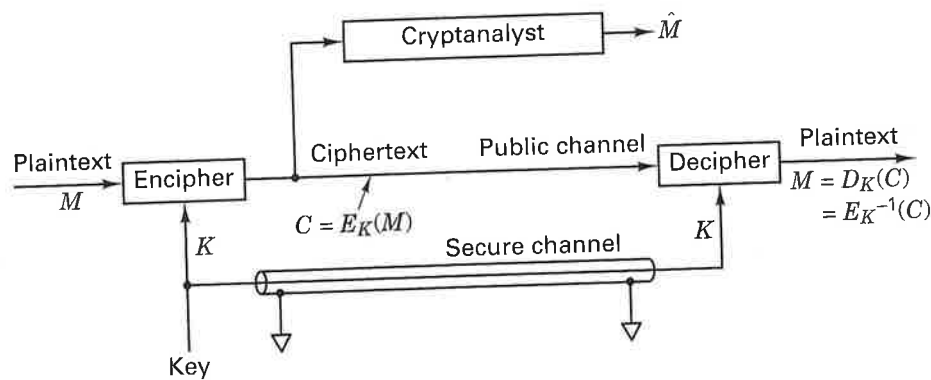


Figure 14.1 Model of a cryptographic channel.

The parameter K refers to a set of symbols or characters called a *key*, which dictates a specific encryption transformation, E_K , from a family of cryptographic transformations. Originally, the security of cryptosystems depended on the secrecy of the entire encryption process, but eventually systems were developed for which the general nature of the encryption transformation or algorithm could be publicly revealed, since the security of the system depended on the specific key. The key is supplied along with the plaintext message for encryption, and along with the ciphertext message for decryption. There is a close analogy here with a general-purpose computer and a computer program. The computer, like the cryptosystem, is capable of a large variety of transformations, from which the computer program, like the specific key, selects one. In most cryptosystems, anyone with access to the key can both encrypt and decrypt messages. The key is transmitted to the community of authorized users over a secure channel (as an example, a courier may be used to hand-carry the sensitive key information); the key usually remains unchanged for a considerable number of transmissions. The goal of the *cryptanalyst* (eavesdropper or adversary) is to produce an estimate of the plaintext, $\hat{M}$, by analyzing the ciphertext obtained from the public channel, without benefit of the key.

Encryption schemes fall into two generic categories: *block encryption*, and *data-stream* or simply *stream encryption*. With block encryption, the plaintext is segmented into blocks of fixed size; each block is encrypted independently from the others. For a given key, a particular plaintext block will therefore be carried into the same ciphertext block each time it appears (similar to block encoding). With data-stream encryption, similar to convolutional coding, there is no fixed block size. Each plaintext bit, m_i , is encrypted with the i th element, k_i , of a sequence of symbols (key stream) generated with the key. The encryption is *periodic* if the key stream repeats itself after p characters for some fixed p ; otherwise, it is nonperiodic.

In general, the properties desired in an encryption scheme are quite different from those desired in a channel coding scheme. For example, with encryption, plaintext data should never appear directly in the ciphertext, but with channel coding, codes are often in *systematic form* comprising unaltered message bits plus par-

ity bits (see Section 6.4.5). Consider another example of the differences between encryption and channel coding. With block encryption, a single bit error at the input of the decryptor might change the value of many of the output bits in the block. This effect, known as *error propagation*, is often a desirable cryptographic property since it makes it difficult for unauthorized users to succeed in spoofing a system. However, in the case of channel coding, we would like the system to correct as many errors as possible, so that the output is relatively unaffected by input errors.

14.1.2 System Goals

The major requirements for a cryptosystem can be stated as follows:

1. To provide an *easy* and *inexpensive* means of encryption and decryption to all authorized users in possession of the appropriate key
2. To ensure that the cryptanalyst's task of producing an estimate of the plaintext without benefit of the key is made *difficult* and *expensive*

Successful cryptosystems are classified as being either *unconditionally secure* or *computationally secure*. A system is said to be *unconditionally secure* when the amount of information available to the cryptanalyst is insufficient to determine the encryption and decryption transformations, no matter how much computing power the cryptanalyst has available. One such system, called a *one-time pad*, involves encrypting a message with a random key that is used one time only. The key is never reused; hence the cryptanalyst is denied information that might be useful against subsequent transmissions with the same key. Although such a system is *unconditionally secure* (see Section 14.2.1), it has limited use in a conventional communication system, since a new key would have to be distributed for each new message—a great logistical burden. The distribution of keys to the authorized users is a major problem in the operation of any cryptosystem, even when a key is used for an extended period of time. Although some systems can be proven to be *unconditionally secure*, currently there is no known way to demonstrate security for an arbitrary cryptosystem. Hence the specifications for most cryptosystems rely on the less formal designation of *computational security* for x number of years, which means that under circumstances favorable to the cryptanalyst (i.e., using state-of-the-art computers) the system security could be broken in a period of x years, but could not be broken in less than x years.

14.1.3 Classic Threats

The weakest classification of cryptanalytic threat on a system is called a *ciphertext-only attack*. In this attack the cryptanalyst might have *some* knowledge of the general system and the language used in the message, but the only significant data available to him is the encrypted transmission intercepted from the public channel.

A more serious threat to a system is called a *known plaintext attack*; it involves knowledge of the plaintext *and* knowledge of its ciphertext counterpart. The

$$\begin{aligned} \text{Plaintext} \\ l &= D_K(C) \\ &= E_K^{-1}(C) \end{aligned}$$

a key, which cryptographic on the secrecy ed for which ld be publicly ey. The key is ong with the ith a general- cryptosystem, uter program, i access to the o the commu- ourier may be y remains un- e cryptanalyst , $\hat{M}$, by analyz- of the key. rryption, and ae plaintext is endently from ore be carried ock encoding). re is no fixed nt, k_i , of a section is *periodic* otherwise, it is

quite different ith encryption, h channel cod- e bits plus par-

rigid structure of most business forms and programming languages often provides an opponent with much a priori knowledge of the details of the plaintext message. Armed with such knowledge and with a ciphertext message, the cryptanalyst can mount a known plaintext attack. In the diplomatic arena, if an encrypted message directs a foreign minister to make a particular public statement, and if he does so without paraphrasing the message, the cryptanalyst may be privy to both the ciphertext *and* its exact plaintext translation. While a known plaintext attack is not always possible, its occurrence is frequent enough that a system is not considered secure unless it is designed to be secure against the plaintext attack [2].

When the cryptanalyst is in the position of *selecting* the plaintext, the threat is termed a *chosen plaintext attack*. Such an attack was used by the United States to learn more about the Japanese cryptosystem during World War II. On May 20, 1942, Admiral Yamamoto, Commander-in-Chief of the Imperial Japanese Navy, issued an order spelling out the detailed tactics to be used in the assault of Midway island. This order was intercepted by the Allied listening posts. By this time, the Americans had learned enough of the Japanese code to decrypt most of the message. Still in doubt, however, were some important parts, such as the *place* of the assault. They suspected that the characters "AF" meant Midway island, but to be sure, Joseph Rochefort, head of the Combat Intelligence Unit, decided to use a chosen plaintext attack to trick the Japanese into providing concrete proof. He had the Midway garrison broadcast a distinctive plaintext message in which Midway reported that its fresh-water distillation plant had broken down. The American cryptanalysts needed to wait only two days before they intercepted a Japanese ciphertext message stating that AF was short of fresh water [1].

14.1.4 Classic Ciphers

One of the earliest examples of a monoalphabetic cipher was the *Caesar Cipher*, used by Julius Caesar during the Gallic wars. Each plaintext letter is replaced with a new letter obtained by an *alphabetic shift*. Figure 14.2a illustrates such an encryption transformation, consisting of three end-around shifts of the alphabet. When using this Caesar's alphabet, the message, "now is the time" is encrypted as follows:

Plaintext:	N	O	W	I	S	T	H	E	T	I	M	E
Ciphertext:	Q	R	Z	L	V	W	K	H	W	L	P	H

The decryption key is simply the number of alphabetic shifts; the code is changed by choosing a new key. Another classic cipher system, illustrated in Figure 14.2b, is called the *Polybius square*. Letters I and J are first combined and treated as a single character since the final choice can easily be decided from the context of the message. The resulting 25 character alphabet is arranged in a 5×5 array. Encryption of any character is accomplished by choosing the appropriate row-column (or column-row) number pair. An example of encryption with the use of the Polybius square follows:

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

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I	J
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

(b)

Figure 14.2 (a) Caesar's alphabet with a shift of 3. (b) Polybius square.

Plaintext: N O W I S T H E T I M E
 Chiphertext: 33 43 25 42 34 44 32 51 44 42 23 51

The code is changed by a rearrangement of the letters in the 5×5 array.

The *Trithemius progressive key*, shown in Figure 14.3, is an example of a *polyalphabetic cipher*. The row labeled shift 0 is identical to the usual arrangement of the alphabet. The letters in the next row are shifted one character to the left with an end-around shift for the leftmost position. Each successive row follows the same pattern of shifting the alphabet one character to the left as compared to the prior row. This continues until the alphabet has been depicted in all possible arrangements of end-around shifts. One method of using such an alphabet is to select the first cipher character from the shift 1 row, the second cipher character from the shift 2 row, and so on. An example of such encryption is

Plaintext: N O W I S T H E T I M E
 Chiphertext: O Q Z M X Z O M C S X Q

There are several interesting ways that the Trithemius progressive key can be used. One way, called the *Vigenere key method*, employs a keyword. The key dictates the row choices for encryption and decryption of each successive character in the message. For example, suppose that the word "TYPE" is selected as the key; then an example of the Vigenere encryption method is

Key: T Y P E T Y P E T Y P E
 Plaintext: N O W I S T H E T I M E
 Chiphertext: G M L M L R W I M G B I

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
Shift:	0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	R	U	V	W	X	Y	Z
	1	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	A
	2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	O	R	S	T	U	V	W	X	Y	Z	A	B
	3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
	4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
	5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
	6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
	7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
	8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
	9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
	10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
	11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
	12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
	13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
	14	O	P	Q	R	T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	15	P	Q	R	S	S	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
	17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
	18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
	19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
	20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
	21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
	23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
	24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
	25	Z	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

Figure 14.3 Trithemius progressive key.

where the first letter, T, of the key indicates that the row choice for encrypting the first plaintext character is the row starting with T (shift 19). The next row choice starts with Y (shift 24), and so on. A variation of this key method, called the *Vigenere auto (plain) key method*, starts with a single letter or word used as a *priming key*. The priming key dictates the starting row or rows for encrypting the first or first few plaintext characters, as in the preceding example. Next, the *plaintext characters* themselves are used as the key for choosing the rows for encryption. An example using the letter "F" as the priming key is

Key:	F	N	O	W	I	S	T	H	E	T	I	M
Plaintext:	N	O	W	I	S	T	H	E	T	I	M	E
Ciphertext:	S	B	K	E	A	L	A	L	X	B	U	Q

/ w x y z
 ✓ W X Y Z
 V X Y Z A
 X Y Z A B
 Y Z A B C
 Z A B C D
 A B C D E
 B C D E F
 C D E F G
 D E F G H
 E F G H I
 F G H I J
 G H I J K
 H I J K L
 I J K L M
 J K L M N
 K L M N O
 L M N O P
 M N O P Q
 N O P Q R
 O P Q R S
 P Q R S T
 Q R S T U
 R S T U V
 S T U V W
 T U V W X
 U V W X Y

encrypting the
 next row choice
 method, called the
 auto key method,
 is used as a *priming*
 key. The first or
plaintext character
 is used for encryption. An ex-

I M
 M E
 U Q

With the auto key method, it should be clear that feedback has been introduced to the encryption process. With this feedback, the choice of the ciphertext is dictated by the contents of the message.

A final variation of the Vigenere method, called the *Vigenere auto (cipher) key method*, is similar to the plain key method in that a priming key and feedback are used. The difference is that after encryption with the priming key, each successive key character is the sequence is obtained from the prior *ciphertext character* instead of from the plaintext character. An example should make this clear; as before, the letter "F" is used as the priming key:

Key:	F	S	G	C	K	C	V	C	G	Z	H	T
Plaintext:	N	O	W	I	S	T	H	E	T	I	M	E
Ciphertext:	S	G	C	K	C	V	C	G	Z	H	T	X

Although each key character can be found from its preceding ciphertext character, it is functionally dependent on *all* the preceding characters in the message plus the priming key. This has the effect of diffusing the statistical properties of the plaintext across the ciphertext, making statistical analysis very difficult for a cryptanalyst. One weakness of the cipher key example depicted here is that the ciphertext contains key characters which will be exposed on the public channel "for all to see." Variations of this method can be employed to prevent such overt exposure [3]. By today's standards Vigenere's encryption schemes are not very secure; his basic contribution was the discovery that nonrepeating key sequences could be generated by using the messages themselves or functions of the messages.

14.2 THE SECRECY OF A CIPHER SYSTEM

14.2.1 Perfect Secrecy

Consider a cipher system with a finite message space $\{M\} = M_0, M_1, \dots, M_{N-1}$ and a finite ciphertext space $\{C\} = C_0, C_1, \dots, C_{U-1}$. For any M_i , the a priori probability that M_i is transmitted is $P(M_i)$. Given that C_j is received, the a posteriori probability that M_i was transmitted is $P(M_i|C_j)$. A cipher system is said to have *perfect secrecy* if for every message M_i and every ciphertext C_j , the a posteriori probability is equal to the a priori probability:

$$P(M_i|C_j) = P(M_i) \tag{14.2}$$

Thus, for a system with perfect secrecy, a cryptanalyst who intercepts C_j obtains no further information to enable him or her to determine which message was transmitted. A necessary and sufficient condition for perfect secrecy is that for every M_i and C_j ,

$$P(C_j|M_i) = P(C_j) \tag{14.3}$$

The schematic in Figure 14.4 illustrates an example of perfect secrecy. In this example, $\{M\} = M_0, M_1, M_2, M_3$, $\{C\} = C_0, C_1, C_2, C_3$, $\{K\} = K_0, K_1, K_2, K_3$, $N = U = 4$,

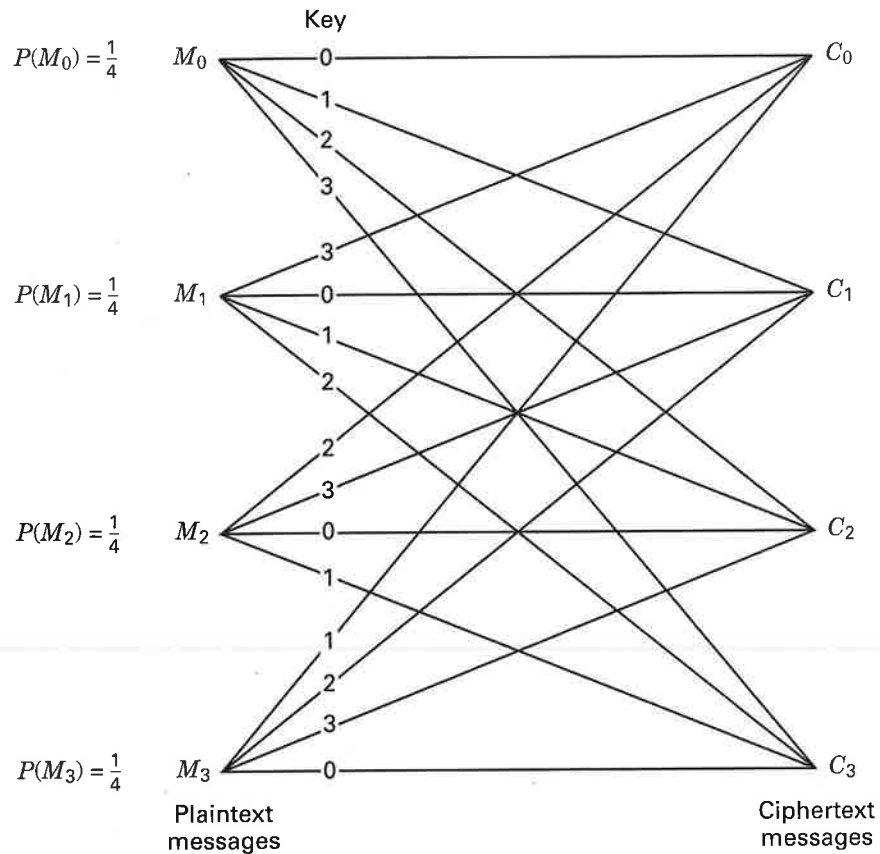


Figure 14.4 Example of perfect secrecy.

and $P(M_i) = P(C_j) = \frac{1}{4}$. The transformation from message to ciphertext is obtained by

$$C_s = T_{K_j}(M_i) \quad (14.4)$$

$$s = (i + j) \text{ modulo-} N$$

where T_{K_j} indicates a transformation under the key, K_j , and x modulo- y is defined as the remainder of dividing x by y . Thus $s = 0, 1, 2, 3$. A cryptanalyst intercepting one of the ciphertext messages $C_s = C_0, C_1, C_2$, or C_3 would have no way of determining which of the four keys was used, and therefore whether the correct message is M_0, M_1, M_2 , or M_3 . A cipher system in which the number of messages, the number of keys, and the number of ciphertext transformations are all equal is said to have perfect secrecy if and only if the following two conditions are met:

1. There is only one key transforming each message to each ciphertext.
2. All keys are equally likely.

If these conditions are not met, there would be some message M_i such that for a given C_j , there is no key that can decipher C_j into M_i , implying that $P(M_i|C_j) = 0$ for some i and j . The cryptanalyst could then eliminate certain plaintext messages from consideration, thereby simplifying the task. Perfect secrecy is a very desirable objective since it means that the cipher system is unconditionally secure. It should be apparent, however, that for systems which transmit a large number of messages, the amount of key that must be distributed for perfect secrecy can result in formidable management problems, making such systems impractical. Since in a system with perfect secrecy, the number of different keys is at least as great as the number of possible messages, if we allow messages of unlimited length, perfect secrecy requires an infinite amount of key.

Example 14.1 Breaking a Cipher System When the Key Space is Smaller Than the Message Space

Consider that the 29-character ciphertext

G R O B O K B O D R O R O B Y O C Y P I O C D O B I O K B

was produced by a Caesar cipher (see Section 14.1.4) such that each letter has been shifted by K positions, where $1 \leq K \leq 25$. Show how a cryptanalyst can break this code.

Solution

Because the number of possible keys (there are 25) is smaller than the number of possible 29-character meaningful messages (there are a myriad), perfect secrecy cannot be achieved. In the original polyalphabetic cipher of Figure 14.3, a plaintext character is replaced by a letter of increasingly higher rank as the row number (K) increases. Hence, in analyzing the ciphertext, we reverse the process by creating rows such that each ciphertext letter is replaced by letters of decreasing rank. The cipher is easily broken by trying all the keys, from 1 to 25, as shown in Figure 14.5, yielding only one key ($K = 10$) that produces the meaningful message: WHERE ARE THE HEROES OF YESTERYEAR (The spaces have been added.)

Example 14.2 Perfect Secrecy

We can modify the key space of Example 14.1 to create a cipher having perfect secrecy. In this new cipher system each character in the message is encrypted using a randomly selected key value. The key, K , is now given by the sequence $k_1, k_2, \dots, k_{29}$, where each k_i is a random integer in the range (1, 25) dictating the shift used for the i th character; thus there are a total of $(25)^{29}$ different key sequences. Then the 29-character ciphertext in Example 14.1 could correspond to any meaningful 29-character message. For example, the ciphertext could correspond to the plaintext (the spaces have been added)

ENGLISH AND FRENCH ARE SPOKEN HERE

derived by the key 2, 4, 8, 16, 6, 18, 20, ... Most of the 29-character possibilities can be ruled out because they are not meaningful messages (this much is known without the ciphertext). Perfect secrecy is achieved because interception of the ciphertext in this system reveals no additional information about the plaintext message.

Key	Text																												
0	G	R	O	B	O	K	B	O	D	R	O	R	O	B	Y	O	C	Y	P	I	O	C	D	O	B	I	O	K	B
1	F	Q	N	A	N	J	A	N	C	Q	N	Q	N	A	X	N	B	X	O	H	N	B	C	N	A	H	N	J	A
2	E	P	M	Z	M	I	Z	M	B	P	M	P	M	Z	W	M	A	W	N	G	M	A	B	M	Z	G	M	I	Z
3	D	O	L	Y	L	H	Y	L	A	O	L	O	L	Y	V	L	Z	V	M	F	L	Z	A	L	Y	F	L	H	Y
4	C	N	K	X	K	G	X	K	Z	N	K	N	K	X	U	K	Y	U	L	E	K	Y	Z	K	X	E	K	G	X
5	B	M	J	W	J	F	W	J	Y	M	J	M	J	W	T	J	X	T	K	D	J	X	Y	J	W	D	J	F	W
6	A	L	I	V	I	E	V	I	X	L	I	L	I	V	S	I	W	S	J	C	I	W	X	I	V	C	I	E	V
7	Z	K	H	U	H	D	U	H	W	K	H	K	H	U	R	H	V	R	I	B	H	V	W	H	U	B	H	D	U
8	Y	J	G	T	G	C	T	G	V	J	G	J	G	T	Q	G	U	Q	H	A	G	U	V	G	T	A	G	C	T
9	X	I	F	S	F	B	S	F	U	I	F	I	F	S	P	F	T	P	G	Z	F	T	U	F	S	Z	F	B	S
10	W	H	E	R	E	A	R	E	T	H	E	H	E	R	O	E	S	O	F	Y	E	S	T	E	R	Y	E	A	R
11	V	G	D	Q	D	Z	Q	D	S	G	D	G	D	Q	N	D	R	N	E	X	D	R	S	D	Q	X	D	Z	Q
12	U	F	C	P	C	Y	P	C	R	F	C	F	C	P	M	C	Q	M	D	W	C	Q	R	C	P	W	C	Y	P
13	T	E	B	O	B	X	O	B	Q	E	B	E	B	O	L	B	P	L	C	V	B	P	Q	B	O	V	B	X	O
14	S	D	A	N	A	W	N	A	P	D	A	D	A	N	K	A	O	K	B	U	A	O	P	A	N	U	A	W	N
15	R	C	Z	M	Z	V	M	Z	O	C	Z	C	Z	M	J	Z	N	J	A	T	Z	N	O	Z	M	T	Z	V	M
16	Q	B	Y	L	Y	U	L	Y	N	B	Y	B	Y	L	I	Y	M	I	Z	S	Y	M	N	Y	L	S	Y	U	L
17	P	A	X	K	X	T	K	X	M	A	X	A	X	K	H	X	L	H	Y	R	X	L	M	X	K	R	X	T	K
18	O	Z	W	J	W	S	J	W	L	Z	W	Z	W	J	G	W	K	G	X	Q	W	K	L	W	J	Q	W	S	J
19	N	Y	V	I	V	R	I	V	K	Y	V	Y	V	I	F	V	J	F	W	P	V	J	K	V	I	P	V	R	I
20	M	X	U	H	U	Q	H	U	J	X	U	X	U	H	E	U	I	E	V	O	U	I	J	U	H	O	U	Q	H
21	L	W	T	G	T	P	G	T	I	W	T	W	T	G	D	T	H	D	U	N	T	H	I	T	G	N	T	P	G
22	K	V	S	F	S	O	F	S	H	V	S	V	S	F	C	S	G	C	T	M	S	G	H	S	F	M	S	O	F
23	J	U	R	E	R	N	E	R	G	U	R	U	R	E	B	R	F	B	S	L	R	F	G	R	E	L	R	N	E
24	I	T	Q	D	Q	M	D	Q	F	T	Q	T	Q	D	A	Q	E	A	R	K	Q	E	F	Q	D	K	Q	M	D
25	H	S	P	C	P	L	C	P	E	S	P	S	P	C	Z	P	D	Z	Q	J	P	D	E	P	C	J	P	L	C

Figure 14.5 Example of breaking a cipher system when the key space is smaller than the message space.

14.2.2 Entropy and Equivocation

As discussed in Chapter 9, the amount of information in a message is related to the probability of occurrence of the message. Messages with probability of either 0 or 1 contain no information, since we can be very confident concerning our prediction of their occurrence. The more uncertainty there is in predicting the occurrence of a message, the greater is the information content. Hence when each of the messages in a set is equally likely, we can have *no* confidence in our ability to predict the occurrence of a particular message, and the uncertainty or information content of the message is maximum.

Entropy, $H(X)$, is defined as the average amount of information per message. It can be considered a measure of how much *choice* is involved in the selection of a message X . It is expressed by the following summation over all possible messages:

$$H(X) = - \sum_X P(X) \log_2 P(X) = \sum_X P(X) \log_2 \frac{1}{P(X)} \quad (14.5)$$

When the logarithm is taken to the base 2, as shown, $H(X)$ is the *expected number of bits* in an *optimally encoded* message X . This is not quite the measure that a cryptanalyst desires. He will have intercepted some ciphertext and will want to know how confidently he can predict a message (or key) given that this particular ciphertext was sent. *Equivocation*, defined as the conditional entropy of X given Y , is a more useful measure for the cryptanalyst in attempting to break the cipher and is given by

$$\begin{aligned} H(X|Y) &= - \sum_{X,Y} P(X,Y) \log_2 P(X|Y) \\ &= \sum_Y P(Y) \sum_X P(X|Y) \log_2 \frac{1}{P(X|Y)} \end{aligned} \quad (14.6)$$

Equivocation can be thought of as the uncertainty that message X was sent, having received Y . The cryptanalyst would like $H(X|Y)$ to approach zero as the amount of intercepted ciphertext, Y , increases.

Example 14.3 Entropy and Equivocation

Consider a sample message set consisting of eight equally likely messages $\{X\} = X_1, X_2, \dots, X_8$.

- Find the entropy associated with a message from the set $\{X\}$.
- Given another equally likely message set $\{Y\} = Y_1, Y_2$. Consider that the occurrence of each message Y narrows the possible choices of X in the following way:

If Y_1 is present: only X_1, X_2, X_3 , or X_4 is possible

If Y_2 is present: only X_5, X_6, X_7 , or X_8 is possible

Find the equivocation of message X conditioned on message Y .

Solution

- $P(X) = \frac{1}{8}$
 $H(X) = 8[\frac{1}{8} \log_2 8] = 3$ bits/message
- $P(Y) = \frac{1}{2}$. For each Y , $P(X|Y) = \frac{1}{4}$ for four of the X 's and $P(X|Y) = 0$ for the remaining four X 's. Using Equation (14.6), we obtain

$$H(X|Y) = 2[(\frac{1}{2})4(\frac{1}{4} \log_2 4)] = 2 \text{ bits/message}$$

We see that knowledge of Y has reduced the uncertainty of X from 3 bits/message to 2 bits/message.

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

space is

e is related to the
 ty of either 0 or 1
 ng our prediction
 e occurrence of a
 h of the messages
 to predict the oc-
 cion content of the

14.2.3 Rate of a Language and Redundancy

The *true rate* of a language is defined as the average number of *information bits* contained in each character and is expressed for messages of length N by

$$r = \frac{H(X)}{N} \quad (14.7)$$

where $H(X)$ is the message entropy, or the number of bits in the *optimally encoded* message. For large N , estimates of r for written English range between 1.0 and 1.5 bits/character [4]. The *absolute rate* or maximum entropy of a language is defined as the maximum number of information bits contained in each character assuming that all possible sequences of characters are equally likely. The absolute rate is given by

$$r' = \log_2 L \quad (14.8)$$

where L is the number of characters in the language. For the English alphabet $r' = \log_2 26 = 4.7$ bits/character. The true rate of English is, of course, much less than its absolute rate since, like most languages, English is highly redundant and structured.

The *redundancy* of a language is defined in terms of its true rate and absolute rate as

$$D = r' - r \quad (14.9)$$

For the English language with $r' = 4.7$ bits/character and $r = 1.5$ bits/character, $D = 3.2$, and the ratio $D/r' = 0.68$ is a measure of the redundancy in the language.

14.2.4 Unicity Distance and Ideal Secrecy

We stated earlier that perfect secrecy requires an infinite amount of key if we allow messages of unlimited length. With a finite key size, the equivocation of the key $H(K|C)$ generally approaches zero, implying that the key can be uniquely determined and the cipher system can be broken. The *unicity distance* is defined as the smallest amount of ciphertext, N , such that the key equivocation $H(K|C)$ is close to zero. Therefore, the unicity distance is the amount of ciphertext needed to uniquely determine the key and thus break the cipher system. Shannon [5] described an *ideal secrecy* system as one in which $H(K|C)$ does not approach zero as the amount of ciphertext approaches infinity; that is, no matter how much ciphertext is intercepted, the key cannot be determined. The term "ideal secrecy" describes a system that does not achieve perfect secrecy but is nonetheless unbreakable (unconditionally secure) because it does not reveal enough information to determine the key.

Most cipher systems are too complex to determine the probabilities required to derive the unicity distance. However, it is sometimes possible to approximate unicity distance, as shown by Shannon [5] and Hellman [6]. Following Hellman, assume that each plaintext and ciphertext message comes from a finite alphabet of L symbols.

Thus there are $2^{r'N}$ possible messages of length, N , where r' is the absolute rate of the language. We can consider the total message space partitioned into two classes, meaningful messages, M_1 , and meaningless messages M_2 . We then have

$$\text{number of meaningful messages} = 2^{rN} \quad (14.10)$$

$$\text{number of meaningless messages} = 2^{r'N} - 2^{rN} \quad (14.11)$$

where r is the true rate of the language, and where the a priori probabilities of the message classes are

$$P(M_1) = \frac{1}{2^{rN}} = 2^{-rN} \quad M_1 \text{ meaningful} \quad (14.12)$$

$$P(M_2) = 0 \quad M_2 \text{ meaningless} \quad (14.13)$$

Let us assume that there are $2^{H(K)}$ possible keys (size of the key alphabet), where $H(K)$ is the entropy of the key (number of bits in the key). Assume that all keys are equally likely; that is,

$$P(K) = \frac{1}{2^{H(K)}} = 2^{-H(K)} \quad (14.14)$$

The derivation of the unicity distance is based on a *random cipher* model, which states that for each key K and ciphertext C , the decryption operation $D_K(C)$ yields an independent random variable distributed over all the possible $2^{r'N}$ messages (both meaningful and meaningless). Therefore, for a given K and C , the $D_K(C)$ operation can produce any one of the plaintext messages with equal probability.

Given an encryption described by $C_i = E_{K_i}(M_i)$, a *false solution* F arises whenever encryption under another key K_j could also produce C_i either from the message M_i or from some other message M_j ; that is,

$$C_i = E_{K_i}(M_i) = E_{K_j}(M_i) = E_{K_j}(M_j) \quad (14.15)$$

A cryptanalyst intercepting C_i would not be able to pick the correct key and hence could not break the cipher system. We are not concerned with the decryption operations that produce *meaningless* messages because these are easily rejected.

For every correct solution to a particular ciphertext there are $2^{H(K)} - 1$ incorrect keys, each of which has the same probability $P(F)$ of yielding a false solution. Because each meaningful plaintext message is assumed equally likely, the probability of a false solution, is the same as the probability of getting a meaningful message, namely,

$$P(F) = \frac{2^{rN}}{2^{r'N}} = 2^{(r-r')N} = 2^{-DN} \quad (14.16)$$

where $D = r' - r$ is the redundancy of the language. The expected number of false solutions $\bar{F}$ is then

$$\begin{aligned} \bar{F} &= [2^{H(K)} - 1]P(F) = [2^{H(K)} - 1]2^{-DN} \\ &\approx 2^{H(K)-DN} \end{aligned} \quad (14.17)$$

Because of the rapid decrease of $\bar{F}$ with increasing N ,

$$\log_2 \bar{F} = H(K) - DN = 0 \quad (14.18)$$

is defined as the point where the number of false solutions is sufficiently small so that the cipher can be broken. The resulting unicity distance is therefore

$$N = \frac{H(K)}{D} \quad (14.19)$$

We can see from Equation (14.17) that if $H(K)$ is much larger than DN , there will be a large number of meaningful decryptions, and thus a small likelihood of a cryptanalyst distinguishing which meaningful message is the correct message. In a loose sense, DN represents the number of equations available for solving for the key, and $H(K)$ the number of unknowns. When the number of equations is smaller than the number of unknown key bits, a unique solution is not possible and the system is said to be unbreakable. When the number of equations is larger than the number of unknowns, a unique solution is possible and the system can no longer be characterized as unbreakable (although it may still be computationally secure).

It is the predominance of meaningless decryptions that enables cryptograms to be broken. Equation (14.19) indicates the value of using *data compression* techniques prior to encryption. Data compression removes redundancy, thereby increasing the unicity distance. Perfect data compression would result in $D = 0$ and $N = \infty$ for any key size.

Example 14.4 Unicity Distance

Calculate the unicity distance for a written English encryption system, where the key is given by the sequence $k_1, k_2, \dots, k_{29}$, where each k_i is a random integer in the range (1, 25) dictating the shift number (Figure 14.3) for the i th character. Assume that each of the possible key sequences is equally likely.

Solution

There are $(25)^{29}$ possible key sequences, each of which is equally likely. Therefore, using Equations (14.5), (14.8), and (14.19) we have

$$\text{Key entropy: } H(K) = \log_2 (25)^{29} = 135 \text{ bits}$$

$$\text{Absolute rate for English: } r' = \log_2 26 = 4.7 \text{ bits/character}$$

$$\text{Assumed true rate for English: } r = 1.5 \text{ bits/character}$$

$$\text{Redundancy: } D = r' - r = 3.2 \text{ bits/character}$$

$$N = \frac{H(K)}{D} = \frac{135}{3.2} \approx 43 \text{ characters}$$

In Example 14.2, perfect secrecy was illustrated using the same type of key sequence described here, with a 29-character message. In this example we see that if the available ciphertext is 43 characters long (which implies that some portion of the key sequence must be used twice), a unique solution may be possible. However, there is

no indication as to the computational difficulty in finding the solution. Even though we have estimated the theoretical amount of ciphertext required to break the cipher, it might be computationally infeasible to accomplish this.

14.3 PRACTICAL SECURITY

For ciphertext sequences greater than the unicity distance any system can be solved, in principle, merely by trying each possible key until the unique solution is obtained. This is completely impractical, however, except when the key is extremely small. For example, for a key configured as a permutation of the alphabet, there are $26! \approx 4 \times 10^{26}$ possibilities (considered small in the cryptographic context). In an exhaustive search, one might expect to reach the right key at about halfway through the search. If we assume that each trial requires a computation time of $1 \mu s$, the total search time exceeds 10^{12} years. Hence techniques other than a brute-force search (e.g., statistical analysis) must be employed if a cryptanalyst is to have any hope of success.

14.3.1 Confusion and Diffusion

A statistical analysis using the frequency of occurrence of individual characters and character combinations can be used to solve many cipher systems. Shannon [5] suggested two encryption concepts for frustrating the statistical endeavors of the cryptanalyst. He termed these encryption transformations confusion and diffusion. *Confusion* involves substitutions that render the final relationship between the key and ciphertext as complex as possible. This makes it difficult to utilize a statistical analysis to narrow the search to a particular subset of the key variable space. *Confusion* ensures that the majority of the key is needed to decrypt even very short sequences of ciphertext. *Diffusion* involves transformations that smooth out the statistical differences between characters and between character combinations. An example of diffusion with a 26-letter alphabet is to transform a message sequence $M = M_0, M_1, \dots$ into a new message sequence $Y = Y_0, Y_1, \dots$ according to the relationship

$$Y_n = \sum_{i=0}^{s-1} M_{n+i} \text{ modulo-26} \quad (14.20)$$

where each character in the sequence is regarded as an integer modulo-26, s is some chosen integer, and $n = 0, 1, 2, \dots$. The new message, Y , will have the same redundancy as the original message, M , but the letter frequencies of Y will be more uniform than in M . The effect is that the cryptanalyst needs to intercept a longer sequence of ciphertext before any statistical analysis can be useful.

14.3.2 Substitution

Substitution encryption techniques, such as the Caesar cipher and the Trithemius progressive key cipher, are widely used in puzzles. Such simple substitution ciphers offer little encryption protection. For a substitution technique to fulfill Shannon's

concept of *confusion*, a more complex relationship is required. Figure 14.6 shows one example of providing greater substitution complexity through the use of a nonlinear transformation. In general, n input bits are first represented as one of 2^n different characters (binary-to-octal transformation in the example of Figure 14.6). The set of 2^n characters are then permuted so that each character is transposed to one of the others in the set. The character is then converted back to an n -bit output.

It can be easily shown that there are $(2^n)!$ different substitution or connection patterns possible. The cryptanalyst's task becomes computationally unfeasible as n gets large, say $n = 128$; then $2^n = 10^{38}$, and $(2^n)!$ is an astronomical number. We recognize that for $n = 128$, this substitution box (S-box) transformation is complex (confusion). However, although we can identify the S-box with $n = 128$ as ideal, its implementation is not feasible because it would require a unit with $2^n = 10^{38}$ wiring connections.

To verify that the S-box example in Figure 14.6 performs a *nonlinear transformation*, we need only use the superposition theorem stated below as a test. Let

$$\begin{aligned} C &= Ta + Tb \\ C' &= T(a + b) \end{aligned} \quad (14.21)$$

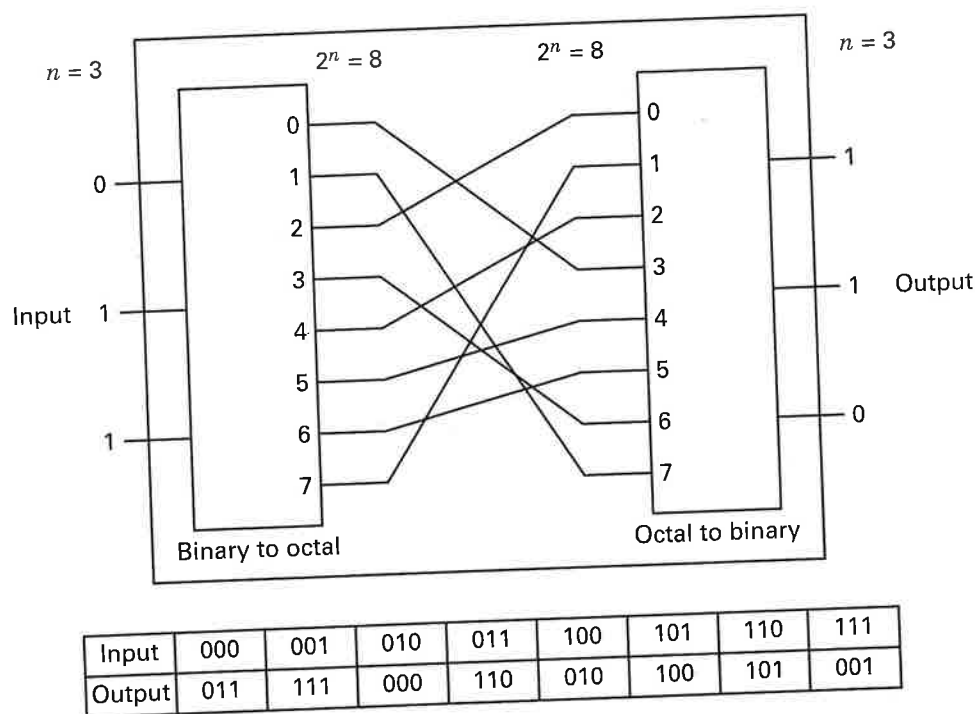


Figure 14.6 Substitution box.

where a and b are input terms, C and C' are output terms, and T is the transformation. Then

If T is linear: $C = C'$ for all inputs

If T is nonlinear: $C \neq C'$

Suppose that $a = 001$ and $b = 010$; then, using T as described in Figure 14.6, we obtain

$$C = T(001) \oplus T(010) = 111 \oplus 000 = 111$$

$$C' = T(001 \oplus 010) = T(011) = 110$$

where the symbol $\oplus$ represents modulo-2 addition. Since $C \neq C'$, the S -box is nonlinear.

14.3.3 Permutation

In permutation (transposition), the positions of the plaintext letters in the message are simply rearranged, rather than being substituted with other letters of the alphabet as in the classic ciphers. For example, the word THINK might appear, after permutation, as the ciphertext HKTNI. Figure 14.7 represents an example of binary data permutation (a linear operation). Here we see that the input data are simply rearranged or permuted (P-box). The technique has one major disadvantage when used alone; it is vulnerable to trick messages. A trick message is

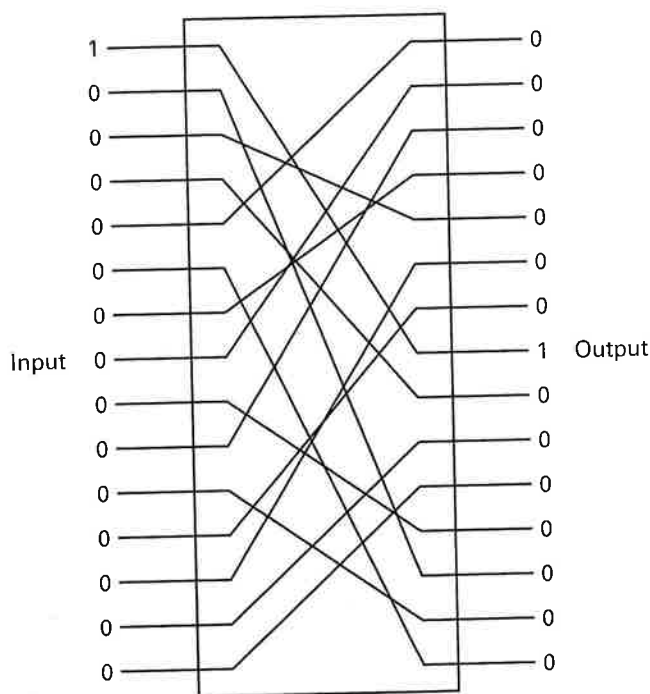


Figure 14.7 Permutation box.

illustrated in Figure 14.7. A single 1 at the input and all the rest 0 quickly reveals one of the internal connections. If the cryptanalyst can subject the system to a plaintext attack, he will transmit a sequence of such trick messages, moving the single 1 one position for each transmission. In this way, each of the connections from input to output is revealed. This is an example of why a system's security should not depend on its architecture.

14.3.4 Product Cipher System

For transformation involving reasonable numbers of n -message symbols, both of the foregoing cipher systems (the S -box and the P -box) are by themselves wanting. Shannon [5] suggested using a *product cipher* or a combination of S -box and P -box transformations, which together could yield a cipher system more powerful than either one alone. This approach of alternately applying substitution and permutation transformations has been used by IBM in the LUCIFER system [7, 8], and has become the basis for the national Data Encryption Standard (DES) [9]. Figure 14.8 illustrates such a combination of P -boxes and S -boxes. Decryption is accomplished by running the data backward, using the inverse of each S -box. The system as pictured in Figure 14.8 is difficult to implement since each S -box is different, a randomly generated key is not usable, and the system does not lend itself to repeated use of the same circuitry. To avoid these difficulties, the LUCIFER system [8] used two different types of S -boxes, S_1 and S_0 , which could be publicly revealed. Figure 14.9 illustrates such a system. The input data are transformed by the sequence of S -boxes and P -boxes under the dictates of a key. The 25-bit key in this example designates, with a binary one or zero, the choice (S_1 or S_0) of each of the 25 S -boxes

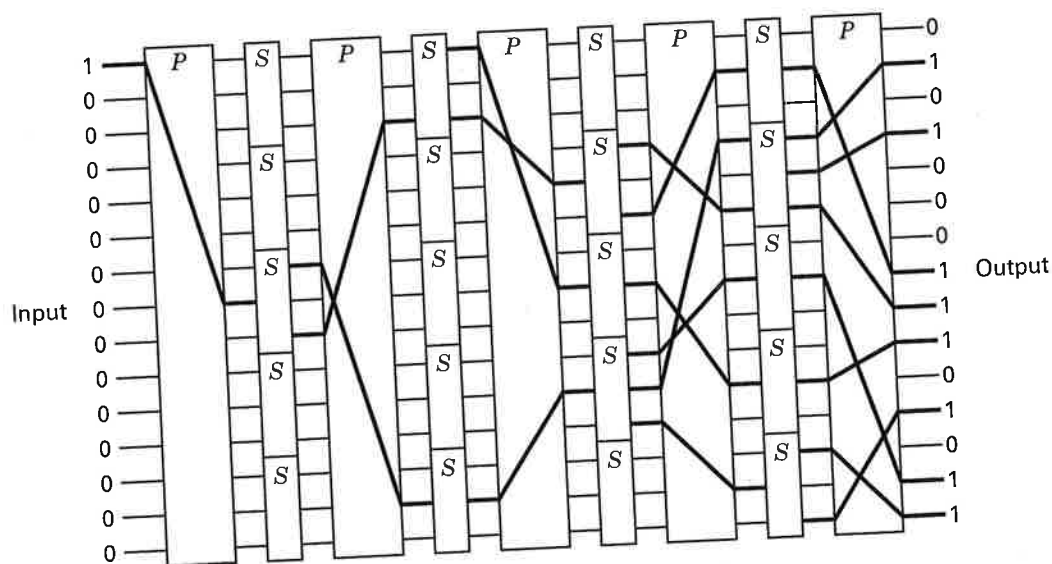
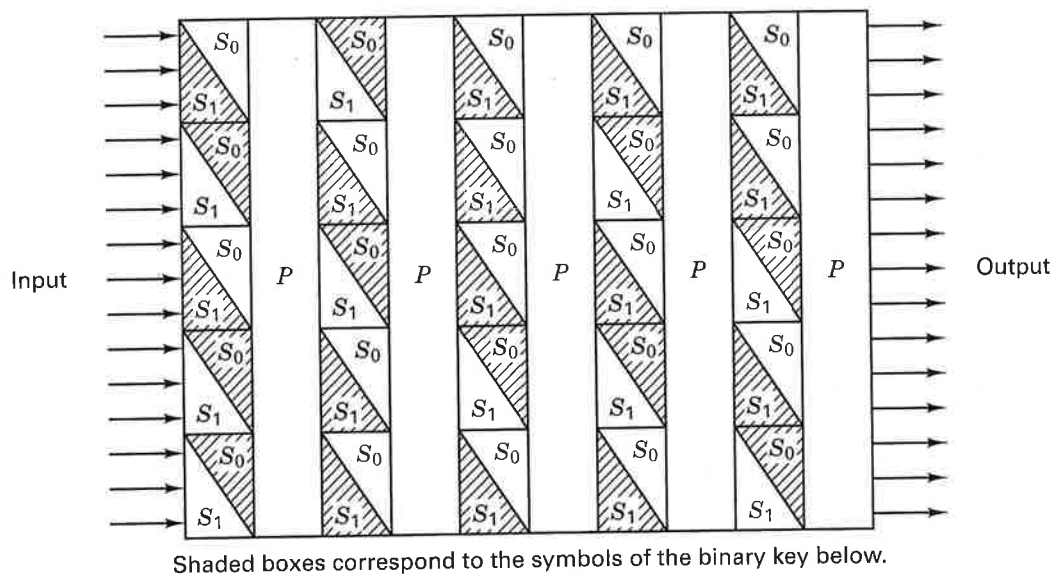


Figure 14.8 Product cipher system.



Example of binary key
 1 0 1 0 0 0 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 0 1 0

Figure 14.9 Individual keying capability.

in the block. The details of the encryption devices can be revealed since security of the system is provided by the key.

The iterated structure of the product cipher system in Figure 14.9 is typical of most present-day block ciphers. The messages are partitioned into successive blocks of n bits, each of which is encrypted with the same key. The n -bit block represents one of 2^n different characters, allowing for $(2^n)!$ different substitution patterns. Consequently, for a reasonable implementation, the substitution part of the encryption scheme is performed in parallel on small segments of the block. An example of this is seen in the next section.

14.3.5 The Data Encryption Standard

In 1977, the National Bureau of Standards adopted a modified Lucifer system as the national Data Encryption Standard (DES) [9]. From a system input-output point of view, DES can be regarded as a block encryption system with an alphabet size of 2^{64} symbols, as shown in Figure 14.10. An input block of 64 bits, regarded as a plaintext symbol in this alphabet, is replaced with a new ciphertext symbol. Figure 14.11 illustrates the system functions in block diagram form. The encryption algorithm starts with an initial permutation (IP) of the 64 plaintext bits, described in the IP-table (Table 14.1). The IP-table is read from left to right and from top to bottom, so that bits $x_1, x_2, \dots, x_{64}$ are permuted to $x_{58}, x_{50}, \dots, x_7$. After this initial permutation, the heart of the encryption algorithm consists of 16 iterations using

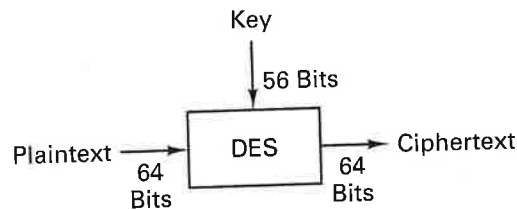


Figure 14.10 Data encryption standard (DES) viewed as a block encryption system.

the standard building block (SBB) shown in Figure 14.12. The standard building block uses 48 bits of key to transform the 64 input data bits into 64 output data bits, designated as 32 left-half bits and 32 right-half bits. The output of each building block becomes the input to the next building block. The input right-half 32 bits (R_{i-1}) are copied unchanged to become the output left-half 32 bits (L_i). The R_{i-1} bits are also *extended* and transformed into 48 bits with the E -table (Table 14.2), and then modulo-2 summed with the 48 bits of the key. As in the case of the IP-table, the E -table is read from left to right and from top to bottom. The table expands bits

$$R_{i-1} = x_1, x_2, \dots, x_{32}$$

into

$$(R_{i-1})_E = x_{32}, x_1, x_2, \dots, x_{32}, x_1 \quad (14.22)$$

Notice that the bits listed in the first and last columns of the E -table are those bit positions that are used twice to provide the 32 bit-to-48 bit expansion.

Next, $(R_{i-1})_E$ is modulo-2 summed with the i th key selection, explained later, and the result is segmented into eight 6-bit blocks

$$B_1, B_2, \dots, B_8$$

That is,

$$(R_{i-1})_E \oplus K_i = B_1, B_2, \dots, B_8 \quad (14.23)$$

Each of the eight 6-bit blocks, B_j , is then used as an input to an S -box function which returns a 4-bit block, $S_j(B_j)$. Thus the input 48 bits are transformed by the S -box to 32 bits. The S -box mapping function, S_j , is defined in Table 14.3. The transformation of $B_j = b_1, b_2, b_3, b_4, b_5, b_6$ is accomplished as follows. The integer corresponding to bits, b_1, b_6 selects a row in the table, and the integer corresponding to bits b_2, b_3, b_4, b_5 selects a column in the table. For example, if $b_1 = 110001$, then S_1 returns the value in row 3, column 8, which is the integer 5 and is represented by the bit sequence 0101. The resulting 32-bit block out of the S -box is then permuted using the P -table (Table 14.4). As in the case of the other tables, the P -table is read from left to right and from top to bottom, so that bits $x_1, x_2, \dots, x_{32}$ are permuted to $x_{16}, x_7, \dots, x_{25}$. The 32-bit output of the P -table is modulo-2 summed with the input left-half 32 bits (L_{i-1}), forming the output right-half 32 bits (R_i).

The algorithm of the standard building block can be represented by

Standar
building
block

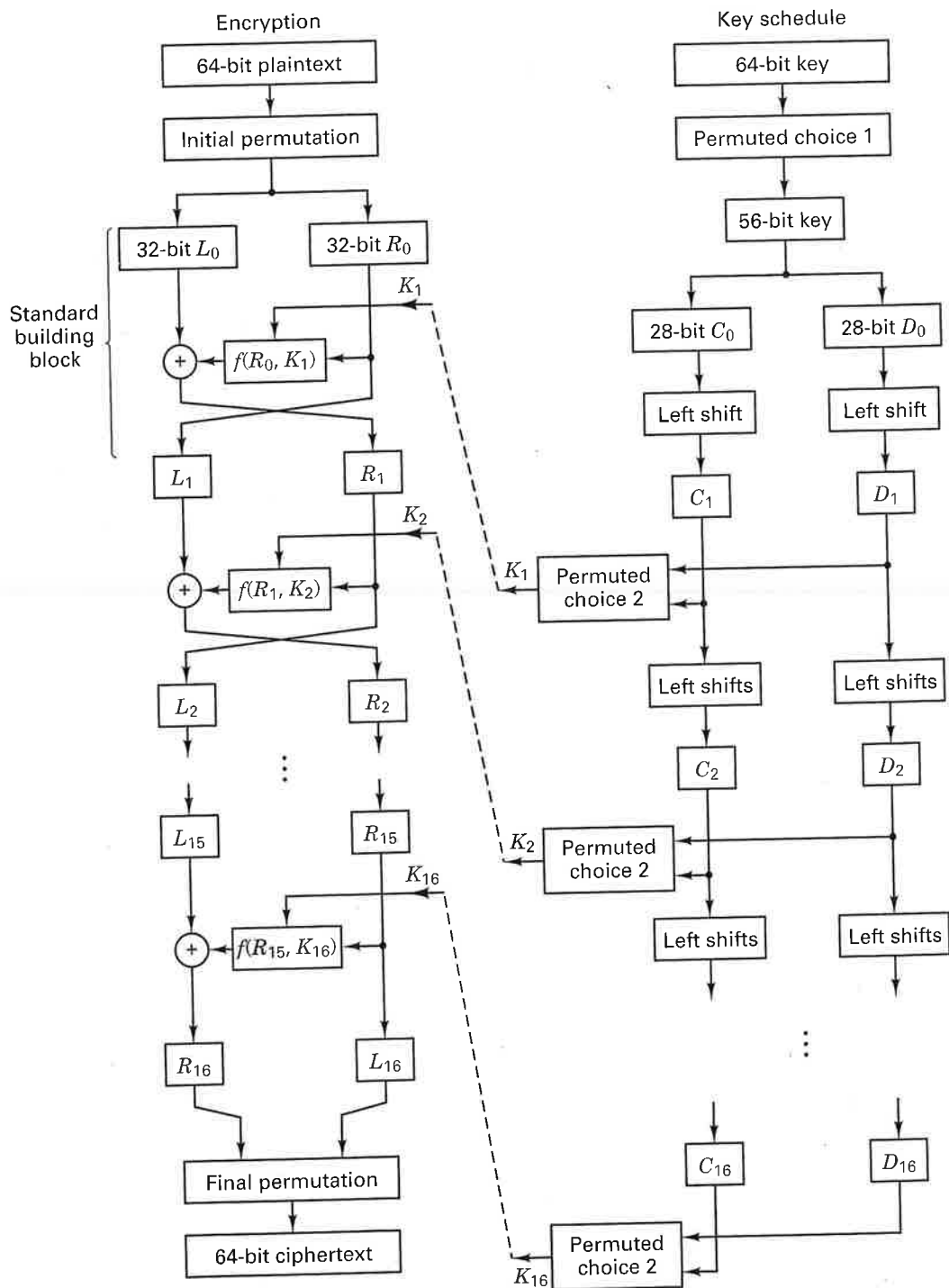


Figure 14.11 Data encryption standard.

TABLE 14.1 Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

$$L_i = R_{i-1} \quad (14.24)$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \quad (14.25)$$

where $f(R_{i-1}, K_i)$ denotes the functional relationship comprising the E -table, S -box, and P -table we have described. After 16 iterations of the SBB, the data are transposed according to the final inverse permutation (IP^{-1}) described in the IP^{-1} -table (Table 14.5), where the output bits are read from left to right and from top to bottom, as before.

To decrypt, the same algorithm is used but the key sequence that is used in the standard building block is taken in the reverse order. Note that the value of $f(R_{i-1}, K_i)$ which can also be expressed in terms of the output of the i th block as $f(L_i, K_i)$, makes the decryption process possible.

14.3.5.1 Key Selection

Key selection also proceeds in 16 iterations, as seen in the key schedule portion of Figure 14.11. The input key consists of a 64-bit block with 8 parity bits in positions 8, 16, ..., 64. The permuted choice 1 (PC-1) discards the parity bits and permutes the remaining 56 bits as shown in Table 14.6. The output of PC-1 is split into two halves, C and D , of 28 bits each. Key selection proceeds in 16 iterations in

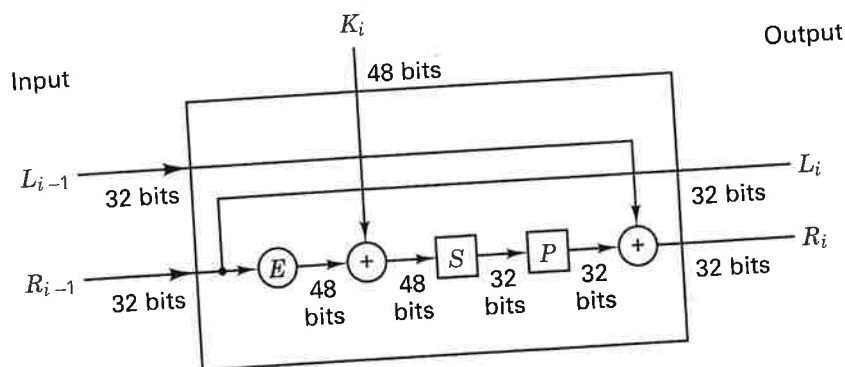


Figure 14.12 Standard building block (SBB).

TABLE 14.2 E-Table Bit Selection

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

TABLE 14.3 S-Box Selection Functions

Row	Column																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S_1
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S_2
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S_3
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S_4
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S_5
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S_6
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	4	3	2	12	9	5	15	0	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S_7
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S_8
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

TABLE 14.4 P-Table Permutation

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

order to provide a different set of 48 key bits to each SBB encryption iteration. The C and D blocks are successively shifted according to

$$C_i = \text{LS}_i(C_{i-1}) \quad \text{and} \quad D_i = \text{LS}_i(D_{i-1}) \quad (14.26)$$

where LS_i is a left circular shift by the number of positions shown in Table 14.7. The sequence C_i, D_i is then transposed according to the permuted choice 2 (PC-2) shown in Table 14.8. The result is the key sequence K_i , which is used in the i th iteration of the encryption algorithm.

The DES can be implemented as a block encryption system (see Figure 14.11), which is sometimes referred to as a *codebook* method. A major disadvantage of this method is that a given block of input plaintext will always result in the same output ciphertext (under the same key). Another encryption mode, called the *cipher feedback* mode, encrypts single bits rather than characters, resulting in a stream encryption system [3]. With the cipher feedback scheme (described later), the encryption of a segment of plaintext not only depends on the key and the current data, but also on some of the earlier data.

Since the late 1970s, two points of contention have been widely publicized about the DES [10]. The first concerns the key variable length. Some researchers felt that 56 bits are not adequate to preclude an exhaustive search. The second concerns the details of the internal structure of the S-boxes, which were never released by IBM. The National Security Agency (NSA), which had been involved in the testing of the DES algorithm, had requested that the information not be publicly discussed, because it was sensitive. The critics feared that NSA had been involved in design selections that would allow NSA to "tap into" any DES-encrypted messages [10]. DES is no longer a viable choice for strong encryption. The 56-bit key

TABLE 14.5 Final Permutation (IP^{-1})

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

TABLE 14.6 Key Permutation PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

can be found in a matter of days with relatively inexpensive computer tools [11]. (Some alternative algorithms are discussed in Section 14.6.)

14.4 STREAM ENCRYPTION

Earlier, we defined a *one-time pad* as an encryption system with a random key, used one time only, that exhibits unconditional security. One can conceptualize a stream encryption implementation of a one-time pad using a truly random key stream (the key sequence never repeats). Thus, perfect secrecy can be achieved for an infinite number of messages, since each message would be encrypted with a different portion of the random key stream. The development of stream encryption schemes represents an attempt to emulate the one-time pad. Great emphasis was placed on generating key streams that appeared to be random, yet could easily be implemented for decryption, because they could be generated by algorithms. Such stream encryption techniques use pseudorandom (PN) sequences, which derive their name from the fact that they appear random to the casual observer; binary

TABLE 14.7 Key Schedule of Left Shifts

Iteration, i	Number of left shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

32
31
30
29
28
27
26
25

TABLE 14.8 Key Permutation PC-2

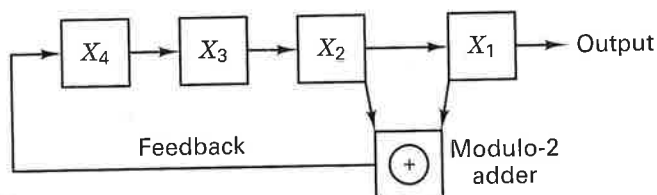
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

pseudorandom sequences have statistical properties similar to the random flipping of a fair coin. However, the sequences, of course, are deterministic (see Section 12.2). These techniques are popular because the encryption and decryption algorithms are readily implemented with feedback shift registers. At first glance it may appear that a PN key stream can provide the same security as the one-time pad, since the period of the sequence generated by a maximum-length linear shift register is $2^n - 1$ bits, where n is the number of stages in the register. If the PN sequence were implemented with a 50-stage register and a 1-MHz clock rate, the sequence would repeat every $2^{50} - 1$ microseconds, or every 35 years. In this era of large-scale integrated (LSI) circuits, it is just as easy to provide an implementation with 100 stages, in which case the sequence would repeat every 4×10^{16} years. Therefore, one might suppose that since the PN sequence does not repeat itself for such a long time, it would appear truly random and yield perfect secrecy. There is one important difference between the PN sequence and a truly random sequence used by a one-time pad. The PN sequence is generated by an algorithm; thus, knowing the algorithm, one knows the entire sequence. In Section 14.4.2 we will see that an encryption scheme that uses a linear feedback shift register in this way is very vulnerable to a *known plaintext attack*.

14.4.1 Example of Key Generation Using a Linear Feedback Shift Register

Stream encryption techniques generally employ shift registers for generating their PN key sequence. A shift register can be converted into a pseudorandom sequence generator by including a feedback loop that computes a new term for the first stage based on the previous n terms. The register is said to be linear if the numerical operation in the feedback path is linear. The PN generator example from Section 12.2 is repeated in Figure 14.13. For this example, it is convenient to number the stages as shown in Figure 14.13, where $n = 4$ and the outputs from stages 1 and 2 are modulo-2 added (linear operation) and fed back to stage 4. If the initial state of stages (x_4, x_3, x_2, x_1) is 1 0 0 0, the succession of states triggered by clock pulses would be 1 0 0 0, 0 1 0 0, 0 0 1 0, 1 0 0 1, 1 1 0 0, and so on. The output sequence is made up of the bits shifted out from the rightmost stage of the register, that is, 1 1 1 0 1 0 1 1 0 0 1 0 0 0, where the rightmost bit in this sequence is the earliest output and the leftmost bit is the most recent output. Given any linear feedback shift register of degree n , the output sequence is ultimately periodic.

Figure 14.13 Linear feedback shift register example.



14.4.2 Vulnerabilities of Linear Feedback Shift Registers

An encryption scheme that uses a linear feedback shift register (LFSR) to generate the key stream is very vulnerable to attack. A cryptanalyst needs only $2n$ bits of plaintext and its corresponding ciphertext to determine the feedback taps, the initial state of the register, and the entire sequence of the code. In general, $2n$ is very small compared with the period $2^n - 1$. Let us illustrate this vulnerability with the LFSR example illustrated in Figure 14.13. Imagine that a cryptanalyst who knows nothing about the internal connections of the LFSR manages to obtain $2n = 8$ bits of ciphertext and its plaintext equivalent:

Plaintext: 0 1 0 1 0 1 0 1
Ciphertext: 0 0 0 0 1 1 0 0

Where, the rightmost bit is the earliest received and the leftmost bit is the most recent that was received.

The cryptanalyst adds the two sequences together, modulo-2, to obtain the segment of the key stream, 0 1 0 1 1 0 0 1, illustrated in Figure 14.14. The key stream sequence shows the contents of the LFSR stages at various times. The rightmost border surrounding four of the key bits shows the contents of the shift register at time t_1 . As we successively slide the "moving" border one digit to the left, we see the shift register contents at times $t_2, t_3, t_4, \dots$. From the linear structure of the four-stage shift register, we can write

$$g_4x_4 + g_3x_3 + g_2x_2 + g_1x_1 = x_5 \quad (14.27)$$

where x_5 is the digit fed back to the input and g_i ($= 1$ or 0) defines the i th feedback connection. For this example, we can thus write the following four equations with four unknowns, by examining the contents of the shift register at the four times shown in Figure 14.14:

$$\begin{aligned} g_4(1) + g_3(0) + g_2(0) + g_1(1) &= 1 \\ g_4(1) + g_3(1) + g_2(0) + g_1(0) &= 0 \\ g_4(0) + g_3(1) + g_2(1) + g_1(0) &= 1 \\ g_4(1) + g_3(0) + g_2(1) + g_1(1) &= 0 \end{aligned} \quad (14.28)$$

The solution of Equations (14.28) is $g_1 = 1, g_2 = 1, g_3 = 0, g_4 = 0$, corresponding to the LFSR shown in Figure 14.13. The cryptanalyst has thus learned the connections of

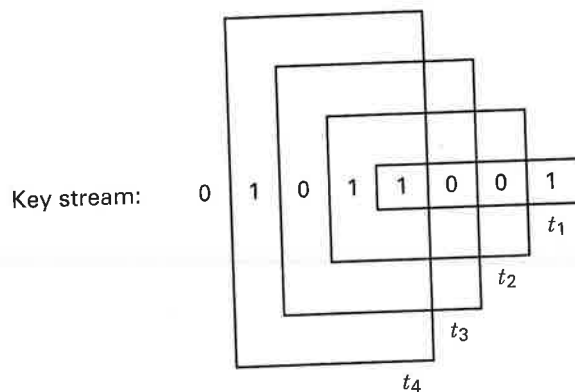
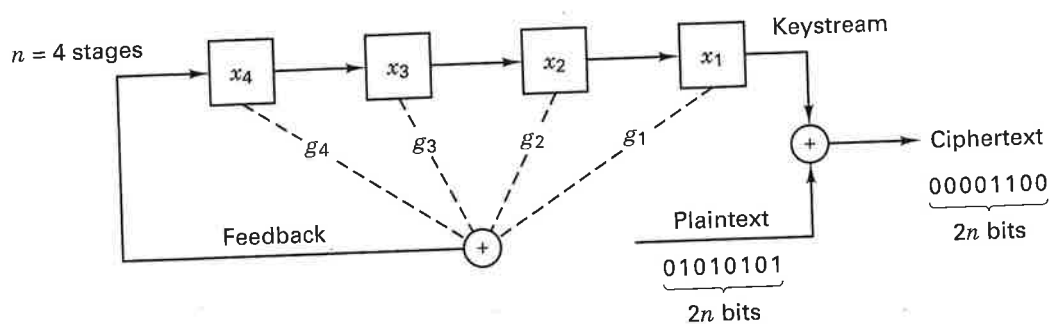


Figure 14.14 Example of vulnerability of a linear feedback shift register.

the LFSR, together with the starting state of the register at time t_1 . He can therefore know the sequence for all time [3]. To generalize this example for any n -stage LFSR, we rewrite Equation (14.27) as follows:

$$x_{n+1} = \sum_{i=1}^n g_i x_i \quad (14.29)$$

We can write Equation (14.29) as the matrix equation

$$\mathbf{x} = \mathbf{X}\mathbf{g} \quad (14.30)$$

where

$$\mathbf{x} = \begin{bmatrix} x_{n+1} \\ x_{n+2} \\ \vdots \\ x_{2n} \end{bmatrix} \quad \mathbf{g} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{bmatrix}$$

and

$$\mathbf{X} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ x_2 & x_3 & \cdots & x_{n+1} \\ \vdots & \vdots & & \vdots \\ x_n & x_{n+1} & \cdots & x_{2n-1} \end{bmatrix}$$

It can be shown [3] that the columns of $\mathbf{X}$ are linearly independent; thus $\mathbf{X}$ is nonsingular (its determinant is nonzero) and has an inverse. Hence,

$$\mathbf{g} = \mathbf{X}^{-1} \mathbf{x} \quad (14.31)$$

The matrix inversion requires at most on the order of n^3 operations and is thus easily accomplished by computer for any reasonable value of n . For example, if $n = 100$, $n^3 = 10^6$, and a computer with a $1\text{-}\mu\text{s}$ operation cycle would require 1 s for the inversion. The weakness of a LFSR is caused by the linearity of Equation (14.31). The use of *nonlinear feedback* in the shift register makes the cryptanalyst's task much more difficult, if not computationally intractable.

14.4.3 Synchronous and Self-Synchronous Stream Encryption Systems

We can categorize stream encryption systems as either *synchronous* or *self-synchronous*. In the former, the key stream is generated independently of the message, so that a lost character during transmission necessitates a resynchronization of the transmission and receiver key generators. A synchronous stream cipher is shown in Figure 14.15. The starting state of the key generator is initialized with a known input, I_0 . The ciphertext is obtained by the modulo addition of the i th key character, k_i , with the i th message character, m_i . Such synchronous ciphers are generally designed to utilize *confusion* (see Section 14.3.1) but not *diffusion*. That is, the encryption of a character is not diffused over some block length of message. For this reason, synchronous stream ciphers do not exhibit *error propagation*.

In a *self-synchronous* stream cipher, each key character is derived from a fixed number, n , of the preceding ciphertext characters, giving rise to the name *cipher feedback*. In such a system, if a ciphertext character is lost during

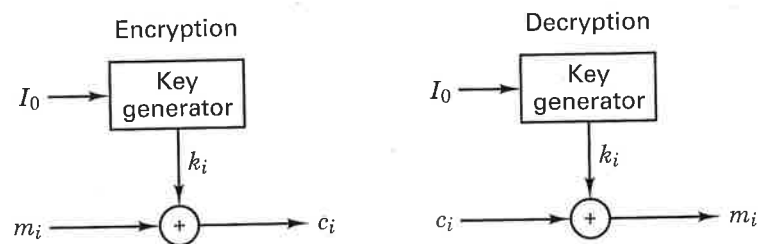


Figure 14.15 Synchronous stream cipher.

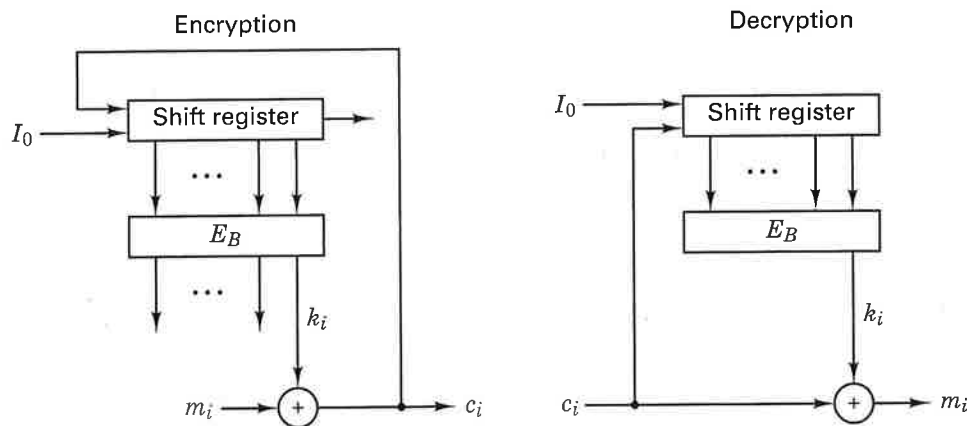


Figure 14.16 Cipher feedback mode.

Figure
tosyste

transmission, the error propagates forward for n characters, but the system resynchronizes itself after n correct ciphertext characters are received.

In Section 14.1.4 we looked at an example of cipher feedback in the Vigenere auto key cipher. We saw that the advantages of such a system are that (1) a nonrepeating key is generated, and (2) the statistics of the plaintext message are diffused throughout the ciphertext. However, the fact that the key was exposed in the ciphertext was a basic weakness. This problem can be eliminated by passing the ciphertext characters through a nonlinear block cipher to obtain the key characters. Figure 14.16 illustrates a shift register key generator operating in the cipher feedback mode. Each output ciphertext character, c_i (formed by the modulo addition of the message character, m_i , and the key character, k_i), is fed back to the input of the shift register. As before, initialization is provided by a known input, I_0 . At each iteration, the output of the shift register is used as input to a (nonlinear) block encryption algorithm E_B . The low-order output character from E_B becomes the next key character, k_{i+1} , to be used with the next message character, m_{i+1} . Since, after the first few iterations, the input to the algorithm depends only on the ciphertext, the system is self-synchronizing.

14.5 PUBLIC KEY CRYPTOSYSTEMS

The concept of public key cryptosystems was introduced in 1976 by Diffie and Hellman [12]. In conventional cryptosystems the encryption algorithm can be revealed since the security of the system depends on a safeguarded key. The same key is used for both encryption and decryption. Public key cryptosystems utilize *two different* keys, one for encryption and the other for decryption. In public key cryptosystems, not only the encryption algorithm but also the encryption key can be publicly revealed without compromising the security of the system. In fact, a public directory, much like a telephone directory, is envisioned, which contains the

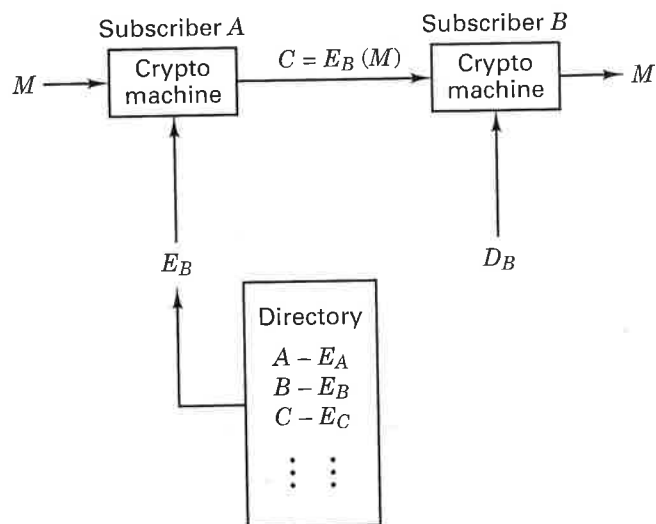
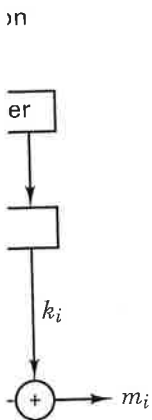


Figure 14.17 Public key cryptosystem.

encryption keys of all the subscribers. Only the decryption keys are kept secret. Figure 14.17 illustrates such a system. The important features of a public key cryptosystem are as follows:

1. The encryption algorithm E_K and the decryption algorithm D_K are invertible transformations on the plaintext M , or the ciphertext C , defined by the key K . That is, for each K and M , if $C = E_K(M)$, then $M = D_K(C) = D_K[E_K(M)]$.
2. For each K , E_K and D_K are easy to compute.
3. For each K , the computation of D_K from E_K is computationally intractable.

Such a system would enable secure communication between subscribers who have never met or communicated before. For example, as seen in Figure 14.17, subscriber A can send a message, M , to subscriber B by looking up B 's encryption key in the directory and applying the encryption algorithm, E_B , to obtain the ciphertext $C = E_B(M)$, which he transmits on the public channel. Subscriber B is the only party who can decrypt C by applying his decryption algorithm, D_B , to obtain $M = D_B(C)$.

14.5.1 Signature Authentication Using a Public Key Cryptosystem

Figure 14.18 illustrates the use of a public key cryptosystem for signature authentication. Subscriber A "signs" his message by first applying his decryption algorithm, D_A , to the message, yielding $S = D_A(M) = E_A^{-1}(M)$. Next, he uses the encryption algorithm, E_B , of subscriber B to encrypt S , yielding $C = E_B(S) = E_B[E_A^{-1}(M)]$, which he transmits on a public channel. When subscriber B receives C , he first decrypts it using his private decryption algorithm, D_B , yielding $D_B(C) = E_A^{-1}(M)$. Then he applies the encryption algorithm of subscriber A to produce $E_A[E_A^{-1}(M)] = M$.

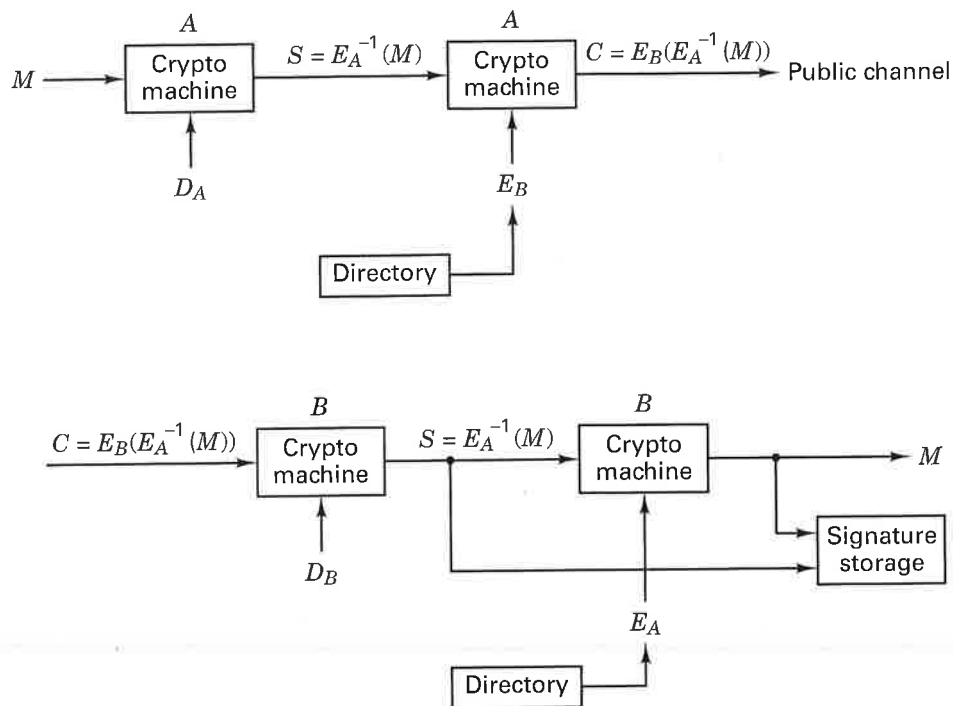


Figure 14.18 Signature authentication using a public key cryptosystem.

If the result is an intelligible message, it must have been initiated by subscriber A , since no one else could have known A 's secret decryption key to form $S = D_A(M)$. Notice that S is both message dependent and signer dependent, which means that while B can be sure that the received message indeed came from A , at the same time A can be sure that no one can attribute any false messages to him.

14.5.2 A Trapdoor One-Way Function

Public key cryptosystems are based on the concept of trapdoor one-way functions. Let us first define, a *one-way function* as an easily computed function whose inverse is computationally infeasible to find. For example, consider the function $y = x^5 + 12x^3 + 107x + 123$. It should be apparent that given x , y is easy to compute, but given y , x is relatively difficult to compute. A *trapdoor one-way function* is a one-way function whose inverse is easily computed if certain features, used to design the function, are known. Like a trapdoor, such functions are easy to go through in one direction. Without special information the reverse process takes an impossibly long time. We will apply the concept of a trapdoor in Section 14.5.5, when we discuss the Merkle–Hellman scheme.

14.5.3 The Rivest-Shamir-Adelman Scheme

In the Rivest-Shamir-Adelman (RSA) scheme messages are first represented as integers in the range $(0, n - 1)$. Each user chooses his own value of n and another pair of positive integers e and d , in a manner to be described below. The user places his encryption key, the number pair (n, e) , in the public directory. The decryption key consists of the number pair (n, d) , of which d is kept secret. Encryption of a message M and decryption of a ciphertext C are defined as follows:

$$\text{Encryption: } C = E(M) = (M)^e \text{ modulo-} n \quad (14.32)$$

$$\text{Decryption: } M = D(C) = (C)^d \text{ modulo-} n$$

They are each easy to compute and the results of each operation are integers in the range $(0, n - 1)$. In the RSA scheme, n is obtained by selecting two large prime numbers p and q and multiplying them together:

$$n = pq \quad (14.33)$$

Although n is made public, p and q are kept hidden, due to the great difficulty in factoring n . Then

$$\phi(n) = (p - 1)(q - 1) \quad (14.34)$$

called *Euler's totient function*, is formed. The parameter $\phi(n)$ has the interesting property [12] that for any integer X in the range $(0, n - 1)$ and any integer k ,

$$X = X^{k\phi(n)+1} \text{ modulo-} n \quad (14.35)$$

Therefore, while all other arithmetic is done modulo- n , arithmetic in the exponent is done modulo- $\phi(n)$. A large integer, d , is randomly chosen so that it is relatively prime to $\phi(n)$, which means that $\phi(n)$ and d must have no common divisors other than 1, expressed as

$$\text{gcd}[\phi(n), d] = 1 \quad (14.36)$$

where gcd means "greatest common divisor." Any prime number greater than the larger of (p, q) will suffice. Then the integer e , where $0 < e < \phi(n)$, is found from the relationship

$$ed \text{ modulo-}\phi(n) = 1 \quad (14.37)$$

which, from Equation (14.35), is tantamount to choosing e and d to satisfy

$$X = X^{ed} \text{ modulo-} n \quad (14.38)$$

Therefore,

$$E[D(X)] = D[E(X)] = X \quad (14.39)$$

and decryption works correctly. Given an encryption key (n, e) , one way that a cryptanalyst might attempt to break the cipher is to factor n into p and q , compute $\phi(n) = (p - 1)(q - 1)$, and compute d from Equation (14.37). This is all straightforward except for the factoring of n .

The RSA scheme is based on the fact that it is easy to generate two large prime numbers, p and q , and multiply them together, but it is very much more difficult to factor the result. The product can therefore be made public as part of the encryption key, without compromising the factors that would reveal the decryption key corresponding to the encryption key. By making each of the factors roughly 100 digits long, the multiplication can be done in a fraction of a second, but the exhaustive factoring of the result should take billions of years [2].

14.5.3.1 Use of the RSA Scheme

Using the example in Reference [13], let $p = 47$, $q = 59$. Therefore, $n = pq = 2773$ and $\phi(n) = (p - 1)(q - 1) = 2668$. The parameter d is chosen to be relatively prime to $\phi(n)$. For example, choose $d = 157$. Next, the value of e is computed as follows (the details are shown in the next section):

$$ed \text{ modulo } \phi(n) = 1$$

$$157e \text{ modulo } 2668 = 1$$

Therefore, $e = 17$. Consider the plaintext example

ITS ALL GREEK TO ME

By replacing each letter with a two-digit number in the range (01, 26) corresponding to its position in the alphabet, and encoding a blank as 00, the plaintext message can be written as

0920 1900 0112 1200 0718 0505 1100 2015 0013 0500

Each message needs to be expressed as an integer in the range $(0, n - 1)$; therefore, for this example, encryption can be performed on blocks of four digits at a time since this is the maximum number of digits that will always yield a number less than $n - 1 = 2772$. The first four digits (0920) of the plaintext are encrypted as follows:

$$C = (M)^e \text{ modulo-} n = (920)^{17} \text{ modulo-} 2773 = 948$$

Continuing this process for the remaining plaintext digits, we get

$$C = 0948 \ 2342 \ 1084 \ 1444 \ 2663 \ 2390 \ 0778 \ 0774 \ 0219 \ 1655$$

The plaintext is returned by applying the decryption key, as follows:

$$M = (C)^{157} \text{ modulo-} 2773$$

14.5.3.2 How to Compute e

A variation of Euclid's algorithm [14] for computing the gcd of $\phi(n)$ and d is used to compute e . First, compute a series $x_0, x_1, x_2, \dots$, where $x_0 = \phi(n)$, $x_1 = d$, and $x_{i+1} = x_{i-1} \text{ modulo-} x_i$, until an $x_k = 0$ is found. Then the gcd $(x_0, x_1) = x_{k-1}$. For each x_i compute numbers a_i and b_i such that $x_i = a_i x_0 + b_i x_1$. If $x_{k-1} = 1$, then b_{k-1} is the multiplicative inverse of $x_1 \text{ modulo-} x_0$. If b_{k-1} is a negative number, the solution is $b_{k-1} + \phi(n)$.

Example 14.5 Computation of e from d and $\phi(n)$

For the previous example, with $p = 47$, $q = 59$, $n = 2773$, $\phi(n) = 2688$, and d chosen to be 157, use the Euclid algorithm to verify that $e = 17$.

Solution

i	x_i	a_i	b_i	y_i
0	2668	1	0	
1	157	0	1	16
2	156	1	-16	1
3	1	-1	17	

where

$$y_i = \left\lfloor \frac{x_{i-1}}{x_i} \right\rfloor$$

$$x_{i+1} = x_{i-1} - y_i x_i$$

$$a_{i+1} = a_{i-1} - y_i a_i$$

$$b_{i+1} = b_{i-1} - y_i b_i$$

Hence

$$e = b_3 = 17$$

14.5.4 The Knapsack Problem

The classic knapsack problem is illustrated in Figure 14.19. The knapsack is filled with a subset of the items shown with weights indicated in grams. Given the weight of the filled knapsack (the scale is calibrated to deduct the weight of the empty knapsack), determine which items are contained in the knapsack. For this simple example, the solution can easily be found by trial and error. However, if there are 100 possible items in the set instead of 10, the problem may become computationally infeasible.

Let us express the knapsack problem in terms of a knapsack vector and a data vector. The knapsack vector is an n -tuple of distinct integers (analogous to the set of possible knapsack items)

$$\mathbf{a} = a_1, a_2, \dots, a_n$$

The data vector is an n -tuple of binary symbols

$$\mathbf{x} = x_1, x_2, \dots, x_n$$

The knapsack, S , is the sum of a subset of the components of the knapsack vector:

$$\begin{aligned} S &= \sum_{i=1}^n a_i x_i \quad \text{where } x_i = 0, 1 \\ &= \mathbf{a} \mathbf{x} \end{aligned} \tag{14.40}$$

The knapsack problem can be stated as follows: Given S and knowing $\mathbf{a}$, determine $\mathbf{x}$.

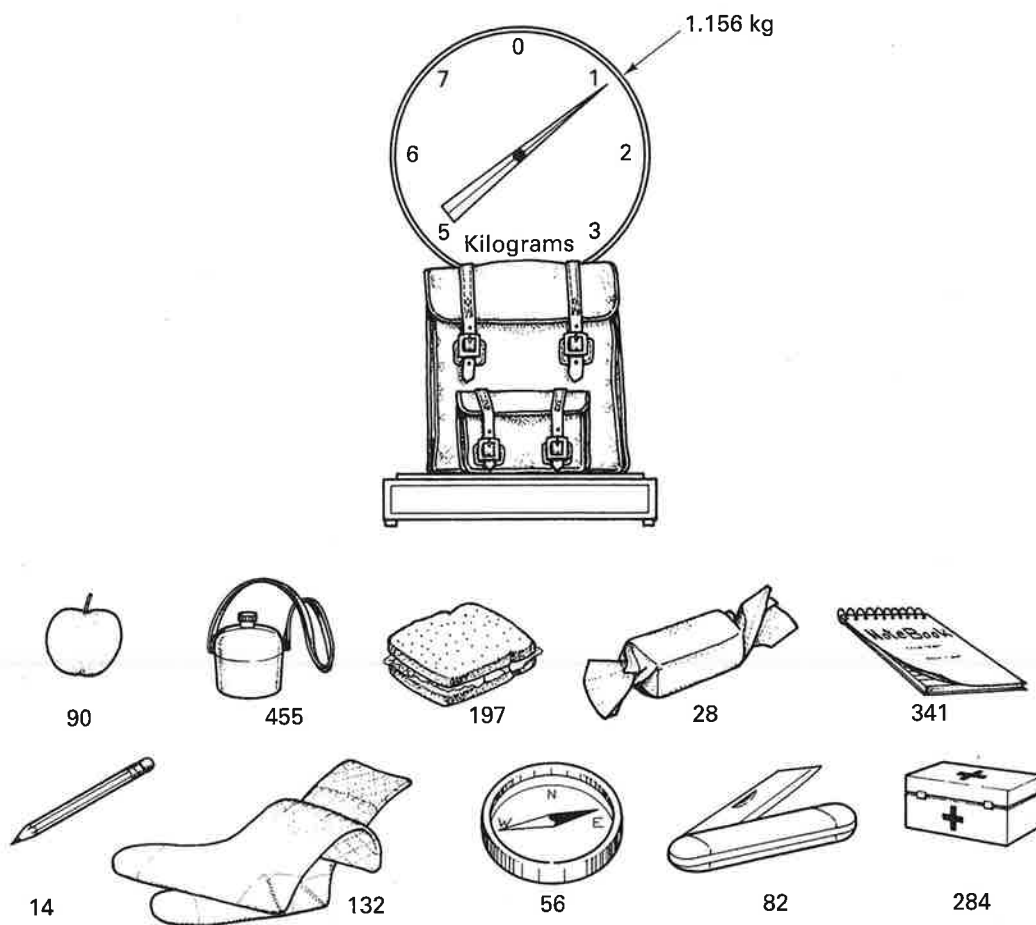


Figure 14.19 Knapsack problem.

Example 14.6 Knapsack Example

Given $\mathbf{a} = 1, 2, 4, 8, 16, 32$ and $S = \mathbf{a}\mathbf{x} = 26$, find $\mathbf{x}$.

Solution

In this example $\mathbf{x}$ is seen to be the *binary* representation of S . The decimal-to-binary conversion should appear more familiar with $\mathbf{a}$ expressed as $2^0, 2^1, 2^2, 2^3, 2^4, 2^5$. The data vector $\mathbf{x}$ is easily found since $\mathbf{a}$ in this example is *super-increasing*, which means that each component of the n -tuple $\mathbf{a}$ is larger than the sum of the preceding components. That is,

$$a_i > \sum_{j=1}^{i-1} a_j \quad i = 2, 3, \dots, n \quad (14.41)$$

When $\mathbf{a}$ is super-increasing, the solution of $\mathbf{x}$ is found by starting with $x_n = 1$ if $S \geq a_n$ (otherwise $x_n = 0$) and continuing according to the relationship

$$x_i = \begin{cases} 1 & \text{if } S - \sum_{j=i+1}^n x_j a_j \geq a_i \\ 0 & \text{otherwise} \end{cases} \quad (14.42)$$

where $i = n - 1, n - 2, \dots, 1$. From Equation (14.42) it is easy to compute $\mathbf{x} = 010110$.

Example 14.7 Knapsack Example

Given $\mathbf{a} = 171, 197, 459, 1191, 2410, 4517$ and $S = \mathbf{ax} = 3798$, find $\mathbf{x}$.

Solution

As in Example 14.6, $\mathbf{a}$ is super-increasing; therefore, we can compute $\mathbf{x}$ using Equation (14.42), which again yields

$$\mathbf{x} = 010110$$

14.5.5 A Public Key Cryptosystem Based on a Trapdoor Knapsack

This scheme, also known as the Merkle–Hellman scheme [15], is based on the formation of a knapsack vector that is not super-increasing and is therefore not easy to solve. However, an essential part of this knapsack is a *trapdoor* that enables the authorized user to solve it.

First, we form a super-increasing n -tuple $\mathbf{a}'$. Then we select a prime number M such that

$$M > \sum_{i=1}^n a'_i \quad (14.43)$$

We also select a random number W , where $1 < W < M$, and we form W^{-1} to satisfy the following relationship:

$$WW^{-1} \text{ modulo-} M = 1 \quad (14.44)$$

the vector $\mathbf{a}'$ and the numbers M , W , and W^{-1} are all kept hidden. Next, we form $\mathbf{a}$ with the elements from $\mathbf{a}'$, as follows:

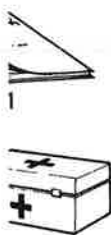
$$a_i = Wa'_i \text{ modulo-} M \quad (14.45)$$

The formation of $\mathbf{a}$ using Equation (14.45) constitutes forming a knapsack vector with a *trapdoor*. When a data vector $\mathbf{x}$ is to be transmitted, we multiply $\mathbf{x}$ by $\mathbf{a}$, yielding the number S , which is sent on the public channel. Using Equation (14.45), S can be written as follows:

$$S = \mathbf{ax} = \sum_{i=1}^n a_i x_i = \sum_{i=1}^n (Wa'_i \text{ modulo-} M) x_i \quad (14.46)$$

The authorized user receives S and, using Equation (14.44), converts it to S' :

$$S' = W^{-1}S \text{ modulo-} M = W^{-1} \sum_{i=1}^n (Wa'_i \text{ modulo-} M) x_i \text{ modulo-} M$$



$$\begin{aligned}
&= \sum_{i=1}^n (W^{-1} W a'_i \text{ modulo-} M) x_i \text{ modulo-} M & (14.47) \\
&= \sum_{i=1}^n a'_i x_i \text{ modulo-} M \\
&= \sum_{i=1}^n a'_i x_i
\end{aligned}$$

Since the authorized user knows the secretly held super-increasing vector $\mathbf{a}'$, he or she can use S' to find $\mathbf{x}$.

14.5.5.1 Use of the Merkle-Hellman Scheme

Suppose that user A wants to construct public and private encryption functions. He first considers the super-increasing vector $\mathbf{a}' = (171, 197, 459, 1191, 2410, 4517)$

$$\sum_{i=1}^6 a'_i = 8945$$

He then chooses a prime number M larger than 8945, a random number W , where $1 \leq W < M$, and calculates W^{-1} to satisfy $WW^{-1} = 1 \text{ modulo-} M$.

$$\left. \begin{array}{l} \text{Choose } M = 9109 \\ \text{choose } W = 2251 \\ \text{then } W^{-1} = 1388 \end{array} \right\} \text{kept hidden}$$

He then forms the trapdoor knapsack vector as follows:

$$\begin{aligned}
a_i &= a'_i 2251 \text{ modulo-} 9109 \\
\mathbf{a} &= 2343, 6215, 3892, 2895, 5055, 2123
\end{aligned}$$

User A makes public the vector $\mathbf{a}$, which is clearly not super-increasing. Suppose that user B wants to send a message to user A .

If $\mathbf{x} = 010110$ is the message to be transmitted, user B forms

$$S = \mathbf{a}\mathbf{x} = 14,165 \text{ and transmits it to user } A$$

User A , who receives S , converts it to S' :

$$\begin{aligned}
S' &= \mathbf{a}'\mathbf{x} = W^{-1}S \text{ modulo-} M \\
&= 1388 \cdot 14,165 \text{ modulo-} 9109 \\
&= 3798
\end{aligned}$$

Using $S' = 3798$ and the super-increasing vector $\mathbf{a}'$, user A easily solves for $\mathbf{x}$.

The Merkle-Hellman scheme is now considered broken [16], leaving the RSA scheme (as well as others discussed later) as the algorithms that are useful for implementing public key cryptosystems.

14.6 PRETTY GOOD PRIVACY

Pretty Good Privacy (PGP) is a security program that was created by Phil Zimmerman [17] and published in 1991 as free-of-charge shareware. It has since become the "de facto" standard for electronic mail (e-mail) and file encryption. PGP, widely used as version 2.6, remained essentially unchanged until PGP version 5.0 (which is compatible with version 2.6) became available. Table 14.9 illustrates the algorithms used in versions 2.6, 5.0, and later.

As listed in Table 14.9, PGP uses a variety of encryption algorithms, including both private-key- and public-key-based systems. A private-key algorithm (with a new session key generated at each session) is used for encryption of the message. The private-key algorithms offered by PGP are International Data Encryption Algorithm (IDEA), Triple-DES (Data Encryption Standard), and CAST (named after the inventors Carlisle Adams and Stafford Tavares [19]). A public-key algorithm is used for the encryption of each session key. The public-key algorithms offered by PGP are the RSA algorithm, described in Section 14.5.3, and the Diffie-Hellman algorithm.

Public-key algorithms are also used for the creation of digital signatures. PGP version 5.0 uses the Digital Signature Algorithm (DSA) specified in the NIST Digital Signature Standard (DSS). PGP version 2.6 uses the RSA algorithm for its digital signatures. If the available channel is insecure for key exchange, it is safest to use a public-key algorithm. If a secure channel is available, then private-key encryption is preferred, since it typically offers improved speed over public-key systems.

The technique for message encryption employed by PGP version 2.6 is illustrated in Figure 14.20. The plaintext is compressed with the ZIP algorithm prior to encryption. PGP uses the ZIP routine written by Jean-Loup Gailly, Mark Alder, and Richard B. Wales [18]. If the compressed text is shorter than the uncompressed text, the compressed text will be encrypted, otherwise the uncompressed text is encrypted.

TABLE 14.9 PGP 2.6 versus PGP 5.0 and Later

Function	PGP Version 2.6 Algorithm Used [17]	PGP Version 5.0 and Later Algorithm Used [18]
Encryption of message using private-key algorithm with private-session key	IDEA	Triple-DES, CAST, or IDEA
Encryption of private-session key with public-key algorithm	RSA	RSA or Diffie-Hellman (the Elgamal variation)
Digital Signature	RSA	RSA and NIST ¹ Digital Signature Standard (DSS) ²
Hash Function used for creating message digest for Digital Signatures	MD5	SHA-1

¹ National Institute of Standards and Technology, a division of the U.S. Department of Commerce.

² Digital Signature Standard selected by NIST.

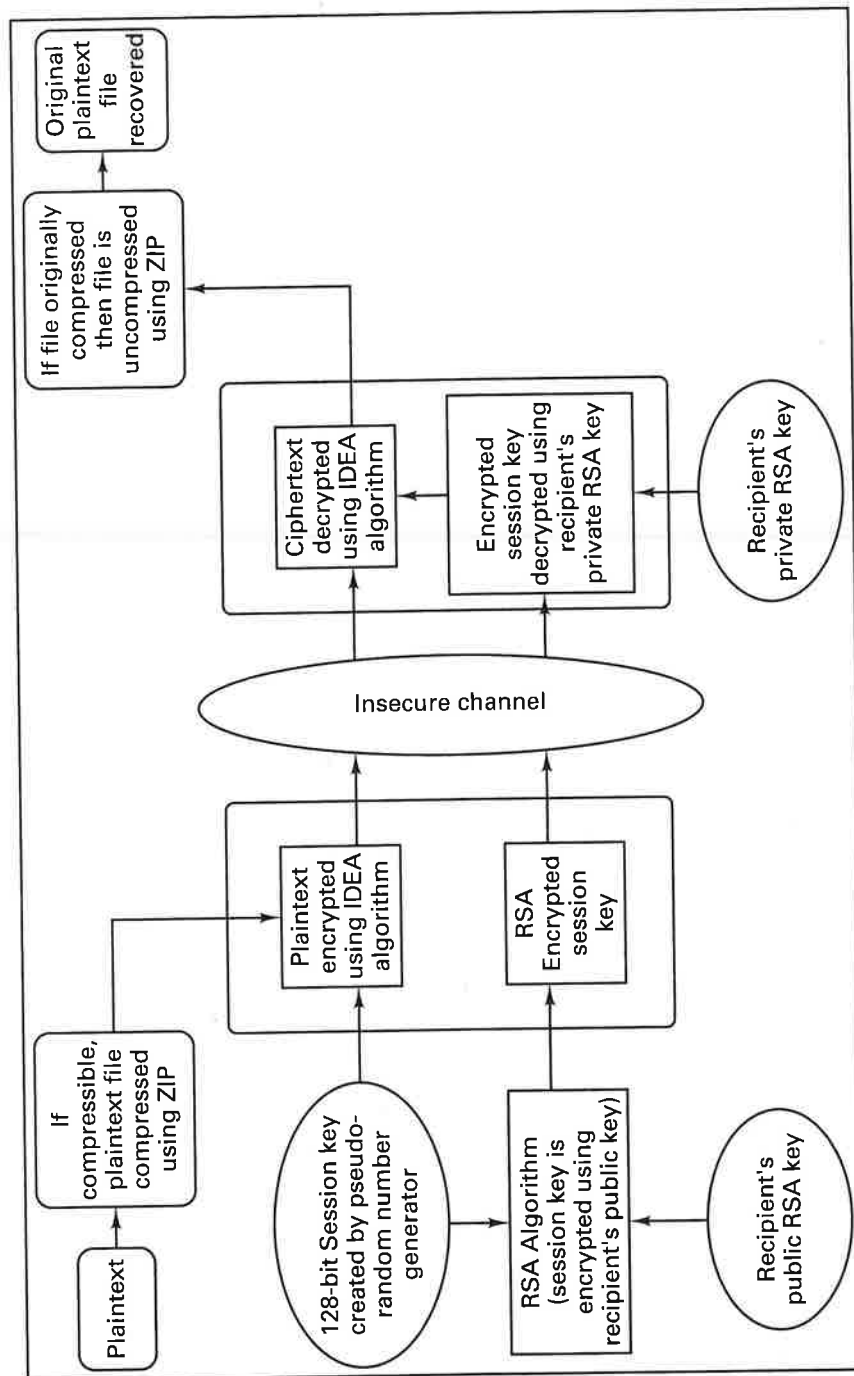


Figure 14.20 The PGP technique.

Small files (approximately 30 characters for ASCII files) will not benefit from compression. Additionally, PGP recognizes files previously compressed by popular compression routines, such as PKZIP, and will not attempt to compress them. Data compression removes redundant character strings in a file and produces a more uniform distribution of characters. Compression provides a shorter file to encrypt and decrypt (which reduces the time needed to encrypt, decrypt, and transmit a file), but compression is also advantageous because it can hinder some cryptanalytic attacks that exploit redundancy. If compression is performed on a file, it should occur *prior to encryption* (never afterwards). Why is that a good rule to follow? Because a *good* encryption algorithm yields ciphertext with a nearly statistically uniform distribution of characters; therefore, if a data compression algorithm came after such encryption, it should result in no compression at all. If any ciphertext can be compressed, then the encryption algorithm that formed that ciphertext was a poor algorithm. A compression algorithm should be *unable* to find redundant patterns in text that was encrypted by a good encryption algorithm.

As shown in Figure 14.20, PGP Version 2.6 begins file encryption by creating a 128-bit session key using a pseudo-random number generator. The compressed plaintext file is then encrypted with the IDEA private-key algorithm using this random session key. The random session key is then encrypted by the RSA public-key algorithm using the *recipient's public key*. The RSA-encrypted session key and the IDEA-encrypted file are sent to the recipient. When the recipient needs to read the file, the encrypted session key is first decrypted with RSA using the *recipient's private key*. The ciphertext file is then decrypted with IDEA using the decrypted session key. After uncompression, the recipient can read the plaintext file.

14.6.1 Triple-DES, CAST, and IDEA

As listed in Table 14.9, PGP offers three block ciphers for message encryption, Triple-DES, CAST, and IDEA. All three ciphers operate on 64-bit blocks of plaintext and ciphertext. Triple-DES has a key size of 168-bits while CAST and IDEA use key lengths of 128 bits.

14.6.1.1 Description of Triple-DES

The Data Encryption Standard (DES) described in Section 14.3.5 has been used since the late 1970s, but some have worried about its security because of its relatively small key size (56 bits). With Triple-DES, the message to be encrypted is run through the DES algorithm 3 times (the second DES operation is run in decrypt mode); each operation is performed with a different 56-bit key. As illustrated in Figure 14.21, this gives the effect of a 168-bit key length.

14.6.1.2 Description of CAST

CAST is a family of block ciphers developed by Adams and Tavares [19]. PGP version 5.0 uses a version of CAST known as CAST5, or CAST-128. This version has a block size of 64-bits and a key length of 128-bits. The CAST algorithm uses six S-boxes with an 8-bit input and a 32-bit output. By comparison, DES uses

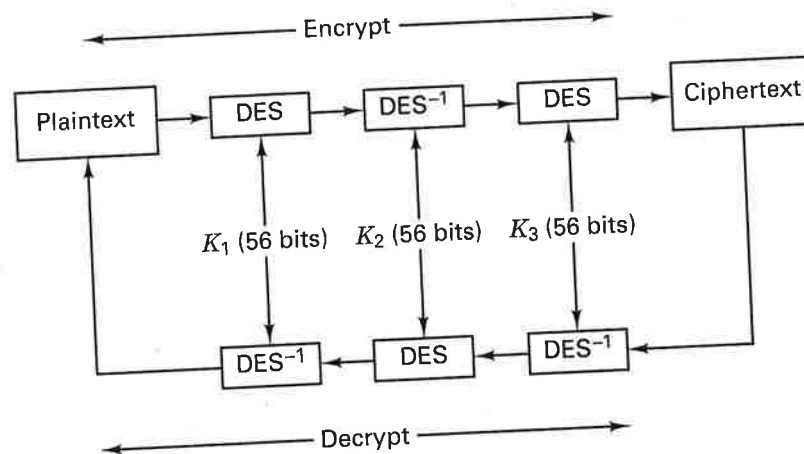


Figure 14.21 Encryption/decryption with triple-DES.

eight *S*-boxes with a 6-bit input and a 4-bit output. The *S*-boxes in Cast-128 were designed to provide highly nonlinear transformations, making this algorithm particularly resistant to cryptanalysis [11].

14.6.1.3 Description of IDEA

The International Data Encryption Algorithm (IDEA) is a block cipher designed by Xuejia Lai and James Massey [19]. It is a 64-bit iterative block cipher (involving eight iterations or rounds) with a 128-bit key. The security of IDEA relies on the use of three types of arithmetic operations on 16-bit words. The operations are addition modulo 2^{16} , multiplication modulo $2^{16} + 1$, and bit-wise exclusive-OR (XOR). The 128-bit key is used for the iterated encryption and decryption in a re-ordered fashion. As shown in Table 14.10, the original key K_0 is divided into eight 16-bit subkeys $Z_x^{(R)}$, where x is the subkey number of the round R . Six of these subkeys are used in round 1, and the remaining two are used in round 2. K_0 is then rotated 25 bits to the left yielding K_1 , which is in turn divided into eight subkeys; the first 4 of these subkeys are used in round 2, and the last four in round 3. The process continues, as shown in Table 14.10, yielding a total of 52 subkeys.

The subkey schedule for each round is listed in Table 14.11 for both encryption and decryption rounds. Decryption is carried out in the same manner as encryption. The decryption subkeys are calculated from the encryption subkeys, as shown in Table 14.11, where it is seen that the decryption subkeys are either the additive or multiplicative inverses of the encryption subkeys.

The message is divided into 64-bit data blocks. These blocks are then divided into four 16-bit subblocks: M_1 , M_2 , M_3 , and M_4 . A sequence of such four subblocks becomes the input to the first round of IDEA algorithm. This data is manipulated for a total of eight rounds. Each round uses a different set of six subkeys as specified in Table 14.11. After a round, the second and third 16-bit data subblocks are

TABLE 14.10 IDEA formation of Subkeys

128-bit key (divided into eight 16-bit subkeys)	Bit string from which keys are derived
$Z_1^1 Z_2^1 Z_3^1 Z_4^1 Z_5^1 Z_6^1 Z_1^2 Z_2^2$	K_0 = Original 128-bit key
$Z_3^2 Z_4^2 Z_5^2 Z_6^2 Z_1^3 Z_2^3 Z_3^3 Z_4^3$	K_1 = 25-bit rotation of K_0
$Z_5^3 Z_6^3 Z_1^4 Z_2^4 Z_3^4 Z_4^4 Z_5^4 Z_6^4$	K_2 = 25-bit rotation of K_1
$Z_1^5 Z_2^5 Z_3^5 Z_4^5 Z_5^5 Z_6^5 Z_1^6 Z_2^6$	K_3 = 25-bit rotation of K_2
$Z_3^6 Z_4^6 Z_5^6 Z_6^6 Z_1^7 Z_2^7 Z_3^7 Z_4^7$	K_4 = 25-bit rotation of K_3
$Z_5^7 Z_6^7 Z_1^8 Z_2^8 Z_3^8 Z_4^8 Z_5^8 Z_6^8$	K_5 = 25-bit rotation of K_4
$Z_1^{\text{out}} Z_2^{\text{out}} Z_3^{\text{out}} Z_4^{\text{out}}$	First 64 bits of K_6 where K_6 = 25-bit rotation of K_5

swapped. After the completion of the eighth round, the four subblocks are manipulated in a final output transformation. For the representation of $Z_x^{(R)}$ shown in Tables 14.10 and 14.11, the round number is shown without parentheses for ease of notation.

Each round consists of the steps shown in Table 14.12. The final values from steps 11–14 form the output of the round. The two inner 16-bit data subblocks (except for the last round) are swapped, and then these four subblocks are the input to the next round. This technique continues for a total of 8 rounds. After round 8, the final output transformation is as follows:

1. $M_1 \times Z_1^{\text{out}}$ (first subkey of output transformation)
2. $M_2 + Z_2^{\text{out}}$
3. $M_3 + Z_3^{\text{out}}$
4. $M_4 \times Z_4^{\text{out}}$

TABLE 14.11 IDEA Subkey Schedule

Round	Set of Encryption Subkeys	Set of Decryption Subkeys
1	$Z_1^1 Z_2^1 Z_3^1 Z_4^1 Z_5^1 Z_6^1$	$(Z_1^{\text{out}})^{-1} - Z_2^{\text{out}} - Z_3^{\text{out}} (Z_4^{\text{out}})^{-1} Z_5^8 Z_6^8$
2	$Z_1^2 Z_2^2 Z_3^2 Z_4^2 Z_5^2 Z_6^2$	$(Z_1^8)^{-1} - Z_2^8 - Z_3^8 (Z_4^8)^{-1} Z_5^7 Z_6^7$
3	$Z_1^3 Z_2^3 Z_3^3 Z_4^3 Z_5^3 Z_6^3$	$(Z_1^7)^{-1} - Z_2^7 - Z_3^7 (Z_4^7)^{-1} Z_5^6 Z_6^6$
4	$Z_1^4 Z_2^4 Z_3^4 Z_4^4 Z_5^4 Z_6^4$	$(Z_1^6)^{-1} - Z_2^6 - Z_3^6 (Z_4^6)^{-1} Z_5^5 Z_6^5$
5	$Z_1^5 Z_2^5 Z_3^5 Z_4^5 Z_5^5 Z_6^5$	$(Z_1^5)^{-1} - Z_2^5 - Z_3^5 (Z_4^5)^{-1} Z_5^4 Z_6^4$
6	$Z_1^6 Z_2^6 Z_3^6 Z_4^6 Z_5^6 Z_6^6$	$(Z_1^4)^{-1} - Z_2^4 - Z_3^4 (Z_4^4)^{-1} Z_5^3 Z_6^3$
7	$Z_1^7 Z_2^7 Z_3^7 Z_4^7 Z_5^7 Z_6^7$	$(Z_1^3)^{-1} - Z_2^3 - Z_3^3 (Z_4^3)^{-1} Z_5^2 Z_6^2$
8	$Z_1^8 Z_2^8 Z_3^8 Z_4^8 Z_5^8 Z_6^8$	$(Z_1^2)^{-1} - Z_2^2 - Z_3^2 (Z_4^2)^{-1} Z_5^1 Z_6^1$
Output Transformation	$Z_1^{\text{out}} Z_2^{\text{out}} Z_3^{\text{out}} Z_4^{\text{out}}$	$(Z_1^1)^{-1} - Z_2^1 - Z_3^1 (Z_4^1)^{-1}$

Example 14.8 The First Round of the IDEA Cipher

Consider that the message is the word "HI", which we first transform to hexadecimal (hex) notation. We start with the ASCII code table in Figure 2.3, where bit 1 is the least significant bit (LSB). We then add an eighth zero-value most significant bit (MSB), which might ordinarily be used for parity, and we transform four bits at a time reading from MSB to LSB. Thus, the letter H in the message transforms to 0048 and

TABLE 14.12 IDEA Operational Steps in Each Round

1. $M_1 \times Z_1^{(R)}$
2. $M_2 + Z_2^{(R)}$
3. $M_3 + Z_3^{(R)}$
4. $M_4 \times Z_4^{(R)}$
5. XOR³ the results from steps 1 and 3.
6. XOR the results from steps 2 and 4.
7. Result from step 5 and $Z_5^{(R)}$ are multiplied.
8. Result from step 6 and 7 are added.
9. Result from step 8 and $Z_6^{(R)}$ are multiplied.
10. Results from steps 7 and 9 are added.
11. XOR the results from steps 1 and 9.
12. XOR the results from steps 3 and 9.
13. XOR the results from steps 2 and 10.
14. XOR the results from steps 4 and 10.

³ The exclusive-OR (XOR) operation is defined as: 0 XOR 0 = 0, 0 XOR 1 = 1, 1 XOR 0 = 1, and 1 XOR 1 = 0

the letter I transforms to 0049. For this example, we choose a 128-bit key, K_0 , expressed with eight groups or *subkeys* of 4-hex digits each, as follows: $K_0 = 0008\ 0007\ 0006\ 0005\ 0004\ 0003\ 0002\ 0001$, where the rightmost subkey is the least significant. Using this key and the IDEA cipher, find the output of round 1.

Solution

The message is first divided into 64-bit data blocks. Each of these blocks is then divided into subblocks, M_i , where $i = 1, \dots, 4$, each subblock containing 16-bits or 4-hex digits. In this example the message "HI" is only 16-bits in length, hence (using hex notation) $M_1 = 4849$ and $M_2 = M_3 = M_4 = 0000$. Addition is performed modulo 2^{16} , and multiplication is performed modulo $2^{16} + 1$. For the first round, the specified 128-bit key is divided into eight 16-bit subkeys starting with the least significant group of hex digits, as follows: $Z_1^{(1)} = 0001$, $Z_2^{(1)} = 0002$, $Z_3^{(1)} = 0003$, $Z_4^{(1)} = 0004$, $Z_5^{(1)} = 0005$, $Z_6^{(1)} = 0006$, $Z_1^{(2)} = 0007$, and $Z_2^{(2)} = 0008$.

The steps outlined in Table 14.11 yield:

1. $M_1 \times Z_1 = 4849 \times 0001 = 4849$.
2. $M_2 + Z_2 = 0000 + 0002 = 0002$.
3. $M_3 + Z_3 = 0000 + 0003 = 0003$.
4. $M_4 \times Z_4 = 0000 \times 0004 = 0000$.
5. The result from step (1) is XOR'ed with the result from step (3) yielding 4849 XOR 0003 = 484A, as follows:

$$\begin{array}{r}
 \begin{array}{cccccc}
 0100 & 1000 & 0100 & 1001 & (4849 \text{ hex converted to binary}) \\
 \text{XOR} & 0000 & 0000 & 0000 & 0011 & (0003 \text{ hex converted to binary}) \\
 \hline
 0100 & 1000 & 0100 & 1010
 \end{array}
 \end{array}$$

Converting back to hex yields: 484A (where A is the hex notation for 1010 binary)

6. Results from steps (2) and (4) are XOR'ed: 0002 XOR 0000 = 0002.
7. Result from step (5) and Z_5 are multiplied: $484A \times 0005 = 6971$.
8. Results from steps (6) and (7) are added: $0002 + 6971 = 6973$.

9. Result from step (8) and Z_6 are multiplied: $6973 \times 0006 = 78B0$.
10. Results from steps (7) and (9) are added: $6971 + 78B0 = E221$.
11. Results from steps (1) and (9) are XOR'ed: $4849 \text{ XOR } 78B0 = 30F9$.
12. Results from steps (3) and (9) are XOR'ed: $0003 \text{ XOR } 78B0 = 78B3$.
13. Results from steps (2) and (10) are XOR'ed: $0002 \text{ XOR } E221 = E223$.
14. Results from steps (4) and (10) are XOR'ed: $0000 \text{ XOR } E221 = E221$.

The output of round 1 (the result from steps 11–14) is: 30F9 78B3 E223 E221. Prior to the start of round 2, the two inner words of the round 1 output are swapped. Then, seven additional rounds and a final output transformation are performed.

14.6.2 Diffie-Hellman (Elgamal Variation) and RSA

For encryption of the session key, PGP offers a choice of two public-key encryption algorithms, RSA and the Diffie-Hellman (Elgamal variation) protocol. PGP allows for key sizes of 1024 to 4096 bits for RSA or Diffie-Hellman algorithms. The key size of 1024 bits is considered safe for exchanging most information. The security of the RSA algorithm (see Section 14.5.3) is based on the difficulty of factoring large integers.

The Diffie-Hellman protocol was developed by Whitfield Diffie, Martin E. Hellman, and Ralph C. Merkle in 1976 [19, 20] for public-key exchange over an insecure channel. It is based on the difficulty of the discrete logarithm problem for finite fields [21]. It assumes that it is computationally infeasible to compute g^{ab} knowing only g^a and g^b . U.S. Patent 4,200,770, which expired in 1997, covers the Diffie-Hellman protocol and variations such as Elgamal. The Elgamal variation, which was developed by Taher Elgamal, extends the Diffie-Hellman protocol for message encryption. PGP employs the Elgamal variation of Diffie-Hellman for the encryption of the session-key.

14.6.2.1 Description of Diffie-Hellman, Elgamal Variant:

The protocol has two-system parameter n and g that are both public. Parameter n is a large prime number, and parameter g is an integer less than n that has the following property: for every number p between 1 and $n - 1$ inclusive, there is a power k of g such that $g^k = p \text{ mod } n$. The Elgamal encryption scheme [19, 21] that allows user B to send a message to user A is described below:

- User A randomly chooses a large integer, a (this is user A's private key).
- User A's public key is computed as: $y = g^a \text{ mod } n$.
- User B wishes to send a message M to user A. User B first generates a random number k that is less than n .
- User B computes the following:

$$y_1 = g^k \text{ mod } n$$

$$y_2 = M \times (y^k \text{ mod } n) \text{ (recall that } y \text{ is users A's public key).}$$

- User B sends the ciphertext (y_1, y_2) to user A.

- Upon receiving ciphertext (y_1, y_2) , user A computes the plaintext message M as follows:

$$M = \frac{y_2}{y_1^a \bmod n}$$

Example 14.9 Diffie-Hellman (Elgamal variation) for Message Encryption

Consider that the public-system parameters are $n = 11$ and $g = 7$. Suppose that user A chooses the private key to be $a = 2$. Show how user A's public key is computed. Also, show how user B would encrypt a message $M = 13$ to be sent to user A, and how user A subsequently decrypts the ciphertext to yield the message.

Solution

User A's public key ($y = g^a \bmod n$) is computed as: $y = 7^2 \bmod 11 = 5$. User B wishes to send message $M = 13$ to user A. For this example, let user B randomly choose a value of k (less than $n = 11$) to be $k = 1$. User B computes the ciphertext pair

$$y_1 = g^k \bmod n = 7^1 \bmod 11 = 7$$

$$y_2 = M \times (y^k \bmod n) = 13 \times (5^1 \bmod 11) = 13 \times 5 = 65$$

User A receives the ciphertext $(7, 65)$, and computes message M as follows:

$$M = \frac{y_2}{y_1^a \bmod n} = \frac{65}{7^2 \bmod 11} = \frac{65}{5} = 13$$

14.6.3 PGP Message Encryption

The private-key algorithms that PGP uses for message encryption were presented in Section 14.6.1. The public-key algorithms that PGP uses to encrypt the private-session key were presented in Section 14.6.2. The next example combines the two types of algorithms to illustrate the PGP encryption technique shown in Figure 14.20.

Example 14.10 PGP Use of RSA and IDEA for Encryption

For the encryption of the session key, use the RSA public-key algorithm with the parameters taken from Section 14.5.3.1, where $n = pq = 2773$, the encryption key is $e = 17$, and the decryption key is $d = 157$. The encryption key is the recipient's public key, and the decryption key is the recipient's private key. From Example 14.8, use the session key $K_0 = 0008\ 0007\ 0006\ 0005\ 0004\ 0003\ 0002\ 0001$, and the ciphertext of 30F9 78B3 E223 E221 representing the message "HI", where all the digits are shown in hexadecimal notation. (Note that the ciphertext was created by using only one round of the IDEA algorithm. In the actual implementation, 8 rounds plus an output transformation are performed.) Encrypt the session key, and show the PGP transmission that would be made.

Solution

Following the description in Section 14.5.3.1, the session key will be encrypted using the RSA algorithm with the recipient's public key of 17. For ease of calculation with a simple calculator, let us first transform the session key into groups made up of base-10 digits. In keeping with the requirements of the RSA algorithm, the value ascribed to any group may not exceed $n - 1 = 2772$. Therefore, let us express the 128-bit key in terms of 4-digit groups, where we choose the most significant group (leftmost) to represent 7 bits, and the balance of the 11 groups to represent 11 bits each. The transfor-

mation from base-16 to base-10 digits can best be viewed as a two-step process, (1) conversion to binary and, (2) conversion to base 10. The result is $K_0 = 0000\ 0032\ 0000\ 1792\ 0048\ 0001\ 0512\ 0064\ 0001\ 1024\ 0064\ 0001$. Recall from Equation 14.32, that $C = (M)^e \text{ modulo-} n$ where M will be one of the 4-digit groups of K_0 . The leftmost four groups are encrypted as:

$$\begin{aligned} C_{12} &= (0000)^{17} \text{ mod } 2773 = 0. \\ C_{11} &= (0032)^{17} \text{ mod } 2773 = 2227. \\ C_{10} &= (0000)^{17} \text{ mod } 2773 = 0. \\ C_9 &= (1792)^{17} \text{ mod } 2773 = 2704. \end{aligned}$$

An efficient way to compute modular exponentiation is to use the Square-and-Multiply algorithm. This algorithm [21] reduces the number of modular multiplications needed to be performed from $e - 1$ to at most 2ℓ , where ℓ is the number of bits in the binary representation. Let us demonstrate the use of the Square-and-Multiply algorithm by encrypting one of the session-key decimal groups (the eleventh group from the right, $M_{11} = 0032$), where $n = 2773$ and $e = 17$. In using this algorithm, we first convert e to its binary representation (17 decimal = 10001 binary).

The calculations are illustrated in Table 14.13. Modulo- n math is used, where $n = 2773$ in this example. The second column contains the binary code, with the most significant bit (MSB) in row 1. Each bit value in this column acts to control a result in column 3. The starting value, placed in column 3 row 0, is always 1. Then, the result for any row in column 3 depends on the value of the bit in the corresponding row in column 2; if that entry contains a "1," then the previous row-result is squared and multiplied by the plaintext (32 for this example). If a row in the second column contains a "0," then the result of that row in column 3 equals only the square of the previous row's result. The final value is the encrypted ciphertext ($C = 2227$). Repeating this method for each of the twelve decimal groups that comprise K_0 results in the ciphertext of the session key to be: $C = 0000\ 2227\ 0000\ 2704\ 0753\ 0001\ 1278\ 0272\ 0001\ 1405\ 0272\ 0001$. This RSA-encrypted session key (represented here in decimal) together with the IDEA-encrypted message of 30F9 78B3 E223 E221 (represented here in hex) can now be transmitted over an insecure channel.

TABLE 14.13 The Square-and-Multiply Algorithm with Plaintext = 32

Row Number	Binary representation of e (MSB first)	Modulo multiplication (modulo 2773)
0		1
1	1	$1^2 \times 32 = 32$
2	0	$32^2 = 1024$
3	0	$1024^2 = 382$
4	0	$382^2 = 1728$
5	1	$1728^2 \times 32 = 2227$

14.6.4 PGP Authentication and Signature

The public key algorithms can be used to authenticate or "sign" a message. As illustrated in Figure 14.18, a sender can encrypt a document with his private key (which no one else has access to) prior to encrypting it with the recipient's public

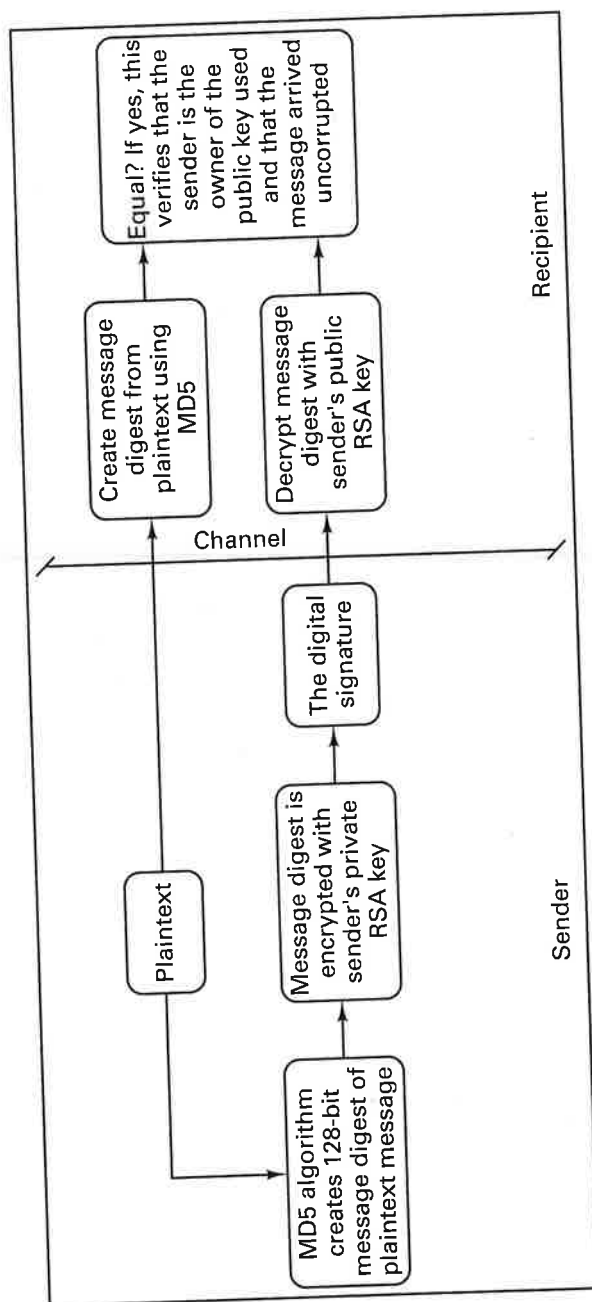


Figure 14.22 PGP signature technique.

key. The recipient must first use his private key to decrypt the message, followed by a second decryption using the sender's public key. This technique encrypts the message for secrecy and also provides authentication of the sender.

Because of the slowness of public-key algorithms, PGP allows for a different method of authenticating a sender. Instead of the time-consuming process of encrypting the entire plaintext message, the PGP approach encrypts a fixed-length message digest created with a one-way hash function. The encryption of the message digest is performed using a public-key algorithm. This method is known as a *digital signature* and is shown in Figure 14.22. A digital signature is used to provide authentication of both *the sender* and *the message*. Authentication of the message provides a verification that the message was not altered in some way. Using this technique, if a message has been altered in any way (i.e. by a forger), its message digest will be different.

PGP version 2.6 uses the MD5 (Message Digest 5) algorithm to create a 128-bit message digest (or hash value) of the plaintext. This hash value is then encrypted with the sender's private key and sent with the plaintext. When the recipient receives the message, he will first decrypt the message digest with the sender's public key. The recipient will then apply the hash function to the plaintext and compare the two message digests. If they match, the signature is valid. In Figure 14.22, the message is sent without encryption (as plaintext), but it may be encrypted by the method illustrated in Figure 14.20.

14.6.4.1 MD5 and SHA-1

MD5 and SHA-1 are hash functions. A hash function $H(x)$ takes an input and returns a fixed-size string h , called the hash value (also known as a message digest). A cryptographic hash function has the following properties:

1. The output length is fixed.
2. The hash value is relatively simple to compute.
3. The function is one way—in other words, it is hard to invert. If given a hash value h , it is computationally infeasible to find the function's input x .
4. The function is *collision free*. A collision-free hash function is a function for which it is infeasible that two different messages will create the same hash value.

The MD5 algorithm used in PGP version 2.6 creates a 128-bit message digest. The MD5 algorithm processes the text in 512-bit blocks through four rounds of data manipulation. Each round uses a different nonlinear function that consists of the logical operators AND, OR, NOT or XOR. Each function is performed 16 times in a round. Bit shifts and scalar additions are also performed in each round [19]. Hans Dobbertin [18] has determined that collisions may exist in MD5. Because of this potential weakness, the PGP specification recommends using the Digital Signature Standard (DSS). DSS uses the SHA-1 (Secure Hash Algorithm-1) algorithm. The SHA-1 algorithm takes a message of less than 2^{64} bits in length and produces a 160-bit

message digest. SHA-1 is similar to MD5 in that it uses a different nonlinear function in each of its 4 rounds. In SHA-1, each function is performed 20 times per round. SHA-1 also uses various scalar additions and bit shifting. The algorithm is slightly slower than MD5 but the larger message digest (160-bit versus 128 bit) makes it more secure against brute-force attacks [19]. A brute-force attack consists of trying many input combinations in an attempt to match the message digest under attack.

14.6.4.2 Digital Signature Standard and RSA

For digital signatures, PGP version 2.6 uses the RSA algorithm for encryption of the hash value produced by the MD5 function; however, versions 5.0 and later adhere to the NIST Digital Signature Standard (DSS) [22]. The NIST DSS requires the use of the SHA-1 hash function. The hash value is then encrypted using the Digital Standard Algorithm (DSA). Like the Diffie-Hellman protocol, DSA is based on the discrete logarithm problem. (Reference [22] contains a detailed description of DSA.)

14.7 CONCLUSION

In this chapter we have presented the basic model and goals of the cryptographic process. We looked at some early cipher systems and reviewed the mathematical theory of secret communications established by Shannon. We defined a system that can exhibit perfect secrecy and established that such systems can be implemented but that they are not practical for use where high-volume communications are required. We also considered practical security systems that employ Shannon's techniques (known as confusion and diffusion) to frustrate the statistical endeavors of a cryptanalyst.

The outgrowth of Shannon's work was utilized by IBM in the LUCIFER system, which later grew into the National Bureau of Standards' Data Encryption Standard (DES). We outlined the DES algorithm in detail. We also considered the use of linear feedback shift registers (LFSR) for stream encryption systems, and demonstrated the intrinsic vulnerability of an LFSR used as a key generator.

We also looked at the area of public-key cryptosystems and examined two schemes, the Rivest-Shamir-Adelman (RSA) scheme, based on the product of two large prime numbers, and the Merkle-Hellman scheme, based on the classical knapsack problem. Finally, we looked at the novel scheme of Pretty Good Privacy (PGP), developed by Phil Zimmerman and published in 1991. PGP utilizes the benefits of both private and public-key systems and has proven to be an important file-encryption method for sending data via electronic mail.

REFERENCES

1. Kahn, D., *The Codebreakers*, Macmillan Publishing Company, New York, 1967.
2. Diffie, W., and Hellman, M.E., "Privacy and Authentication: An Introduction to Cryptography," *Proc. IEEE*, vol. 67, no. 3, Mar. 1979, pp. 397-427.

near function
es per round.
hm is slightly
akes it more
f trying many
tack.

or encryption
. 5.0 and later
DSS requires
ted using the
ocol, DSA is
a detailed de-

cryptographic
mathematical
system that can
ented but that
e required. We
niques (known
ptanalyst.
LUCIFER sys-
ta Encryption
considered the
n systems, and
nerator.
examined two
product of two
n the classical
r Good Privacy
GP utilizes the
e an important

ork, 1967.
roduction to Cryp-

ion Chap. 14

3. Beker, H., and Piper, F., *Cipher Systems*, John Wiley & Sons, Inc., New York, 1982.
4. Denning, D.E.R., *Cryptography and Data Security*, Addison-Wesley Publishing Company, Reading, Mass., 1982.
5. Shannon, C.E., "Communication Theory of Secrecy Systems," *Bell Syst. Tech. J.*, vol. 28, Oct. 1949, pp. 656-715.
6. Hellman, M. E., "An Extension of the Shannon Theory Approach to Cryptography," *IEEE Trans. Inf. Theory*, vol. IT23, May 1978, pp. 289-294.
7. Smith, J. L., "The Design of Lucifer, a Cryptographic Device for Data Communications," *IBM Research Rep. RC-3326*, 1971.
8. Feistel, H. "Cryptography and Computer Privacy," *Sci. Am.*, vol. 228, no. 5, May 1973, pp. 15-23.
9. National Bureau of Standards, "Data Encryption Standard," *Federal Information Processing Standard (FIPS)*, Publication no. 46, Jan. 1977.
10. United States Senate Select Committee on Intelligence, "Unclassified Summary: Involvement of NSA in the Development of the Data Encryption Standard," *IEEE Commun. Soc. Mag.*, vol. 16, no. 6, Nov. 1978, pp. 53-55.
11. Stallings, W., *Cryptography and Network Security, Second Edition*, Prentice Hall, Upper Saddle River, NJ, 1998.
12. Diffie, W., and Hellman, M. E., "New Directions in Cryptography," *IEEE Trans. Inf. Theory*, vol. IT22, Nov. 1976, pp. 644-654.
13. Rivest, R.L., Shamir, A., and Adelman, L., "On Digital Signatures and Public Key Cryptosystems," *Commun. ACM*, vol. 21, Feb. 1978, pp. 120-126.
14. Knuth, D. E., *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, 2nd ed., Addison-Wesley Publishing Company, Reading, Mass., 1981.
15. Merkel, R. C., and Hellman, M. E., "Hiding Information and Signatures in Trap-Door Knapsacks," *IEEE Trans. Inf. Theory*, vol. IT24, Sept. 1978, pp. 525-530.
16. Shamir, A., "A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem," *IEEE 23rd Ann. Symp. Found. Comput. Sci.*, 1982, pp. 145-153.
17. Zimmerman, P. *The Official PGP User's Guide*, MIT Press, Cambridge, 1995.
18. *PGP Freeware User's Guide, Version 6.5*, Network Associates, Inc., 1999.
19. Schneier, B., *Applied Cryptography*, John Wiley & Sons, New York, 1996.
20. Hellman, M. E., Martin, Bailey, Diffie, W., and Merkle, R. C., *United States Patent 4,200,700: Cryptographic Apparatus and Method*, United States Patent and Trademark Office, Washington, DC, 1980.
21. Stinson, Douglas, *Cryptography Theory and Practice*. CRC Press, Boca Raton, FL, 1995.
22. *Digital Signature Standard* (Federal Information Processing Standards Publication 186-1), Government Printing Office, Springfield, VA, Dec. 15, 1998.

PROBLEMS

- 14.1.** Let X be an integer variable represented with 64 bits. The probability is $\frac{1}{2}$ that X is in the range $(0, 2^{16} - 1)$, the probability is $\frac{1}{4}$ that X is in the range $(2^{16}, 2^{32} - 1)$, and the probability is $\frac{1}{4}$ that X is in the range $(2^{32}, 2^{64} - 1)$. Within each range the values are equally likely. Compute the entropy of X .

- 14.2. A set of equally likely weather messages are: sunny (S), cloudy (C), light rain (L), and heavy rain (H). Given the added information concerning the time of day (morning or afternoon), the probabilities change as follows:

$$\text{Morning: } P(S) = \frac{1}{8}, P(C) = \frac{1}{8}, P(L) = \frac{3}{8}, P(H) = \frac{3}{8}$$

$$\text{Afternoon: } P(S) = \frac{3}{8}, P(C) = \frac{3}{8}, P(L) = \frac{1}{8}, P(H) = \frac{1}{8}$$

- (a) Find the entropy of the weather message.
 - (b) Find the entropy of the message conditioned on the time of day.
- 14.3. The Hawaiian alphabet has only 12 letters—the vowels, a, e, i, o, u, and the consonants, h, k, l, m, n, p, w. Assume that each vowel occurs with probability 0.116, and that each consonant occurs with probability 0.06. Also assume that the average number of information bits per letter is the same as that for the English language. Calculate the unicity distance for an encrypted Hawaiian message if the key sequence consists of a random permutation of the 12-letter alphabet.
- 14.4. Estimate the unicity distance for an English language encryption system that uses a key sequence made up of 10 random alphabetic characters:
- (a) Where each key character can be any one of the 26 letters of the alphabet (duplicates are allowed).
 - (b) Where the key characters may not have any duplicates.
- 14.5. Repeat Problem 14.4 for the case where the key sequence is made up of ten integers randomly chosen from the set of numbers 0 to 999.
- 14.6. (a) Find the unicity distance for a DES system which encrypts 64-bit blocks (eight alphabetic characters) using a 56-bit key.
- (b) What is the effect on the unicity distance in part (a) if the key is increased to 128 bits?
- 14.7. In Figures 14.8 and 14.9, *P*-boxes and *S*-boxes alternate. Is this arrangement any more secure than if all the *P*-boxes were first grouped together, followed by all the *S*-boxes similarly grouped together? Justify your answer.
- 14.8. What is the output of the first iteration of the DES algorithm when the plaintext and the key are each made up of zero sequences?
- 14.9. Consider the 10-bit plaintext sequence 0 1 0 1 1 0 1 0 0 1 and its corresponding ciphertext sequence 0 1 1 1 0 1 1 0 1 0, where the rightmost bit is the earliest bit. Describe the five-stage linear feedback shift register (LFSR) that produced the key sequence and show the initial state of the register. Is the output sequence of maximal length?
- 14.10. Following the RSA algorithm and parameters in Example 14.5, compute the encryption key, e , when the decryption key is chosen to be 151.
- 14.11. Given e and d that satisfy $ed \bmod \phi(n) = 1$, and a message that is encoded as an integer number, M , in the range $(0, n - 1)$ such that the $\gcd(M, n) = 1$. Prove that $(M^e \bmod n)^d \bmod n = M$.
- 14.12. Use the RSA scheme to encrypt the message $M = 3$. Use the prime numbers $p = 5$ and $q = 7$. Choose the decryption key, d , to be 11, and calculate the value of the encryption key, e .
- 14.13. Consider the following for the RSA scheme.
- (a) If the prime numbers are $p = 7$ and $q = 11$, list five allowable values for the decryption key, d .

- (b) If the prime numbers are $p = 13$, $q = 31$, and the decryption key is $d = 37$, find the encryption key, e , and describe how you would use it to encrypt the word "DIGITAL."
- 14.14.** Use the Merkle–Hellman public key scheme with the super-increasing vector, $\mathbf{a}' = 1, 3, 5, 10, 20$. Use the following additional parameters: a large prime number $M = 51$ and a random number $W = 37$.
- (a) Find the nonsuper-increasing vector, $\mathbf{a}$, to be made public, and encrypt the data vector 1 1 0 1 1.
- (b) Show the steps by which an authorized receiver decrypts the ciphertext.
- 14.15.** Using the Diffie–Hellman (Elgamal variation) protocol, encrypt the message $M = 7$. The system parameters are $n = 17$ and $g = 3$. The recipient's private key is $a = 4$. Determine the recipient's public key. For message encryption with the randomly selected k , use $k = 2$. Verify the accuracy of the ciphertext by performing decryption using the recipient's private key.
- 14.16.** Find the hexadecimal (hex) value of the message "no" after one round of the IDEA algorithm. The session key in hex notation is = 0002 0003 0002 0003 0002 0003 0002 0003, where the rightmost 4-digit group represents the subkey Z_1 . For the message "no," let each ASCII character be represented by a 16-bit data subblock, where "n" = 006E and "o" = 006F.
- 14.17.** In the PGP Example 14.10, the IDEA session key is encrypted using the RSA algorithm. The resulting encrypted session key (in base-10 notation) was: 0000 2227 0000 2704 0753 0001 1278 0272 0001 1405 0272 0001, where the least significant (rightmost) group is group 1. Using the decryption key, decrypt group 11 of this session key using the Square-and-Multiply technique.

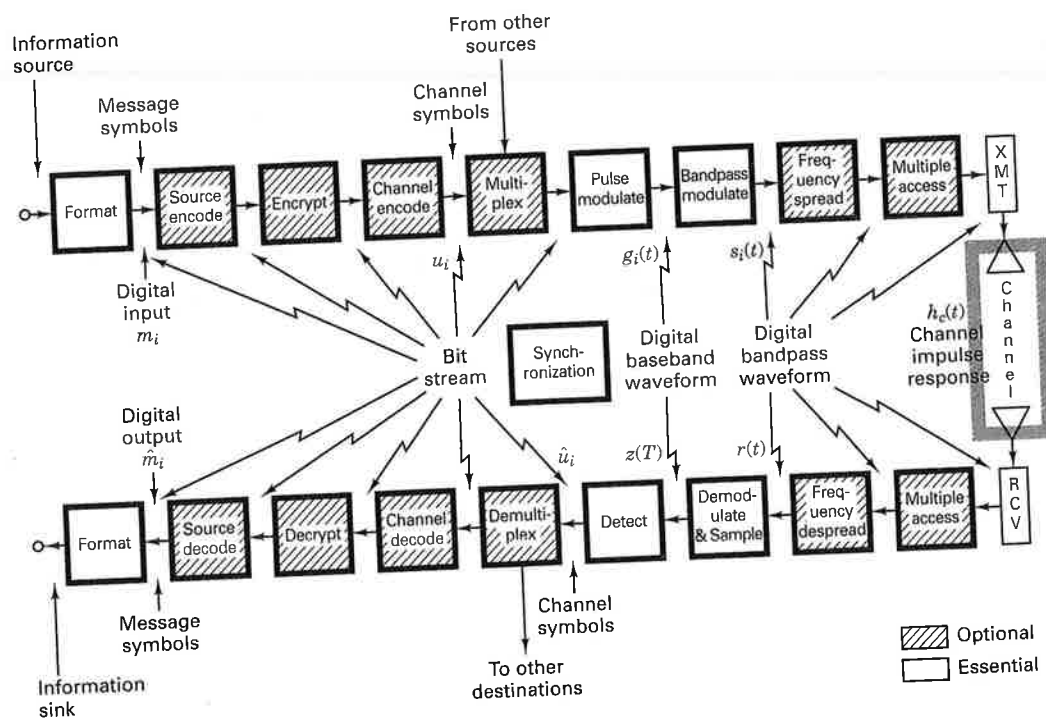
QUESTIONS

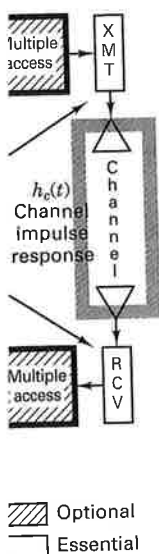
- 14.1.** What are the two major requirements for a useful *cryptosystem*? (See Section 14.1.2.)
- 14.2.** Shannon suggested two encryption concepts that he termed *confusion* and *diffusion*. Explain what these terms mean. (See Section 14.3.1.)
- 14.3.** If *high-level security* is desired, explain why a linear feedback shift register (LFSR) would not be used. (See Section 14.4.2.)
- 14.4.** Explain the major difference between conventional cryptosystems and *public key cryptosystems*. (See Section 14.5.)
- 14.5.** Describe the steps used for message encryption employed by the *Data Encryption Standard* (DES). How different is the operation when using Triple-DES? (See Sections 14.3.5 and 14.6.1.1)
- 14.6.** Describe the steps used for message encryption employed by version 2.6 of the *Pretty Good Privacy* (PGP) technique. (See Section 14.6.1.3.)

EXERCISES

Using the Companion CD, run the exercises associated with Chapter 14.

Fading Channels





When the mechanisms that cause fading in communication channels were first modeled in the 1950s and 1960s, the principles developed were primarily applied to over-the-horizon communications covering a wide range of frequency bands. The 3-30 MHz high-frequency (HF) band used for ionospheric propagation, and the 300 MHz-3 GHz ultra-high-frequency (UHF), as well as the 3-30 GHz super-high-frequency (SHF) bands used for tropospheric scatter are examples of channels affected by fading phenomena. Although the fading effects in mobile radio channels are somewhat different than those encountered in ionospheric and tropospheric channels, the early models are still quite useful in helping to characterize the fading effects in mobile digital communication systems. This chapter emphasizes so-called *Rayleigh fading*, primarily in the UHF band, that affects mobile systems, such as cellular and personal communication systems (PCS). Emphasis is also placed on the fundamental fading manifestations, the types of degradation, and the methods for mitigating the degradation. Two examples of specific mitigation techniques are examined—the Viterbi equalizer, implemented in the Global System for Mobile (GSM) Communications, and the Rake receiver, used in CDMA systems built to meet Interim Standard-95 (IS-95).

15.1 THE CHALLENGE OF COMMUNICATING OVER FADING CHANNELS

In the analysis of communication system performance, the classical (ideal) additive-white-Gaussian-noise (AWGN) channel, with statistically independent Gaussian noise samples corrupting data samples free of intersymbol interference

(ISI), is the usual starting point for developing basic performance results. An important source of performance degradation is thermal noise generated in the receiver. Another source of degradation stems from both natural and man-made sources of noise and interference that enter the receiving antenna, and can be quantified by a parameter called *antenna temperature*. (See Section 5.5.5.) Thermal noise typically has a flat power spectral density over the signal band and a zero-mean Gaussian voltage probability density function (pdf). In mobile communication systems, the external noise and interference are often more significant than the receiver thermal noise. When modeling practical systems, the next step is the introduction of bandlimiting filters. Filtering in the transmitter usually serves to satisfy some regulatory requirement on spectral containment. Filtering in the receiver is often the result of implementing a matched filter, as treated in Section 3.2.2. Due to the bandlimiting and phase-distortion properties of filters, special signal design and equalization techniques may be required to mitigate the filter-induced ISI.

If a radio channel's propagating characteristics are not specified, one usually infers that the signal attenuation versus distance behaves as if propagation takes place over ideal free space. The model of free space treats the region between the transmit and receive antennas as being free of all objects that might absorb or reflect radio frequency (RF) energy. It also assumes that, within this region, the atmosphere behaves as a perfectly uniform and nonabsorbing medium. Furthermore, the earth is treated as being infinitely far away from the propagating signal (or, equivalently, as having a reflection coefficient that is negligible). Basically, in this idealized free-space model, the attenuation of RF energy between the transmitter and receiver behaves according to an inverse-square law. The received power expressed in terms of transmitted power is attenuated by a factor $L_s(d)$. This factor, shown below, is called *path loss* or *free space loss*, and is predicated on the receiving antenna being isotropic (see Section 5.3.1.1).

$$L_s(d) = \left(\frac{4 \pi d}{\lambda} \right)^2 \quad (15.1)$$

In Equation (15.1), d is the distance between the transmitter and the receiver, and λ is the wavelength of the propagating signal. For this case of idealized propagation, received signal power is very predictable. For most practical channels, where signal propagation takes place in the atmosphere and near the ground, the free-space propagation model is inadequate to describe the channel behavior and predict system performance. In a wireless mobile communication system, a signal can travel from transmitter to receiver over multiple reflective paths. This phenomenon, referred to as *multipath propagation*, can cause fluctuations in the received signal's amplitude, phase, and angle of arrival, giving rise to the terminology *multipath fading*. Another name, *scintillation*, which originated in radio astronomy, is used to describe the fading caused by physical changes in the propagating medium, such as variations in the electron density of the ionospheric layers that reflect high frequency (HF) radio signals. Both fading and scintillation refer to a signal's random fluctuations; the main difference is that scintillation involves mechanisms (e.g., electrons) that are much smaller than a wavelength. The end-to-end modeling

and design of systems that incorporate techniques to mitigate the effects of fading are usually more challenging than those whose sole source of performance degradation is AWGN.

15.2 CHARACTERIZING MOBILE-RADIO PROPAGATION

Figure 15.1 represents an overview of fading-channel manifestations. It starts with two types of fading effects that characterize mobile communications: large-scale fading and small-scale fading. Large-scale fading represents the average signal power attenuation or the path loss due to motion over large areas. In Figure 15.1, the large-scale fading manifestation is shown in blocks 1, 2, and 3. This phenomenon is affected by prominent terrain contours (e.g., hills, forests, billboards, clumps of buildings, etc.) between the transmitter and receiver. The receiver is often said to be “shadowed” by such prominences. The statistics of large-scale fading provide a way of computing an estimate of path loss as a function of distance. This is often described in terms of a mean-path loss (n th-power law) and a log-normally distributed variation about the mean. Small-scale fading refers to the dramatic changes in signal amplitude and phase that can be experienced as a result of small changes (as small as a half-wavelength) in the spatial positioning between a receiver and transmitter. As indicated in Figure 15.1, blocks 4, 5, and 6, small-scale fading manifests itself in two mechanisms—time-spreading of the signal (or signal dispersion) and time-variant behavior of the channel. For mobile-radio applications, the channel is time-variant because motion between the transmitter and receiver results in propagation path changes. The rate of change of these propagation conditions accounts for the fading rapidity (rate of change of the fading impairments). Small-scale fading is called *Rayleigh fading* if there are multiple reflective paths that are large in number, and if there is no line-of-sight signal component; the envelope of such a received signal is statistically described by a Rayleigh pdf. When there is a dominant nonfading signal component present, such as a line-of-sight propagation path, the small-scale fading envelope is described by a Rician pdf [1]. In other words, the small-scale fading statistics are said to be Rayleigh whenever the line of sight path is blocked, and Rician otherwise. A mobile radio roaming over a large area must process signals that experience both types of fading: small-scale fading superimposed on large-scale fading.

Large-scale fading (attenuation or path loss) can be considered as a spatial average over the small-scale fluctuations of the signal. It is generally evaluated by averaging the received signal over 10 to 30 wavelengths, in order to decouple the small-scale (mostly Rayleigh) fluctuations from the large-scale shadowing effects (typically log-normal). There are three basic mechanisms that impact signal propagation in a mobile communication system [1]:

- *Reflection* occurs when a propagating electromagnetic wave impinges upon a smooth surface with very large dimensions relative to the RF signal wavelength (λ).
- *Diffraction* occurs when the propagation path between the transmitter and receiver is obstructed by a dense body with dimensions that are large relative to λ , causing secondary waves to be formed behind the obstructing body.

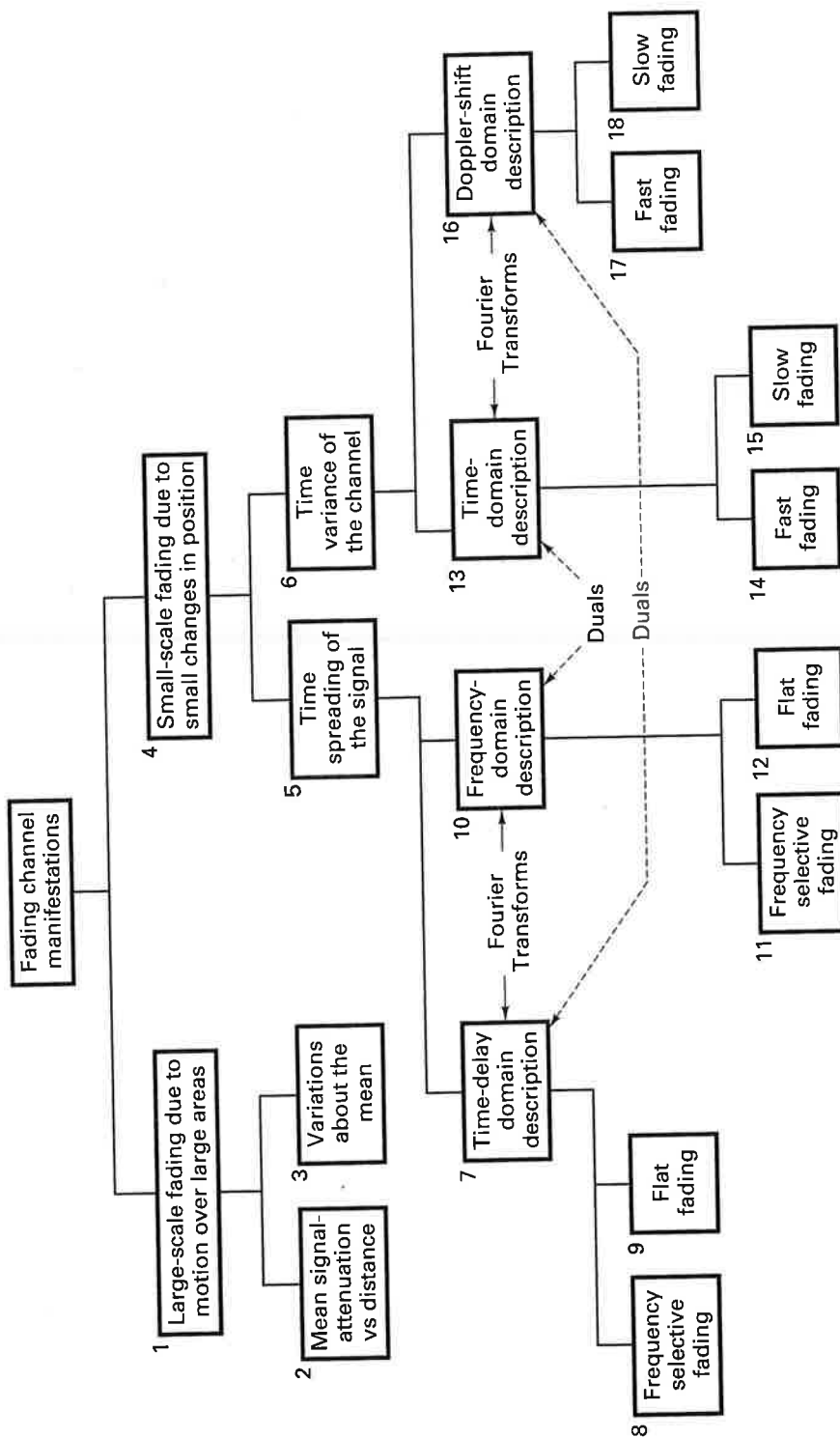


Figure 15.1 Fading channel manifestations.

Diffraction is a phenomenon that accounts for RF energy traveling from transmitter to receiver without a line-of-sight path between the two. It is often termed *shadowing* because the diffracted field can reach the receiver even when shadowed by an impenetrable obstruction.

- *Scattering* occurs when a radio wave impinges on either a large, rough surface or any surface whose dimensions are on the order of λ or less, causing the energy to be spread out (scattered) or reflected in all directions. In an urban environment, typical signal obstructions yielding scattering include lampposts, street signs, and foliage. The name *scatterer* applies to any obstruction in the propagation path that causes a signal to be reflected or scattered.

Figure 15.1 may serve as a table of contents for the sections that follow. The two manifestations of small-scale fading, signal time-spreading (signal dispersion) and the time-variant nature of the channel, will be examined in two domains: time and frequency, as indicated in Figure 15.1, blocks 7, 10, 13, and 16. For signal dispersion, the fading degradation types are categorized as being frequency-selective or frequency-nonselective (flat), as listed in blocks 8, 9, 11, and 12. For the time-variant manifestation, the fading degradation types are categorized as fast fading or slow fading, as listed in blocks 14, 15, 17, and 18. The labels indicating Fourier transforms and duals will be explained later.

Figure 15.2 is a convenient pictorial (not a precise graphical representation) showing the various contributions that must be considered when estimating path loss for link budget analysis in a mobile radio application [2]: (1) mean path loss as a function of distance, due to large-scale fading, (2) near-worst-case variations about the mean path loss or large-scale fading margin (typically 6–10 dB), and (3) near-worst-case Rayleigh or small-scale fading margin (typically 20–30 dB). In Figure 15.2, the annotations “ $\approx 1\text{--}2\%$ ” indicate a suggested area (probability) under the tail of each pdf as a design goal. Hence, the amount of margin indicated is intended to provide adequate received signal power for approximately 98 to 99 percent of each type of fading variation (large- and small-scale).

Using complex notation, a transmitted signal is written as:

$$s(t) = \text{Re}\{g(t)e^{j2\pi f_c t}\} \quad (15.2)$$

where $\text{Re}\{\cdot\}$ denotes the real part of $\{\cdot\}$, and f_c is the carrier frequency. The baseband waveform $g(t)$ is called the complex envelope of $s(t)$ (see Section 4.6), and can be expressed as

$$g(t) = |g(t)|e^{j\phi(t)} = R(t)e^{j\phi(t)} \quad (15.3)$$

where $R(t) = |g(t)|$ is the envelope magnitude, and $\phi(t)$ is its phase. For a purely phase- or frequency-modulated signal, $R(t)$ will be constant, and in general, will vary slowly compared to $t = 1/f_c$.

In a fading environment, $g(t)$ will be modified by a complex dimensionless multiplicative factor $\alpha(t)e^{-j\theta(t)}$. (Later, we show this derivation.) The *modified* baseband waveform can be written as $\alpha(t)e^{-j\theta(t)}g(t)$; but for now, let us examine the magnitude, $\alpha(t)R(t)$ of this envelope, which can be expressed in terms of three positive terms, as follows [3]:

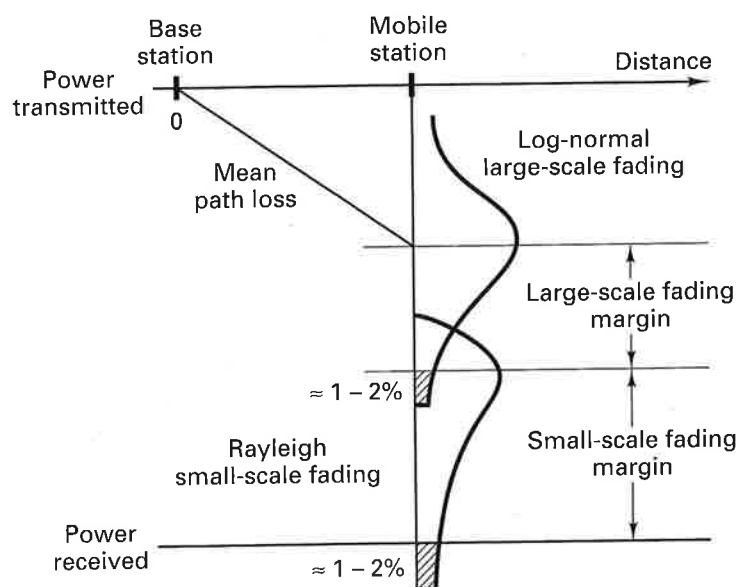


Figure 15.2 Link-budget considerations for a fading channel.
Source: Greenwood, D. and Hanzo, L., "Characterization of Mobile Radio Channels," *Mobile Radio Communications*, edited by R. Steele, Chapter 2, Pentech Press, London, 1994.

$$\alpha(t) R(t) = m(t) \times r_0(t) \times R(t) \quad (15.4)$$

where $m(t)$ is called the *large-scale-fading component* of the envelope, and $r_0(t)$ is called the *small-scale-fading component*. Sometimes, $m(t)$ is referred to as the *local mean* or *log-normal fading*, because generally its measured values can be statistically described by a log-normal pdf; or equivalently, when measured in decibels, $m(t)$ has a Gaussian pdf. Furthermore, $r_0(t)$ is sometimes referred to as *multipath* or *Rayleigh fading*. For the case of a mobile radio, Figure 15.3 illustrates the relationship between $\alpha(t)$ and $m(t)$. In this figure, we consider that an *unmodulated* carrier wave is being transmitted, which in the context of Equation (15.4) means that for all time, $R(t) = 1$. Figure 15.3a is a representative plot of signal power received versus antenna displacement (typically in units of wavelength). The signal power received is of course a function of the multiplicative factor $\alpha(t)$. Small-scale fading superimposed on large-scale fading can be readily identified. The typical antenna displacement between adjacent signal-strength nulls due to small-scale fading is approximately a half wavelength. In Figure 15.3b, the large-scale fading or local mean $m(t)$ has been removed in order to view the small-scale fading $r_0(t)$, referred to some average constant power. Recall that $m(t)$ can generally be evaluated by averaging the received envelope over 10 to 30 wavelengths. The log-normal fading is a relatively slow varying function of position, while the Rayleigh fading is a relatively fast varying function of position. Note that for an application involving motion, such as the case of a radio in a moving vehicle, a function of position is