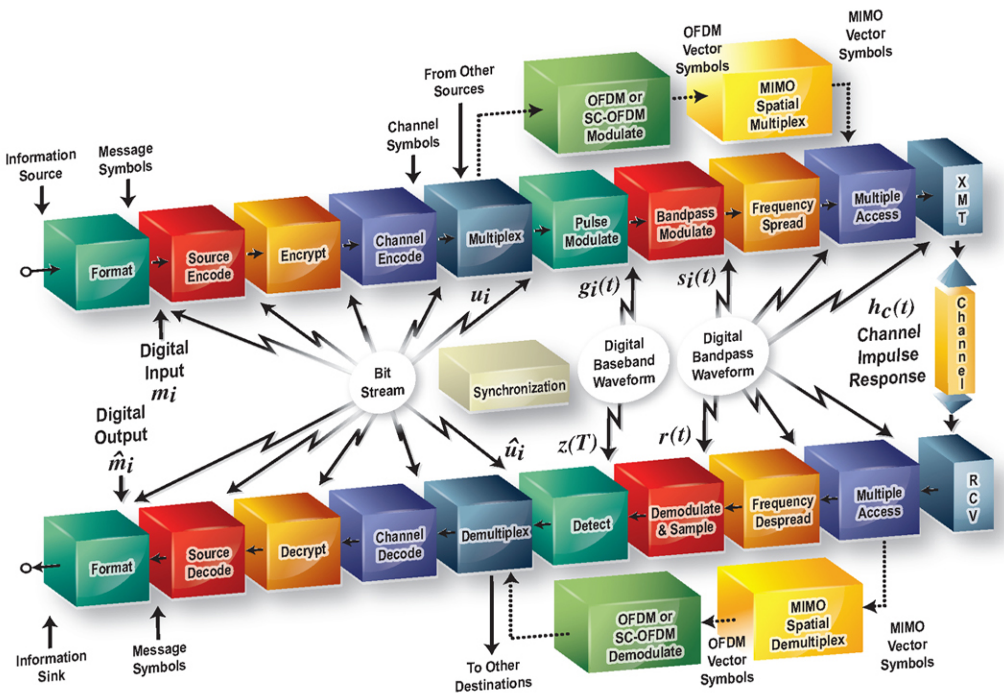
THIRD EDITION

Digital Communications

Fundamentals and Applications



Bernard Sklar *and* fred harris

DIGITAL COMMUNICATIONS

Fundamentals and Applications

Third Edition

Bernard Sklar

fred harris

Pearson

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/

Library of Congress Control Number: 2020937221

ISBN-13: 978-0-13-458856-8
ISBN-10: 0-13-458856-8

ScoutAutomatedPrintCode

Editor-in-Chief: *Mark L. Taub*
Acquisitions Editor: *Malobika Chakraborty*
Managing Editor: *Sandra Schroeder*
Senior Project Editor: *Lori Lyons*
Production Manager: *Aswini Kumar/codeMantra*
Copy Editor: *Kitty Wilson*
Indexer: *Cheryl Ann Lenser*
Proofreader: *Betty Pessagno*
Cover Designer: *Chuti Prasertsith*
Composition/Graphics: *codeMantra*

To those from whom we learned,
To those we taught, and
To those we love.

This page intentionally left blank

Contents at a Glance

THE FOLLOWING ELEMENTS WILL BE ONLINE ONLY

Contents

5 COMMUNICATIONS LINK ANALYSIS 235

6 CHANNEL CODING: PART 1: WAVEFORM CODES AND BLOCK CODES 297

8 CHANNEL CODING: PART 3: TURBO CODES AND LOW-DENSITY PARITY CHECK (LDPC) CODES 471

9 MODULATION AND CODING TRADE-OFFS 549

THE FOLLOWING ELEMENTS WILL BE ONLINE ONLY

These online elements can be found at informit.com/title/9780134588568

Preface

This third edition of *Digital Communications: Fundamentals and Applications* is an updated version of the original and second edition publications. The following key features have been updated and changed:

- We added a chapter dealing with orthogonal frequency-division multiplexing (OFDM). OFDM utilizes closely spaced orthogonal subcarriers. Data are partitioned into groups, such that each group is assigned a subcarrier, which is then modulated in a conventional way. The technique allows for elegant mitigation of intersymbol interference and intercarrier interference. A cyclic prefix is used to "trick" the channel into performing circular convolution instead of linear convolution, which helps maintain orthogonality. The primary advantage of OFDM is its ability to cope with severe channel multipath conditions, such as frequency-selective fading, without requiring complex equalization filters.
- We also added a chapter on multiple input, multiple output (MIMO) systems. The uniqueness of MIMO stems from time being complemented with the spatial dimension obtained by using several antennas (at the transmitter and receiver). We focus on spatial multiplexing and space-time coding to examine how MIMO can improve BER or increase capacity or both—without expending additional power or bandwidth. It involves exploiting any multipath channel conditions. We examine available trade-offs between capacity and robustness and consider multi-user (MU-MIMO) systems where multiple independent users can simultaneously access MIMO base stations.

- In order to keep the book in single-volume format while adding the new OFDM and MIMO chapters, we chose to provide some of the material online. Therefore, in this third edition, the material on encryption and decryption has become Chapter 17, which can now be accessed on the companion website, as described later in the section "Additional Book Resources." Also available online are Appendixes A through G. These online elements can be found at informit.com/title/9780134588568.

This third edition is intended to provide comprehensive coverage of digital communication systems for senior-level undergraduates, first-year graduate students, and practicing engineers. Although the emphasis is on digital communications, necessary analog fundamentals are included because analog waveforms are used for the radio transmission of digital signals. The key feature of a digital communication system is that it deals with a finite set of discrete messages, in contrast to an analog communication system, in which messages are defined on a continuum. The objective at the receiver of a digital system is not to reproduce a waveform with precision; it is instead to determine from a noise-perturbed signal which of the finite set of waveforms had been sent by the transmitter. In fulfillment of this objective, there has arisen an impressive assortment of signal processing techniques.

This book describes these signal processing techniques in the context of a unified structure, a block diagram that appears at the beginning of each chapter. In each chapter, applicable portions of the diagram are emphasized. One of the main purposes of this book is to ensure awareness of the "big picture," even while delving into the details. Signals and key processing steps are traced from the information source through the transmitter, channel, receiver, and, ultimately, to the information sink. Signal transformations are organized according to functional classes: formatting and source coding, baseband signaling, bandpass signaling, equalization, channel coding, multiplexing and multiple access, spreading, and synchronization. Throughout the book, emphasis is placed on system goals and the need to trade off basic system parameters such as signal-to-noise ratio, probability of error, and bandwidth expenditure.

ORGANIZATION OF THE BOOK

Chapter 1 introduces the overall digital communication system and the basic signal transformations that are highlighted in subsequent chapters. Some basic ideas of random variables and the *additive white Gaussian noise* (AWGN) model are reviewed. Also, the relationship between power spectral density and autocorrelation and the basics of signal transmission through linear systems are established. **Chapter 2** covers the signal processing step known as *formatting,* which is used to render an information signal compatible with a digital system. **Chapter 3** emphasizes *baseband signaling,* the detection of signals in Gaussian noise, and receiver optimization. **Chapter 4** deals with *bandpass signaling* and its associated modulation and demodulation/detection techniques. **Chapter 5** deals with *link analysis*, an important subject for providing overall system insight; it considers some subtleties that are often missed. Chapters 6, 7, and 8 deal with *channel coding*—a cost-effective way of providing a variety of system performance trade-offs. **Chapter 6** emphasizes *linear block codes*, **Chapter 7** deals with *convolutional codes* and *Reed–Solomon codes*, and **Chapter 8** deals with *turbo codes* and *low-density parity-check* (LDPC) codes.

Chapter 9 considers various modulation/coding system *trade-offs* related to probability of bit-error performance, bandwidth efficiency, and signal-to-noise ratio. It also treats the area of coded modulation, covering topics such as *trellis-coded modulation.* **Chapter 10** deals with *synchronization* for digital systems. It covers phase-locked loop implementation for achieving carrier synchronization. It covers bit synchronization, frame synchronization, and network synchronization, and it introduces some ways of performing synchronization using digital methods.

Chapter 11 treats *multiplexing* and *multiple access*. It explores techniques that are available for utilizing the communication resource efficiently. **Chapter 12** introduces *spread-spectrum* techniques and their application in areas such as multiple access, ranging, and interference rejection. This technology is important for both military and commercial applications. **Chapter 13** deals with *source coding*, which is a special class of data formatting. Both formatting and source coding involve digitization of data; the main difference between them is that source coding additionally involves reducing data redundancy. Rather than consider source coding immediately after formatting, we purposely treat it in a later chapter to avoid interrupting the presentation flow of the basic processing steps. **Chapter 14** deals with *fading channels.* Here, we deal with applications such as mobile radios, where characterization of the channel is much more involved than that of a nonfading one. The design of a communication system that can withstand the degradation effects of fading can be much more challenging than the design of its nonfading counterpart. In Chapter 14, we describe a variety of techniques that can mitigate the effects of fading, and we show some successful designs that have been implemented. **Chapters 15 and 16**, new additions to this third edition dealing with OFDM and MIMO, respectively, were summarized at the beginning of this preface. **Chapter 17** (which is available online at informit.com/title/9780134588568) covers basic *encryption/decryption* ideas. It includes some classical concepts, as well as a class of systems called *public key cryptosystems*, and the email encryption software known as *Pretty Good Privacy* (PGP).

Appendixes A–G are available online at informit.com/title/9780134588568. We assume the reader is familiar with Fourier methods and convolution. **Appendix A** reviews these techniques, emphasizing properties that are particularly useful in the study of communication theory. We also assume the reader has a knowledge of basic probability and some familiarity with random variables. **Appendix B** builds on these disciplines for a short treatment on statistical decision theory, with emphasis on hypothesis testing—which is very important in the understanding of detection theory. **Appendix C** shows the inputs to a bank of N correlators representing a white Gaussian noise process. **Appendix D** shows a list of often-used identities. **Appendix E** serves as a short tutorial on s-domain, z-domain, and digital filtering. **Appendix F** reviews the inverse and forward discrete Fourier transforms of finite-length sequences and their relationship to continuous and sampled signals and to continuous and sampled spectra. Understanding these relationships can provide valuable insight into the OFDM process. **Appendix G** provides a List of Symbols used throughout the book.

If the book is used for a two-term course, a simple partitioning is suggested: The first eight chapters can be taught in the first term, and the last eight chapters in the second term. If the book is used for a one-term introductory course, it is suggested that the course material be selected from the following chapters: 1, 2, 3, 4, 5, 6, 7, 9, 10, and 12.

ADDITIONAL BOOK RESOURCES

This third edition is supported by three Internet resources that have been set up to provide access to ancillary material, bonus material, and errata sheets for instructors as well as students. The first site is for instructors only, and the other two are for all readers:

- The **Pearson Instructor Resource Center (IRC)** provides ancillary information for instructors, such as solution manual, exam problems, and class exercises. To access a Solutions Manual for this book, go to **https://www.pearson.com/us/higher-education/subject-catalog/download-instructor-resources.html** to register, or to sign in if you already have an account.
- The **Pearson InformIT** page (informit.com/title/9780134588568) houses Chapter 17 on encryption and decryption, Appendixes A though G, and ancillary material that may be deemed useful during the life of the book
- **MATLAB Central File Exchange** is an online community hosted by MathWorks. MATLAB code for the book is available on the File Exchange for instructors and students alike and is particularly helpful in solving some of the end-of-chapter problems. See https://www.mathworks.com/matlabcentral/profile/authors/216378-bernard-sklar.

Register your copy of *Digital Communications: Fundamentals and Applications* at informit.com for convenient access to updates and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN 9780134588568 and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

Acknowledgments

It is difficult to write a technical book without contributions from many others. Bernard Sklar has received an abundance of such assistance while creating the earlier editions, for which he is deeply grateful. For their generous help, he thanks Dr. Andrew Viterbi, Dr. Marv Simon, Dr. Bill Lindsey, Dr. Jim Omura, Dr. Chuck Wheatley, Dr. Adam Lender, Dr. Joe Odenwalder, Dr. Bob Price, Dr. Bob Bogusch, Dr. Maurice King, Don Martin, Ned Feldman, Dr. Maury Schiff, Dr. Ed Tiedeman, Dr. Dariush Divsalar, Dr. Tom Stanley, Phil Kossin, and Professors Dan Bukofzer, Wayne Stark, Larry Milstein, Ray Pickholtz, Daniel Costello, and Ted Rappaport.

He offers special thanks for technical clarifications from his son, Dean Sklar, who took on the difficult role of being his father's chief critic and "devil's advocate." He is indebted to Professor Bob Stewart of the University of Strathclyde, Glasgow, for his contribution of online Appendix E. He also wants to thank his original publisher at Prentice Hall, Bernard Goodwin, for believing in him. He is extremely grateful to his wife, Gwen, for her encouragement, devotion, and valuable advice. Finally, he is especially grateful for the contributions of his Pearson editors, Malobika Chakraborty, Lori Lyons, and Kitty Wilson, and Aswini Kumar, for their tireless and dedicated work to bring this third edition to fruition.

Professor fred harris recalls with pleasure the many years that he and Dr. Sklar have worked together to present insightful lectures on communication and signal processing tasks and have shared and polished fresh perspectives on material developed for presentations. He also identifies former students and now colleagues, Elettra Venosa, Xiaofei Chen, and Richard Bell, for supporting activities in the same areas. He thanks Anthony Constantinides, Bob McGwier, and Doug Jager, who have posed good questions and contributed nourishing environments that enriched our collective professional and personal lives. He offers thanks to John Proakis, a special colleague who has reviewed the synchronization chapter and who has nurtured many of us with his rich body of publications.

Bernard Sklar

Tarzana, California

fred harris

San Diego, California

About the Authors

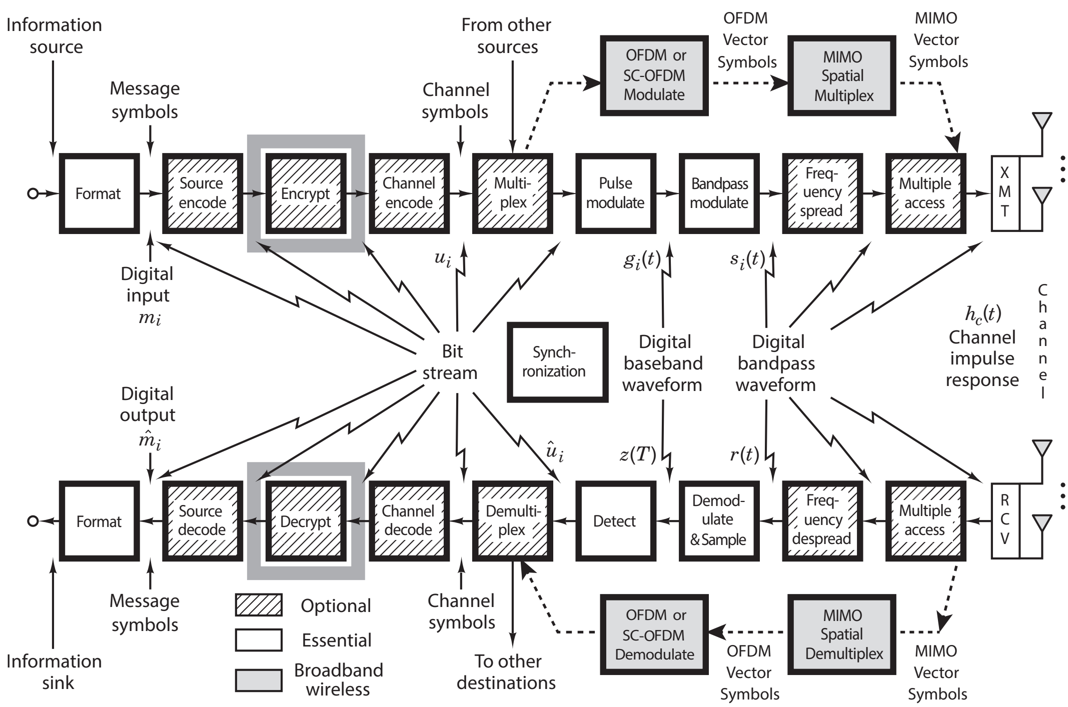
DR. BERNARD SKLAR has over 40 years of experience in technical design and management positions at Republic Aviation, Hughes Aircraft, Litton Industries, and The Aerospace Corporation, where he helped develop the MILSTAR satellite system. He is now head of advanced systems at Communications Engineering Services, a consulting company he founded in 1984. He has taught engineering courses at several universities, including UCLA and USC, and has trained professional engineers worldwide.

DR. FREDRIC J. HARRIS is a professor of electrical engineering and the CUBIC signal processing chair at San Diego State University and an internationally renowned expert on DSP and communication systems. He is also the co-inventor of the Blackman–Harris filter. He has extensively published many technical papers, the most famous being the seminal 1978 paper "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform." He is also the author of the textbook *Multi-rate Signal Processing for Communication Systems* and the source coding chapter in the previous edition of this book.

This page intentionally left blank

CHAPTER 17

Encryption and Decryption



17.1 MODELS, GOALS, AND EARLY CIPHER SYSTEMS

17.1.1 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 17.1 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:

$$D_K(C) = E_K^{-1}\left[E_K(M)\right] = M \tag{17.1}$$

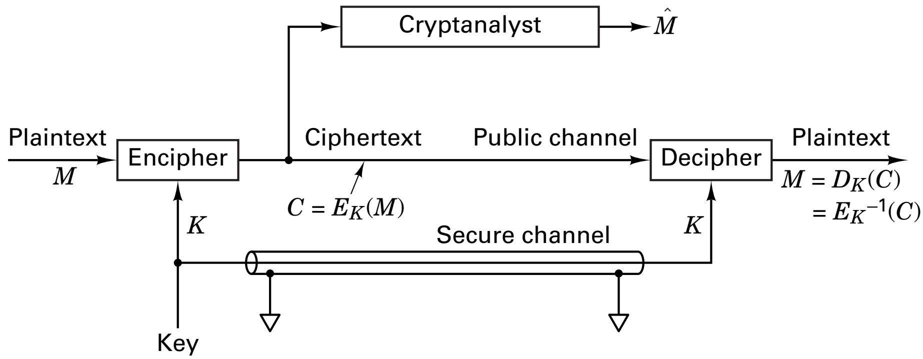


Figure 17.1 Model of a cryptographic channel.

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 ith 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 parity 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.

17.1.2 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 17.2.1), 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.

17.1.3 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 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].

17.1.4 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 17.2a 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 17.2b, 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	IJ	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

(b)

Figure 17.2 (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
Ciphertext:	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 17.3, 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
Ciphertext:	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
Ciphertext:	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 17.3 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

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 in 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.

17.2 THE SECRECY OF A CIPHER SYSTEM

17.2.1 Perfect Secrecy

Consider a cipher system with a finite message space $\{M\} = M_0, M_1, \ldots, M_{N-1}$ and a finite ciphertext space $\{C\} = C_0, C_1, \ldots, C_{U-1}$. 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:

$$P(M_i|C_j) = P(M_i) \tag{17.2}$$

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,

$$P(C_j|M_i) = P(C_j) \tag{17.3}$$

The schematic in Figure 17.4 illustrates an example of perfect secrecy. In this example, $\{M\} = M_0, M_1, M_2, M_3$, $\{C\} = C_0, C_1, C_2, C_3$, $\{K\} = K_0, K_1, K_2, K_3$, $N = U = 4$,

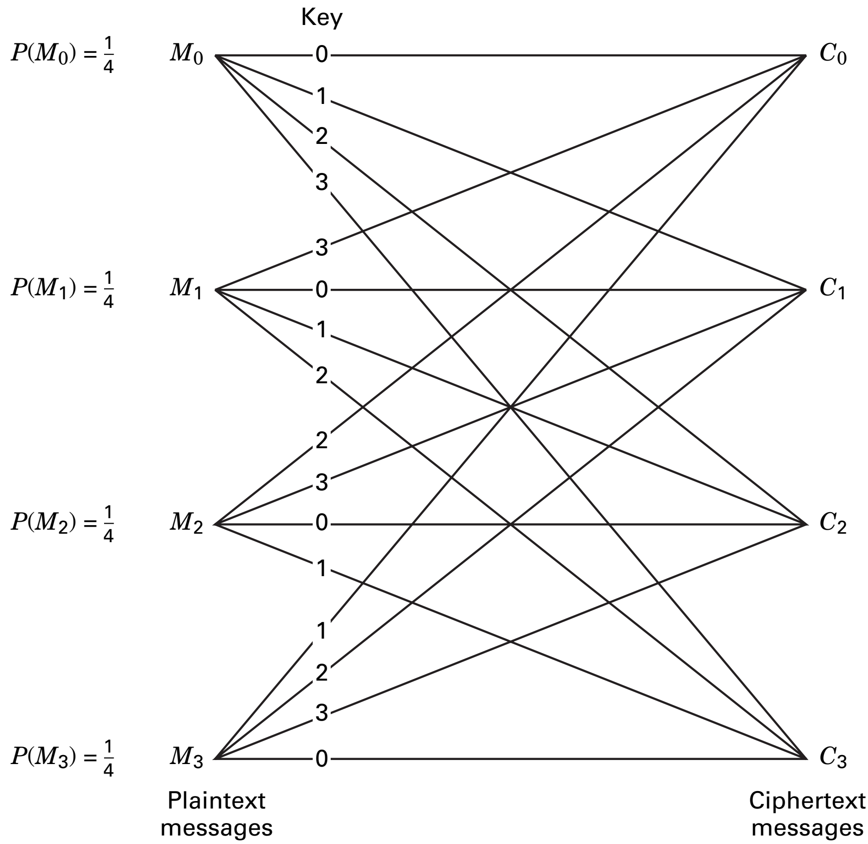


Figure 17.4 Example of perfect secrecy.

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

$$\begin{aligned} C_s &= T_{K_j}(M_i) \\ s &= (i + j) \text{ modulo-}N \end{aligned} \tag{17.4}$$

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 17.1 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 17.1.4) 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 17.3, 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 17.5, yielding only one key ($K = 10$) that produces the meaningful message: WHERE ARE THE HEROES OF YESTERYEAR (The spaces have been added.)

Example 17.2 Perfect Secrecy

We can modify the key space of Example 17.1 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 $k_1, k_2, \ldots, k_{29}$, where each k_i is a random integer in the range (1, 25) dictating the shift used for the ith character; thus there are a total of $(25)^{29}$ different key sequences. Then the 29-character ciphertext in Example 17.1 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	0	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 17.5 Example of breaking a cipher system when the key space is smaller than the message space.

17.2.2 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:

$$H(X) = -\sum_X P(X)\log_2 P(X) = \sum_X P(X)\log_2 \frac{1}{P(X)} \tag{17.5}$$

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

$$H(X|Y) = -\sum_{X,Y} P(X,Y)\log_2 P(X|Y) \tag{17.6}$$

$$= \sum_Y P(Y) \sum_X P(X|Y)\log_2 \frac{1}{P(X|Y)}$$

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 17.3 Entropy and Equivocation

Consider a sample message set consisting of eight equally likely messages $\{X\} = X_1, X_2, \ldots, X8$.

(a) Find the entropy associated with a message from the set $\{X\}$.
(b) 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

(a) $P(X) = \frac{1}{8}$
$H(X) = 8[(\frac{1}{8})\log_2 8] = 3$ bits/message
(b) $P(Y) = \frac{1}{2}$. 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 (17.6), we obtain

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

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

17.2.3 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

$$r = \frac{H(X)}{N} \tag{17.7}$$

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

$$r' = \log_2 L \tag{17.8}$$

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

$$D = r' - r \tag{17.9}$$

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.

17.2.4 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

$$\text{number of meaningful messages} = 2^{rN} \tag{17.10}$$

$$\text{number of meaningless messages} = 2^{r'N} - 2^{rN} \tag{17.11}$$

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

$$P(M_1) = \frac{1}{2^{rN}} = 2^{-rN} \qquad M_1 \text{ meaningful} \tag{17.12}$$

$$P(M_2) = 0 \qquad M_2 \text{ meaningless} \tag{17.13}$$

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,

$$P(K) = \frac{1}{2^{H(K)}} = 2^{-H(K)} \tag{17.14}$$

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,

$$C_i = E_{K_i}(M_i) = E_{K_j}(M_i) = E_{K_j}(M_j) \tag{17.15}$$

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,

$$P(F) = \frac{2^{rN}}{2^{r'N}} = 2^{(r-r')N} = 2^{-DN} \tag{17.16}$$

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

$$\bar{F} = [2^{H(K)} - 1]P(F) = [2^{H(K)} - 1]2^{-DN} \approx 2^{H(K)-DN} \tag{17.17}$$

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

$$\log_2 \bar{F} = H(K) - DN = 0 \tag{17.18}$$

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

$$N = \frac{H(K)}{D} \tag{17.19}$$

We can see from Equation (17.17) 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 (17.19) 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 17.4 Unicity Distance

Calculate the unicity distance for a written English encryption system, where the key is given by the sequence $k_1, k_2, \ldots, k_{29}$, where each k_i is a random integer in the range (1, 25) dictating the shift number (Figure 17.3) for the ith 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 (17.5), (17.8), and (17.19) we have

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

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

Assumed true rate for English: $r = 1.5$ bits/character

Redundancy: $D = r' - r = 3.2$ bits/character

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

In Example 17.2, 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.

17.3 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 μs, 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.

17.3.1 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 $M = M_0, M_1, \ldots$ into a new message sequence $Y = Y_0, Y_1, \ldots$ according to the relationship

$$Y_n = \sum_{i=0}^{s-1} M_{n+i} \quad \text{modulo-26} \tag{17.20}$$

where each character in the sequence is regarded as an integer modulo-26, s is some chosen integer, and $n = 0, 1, 2, \ldots$. 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.

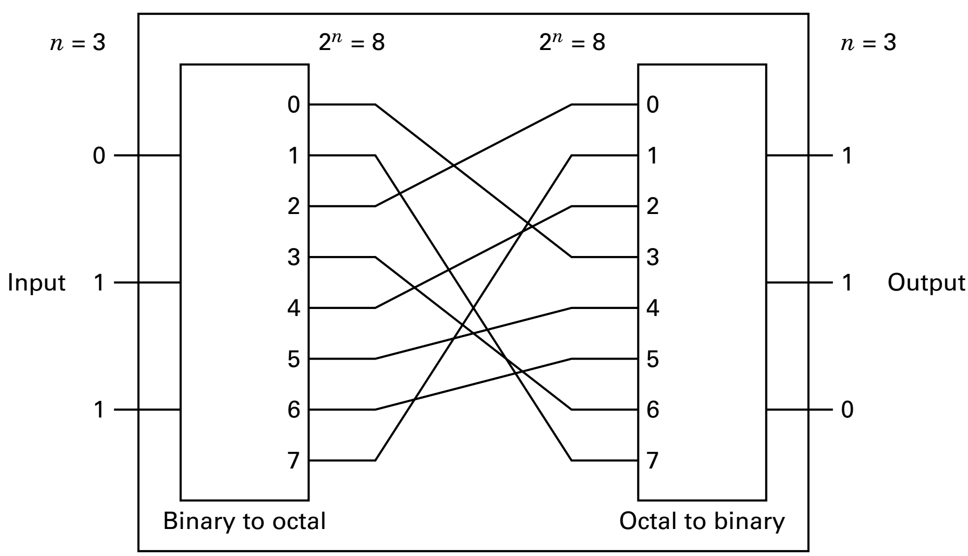
17.3.2 Substitution

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 17.6 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 17.6). The set of 2^n characters is 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 17.6 performs a *nonlinear transformation,* we need only use the superposition theorem stated below as a test. Let

$$\begin{aligned} C &= Ta + Tb \\ C' &= T(a + b) \end{aligned} \tag{17.21}$$



Input	000	001	010	011	100	101	110	111
Output	011	111	000	110	010	100	101	001

Figure 17.6 Substitution box.

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

$$\text{If } T \text{ is linear: } C = C' \text{ for all inputs}$$

$$\text{If } T \text{ is nonlinear: } C \neq C'$$

Suppose that $a = 001$ and $b = 010$; then, using T as described in Figure 17.6, 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.

17.3.3 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 17.7 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

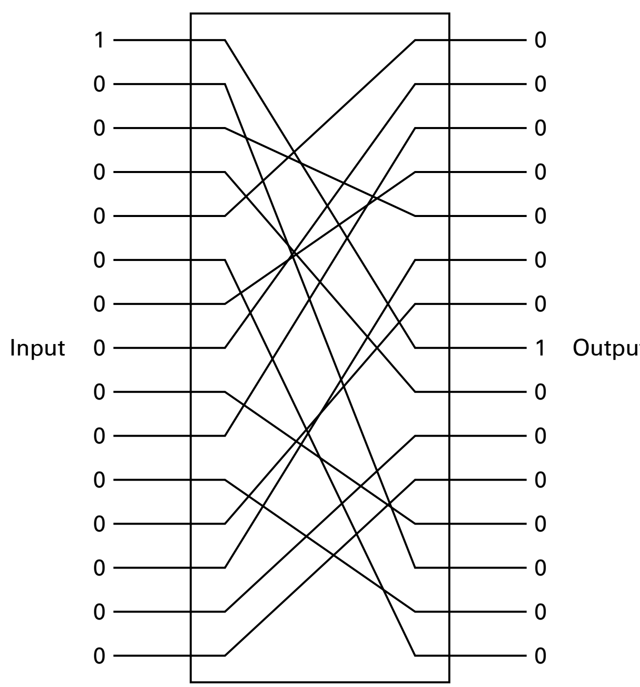


Figure 17.7 Permutation box.

illustrated in Figure 17.7. 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.

17.3.4 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 17.8 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 17.8 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 17.9 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

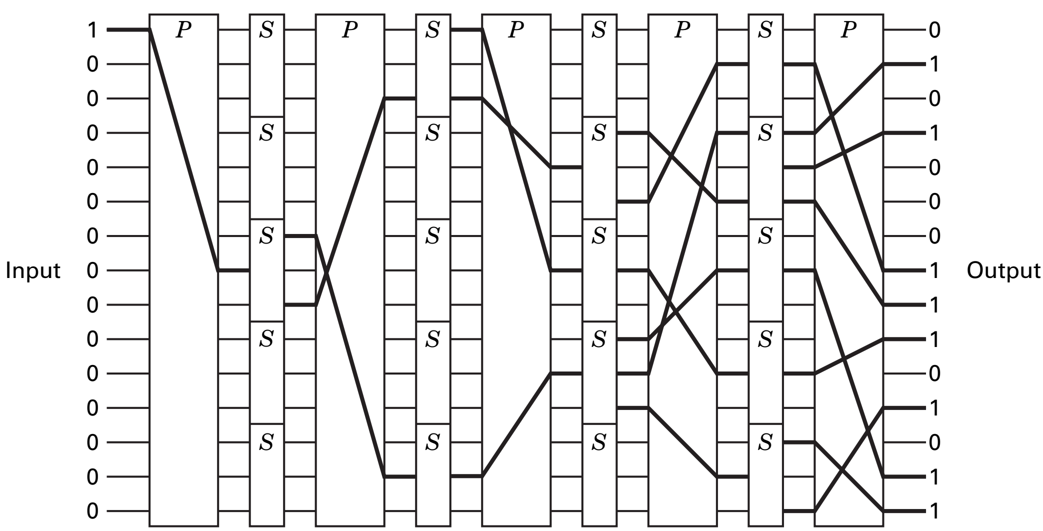
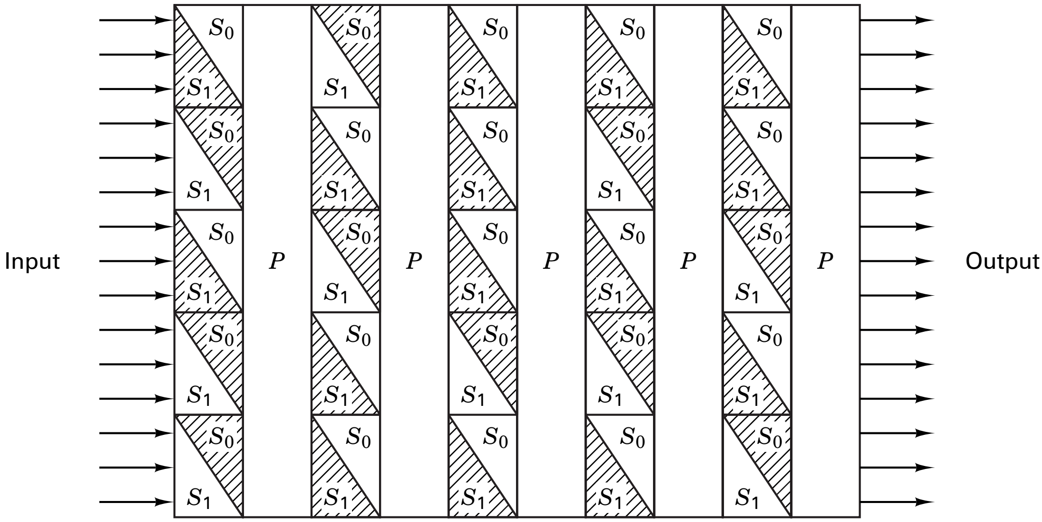


Figure 17.8 Product cipher system.



Shaded boxes correspond to the symbols of the binary key below.

Example of binary key

1 0 1 0 0 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 1 0 1 0

Figure 17.9 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 17.9 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.

17.3.5 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 17.10. An input block of 64 bits, regarded as a plaintext symbol in this alphabet, is replaced with a new ciphertext symbol. Figure 17.11 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 17.1). The IP-table is read from left to right and from top to bottom, so that bits $x_1, x_2, \ldots, x_{64}$ are permuted to $x_{58}, x_{50}, \ldots, x_7$. After this initial permutation, the heart of the encryption algorithm consists of 16 iterations using

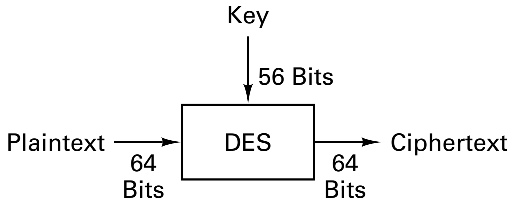


Figure 17.10 Data encryption standard (DES) viewed as a block encryption system.

the standard building block (SBB) shown in Figure 17.12. 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 17.2), 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

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

into

$$(R_{i-1})_E = x_{32}, x_1, x_2, \ldots, x_{32}, x_1 \tag{17.22}$$

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 ith key selection, explained later, and the result is segmented into eight 6-bit blocks

$$B_1, B_2, \ldots, B_8$$

That is,

$$(R_{i-1})_E \oplus K_i = B_1, B_2, \cdots, B_8 \tag{17.23}$$

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 17.3. 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 17.4). 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 $x_1, x_2, \ldots, x_{32}$ are permuted to $x_{16}, x_7, \ldots, x_{25}$. 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

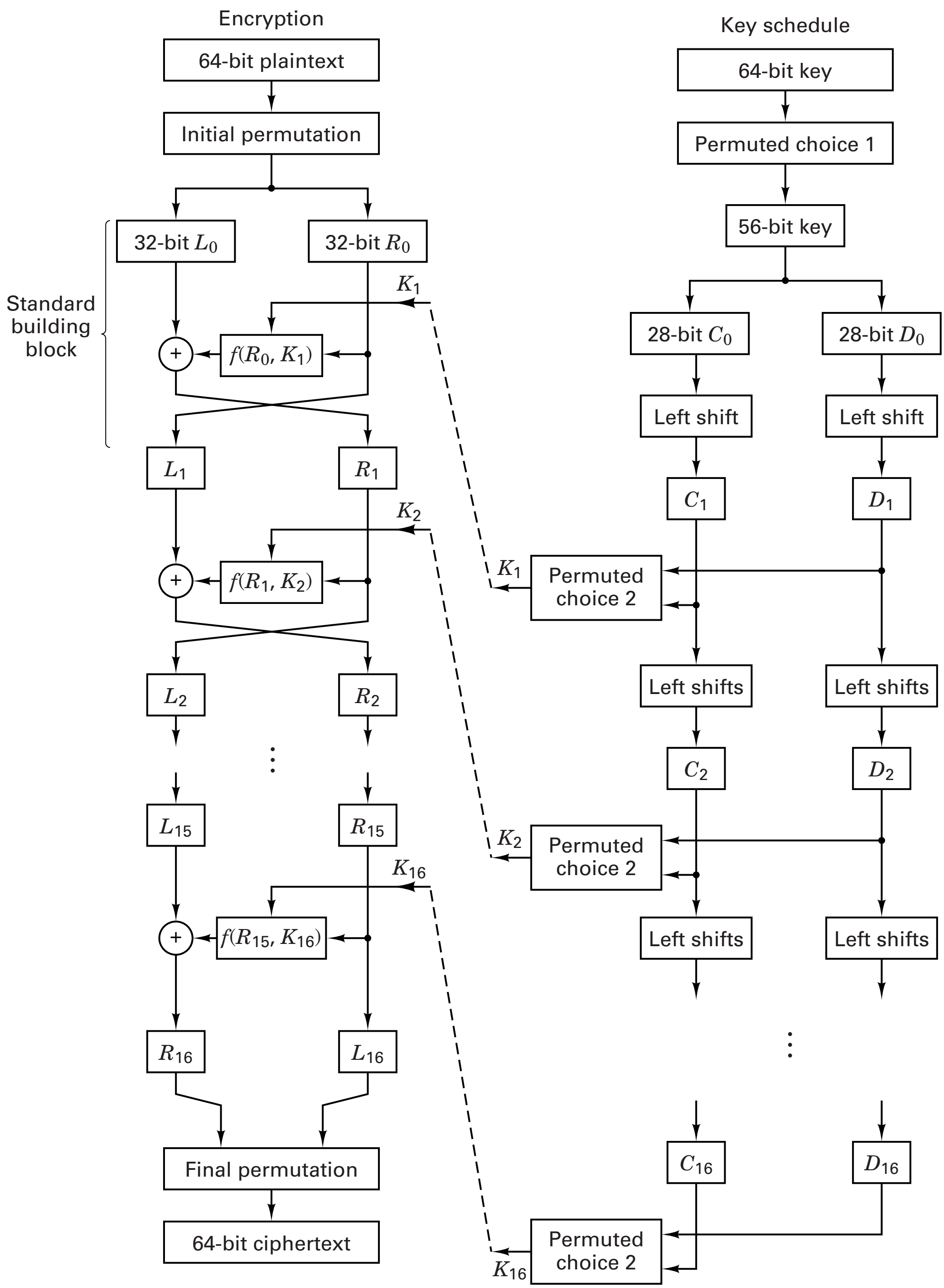


Figure 17.11 Data encryption standard.

TABLE 17.1 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

$$L_i = R_{i-1} \tag{17.24}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \tag{17.25}$$

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 17.5), 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 ith block as $f(L_i, K_i)$, makes the decryption process possible.

17.3.5.1 Key Selection

Key selection also proceeds in 16 iterations, as seen in the key schedule portion of Figure 17.11. 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 17.6. The output of PC-1 is split into two halves, C and D, of 28 bits each. Key selection proceeds in 16 iterations in

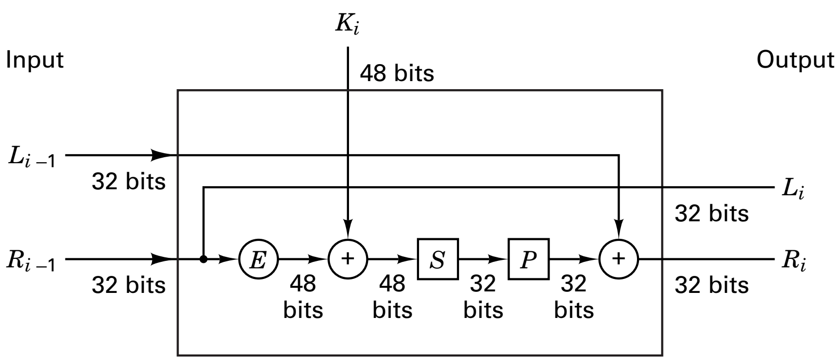


Figure 17.12 Standard building block (SBB).

TABLE 17.2 *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 17.3 *S*-Box Selection Functions

	Column																
Row	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	
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	S_1
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	
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	S_2
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	
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	S_3
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	
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	S_4
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	
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	S_5
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	
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	S_6
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	
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	S_7
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	
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	S_8
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 17.4 *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

$$C_i = \mathrm{LS}_i(C_{i-1}) \quad \text{and} \quad D_i = \mathrm{LS}_i(D_{i-1}) \tag{17.26}$$

where LS_i is a left circular shift by the number of positions shown in Table 17.7. The sequence C_i, D_i is then transposed according to the permuted choice 2 (PC-2) shown in Table 17.8. The result is the key sequence K_i, which is used in the ith iteration of the encryption algorithm.

The DES can be implemented as a block encryption system (see Figure 17.11), 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 17.5 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 17.6 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 17.6.)

17.4 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 17.7 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

TABLE 17.8 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 17.4.2 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.*

17.4.1 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 17.13. For this example, it is convenient to number the stages as shown in Figure 17.13, 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 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.

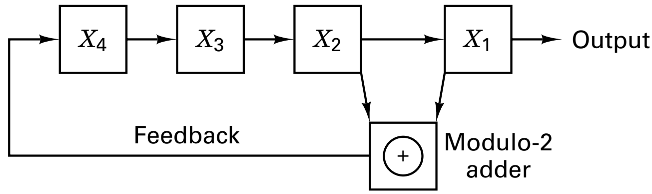


Figure 17.13 Linear feedback shift register example.

17.4.2 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 17.13. 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 17.14. 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 $t_2, t_3, t_4, \ldots$. From the linear structure of the four-stage shift register, we can write

$$g_4 x_4 + g_3 x_3 + g_2 x_2 + g_1 x_1 = x_5 \tag{17.27}$$

where x_5 is the digit fed back to the input and g_i (= 1 or 0) defines the ith 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 17.14:

$$\begin{aligned} g_4(1) + g_3(0) + g_2(0) + g_1(1) &= 1 \\ g_4(1) + g_3(1) + g_2(0) + g_1(0) &= 0 \\ g_4(0) + g_3(1) + g_2(1) + g_1(0) &= 1 \\ g_4(1) + g_3(0) + g_2(1) + g_1(1) &= 0 \end{aligned} \tag{17.28}$$

The solution of Equations (17.28) is $g_1 = 1, g_2 = 1, g_3 = 0, g_4 = 0$, corresponding to the LFSR shown in Figure 17.13. The cryptanalyst has thus learned the connections of

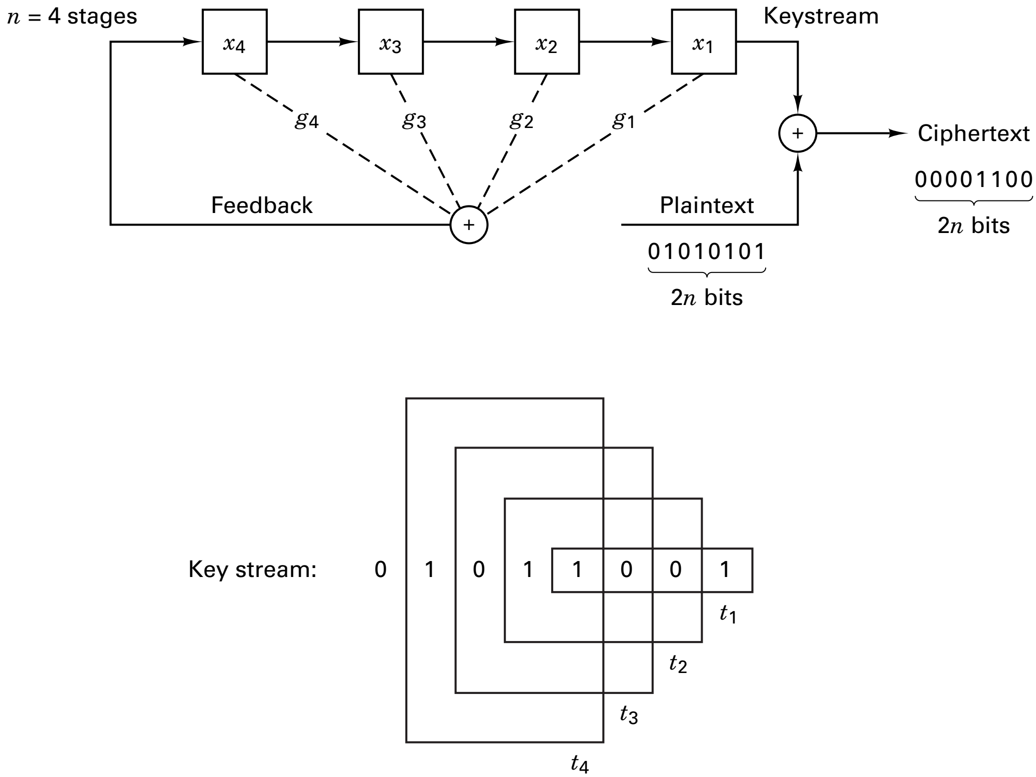


Figure 17.14 Example of vulnerability of a linear feedback shift register.

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 (17.27) as follows:

$$x_{n+1} = \sum_{i=1}^{n} g_i x_i \tag{17.29}$$

We can write Equation (17.29) as the matrix equation

$$\mathbf{x} = \mathbf{X}\mathbf{g} \tag{17.30}$$

where

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

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 **X** are linearly independent; thus **X** is nonsingular (its determinant is nonzero) and has an inverse. Hence,

$$\mathbf{g} = \mathbf{X}^{-1}\,\mathbf{x} \tag{17.31}$$

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-μs operation cycle would require 1 s for the inversion. The weakness of a LFSR is caused by the linearity of Equation (17.31). The use of *nonlinear feedback* in the shift register makes the cryptanalyst's task much more difficult, if not computationally intractable.

17.4.3 Synchronous and Self-Synchronous Stream Encryption Systems

We can categorize stream encryption systems as either *synchronous* of *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 17.15. 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 ith key character, k_i, with the ith message character, m_i. Such synchronous ciphers are generally designed to utilize *confusion* (see Section 17.3.1) 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

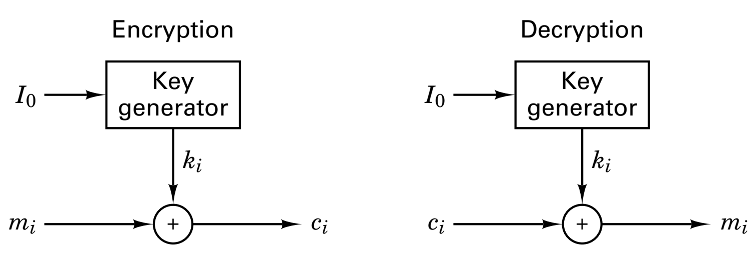


Figure 17.15 Synchronous stream cipher.

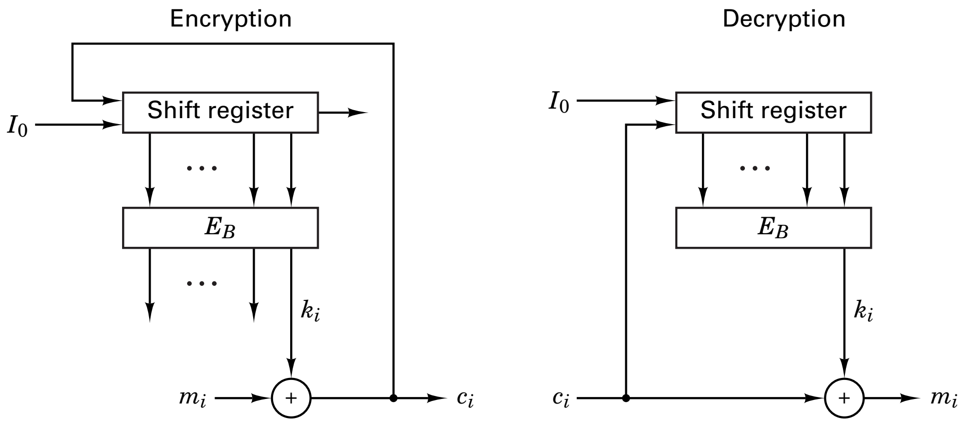


Figure 17.16 Cipher feedback mode.

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

In Section 17.1.4 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 17.16 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.

17.5 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

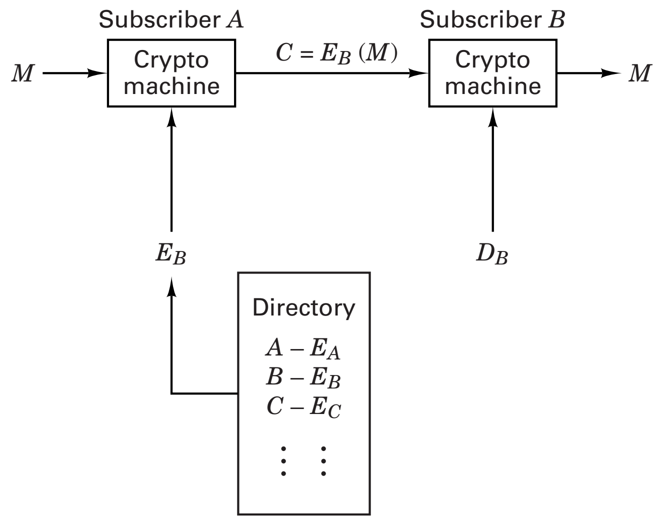


Figure 17.17 Public key cryptosystem.

encryption keys of all the subscribers. Only the decryption keys are kept secret. Figure 17.17 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 17.17, 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 $M = D_B(C)$.

17.5.1 Signature Authentication Using a Public Key Cryptosystem

Figure 17.18 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 $E_A[E_A^{-1}(M)] = M$.

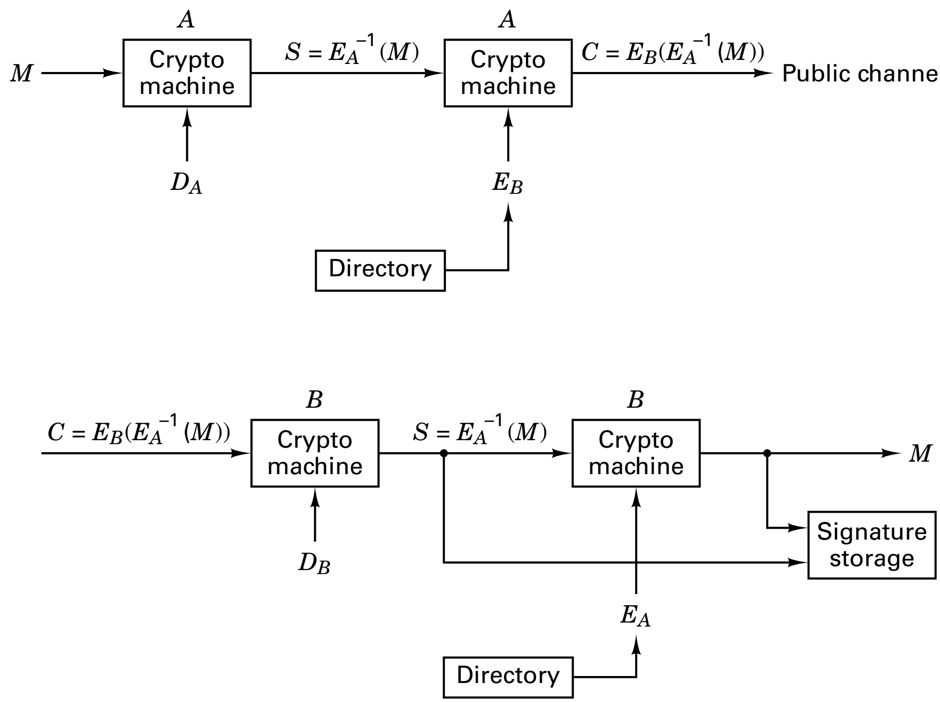


Figure 17.18 Signature authentication using a public key cryptosystem.

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.

17.5.2 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 17.5.5, when we discuss the Merkle–Hellman scheme.

17.5.3 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:

$$\begin{aligned} &\text{Encryption:} \quad C = E(M) = (M)^e \text{ modulo-}n \\ &\text{Decryption:} \quad M = D(C) = (C)^d \text{ modulo-}n \end{aligned} \tag{17.32}$$

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:

$$n = pq \tag{17.33}$$

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

$$\phi(n) = (p - 1)(q - 1) \tag{17.34}$$

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,

$$X = X^{k\phi(n)+1} \text{ modulo-}n \tag{17.35}$$

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

$$\gcd[\phi(n), d] = 1 \tag{17.36}$$

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

$$ed \text{ modulo-}\phi(n) = 1 \tag{17.37}$$

which, from Equation (17.35), is tantamount to choosing e and d to satisfy

$$X = X^{ed} \text{ modulo-}n \tag{17.38}$$

Therefore,

$$E[D(X)] = D[E(X)] = X \tag{17.39}$$

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 (17.37). 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].

17.5.3.1 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):

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

$$157e \text{ modulo } 2688 = 1$$

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:

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

17.5.3.2 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 $x_0, x_1, x_2, \ldots$, where $x_0 = \phi(n)$, $x_1 = d$, and $x_{i+1} = x_{i-1}$ 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 modulo-x_0. If b_{k-1} is a negative number, the solution is $b_{k-1} + \phi(n)$.

Example 17.5 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

$$e = b_3 = 17$$

17.5.4 The Knapsack Problem

The classic knapsack problem is illustrated in Figure 17.19. 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)

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

The data vector is an n-tuple of binary symbols

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

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

$$\begin{aligned} S &= \sum_{i=1}^{n} a_i x_i \quad \text{where } x_i = 0, 1 \\ &= \mathbf{ax} \end{aligned} \tag{17.40}$$

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

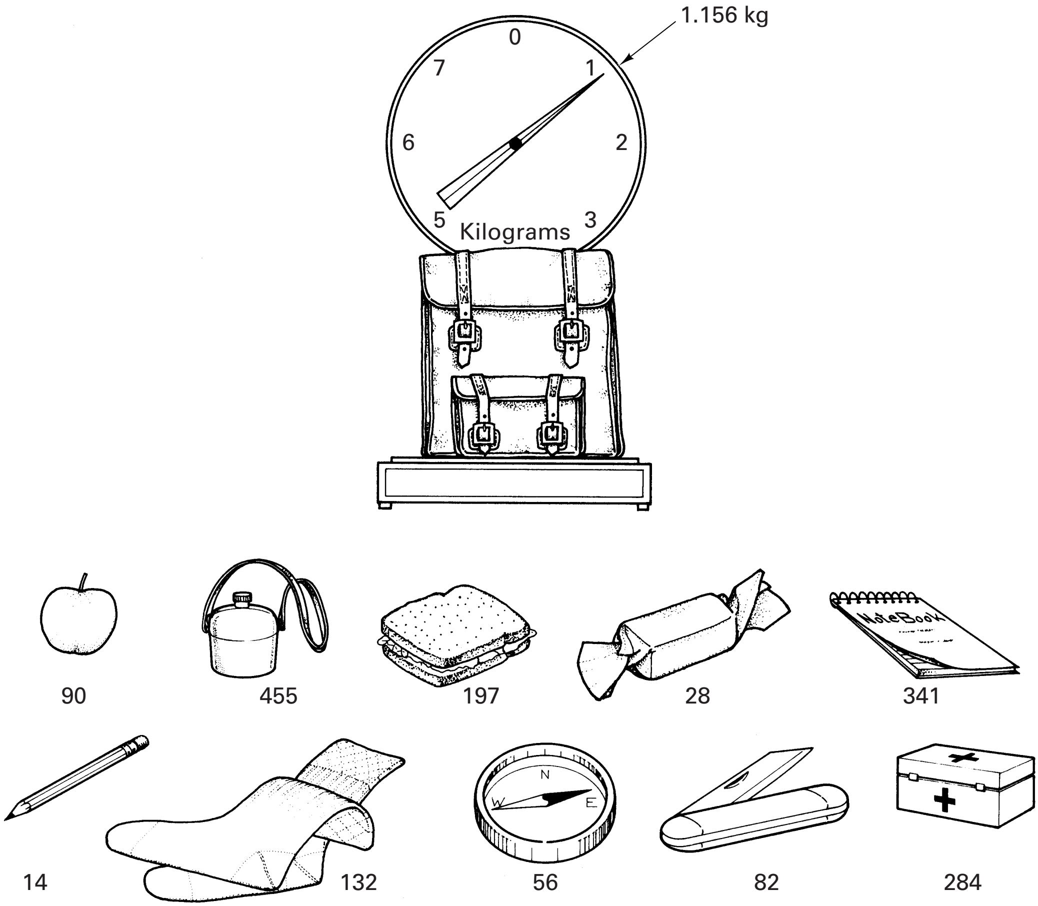


Figure 17.19 Knapsack problem.

Example 17.6 Knapsack Example

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

$$a_i > \sum_{j=1}^{i-1} a_j \qquad i = 2, 3, \ldots, n \tag{17.41}$$

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

$$x_i = \begin{cases} 1 & \text{if } S - \sum_{j=i+1}^{n} x_j a_j \geq a_i \\ 0 & \text{otherwise} \end{cases} \tag{17.42}$$

where $i = n - 1, n - 2, \ldots, 1$. From Equation (17.42) it is easy to compute $\mathbf{x} =$ 0 1 0 1 1 0.

Example 17.7 Knapsack Example

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

Solution

As in Example 17.6, $\mathbf{a}$ is super-increasing; therefore, we can compute $\mathbf{x}$ using Equation (17.42), which again yields

$$\mathbf{x} = 0\,1\,0\,1\,1\,0$$

17.5.5 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

$$M > \sum_{i=1}^{n} a'_i \tag{17.43}$$

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

$$WW^{-1} \text{ modulo-}M = 1 \tag{17.44}$$

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:

$$a_i = Wa'_j \text{ modulo-}M \tag{17.45}$$

The formation of $\mathbf{a}$ using Equation (17.45) 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 (17.45), S can be written as follows:

$$S = \mathbf{ax} = \sum_{i=1}^{n} a_i x_i = \sum_{i=1}^{n} (Wa'_i \text{ modulo-}M) x_i \tag{17.46}$$

The authorized user receives S and, using Equation (17.44), converts it to S':

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

$$= \sum_{i=1}^{n} (W^{-1}\, Wa'_i \text{ modulo-}M)\, x_i \text{ modulo-}M \tag{17.47}$$

$$= \sum_{i=1}^{n} a'_i x_i \text{ modulo-}M$$

$$= \sum_{i=1}^{n} a'_i x_i$$

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

17.5.5.1 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 $WW^{-1} = 1$ 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:

$$a_i = a'_i 2251 \text{ modulo-}9109$$

$$\mathbf{a} = 2343,\ 6215,\ 3892,\ 2895,\ 5055,\ 2123$$

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 $\mathbf{x} = 0\ 1\ 0\ 1\ 1\ 0$ is the message to be transmitted, user B forms

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

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.

17.6 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 17.9 illustrates the algorithms used in versions 2.6, 5.0, and later.

As listed in Table 17.9, 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 17.5.3, 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 17.20. 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 17.9 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.

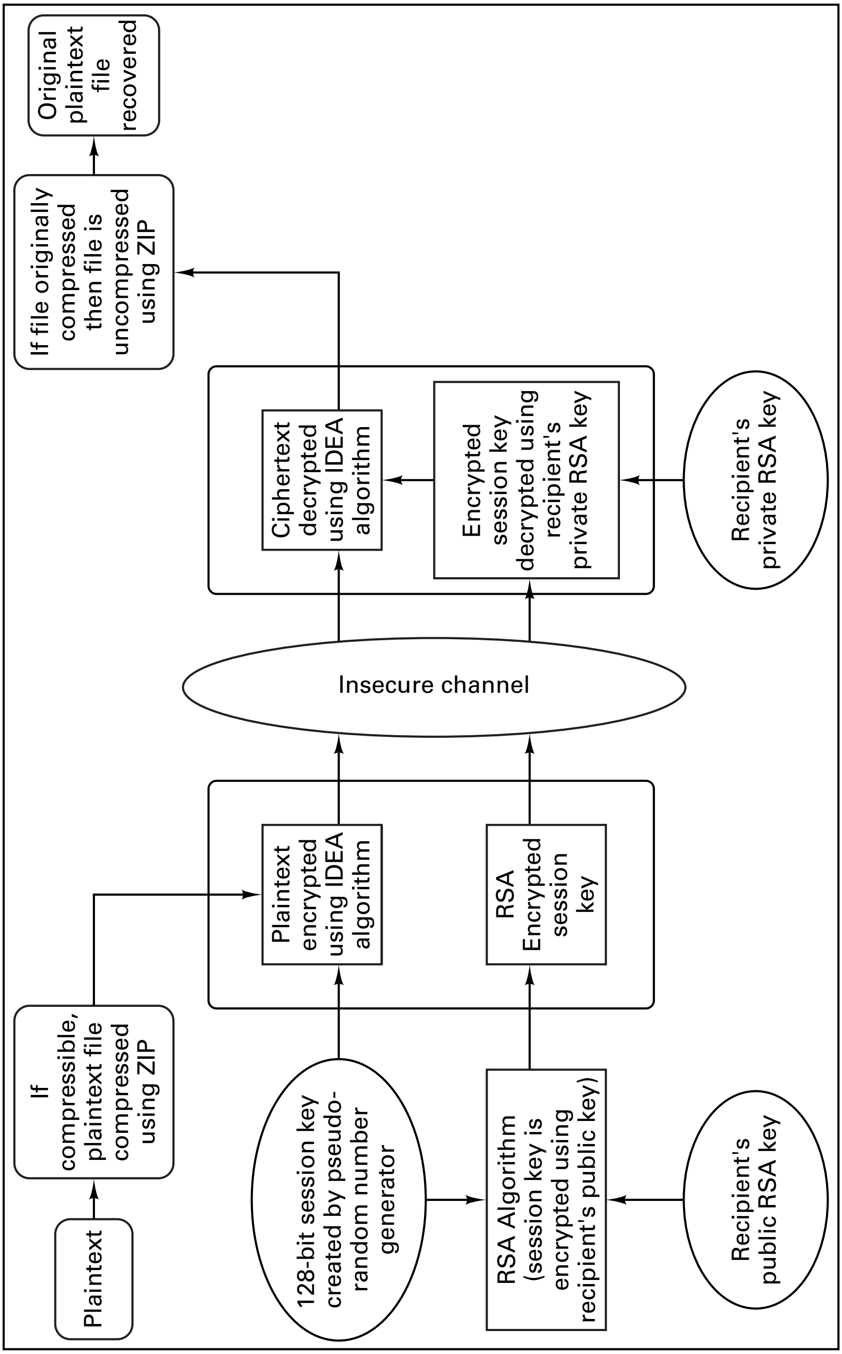


Figure 17.20 The PGP technique.

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 17.20, 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.

17.6.1 Triple-DES, CAST, and IDEA

As listed in Table 17.9, 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.

17.6.1.1 Description of Triple-DES

The Data Encryption Standard (DES) described in Section 17.3.5 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 17.21, this gives the effect of a 168-bit key length.

17.6.1.2 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

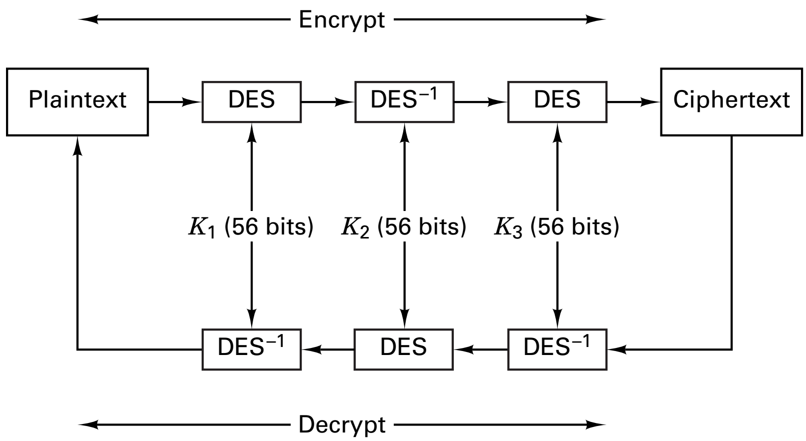


Figure 17.21 Encryption/decryption with triple-DES.

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].

17.6.1.3 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 reordered fashion. As shown in Table 17.10, 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 17.10, yielding a total of 52 subkeys.

The subkey schedule for each round is listed in Table 17.11 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 17.11, 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 17.11. After a round, the second and third 16-bit data subblocks are

TABLE 17.10 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
$Z_1^{out}\ Z_2^{out}\ Z_3^{out}\ Z_4^{out}$	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 17.10 and 17.11, the round number is shown without parentheses for ease of notation.

Each round consists of the steps shown in Table 17.12. 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. $M_1 \times Z_1^{out}$ (first subkey of output transformation)
2. $M_2 + Z_2^{out}$
3. $M_3 + Z_3^{out}$
4. $M_4 \times Z_4^{out}$

TABLE 17.11 IDEA Subkey Schedule

Round	Set of Encryption Subkeys	Set of Decryption Subkeys
1	$Z_1^1\ Z_2^1\ Z_3^1\ Z_4^1\ Z_5^1\ Z_6^1$	$(Z_1^{out})^{-1} - Z_2^{out} - Z_3^{out}\ (Z_4^{out})^{-1}\ Z_5^8\ Z_6^8$
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	$Z_1^{out}\ Z_2^{out}\ Z_3^{out}\ Z_4^{out}$	$(Z_1^1)^{-1} - Z_2^1 - Z_3^1\ (Z_4^1)^{-1}$

Example 17.8 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 17.12 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. XOR[3] the results from steps 1 and 3.
6. XOR the results from steps 2 and 4.
7. Results from step 5 and $Z_5^{(R)}$ are multiplied.
8. Results from step 6 and 7 are added.
9. Results 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 $i = 1, \ldots 4$, 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 17.11 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:

	0100	1000	0100	1001	(4849 hex converted to binary)
XOR	0000	0000	0000	0011	(0003 hex converted to binary)
	0100	1000	0100	1010	

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. Results 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. Results 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 XOR 78B0 = 30F9.
12. Results from steps (3) and (9) are XOR'ed: 0003 XOR 78B0 = 78B3.
13. Results from steps (2) and (10) are XOR'ed: 0002 XOR E221 = E223.
14. Results from steps (4) and (10) are XOR'ed: 0000 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.

17.6.2 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 17.5.3) 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.

17.6.2.1 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 $g^k = p \bmod n$. 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: $y = g^a \bmod n$.
- 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:

$$y_1 = g^k \bmod n$$

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

- 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 17.9 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:

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

17.6.3 PGP Message Encryption

The private-key algorithms that PGP uses for message encryption were presented in Section 17.6.1. The public-key algorithms that PGP uses to encrypt the private-session key were presented in Section 17.6.2. The next example combines the two types of algorithms to illustrate the PGP encryption technique shown in Figure 17.20.

Example 17.10 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 17.5.3.1, 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 17.8, 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 17.5.3.1, 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 transformation 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 17.32 that $C = (M)^e$ modulo-n where M will be one of the 4-digit groups of K_0. The leftmost four groups are encrypted as:

$$C_{12} = (0000)^{17} \bmod 2773 = 0.$$
$$C_{11} = (0032)^{17} \bmod 2773 = 2227.$$
$$C_{10} = (0000)^{17} \bmod 2773 = 0.$$
$$C_9 = (1792)^{17} \bmod 2773 = 2704.$$

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 17.13. 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 17.13 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	$1728^2 \times 32 = 2227$

17.6.4 PGP Authentication and Signature

The public key algorithms can be used to authenticate or "sign" a message. As illustrated in Figure 17.18, 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

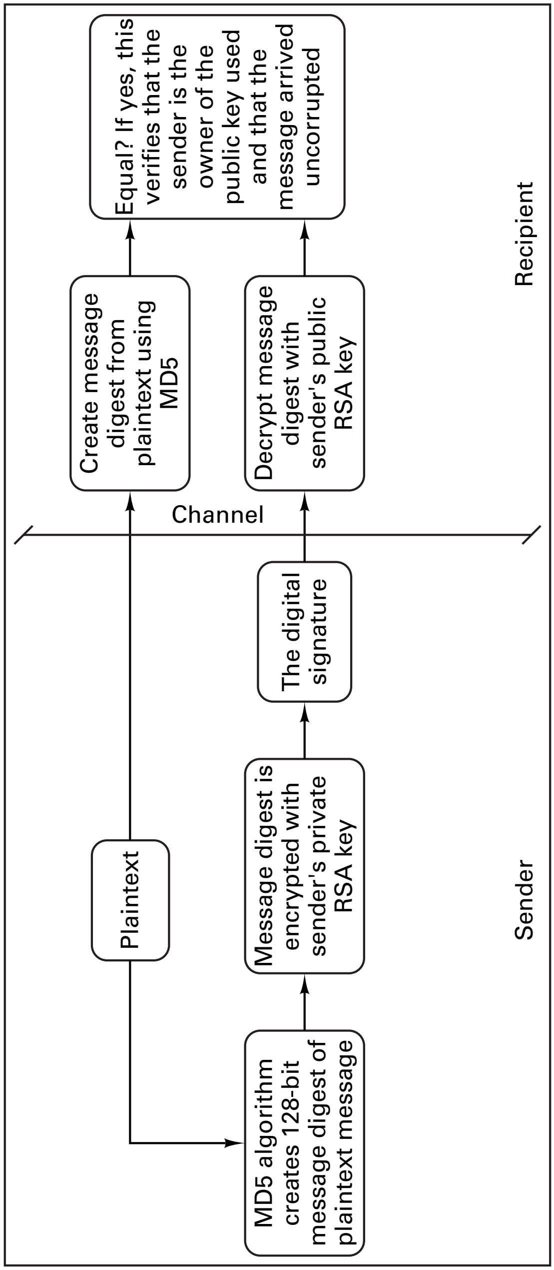


Figure 17.22 PGP signature technique.

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 17.22. 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 17.22, the message is sent without encryption (as plaintext), but it may be encrypted by the method illustrated in Figure 17.20.

17.6.4.1 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.

17.6.4.2 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.)

17.7 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.

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

17.1. 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.

17.2. 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:

$$\text{Morning:}\quad P(S) = \tfrac{1}{8},\quad P(\mathrm{C}) = \tfrac{1}{8},\quad P(\mathrm{L}) = \tfrac{3}{8},\quad P(\mathrm{H}) = \tfrac{3}{8}$$
$$\text{Afternoon:}\quad P(\mathrm{S}) = \tfrac{3}{8},\quad P(\mathrm{C}) = \tfrac{3}{8},\quad P(\mathrm{L}) = \tfrac{1}{8},\quad P(\mathrm{H}) = \tfrac{1}{8}$$

(a) Find the entropy of the weather message.
(b) Find the entropy of the message conditioned on the time of day.

17.3. 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.

17.4. 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.

17.5. Repeat Problem 17.4 for the case where the key sequence is made up of ten integers randomly chosen from the set of numbers 0 to 999.

17.6. **(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?

17.7. In Figures 17.8 and 17.9, *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.

17.8. 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?

17.9. 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?

17.10. Following the RSA algorithm and parameters in Example 17.5, compute the encryption key, e, when the decryption key is chosen to be 151.

17.11. Given e and d that satisfy ed modulo-$\phi(n) = 1$, 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 $(M^e \text{ modulo-}n)^d$ modulo-$n = M$.

17.12. 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.

17.13. 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."

17.14. 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.

17.15. 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.

17.16. 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.

17.17. In the PGP Example 17.10, 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

17.1. What are the two major requirements for a useful *cryptosystem*? (See Section 17.1.2.)

17.2. Shannon suggested two encryption concepts that he termed *confusion* and *diffusion.* Explain what these terms mean. (See Section 17.3.1.)

17.3. If *high-level security* is desired, explain why a linear feedback shift register (LFSR) would not be used. (See Section 17.4.2.)

17.4. Explain the major difference between conventional cryptosystems and *public key cryptosystems.* (See Section 17.5.)

17.5. 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 17.3.5 and 17.6.1.1)

17.6. Describe the steps used for message encryption employed by version 2.6 of the *Pretty Good Privacy* (PGP) technique. (See Section 17.6.1.3.)

Index

NUMBERS

A

B

C

D

E

F

G

H

I

J

K

L

M

Q

R

S

T

U

V

W

Z