

# CRYPTOGRAPHY AND NETWORK SECURITY: *Principles and Practice*

SECOND EDITION

William Stallings

Prentice Hall  
Upper Saddle River, New Jersey 07458

Library of Congress Cataloging-in-Publication Data

Stallings, William.

Cryptography and network security : principles and practice /  
William Stallings. -- 2nd ed.

p. cm.

Rev. ed. of: Network and Internetwork Security.

Includes bibliographical references and index.

ISBN 0-13-869017-0

1. Computer networks--Security measures. 2. Data encryption  
(Computer science) 3. Coding theory. 4. Computer security.

I. Stallings, William. Cryptography and network security.

II. Title.

TK5105.59.S713 1998

005.8--dc21

98-15676

CIP

*Acquisitions editor:* Laura Steele

*Editorial/production supervision:* Rose Kernan

*Editor-in-chief:* Marcia Horton

*Managing editor:* Bayani Mendoza de Leon

*Copy editing:* Patricia Daly

*Art director and cover designer:* Heather Scott

*Director of production and manufacturing:* David W. Riccardi

*Manufacturing Buyer:* Donna Sullivan and Pat Brown

*Editorial assistant:* Catherine Kaibni

© 1999, 1995 by Prentice-Hall, Inc.

Upper Saddle River, New Jersey 07458

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

All right reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5

ISBN 0-13-869017-0

Prentice-Hall International (UK) Limited, *London*

Prentice-Hall of Australia Pty. Limited, *Sydney*

Prentice-Hall of Canada, Inc., *Toronto*

Prentice-Hall Hispanoamericana, S. A., *Mexico*

Prentice-Hall of India Private Limited, *New Delhi*

Prentice-Hall of Japan, Inc., *Tokyo*

Pearson Education Asia Pte. Ltd., *Singapore*



*To Antigone  
never dull  
never boring  
always a Sage*



# CONTENTS

## **Preface, xiii**

## **Chapter 1 Introduction, 1**

- 1.1 Attacks, Services, and Mechanisms, 3
- 1.2 Security Attacks, 6
- 1.3 Security Services, 9
- 1.4 A Model for Internetwork Security, 11
- 1.5 Outline of this Book, 13
- 1.6 Recommended Reading, 16
- Appendix 1A: Internet and Web Resources, 16

## **PART ONE CONVENTIONAL ENCRYPTION, 19**

## **Chapter 2 Conventional Encryption: Classical Techniques, 21**

- 2.1 Conventional Encryption Model, 22
- 2.2 Steganography, 26
- 2.3 Classical Encryption Techniques, 28
- 2.4 Recommended Reading, 44
- 2.5 Problems, 45

## **Chapter 3 Conventional Encryption: Modern Techniques, 49**

- 3.1 Simplified DES, 50
- 3.2 Block Cipher Principles, 57
- 3.3 The Data Encryption Standard, 65
- 3.4 The Strength of DES, 74
- 3.5 Differential and Linear Cryptanalysis, 76
- 3.6 Block Cipher Design Principles, 79
- 3.7 Block Cipher Modes of Operation, 83

**viii CONTENTS**

- 3.8 Recommended Reading, 89
- 3.9 Problems, 89
- Appendix 3A: Bent Functions, 91

**Chapter 4 Conventional Encryption: Algorithms, 93**

- 4.1 Triple DES, 93
- 4.2 International Data Encryption Algorithm, 99
- 4.3 Blowfish, 109
- 4.4 RC5, 114
- 4.5 CAST-128, 120
- 4.6 RC2, 124
- 4.7 Characteristics of Advanced Symmetric Block Ciphers, 126
- 4.8 Problems, 127

**Chapter 5 Confidentiality Using Conventional Encryption, 131**

- 5.1 Placement of Encryption Function, 132
- 5.2 Traffic Confidentiality, 139
- 5.3 Key Distribution, 141
- 5.4 Random Number Generation, 149
- 5.5 Recommended Reading, 156
- 5.6 Problems, 157

**PART TWO PUBLIC-KEY ENCRYPTION  
AND HASH FUNCTIONS, 161**

**Chapter 6 Public-Key Cryptography, 163**

- 6.1 Principles of Public-Key Cryptosystems, 164
- 6.2 The RSA Algorithm, 173
- 6.3 Key Management, 182
- 6.4 Diffie-Hellman Key Exchange, 190
- 6.5 Elliptic Curve Cryptography, 193
- 6.6 Recommended Reading, 199
- 6.7 Problems, 199
- Appendix 6A: The Complexity of Algorithms, 203

**Chapter 7 Introduction to Number Theory, 207**

- 7.1 Prime and Relatively Prime Numbers, 208
- 7.2 Modular Arithmetic, 211
- 7.3 Fermat's and Euler's Theorems, 217

- 7.4 Testing for Primality, 221
- 7.5 Euclid's Algorithm, 223
- 7.6 The Chinese Remainder Theorem, 226
- 7.7 Discrete Logarithms, 228
- 7.8 Recommended Reading, 233
- 7.9 Problems, 233

## **Chapter 8 Message Authentication and Hash Functions, 237**

- 8.1 Authentication Requirements, 238
- 8.2 Authentication Functions, 239
- 8.3 Message Authentication Codes, 249
- 8.4 Hash Functions, 253
- 8.5 Security of Hash Functions and MACs, 259
- 8.6 Recommended Reading, 263
- 8.7 Problems, 263
- Appendix 8A: Mathematical Basis of Birthday Attack, 264

## **Chapter 9 Hash and Mac Algorithms, 271**

- 9.1 MD5 Message Digest Algorithm, 272
- 9.2 Secure Hash Algorithm (SHA-1), 281
- 9.3 RIPEMD-160, 286
- 9.4 HMAC, 293
- 9.5 Problems, 298

## **Chapter 10 Digital Signatures and Authentication Protocols, 299**

- 10.1 Digital Signatures, 299
- 10.2 Authentication Protocols, 303
- 10.3 Digital Signature Standard, 311
- 10.4 Recommended Reading, 314
- 10.5 Problems, 315
- Appendix 10A: Proof of the DSS Algorithm, 317

# **PART THREE NETWORK SECURITY PRACTICE, 321**

## **Chapter 11 Authentication Applications, 323**

- 11.1 Kerberos, 323
- 11.2 X.509 Directory Authentication Service, 341
- 11.3 Recommended Reading, 350
- 11.4 Problems, 350
- Appendix 11A: Kerberos Encryption Techniques, 351

**Chapter 12 Electronic Mail Security, 355**

- 12.1 Pretty Good Privacy, 356
- 12.2 S/MIME, 374
- 12.3 Recommended Readings, 390
- 12.4 Problems, 391
- Appendix 12A: Data Compression Using ZIP, 391
- Appendix 12B: Radix-64 Conversion, 394
- Appendix 12C: PGP Random Number Generation, 395

**Chapter 13 IP Security, 399**

- 13.1 IP Security Overview, 400
- 13.2 IP Security Architecture, 402
- 13.3 Authentication Header, 408
- 13.4 Encapsulating Security Payload, 413
- 13.5 Combining Security Associations, 418
- 13.6 Key Management, 421
- 13.7 Recommended Reading, 432
- 13.8 Problems, 432
- Appendix 13A: Internetworking and Internet Protocols, 433

**Chapter 14 Web Security, 441**

- 14.1 Web Security Requirements, 441
- 14.2 Secure Sockets Layer and Transport Layer Security, 444
- 14.3 Secure Electronic Transaction, 461
- 14.4 Recommended Reading, 472
- 14.5 Problems, 473

**PART FOUR SYSTEM SECURITY, 475**

**Chapter 15 Intruders, Viruses, and Worms, 477**

- 15.1 Intruders, 478
- 15.2 Viruses and Related Threats, 501
- 15.3 Recommended Reading, 514
- 15.4 Problems, 514

**Chapter 16 Firewalls, 517**

- 16.1 Firewall Design Principles, 517
- 16.2 Trusted Systems, 527
- 16.3 Recommended Reading, 532
- 16.4 Problems, 533

**Appendix A Projects for Teaching Cryptography and Network Security, 535**

- A.1 Research Projects, 536
- A.2 Programming Projects, 536
- A.3 Reading/Report Assignments, 537

**Glossary, 539**

**References, 545**

**Index, 559**





# PREFACE

*"The tie, if I might suggest it, sir, a shade more tightly knotted. One aims at the perfect butterfly effect. If you will permit me —"*

*"What does it matter, Jeeves, at a time like this? Do you realize that Mr. Little's domestic happiness is hanging in the scale?"*

*"There is no time, sir, at which ties do not matter."*

**—Very Good, Jeeves! P. G. Wodehouse**

In this age of universal electronic connectivity, of viruses and hackers, of electronic eavesdropping and electronic fraud, there is indeed no time at which security does not matter. Two trends have come together to make the topic of this book of vital interest. First, the explosive growth in computer systems and their interconnections via networks has increased the dependence of both organizations and individuals on the information stored and communicated using these systems. This, in turn, has led to a heightened awareness of the need to protect data and resources from disclosure, to guarantee the authenticity of data and messages, and to protect systems from network-based attacks. Second, the disciplines of cryptography and network security have matured, leading to the development of practical, readily available applications to enforce network security.

## OBJECTIVES

It is the purpose of this book to provide a practical survey of both the principles and practice of cryptography and network security. In the first two parts of the book, the basic issues to be addressed by a network security capability are explored by providing a tutorial and survey of cryptography and network security technology. The latter part of the book deals with the practice of network security: practical applications that have been implemented and are in use to provide network security.

The subject, and therefore this book, draws on a variety of disciplines. In particular, it is impossible to appreciate the significance of some of the

techniques discussed in this book without a basic understanding of number theory and some results from probability theory. Nevertheless, an attempt has been made to make the book self-contained. The book presents not only the basic mathematical results that are needed but provides the reader with an intuitive understanding of those results. Such background material is introduced as needed. This approach helps to motivate the material that is introduced, and the author considers this preferable to simply presenting all of the mathematical material in a lump at the beginning of the book.

## INTENDED AUDIENCE

The book is intended for both an academic and a professional audience. As a textbook, it is intended as a one-semester undergraduate course in cryptography and network security for computer science, computer engineering, and electrical engineering majors. The book also serves as a basic reference volume and is suitable for self-study.

### Plan of the Book

The book is organized in four parts:

- I. **Conventional Encryption:** A detailed examination of conventional encryption algorithms and design principles, including a discussion of the use of conventional encryption for confidentiality.
- II. **Public-Key Encryption and Hash Functions:** A detailed examination of public-key encryption algorithms and design principles. This part also examines the use of message authentication codes and hash functions, as well as digital signatures and public-key certificates.
- III. **Network Security Practice:** Covers important network security tools and applications, including Kerberos, X.509v3 certificates, PGP, S/MIME, IP Security, SSL/TLS, and SET.
- IV. **System Security:** Looks at system-level security issues, including the threat of and countermeasures for intruders and viruses, and the use of firewalls and trusted systems.

A more detailed, chapter-by-chapter summary appears at the end of Chapter 1. In addition, the book includes an extensive glossary, a list of frequently used acronyms, and a bibliography. There are also end-of-chapter problems and suggestions for further reading.

## INTERNET SERVICES FOR INSTRUCTORS AND STUDENTS

There is a Web page for this book that provides support for students and instructors. The page includes links to relevant sites, transparency masters of figures in the book in PDF (Adobe Acrobat) format, and sign-up information for the book's

Internet mailing list. The Web page is at <http://www.shore.net/~ws/Security2e.html>. An Internet mailing list has been set up so that instructors using this book can exchange information, suggestions, and questions with each other and with the author. As soon as typos or other errors are discovered, an errata list for this book will be available at <http://www.shore.net/~ws>.

## PROJECTS FOR TEACHING CRYPTOGRAPHY AND NETWORK SECURITY

For many instructors, an important component of a cryptography or security course is a project or set of projects by which the student gets hands-on experience to reinforce concepts from the text. This book provides an unparalleled degree of support for including a projects component in the course. The instructor's manual not only includes guidance on how to assign and structure the projects, but also includes a set of suggested projects that covers a broad range of topics from the text:

- **Research Projects:** A series of research assignments that instruct the student to research a particular topic on the Internet and write a report.
- **Programming Projects:** A series of programming projects that cover a broad range of topics and that can be implemented in any suitable language on any platform.
- **Reading/Report Assignments:** A list of papers in the literature, one for each chapter, that can be assigned for the student to read and then write a short report.

See Appendix A for details.

## WHAT'S NEW IN THE SECOND EDITION

In the four years since the first edition of this book was published, the field has seen continued innovations and improvements. In this new edition, I try to capture these changes while maintaining a broad and comprehensive coverage of the entire field. To begin this process of revision, the first edition of this book was extensively reviewed by a number of professors who teach the subject. The result is that, in many places, the narrative has been clarified and tightened, and illustrations have been improved. Also, a number of new “field-tested” problems have been added.

An obvious change is in the title; the book is now called *Cryptography and Network Security*, to reflect the central role of cryptographic algorithms in network security. The book has also been drastically reorganized to provide a more logical sequence both for classroom instruction and self-study.

Beyond these refinements to improve pedagogy and user friendliness, there have been major substantive changes throughout the book. Highlights include the following:

- **New—Discussion of Block Cipher Design:** Three sections that discuss the structure of block ciphers, block cipher design principles, and features of

recent advanced ciphers have been added, greatly strengthening the overall discussion of conventional encryption.

- **New—Coverage of Additional Conventional Encryption Algorithms:** The book now includes coverage of recent algorithms that are being found in commercial products and Internet standards, including Blowfish, RC5, and CAST-128.
- **New—Treatment of Elliptic Curve Cryptography:** This is becoming an important alternative to RSA and Diffie-Hellman for public-key cryptography.
- **Expanded—Coverage of Number Theory:** The coverage has been expanded to an entire chapter and includes numerous worked-out examples to clarify this abstract subject.
- **New—Discussion of Hash Code and MAC Design:** Discussion has been added on design principles and security of hash functions and message authentication codes.
- **New—Coverage of Additional Hash and MAC Algorithms:** The book now includes coverage of recent algorithms that are being found in commercial products and Internet standards, including RIPEMD-160 and HMAC.
- **Expanded—Coverage of X.509 and New Treatment of X.509v3:** X.509 public-key certificates, especially version 3, are now found in numerous products and Internet standards.
- **New—Coverage of S/MIME:** S/MIME has become the standard for commercial secure electronic mail.
- **New—Chapter on IP Security:** IPsec is an important new set of standards for constructing virtual private networks and for end-to-end security over the Internet. An entire chapter has been added to treat this important topic.
- **New—Chapter on Web Security:** Web security has become one of the most important areas of network security and raises many new challenges. An entire chapter has been added to treat this topic. The chapter covers two major Web security standards:
  - **Secure Socket Layer (SSL) and Transport Layer Security (TLS):** SSL is the defacto standard for Web security found in virtually all browser and server offerings; TLS is an emerging Internet standard intended to replace SSL.
  - **Secure Electronic Transactions (SET):** SET is the emerging standard for secure electronic commerce over the Web.
- **New—Chapter on Firewalls:** Firewalls have emerged as the product of choice for protecting corporate sites that are connected to the Internet.
- **New—Expanded Instructor Support:** The instructor's manual, as before, includes solutions to all of the problems in the book. In addition, the manual provides support for student projects, as described previously.
- **Other changes:** These include the following:
  - A new section on bent functions, which are important in the design of S-boxes for conventional encryption algorithms.
  - Pointers to relevant Web sites are found in the Recommended Reading section of many chapters.
  - Dozens of new homework problems have been added.

## ACKNOWLEDGMENTS

This book has benefited from review by numerous experts in the field, who gave generously of their time and expertise. The following people reviewed all or a large part of the first edition: Shivakumar Sastry (Rockwell Automation), Alan Sherman (University of MD Baltimore County), Tom Dunigan (University of Tennessee), Dan Boneh (Stanford University), and Sushil Jajodia (George Mason University).

In addition, I was extremely fortunate to have reviews of individual topics by “subject-area gurus,” including Ron Rivest at MIT (RC5 and MD5); Tim Mathews of RSA Data Security (S/MIME); Bruce Schneier of Counterpane Systems (Blowfish); Carlisle Adams of Entrust Technologies (CAST and bent functions); Alfred Menezes of University of Waterloo (Elliptic Curve Cryptography); Terry Ritter of Ritter Software Engineering (bent functions); William G. Sutton, Editor/Publisher of the Cryptogram (Classical Encryption); Aviel Rubin of AT&T Labs (Number Theory); Bart Preneel of Katholieke Universiteit Leuven (RIPEMD-160); Michael Markowitz of Information Security Corporation (SHA and DSS); Xuejia Lai of R3 Security Engineering AG (IDEA); Don Davis of Openvision Technologies (Kerberos); Steve Kent (X.509); Phil Zimmermann of Boulder Software Engineering (PGP); and Ed Scheafer of Santa Clara University (Simplified DES).

The following people contributed project assignments that appear in the instructor’s manual: Henning Schulzrinne (Columbia University); Cetin Kaya Koc (Oregon State University); and David Balenson (Trusted Information Systems and George Washington University).

Also, I would like to acknowledge those who contributed homework problems: Luke O’Connor; Joseph Kusmiss of MITRE; Carl Ellison of Stratus; Shunmugavel Rajarathinam; and Jozef Vyskoc, who contributed those marvelous Sherlock Holmes problems.

With all this assistance, little remains for which I can take full credit. However, I am proud to say that, with no help whatsoever, I selected all of the quotations.



CRYPTOGRAPHY AND  
NETWORK SECURITY:  
*Principles and Practice*

SECOND EDITION





# CHAPTER 1

## INTRODUCTION

*The combination of space, time, and strength that must be considered as the basic elements of this theory of defense makes this a fairly complicated matter. Consequently, it is not easy to find a fixed point of departure.*

—*On War*, Carl Von Clausewitz

*The art of war teaches us to rely not on the likelihood of the enemy's not coming, but on our own readiness to receive him; not on the chance of his not attacking, but rather on the fact that we have made our position unassailable.*

—*The Art of War*, Sun Tzu

The requirements of *information security* within an organization have undergone two major changes in the last several decades. Before the widespread use of data processing equipment, the security of information felt to be valuable to an organization was provided primarily by physical and administrative means. An example of the former is the use of rugged filing cabinets with a combination lock for storing sensitive documents. An example of the latter is personnel screening procedures used during the hiring process.

With the introduction of the computer, the need for automated tools for protecting files and other information stored on the computer became evident. This is especially the case for a shared system, such as a time-sharing system, and the need is even more acute for systems that can be accessed over a public telephone or data network. The generic name for the collection of tools designed to protect data and to thwart hackers is *computer security*.

The second major change that affected security is the introduction of distributed systems and the use of networks and communications facilities for

## 2 CHAPTER 1 / INTRODUCTION

carrying data between terminal user and computer and between computer and computer. Network security measures are needed to protect data during their transmission. In fact, the term *network security* is somewhat misleading, because virtually all business, government, and academic organizations interconnect their data processing equipment with a collection of interconnected networks. Such a collection is often referred to as an internet.<sup>1</sup>

There are no clear boundaries between these two forms of security. For example, one of the most publicized types of attack on information systems is the computer virus. A virus may be introduced into a system physically when it arrives on a diskette and is subsequently loaded onto a computer. Viruses may also arrive over an internet. In either case, once the virus is resident on a computer system, internal computer security tools are needed to detect and recover from the virus.

This book focuses on internet security, which consists of measures to deter, prevent, detect, and correct security violations that involve the transmission of information. That is a broad statement that covers a host of possibilities. To give you a feel for the areas covered in this book, consider the following examples of security violations:

1. User A transmits a file to user B. The file contains sensitive information (e.g., payroll records) that are to be protected from disclosure. User C, who is not authorized to read the file, is able to monitor the transmission and capture a copy of the file during its transmission.
2. A network manager, D, transmits a message to a computer, E, under its management. The message instructs computer E to update an authorization file to include the identities of a number of new users who are to be given access to that computer. User F intercepts the message, alters its contents to add or delete entries, and then forwards the message to E, which accepts the message as coming from manager D and updates its authorization file accordingly.
3. Rather than intercept a message, user F constructs its own message with the desired entries and transmits that message to E as if it had come from manager D. Computer E accepts the message as coming from manager D and updates its authorization file accordingly.
4. An employee is fired without warning. The personnel manager sends a message to a server system to invalidate the employee's account. When the invalidation is accomplished, the server is to post a notice to the employee's file as confirmation of the action. The employee is able to intercept the message and delay it long enough to make a final access to the server to retrieve sensitive information. The message is then forwarded, the action taken, and the confirmation posted. The employee's action may go unnoticed for some considerable time.
5. A message is sent from a customer to a stockbroker with instructions for various transactions. Subsequently, the investments lose value and the customer denies sending the message.

---

<sup>1</sup>We use the term *internet*, with a lowercase "i," to refer to any interconnected collection of network. A corporate intranet is an example of an internet. The Internet with a capital "I" may be one of the facilities used by an organization to construct its internet.

Although this list by no means exhausts the possible types of security violations, it illustrates the range of concerns of network security.

Internetwork security is both fascinating and complex. Some of the reasons follow:

1. Security involving communications and networks is not as simple as it might first appear to the novice. The requirements seem to be straightforward; indeed, most of the major requirements for security services can be given self-explanatory one-word labels: confidentiality, authentication, nonrepudiation, integrity. But the mechanisms used to meet those requirements can be quite complex, and understanding them may involve rather subtle reasoning.
2. In developing a particular security mechanism or algorithm, one must always consider potential countermeasures. In many cases, countermeasures are designed by looking at the problem in a completely different way, therefore exploiting an unexpected weakness in the mechanism.
3. Because of point 2, the procedures used to provide particular services are often counterintuitive: It is not obvious from the statement of a particular requirement that such elaborate measures are needed. It is only when the various countermeasures are considered that the measures used make sense.
4. Having designed various security mechanisms, it is necessary to decide where to use them. This is true both in terms of physical placement (e.g., at what points in a network are certain security mechanisms needed) and in a logical sense (e.g., at what layer or layers of an architecture such as TCP/IP should mechanisms be placed).
5. Security mechanisms usually involve more than a particular algorithm or protocol. They usually also require that participants be in possession of some secret information (e.g., an encryption key), which raises questions about the creation, distribution, and protection of that secret information. There is also a reliance on communications protocols whose behavior may complicate the task of developing the security mechanism. For example, if the proper functioning of the security mechanism requires setting time limits on the transit time of a message from sender to receiver, then any protocol or network that introduces variable, unpredictable delays may render such time limits meaningless.

Thus, there is much to consider. This chapter provides a general overview of the subject matter that structures the material in the remainder of the book. We begin with a discussion of the types of attacks that create the need for network security services and mechanisms. Then we develop a general overall model within which the security services and mechanisms can be viewed.

## 1.1 ATTACKS, SERVICES, AND MECHANISMS

To assess the security needs of an organization effectively and to evaluate and choose various security products and policies, the manager responsible for security needs some systematic way of defining the requirements for security and characterizing the approaches to satisfying those requirements. One approach is to consider three aspects of information security:

- **Security attack:** Any action that compromises the security of information owned by an organization.
- **Security mechanism:** A mechanism that is designed to detect, prevent, or recover from a security attack.
- **Security service:** A service that enhances the security of the data processing systems and the information transfers of an organization. The services are intended to counter security attacks, and they make use of one or more security mechanisms to provide the service.

### Services

Let us consider these topics briefly, in reverse order. We can think of information security services as replicating the types of functions normally associated with physical documents. Much of the activity of humankind, in areas as diverse as commerce, foreign policy, military action, and personal interactions, depends on the use of documents and on both parties to a transaction having confidence in the integrity of those documents. Documents typically have signatures and dates; they may need to be protected from disclosure, tampering, or destruction; they may be notarized or witnessed; may be recorded or licensed, and so on.

As information systems become ever more pervasive and essential to the conduct of our affairs, electronic information takes on many of the roles traditionally performed by paper documents. Accordingly, the types of functions traditionally associated with paper documents must be performed on documents that exist in electronic form. Several aspects of electronic documents make the provision of such functions or services challenging:

1. It is usually possible to discriminate between an original paper document and a xerographic copy. However, an electronic document is merely a sequence of bits; there is no difference whatsoever between the “original” and any number of copies.
2. An alteration to a paper document may leave some sort of physical evidence of the alteration. For example, an erasure can result in a thin spot or a roughness in the surface. Altering bits in a computer memory or in a signal leaves no physical trace.
3. Any “proof” process associated with a physical document typically depends on the physical characteristics of that document (e.g., the shape of a handwritten signature or an embossed notary seal). Any such proof of authenticity of an electronic document must be based on internal evidence present in the information itself.

Table 1.1 lists some of the common functions traditionally associated with documents and for which analogous function for electronic documents and messages are required. We can think of these functions as requirements to be met by a security facility.

The list of Table 1.1 is lengthy and is not by itself a useful guide to organizing a security facility. Computer and network security research and development have instead focused on three or four general security services that encompass the vari-

**Table 1.1** A Partial List of Common Information Integrity Functions [SIMM92b]

<ul style="list-style-type: none"> <li>• Identification</li> <li>• Authorization</li> <li>• License and/or certification</li> <li>• Signature</li> <li>• Witnessing (notarization)</li> <li>• Concurrence</li> <li>• Liability</li> <li>• Receipts</li> <li>• Certification of origination and/or receipt</li> </ul>	<ul style="list-style-type: none"> <li>• Endorsement</li> <li>• Access (egress)</li> <li>• Validation</li> <li>• Time of occurrence</li> <li>• Authenticity—software and/or files</li> <li>• Vote</li> <li>• Ownership</li> <li>• Registration</li> <li>• Approval/disapproval</li> <li>• Privacy (secrecy)</li> </ul>
--	--

ous functions required of an information security facility. One useful classification<sup>2</sup> of security services is the following:

- **Confidentiality:** Ensures that the information in a computer system and transmitted information are accessible only for reading by authorized parties. This type of access includes printing, displaying, and other forms of disclosure, including simply revealing the existence of an object.
- **Authentication:** Ensures that the origin of a message or electronic document is correctly identified, with an assurance that the identity is not false.
- **Integrity:** Ensures that only authorized parties are able to modify computer system assets and transmitted information. Modification includes writing, changing, changing status, deleting, creating, and delaying or replaying of transmitted messages.
- **Nonrepudiation:** Requires that neither the sender nor the receiver of a message be able to deny the transmission.
- **Access control:** Requires that access to information resources may be controlled by or for the target system.
- **Availability:** Requires that computer system assets be available to authorized parties when needed.

## Mechanisms

There is no single mechanism that will provide all the services just listed or perform all the functions listed in Table 1.1. As this book proceeds, we will see a variety of mechanisms that come into play. However, we can note at this point that there is one particular element that underlies most of the security mechanisms in use: cryptographic techniques. Encryption or encryption-like transformations of information are the most common means of providing security. Thus, this book focuses on the development, use, and management of such techniques.

<sup>2</sup>There is no universal agreement about many of the terms used in the security literature. For example, the term *integrity* is sometimes used to refer to all aspects of information security. The term *authentication* is sometimes used to refer both to verification of identity and to the various functions listed under integrity in the following list.

**Table 1.2** Reasons for Cheating [SIMM92b]

- 
1. Gain unauthorized access to information (i.e., violate secrecy or privacy).
  2. Impersonate another user either to shift responsibility (i.e., liability) or else to use the other's license for the purpose of:
    - a. originating fraudulent information,
    - b. modifying legitimate information,
    - c. using fraudulent identity to gain unauthorized access,
    - d. fraudulently authorizing transactions or endorsing them.
  3. Disavow responsibility or liability for information the cheater did originate.
  4. Claim to have received from some other user information that the cheater created (i.e., fraudulent attribution of responsibility or liability).
  5. Claim to have sent to a receiver (at a specified time) information that was not sent (or was sent at a different time).
  6. Either disavow receipt of information that was in fact received, or claim a false time of receipt.
  7. Enlarge cheater's legitimate license (for access, origination, distribution, etc.).
  8. Modify (without authority to do so) the license of others (fraudulently enroll others, restrict or enlarge existing licenses, etc.).
  9. Conceal the presence of some information (a covert communication) in other information (the overt communication).
  10. Insert self into a communications link between other users as an active (undetected) relay point.
  11. Learn who accesses which information (sources, files, etc.) and when the accesses are made even if the information itself remains concealed (e.g., a generalization of traffic analysis from communications channels to data bases, software, etc.).
  12. Impeach an information integrity protocol by revealing information the cheater is supposed to (by the terms of the protocol) keep secret.
  13. Pervert the function of software, typically by adding a covert function.
  14. Cause others to violate a protocol by means of introducing incorrect information.
  15. Undermine confidence in a protocol by causing apparent failures in the system.
  16. Prevent communication among other users, in particular, surreptitious interference to cause authentic communication to be rejected as unauthentic.
- 

### Attacks

As G. J. Simmons perceptively points out, information security is about how to prevent cheating or, failing that, to detect cheating in information-based systems wherein the information itself has no meaningful physical existence [SIMM92a].

Table 1.2 lists some of the more obvious examples of cheating, each of which has arisen in a number of real-world cases. These are examples of specific attacks that an organization or an individual (or an organization on behalf of its employees) may need to counter. The nature of the attack that concerns an organization varies greatly from one set of circumstances to another. Fortunately, we can approach the problem from a different angle by looking at the generic types of attack that might be encountered. That is the subject of the next section.

## 1.2 SECURITY ATTACKS

Attacks on the security of a computer system or network are best characterized by viewing the function of the computer system as providing information. In general, there is a flow of information from a source, such as a file or a region of main memory, to a destination, such as another file or a user. This normal flow is depicted in

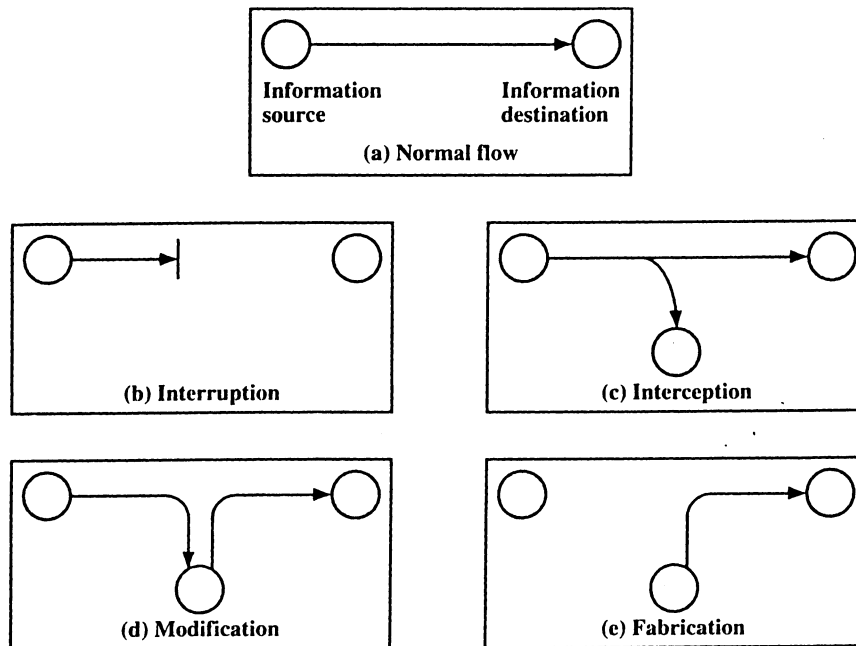


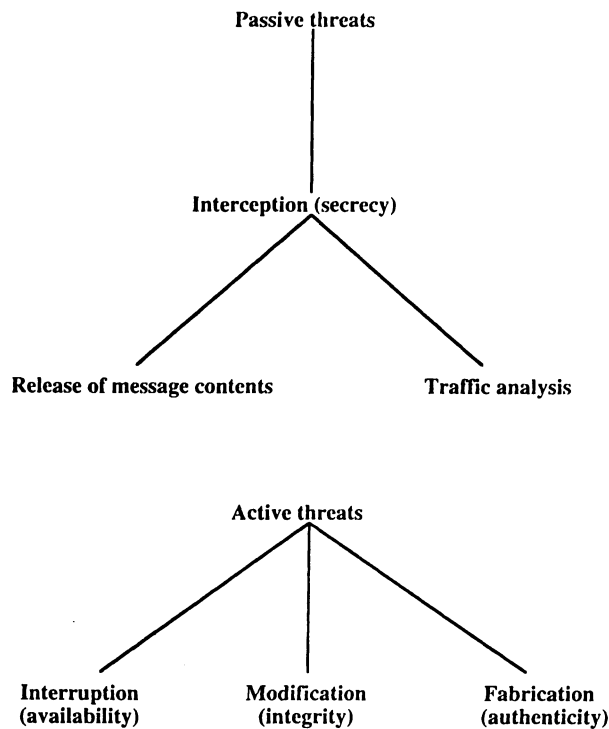
Figure 1.1 Security Threats.

Figure 1.1a. The remaining parts of the figure show the following four general categories of attack:

- **Interruption:** An asset of the system is destroyed or becomes unavailable or unusable. This is an attack on **availability**. Examples include destruction of a piece of hardware, such as a hard disk, the cutting of a communication line, or the disabling of the file management system.
- **Interception:** An unauthorized party gains access to an asset. This is an attack on **confidentiality**. The unauthorized party could be a person, a program, or a computer. Examples include wiretapping to capture data in a network, and the illicit copying of files or programs.
- **Modification:** An unauthorized party not only gains access to but tampers with an asset. This is an attack on **integrity**. Examples include changing values in a data file, altering a program so that it performs differently, and modifying the content of messages being transmitted in a network.
- **Fabrication:** An unauthorized party inserts counterfeit objects into the system. This is an attack on **authenticity**. Examples include the insertion of spurious messages in a network or the addition of records to a file.

A useful categorization of these attacks is in terms of passive attacks and active attacks (Figure 1.2).<sup>3</sup>

<sup>3</sup>The categorization of attacks in this section is based on one initially proposed by Steve Kent [KENT77]. This breakdown is still valid today and is the basis for most descriptions of security attacks.



**Figure 1.2** Active and Passive Network Security Threats.

### Passive Attacks

Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the opponent is to obtain information that is being transmitted. Two types of passive attacks are release of message contents and traffic analysis.

The **release of message contents** is easily understood. A telephone conversation, an electronic mail message, and a transferred file may contain sensitive or confidential information. We would like to prevent the opponent from learning the contents of these transmissions.

The second passive attack, **traffic analysis**, is more subtle. Suppose that we had a way of masking the contents of messages or other information traffic so that opponents, even if they captured the message, could not extract the information from the message. The common technique for masking contents is encryption. If we had encryption protection in place, an opponent might still be able to observe the pattern of these messages. The opponent could determine the location and identity of communicating hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of the communication that was taking place.

Passive attacks are very difficult to detect because they do not involve any alteration of the data. However, it is feasible to prevent the success of these attacks. Thus, the emphasis in dealing with passive attacks is on prevention rather than detection.



### Active Attacks

The second major category of attack is **active attacks**. These attacks involve some modification of the data stream or the creation of a false stream and can be subdivided into four categories: masquerade, replay, modification of messages, and denial of service.

A **masquerade** takes place when one entity pretends to be a different entity. A masquerade attack usually includes one of the other forms of active attack. For example, authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an authorized entity with few privileges to obtain extra privileges by impersonating an entity that has those privileges.

**Replay** involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect.

**Modification of messages** simply means that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect. For example, a message meaning "Allow John Smith to read confidential file *accounts*" is modified to mean "Allow Fred Brown to read confidential file *accounts*."

The **denial of service** prevents or inhibits the normal use or management of communications facilities. This attack may have a specific target; for example, an entity may suppress all messages directed to a particular destination (e.g., the security audit service). Another form of service denial is the disruption of an entire network, either by disabling the network or by overloading it with messages so as to degrade performance.

Active attacks present the opposite characteristics of passive attacks. Whereas passive attacks are difficult to detect, measures are available to prevent their success. On the other hand, it is quite difficult to prevent active attacks absolutely, because to do so would require physical protection of all communications facilities and paths at all times. Instead, the goal is to detect them and to recover from any disruption or delays caused by them. Because the detection has a deterrent effect, it may also contribute to prevention.

## 1.3 SECURITY SERVICES

### Confidentiality

Confidentiality is the protection of transmitted data from passive attacks. With respect to the release of message contents, several levels of protection can be identified. The broadest service protects all user data transmitted between two users over a period of time. For example, if a virtual circuit is set up between two systems, this broad protection would prevent the release of any user data transmitted over the virtual circuit. Narrower forms of this service can also be defined, including the protection of a single message or even specific fields within a message. These refinements are less useful than the broad approach and may even be more complex and expensive to implement.

The other aspect of confidentiality is the protection of traffic flow from analysis. This requires that an attacker not be able to observe the source and destination, frequency, length, or other characteristics of the traffic on a communications facility.

### **Authentication**

The authentication service is concerned with assuring that a communication is authentic. In the case of a single message, such as a warning or alarm signal, the function of the authentication service is to assure the recipient that the message is from the source that it claims to be from. In the case of an ongoing interaction, such as the connection of a terminal to a host, two aspects are involved. First, at the time of connection initiation, the service assures that the two entities are authentic (that is, that each is the entity that it claims to be). Second, the service must assure that the connection is not interfered with in such a way that a third party can masquerade as one of the two legitimate parties for the purposes of unauthorized transmission or reception.

### **Integrity**

As with confidentiality, integrity can apply to a stream of messages, a single message, or selected fields within a message. Again, the most useful and straightforward approach is total stream protection.

A connection-oriented integrity service, one that deals with a stream of messages, assures that messages are received as sent, with no duplication, insertion, modification, reordering, or replays. The destruction of data is also covered under this service. Thus, the connection-oriented integrity service addresses both message stream modification and denial of service. On the other hand, a connectionless integrity service, one that deals with individual messages only without regard to any larger context, generally provides protection against message modification only. Kent points out that a hybrid service can be offered for applications that require some protection against replay and reordering but that do not require strict sequencing [KENT93a].

We can make a distinction between the service with and without recovery. Because the integrity service relates to active attacks, we are concerned with detection rather than prevention. If a violation of integrity is detected, then the service may simply report this violation, and some other portion of software or human intervention is required to recover from the violation. Alternatively, there are mechanisms available to recover from the loss of integrity of data, as we will review subsequently. The incorporation of automated recovery mechanisms is, in general, the more attractive alternative.

### **Nonrepudiation**

Nonrepudiation prevents either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the message was in fact sent by the alleged sender. Similarly, when a message is received, the sender can

### Access Control

In the context of network security, access control is the ability to limit and control the access to host systems and applications via communications links. To achieve this control, each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual.

### Availability

A variety of attacks can result in the loss of or reduction in availability. Some of these attacks are amenable to automated countermeasures, such as authentication and encryption, whereas others require some sort of physical action to prevent or recover from loss of availability of elements of a distributed system.

## 1.4 A MODEL FOR NETWORK SECURITY

A model for much of what we will be discussing is captured, in very general terms, in Figure 1.3. A message is to be transferred from one party to another across some sort of internet. The two parties, who are the *principals* in this transaction, must cooperate for the exchange to take place. A logical information channel is established by defining a route through the internet from source to destination and by the cooperative use of communication protocols (e.g., TCP/IP) by the two principals.

Security aspects come into play when it is necessary or desirable to protect the information transmission from an opponent who may present a threat to confidentiality, authenticity, and so on. All the techniques for providing security have two components:

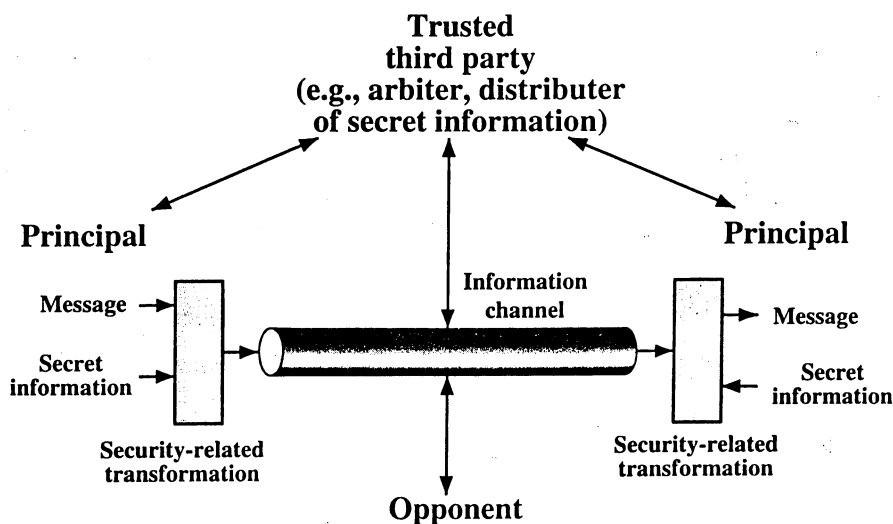


Figure 1.3 Model for Network Security.

- A security-related transformation on the information to be sent. Examples include the encryption of the message, which scrambles the message so that it is unreadable by the opponent, and the addition of a code based on the contents of the message, which can be used to verify the identity of the sender.
- Some secret information shared by the two principals and, it is hoped, unknown to the opponent. An example is an encryption key used in conjunction with the transformation to scramble the message before transmission and unscramble it on reception.<sup>4</sup>

A trusted third party may be needed to achieve secure transmission. For example, a third party may be responsible for distributing the secret information to the two principals while keeping it from any opponent. Or a third party may be needed to arbitrate disputes between the two principals concerning the authenticity of a message transmission.

This general model shows that there are four basic tasks in designing a particular security service:

1. Design an algorithm for performing the security-related transformation. The algorithm should be such that an opponent cannot defeat its purpose.
2. Generate the secret information to be used with the algorithm.
3. Develop methods for the distribution and sharing of the secret information.
4. Specify a protocol to be used by the two principals that makes use of the security algorithm and the secret information to achieve a particular security service.

Much of this book concentrates on the types of security mechanisms and services that fit into the model shown in Figure 1.3. However, there are other security-related situations of interest that do not neatly fit this model but that are considered in this book. A general model of these other situations is illustrated by Figure 1.4, which reflects a concern for protecting an information system from unwanted access. Most readers are familiar with the concerns caused by the existence of hackers, who attempt to penetrate systems that can be accessed over a network. The hacker can be someone who, with no malign intent, simply gets satisfaction from breaking and entering a computer system. Or, the intruder can be a disgruntled employee who wishes to do damage, or a criminal who seeks to exploit computer assets for financial gain (e.g., obtaining credit card numbers or performing illegal money transfers).

Another type of unwanted access is the placement in a computer system of logic that exploits vulnerabilities in the system and that can affect application programs as well as utility programs, such as editors and compilers. Two kinds of threats can be presented by programs:

- **Information access threats** intercept or modify data on behalf of users who should not have access to that data.
- **Service threats** exploit service flaws in computers to inhibit use by legitimate users.

---

<sup>4</sup>Chapter 6 discusses a form of encryption, known as public-key encryption, in which only one of the two principals needs to have the secret information.

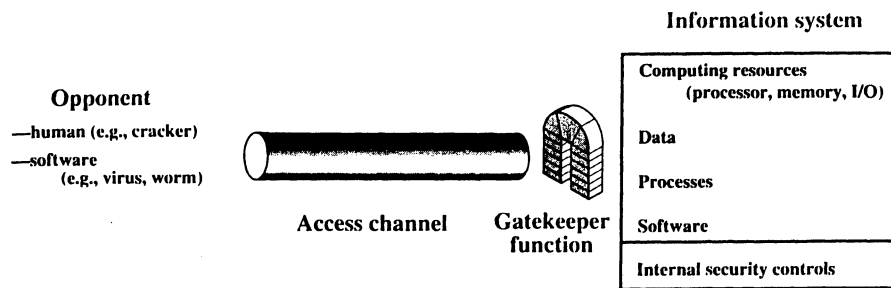


Figure 1.4 Network Access Security Model.

Viruses and worms are two examples of software attacks. Such attacks can be introduced into a system by means of a diskette that contains the unwanted logic concealed in otherwise useful software. They can also be inserted into a system across a network; this latter mechanism is of more concern in network security.

The security mechanisms needed to cope with unwanted access fall into two broad categories (see Figure 1.4). The first category might be termed a gatekeeper function. It includes password-based login procedures that are designed to deny access to all but authorized users and screening logic that is designed to detect and reject worms, viruses, and other similar attacks. Once access is gained, by either an unwanted user or unwanted software, the second line of defense consists of a variety of internal controls that monitor activity and analyze stored information in an attempt to detect the presence of unwanted intruders.

## 1.5 OUTLINE OF THIS BOOK

This chapter serves as an introduction to the entire book. A brief synopsis of the remaining chapters follows.

### Conventional Encryption: Classical Techniques

By far the most important automated tool for network and communications security is encryption. Two forms of encryption are in common use: conventional, or symmetric, encryption and public-key, or asymmetric, encryption. Chapter 2 describes classical conventional encryption techniques. It provides a gentle and interesting introduction to cryptography and cryptanalysis and highlights important concepts.

### Conventional Encryption: Modern Techniques and Algorithms

Chapter 3 introduces the principles of modern symmetric cryptography, with an emphasis on the most widely used encryption technique, the data encryption standard (DES). The chapter includes a discussion of design considerations and cryptanalysis and introduces the Feistel cipher, which is the basic structure of most modern conventional encryption schemes. Chapter 4 extends the discussion to include some of the most important contemporary block cipher algorithms.

### **Confidentiality Using Conventional Encryption**

Beyond questions dealing with the actual construction of a conventional encryption algorithm, a number of design issues relate to the use of conventional encryption to provide confidentiality. Chapter 5 surveys the most important of these issues. The chapter includes a discussion of end-to-end versus link encryption, techniques for achieving traffic confidentiality, and key distribution techniques. An important related topic, random number generation, is also addressed.

### **Public-Key Encryption**

After conventional encryption, the other major form of encryption is public-key encryption, which has revolutionized communications security. Chapter 6 introduces public-key encryption and concentrates on its use to provide confidentiality. The Rivest-Shamir-Adleman (RSA) algorithm is examined in detail, and the issue of key management is reconsidered. The chapter also covers the widely used Diffie-Hellman key exchange technique and looks at a more recent public-key approach based on elliptic curves.

### **Introduction to Number Theory**

Most public-key schemes are based on number theory. While the reader can take the number-theoretic results on faith, it is useful to have a basic grasp of the concepts of number theory. Chapter 7 provides an overview and numerous examples to clarify the concepts.

### **Authentication and Hash Functions**

Of equal importance to confidentiality as a security measure is authentication. At a minimum, message authentication assures that a message comes from the alleged source. In addition, authentication can include protection against modification, delay, replay, and reordering. Chapter 8 begins with an analysis of the requirements for authentication and then provides a systematic presentation of approaches to authentication. A key element of authentication schemes is the use of an authenticator, usually either a message authentication code (MAC) or a hash function. Design considerations for both of these types of algorithms are examined, and several specific examples are analyzed. Chapter 9 extends the discussion to include some of the most important contemporary hash functions and the Internet-standard MAC known as HMAC.

### **Digital Signatures and Authentication Protocols**

An important type of authentication is the digital signature. Chapter 10 examines the techniques used to construct digital signatures and looks at an important standard, the Digital Signature Standard (DSS).

The various authentication techniques based on digital signatures are building blocks in putting together authentication algorithms. The design of such algorithms involves the analysis of subtle attacks that can defeat many apparently secure protocols. This issue is also addressed in Chapter 8.

## **Authentication Applications**

Chapter 11 is a survey of two of the most important authentication specifications in current use. Kerberos is an authentication protocol based on conventional encryption that has received widespread support and is used in a variety of systems. X.509 specifies an authentication algorithm and defines a certificate facility. The latter enables users to obtain certificates of public keys so that a community of users can have confidence in the validity of the public keys. This facility is employed as a building block in a number applications.

## **Electronic Mail Security**

The most heavily used distributed application is electronic mail, and there is increasing interest in providing authentication and confidentiality services as part of an electronic mail facility. Chapter 12 looks at the two approaches likely to dominate electronic mail security in the near future. Pretty Good Privacy (PGP) is a widely used scheme that does not depend on any organization or authority. Thus, it is as well suited to individual, personal use as it is to incorporation in network configurations operated by organizations. S/MIME (Secure/Multipurpose Internet Mail Extension) was developed specifically to be an Internet Standard.

## **IP Security**

The Internet Protocol (IP) is the central element in the Internet and private intranets. Security at the IP level, accordingly, is important to the design of any internetwork-based security scheme. Chapter 13 looks at the IP security scheme that has been developed to operate both with the current IP and the emerging next-generation IP, known as IPv6.

## **Web Security**

The explosive growth in the use of the World Wide Web for electronic commerce and to disseminate information has generated the need for strong Web-based security. Chapter 14 provides a survey of this important new security area, and looks at two key standards: Secure Sockets Layer (SSL) and Secure Electronic Transaction (SET).

## **Intruders, Viruses, and Worms**

Chapter 15 examines a variety of information access and service threats presented by hackers and programs that exploit vulnerabilities in network-based computing systems. The chapter begins with a discussion of the types of attacks that can be made by unauthorized users, or intruders, and analyzes various approaches to prevention and detection. Programmed threats, including viruses, are then discussed.

## **Firewalls**

A standard approach to the protection of local computer assets from external threats is the use of a firewall. Chapter 16 discusses the principles of firewall design and looks at specific techniques.

## 1.6 RECOMMENDED READING

[PFLE97] provides a good introduction to both computer and network security. Another excellent survey is [ABRA95].

ABRA95 Abrams, M.; Jajodia, S.; and Podell, H., eds. *Information Security: An Integrated Collection of Essays*. Los Alamitos, CA: IEEE Computer Society Press, 1995.

PFLE97 Pfleeger, C. *Security in Computing*. Upper Saddle River, NJ: Prentice Hall, 1997.

## APPENDIX 1A INTERNET AND WEB RESOURCES

There are a number of resources available on the Internet and the Web to support this book and to help one keep up with developments in this field.

### Web Sites for This Book

A special Web page has been set up for this book at <http://www.shore.net/~ws/Security2e.html>. The site includes the following:

- Links to other Web sites, including the sites listed in this appendix, provide a gateway to relevant resources on the Web.
- On-line transparency masters are provided of most of the figures in this book in PDF (Adobe Acrobat) format.
- Sign-up information is provided for the book's Internet mailing list.
- I also hope to include links to home pages for courses based on the book; these pages may be useful to other instructors in providing ideas about how to structure their course.

As soon as any typos or other errors are discovered, an errata list for this book will be available at <http://www.shore.net/~ws>. The file will be updated as needed. Please e-mail any errors that you spot to [ws@shore.net](mailto:ws@shore.net). Errata sheets for my other books are at the same Web site, as well as discount ordering information for the books.

### Other Web Sites

There are numerous Web sites that provide some sort of information related to the topics of this book. Here is a sample:<sup>5</sup>

- **COAST:** Comprehensive set of links related to cryptography and network security.

---

<sup>5</sup>Because the URLs for Web sites tend to change frequently, I have not included these in the book. For all of the Web sites listed in the book, the appropriate link can be found at this book's web site.



- **IETF Security Area:** Keeps up-to-date on Internet security standardization efforts.
- **Computer and Network Security Reference Index:** A good index to vendor and commercial products, FAQs, newsgroup archives, papers, and other Web sites.
- **The Cryptography FAQ:** Lengthy and worthwhile FAQ covering all aspects of cryptography.
- **Tom Dunigan's Security Page:** An excellent list of pointers to cryptography and network security Web sites.
- **IEEE Technical Committee on Security and Privacy:** Copies of their newsletter, information on IEEE-related activities.

In subsequent chapters, pointers to more specific Web sites can be found in the "Recommended Reading" section.

### USENET Newsgroups

A number of USENET newsgroups are devoted to some aspect of network security or cryptography. As with virtually all USENET groups, there is a high noise-to-signal ratio, but it is worth experimenting to see if any meet your needs. The most relevant are as follows:

- **sci.crypt.:** A general discussion of cryptology and related topics.
- **sci.crypt.research:** The best group to follow. This is a moderated newsgroup that deals with research topics; postings must have some relationship to the technical aspects of cryptology.
- **alt.security:** A general discussion of security topics.
- **comp.security.firewalls:** A general discussion of firewall products and technology.



**PART  
ONE**

# Conventional Encryption



# CHAPTER 2

---

## CONVENTIONAL ENCRYPTION: CLASSICAL TECHNIQUES

*Many savages at the present day regard their names as vital parts of themselves, and therefore take great pains to conceal their real names, lest these should give to evil-disposed persons a handle by which to injure their owners.*

*—The Golden Bough, Sir James George Frazer*

Conventional encryption, also referred to as symmetric encryption or single-key encryption, was the only type of encryption in use prior to the development of public-key encryption.<sup>1</sup> It remains by far the most widely used of the two types of encryption. This chapter and the next two study a variety of conventional encryption algorithms; some of the issues surrounding the use of keys for conventional encryption are discussed in Chapter 5.

We begin with a look at a general model for the conventional encryption process; this will enable us to understand the context within which the algorithms are used. Next, we examine a variety of algorithms in use before the computer era. Chapter 3 examines the most widely used contemporary encryption algorithm: DES.

---

<sup>1</sup>Public-key encryption was first described in the open literature in 1976; the National Security Agency (NSA) claims to have discovered it some years earlier.

## 2.1 CONVENTIONAL ENCRYPTION MODEL

Figure 2.1 illustrates the conventional encryption process. The original intelligible message, referred to as *plaintext*, is converted into apparently random nonsense, referred to as *ciphertext*. The encryption process consists of an algorithm and a key. The key is a value independent of the plaintext. The algorithm will produce a different output depending on the specific key being used at the time. Changing the key changes the output of the algorithm.

Once the ciphertext is produced, it may be transmitted. Upon reception, the ciphertext can be transformed back to the original plaintext by using a decryption algorithm and the same key that was used for encryption.

The security of conventional encryption depends on several factors. First, the encryption algorithm must be powerful enough that it is impractical to decrypt a message on the basis of the ciphertext alone. Beyond that, the security of conventional encryption depends on the secrecy of the key, not the secrecy of the algorithm. That is, it is assumed that it is impractical to decrypt a message on the basis of the ciphertext *plus* knowledge of the encryption/decryption algorithm. In other words, we do not need to keep the algorithm secret; we need to keep only the key secret.

This feature of conventional encryption is what makes it feasible for widespread use. The fact that the algorithm need not be kept secret means that manufacturers can and have developed low-cost chip implementations of data encryption algorithms. These chips are widely available and incorporated into a number of products. With the use of conventional encryption, the principal security problem is maintaining the secrecy of the key.

Let us take a closer look at the essential elements of a conventional encryption scheme, using Figure 2.2. A source produces a message in plaintext,  $X = [X_1, X_2, \dots, X_M]$ . The  $M$  elements of  $X$  are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. Nowadays, the binary alphabet  $\{0, 1\}$  is typically used. For encryption, a key of the form  $K = [K_1, K_2, \dots, K_J]$  is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination.

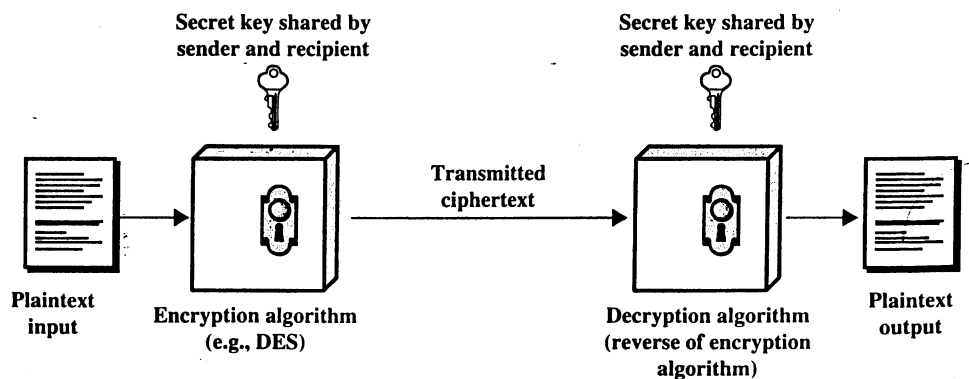


Figure 2.1 Simplified Model of Conventional Encryption.

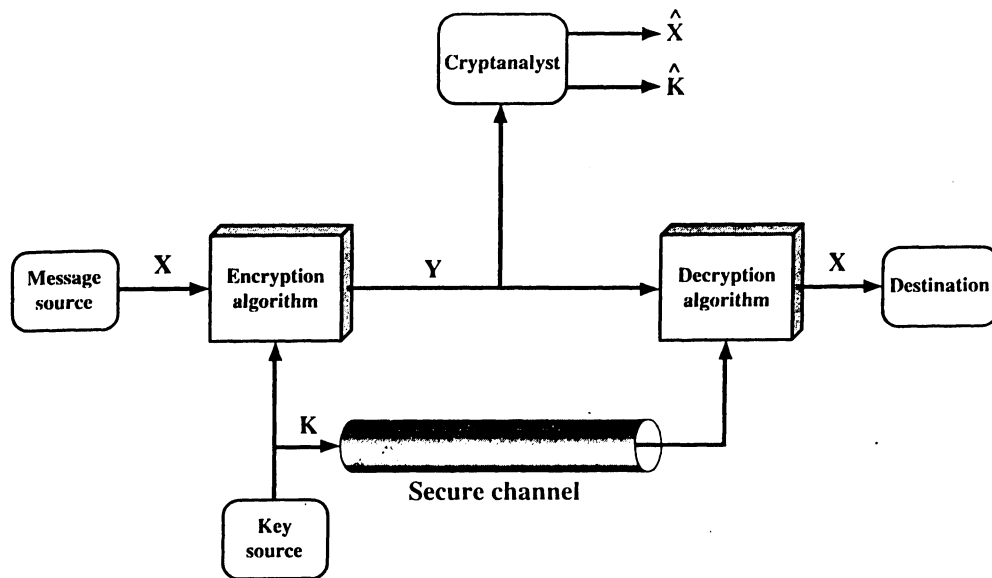


Figure 2.2 Model of Conventional Cryptosystem.

With the message  $X$  and the encryption key  $K$  as input, the encryption algorithm forms the ciphertext  $Y = [Y_1, Y_2, \dots, Y_N]$ . We can write this as

$$Y = E_K(X)$$

This notation indicates that  $Y$  is produced by using encryption algorithm  $E$  as a function of the plaintext  $X$ , with the specific function determined by the value of the key  $K$ .

The intended receiver, in possession of the key, is able to invert the transformation:

$$X = D_K(Y)$$

An opponent, observing  $Y$  but not having access to  $K$  or  $X$ , may attempt to recover  $X$  or  $K$  or both  $X$  and  $K$ . It is assumed that the opponent knows the encryption ( $E$ ) and decryption ( $D$ ) algorithms. If the opponent is interested in only this particular message, then the focus of the effort is to recover  $X$  by generating a plaintext estimate  $\hat{X}$ . Often, however, the opponent is interested in being able to read future messages as well, in which case an attempt is made to recover  $K$  by generating an estimate  $\hat{K}$ .

### Cryptography

Cryptographic systems are generically classified along three independent dimensions:

1. **The type of operations used for transforming plaintext to ciphertext.** All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext (bit, letter, group of bits or letters) is

mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (that is, that all operations be reversible). Most systems, referred to as product systems, involve multiple stages of substitutions and transpositions.

2. **The number of keys used.** If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver each uses a different key, the system is referred to as asymmetric, two-key, or public-key encryption.
3. **The way in which the plaintext is processed.** A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time, as it goes along.

### Cryptanalysis

The process of attempting to discover X or K or both is known as *cryptanalysis*. The strategy used by the cryptanalyst depends on the nature of the encryption scheme and the information available to the cryptanalyst.

Table 2.1 summarizes the various types of cryptanalytic attacks based on the amount of information known to the cryptanalyst. The most difficult problem is presented when all that is available is the *ciphertext only*. In some cases, not even the encryption algorithm is known, but in general we can assume that the opponent does know the algorithm used for encryption. One possible attack under these circumstances is the brute-force approach of trying all possible keys. If the key space is very large, this becomes impractical. Thus, the opponent must rely on an analysis

**Table 2.1** Types of Attacks on Encrypted Messages

Type of Attack	Known to Cryptanalyst
Ciphertext only	<ul style="list-style-type: none"> <li>• Encryption algorithm</li> <li>• Ciphertext to be decoded</li> </ul>
Known plaintext	<ul style="list-style-type: none"> <li>• Encryption algorithm</li> <li>• Ciphertext to be decoded</li> <li>• One or more plaintext-ciphertext pairs formed with the secret key</li> </ul>
Chosen plaintext	<ul style="list-style-type: none"> <li>• Encryption algorithm</li> <li>• Ciphertext to be decoded</li> <li>• Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key</li> </ul>
Chosen ciphertext	<ul style="list-style-type: none"> <li>• Encryption algorithm</li> <li>• Ciphertext to be decoded</li> <li>• Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key</li> </ul>
Chosen text	<ul style="list-style-type: none"> <li>• Encryption algorithm</li> <li>• Ciphertext to be decoded</li> <li>• Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key</li> <li>• Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key</li> </ul>



of the ciphertext itself, generally applying various statistical tests to it. To use this approach, the opponent must have some general idea of the type of plaintext that is concealed, such as English or French text, an MS-DOS EXE file, a Java source listing, an accounting file, and so on.

The ciphertext-only attack is the easiest to defend against because the opponent has the least amount of information to work with. In many cases, however, the analyst has more information. The analyst may be able to capture one or more plaintext messages as well as their encryptions. Or the analyst may know that certain plaintext patterns will appear in a message. For example, a file that is encoded in the Postscript format always begins with the same pattern, or there may be a standardized header or banner to an electronic funds transfer message, and so on. All these are examples of *known plaintext*. With this knowledge, the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed.

Closely related to the known-plaintext attack is what might be referred to as a probable-word attack. If the opponent is working with the encryption of some general prose message, he or she may have little knowledge of what is in the message. However, if the opponent is after some very specific information, then parts of the message may be known. For example, if an entire accounting file is being transmitted, the opponent may know the placement of certain key words in the header of the file. As another example, the source code for a program developed by Corporation X might include a copyright statement in some standardized position.

If the analyst is able somehow to get the source system to insert into the system a message chosen by the analyst, then a *chosen-plaintext* attack is possible. An example of this strategy is differential cryptanalysis, explored Chapter 3. In general, if the analyst is able to choose the messages to encrypt, the analyst may deliberately pick patterns that can be expected to reveal the structure of the key.

Table 2.1 lists two other types of attack: chosen ciphertext and chosen text. These are less commonly employed as cryptanalytic techniques but are nevertheless possible avenues of attack.

Only relatively weak algorithms fail to withstand a ciphertext-only attack. Generally, an encryption algorithm is designed to withstand a known-plaintext attack.

Two more definitions are worthy of note. An encryption scheme is **unconditionally secure** if the ciphertext generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext is available. That is, no matter how much time an opponent has, it is impossible for him or her to decrypt the ciphertext, simply because the required information is not there. With the exception of a scheme known as the one-time pad (described later in this chapter), there is no encryption algorithm that is unconditionally secure. Therefore, all that the users of an encryption algorithm can strive for is an algorithm that meets one or both of the following criteria:

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

An encryption scheme is said to be **computationally secure** if the foregoing two criteria are met. The rub is that it is very difficult to estimate the amount of effort required to cryptanalyze ciphertext successfully.

**Table 2.2** Average Time Required for Exhaustive Key Search

Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/ $\mu$ s	Time required at $10^6$ encryptions/ $\mu$ s
32	$2^{32} = 4.3 \times 10^9$	$2^{31} \mu\text{s} = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55} \mu\text{s} = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127} \mu\text{s} = 5.4 \times 10^{24}$ years	$5.4 \times 10^{18}$ years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \mu\text{s} = 6.4 \times 10^{12}$ years	$6.4 \times 10^6$ years

As a first cut, we can consider the time required to use a brute-force approach, which simply involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. Table 2.2 shows how much time is involved for various key spaces. Results are shown for three binary key sizes. The 56-bit key size is used with the DES (Data Encryption Standard) algorithm. Results are also shown for what are called substitution codes that use a 26-character key (discussed later), in which all possible permutations of the 26 characters serve as keys. For each key size, the results are shown assuming that it takes 1  $\mu$ s to perform a single decryption, which is a reasonable order of magnitude for today's machines. With the use of massively parallel organizations of microprocessors, it may be possible to achieve processing rates many orders of magnitude greater. The final column of Table 2.2 considers the results for a system that can process 1 million keys per microsecond. As you can see, at this performance level, DES can no longer be considered computationally secure.

All forms of cryptanalysis for conventional encryption schemes are designed to exploit the fact that traces of structure or pattern in the plaintext may survive encryption and be discernible in the ciphertext. This will become clear as we examine various conventional encryption schemes in this chapter. We will see in Chapter 6 that cryptanalysis for public-key schemes proceeds from a fundamentally different premise—namely, that the mathematical properties of the pair of keys may make it possible for one of the two keys to be deduced from the other.

## 2.2 STEGANOGRAPHY

We begin with a discussion of a technique that is, strictly speaking, not encryption—namely, steganography.

A plaintext message may be hidden in one of two ways. The methods of steganography conceal the existence of the message, whereas the methods of cryptography render the message unintelligible to outsiders by various transformations of the text.<sup>2</sup> Table 2.3, taken from [KAHN96], indicates the place of steganography in the spectrum of security techniques. In Kahn's terminology, *security* is defined as

<sup>2</sup>*Steganography* was an obsolete word that was revived by David Kahn and given the meaning it has today [KAHN96].

**Table 2.3** Signal Security and Intelligence Techniques

Signal Security	Signal Intelligence
Communication Security	Communication Intelligence
Steganography (invisible inks, open codes, messages in hollow heels)	Interception
Cryptography (codes and ciphers)	Cryptanalysis
Traffic Security (call-sign changes, dummy messages)	Traffic analysis (message-flow studies, radio fingerprinting)
Electronic Security	Electronic Intelligence
Emission security (shifting of frequencies, spread spectrum)	Electronic reconnaissance (eavesdropping)
Counter-countermeasures ("looking through" jammed radar)	Countermeasures (jamming radar and false radio echoes)

methods of protecting information and *intelligence* is defined as methods of retrieving information.

A simple form of steganography, but one that is time-consuming to construct, is one in which an arrangement of words or letters within an apparently innocuous text spells out the real message. For example, the sequence of first letters of each word of the overall message spells out the hidden message. Figure 2.3 shows an example in which a subset of the words of the overall message is used to convey the hidden message.

Various other techniques have been used historically; some examples are the following [MYER91]:

- **Character marking:** Selected letters of printed or typewritten text are over-written in pencil. The marks are ordinarily not visible unless the paper is held at an angle to bright light.
- **Invisible ink:** A number of substances can be used for writing but leave no visible trace until heat or some chemical is applied to the paper.
- **Pin punctures:** Small pin punctures on selected letters are ordinarily not visible unless the paper is held up in front of a light.
- **Typewriter correction ribbon:** Used between lines typed with a black ribbon, the results of typing with the correction tape are visible only under a strong light.

Although these techniques may seem archaic, they have contemporary equivalents. [WAYN93] proposes hiding a message by using the least significant bits of frames on a CD. For example, the Kodak Photo CD format's maximum resolution is 2048 by 3072 pixels, with each pixel containing 24 bits of RGB color information. The least significant bit of each 24-bit pixel can be changed without greatly affecting the quality of the image. The result is that you can hide a 2.3-megabyte message in a single digital snapshot. There are now a number of software packages available that take this type of approach to steganography [JOHN97].

Steganography has a number of drawbacks when compared to encryption. It requires a lot of overhead to hide a relatively few bits of information, although using

*3rd March*

*Dear George,*

*Greetings to all at Oxford. Many thanks for your letter and for the Summer examination package. All Entry Forms and Fees Forms should be ready for final despatch to the Syndicate by Friday 20th or at the very latest, I'm told, by the 21st. Admin has improved here, though there's room for improvement still; just give us all two or three more years and we'll really show you! Please don't let these wretched 16+ proposals destroy your basic O and A pattern. Certainly this sort of change, if implemented immediately, would bring chaos.*

*Sincerely yours,*

**Figure 2.3** A Puzzle for Inspector Morse. *SOURCE:* The Silent World of Nicholas Quinn, by Colin Dexter.

some scheme like that proposed in the preceding paragraph may make it more effective. Also, once the system is discovered, it becomes virtually worthless. This problem, too, can be overcome if the insertion method depends on some sort of key (e.g., see Problem 2.3). Alternatively, a message can be first encrypted and then hidden using steganography.

The advantage of steganography is that it can be employed by parties who have something to lose should the fact of their secret communication (not necessarily the content) be discovered. Encryption flags traffic as important or secret or may identify the sender or receiver as someone with something to hide.

## 2.3 CLASSICAL ENCRYPTION TECHNIQUES

In this section, we examine a sampling of what might be called classical encryption

to conventional encryption used today and the types of cryptanalytic attacks that must be anticipated.

First, we examine the two basic building blocks of all encryption techniques: substitution and transposition. Finally, we discuss systems that combine both substitution and transposition.

### Substitution Techniques

A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols.<sup>3</sup> If the plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns.

#### Caesar Cipher

The earliest known use of a substitution cipher, and the simplest, was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet. For example,

```
plain:  meet me after the toga party
cipher: PHHW PH DIWHU WKH WRJD SDUWB
```

Note that the alphabet is wrapped around, so that the letter following Z is A. We can define the transformation by listing all possibilities, as follows:

```
plain:  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
cipher: 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
```

If we assign a numerical equivalent to each letter ( $a = 1$ ,  $b = 2$ , etc.), then the algorithm can be expressed as follows. For each plaintext letter  $p$ , substitute the ciphertext letter  $C$ :

$$C = E(p) = (p + 3) \bmod(26)$$

A shift may be of any amount, so that the general Caesar algorithm is

$$C = E(p) = (p + k) \bmod(26)$$

where  $k$  takes on a value in the range 1 to 25. The decryption algorithm is simply

$$p = D(C) = (C - k) \bmod(26)$$

If it is known that a given ciphertext is a Caesar cipher, then a brute-force cryptanalysis is easily performed: Simply try all the 25 possible keys. Figure 2.4 shows the results of applying this strategy to the example ciphertext. In this case, the plaintext leaps out as occupying the third line.

<sup>3</sup>When letters are involved, the following conventions are used in this book. Plaintext is always in lowercase; ciphertext is in uppercase; key values are in italicized lowercase.

KEY	PHHW PH DIWHU WKH WRJD SDUWB
1	oggv og chvgt vjg vqic rctva
2	nffu nf bgufs uif uphb qbsuz
3	meet me after the toga party
4	ldds ld zesdq sgd snfz ozqsx
5	kccr kc ydrpc rfc rmey nyprw
6	jbbq jb xcqbo qeb qldx mxoqv
7	iaap ia wbpan pda pkcw lwnpu
8	hzzo hz vaozm ocz objv kvmot
9	gyyn gy uznyl nby niau julns
10	fxxm fx tymxk max mhzt itkmr
11	ewwl ew sxlwj lzw lgys hsjlq
12	dvvk dv rwkvi kyv kfxr grikp
13	cuuj cu qvjuh jxu jewq fqhjo
14	btti bt puitg iwt idvp epgin
15	assh as othsf hvs hcuo dofhm
16	zrrg zr nsgre gur gbtc cnegl
17	yqqf yq mrfqd ftq fasm bmdfk
18	xppe xp lqepc esp ezrl alcej
19	wood wo kpdob dro dyqk zkbdi
20	vnnc vn jocna cqn expj yjach
21	ummb um inbmz bpm bwoi xizbg
22	tlla tl hmaly aol avnh whyaf
23	skkz sk glzxx znk zumg vxgze
24	rjjy rj fkyjw ymj ytlf ufwyd
25	qiix qi ejxiv xli xske tevxc

**Figure 2.4** Brute-Force Cryptanalysis of Caesar Cipher.

Three important characteristics of this problem enabled us to use a brute-force cryptanalysis:

1. The encryption and decryption algorithms are known.
2. There are only 25 keys to try.
3. The language of the plaintext is known and easily recognizable.

In most networking situations, we can assume that the algorithms are known. What generally makes brute-force cryptanalysis impractical is the use of an algorithm that employs a large number of keys. For example, the DES algorithm, examined in Chapter 3, makes use of a 56-bit key, giving a key space of  $2^{56}$  or greater than  $7 \times 10^{16}$  possible keys.

The third characteristic is also significant. If the language of the plaintext is unknown, then plaintext output may not be recognizable. Furthermore, the input may be abbreviated or compressed in some fashion, again making recognition

~+Wu"- Ω-O)≤4{∞‡. ë~Ω%ràu·-í ∅-Z-  
 Ú≠2Ô#Åæð æ<q7,Ωn·@3NÔÚ Ėz'Y-f∞Í[±Ů\_ èΩ,<NO-±«~xã Åäfeü3Å  
 x)ö\$sk°Å  
 \_yí ^ΔÉ] ,π J/'iTê&1 'c<uΩ-  
 ÄD(G WÄC~y\_iöÄW PÔ1<îŮ†ç},π;~î^üÑπ~≈~L~9OgflO~&Æ≤ ~≤ ∅Ô\$~:  
 ~Æ!SGqèvo^ ú\,S>h<-\*6ø‡%x'~|fiÓ#≈~my%~>ñP<,fi Áj Å0¿~Zù-  
 Ω"Ö-6Æÿ{%,ΩÊó ,i π+Áî·úO2çSÿ'O-  
 2Äñßi /@^"ΠK°PÆπ,úé^'3Σ~ö^ÔZî"Y~ÿΩæY> Ω+eô/'<KÆ¿\*+~"≤ú~  
 B ZøK~Qßÿüf, !ðñîzss/]>ÈQ ü

**Figure 2.5** Sample of Compressed Text.

difficult. For example, Figure 2.5 shows a portion of a text file compressed using an algorithm called ZIP. If this file is then encrypted with a simple substitution cipher (expanded to include more than just 26 alphabetic characters), then the plaintext may not be recognized when it is uncovered in the brute-force cryptanalysis.

### Monoalphabetic Ciphers

With only 25 possible keys, the Caesar cipher is far from secure. A dramatic increase in the key space can be achieved by allowing an arbitrary substitution. Recall the assignment for the Caesar cipher:

```
plain:   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
cipher:  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
```

If, instead, the "cipher" line can be any permutation of the 26 alphabetic characters, then there are 26! or greater than  $4 \times 10^{26}$  possible keys. This is 10 orders of magnitude greater than the key space for DES and would seem to eliminate brute-force techniques for cryptanalysis.

There is, however, another line of attack. If the cryptanalyst knows the nature of the plaintext (e.g., noncompressed English text), then the analyst can exploit the regularities of the language. To see how such a cryptanalysis might proceed, we give a partial example here that is adapted from one in [SINK66]. The ciphertext to be solved is

```
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDÈMETSXAIZ
VUEPHZHMDZSHZOWSFPAPPDTSVPPQUZWMXUZHUSX
EPYEPOPDZSZUFPOMBZWPFPUPZHMDJUDTMOHMQ
```

As a first step, the relative frequency of the letters can be determined and compared to a standard frequency distribution for English, such as is shown in Figure 2.6 (based on [SEBE89]). If the message were long enough, this technique alone might be sufficient, but because this is a relatively short message, we cannot expect an exact match. In any case, the relative frequency of the letters in the ciphertext (in percentages) are as follows:

P	13.33	H	5.83	F	3.33	B	1.67	C	0.00
Z	11.67	D	5.00	W	3.33	G	1.67	K	0.00
S	8.33	E	5.00	Q	2.50	Y	1.67	L	0.00
U	8.33	V	4.17	T	2.50	I	0.83	N	0.00
O	7.50	X	4.17	A	1.67	J	0.83	R	0.00
M	6.67								

Comparing this breakdown with Figure 2.6, it seems likely that cipher letters P and Z are the equivalents of plain letters e and t, but it is not certain which is which. The letters S, U, O, M, and H are all of relatively high frequency and probably correspond to plain letters from the set {r, n, i, o, a, s}. The letters with the lowest frequencies (namely, A, B, G, Y, I, J) are likely included in the set {w, v, b, k, x, q, j, z}.

There are a number of ways to proceed at this point. We could make some tentative assignments and start to fill in the plaintext to see if it looks like a reasonable “skeleton” of a message. A more systematic approach is to look for other regularities. For example, certain words may be known to be in the text. Or we could look for repeating sequences of cipher letters and try to deduce their plaintext equivalents.

A powerful tool is to look at the frequency of two-letter combinations, known as digrams. A table similar to Figure 2.6 could be drawn up showing the relative frequency of digrams. The most common such digram is th. In our ciphertext, the most common digram is ZW, which appears three times. So we make the correspondence of Z with t and W with h. Then, by our earlier hypothesis, we can equate P with e. Now notice that the sequence ZWP appears in the ciphertext, and we can now translate that sequence as “the.” This is the most frequent trigram (three-letter combination) in English, which seems to indicate that we are on the right track.

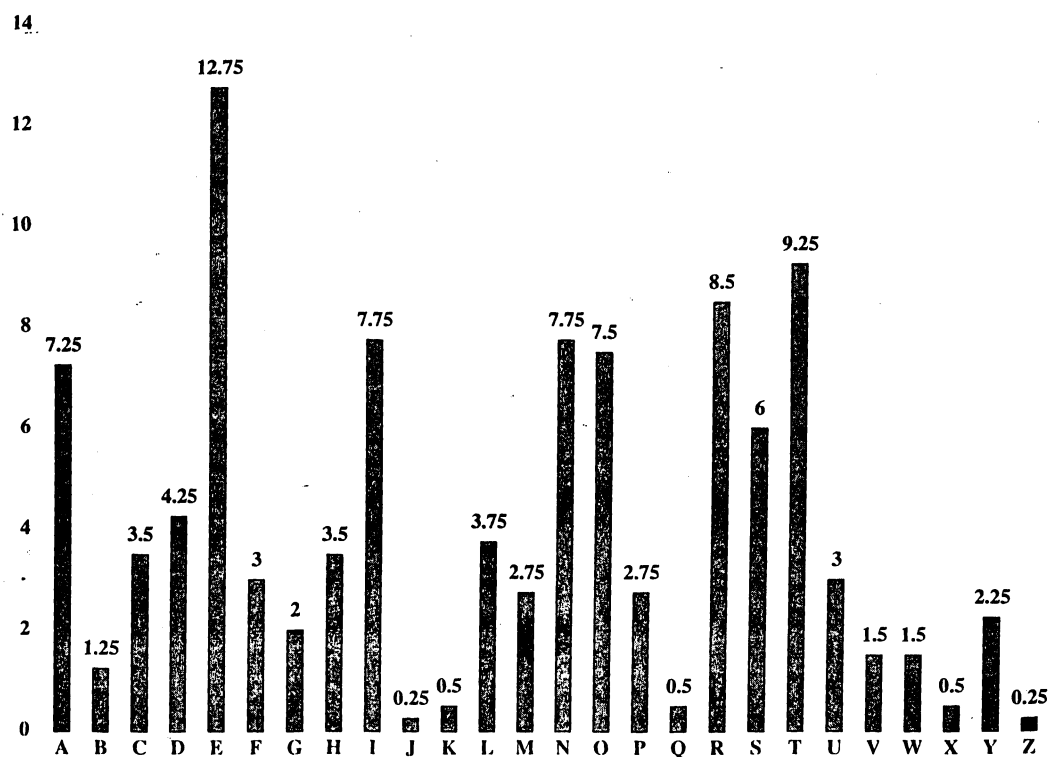


Figure 2.6 Relative Frequency of Letters in English Text.



Next, notice the sequence ZWSZ in the first line. We do not know that these four letters form a complete word, but if they do, it is of the form th\_t. Therefore, S equates with a.

So far, then, we have

```

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
t a      e e t e a t h a t e e a      a

VUEPHZHMZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
e t    t a t h a e e e a e t h    t a

EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
e e e t a t e    t h e    t

```

Only four letters have been identified, but already we have quite a bit of the message. Continued analysis of frequencies plus trial and error should easily yield a solution from this point. The complete plaintext, with spaces added between words, follows:

```

it was disclosed yesterday that several informal but
direct contacts have been made with political
representatives of the viet cong in moscow

```

Monoalphabetic ciphers are easy to break because they reflect the frequency data of the original alphabet. A countermeasure is to provide multiple substitutes, known as homophones, for a single letter. For example, the letter e could be assigned a number of different cipher symbols, such as 16, 74, 35, and 21, with each homophone used in rotation, or randomly. If the number of symbols assigned to each letter is proportional to the relative frequency of that letter, then single-letter frequency information is completely obliterated. The great mathematician Carl Friedrich Gauss believed that he had devised an unbreakable cipher using homophones. However, even with homophones, each element of plaintext affects only one element of ciphertext, and multiple-letter patterns (e.g., digram frequencies) still survive in the ciphertext, making cryptanalysis relatively straightforward.

Two principal methods are used in substitution ciphers to lessen the extent to which the structure of the plaintext survives in the ciphertext: One approach is to encrypt multiple letters of plaintext, and the other is to use multiple cipher alphabets. We briefly examine each.

### Playfair Cipher

The best-known multiple-letter encryption cipher is the Playfair, which treats digrams in the plaintext as single units and translates these units into ciphertext digrams.<sup>4</sup>

The Playfair algorithm is based on the use of a  $5 \times 5$  matrix of letters constructed using a keyword. Here is an example, solved by Lord Peter Wimsey in Dorothy Sayers's *Have His Carcase*.<sup>5</sup>

<sup>4</sup>This cipher was actually invented by British scientist Sir Charles Wheatstone in 1854, but it bears the name of his friend Baron Playfair of St. Andrews, who championed the cipher at the British foreign office.

<sup>5</sup>Sayers's book provides an absorbing account of a probable-word attack.

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

In this case, the keyword is *monarchy*. The matrix is constructed by filling in the letters of the keyword (minus duplicates) from left to right and from top to bottom, and then filling in the remainder of the matrix with the remaining letters in alphabetic order. The letters I and J count as one letter. Plaintext is encrypted two letters at a time, according to the following rules:

1. Repeating plaintext letters that would fall in the same pair are separated with a filler letter, such as x, so that balloon would be enciphered as ba lx lo on.
2. Plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row circularly following the last. For example, ar is encrypted as RM.
3. Plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the row circularly following the last. For example, mu is encrypted as CM.
4. Otherwise, each plaintext letter is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter. Thus, hs becomes BP and ea becomes IM (or JM, as the encipherer wishes).

The Playfair cipher is a great advance over simple monoalphabetic ciphers. For one thing, whereas there are only 26 letters, there are  $26 \times 26 = 676$  digrams, so that identification of individual digrams is more difficult. Furthermore, the relative frequencies of individual letters exhibit a much greater range than that of digrams, making frequency analysis much more difficult. For these reasons, the Playfair cipher was for a long time considered unbreakable. It was used as the standard field system by the British Army in World War I and still enjoyed considerable use by the U.S. Army and other allied forces during World War II.

Despite this level of confidence in its security, the Playfair cipher is relatively easy to break because it still leaves much of the structure of the plaintext language intact. A few hundred letters of ciphertext are generally sufficient.

One way of revealing the effectiveness of the Playfair and other ciphers is shown in Figure 2.7, based on [SIMM93]. The line labeled *plaintext* plots the frequency distribution of the more than 70,000 alphabetic characters in the *Encyclopaedia Britannica* article on cryptology.<sup>6</sup> This is also the frequency distribution of any monoalphabetic substitution cipher. The plot was developed in the following way: The number of occurrences of each letter in the text was counted and divided by the number of occurrences of the letter e (the most frequently used letter). As a

<sup>6</sup>I am indebted to Gustavus Simmons for providing the plots and explaining their method of construction.

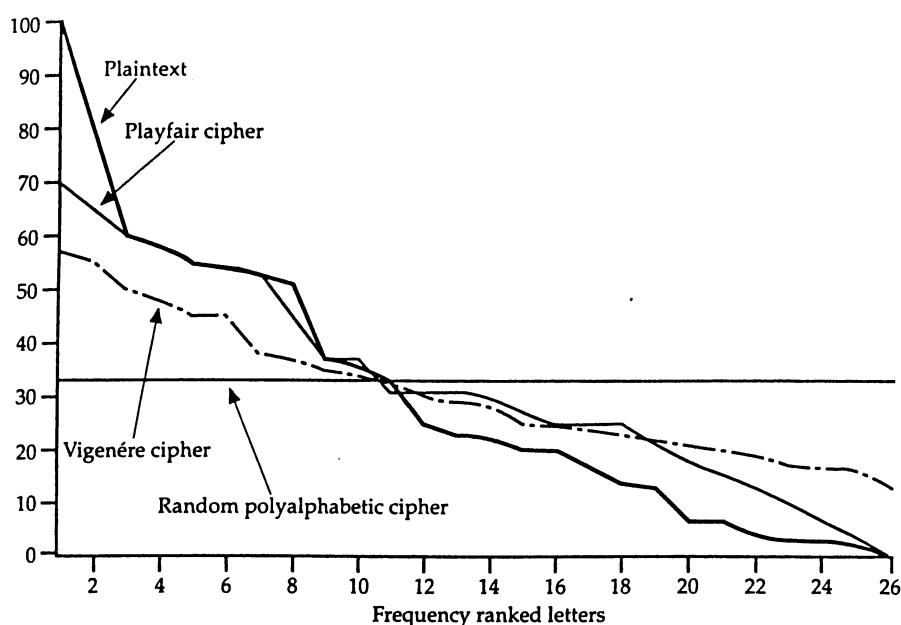


Figure 2.7 Relative Frequency of Occurrence of Letters.

result, e has a relative frequency of 1, t of about 0.76, and so on. The points on the horizontal axis correspond to the letters in order of decreasing frequency.

Figure 2.7 also shows the frequency distribution that results when the text is encrypted using the Playfair cipher. To normalize the plot, the number of occurrences of each letter in the ciphertext was again divided by the number of occurrences of e in the plaintext. The resulting plot therefore shows the extent to which the frequency distribution of letters, which makes it trivial to solve substitution ciphers, is masked by encryption. If the frequency distribution information were totally concealed in the encryption process, the ciphertext plot of frequencies would be flat, and cryptanalysis using ciphertext only would be effectively impossible. As the figure shows, the Playfair cipher has a flatter distribution than does plaintext, but nevertheless it reveals plenty of structure for a cryptanalyst to work with.

### Hill Cipher

Another interesting multiletter cipher is the Hill cipher, developed by the mathematician Lester Hill in 1929. The encryption algorithm takes  $m$  successive plaintext letters and substitutes for them  $m$  ciphertext letters. The substitution is determined by  $m$  linear equations in which each character is assigned a numerical value ( $a = 0, b = 1, \dots, z = 25$ ). For  $m = 3$ , the system can be described as follows:

$$C_1 = (k_{11}P_1 + k_{12}P_2 + k_{13}P_3) \bmod 26$$

$$C_2 = (k_{21}P_1 + k_{22}P_2 + k_{23}P_3) \bmod 26$$

$$C_3 = (k_{31}P_1 + k_{32}P_2 + k_{33}P_3) \bmod 26$$

This can be expressed in term of column vectors and matrices:

$$\begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

or

$$\mathbf{C} = \mathbf{K}\mathbf{P}$$

where  $\mathbf{C}$  and  $\mathbf{P}$  are column vectors of length 3, representing the plaintext and ciphertext, and  $\mathbf{K}$  is a  $3 \times 3$  matrix, representing the encryption key. Operations are performed mod 26.

For example, consider the plaintext "paymoremoney," and use the encryption key

$$\mathbf{K} = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}$$

The first three letters of the plaintext are represented by the vector (15 0 24). Then  $K(15\ 0\ 24) = (375\ 819\ 486) \bmod 26 = (11\ 13\ 18) = \text{LNS}$ . Continuing in this fashion, the ciphertext for the entire plaintext is LNSHDLEWMTRW.

Decryption requires using the inverse of the matrix  $\mathbf{K}$ . The inverse  $\mathbf{K}^{-1}$  of a matrix  $\mathbf{K}$  is defined by the equation  $\mathbf{K}\mathbf{K}^{-1} = \mathbf{K}^{-1}\mathbf{K} = \mathbf{I}$ , where  $\mathbf{I}$  is the matrix that is all zeros except for ones along the main diagonal from upper left to lower right. The inverse of a matrix does not always exist, but when it does, it satisfies the preceding equation. In this case, the inverse is

$$\mathbf{K}^{-1} = \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

This is demonstrated as follows:

$$\begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix} \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix} = \begin{pmatrix} 443 & 442 & 442 \\ 858 & 495 & 780 \\ 494 & 52 & 365 \end{pmatrix} \bmod 26 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

It is easily seen that if the matrix  $\mathbf{K}^{-1}$  is applied to the ciphertext, then the plaintext is recovered. To explain how the inverse of a matrix is determined, we make an exceedingly brief excursion into linear algebra; the interested reader must consult any text on that subject for greater detail. For any square matrix ( $m \times m$ ), the determinant equals the sum of all the products that can be formed by taking exactly one element from each row and exactly one element from each column, with certain of the product terms preceded by a minus sign. For a  $2 \times 2$  matrix

$$\begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix}$$

the determinant is  $k_{11}k_{22} - k_{12}k_{21}$ . For a  $3 \times 3$  matrix, the value of the determinant is  $k_{11}k_{22}k_{33} + k_{21}k_{32}k_{13} + k_{31}k_{12}k_{23} - k_{31}k_{22}k_{13} - k_{21}k_{12}k_{33} - k_{11}k_{32}k_{23}$ . If a square matrix  $\mathbf{A}$  has a nonzero determinant, then the inverse of the matrix is computed as  $[\mathbf{A}^{-1}]_{ij} = (-1)^{i+j} (D_{ij})/\det(\mathbf{A})$ , where  $(D_{ij})$  is the subdeterminant formed by deleting the  $i$ th row and the  $j$ th column of  $\mathbf{A}$  and  $\det(\mathbf{A})$  is the determinant of  $\mathbf{A}$ . For our purposes, all arithmetic is done mod 26.

In general terms, the Hill system can be expressed as follows:

$$\mathbf{C} = \mathbf{E}_K(\mathbf{P}) = \mathbf{K}\mathbf{P}$$

$$\mathbf{P} = \mathbf{D}_K(\mathbf{C}) = \mathbf{K}^{-1}\mathbf{C} = \mathbf{K}^{-1}\mathbf{K}\mathbf{P} = \mathbf{P}$$

As with Playfair, the strength of the Hill cipher is that it completely hides single-letter frequencies. Indeed, with Hill, the use of a larger matrix hides more frequency information. Thus a  $3 \times 3$  Hill cipher hides not only single-letter but two-letter frequency information.

Although the Hill cipher is strong against a ciphertext-only attack, it is easily broken with a known plaintext attack. For an  $m \times m$  Hill cipher, suppose we have  $m$  plaintext-ciphertext pairs, each of length  $m$ . We label the pairs  $\mathbf{P}_j = (p_{1j}, p_{2j}, \dots, p_{mj})$  and  $\mathbf{C}_j = (C_{1j}, C_{2j}, \dots, C_{mj})$  such that  $\mathbf{C}_j = \mathbf{K}\mathbf{P}_j$  for  $1 \leq j \leq m$  and for some unknown key matrix  $\mathbf{K}$ . Now define two  $m \times m$  matrices  $\mathbf{X} = (p_{ij})$  and  $\mathbf{Y} = (C_{ij})$ . Then we can form the matrix equation  $\mathbf{Y} = \mathbf{X}\mathbf{K}$ . If  $\mathbf{X}$  has an inverse, then we can determine  $\mathbf{K} = \mathbf{X}^{-1}\mathbf{Y}$ . If  $\mathbf{X}$  is not invertible, then a new version of  $\mathbf{X}$  can be formed with additional plaintext-ciphertext pairs until an invertible  $\mathbf{X}$  is obtained.

We use an example taken from [STIN95]. Suppose that the plaintext "friday" is encrypted using a  $2 \times 2$  Hill cipher to yield the ciphertext PQCFKU. Thus, we know that  $\mathbf{K}(5 \ 17) = (15 \ 16)$ ;  $\mathbf{K}(8 \ 3) = (2 \ 5)$ ; and  $\mathbf{K}(0 \ 24) = (10 \ 20)$ . Using the first two plaintext-ciphertext pairs, we have

$$\begin{pmatrix} 15 & 16 \\ 2 & 5 \end{pmatrix} = \begin{pmatrix} 5 & 17 \\ 8 & 3 \end{pmatrix} \mathbf{K}$$

The inverse of  $\mathbf{X}$  can be computed:

$$\begin{pmatrix} 5 & 17 \\ 8 & 3 \end{pmatrix}^{-1} = \begin{pmatrix} 9 & 1 \\ 2 & 15 \end{pmatrix}$$

so

$$\mathbf{K} = \begin{pmatrix} 9 & 1 \\ 2 & 15 \end{pmatrix} \begin{pmatrix} 15 & 16 \\ 2 & 5 \end{pmatrix} = \begin{pmatrix} 7 & 19 \\ 8 & 3 \end{pmatrix}$$

This result is verified by testing the remaining plaintext-ciphertext pair.

## Polyalphabetic Ciphers

Another way to improve on the simple monoalphabetic technique is to use different monoalphabetic substitutions as one proceeds through the plaintext message. The general name for this approach is polyalphabetic cipher. All these techniques have the following features in common:

1. A set of related monoalphabetic substitution rules is used.
2. A key determines which particular rule is chosen for a given transformation.

The best-known, and one of the simplest, such algorithms is referred to as the Vigenère cipher. In this scheme, the set of related monoalphabetic substitution rules consists of the 26 Caesar ciphers, with shifts of 0 through 25. Each cipher is denoted by a key letter, which is the ciphertext letter that substitutes for the plaintext letter *a*. Thus, a Caesar cipher with a shift of 3 is denoted by the key value *d*.

To aid in understanding the scheme and to aid in its use, a matrix known as the Vigenère tableau is constructed (Table 2.4). Each of the 26 ciphers is laid out horizontally, with the key letter for each cipher to its left. A normal alphabet for the plaintext runs across the top. The process of encryption is simple: Given a key letter  $x$  and a plaintext letter  $y$ , the ciphertext letter is at the intersection of the row labeled  $x$  and the column labeled  $y$ ; in this case the ciphertext is V.

To encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword. For example, if the keyword is *deceptive*, the message “we are discovered save yourself” is encrypted as follows:

```
key:      deceptive
plaintext: wearediscoveredsaveyourself
ciphertext: ZICVTWQNGRZGVTWAVZHCQYGLMGJ
```

Decryption is equally simple. The key letter again identifies the row. The position of the ciphertext letter in that row determines the column, and the plaintext letter is at the top of that column.

The strength of this cipher is that there are multiple ciphertext letters for each plaintext letter, one for each unique letter of the keyword. Thus, the letter frequency information is obscured. However, not all knowledge of the plaintext structure is lost. For example, Figure 2.7 shows the frequency distribution for a Vigenère cipher with a keyword of length 9. An improvement is achieved over the Playfair cipher, but considerable frequency information remains.

It is instructive to sketch a method of breaking this cipher, because the method reveals some of the mathematical principles that apply in cryptanalysis.

First, suppose that the opponent believes that the ciphertext was encrypted using either monoalphabetic substitution or a Vigenère cipher. A simple test can be made to make a determination. If a monoalphabetic substitution is used, then the statistical properties of the ciphertext should be the same as that of the language of the plaintext. Thus, referring to Figure 2.6, there should be one cipher letter with a relative frequency of occurrence of about 12.75%, one with about 9.25%, and so on. If only a single message is available for analysis, we would not expect an exact match of this small sample with the statistical profile of the plaintext language. Neverthe-

**Table 2.4** The Modern Vigenère Tableau

	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
a	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
b	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
c	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	B
d	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
e	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
f	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
g	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
h	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
i	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
j	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
k	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
l	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
m	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
n	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
o	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	N
p	P	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
q	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
r	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
s	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
t	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
u	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
v	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
w	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
x	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
y	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
z	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

If, on the other hand, a Vigenère cipher is suspected, then progress depends on determining the length of the keyword, as will be seen in a moment. For now, let us concentrate on how the keyword length can be determined. The important insight that leads to a solution is the following: If two identical sequences of plaintext letters occur at a distance that is an integer multiple of the keyword length, they will generate identical ciphertext sequences. In the foregoing example, two instances of the sequence “red” are separated by nine character positions. Consequently, in both cases, r is encrypted using key letter e, e is encrypted using key letter p, and d is encrypted using key letter t. Thus, in both cases the ciphertext sequence is VTW.

An analyst looking at only the ciphertext would detect the repeated sequences VTW at a displacement of 9 and make the assumption that the keyword is either three or nine letters in length. The appearance of VTW twice could be by chance and not reflect identical plaintext letters encrypted with identical key letters. However, if the message is long enough, there will be a number of such repeated ciphertext sequences. By looking for common factors in the displacements of the various sequences, the analyst should be able to make a good guess of the keyword length.

Solution of the cipher now depends on an important insight. If the keyword length is  $N$ , then the cipher, in effect, consists of  $N$  monoalphabetic substitution ciphers. For example, with the keyword DECEPTIVE, the letters in positions 1, 10, 19, and so on are all encrypted with the same monoalphabetic cipher. Thus, we can use the known frequency characteristics of the plaintext language to attack each of the monoalphabetic ciphers separately.

The periodic nature of the keyword can be eliminated by using a nonrepeating keyword that is as long as the message itself. Vigenère proposed what is referred to as an autokey system, in which a keyword is concatenated with the plaintext itself to provide a running key. For our example,

```
key:      deceptivewearediscoveredsav
plaintext: wearediscoveredsaveyourself
ciphertext: ZICVTWQNGKZEIIGASXSTSLVVWLA
```

Even this scheme is vulnerable to cryptanalysis. Because the key and the plaintext share the same frequency distribution of letters, a statistical technique can be applied. For example, e enciphered by e, by Figure 2.6, can be expected to occur with a frequency of  $(0.1275)^2 \approx 0.0163$ , whereas t enciphered by t would occur only about half as often. These regularities can be exploited to achieve successful cryptanalysis.<sup>7</sup>

The ultimate defense against such a cryptanalysis is to choose a keyword that is as long as the plaintext and has no statistical relationship to it. Such a system was introduced by an AT&T engineer named Gilbert Vernam in 1918. His system works on binary data rather than letters. The system can be expressed succinctly as follows:

$$C_i = p_i \oplus k_i$$

where

<sup>7</sup>Although the techniques for breaking a Vigenère cipher are by no means complex, a 1917 issue of *Scientific American* characterized this system as “impossible of translation.” This is a point worth



$p_i$  =  $i$ th binary digit of plaintext  
 $k_i$  =  $i$ th binary digit of key  
 $C_i$  =  $i$ th binary digit of ciphertext  
 $\oplus$  = exclusive-or (XOR) operation

Thus, the ciphertext is generated by performing the bitwise XOR of the plaintext and the key. Because of the properties of the XOR, decryption simply involves the same bitwise operation:

$$p_i = C_i \oplus k_i$$

The essence of this technique is the means of construction of the key. Vernam proposed the use of a running loop of tape that eventually repeated the key, so that in fact the system worked with a very long but repeating keyword. Although such a scheme, with a long key, presents formidable cryptanalytic difficulties, it can be broken with sufficient ciphertext, the use of known or probable plaintext sequences, or both.

An Army Signal Corp officer, Joseph Mauborgne, proposed an improvement to the Vernam cipher that yields the ultimate in security. Mauborgne suggested using a random key that was truly as long as the message, with no repetitions. Such a scheme, known as a **one-time pad**, is unbreakable. It produces random output that bears no statistical relationship to the plaintext. Because the ciphertext contains no information whatsoever about the plaintext, there is simply no way to break the code. The practical difficulty with this method is that sender and receiver must be in possession of, and protect, the random key. Thus, the Vernam cipher, despite its preeminence among ciphers, is rarely used.

### Transposition Techniques

All the techniques examined so far involve the substitution of a ciphertext symbol for a plaintext symbol. A very different kind of mapping is achieved by performing some sort of permutation on the plaintext letters. This technique is referred to as a transposition cipher.

The simplest such cipher is the rail fence technique, in which the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows. For example, to encipher the message "meet me after the toga party" with a rail fence of depth 2, we write the following:

```

m e m a t r h t g p r y
e t e f e t e o a a t

```

The encrypted message is:

```

MEMATRHTGPRYETEFETEOAAT

```

This sort of thing would be trivial to cryptanalyze. A more complex scheme is to write the message in a rectangle, row by row, and read the message off, column by column, but permute the order of the columns. The order of the columns then

```

Key:      4 3 1 2 5 6 7
Plaintext: a t t a c k p
           o s t p o n e
           d u n t i l t
           w o a m x y z
Ciphertext: TTNAAPTMTSUOAODWCOIXKNLYPETZ

```

A pure transposition cipher is easily recognized because it has the same letter frequencies as the original plaintext. For the type of columnar transposition just shown, cryptanalysis is fairly straightforward and involves laying out the ciphertext in a matrix and playing around with column positions. Digram and trigram frequency tables can be useful.

The transposition cipher can be made significantly more secure by performing more than one stage of transposition. The result is a more complex permutation that is not easily reconstructed. Thus, if the foregoing message is reencrypted using the same algorithm.

```

Key:      4 3 1 2 5 6 7
Input:    t t n a a p t
          m t s u o a o
          d w c o i x k
          n l y p e t z
Output:   NSCYAUOPTTWLTMDNAOIEPAXTTOKZ

```

To visualize the result of this double transposition, designate the letters in the original plaintext message by the numbers designating their position. Thus, with 28 letters in the message, the original sequence of letters is

```

01 02 03 04 05 06 07 08 09 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25 26 27 28

```

After the first transposition we have

```

03 10 17 24 04 11 18 25 02 09 16 23 01 08
15 22 05 12 19 26 06 13 20 27 07 14 21 28

```

which has a somewhat regular structure. But after the second transposition, we have

```

17 09 05 27 24 16 12 07 10 02 22 20 03 25
15 13 04 23 19 14 11 01 26 21 18 08 06 28

```

This is a much less structured permutation and is much more difficult to cryptanalyze.

### Rotor Machines

The example just given suggests that multiple stages of encryption can produce an algorithm that is significantly more difficult to cryptanalyze. This is as true of substitution ciphers as it is of transposition ciphers. Before the introduction of DES, the most important application of the principle of multiple stages of encryption was a class of systems known as rotor machines.<sup>8</sup>

---

<sup>8</sup> Rotor machines were used by both Germany (Enigma) and Japan (Purple) in the war's outcome.

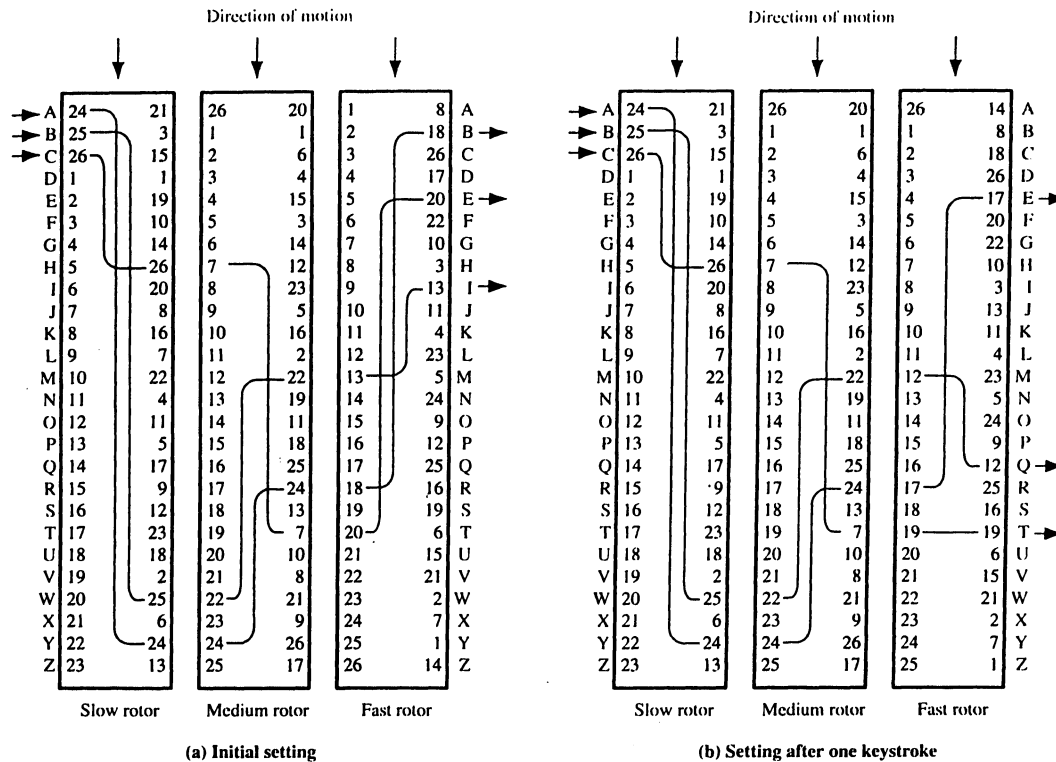


Figure 2.8 Three-Rotor Machine With Wiring Represented by Numbered Contacts.

The basic principle of the rotor machine is illustrated in Figure 2.8. The machine consists of a set of independently rotating cylinders through which electrical pulses can flow. Each cylinder has 26 input pins and 26 output pins, with internal wiring that connects each input pin to a unique output pin. For simplicity, only three of the internal connections in each cylinder are shown.

If we associate each input and output pin with a letter of the alphabet, then a single cylinder defines a monoalphabetic substitution. For example, in Figure 2.8, if an operator depresses the key for the letter A, an electric signal is applied to the first pin of the first cylinder and flows through the internal connection to the twenty-fifth output pin.

Consider a machine with a single cylinder. After each input key is depressed, the cylinder rotates one position, so that the internal connections are shifted accordingly. Thus, a different monoalphabetic substitution cipher is defined. After 26 letters of plaintext, the cylinder would be back to the initial position. Thus, we have a polyalphabetic substitution algorithm with a period of 26.

A single-cylinder system is trivial and does not present a formidable cryptanalytic task. The power of the rotor machine is in the use of multiple cylinders, in which the output pins of one cylinder are connected to the input pins of the next. Figure 2.8 shows a three-cylinder system. The left half of the figure shows a position in which

the input from the operator to the first pin (plaintext letter a) is routed through the three cylinders to appear at the output of the second pin (ciphertext letter B).

With multiple cylinders, the one farthest from the operator input rotates one pin position with each keystroke. The right half of Figure 2.8 shows the system's configuration after a single keystroke. For every complete rotation of the outer cylinder, the middle cylinder rotates one pin position. Finally, for every complete rotation of the middle cylinder, the inner cylinder rotates one pin position. This is the same type of operation seen with an odometer. The result is that there are  $26 \times 26 \times 26 = 17,576$  different substitution alphabets used before the system repeats. The addition of fourth and fifth rotors results in periods of 456,976 and 11,881,376 letters, respectively. As David Kahn eloquently put it, referring to a five-rotor machine [KAHN96, page 413],

a period of that length thwarts any practical possibility of a straightforward solution on the basis of letter frequency. This general solution would need about 50 letters per cipher alphabet, meaning that all five rotors would have to go through their combined cycle 50 times. The ciphertext would have to be as long as all the speeches made on the floor of the Senate and the House of Representatives in three successive sessions of Congress. No cryptanalyst is likely to bag that kind of trophy in his lifetime; even diplomats, who can be as verbose as politicians, rarely scale those heights of loquacity.

The significance of the rotor machine today is that it points the way to the most widely used cipher ever: the Data Encryption Standard (DES). This we examine in Chapter 3.

## 2.4 RECOMMENDED READING

Anyone interested in the history of code making and code breaking simply must read [KAHN96]. Although it is concerned more with the impact of cryptology than its technical development, [KAHN96] is an excellent introduction and makes for exciting reading. A short treatment covering the techniques of this chapter, and more, is [GARD72]. There are many books that cover classical cryptography in a more technical vein; one of the best is [SINK66]. [KORN96] is a delightful book to read and contains a lengthy section on classical techniques. [SIMM93] is a model of clarity and conciseness; its 14 pages contain a perceptive history of cryptology and a good overview of cryptographic techniques.

A detailed mathematical treatment of rotor machines is contained in [KOHN81].

The best, and one of the only, book-length treatments of steganography is [WAYN96].

**GARD72** Gardner, M. *Codes, Ciphers, and Secret Writing*. New York: Dover, 1972.

**KAHN96** Kahn, D. *The Codebreakers: The Story of Secret Writing*. New York: Scribner, 1996.

**KOHN81** Konheim, A. *Cryptography: A Primer*. New York: Wiley, 1981.

**KORN96** Korner, T. *The Pleasures of Counting*. Cambridge, England: Cambridge University Press, 1996.

**SIMM93** Simmons, G. "Cryptology." *Encyclopaedia Britannica*, 1993.

**SINK66** Sinkov, A. *Elementary Cryptanalysis: A Mathematical Approach*. Washington, DC: The Mathematical Association of America, 1966.

**WAYN96** Wayne, P. *Disappearing Cryptography*. Boston: AP Professional Books, 1996.

## 2.5 PROBLEMS

- 2.1 What is the message embedded in Figure 2.3?
- 2.2 The purpose of this problem is to show the unbreakability of the one-time pad. Suppose that we are using a Vigenère scheme with 27 characters in which the twenty-seventh character is the space character, but with a one-time key that is as long as the message. Given the ciphertext

ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS

find the key that yields the following plaintext:

MR MUSTARD WITH THE CANDLESTICK IN THE HALL

and find another key that yields the following plaintext:

MISS SCARLET WITH THE KNIFE IN THE LIBRARY

Comment on the result.

- 2.3 In one of Dorothy Sayers's mysteries, Lord Peter is confronted with the message shown in Figure 2.9. He also discovers the key to the message, which is a sequence of integers:

787656543432112343456567878878765654  
3432112343456567878878765654433211234

- a. Decrypt the message. *Hint:* What is the largest integer value?
- b. If the algorithm is known but not the key, how secure is the scheme?
- c. If the key is known but not the algorithm, how secure is the scheme?
- 2.4 The following ciphertext was generated using a simple substitution algorithm:

53++†305))6\*;4826)4+. )4+);806\*;48†8†60))85;;]8\*;;†\*8†83  
(88)5\*†;46(;88\*96\*?;8)\*†(;485);5\*†2:††(;4956\*2(5\*-4)8†8\*  
;4069285);)6†8)4++;1(†9;48081;8:8†1;48†85;4)485†528806\*81  
(†9;48;(88;4(†?34;48)4†;161;;188;†?;

*I thought to see the fairies in the fields, but I saw only the evil elephants with their black backs. Woe! how that sight awed me! The elves danced all around and about while I heard voices calling clearly. Ah! how I tried to see—throw off the ugly cloud—but no blind eye of a mortal was permitted to spy them. So then came minstrels, having gold trumpets, harps and drums. These played very loudly beside me, breaking that spell. So the dream vanished, whereat I thanked Heaven. I shed many tears before the thin moon rose up, frail and faint as a sickle of straw. Now though the Enchanter gnash his teeth vainly, yet shall he return as the spring returns. Oh, wretched man! Hell gapes, Erebus now lies open. The mouths of Death wait on thy end.*

Figure 2.9 A Puzzle for Lord Peter.

Decrypt this message. *Hints:*

1. As you know, the most frequently occurring letter in English is e. Therefore, the first or second (or perhaps third?) most common character in the message is likely to stand for e. Also, e is often seen in pairs (e.g., meet, fleet, speed, seen, been, agree, etc.). Try to find a character in the ciphertext that decodes to e.
2. The most common word in English is "the." Use this fact to guess the characters that stand for t and h.
3. Decipher the rest of the message by deducing additional words.

*Warning:* The resulting message is in English but may not make much sense on a first reading.

- 2.5 One way to solve the key distribution problem is to use a line from a book that both the sender and the receiver possess. Typically, at least in spy novels, the first sentence of a book serves as the key. The particular scheme discussed in this problem is from one of the best suspense novels involving secret codes, *Talking to Strange Men*, by Ruth Rendell. Work this problem without consulting that book.

Consider the following message:

SIDKHKDM AF HCRKIABIE SHIMC KD LFEAILA

This ciphertext was produced using the first sentence of *The Other Side of Silence* (a book about the spy Kim Philby):

The snow lay thick on the steps and the snowflakes driven by the wind looked black in the headlights of the cars.

A simple substitution cipher was used.

- a. What is the encryption algorithm?
- b. How secure is it?
- c. To make the key distribution problem simple, both parties can agree to use the first or last sentence of a book as the key. To change the key, they simply need to agree on a new book. The use of the first sentence would be preferable to the use of the last. Why?

- 2.6 In one of his cases, Sherlock Holmes was confronted with the following message:

534 C2 13 127 36 31 4 17 21 41  
DOUGLAS 109 293 5 37 BIRLSTONE  
26 BIRLSTONE 9 127 171

Although Watson was puzzled, Holmes was able immediately to deduce the type of cipher. Can you?

- 2.7 A disadvantage of the general monoalphabetic cipher is that both sender and receiver must commit the permuted cipher sequence to memory. A common technique for avoiding this is to use a keyword from which the cipher sequence can be generated. For example, using the keyword *CIPHER*, write out the keyword followed by unused letters in normal order and match this against the plaintext letters:

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

If it is felt that this process does not produce sufficient mixing, write the remaining letters on successive lines and then generate the sequence by reading down the columns:

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

This yields the sequence

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

Such a system is used in the example in Section 2.3. Determine the keyword.

- 2.8** How many possible keys does a Playfair cipher have? Express your answer as an approximate power of 2.
- 2.9** We have shown that the Hill cipher succumbs to a known plaintext attack if sufficient plaintext-ciphertext pairs are provided. It is even easier to solve the Hill cipher if a chosen plaintext attack can be mounted. Describe such an attack.





# CHAPTER 3

## CONVENTIONAL ENCRYPTION: MODERN TECHNIQUES

*All the afternoon Mungo had been working on Stern's code, principally with the aid of the latest messages which he had copied down at the Nevin Square drop. Stern was very confident. He must be well aware London Central knew about that drop. It was obvious that they didn't care how often Mungo read their messages, so confident were they in the impenetrability of the code.*

*—Talking to Strange Men, Ruth Rendell*

**T**he objective of this chapter is to illustrate the principles of modern conventional encryption. For this purpose, we focus on the most widely used conventional encryption algorithms: the Data Encryption Standard (DES). Although numerous conventional encryption algorithms have been developed since the introduction of DES, it remains the most important such algorithm. Further, a detailed study of DES provides an understanding of the principles used in other conventional encryption algorithms. We examine other important conventional encryption algorithms in Chapter 4.

Compared to public-key encryption schemes such as RSA, the structure of DES, and most conventional encryption algorithms, is very complex and cannot be explained as easily as RSA and similar algorithms. Accordingly, we begin with a simplified version of DES. This version allows the reader to perform encryption and decryption by hand and gain a good under-

standing of the working of the algorithm details. Classroom experience indicates that a study of this simplified version enhances understanding of DES.<sup>1</sup>

Following the discussion of the simplified version of DES, this chapter then looks at the general principles of symmetric block ciphers, which are the type of conventional encryption algorithms studied in this book. Next, we cover full DES. Following this look at a specific algorithm, we return to a more general discussion of block cipher design.

### 3.1 SIMPLIFIED DES

Simplified DES is an educational rather than a secure encryption algorithm. It has similar properties and structure to DES with much smaller parameters. It was developed by Professor Edward Schaefer of Santa Clara University [SCHA96]. The reader might find it useful to work through an example by hand while following the discussion in this section.<sup>2</sup>

#### Overview

Figure 3.1 illustrates the overall structure of the simplified DES, which we will refer to as S-DES. The S-DES encryption algorithm takes an 8-bit block of plaintext (example: 10111101) and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext.

The encryption algorithm involves five functions: an initial permutation (IP); a complex function labeled  $f_k$ , which involves both permutation and substitution operations and depends on a key input; a simple permutation function that switches (SW) the two halves of the data; the function  $f_k$  again, and finally a permutation function that is the inverse of the initial permutation ( $IP^{-1}$ ). As was mentioned in Chapter 2, the use of multiple stages of permutation and substitution results in a more complex algorithm, which increases the difficulty of cryptanalysis.

The function  $f_k$  takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. The algorithm could have been designed to work with a 16-bit key, consisting of two 8-bit subkeys, one used for each occurrence of  $f_k$ . Alternatively, a single 8-bit key could have been used, with the same key used twice in the algorithm. A compromise is to use a 10-bit key from which two 8-bit subkeys are generated, as depicted in Figure 3.1. In this case, the key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey ( $K_1$ ). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey ( $K_2$ ).

<sup>1</sup>However, you may safely skip Section 3.1, at least on a first reading. If you get lost or bogged down in the details of DES, then you can go back and start with simplified DES.

<sup>2</sup>This section is based on Schaefer's paper.

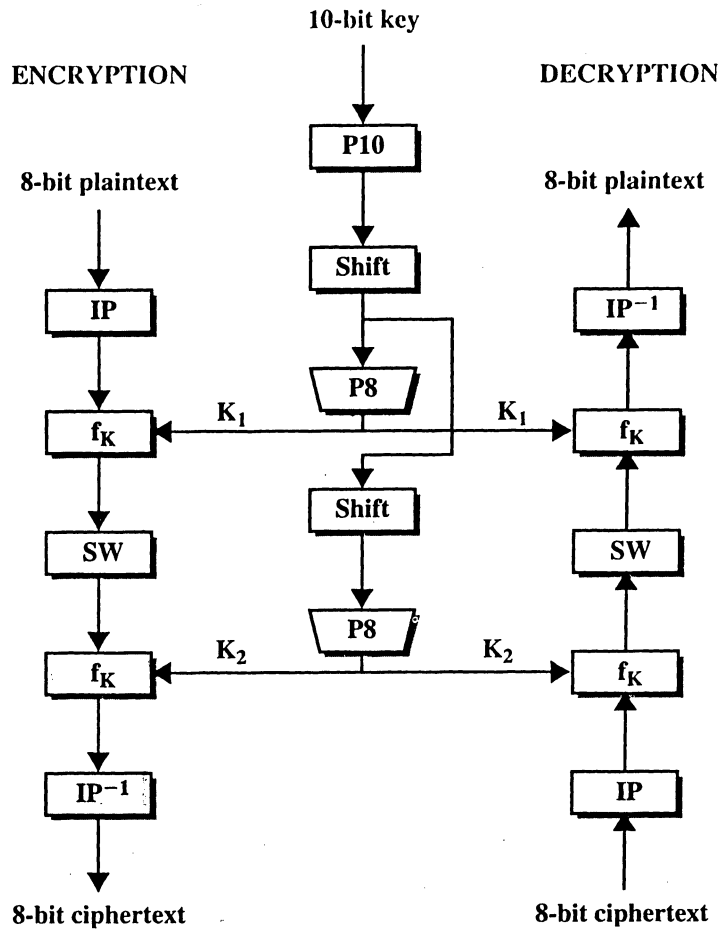


Figure 3.1 Simplified DES Scheme.

We can concisely express the encryption algorithm as a composition of functions:

$$IP^{-1} \circ f_{K_2} \circ SW \circ f_{K_1} \circ IP$$

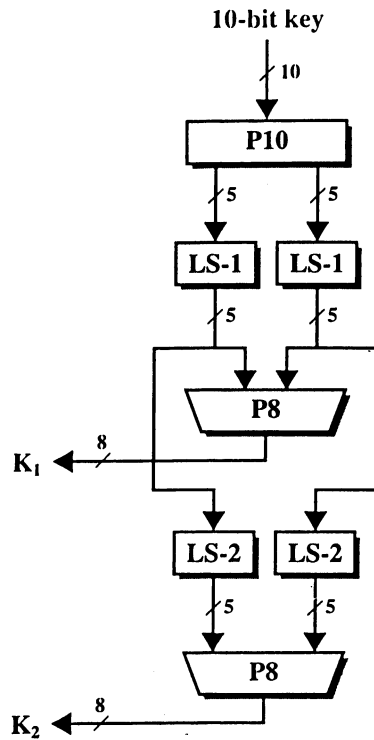
which can also be written as

$$\text{ciphertext} = IP^{-1}(f_{K_2}(SW(f_{K_1}(IP(\text{plaintext}))))))$$

where

$$K_1 = P8(\text{Shift}(P10(\text{key})))$$

$$K_2 = P8(\text{Shift}(\text{Shift}(P10(\text{key}))))$$



**Figure 3.2** Key Generation for Simplified DES.

Decryption is also shown in Figure 3.1 and is essentially the reverse of encryption:

$$\text{plaintext} = \text{IP}^{-1}(\text{f}_{\text{K}_2}(\text{SW}(\text{f}_{\text{K}_1}(\text{IP}(\text{ciphertext}))))))$$

We now examine the elements of S-DES in more detail.

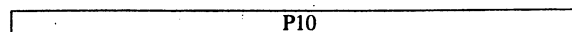
### S-DES Key Generation

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm. Figure 3.2 depicts the stages followed to produce the subkeys.

First, permute the key in the following fashion. Let the 10-bit key be designated as  $(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10})$ . Then the permutation P10 is defined as

$$\text{P10}(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}) = (k_3, k_5, k_2, k_7, k_4, k_{10}, k_1, k_9, k_8, k_6)$$

P10 can be concisely defined by the display:



This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on. For example, the key (1010000010) is permuted to (1000001100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000).

Next we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

The result is subkey 1 ( $K_1$ ). In our example, this yields (10100100).

We then go back to the pair of 5-bit strings produced by the two LS-1 functions and perform a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce  $K_2$ . In our example, the result is (01000011).

## S-DES Encryption

Figure 3.3 shows the S-DES encryption algorithm in greater detail. As was mentioned, encryption involves the sequential application of five functions. We examine each of these.

### Initial and Final Permutations

The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function:

IP							
2	6	3	1	4	8	5	7

This retains all 8 bits of the plaintext but mixes them up. At the end of the algorithm, the inverse permutation is used:

$IP^{-1}$							
4	1	3	5	7	2	8	6

It is easy to show by example that the second permutation is indeed the reverse of the first; that is,  $IP^{-1}(IP(X)) = X$ .

### The Function $f_k$

The most complex component of S-DES is the function  $f_k$ , which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let  $L$  and  $R$  be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to  $f_k$ , and let  $F$  be a mapping (not necessarily one-to-one) from 4-bit strings to 4-bit strings. Then we let

$$f_k(L, R) = (L \oplus F(R, SK), R)$$

where  $SK$  is a subkey and  $\oplus$  is the bit-by-bit exclusive-OR function. For example, suppose the output of the IP stage in Figure 3.3 is (10111101) and  $F(1101, SK) = (1110)$  for some key  $SK$ . Then  $f_k(10111101) = (01011101)$  because  $(1011) \oplus (1110) = (0101)$ .

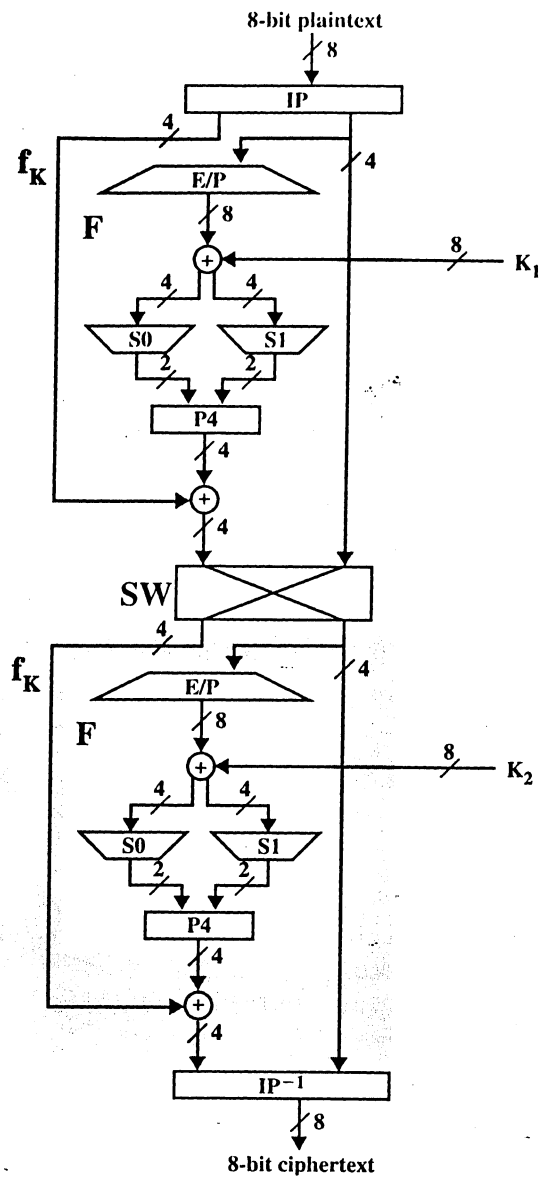
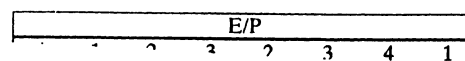


Figure 3.3 Simplified DES Scheme Encryption Detail.

We now describe the mapping  $F$ . The input is a 4-bit number  $(n_1n_2n_3n_4)$ . The first operation is an expansion/permutation operation:



For what follows, it is clearer to depict the result in this fashion:

$$\begin{array}{c|cc|c} n_4 & n_1 & n_2 & n_3 \\ \hline n_2 & n_3 & n_4 & n_1 \end{array}$$

The 8-bit subkey  $K_1 = (k_{11}, k_{12}, k_{13}, k_{14}, k_{15}, k_{16}, k_{17}, k_{18})$  is added to this value using exclusive-OR:

$$\begin{array}{c|cc|c} n_4 + k_{11} & n_1 + k_{12} & n_2 + k_{13} & n_3 + k_{14} \\ \hline n_2 + k_{15} & n_3 + k_{16} & n_4 + k_{17} & n_1 + k_{18} \end{array}$$

Let us rename these 8 bits:

$$\begin{array}{c|cc|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ \hline p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \end{array}$$

The first four bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2-bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2-bit output. These two boxes are defined as follows:

$$\begin{array}{c} \begin{array}{c} 0 \quad 1 \quad 2 \quad 3 \\ 0 \begin{bmatrix} 1 & 0 & 3 & 2 \end{bmatrix} \\ S0 = 1 \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} \\ 2 \begin{bmatrix} 0 & 2 & 1 & 3 \end{bmatrix} \\ 3 \begin{bmatrix} 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \quad \begin{array}{c} 0 \quad 1 \quad 2 \quad 3 \\ 0 \begin{bmatrix} 0 & 1 & 2 & 3 \end{bmatrix} \\ S1 = 1 \begin{bmatrix} 2 & 0 & 1 & 3 \end{bmatrix} \\ 2 \begin{bmatrix} 3 & 0 & 1 & 0 \end{bmatrix} \\ 3 \begin{bmatrix} 2 & 1 & 0 & 3 \end{bmatrix} \end{array} \end{array}$$

The S-boxes operate as follows: The first and fourth input bits are treated as 2-bit numbers that specify a row of the S-box, and the second and third input bits specify a column of the S-box. The entry in that row and column, in base 2, is the 2-bit output. For example, if  $(p_{0,0}p_{0,3}) = (00)$  and  $(p_{0,1}p_{0,2}) = (10)$ , then the output is from row 0, column 2 of S0, which is 3, or (11) in binary. Similarly,  $(p_{1,0}p_{1,3})$  and  $(p_{1,1}p_{1,2})$  are used to index into a row and column of S1 to produce an additional 2 bits.

Next, the 4 bits produced by S0 and S1 undergo a further permutation as follows:

P4			
2	4	3	1

The output of P4 is the output of the function F.

### The Switch Function

The function  $f_K$  only alters the leftmost 4 bits of the input. The switch function (SW) interchanges the left and right 4 bits so that the second instance of  $f_K$  operates on a different 4 bits. In this second instance, the E/P, S0, S1, and P4 functions are the same. The key input is  $K_2$ .

## Analysis of Simplified DES

A brute-force attack on simplified DES is certainly feasible. With a 10-bit key, there are only  $2^{10} = 1024$  possibilities. Given a ciphertext, an attacker can try each possibility and analyze the result to determine if it is reasonable plaintext.

What about cryptanalysis? Let us consider a known plaintext attack in which a single plaintext  $(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8)$  and its ciphertext output  $(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8)$  are known and the key  $(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10})$  is unknown. Then each  $c_i$  is a polynomial function  $g_i$  of the  $p_j$ 's and  $k_j$ 's. We can therefore express the encryption algorithm as 8 nonlinear equations in 10 unknowns. There are a number of possible solutions, but each of these could be calculated and then analyzed. Each of the permutations and additions in the algorithm are linear maps. The non-linearity comes from the S-boxes. It is useful to write down the equations for these boxes. For clarity, rename  $(p_{0,0}, p_{0,1}, p_{0,2}, p_{0,3}) = (a, b, c, d)$  and  $(p_{1,1}, p_{1,1}, p_{1,2}, p_{1,3}) = (w, x, y, z)$ , and let the 4-bit output be  $(q, r, s, t)$ . Then the operation of the S0 is defined by the following equations:

$$q = abcd + ab + ac + b + d$$

$$r = abcd + abd + ab + ac + ad + a + c + 1$$

where all additions are modulo 2. Similar equations define S1. Alternating linear maps with these nonlinear maps results in complex polynomial expressions for the ciphertext bits, making cryptanalysis difficult. To visualize the scale of the problem, note that a polynomial equation in 10 unknowns in binary arithmetic can have  $2^{10}$  possible terms. On average, we might therefore expect each of the 8 equations to have  $2^9$  terms. The interested reader might try to find these equations with a symbolic processor. Either the reader or the software will give up before much progress is made.

### Relationship to DES

DES operates on 64-bit blocks of input. The encryption scheme can be defined as

$$IP^{-1} \circ f_{K_{16}} \circ SW \circ f_{K_{15}} \circ SW \circ \dots \circ SW \circ f_{K_1} \circ IP$$

A 56-bit key is used, from which sixteen 48-bit subkeys are calculated. There is an initial permutation of 56 bits followed by a sequence of shifts and permutations of 48 bits.

Within the encryption algorithm, instead of F acting on 4 bits  $(n_1 n_2 n_3 n_4)$ , it acts on 32 bits  $(n_1 \dots n_{32})$ . After the initial expansion/permutation, the output of 48 bits can be diagrammed as

$n_{32}$	$n_1$	$n_2$	$n_3$	$n_4$	$n_5$
$n_4$	$n_5$	$n_6$	$n_7$	$n_8$	$n_9$
.		.			.
.		.			.
.		.			.
$n_{28}$	$n_{29}$	$n_{30}$	$n_{31}$	$n_{32}$	$n_1$

This matrix is added (exclusive-OR) to a 48-bit subkey. There are 8 rows, corresponding to 8 S-boxes. Each S-box has 4 rows and 16 columns. The first and last bit of a row of the preceding matrix picks out a row of an S-box, and the middle 4 bits pick out a column.



## 3.2 BLOCK CIPHER PRINCIPLES

Virtually all symmetric block encryption algorithms in current use are based on a structure referred to as a Feistel block cipher [FEIS73]. For that reason, it is important to examine the design principles of the Feistel cipher. We begin with a comparison of stream ciphers and block ciphers. Then we discuss the motivation for the Feistel block cipher structure. Finally, we discuss some of its implications.

### Stream Ciphers and Block Ciphers

A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. Examples of classical stream ciphers are the autokeyed Vigenère cipher and the Vernam cipher. A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 bits is used. Using various of the modes of operation explained later in this chapter, a block cipher can be used to achieve the same effect as a stream cipher.

Far more effort has gone into analyzing block ciphers. In general, they seem applicable to a broader range of applications than stream ciphers. The vast majority of network-based conventional cryptographic applications make use of block ciphers. Accordingly, the concern in this chapter, and in our discussions throughout the book of symmetric encryption, will be solely on block ciphers.

### Motivation for the Feistel Cipher Structure

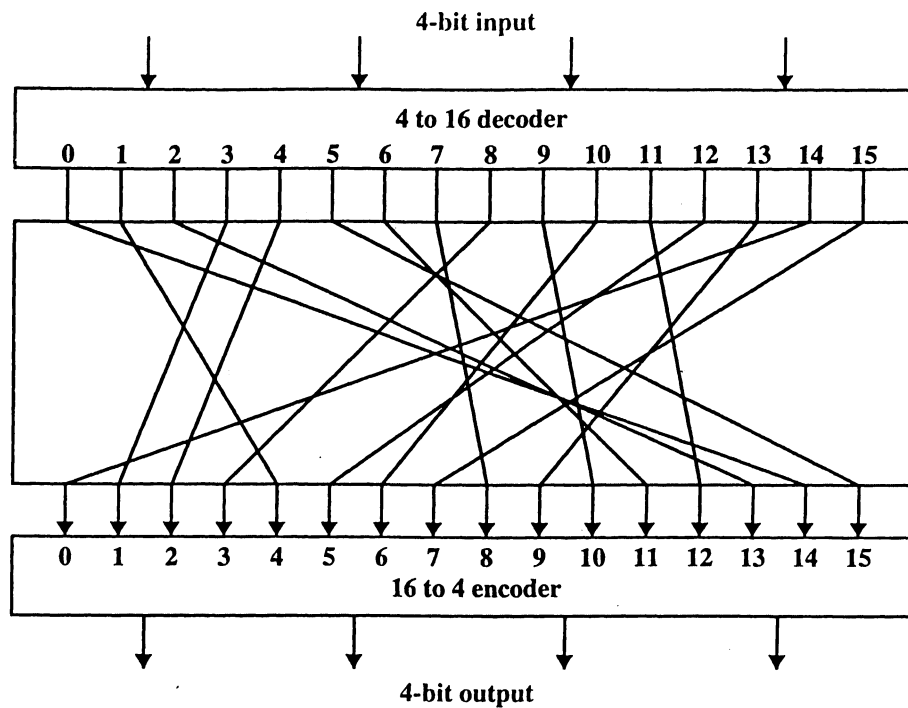
A block cipher operates on a plaintext block of  $n$  bits to produce a ciphertext block of  $n$  bits. There are  $2^n$  possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must produce a unique ciphertext block. Such a transformation is called reversible, or nonsingular. The following examples illustrate nonsingular and singular transformation for  $n = 2$ .

Reversible Mapping		Irreversible Mapping	
Plaintext	Ciphertext	Plaintext	Ciphertext
00	11	00	11
01	10	01	10
10	00	10	01
11	01	11	01

In the latter case, a ciphertext of 01 could have been produced by one of two plaintext blocks. So if we limit ourselves to reversible mappings, the number of different transformations is  $2^n!$ .

Figure 3.4 illustrates the logic of a general substitution cipher for  $n = 4$ . A 4-bit input produces one of 16 possible input states, which is mapped by the substitution cipher into a unique one of 16 possible output states, each of which is represented by 4 ciphertext bits. The encryption and decryption mappings can be defined by a tabulation, as shown in Table 3.1. This is the most general form of block cipher and can be used to define any reversible mapping between plaintext and ciphertext.

58



**Figure 3.4** General  $n$ -bit- $n$ -bit Block Substitution (shown with  $n = 4$ ).

**Table 3.1** Encryption and Decryption Tables for Substitution Cipher of Figure 3.4

Plaintext	Ciphertext	Ciphertext	Plaintext
0000	1110	0000	1110
0001	0100	0001	0011
0010	1101	0010	0100
0011	0001	0011	1000
0100	0010	0100	0001
0101	1111	0101	1100
0110	1011	0110	1010
0111	1000	0111	1111
1000	0011	1000	0111
1001	1010	1001	1101
1010	0110	1010	1001
1011	1100	1011	0110
1100	0101	1100	1011
1101	1001	1101	0010
1110	0000	1110	0000
1111	0111	1111	0101

But there is a practical problem with this approach. If a small block size, such as  $n = 4$ , is used, then the system is equivalent to a classical substitution cipher. Such systems, as we have seen, are vulnerable to a statistical analysis of the plaintext. This weakness is not inherent in the use of a substitution cipher but rather results from the use of a small block size. If  $n$  is large and an arbitrary reversible substitution between plaintext and ciphertext is allowed, then the statistical characteristics of the source plaintext are masked to such an extent that this type of cryptanalysis is infeasible.

An arbitrary reversible substitution cipher for a large block size is not practical, however, from an implementation and performance point of view. For such a transformation, the mapping itself is the key. Consider again Table 3.1, which defines one particular reversible mapping from plaintext to ciphertext for  $n = 4$ . The mapping can be defined by the entries in the second column, which show the value of the ciphertext for each plaintext block. This, in essence, is the key that determines the specific mapping from among all possible mappings. In this case, the key requires 64 bits. In general, for an  $n$ -bit general substitution block cipher, the size of the key is  $n \times 2^n$ . For a 64-bit block, which is a desirable length to thwart statistical attacks, the key size is  $64 \times 2^{64} = 2^{70} \approx 10^{21}$  bits.

In considering these difficulties, Feistel points out that what is needed is an approximation to this ideal block-cipher system for large  $n$ , built up out of components that are easily realizable [FEIS75]. But before turning to Feistel's approach, let us make one other observation. We could confine ourselves to this general block substitution cipher but, to make its implementation tractable, confine ourselves to a subset of the  $2^n!$  possible reversible mappings. For example, suppose we define the mapping in terms of a set of linear equations. In the case of  $n = 4$ , we have

$$\begin{aligned} y_1 &= k_{11}x_1 + k_{12}x_2 + k_{13}x_3 + k_{14}x_4 \\ y_2 &= k_{21}x_1 + k_{22}x_2 + k_{23}x_3 + k_{24}x_4 \\ y_3 &= k_{31}x_1 + k_{32}x_2 + k_{33}x_3 + k_{34}x_4 \\ y_4 &= k_{41}x_1 + k_{42}x_2 + k_{43}x_3 + k_{44}x_4 \end{aligned}$$

where the  $x_i$  are the four binary digits of the plaintext block, the  $y_i$  are the four binary digits of the ciphertext block, the  $k_{ij}$  are the binary coefficients, and arithmetic is mod 2. The key size is just  $n^2$ , in this case 16 bits. The danger with this kind of formulation is that it may be vulnerable to cryptanalysis by an attacker that is aware of the structure of the algorithm. In this example, what we have is essentially the Hill cipher discussed in Chapter 2, applied to binary data rather than characters. As we saw in Chapter 2, a simple linear system such as this is extremely vulnerable.

### The Feistel Cipher

Feistel proposed [FEIS73] that we can approximate the simple substitution cipher by utilizing the concept of a product cipher, which is the performing of two or more basic ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers. In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations. In fact, this is a practical application of a proposal by Claude Shannon to develop a product cipher that alternates *confusion* and *diffusion* functions. We look next at these concepts of diffusion and

confusion and then present the Feistel cipher. But first, it is worth commenting on this remarkable fact: The Feistel cipher structure, which dates back over a quarter century and which, in turn, is based on Shannon's proposal of 1945, is the structure used by virtually all significant symmetric block ciphers currently in use.

### Diffusion and Confusion

The terms *diffusion* and *confusion* were introduced by Claude Shannon to capture the two basic building blocks for any cryptographic system [SHAN49].<sup>3</sup> Shannon's concern was to thwart cryptanalysis based on statistical analysis. The reasoning is as follows: Assume the attacker has some knowledge of the statistical characteristics of the plaintext. For example, in a human-readable message in some language, the frequency distribution of the various letters may be known. Or there may be words or phrases likely to appear in the message (probable words). If these statistics are in any way reflected in the ciphertext, the cryptanalyst may be able to deduce the encryption key, or part of the key, or at least a set of keys likely to contain the exact key. In what Shannon refers to as a strongly ideal cipher, all statistics of the ciphertext are independent of the particular key used. The arbitrary substitution cipher that we discussed previously (Figure 3.4) is such a cipher, but as we have seen, is impractical.

Other than recourse to ideal systems, Shannon suggest two methods for frustrating statistical cryptanalysis: diffusion and confusion. In **diffusion**, the statistical structure of the plaintext is dissipated into long-range statistics of the ciphertext. This is achieved by having each plaintext digit affect the value of many ciphertext digits, which is equivalent to saying that each ciphertext digit is affected by many plaintext digits. An example of diffusion is to encrypt a message  $M = m_1, m_2, m_3, \dots$  of characters with an averaging operation:

$$y_n = \sum_{i=1}^k m_{n+i} \pmod{26}$$

adding  $k$  successive letters to get a ciphertext letter  $y_n$ . One can show that the statistical structure of the plaintext has been dissipated. Thus, the letter frequencies in the ciphertext will be more nearly equal than in the plaintext; the digram frequencies will also be more nearly equal, and so on. In a binary block cipher, diffusion can be achieved by repeatedly performing some permutation on the data followed by applying a function to that permutation; the effect is that bits from different positions in the original plaintext contribute to a single bit of ciphertext.<sup>4</sup>

Every block cipher involves a transformation of a block of plaintext into a block of ciphertext, where the transformation depends on the key. The mechanism

<sup>3</sup>Shannon's 1949 paper appeared originally as a classified report in 1945. Shannon enjoys an amazing and unique position in the history of computer and information science. He not only developed the seminal ideas of modern cryptography but is also responsible for inventing the discipline of information theory. In addition, he founded another discipline, the application of Boolean algebra to the study of digital circuits; this last he managed to toss off as a master's thesis.

<sup>4</sup>Some books on cryptography equate permutation with diffusion. This is incorrect. Permutation, *by itself*, does not change the statistics of the plaintext at the level of individual letters or permuted blocks. For example, in DES, the permutation swaps two 32-bit blocks, so statistics of strings of 32 bits or less are preserved.

of diffusion seeks to make the statistical relationship between the plaintext and ciphertext as complex as possible in order to thwart attempts to deduce the key. On the other hand, **confusion** seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, again to thwart attempts to discover the key. Thus, even if the attacker can get some handle on the statistics of the ciphertext, the way in which the key was used to produce that ciphertext is so complex as to make it difficult to deduce the key. This is achieved by the use of a complex substitution algorithm. In contrast, a simple linear substitution function would add little confusion.

As [ROBS95] points out, so successful are diffusion and confusion in capturing the essence of the desired attributes of a block cipher that they have become the cornerstone of modern block cipher design.

### Feistel Cipher Structure

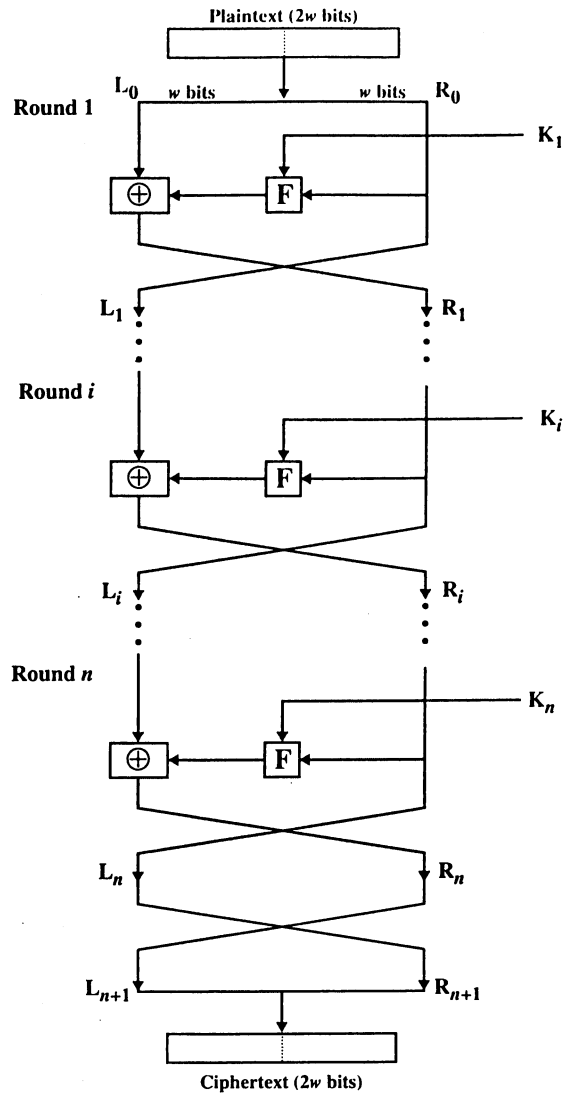
Figure 3.5 depicts the structure proposed by Feistel. The inputs to the encryption algorithm are a plaintext block of length  $2w$  bits and a key  $K$ . The plaintext block is divided into two halves,  $L_0$  and  $R_0$ . The two halves of the data pass through  $n$  rounds of processing and then combine to produce the ciphertext block. Each round  $i$  has as inputs  $L_{i-1}$  and  $R_{i-1}$ , derived from the previous round, as well as a subkey  $K_i$ , derived from the overall  $K$ . In general, the subkeys  $K_i$  are different from  $K$  and from each other.

All rounds have the same structure. A **substitution** is performed on the left half of the data. This is done by applying a *round function*  $F$  to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey  $K_i$ . Following this substitution, a **permutation** is performed that consists of the interchange of the two halves of the data.<sup>5</sup> This structure is a particular form of the substitution-permutation network (SPN) proposed by Shannon.

The exact realization of a Feistel network depends on the choice of the following parameters and design features:

- **Block size:** Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed. A block size of 64 bits is a reasonable tradeoff and is nearly universal in block cipher design.
- **Key size:** Larger key size means greater security but may decrease encryption/decryption speed. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.
- **Number of rounds:** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.

<sup>5</sup>The final round is followed by an interchange that undoes the interchange that is part of the final round. One could simply leave both interchanges out of the diagram, at the sacrifice of some consistency of presentation. In any case, the effective lack of a swap in the final round is done to simplify the implementation of the decryption process, as we shall see.



**Figure 3.5** Classical Feistel Network.

- **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.
- **Round function:** Again, greater complexity generally means greater resistance to cryptanalysis.

There are two other considerations in the design of a Feistel cipher, which will be discussed in greater detail later in this chapter and in the next:

- **Fast software encryption/decryption:** In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.
- **Ease of analysis:** Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

Looking back at Figures 3.1 and 3.3, we can see that SDES exhibits a Feistel structure with two rounds. The one difference from a “pure” Feistel structure is that the algorithm begins and ends with a permutation function. This difference also appears in full DES.

### Feistel Decryption Algorithm

The process of decryption with a Feistel cipher is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the algorithm, but use the subkeys  $K_i$  in reverse order. That is, use  $K_n$  in the first round,  $K_{n-1}$  in the second round, and so on until  $K_1$  is used in the last round. This is a nice feature because it means we need not implement two different algorithms, one for encryption and one for decryption.

To see that the same algorithm with a reversed key order produces the correct result, consider Figure 3.6, which shows the encryption process going down the left-hand side and the decryption process going up the right-hand side for a 16-round algorithm (the result would be the same for any number of rounds). For clarity, we use the notation  $LE_i$  and  $RE_i$  for data traveling through the encryption algorithm and  $LD_i$  and  $RD_i$  for data traveling through the decryption algorithm. The diagram indicates that, at every round, the intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped. To put this another way, let the output of the  $i$ th encryption round be  $LE_i || RE_i$  ( $L_i$  concatenated with  $R_i$ ). Then the corresponding input to the  $(16-i)$ th decryption round is  $RD_i || LD_i$ .

Let us walk through Figure 3.6 to demonstrate the validity of the preceding assertions.<sup>6</sup> After the last iteration of the encryption process, the two halves of the output are swapped, so that the ciphertext is  $RE_{16} || LE_{16}$ . The output of that round is the ciphertext. Now take that ciphertext and use it as input to the same algorithm. The input to the first round is  $RE_{16} || LE_{16}$ , which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process.

<sup>6</sup>To simplify the diagram, it is unwrapped, not showing the swap that occurs at the end of each iteration. But please note that the intermediate result at the end of the  $i$ th stage of the encryption process is the  $2w$ -bit quantity formed by concatenating  $LE_i$  and  $RE_i$ , and that the intermediate result at the end of the  $i$ th stage of the decryption process is the  $2w$ -bit quantity formed by concatenating  $LD_i$  and  $RD_i$ .

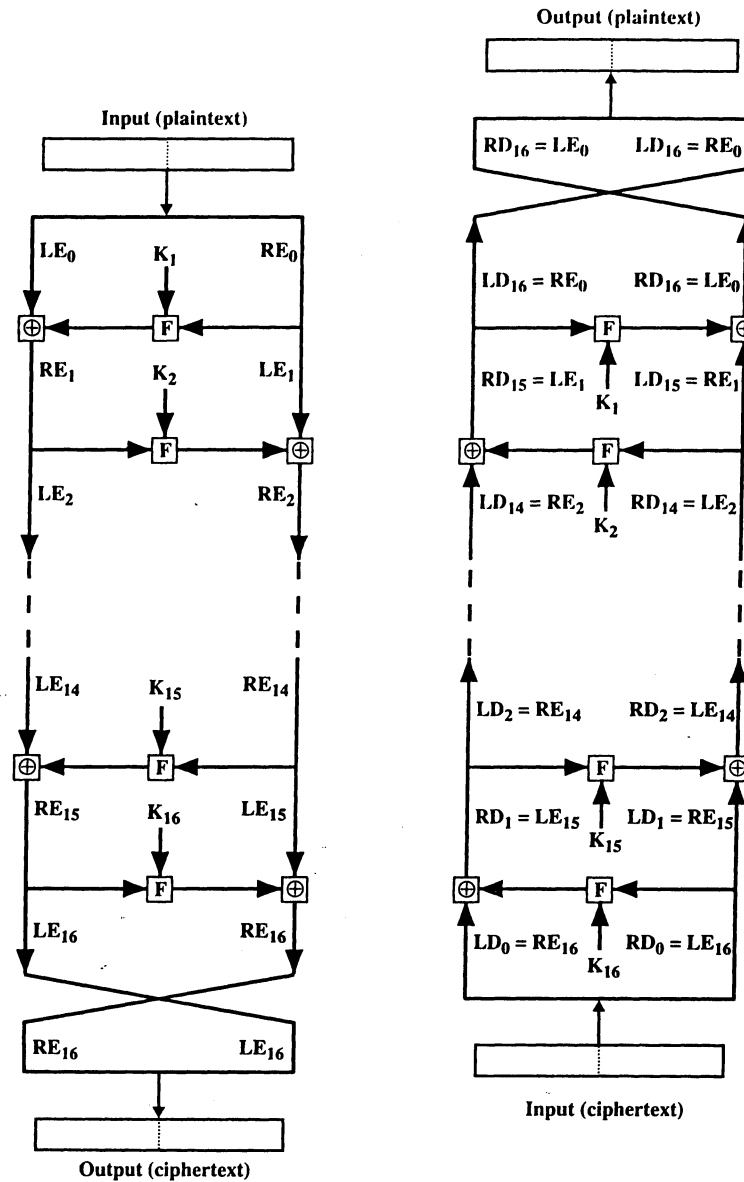


Figure 3.6 Feistel Encryption and Decryption.

Now we would like to show that the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process. First, consider the encryption process. We see that

$$LE_{16} = RE_{15}$$



On the decryption side,

$$\begin{aligned} LD_1 &= RD_0 = LE_{16} = RE_{15} \\ RD_1 &= LD_0 \oplus F(RD_0, K_{16}) \\ &= RE_{16} \oplus F(RE_{15}, K_{16}) \\ &= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16}) \end{aligned}$$

The XOR has the following properties:

$$\begin{aligned} [A \oplus B] \oplus C &= A \oplus [B \oplus C] \\ D \oplus D &= 0 \\ E \oplus 0 &= E \end{aligned}$$

Thus, we have  $LD_1 = RE_{15}$  and  $RD_1 = LE_{15}$ . Therefore, the output of the first round of the decryption process is  $LE_{15} || RE_{15}$ , which is the 32-bit swap of the input to the sixteenth round of the encryption. This correspondence holds all the way through the 16 iterations, as is easily shown. We can cast this process in general terms. For the  $i$ th iteration of the encryption algorithm,

$$\begin{aligned} LE_i &= RE_{i-1} \\ RE_i &= LE_{i-1} \oplus F(RE_{i-1}, K_i) \end{aligned}$$

Rearranging terms,

$$\begin{aligned} RE_{i-1} &= LE_i \\ LE_{i-1} &= RE_i \oplus F(RE_{i-1}, K_i) = RE_i \oplus F(LE_i, K_i) \end{aligned}$$

Thus, we have described the inputs to the  $i$ th iteration as a function of the outputs, and these equations confirm the assignments shown in the right-hand side of Figure 3.6.

Finally, we see that the output of the last round of the decryption process is  $RE_0 || LE_0$ . A 32-bit swap recovers the original plaintext, demonstrating the validity of the Feistel decryption process.

Note that the derivation does not require that  $F$  be a reversible function. To see this, take a limiting case in which  $F$  produces a constant output (e.g., all ones) regardless of the values of its two arguments. The equations still hold.

### 3.3 THE DATA ENCRYPTION STANDARD

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption.

The DES enjoys widespread use. It has also been the subject of much controversy concerning how secure the DES is. To appreciate the nature of the controversy, let us quickly review the history of the DES.



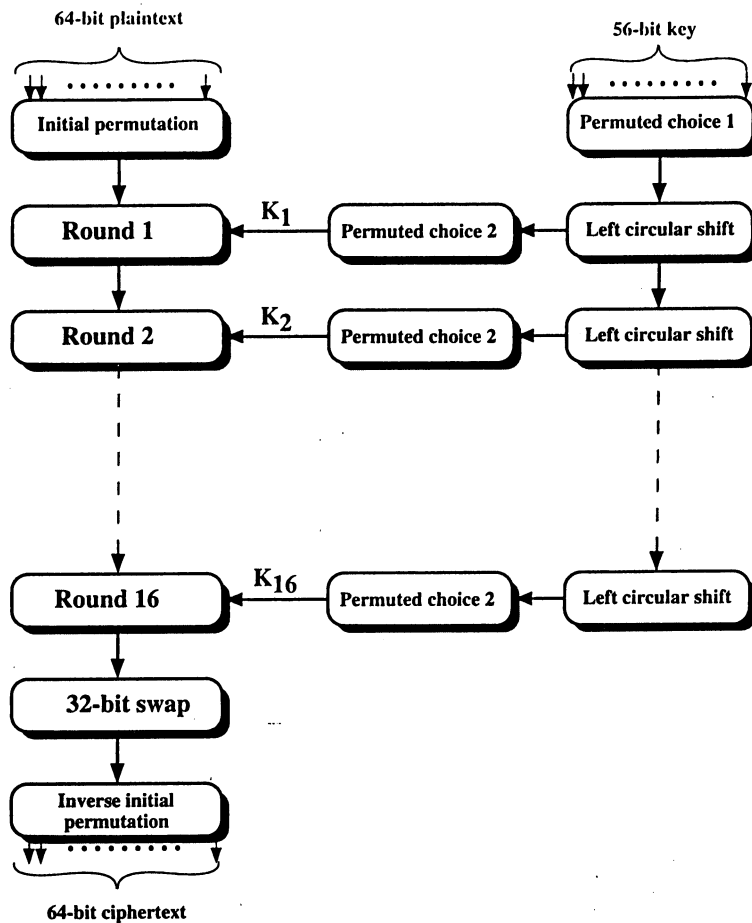


Figure 3.7 General Depiction of DES Encryption Algorithm.

initial permutation (IP) that rearranges the bits to produce the *permuted input*. This is followed by a phase consisting of 16 rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the **preoutput**. Finally, the preoutput is passed through a permutation ( $IP^{-1}$ ) that is the inverse of the initial permutation function, to produce the 64-bit ciphertext. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher, as shown in Figure 3.5.

The right-hand portion of Figure 3.7 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the 16 rounds, a *subkey* ( $K_i$ ) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated iteration of the key bits.

**Table 3.2** Permutation Tables for DES

(a) 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

(b) Inverse Initial 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

(c) Expansion Permutation (E)

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

(d) Permutation Function (P)

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

**Initial Permutation**

The initial permutation and its inverse are defined by tables, as shown in Tables 3.2a and 3.2b, respectively. To see that these two permutation functions are indeed the inverse of each other, consider the following 64-bit input  $M$ :

$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$	$M_8$
$M_9$	$M_{10}$	$M_{11}$	$M_{12}$	$M_{13}$	$M_{14}$	$M_{15}$	$M_{16}$
$M_{17}$	$M_{18}$	$M_{19}$	$M_{20}$	$M_{21}$	$M_{22}$	$M_{23}$	$M_{24}$
$M_{25}$	$M_{26}$	$M_{27}$	$M_{28}$	$M_{29}$	$M_{30}$	$M_{31}$	$M_{32}$
$M_{33}$	$M_{34}$	$M_{35}$	$M_{36}$	$M_{37}$	$M_{38}$	$M_{39}$	$M_{40}$
$M_{41}$	$M_{42}$	$M_{43}$	$M_{44}$	$M_{45}$	$M_{46}$	$M_{47}$	$M_{48}$
$M_{49}$	$M_{50}$	$M_{51}$	$M_{52}$	$M_{53}$	$M_{54}$	$M_{55}$	$M_{56}$
$M_{57}$	$M_{58}$	$M_{59}$	$M_{60}$	$M_{61}$	$M_{62}$	$M_{63}$	$M_{64}$

where  $M_i$  is a binary digit. Then the permutation  $X = IP(M)$  is as follows:

$M_{58}$	$M_{50}$	$M_{42}$	$M_{34}$	$M_{26}$	$M_{18}$	$M_{10}$	$M_2$
$M_{60}$	$M_{52}$	$M_{44}$	$M_{36}$	$M_{28}$	$M_{20}$	$M_{12}$	$M_4$
$M_{62}$	$M_{54}$	$M_{46}$	$M_{38}$	$M_{30}$	$M_{22}$	$M_{14}$	$M_6$
$M_{64}$	$M_{56}$	$M_{48}$	$M_{40}$	$M_{32}$	$M_{24}$	$M_{16}$	$M_8$
$M_{57}$	$M_{49}$	$M_{41}$	$M_{33}$	$M_{25}$	$M_{17}$	$M_9$	$M_1$
$M_{59}$	$M_{51}$	$M_{43}$	$M_{35}$	$M_{27}$	$M_{19}$	$M_{11}$	$M_3$
$M_{61}$	$M_{53}$	$M_{45}$	$M_{37}$	$M_{29}$	$M_{21}$	$M_{13}$	$M_5$
$M_{63}$	$M_{55}$	$M_{47}$	$M_{39}$	$M_{31}$	$M_{23}$	$M_{15}$	$M_7$

If we then take the inverse permutation  $Y = IP^{-1}(X) = IP^{-1}(IP(M))$ , it can be seen that the original ordering of the bits is restored.

### Details of Single Round

Figure 3.8 shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

$$L_i = R_{i-1}$$

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

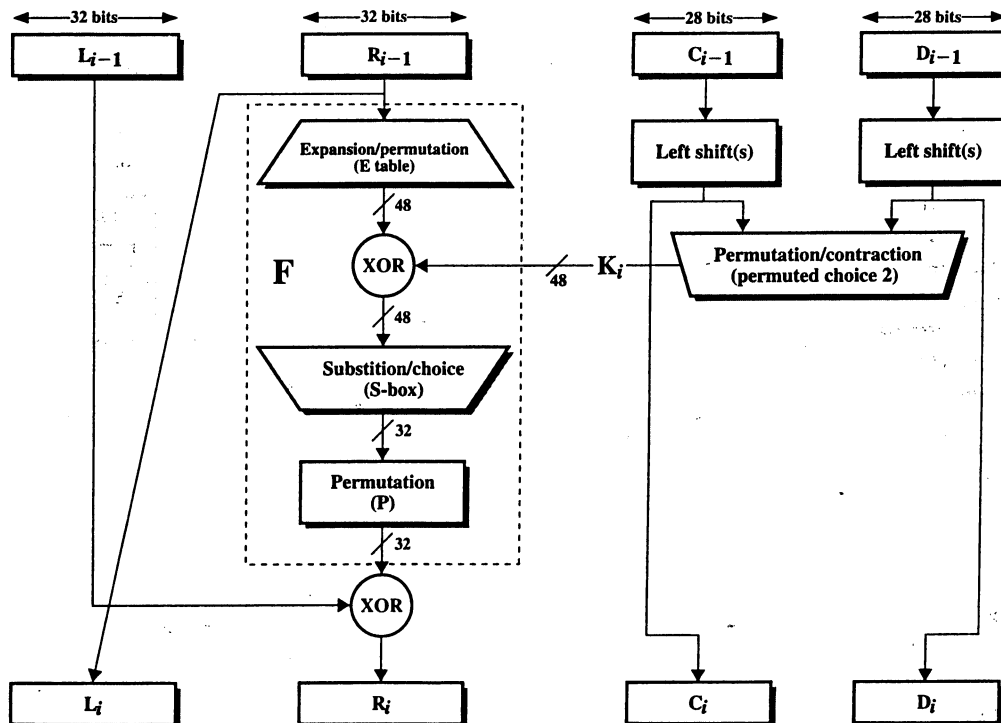


Figure 3.8 Single Round of DES Algorithm.

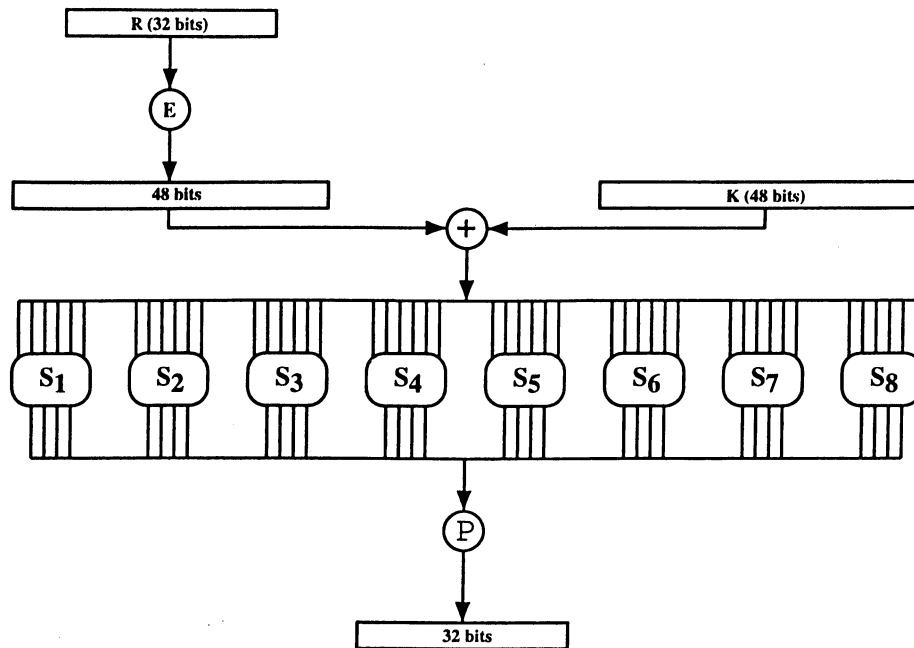


Figure 3.9 Calculation of  $F(R, K)$ .

The round key  $K_i$  is 48 bits. The  $R$  input is 32 bits. This  $R$  input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the  $R$  bits (Table 3.2c). The resulting 48 bits are XORed with  $K_i$ . This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by Table 3.2d.

The role of the S-boxes in the function  $F$  is illustrated in Figure 3.9. The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 3.3, which is interpreted as follows: The first and last bits of the input to box  $S_i$  form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for  $S_i$ . The middle 4 bits select a particular column. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output. For example, in  $S_1$ , for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.

Each row of an S-box defines a general reversible substitution. Figure 3.4 may be useful in understanding the mapping. The figure shows the substitution for row 0 of box  $S_1$ .

The structure of the S-boxes is worth further comment. Ignore for the moment the contribution of the key ( $K_i$ ). If you examine the expansion table, you see that the 32 bits of input are split into groups of 4 bits, and then become groups of 6 bits

971

**Table 3.3** Definition of DES S-Boxes

$S_1$	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

$S_2$	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

$S_3$	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

$S_4$	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

$S_5$	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

$S_6$	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

$S_7$	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

$S_8$	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

by taking the outer bits from the two adjacent groups. For example, if part of the input word is

... efgh ijkl mnop ...

this becomes

... defghi hijklm lnopq ...

The outer two bits of each group select one of four possible substitutions (one row of an S-box). Then a 4-bit output value is substituted for the particular 4-bit input (the middle four input bits). The 32-bit output from the eight S-boxes is then permuted, so that on the next round the output from each S-box immediately affects as many others as possible.

### Key Generation

Returning to Figures 3.7 and 3.8, we see that the 56-bit key used as input to the algorithm is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 3.4a). The resulting 56-bit key is then treated as two 28-bit quantities, labeled  $C_0$  and  $D_0$ . At each round,  $C_{i-1}$  and  $D_{i-1}$  are separately subjected to a circular left shift, or rotation, of 1 or 2 bits, as governed by Table 3.4c. These shifted values serve as input to the next round. They also serve as input to Permuted Choice Two (Table 3.4b), which produces a 48-bit output that serves as input to the function  $f(R_{i-1}, K_i)$ .

**Table 3.4** Tables Used for DES Key Schedule Calculation

(a) Permuted Choice One (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

(b) Permuted Choice Two (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

(c) Schedule of Left Shifts

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1



## DES Decryption

As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed.

## The Avalanche Effect

A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext. If the change were small, this might provide a way to reduce the size of the plaintext or key space to be searched.

DES exhibits a strong avalanche effect. Table 3.5 shows some results taken from [KONH81]. In Table 3.5, two plaintexts that differ by one bit were used:

```
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

with the key

```
0000001 1001011 0100100 1100010 0011100 0011000 0011100 0110010
```

The table shows that after just three rounds, 21 bits differ between the two blocks. On completion, the two ciphertexts differ in 34 bit positions.

Table 3.5b shows a similar test in which a single plaintext is input:

```
01101000 10000101 0010111 01111010 00010011 01110110 11101011 10100100
```

**Table 3.5** Avalanche Effect in DES

(a) Change in Plaintext		(b) Change in Key	
Round	Number of bits that differ	Round	Number of bits that differ
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

with two keys that differ in only one bit position:

```
1110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100
0110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100
```

Again, the results show that about half of the bits in the ciphertext differ and that the avalanche effect is pronounced after just a few rounds.

### 3.4 THE STRENGTH OF DES

Since its adoption as a federal standard, there have been lingering concerns about the level of security provided by DES. These concerns, by and large, fall into two areas: key size and the nature of the algorithm.

#### The Use of 56-Bit Keys

With a key length of 56 bits, there are  $2^{56}$  possible keys, which is approximately  $7.2 \times 10^{16}$  keys. Thus, at face value, a brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years (see Table 2.2) to break the cipher.

However, the assumption of one encryption per microsecond is overly conservative. As far back as 1977, Diffie and Hellman postulated that the technology existed to build a parallel machine with 1 million encryption devices, each of which could perform one encryption per microsecond [DIFF77]. This would bring the average search time down to about 10 hours. The authors estimated that the cost would be about \$20 million in 1977 dollars.

The most rigorous recent analysis of the problem was done by Wiener [WIEN93] and is based on a known plaintext attack. That is, it is assumed that the attacker has at least one (plaintext, ciphertext) pair. Wiener takes care to provide the details of his design. According to Wiener,

There have been numerous unverifiable claims about how fast the DES key space can be searched. To avoid adding to this list of questionable claims, a great deal of detail in the design of a key search machine is included in the appendices. This detailed work was done to obtain an accurate assessment of the cost of the machine and the time required to find a DES key. There are no plans to actually build such a machine.

Wiener reports on the design of a chip that uses pipelined techniques to achieve a key search rate of 50 million keys per second. Using 1993 costs, he designed a module that costs \$100,000 and contains 5760 key search chips. With this design, the following results are obtained:

Key Search Machine Unit Cost	Expected Search Time
\$100,000	35 hours
\$1,000,000	3.5 hours
\$10,000,000	21 minutes

In addition, Wiener estimates a one-time development cost of about \$500,000. In a 1997 update [WIEN97], Wiener sliced the times by a factor of six for the same cost (e.g., a \$100,000 machine has an expected search time of 6 hours).

Although Wiener's work is of major significance, it is a hypothetical design, not yet constructed. A more dramatic demonstration of the vulnerability of DES was the secret-key challenge issued by RSA Laboratories. The challenge, which offered a \$10,000 reward, was to find a DES key given a ciphertext for a plaintext consisting of an unknown plaintext message preceded by three known blocks of text containing the 24-character phrase "the unknown message is:". RSA issued the challenge on January 29, 1997. Responding to the challenge, Rocke Verser, an independent consultant, developed a brute-force program and distributed it over the Internet. The project linked numerous machines over the Internet and eventually grew to over 70,000 systems. As each new computer volunteer signed on, the project team created new portions of the DES key space for the new machine to test. The project began on February 18, 1997 and ended 96 days later when the correct key was found after examining about one-quarter of all possible keys [RSA97]. This challenge demonstrated the power of distributed personal computers to attack hard cryptographic problems.

However, there is more to a key-search attack than simply running through all possible keys. Unless known plaintext is provided, the analyst must be able to recognize plaintext as plaintext. If the message is just plain text in English, then the result pops out easily, although the task of recognizing English would have to be automated. If the text message has been compressed before encryption, then recognition is more difficult. And if the message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate. Thus, to supplement the brute-force approach, some degree of knowledge about the expected plaintext is needed, and some means of automatically distinguishing plaintext from garble is also needed.

The Wiener design and the secret-key challenge represent the culmination of years of concern about the security of DES and may, in retrospect, have been a turning point. As of the time of this writing, it still seems reasonable to rely on DES for personal and commercial applications. But the time has come to investigate alternatives for conventional encryption. Promising candidates for DES replacement include triple DES and several newer conventional encryption algorithms, discussed in Chapter 4.

### The Nature of the DES Algorithm

Another concern is the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. The focus of concern has been on the eight substitution tables, or S-boxes, that are used in each iteration. Because the design criteria for these boxes, and indeed for the entire algorithm, were not made public, there is a suspicion that the boxes were constructed in such a way that cryptanalysis is possible for an opponent who knows the weaknesses in the S-boxes. This assertion is tantalizing, and over the years a number of regularities and unexpected behaviors of the S-boxes have been discovered. Despite this, no one has so far succeeded in discovering the supposed fatal weaknesses in the S-boxes.<sup>8</sup>

---

<sup>8</sup>At least, no one has publicly acknowledged such a discovery.

### 3.5 DIFFERENTIAL AND LINEAR CRYPTANALYSIS

For most of its life, the prime concern with DES has been its vulnerability to brute-force attack because of its relatively short (56 bits) key length. However, there has also been interest in finding cryptanalytic attacks on DES. With the increasing popularity of block ciphers with longer key lengths, including triple DES, brute-force attacks have become increasingly impractical. Thus, there has been increased emphasis on cryptanalytic attacks on DES and other symmetric block ciphers. In this section, we provide a brief overview of the two most powerful and promising approaches: differential cryptanalysis and linear cryptanalysis.

#### Differential Cryptanalysis

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. In this section, we discuss the technique and its applicability to DES.

##### History

Differential cryptanalysis was not reported in the open literature until 1990. The first published effort appears to have been the cryptanalysis of a block cipher called FEAL by Murphy [MURP90]. This was followed by a number of papers by Biham and Shamir, who demonstrated this form of attack on a variety of encryption algorithms and hash functions; their results are summarized in [BIHA93].

The most publicized results for this approach have been those that have application to DES. Differential cryptanalysis is the first published attack that is capable of breaking DES in less than  $2^{55}$  complexity. The scheme, as reported in [BIHA93], can successfully cryptanalyze DES with an effort on the order of  $2^{47}$ , requiring  $2^{47}$  chosen plaintexts. Although  $2^{47}$  is certainly significantly less than  $2^{55}$ , the need to find  $2^{47}$  chosen plaintexts makes this attack of only theoretical interest.

Although differential cryptanalysis is a powerful tool, it does not do very well against DES. The reason, according to a member of the IBM team that designed DES [COPP94], is that differential cryptanalysis was known to the team as early as 1974. The need to strengthen DES against attacks using differential cryptanalysis played a large part in the design of the S-boxes and the permutation P. As evidence of the impact of these changes, consider these comparable results reported in [BIHA93]. Differential cryptanalysis of an eight-round LUCIFER algorithm requires only 256 chosen plaintexts, whereas an attack on an eight-round version of DES requires  $2^{14}$  chosen plaintexts.

Differential cryptanalysis also influenced the design of IDEA, one of the most important conventional encryption algorithms since DES (described in Chapter 4). The designers of IDEA were unaware of differential cryptanalysis when they published their original design, known as the Proposed Encryption Standard (PES) [LAI90]. PES uses a 128-bit key yet is vulnerable to differential cryptanalysis with a level of effort of only  $2^{64}$ . The authors made minor modifications to PES to produce the IDEA design [LAI91], and they claim that IDEA is invulnerable to differential cryptanalysis.

### Differential Cryptanalysis Attack

The differential cryptanalysis attack is complex; [BIHA93] provides a complete description. Here, we provide a brief overview so that you can get the flavor of the attack.

We begin with a change in notation for DES. Consider the original plaintext block  $m$  to consist of two halves  $m_0, m_1$ . Each round of DES maps the right-hand input into the left-hand output and sets the right-hand output to be a function of the left-hand input and the subkey for this round. So, at each round, only one new 32-bit block is created. If we label each new block  $m_i$  ( $2 \leq i \leq 17$ ), then the intermediate message halves are related as follows:

$$m_{i+1} = m_{i-1} \oplus f(m_i, K_i), \quad i = 1, 2, \dots, 16$$

In differential cryptanalysis, we start with two messages,  $m$  and  $m'$ , with a known XOR difference  $\Delta m = m \oplus m'$ , and consider the difference between the intermediate message halves:  $\Delta m_i = m_i \oplus m'_i$ . Then we have

$$\begin{aligned} \Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus f(m_i, K_i)] \oplus [m'_{i-1} \oplus f(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [f(m_i, K_i) \oplus f(m'_i, K_i)] \end{aligned}$$

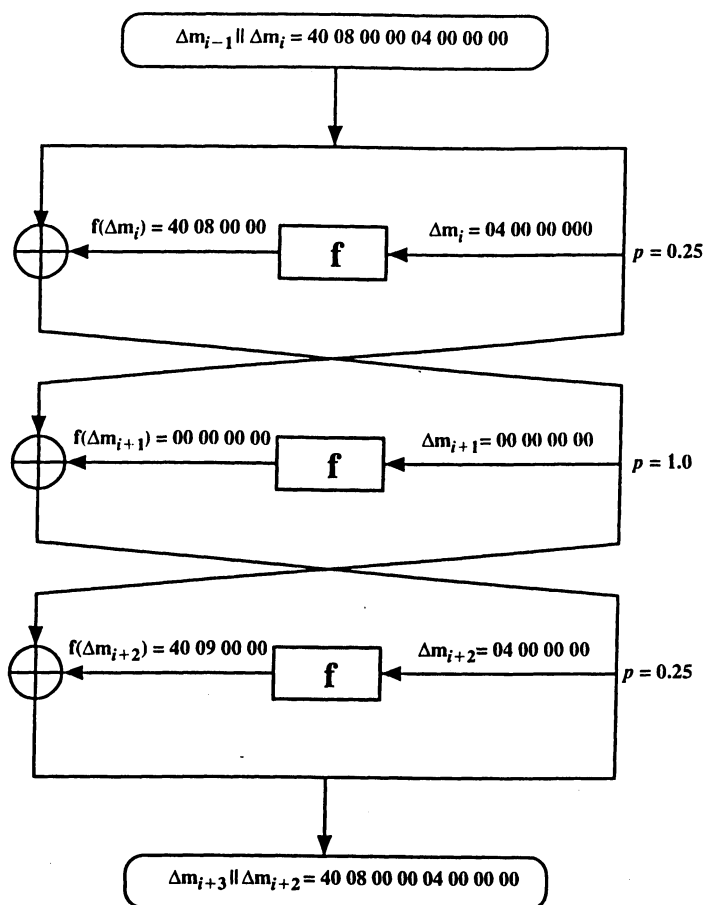
Suppose that many pairs of inputs to  $f$  with the same difference yield the same output difference if the same subkey is used. To put this more precisely, let us say that  $X$  may cause  $Y$  with probability  $p$ , if for a fraction  $p$  of the pairs in which the input XOR is  $X$ , the output XOR equals  $Y$ . We want to suppose that there are a number of values of  $X$  that have high probability of causing a particular output difference. Therefore, if we know  $\Delta m_{i-1}$  and  $\Delta m_i$  with high probability, then we know  $\Delta m_{i+1}$  with high probability. Furthermore, if a number of such differences are determined, it is feasible to determine the subkey used in the function  $f$ .

The overall strategy of differential cryptanalysis is based on these considerations for a single round. The procedure is to begin with two plaintext messages  $m$  and  $m'$  with a given difference and trace through a probable pattern of differences after each round to yield a probable difference for the ciphertext. Actually, there are two probable differences for the two 32-bit halves:  $(\Delta m_{17} || \Delta m_{16})$ . Next, we submit  $m$  and  $m'$  for encryption to determine the actual difference under the unknown key and compare the result to the probable difference. If there is a match,

$$E_k(m) \oplus E_k(m') = (\Delta m_{17} || \Delta m_{16})$$

then we suspect that all the probable patterns at all the intermediate rounds are correct. With that assumption, we can make some deductions about the key bits. This procedure must be repeated many times to determine all the key bits.

Figure 3.10, based on a figure in [BIHA93], illustrates the propagation of differences through three rounds of DES. The probabilities shown on the right refer to the probability that a given set of intermediate differences will appear as a function of the input differences. Overall, after three rounds the probability that the output difference is as shown is equal to  $0.25 \times 1 \times 0.25 = 0.0625$ .



**Figure 3.10** Differential Propagation through Three Rounds of DES (numbers in hexadecimal).

### Linear Cryptanalysis

A more recent development is linear cryptanalysis, described in [MATS93]. This attack is based on finding linear approximations to describe the transformations performed in DES. This method can find a DES key given  $2^{47}$  known plaintexts, as compared to  $2^{47}$  chosen plaintexts for differential cryptanalysis. Although this is a minor improvement, because it may be easier to acquire known plaintext rather than chosen plaintext, it still leaves linear cryptanalysis infeasible as an attack on DES. So far, little work has been done by other groups to validate the linear cryptanalytic approach.

We now give a brief summary of the principle on which linear cryptanalysis is based. For a cipher with  $n$ -bit plaintext and ciphertext blocks and an  $m$ -bit key, let

the plaintext block be labeled  $P[1], \dots, P[n]$ , the cipher text block  $C[1], \dots, C[n]$ , and the key  $K[1], \dots, K[m]$ . Then define

$$A[i, j, \dots, k] = A[i] \oplus A[j] \oplus \dots \oplus A[k]$$

The objective of linear cryptanalysis is to find an effective *linear* equation of the form

$$P[\alpha_1, \alpha_2, \dots, \alpha_a] \oplus C[\beta_1, \beta_2, \dots, \beta_b] = K[\gamma_1, \gamma_2, \dots, \gamma_c]$$

(where  $x = 0$  or  $1$ ;  $1 \leq a, b \leq n$ ,  $1 \leq c \leq m$ , and where the  $\alpha$ ,  $\beta$ , and  $\gamma$  terms represent fixed, unique bit locations) that holds with probability  $p \neq 0.5$ . The further  $p$  is from  $0.5$ , the more effective the equation. Once a proposed relation is determined, the procedure is to compute the results of the left-hand side of the preceding equation for a large number of plaintext-ciphertext pairs. If the result is  $0$  more than half the time, assume  $K[\gamma_1, \gamma_2, \dots, \gamma_c] = 0$ . If it is  $1$  most of the time, assume  $K[\gamma_1, \gamma_2, \dots, \gamma_c] = 1$ . This gives us a linear equation on the key bits. Try to get more such relations so that we can solve for the key bits. Because we are dealing with linear equations, the problem can be approached one round of the cipher at a time, with the results combined.

### 3.6 BLOCK CIPHER DESIGN PRINCIPLES

Although much progress has been made in designing block ciphers that are cryptographically strong, the basic principles have not changed all that much since the work of Feistel and the DES design team in the early 1970s. It is useful to begin this discussion by looking at the published design criteria used in the DES effort. Then we look at three critical aspects of block cipher design: the number of rounds, design of the function  $F$ , and key scheduling.

#### DES Design Criteria

The criteria used in the design of DES, as reported in [COPP94], focused on the design of the S-boxes and on the  $P$  function that takes the output of the S boxes (Figure 3.9). The criteria for the S-boxes are as follows:

1. No output bit of any S-box should be too close a linear function of the input bits. Specifically, if we select any output bit and any subset of the six input bits, the fraction of inputs for which this output bit equals the XOR of these input bits should not be close to  $0$  or  $1$ , but rather should be near  $\frac{1}{2}$ .
2. Each row of an S-box (determined by a fixed value of the leftmost and rightmost input bits) should include all 16 possible output bit combinations.
3. If two inputs to an S-box differ in exactly one bit, the outputs must differ in at least two bits.
4. If two inputs to an S-box differ in the two middle bits exactly, the outputs must differ in at least two bits.

5. If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same.
6. For any nonzero 6-bit difference between inputs, no more than 8 of the 32 pairs of inputs exhibiting that difference may result in the same output difference.
7. This is a criterion similar to the previous one, but for the case of three S-boxes.

Coppersmith pointed out that the first criterion in the preceding list was needed because the S-boxes are the only nonlinear part of DES. If the S-boxes were linear (i.e., each output bit is a linear combination of the input bits), the entire algorithm would be linear and easily broken. We have seen this phenomenon with the Hill cipher, which is linear. The remaining criteria were primarily aimed at thwarting differential cryptanalysis and at providing good confusion properties.

The criteria for the permutation P are as follows:

1. The four output bits from each S-box at round  $i$  are distributed so that two of them affect (provide input for) “middle bits” of round  $(i + 1)$  and the other two affect end bits. The two middle bits of input to an S-box are not shared with adjacent S-boxes. The end bits are the two left-hand bits and the two right-hand bits, which are shared with adjacent S-boxes.
2. The four output bits from each S-box affect six different S-boxes on the next round, and no two affect the same S-box.
3. For two S-boxes  $j, k$ , if an output bit from  $S_j$  affects a middle bit of  $S_k$  on the next round, then an output bit from  $S_k$  cannot affect a middle bit of  $S_j$ . This implies that for  $j = k$ , an output bit from  $S_j$  must not affect a middle bit of  $S_j$ .

These criteria are intended to increase the diffusion of the algorithm.

### Number of Rounds

The cryptographic strength of a Feistel cipher derives from three aspects of the design: the number of rounds, the function F, and the key schedule algorithm. Let us look first at the choice of the number of rounds.

The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F. In general, the criterion should be that the number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple brute-force key search attack. This criterion was certainly used in the design of DES. Schneier [SCHN96] observes that for 16-round DES, a differential cryptanalysis attack is slightly less efficient than brute force: The differential cryptanalysis attack requires  $2^{55.1}$  operations,<sup>9</sup> whereas brute force requires  $2^{55}$ . If DES had 15 or fewer rounds, then differential cryptanalysis would require less effort than brute-force key search.

This criterion is attractive because it makes it easy to judge the strength of an algorithm and to compare different algorithms. In the absence of a cryptanalytic breakthrough, the strength of any algorithm that satisfies the criterion can be judged solely on key length.

<sup>9</sup>Recall that differential cryptanalysis of DES requires  $2^{47}$  chosen plaintext. If all you have to work with is known plaintext, then you must sort through a large quantity of known plaintext-ciphertext pairs looking for the useful ones. This brings the level of effort up to  $2^{55.1}$ .



## Design of Function F

The heart of a Feistel block cipher is the function  $F$ . As we have seen, in DES, this function relies on the use of S-boxes. This is also the case for most other symmetric block ciphers, as we shall see in Chapter 4. However, we can make some general comments about the criteria for designing  $F$ . After that, we look specifically at S-box design.

### Design Criteria for F

The function  $F$  provides the element of confusion in a Feistel cipher. Thus, it must be difficult to “unscramble” the substitution performed by  $F$ . One obvious criterion is that  $F$  be nonlinear, as we discussed previously. The more nonlinear  $F$ , the more difficult any type of cryptanalysis will be. There are several measures of nonlinearity, which are beyond the scope of this book. In rough terms, the more difficult it is to approximate  $F$  by a set of linear equations, the more nonlinear  $F$  is.

Several other criteria should be considered in designing  $F$ . We would like the algorithm to have good avalanche properties. Recall that, in general, this means that a change in one bit of the input should produce a change in many bits of the output. A more stringent version of this is the **strict avalanche criterion (SAC)** [WEBS86], which states that any output bit  $j$  of an S-box should change with probability  $\frac{1}{2}$  when any single input bit  $i$  is inverted for all  $i, j$ . Although SAC is expressed in terms of S-boxes, a similar criterion could be applied to  $F$  as a whole. This is important when considering designs that do not include S-boxes.

Another criterion proposed in [WEBS86] is the **bit independence criterion (BIC)**, which states that output bits  $j$  and  $k$  should change independently when any single input bit  $i$  is inverted, for all  $i, j$ , and  $k$ . The SAC and BIC criteria appear to strengthen the effectiveness of the confusion function.

### S-Box Design

One of the most intense areas of research in the field of symmetric block ciphers is that of S-box design. The papers are almost too numerous to count.<sup>10</sup> Here we mention some general principles. In essence, we would like any change to the input vector to an S-box to result in random-looking changes to the output. The relationship should be nonlinear and difficult to approximate with linear functions.

One obvious characteristic of the S-box is its size. An  $n \times m$  S-box has  $n$  input bits and  $m$  output bits. DES has  $6 \times 4$  S-boxes. Both Blowfish and CAST, described in Chapter 4, have  $8 \times 32$  S-boxes. Larger S-boxes, by and large, are more resistant to differential and linear cryptanalysis [SCHN96]. On the other hand, the larger the dimension  $n$ , the (exponentially) larger the lookup table. Thus, for practical reasons, a limit of  $n$  equal to about 8 to 10 is usually imposed. Another practical consideration is that the larger the S-box, the more difficult it is to design it properly.

S-boxes are typically organized different from the manner used in DES. An  $n \times m$  S-box typically consists of  $2^n$  rows of  $m$  bits each. The  $n$  bits of input select one of the rows of the S-box, and the  $m$  bits in that row are the output. For example,

<sup>10</sup> A good summary of S-box design studies through early 1996 can be found in [SCHN96].

in an  $8 \times 32$  S-box, if the input is 00001001, the output consists of the 8 bits in row 9 (the first row is labeled row 0).

Mister and Adams [MIST96] propose a number of criteria for S-box design. Among these are that the S-box should satisfy both SAC and BIC. Mister and Adams also suggest that all linear combinations of S-box columns should be *bent*. Bent functions are a special class of Boolean functions that are highly nonlinear according to certain mathematical criteria [ADAM90]. This is an area beyond the scope of this book, but a brief introduction is provided in Appendix 3A. There has been increasing interest in designing and analyzing S-boxes using bent functions.

A related criterion for S-boxes is proposed and analyzed in [HEYS95]. The authors define the **guaranteed avalanche (GA)** criterion as follows: An S-box satisfies GA of order  $\gamma$  if, for a 1-bit input change, at least  $\gamma$  output bits change. The authors conclude that a GA in the range of order 2 to order 5 provides strong diffusion characteristics for the overall encryption algorithm.

For larger S-boxes, such as  $8 \times 32$ , the question arises as to the best method of selecting the S-box entries in order to meet the type of criteria we have been discussing. Nyberg, who has written a lot about the theory and practice of S-box design, suggests the following approaches (quoted in [ROBS95]):

- **Random:** Use some pseudorandom number generation or some table of random digits to generate the entries in the S-boxes. This may lead to boxes with undesirable characteristics for small sizes (e.g.,  $6 \times 4$ ) but should be acceptable for large S-boxes (e.g.,  $8 \times 32$ ).
- **Random with testing:** Choose S-box entries randomly, and then test the results against various criteria and throw away those that do not pass.
- **Man-made:** This is a more or less manual approach with only simple mathematics to support it. It is apparently the technique used in the DES design. This approach is difficult to carry through for large S-boxes.
- **Math-made:** Generate S-boxes according to mathematical principles. By using mathematical construction, S-boxes can be constructed that offer proven security against linear and differential cryptanalysis, together with good diffusion. This is the approach taken with CAST, described in Chapter 4.

A variation on the first technique is to use S-boxes that are both random and key dependent. An example of this approach is Blowfish, described in Chapter 4, which starts with S-boxes filled with pseudorandom digits and then alters the contents using the key. A tremendous advantage of key-dependent S-boxes is that, because they are not fixed, it is impossible to analyze the S-boxes ahead of time to look for weaknesses.

### Key Schedule Algorithm

A final area of block cipher design, and one that has received less attention than S-box design, is the key schedule algorithm. With any Feistel block cipher, the key is used to generate one subkey for each round. In general, we would like to select subkeys to maximize the difficulty of deducing individual subkeys and the

difficulty of working back to the main key. No general principles for this have yet been promulgated.

Hall suggests [ADAM94] that, at minimum, the key schedule should guarantee key/ciphertext strict avalanche criterion and bit independence criterion.

### 3.7 BLOCK CIPHER MODES OF OPERATION

The DES algorithm is a basic building block for providing data security. To apply DES in a variety of applications, four “modes of operation” have been defined (FIPS PUB 74, 81). These four modes are intended to cover virtually all the possible applications of encryption for which DES could be used. The modes are summarized in Table 3.6 and described briefly in the remainder of this section. These same modes can be applied for any symmetric block cipher.

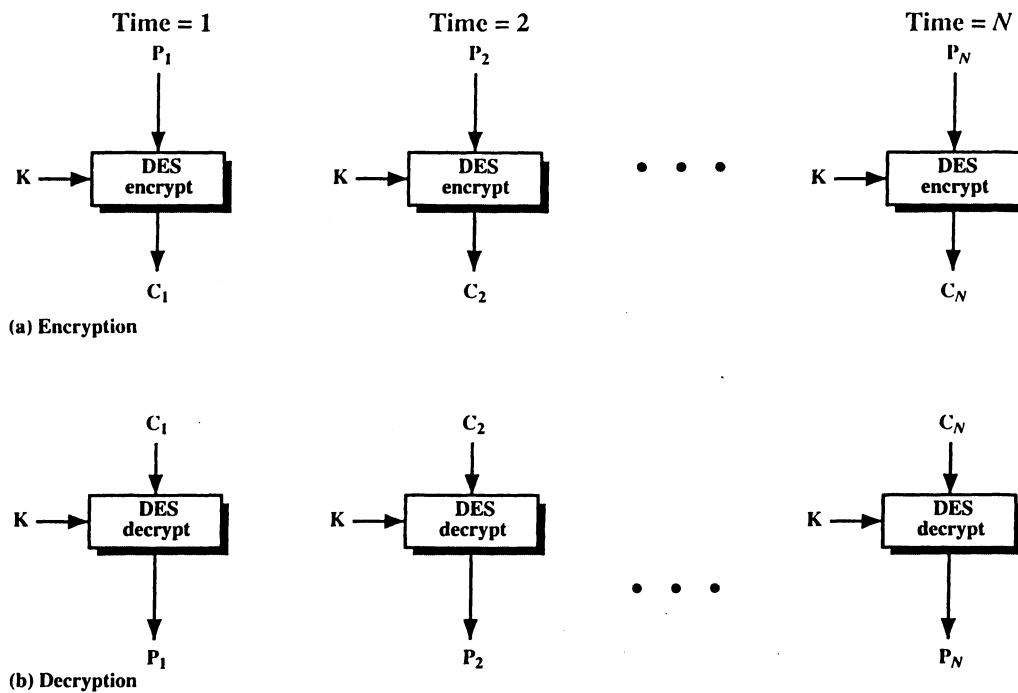
#### Electronic Codebook Mode

The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled 64 bits at a time and each block of plaintext is encrypted using the same key (Figure 3.11). The term *codebook* is used because, for a given key, there is a unique ciphertext for every 64-bit block of plaintext. Therefore, one can imagine a gigantic codebook in which there is an entry for every possible 64-bit plaintext pattern showing its corresponding ciphertext.

For a message longer than 64 bits, the procedure is simply to break the message into 64-bit blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key. In Figure 3.11, the plaintext (padded as necessary) consists of a sequence of 64-bit blocks,  $P_1, P_2, \dots, P_N$ ; the corresponding sequence of ciphertext blocks is  $C_1, C_2, \dots, C_N$ .

**Table 3.6** DES Modes of Operation

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	<ul style="list-style-type: none"> <li>Secure transmission of single values (e.g., an encryption key)</li> </ul>
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	<ul style="list-style-type: none"> <li>General-purpose block-oriented transmission</li> <li>Authentication</li> </ul>
Cipher Feedback (CFB)	Input is processed J bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	<ul style="list-style-type: none"> <li>General-purpose stream-oriented transmission</li> <li>Authentication</li> </ul>
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding DES output.	<ul style="list-style-type: none"> <li>Stream-oriented transmission over noisy channel (e.g., satellite communication)</li> </ul>



**Figure 3.11** Electronic Codebook (ECB) Mode.

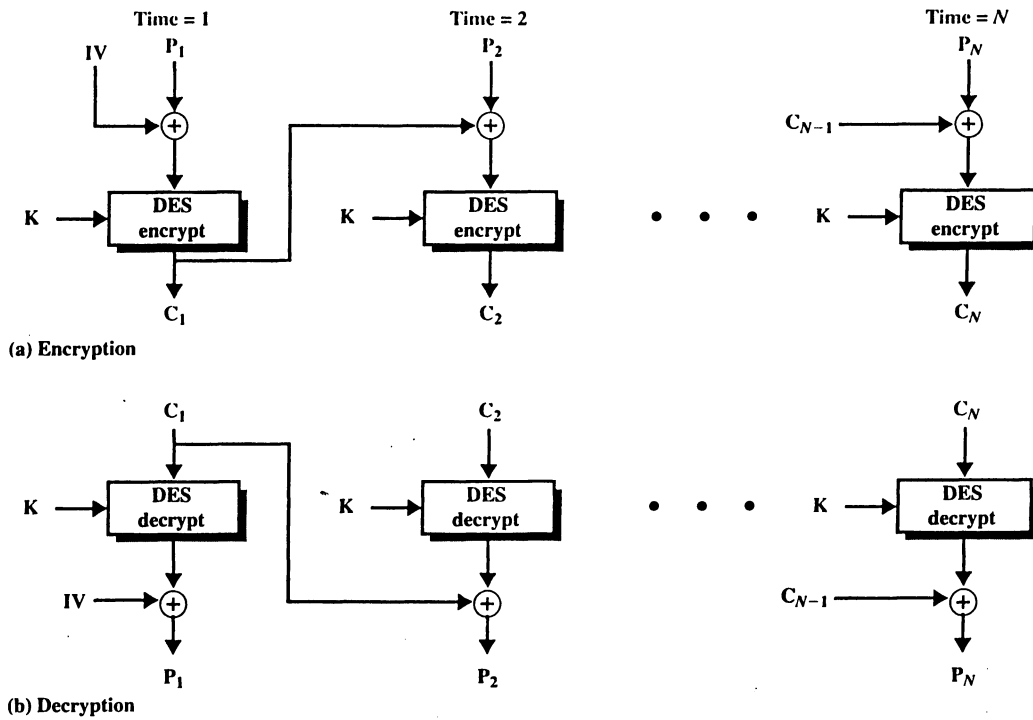
The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES key securely, ECB is the appropriate mode to use.

The most significant characteristic of ECB is that the same 64-bit block of plaintext, if it appears more than once in the message, always produces the same ciphertext.

For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities. For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext-ciphertext pairs to work with. If the message has repetitive elements, with a period of repetition a multiple of 64 bits, then these elements can be identified by the analyst. This may help in the analysis or may provide an opportunity for substituting or rearranging blocks.

### Cipher Block Chaining Mode

To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. A simple way to satisfy this requirement is the cipher block chaining (CBC) mode (Figure 3.12). In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. In effect, we have chained together the processing of the sequence of



**Figure 3.12** Cipher Block Chaining (CBC) Mode.

plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of 64 bits are not exposed.

For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block. To see that this works, we can write

$$C_n = E_K[C_{n-1} \oplus P_n]$$

Then

$$D_K[C_n] = D_K[E_K(C_{n-1} \oplus P_n)]$$

$$D_K[C_n] = (C_{n-1} \oplus P_n)$$

$$C_{n-1} \oplus D_K[C_n] = C_{n-1} \oplus C_{n-1} \oplus P_n = P_n$$

To produce the first block of ciphertext, an initialization vector (IV) is XORed with the first block of plaintext. On decryption, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext.

The IV must be known to both the sender and receiver. For maximum security, the IV should be protected as well as the key. This could be done by sending the IV using ECB encryption. One reason for protecting the IV is as follows: If an

opponent is able to fool the receiver into using a different value for IV, then the opponent is able to invert selected bits in the first block of plaintext. To see this, consider the following:

$$C_1 = E_k(IV \oplus P_1)$$

$$P_1 = IV \oplus D_k(C_1)$$

Now use the notation that  $X[i]$  denotes the  $i$ th bit of the 64-bit quantity  $X$ . Then

$$P_1[i] = IV[i] \oplus D_k(C_1)[i]$$

Then, using the properties of XOR, we can state

$$P_1[i]' = IV[i]' \oplus D_k(C_1)[i]$$

where the prime notation denotes bit complementation. This means that if an opponent can predictably change bits in IV, the corresponding bits of the received value of  $P_1$  can be changed.

For other possible attacks based on knowledge of IV, see [VOYD83].

In conclusion, because of the chaining mechanism of CBC, it is an appropriate mode for encrypting messages of length greater than 64 bits.

In addition to its use to achieve confidentiality, the CBC mode can be used for authentication. This use is described in Chapter 8.

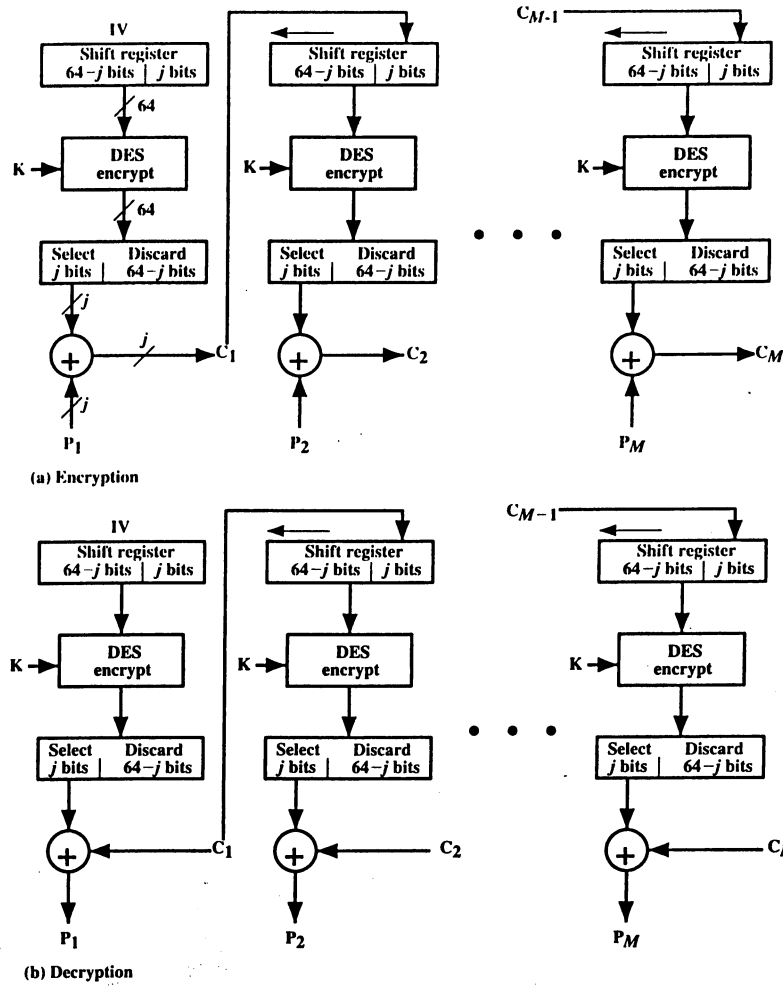
### Cipher Feedback Mode

The DES scheme is essentially a block cipher technique that uses 64-bit blocks. However, it is possible to convert DES into a stream cipher, using either the cipher feedback (CFB) or the output feedback mode. A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.

One desirable property of a stream cipher is that the ciphertext be of the same length as the plaintext. Thus, if 8-bit characters are being transmitted, each character should be encrypted using 8 bits. If more than 8 bits are used, transmission capacity is wasted.

Figure 3.13 depicts the CFB scheme. In the figure, it is assumed that the unit of transmission is  $j$  bits; a common value is  $j = 8$ . As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext.

First, consider encryption. The input to the encryption function is a 64-bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant)  $j$  bits of the output of the encryption function are XORed with the first unit of plaintext  $P_1$  to produce the first unit of ciphertext  $C_1$ , which is then transmitted. In addition, the contents of the shift register are shifted left by  $j$  bits and  $C_1$  is placed in the rightmost (least significant)  $j$  bits of the shift register. This process continues until all plaintext units have been encrypted.



**Figure 3.13** *J*-Bit Cipher Feedback (CFB) Mode.

For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit. Note that it is the *encryption* function that is used, not the decryption function. This is easily explained. Let  $S_j(X)$  be defined as the most significant  $j$  bits of  $X$ . Then

$$C_1 = P_1 \oplus S_j(E(IV))$$

Therefore,

$$P_1 = C_1 \oplus S_j(E(IV))$$

The same reasoning holds for subsequent steps in the process.

In addition to its use to achieve confidentiality, the CFB mode can be used for authentication. This procedure is described in Chapter 8.

### Output Feedback Mode

The output feedback (OFB) mode is similar in structure to that of CFB, as illustrated in Figure 3.14. As can be seen, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register.

One advantage of the OFB method is that bit errors in transmission do not propagate. For example, if a bit error occurs in  $C_1$ , only the recovered value of  $P_1$  is affected; subsequent plaintext units are not corrupted. With CFB,  $C_1$  also serves as input to the shift register and therefore causes additional corruption downstream.

The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB. Consider that complementing a bit in the ciphertext complements the corresponding bit in the recovered plaintext. Thus, controlled changes to the recovered plaintext can be made. This may make it possible for an

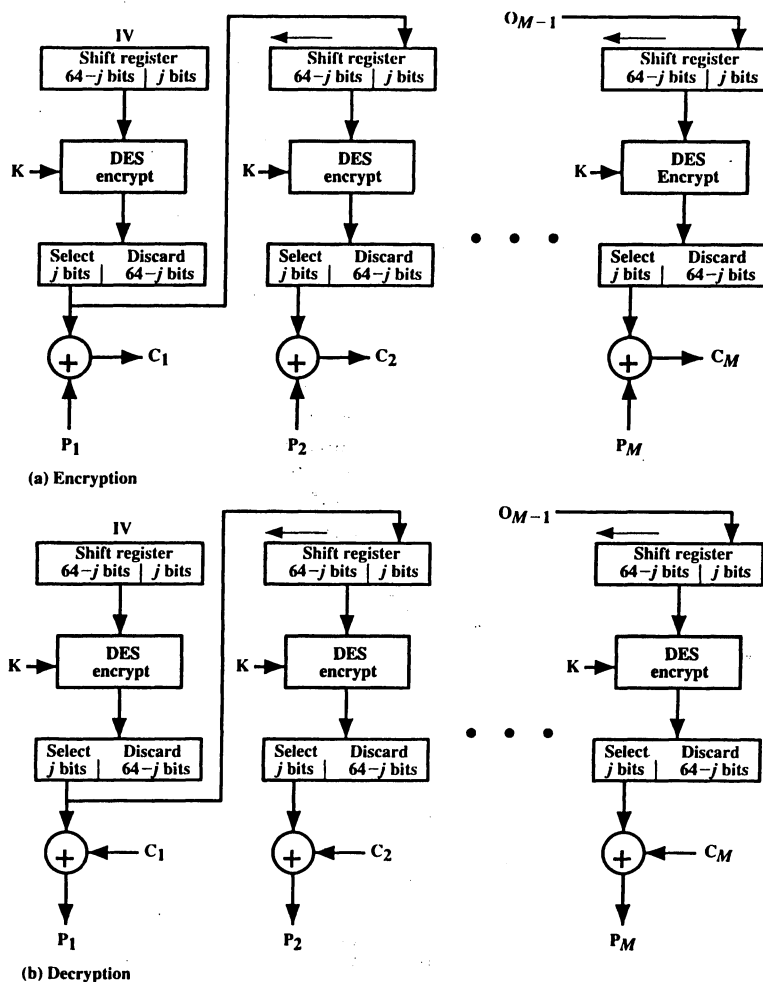


Figure 3.14 L-Bit Output Feedback (OFB) Mode



opponent, by making the necessary changes to the checksum portion of the message as well as to the data portion, to alter the ciphertext in such a way that it is not detected by an error-correcting code. For a further discussion, see [VOYD83].

### 3.8 RECOMMENDED READING

There is a wealth of information on conventional encryption. Some of the more worthwhile references are listed here.

An essential reference work is [SCHN96]. This remarkable work contains descriptions of virtually every cryptographic algorithm and protocol published in the last 15 years. The author pulls together results from journals, conference proceedings, government publications, and standards documents and organizes these into a comprehensive and comprehensible survey. Another worthwhile and detailed survey is [MEME97].

For coverage of contemporary cryptology, Simmons's book of that same name is unequalled [SIMM92]. The book is actually a collection of papers, some original and some of which are revised from earlier versions that appeared in the May 1988 issue of the *Proceedings of the IEEE*. The book provides an in-depth survey of the field. Another excellent treatment, with rigorous mathematical discussion, is [STIN95].

The foregoing references provide coverage of public-key as well as conventional encryption.

[WIEN93] is the definitive treatment of brute-force attacks on DES. [COPP94] looks at the inherent strength of DES and its ability to stand up to cryptanalysis. [BARK91] provides a readable and interesting analysis of the structure of DES and of potential cryptanalytic approaches to DES.

**BARK91** Barker, W. *Introduction to the Analysis of the Data Encryption Standard (DES)*. Laguna Hills, CA: Aegean Park Press, 1991.

**COPP94** Coppersmith, D. "The Data Encryption Standard (DES) and Its Strength Against Attacks." *IBM Journal of Research and Development*, May 1994.

**MEME97** Menezes, A.; Oorschot, P.; and Vanstone, S. *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1997.

**SCHN96** Schneier, B. *Applied Cryptography*. New York: Wiley, 1996.

**SIMM92** Simmons, G., ed. *Contemporary Cryptology: The Science of Information Integrity*. Piscataway, NJ: IEEE Press, 1992.

**STIN95** Stinson, D. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 1995.

**WIEN93** Wiener, M. "Efficient DES Key Search." *Proceedings, Crypto '93*, 1993; published by Springer-Verlag.

### 3.9 PROBLEMS

- 3.1 Refer to Figure 3.2, which depicts key generation for S-DES.
  - a. How important is the initial P10 permutation function?
  - b. How important are the two LS-1 shift functions?
- 3.2 The equations for the variables  $q$  and  $r$  for S-DES are defined in the section on S-DES analysis. Provide the equations for  $s$  and  $t$ .
- 3.3 Using S-DES, decrypt the string (10100010) using the key (011111101) by hand. Show intermediate results after each function ( $IP, F_K, SW, F_K, IP^{-1}$ ). Then decode the first 4 bits of the plaintext string to a letter and the second 4 bits to another letter where we encode A through P in base 2 (i.e., A = 0000, B = 0001, ..., P = 1111). *Hint:* As a mid-way check, after the application of SW, the string should be (00010011).

- 3.4** Let  $\pi$  be a permutation of the integers  $1, 2, \dots, 2^n - 1$ , such that  $\pi(m)$  gives the permuted value of  $m$ ,  $0 \leq m \leq 2^n$ . Put another way,  $\pi$  maps the set of  $n$ -bit integers into itself and no two integers map into the same integer. DES is such a permutation for 64-bit integers. We say that  $\pi$  has a fixed point at  $m$  if  $\pi(m) = m$ . That is, if  $\pi$  is an encryption mapping, then a fixed point corresponds to a message that encrypts to itself. We are interested in the probability that  $\pi$  has no fixed points. Show the somewhat unexpected result that over 60% of mappings will have at least one fixed point.
- 3.5** Consider a block encryption algorithm that encrypts blocks of length  $n$ , and let  $N = 2^n$ . Say we have  $t$  plaintext-ciphertext pairs  $P_i, C_i = E_K[P_i]$  where we assume that the key  $K$  selects one of the  $N!$  possible mappings. Imagine that we wish to find  $K$  by exhaustive search. We could generate key  $K'$  and test whether  $C_i = E_{K'}(P_i)$  for  $1 \leq i \leq t$ . If  $K'$  encrypts each  $P_i$  to its proper  $C_i$  then we have evidence that  $K = K'$ . However, it may be the case that the mappings  $E_K(\bullet)$  and  $E_{K'}(\bullet)$  exactly agree on the  $t$  plaintext-ciphertext pairs  $P_i, C_i$  and agree on no other pairs.
- What is the probability that  $E_K(\bullet)$  and  $E_{K'}(\bullet)$  are in fact distinct mappings?
  - What is the probability that  $E_K(\bullet)$  and  $E_{K'}(\bullet)$  agree on another  $t'$  plaintext-ciphertext pairs where  $0 \leq t' \leq N - t$ ?
- 3.6** Show that DES decryption is, in fact, the inverse of DES encryption.
- 3.7** The 32-bit swap after the sixteenth iteration of the DES algorithm is needed to make the encryption process invertible by simply running the ciphertext back through the algorithm with the key order reversed. This was demonstrated in Problem 3.6. However, it still may not be entirely clear why the 32-bit swap is needed. To demonstrate why, solve the following exercises. First, some notation:

$A||B$  = the concatenation of the bit strings  $A$  and  $B$

$T_i(R||L)$  = the transformation defined by the  $i$ th iteration of the encryption algorithm, for  $1 \leq i \leq 16$

$TD_i(R||L)$  = the transformation defined by the  $i$ th iteration of the decryption algorithm, for  $1 \leq i \leq 16$

$T_{17}(R||L) = L||R$ . This transformation occurs after the sixteenth iteration of the encryption algorithm.

- Show that the composition  $TD_1(IP(IP^{-1}(T_{17}(T_{16}(L_{15}||R_{15}))))$  is equivalent to the transformation that interchanges the 32-bit halves,  $L_{15}$  and  $R_{15}$ . That is, show that

$$TD_1(IP(IP^{-1}(T_{17}(T_{16}(L_{15}||R_{15})))) = R_{15}||L_{15}$$

- Now suppose that we did away with the final 32-bit swap in the encryption algorithm. Then we would want the following equality to hold:

$$TD_1(IP(IP^{-1}(T_{16}(L_{15}||R_{15})))) = L_{15}||R_{15}$$

Does it?

- 3.8** Compare the Initial Permutation table (Table 3.2a) with the Permuted Choice One table (Table 3.4a). Are the structures similar? If so, describe the similarities. What conclusions can you draw from this analysis?
- 3.9** When using the DES algorithm for decryption, the 16 keys ( $K_1, K_2, \dots, K_{16}$ ) are used in reverse order. Therefore, the right hand side of Figure 3.8 is no longer valid. Design a key-generation scheme with the appropriate shift schedule (analogous to Table 3.4c) for the decryption process.
- 3.10 a.** Let  $M'$  be the bitwise complement of  $M$ . Prove that if the complement of the plaintext block is taken and the complement of an encryption key is taken, then the result of encryption with these values is the complement of the original ciphertext. That is,

$$\begin{aligned} \text{If } Y &= \text{DES}_K(X) \\ \text{Then } Y' &= \text{DES}_{K'}(X') \end{aligned}$$

*Hint:* Begin by showing that for any two bit strings of equal length,  $A$  and  $B$ ,  $(A \oplus B)'$

- b. It has been said that a brute-force attack on DES requires searching a key space of  $2^{56}$  keys. Does the result of part (a) change that?
- 3.11 Show that in DES the first 24 bits of each subkey come from the same subset of 28 bits of the initial key and that the second 24 bits of each subkey come from a disjoint subset of 28 bits of the initial key.
- 3.12 With the ECB mode of DES, if there is an error in a block of the transmitted ciphertext, only the corresponding plaintext block is affected. However, in the CBC mode, this error propagates. For example, an error in the transmitted  $C_1$  (Figure 3.12) obviously corrupts  $P_1$  and  $P_2$ .
- Are any blocks beyond  $P_2$  affected?
  - Suppose that there is a bit error in the source version of  $P_1$ . Through how many ciphertext blocks is this error propagated? What is the effect at the receiver?
- 3.13 If a bit error occurs in the transmission of a ciphertext character in 8-bit CFB mode, how far does the error propagate?

## APPENDIX 3A BENT FUNCTIONS<sup>11</sup>

In this appendix, we first define bent functions and their application to S-boxes and then discuss the implications.

Consider a function  $f(x)$  that maps the  $n$ -bit integers into a single bit; this is usually expressed as  $f: \{0,1\}^n \rightarrow \{0,1\}$ ; the argument  $x$  can be represented by a string of  $n$  bits  $(x_{n-1} \dots x_1 x_0)$ . For example, for  $n = 3$ , one such function  $g$  can be enumerated as  $\{g(000) = 0; g(001) = 1; g(010) = 1; g(011) = 0; g(100) = 1; g(101) = 0; g(110) = 1; g(111) = 0\}$ . Such a function can be represented as a column vector in which the  $k$ th entry in the column corresponds to  $g(k)$ . For our example,

$$g = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

The Walsh transform of a function  $f: \{0,1\}^n \rightarrow \{0,1\}$  is defined by

$$W_f(w) = \sum_{x=0}^{2^n-1} (-1)^{f(x)+w \cdot x}$$

where

$$w \cdot x = w_{n-1}x_{n-1} \oplus \dots \oplus w_0x_0$$

and where  $w$  is an integer such that  $0 \leq w \leq 2^n - 1$ . Note that each term in the summation takes on the value  $+1$  or  $-1$ . Thus, for a given value of  $w$ ,  $W_f(w)$  takes on an integer value in the range  $-2^n \leq W_f(w) \leq 2^n$ .

<sup>11</sup>This appendix is based on material supplied by Carlisle Adams.

It can be shown that  $f(x)$  can be expressed using the inverse Walsh transform:

$$f(x) = \frac{1}{2^n} \sum_{w=0}^{2^n-1} W_f(w)(-1)^{w \cdot x}$$

Thus,  $f(x)$  can be expressed as a sum of functions of  $w$ .

The set of bent functions, with  $n$  even, is the set of functions  $f: \{0,1\}^n \rightarrow \{0,1\}$  such that

$$W_f(w) = \pm 2^{n/2} \quad \forall w \in \{0,1\}^n$$

The actual definition of a bent function is a little bit abstract: It is a binary vector whose Walsh transform has a flat magnitude.

Let's think about this in pieces. The Walsh transform is analogous to the Fourier transform (for those familiar with basic engineering concepts), but it is used for discrete—in this case, binary—vectors instead of continuous, time-dependent functions. Like the Fourier transform, taking the Walsh transform moves you into another “domain” (not quite time versus frequency, but similar in concept). Taking the absolute value of the transformed vector gives you all positive numbers (of course). If all these numbers are equal, this transformed value is said to be “flat,” or to have a flat magnitude.

What does this mean? Again, think of the analogy to the Fourier transform. The transformed quantity shows you all the frequencies that are involved in creating the original time-domain function that you started with. The same is true here. The transformed vector shows you all the “pieces” that contributed to the original vector that you started with. The difference is that the operation of taking the Walsh transform is simply a multiplication with a particular matrix (the Walsh matrix, oddly enough!). This matrix is actually composed of all the linear binary vectors of the length you are working with, which means that the transform itself is actually a projection of your original vector onto the set of linear vectors. A flat magnitude therefore means that all linear vectors contribute equally to the vector you are examining, which is another way of saying that your vector is not any “closer” to one linear vector than to any other, which is another way of saying that your vector is as nonlinear as it can be.

How does this relate to S-boxes? An  $n \times m$  S-box maps an  $n$ -bit input into an  $m$ -bit output. As was mentioned, an  $n \times m$  S-box consists of  $2^n$  rows of  $m$  bits each. The  $n$  bits of input select one of the rows of the S-box, and the  $m$  bits in that row are the output. We can view the S-box as a set of  $m$  column vectors  $[c_{m-1}(x) \dots c_1(x)c_0(x)]$ , where  $x$  is the  $n$ -bit input and  $c_i(x)$  is the  $i$ th column in the S-box. Thus, each column of the S-box can be viewed as a function  $c_i: \{0,1\}^n \rightarrow \{0,1\}$ . This allows us to consider the construction and use of S-boxes whose columns are bent functions.

Bent functions have a number of interesting properties, but maximum nonlinearity and perfect (i.e., highest-order) Strict Avalanche Criterion are the ones that are of most interest in S-box design. These ideal characteristics led to their use in the CAST design procedure, discussed in Chapter 4.

# CHAPTER 4

## CONVENTIONAL ENCRYPTION: ALGORITHMS

*"It seems very simple."*

*"It is very simple. But if you don't know what the key is it's virtually indecipherable."*

*—Talking to Strange Men, Ruth Rendell*

**T**his chapter examines some of the most important symmetric block ciphers in current use. The ciphers were selected based on a number of criteria:

1. They exhibit considerable cryptographic strength.
2. They are popular in Internet-based applications.
3. They illustrate modern symmetric block cipher techniques that have been developed since the introduction of DES.

The following algorithms are examined: triple DES, IDEA, Blowfish, RC5, CAST, and RC2. The chapter closes with a summary of important characteristics of advanced symmetric block ciphers.

### 4.1 TRIPLE DES

Given the potential vulnerability of DES to a brute-force attack, there has been considerable interest in finding an alternative. One approach is to design a completely new algorithm, of which several examples are given in

this chapter. Another alternative, which would preserve the existing investment in software and equipment, is to use multiple encryption with DES and multiple keys. We begin by examining the simplest example of this second alternative. We then look at the widely accepted triple DES approach.

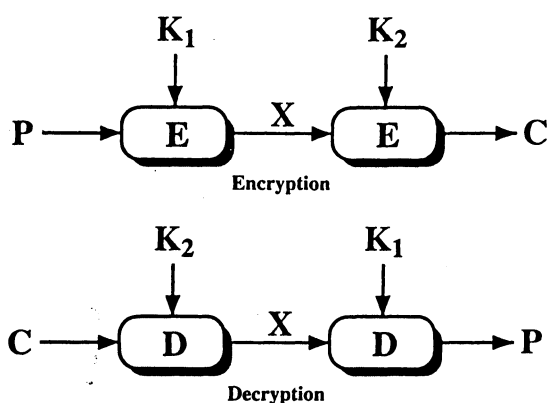
### Double DES

The simplest form of multiple encryption has two encryption stages and two keys (Figure 4.1a). Given a plaintext  $P$  and two encryption keys  $K_1$  and  $K_2$ , ciphertext  $C$  is generated as

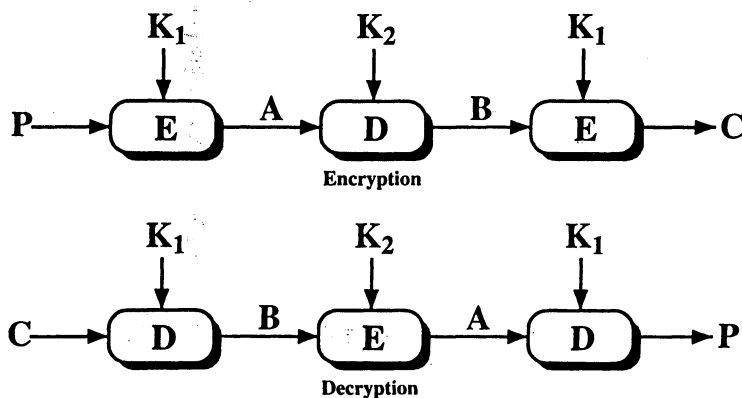
$$C = E_{K_2}[E_{K_1}[P]]$$

Decryption requires that the keys be applied in reverse order:

$$P = D_{K_1}[D_{K_2}[C]]$$



(a) Double Encryption



(b) Triple Encryption

**Figure 4.1** Multiple Encryption.

For DES, this scheme apparently involves a key length of  $56 \times 2 = 112$  bits, resulting in a dramatic increase in cryptographic strength. But we need to examine the algorithm more closely.

### Reduction to a Single Stage

Suppose it were true for DES, for all 56-bit key values, that given any two keys  $K_1$  and  $K_2$ , it would be possible to find a key  $K_3$  such that

$$E_{K_2}[E_{K_1}[P]] = E_{K_3}[P] \quad (4.1)$$

If this were the case, then double encryption, and indeed any number of stages of multiple encryption with DES, would be useless because the result would be equivalent to a single encryption with a single 56-bit key.

On the face of it, it does not appear that Equation (4.1) is likely to hold. Consider that encryption with DES is a mapping of 64-bit blocks to 64-bit blocks. In fact, the mapping can be viewed as a permutation. That is, if we consider all  $2^{64}$  possible input blocks, DES encryption with a specific key will map each block into a unique 64-bit block. Otherwise, if, say, two given input blocks mapped to the same output block, then decryption to recover the original plaintext would be impossible. With  $2^{64}$  possible inputs, how many different mappings are there that generate a permutation of the input blocks? The value is easily seen to be

$$(2^{64})! = 10^{34738000000000000000} > (10^{10^{20}})$$

On the other hand, DES defines one mapping for each different key, for a total number of mappings:

$$2^{56} < 10^{17}$$

Therefore, it is reasonable to assume that if DES is used twice with different keys, it will produce one of the many mappings that is not defined by a single application of DES. Although there was much supporting evidence for this assumption, it was not until 1992 that the assumption was proved [CAMP92].

### Meet-in-the-Middle Attack

Thus, the use of double DES results in a mapping that is not equivalent to a single DES encryption. But there is another way to attack this scheme, one that does not depend on any particular property of DES but that will work against any block encryption cipher.

The algorithm, known as a meet-in-the-middle attack, was first described in [DIFF77]. It is based on the observation that, if we have

$$C = E_{K_2}[E_{K_1}[P]]$$

then (see Figure 4.1a)

$$X = E_{K_1}[P] = D_{K_2}[C]$$

Given a known pair,  $(P, C)$ , the attack proceeds as follows. First, encrypt  $P$  for all  $2^{56}$  possible values of  $K_1$ . Store these results in a table and then sort the table by the values of  $X$ . Next, decrypt  $C$  using all  $2^{56}$  possible values of  $K_2$ . As each decryption is produced, check the result against the table for a match. If a match occurs, then test the two resulting keys against a new known plaintext-ciphertext pair. If the two keys produce the correct ciphertext, accept them as the correct keys.

For any given plaintext  $P$ , there are  $2^{64}$  possible ciphertext values that could be produced by double DES. Double DES uses, in effect, a 112-bit key, so that there are  $2^{112}$  possible keys. Therefore, on average, for a given plaintext  $P$ , the number of different 112-bit keys that will produce a given ciphertext  $C$  is  $2^{112}/2^{64} = 2^{48}$ . Thus, the foregoing procedure will produce about  $2^{48}$  false alarms on the first  $(P, C)$  pair. A similar argument indicates that with an additional 64 bits of known plaintext and ciphertext, the false alarm rate is reduced to  $2^{48-64} = 2^{-16}$ . Put another way, if the meet-in-the-middle attack is performed on two blocks of known plaintext-ciphertext, the probability that the correct keys are determined is  $1 - 2^{-16}$ . The result is that a known plaintext attack will succeed against double DES, which has a key size of 112 bits, with an effort on the order of  $2^{56}$ , not much better than the  $2^{55}$  required for single DES.

### Triple DES with Two Keys

An obvious counter to the meet-in-the-middle attack is to use three stages of encryption with three different keys. This raises the cost of the known-plaintext attack to  $2^{112}$ , which is beyond what is practical now and far into the future. However, it has the drawback of requiring a key length of  $56 \times 3 = 168$  bits, which may be somewhat unwieldy.

As an alternative, Tuchman proposed a triple encryption method that uses only two keys [TUCH79]. The function follows an encrypt-decrypt-encrypt (EDE) sequence (Figure 4.1b):

$$C = E_{K_1}[D_{K_2}[E_{K_1}[P]]]$$

There is no cryptographic significance to the use of decryption for the second stage. Its only advantage is that it allows users of triple DES to decrypt data encrypted by users of the older single DES:

$$C = E_{K_1}[D_{K_1}[E_{K_1}[P]]] = E_{K_1}[P]$$

Triple DES with two keys is a relatively popular alternative to DES and has been adopted for use in the key management standards ANS X9.17 and ISO 8732.<sup>1</sup>

Currently, there are no practical cryptanalytic attacks on triple DES. Copper-smith [COPP94] notes that the cost of a brute-force key search on triple DES is on the order of  $2^{112} \approx (5 \times 10^{33})$  and estimates that the cost of differential cryptanalysis suffers an exponential growth, compared to single DES, exceeding  $10^{52}$ .

<sup>1</sup>(ANS) American National Standard: Financial Institution Key Management (Wholesale). From its title, X9.17 appears to be a somewhat obscure standard. Yet a number of techniques specified in this standard have been adopted for use in other standards and applications, as we shall see throughout this book.

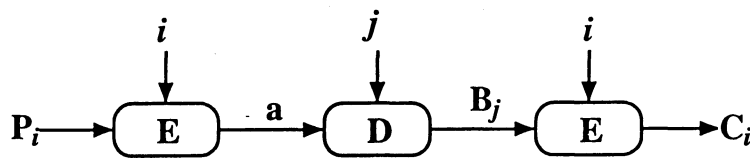


It is worth looking at several proposed attacks on triple DES that, although not practical, give a flavor for the types of attacks that have been considered and that could form the basis for more successful future attacks.

The first serious proposal came from Merkle and Hellman [MERK81]. Their plan involves finding plaintext values that produce a first intermediate value of  $A = 0$  (Figure 4.1b) and then using the meet-in-the-middle attack to determine the two keys. The level of effort is  $2^{56}$ , but the technique requires  $2^{56}$  chosen plaintext-ciphertext pairs, a number unlikely to be provided by the holder of the keys.

A known-plaintext attack is outlined in [OORS90]. This method is an improvement over the chosen-plaintext approach but requires more effort. The attack is based on the observation that if we know  $A$  and  $C$  (Figure 4.1b), then the problem reduces to that of an attack on double DES. Of course, the attacker does not know  $A$ , even if  $P$  and  $C$  are known, as long as the two keys are unknown. However, the attacker can choose a potential value of  $A$  and then try to find a known  $(P, C)$  pair that produces  $A$ . The attack proceeds as follows:

1. Obtain  $n$   $(P, C)$  pairs. This is the known plaintext. Place these in a table (Table 1) sorted on the values of  $P$  (Figure 4.2b).



(a) Two-key triple encryption with candidate pair of keys

$P_i$	$C_i$

(b) Table of  $n$  known plaintext-ciphertext pairs, sorted on  $P$

$B_j$	Key $i$

(c) Table of intermediate values and candidate keys

Figure 4.2 Known-Plaintext Attack on Triple DES.

2. Pick an arbitrary value  $a$  for  $A$ , and create a second table (Figure 4.2c) with entries defined in the following fashion. For each of the  $2^{56}$  possible keys  $K_1 = i$ , calculate the plaintext value  $P_i$  that produces  $a$ :

$$P_i = D_i[a]$$

For each  $P_i$  that matches an entry in Table 1, create an entry in Table 2 consisting of the  $K_1$  value and the value of  $B$  that is produced for the  $(P, C)$  pair from Table 1, assuming that value of  $K_1$ :

$$B = D_i[C]$$

At the end of this step, sort Table 2 on the values of  $B$ .

3. We now have a number of candidate values of  $K_1$  in Table 2 and are in a position to search for a value of  $K_2$ . For each of the  $2^{56}$  possible keys  $K_2 = j$ , calculate the second intermediate value for our chosen value of  $a$ :

$$B_j = D_j[a]$$

At each step, look up  $B_j$  in Table 2. If there is a match, then the corresponding key  $i$  from Table 2 plus this value of  $j$  are candidate values for the unknown keys  $(K_1, K_2)$ . Why? Because we have found a pair of keys  $(i, j)$  that produce a known  $(P, C)$  pair (Figure 4.2a).

4. Test each candidate pair of keys  $(i, j)$  on a few other plaintext-ciphertext pairs. If a pair of keys produces the desired ciphertext, the task is complete. If no pair succeeds, repeat from step 1 with a new value of  $a$ .

For a given known  $(P, C)$ , the probability of selecting the unique value of  $a$  that leads to success is  $1/2^{64}$ . Thus, given  $n$   $(P, C)$  pairs, the probability of success for a single selected value of  $a$  is  $n/2^{64}$ . A basic result from probability theory is that the expected number of draws required to draw one red ball out of a bin containing  $n$  red balls and  $N - n$  green balls is  $(N + 1)/(n + 1)$  if the balls are not replaced. So the expected number of values of  $a$  that must be tried is, for large  $n$ ,

$$\frac{2^{64} + 1}{n + 1} \approx \frac{2^{64}}{n}$$

Thus, the expected running time of the attack is on the order of

$$\left(2^{56}\right) \frac{2^{64}}{n} = 2^{120 - \log_2 n}$$

### Triple DES with Three Keys

Although the attacks just described appear impractical, anyone using two-key triple DES may feel some concern. Thus, many researchers now feel that three-key triple DES is the preferred alternative (e.g., [KALI96]). Three-key triple DES has an effective key length of 168 bits and is defined as follows:

$$C = E_{K_3}[D_{K_2}[E_{K_1}[P]]]$$

Backward compatibility with DES is provided by putting  $K_3 = K_2$  or  $K_1 = K_2$ .

A number of Internet-based applications have adopted three-key triple DES, including PGP and S/MIME, both discussed in Chapter 12.

## 4.2 INTERNATIONAL DATA ENCRYPTION ALGORITHM

The International Data Encryption Algorithm (IDEA) is a symmetric block cipher developed by Xuejia Lai and James Massey of the Swiss Federal Institute of Technology. The original version was published in [LAI90]. A revised version of the algorithm, designed to be stronger against differential cryptanalytic attacks, was presented in [LAI91] and is described in greater detail in [LAI92].

IDEA is one of a number of conventional encryption algorithms that have been proposed in recent years to replace DES, which, as was discussed in Chapter 3, many feel is approaching or has even reached the end of its useful lifetime. In terms of adoption, IDEA is one of the most successful of these proposals. For example, IDEA is included in PGP (discussed in Chapter 12), which alone assures widespread use of the algorithm.

### Design Principles

IDEA is a block cipher that uses a 128-bit key to encrypt data in blocks of 64 bits. By contrast, DES also uses 64-bit blocks but a 56-bit key.

The design goals for IDEA can be grouped into those related to cryptographic strength and those related to ease of implementation.

#### Cryptographic Strength

The following characteristics of IDEA relate to its cryptographic strength:

- **Block length:** The block length should be long enough to deter statistical analysis (that is, to deny the opponent any advantage that some blocks appear more often than others). On the other hand, the complexity of implementing an effective encryption function appears to grow exponentially with block size [WEGE87]. The use of a block size of 64 bits is generally recognized as sufficiently strong. Furthermore, the use of a cipher feedback mode of operation further strengthens this aspect of the algorithm.
- **Key length:** The key length should be long enough to prevent exhaustive key searches. With a length of 128 bits, IDEA seems to be secure in this area far into the future.
- **Confusion:** The ciphertext should depend on the plaintext and key in a complicated and involved way. The objective is to complicate the determination of how the statistics of the ciphertext depend on the statistics of the plaintext. IDEA achieves this goal by using three different operations, as explained later. This is in contrast to DES, which relies principally on the XOR operation and on small nonlinear S-boxes.
- **Diffusion:** Each plaintext bit should influence every ciphertext bit, and each key bit should influence every ciphertext bit. The spreading out of a single plaintext bit over many ciphertext bits hides the statistical structure of the

Let us elaborate on the last two points. In IDEA, **confusion** is achieved by mixing three different operations. Each operation is performed on two 16-bit inputs to produce a single 16-bit output. The operations are

- Bit-by-bit exclusive-OR, denoted as  $\oplus$ .
- Addition of integers modulo  $2^{16}$  (modulo 65536), with inputs and outputs treated as unsigned 16-bit integers. This operation is denoted as  $\boxplus$ .
- Multiplication of integers modulo  $2^{16} + 1$  (modulo 65537), with inputs and outputs treated as unsigned 16-bit integers, except that a block of all zeros is treated as representing  $2^{16}$ . This operation is denoted as  $\odot$ .

For example,

$$0000000000000000 \odot 1000000000000000 = 1000000000000001$$

because

$$2^{16} \times 2^{15} \bmod (2^{16} + 1) = 2^{15} + 1$$

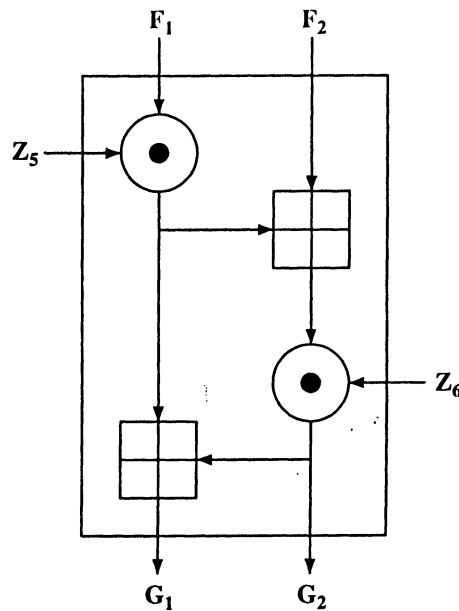
Table 4.1 shows values for the three operations operating on 2-bit numbers (rather than 16-bit numbers). These three operations are incompatible in the sense that

1. No pair of the three operations satisfies a distributive law. For example,

$$a \boxplus (b \odot c) \neq (a \boxplus b) \odot (a \boxplus c)$$

**Table 4.1** Functions Used in IDEA (for operand length of 2 bits)

X	Y	$X \boxplus Y$	$X \odot Y$	$X \oplus Y$
0 00	0 00	0 00	1 01	0 00
0 00	1 01	1 01	0 00	1 01
0 00	2 10	2 10	3 11	2 10
0 00	3 11	3 11	2 10	3 11
1 01	0 00	1 01	0 00	1 01
1 01	1 01	2 10	1 01	0 00
1 01	2 10	3 11	2 10	3 11
1 01	3 11	0 00	3 11	2 10
2 10	0 00	2 10	3 11	2 10
2 10	1 01	3 11	2 10	3 11
2 10	2 10	0 00	0 00	0 00
2 10	3 11	1 01	1 01	1 01
3 11	0 00	3 11	2 10	3 11
3 11	1 01	0 00	3 11	2 10
3 11	2 10	1 01	1 01	1 01
3 11	3 11	2 10	0 00	0 00



**Figure 4.3** Multiplication/Addition (MA) Structure.

2. No pair of the three operations satisfies an associative law. For example,

$$a \oplus (b \oplus c) \neq (a \oplus b) \oplus c$$

The use of these three separate operations in combination provides for a complex transformation of the input, making cryptanalysis much more difficult than with an algorithm such as DES, which relies solely on the XOR function.

In IDEA, **diffusion** is provided by the basic building block of the algorithm, known as the multiplication/addition (MA) structure (Figure 4.3). This structure takes as inputs two 16-bit values derived from the plaintext and two 16-bit subkeys derived from the key and produces two 16-bit outputs. An exhaustive computer check has determined that each output bit of the first round depends on every bit of the plaintext-derived inputs and on every bit of the subkeys [LAI91]. This particular structure is repeated eight times in the algorithm, providing very effective diffusion. Furthermore, it can be shown that this structure uses the least number of operations (four) required to achieve complete diffusion [LAI91].

#### Implementation Considerations

IDEA is designed to facilitate both software and hardware implementation. Hardware implementation, typically in VLSI, is designed to achieve high speed. Software implementation has the advantage of flexibility and low cost. [LAI90] cites the following design principles:

- Design principles for software implementation:
  - Use subblocks: Cipher operations should operate on subblocks that are “natural” for software, such as 8, 16, or 32 bits. IDEA uses 16-bit subblocks.

- Use simple operations: Cipher operations should be easily programmed using addition, shifting, and so on. The three basic elements of IDEA meet this requirement. The most difficult of the three, multiplication modulo  $(2^{16} + 1)$ , can be readily constructed from simple primitive operations (see Problem 4.4).
- Design principles for hardware implementation:
  - Similarity of encryption and decryption: Encryption and decryption should differ only in the way of using the key so that the same device can be used for both encryption and decryption. Like DES, IDEA has a structure that satisfies this requirement.
  - Regular structure: The cipher should have a regular modular structure to facilitate VLSI implementation. IDEA is constructed from two basic modular building blocks repeated multiple times.

### IDEA Encryption

The overall scheme for IDEA encryption is illustrated in Figure 4.4. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext is 64 bits in length and the key is 128 bits in length.

Looking at the left-hand side of the figure, we see that the IDEA algorithm consists of eight rounds followed by a final transformation function. The algorithm divides the input into four 16-bit subblocks. Each of the rounds takes four 16-bit subblocks as input and produces four 16-bit output blocks. The final transformation also produces four 16-bit blocks, which are concatenated to form the 64-bit ciphertext. Each of the rounds also makes use of six 16-bit subkeys, whereas the final transformation uses four subkeys, for a total of 52 subkeys. The right-hand portion of Figure 4.4 indicates that these 52 subkeys are all generated from the original 128-bit key.

#### Details of a Single Round

Now let us look more closely at the algorithm for a single round, as illustrated in Figure 4.5. In fact, Figure 4.5 shows the first round. Subsequent rounds have the same structure but with different subkey and plaintext-derived inputs. We can see that IDEA deviates from the classic Feistel structure. The round begins with a transformation that combines the four input subblocks with four subkeys, using the addition and multiplication operations. This transformation is highlighted as the upper, shaded rectangle. The four output blocks of this transformation are then combined using the XOR operation to form two 16-bit blocks that are input to the MA structure (see Figure 4.3), which is shown as the lower, shaded rectangle. The MA structure also takes two subkeys as input and combines these inputs to produce two 16-bit outputs.

Finally, the four output blocks from the upper transformation are combined with the two output blocks of the MA structure using XOR to produce the four output blocks for this round. Note that the two outputs that are partially generated by the second and third inputs ( $X_2$  and  $X_3$ ) are interchanged to produce the second and third outputs ( $W_{12}$  and  $W_{13}$ ). This increases the mixing of the bits being processed and makes the algorithm more resistant to differential cryptanalysis.

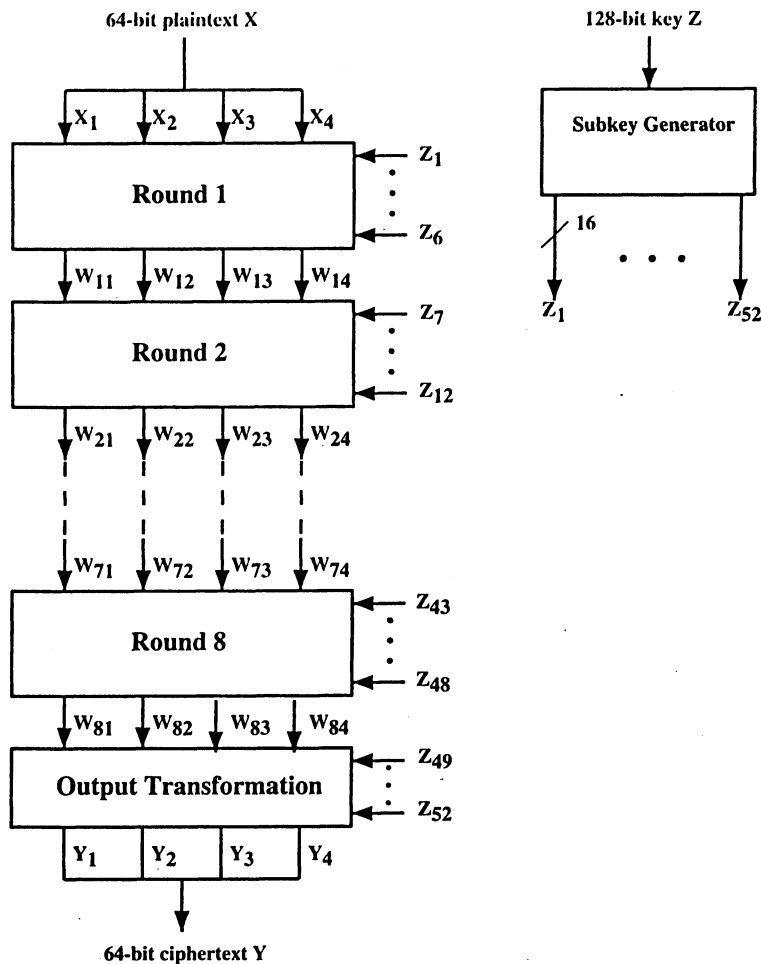


Figure 4.4 Overall IDEA Structure.

The ninth stage of the algorithm, labeled the output transformation stage in Figure 4.4, is shown in Figure 4.6. Note that it has the same structure as the upper, shaded portion of the preceding rounds (Figure 4.5). The only difference is that the second and third inputs are interchanged before being applied to the operational units. In fact, this has the effect of undoing the interchange at the end of the eighth round. The reason for this extra interchange is so that decryption has the same structure as encryption, as will be seen. Note also that this ninth stage requires only four subkey inputs, compared to six subkey inputs for each of the first eight stages.

#### Subkey Generation

Returning to Figure 4.4, we see that 52 16-bit subkeys are generated from the 128-bit encryption key. The scheme for generation is as follows. The first eight subkeys, labeled  $Z_1, Z_2, \dots, Z_8$ , are taken directly from the key, with  $Z_1$  being equal to the first (most significant) 16 bits,  $Z_2$  corresponding to the next 16 bits, and so on.

P104

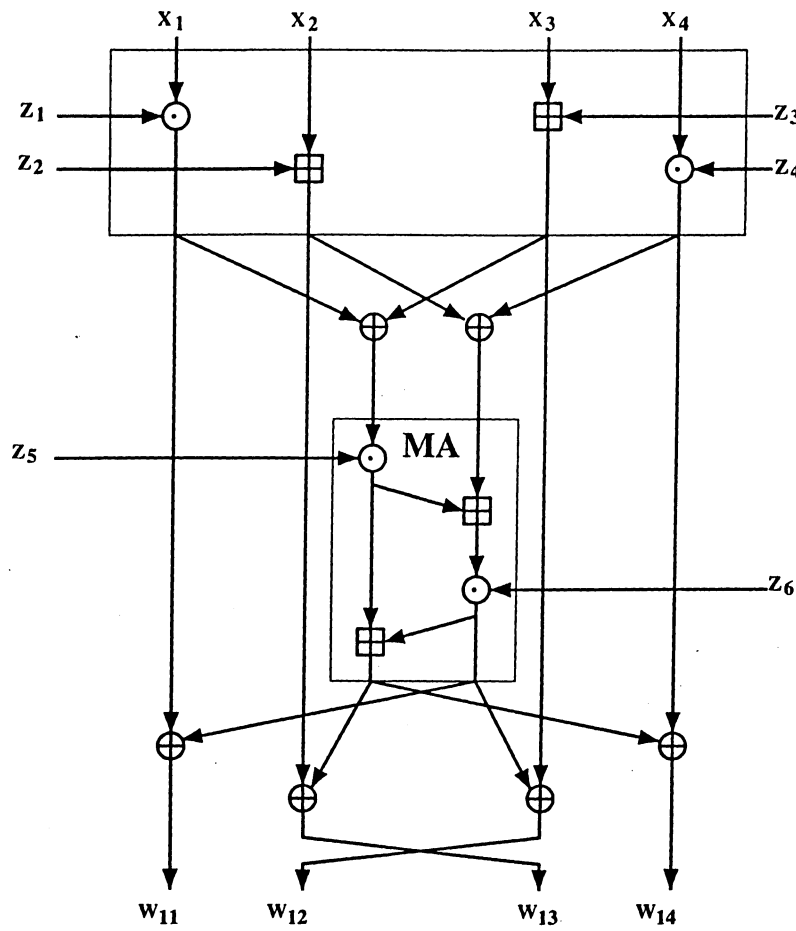
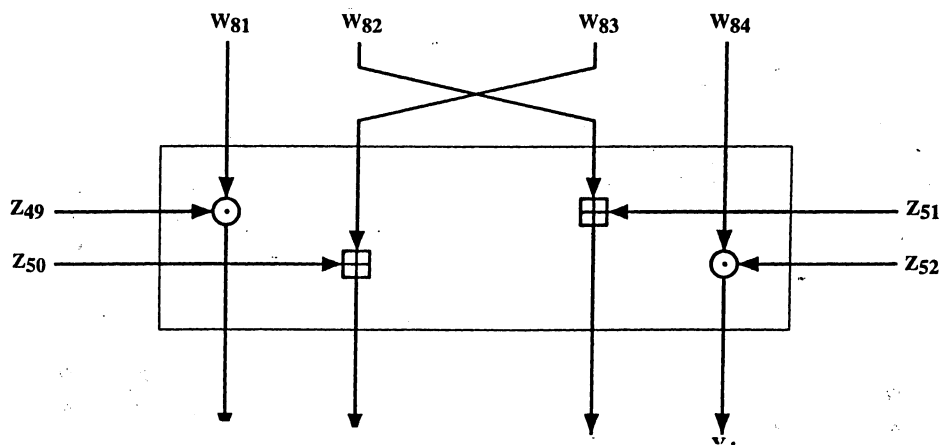


Figure 4.5 Single Round of IDEA (first round).





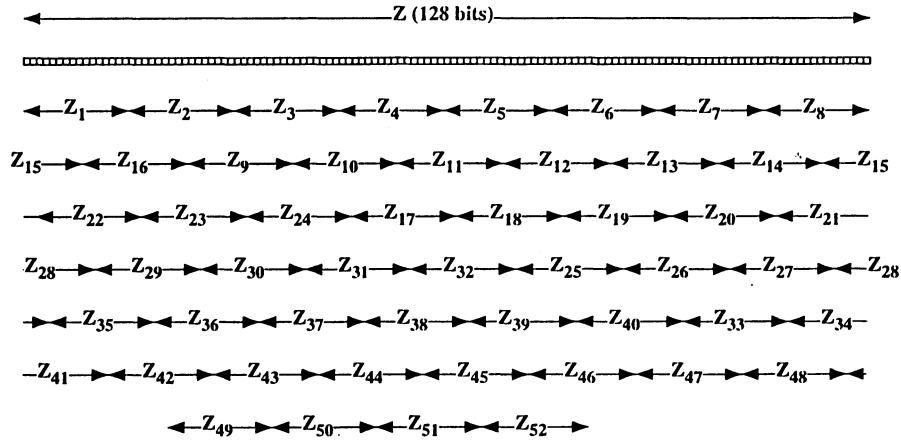


Figure 4.7 IDEA Subkeys.

Then a circular left shift of 25 bit positions is applied to the key, and the next eight subkeys are extracted. This procedure is repeated until all 52 subkeys are generated. Figure 4.7 indicates the bit assignments for all subkeys with respect to the original key.

This scheme provides an effective technique for varying the key bits used for subkeys in the eight rounds. Note that the first-subkey used in each round uses a different set of bits from the key. If the key as a whole is labeled  $Z[1..128]$ , then the first key of the eight rounds has the following bit assignments:

$$\begin{array}{ll}
 Z_1 = Z[1..16] & Z_{25} = Z[76..91] \\
 Z_7 = Z[94..112] & Z_{31} = Z[44..59] \\
 Z_{13} = Z[90..105] & Z_{37} = Z[34..52] \\
 Z_{19} = Z[83..98] & Z_{43} = Z[30..45]
 \end{array}$$

The 96 subkey bits used for a round are, with the exception of the first and eighth rounds, not contiguous, so that there is not even a simple shift relationship between the subkeys of one round and those of another. The reason for this phenomenon is that only six subkeys are used in each round, whereas eight subkeys are extracted with each rotation of the key.

### IDEA Decryption

The process of decryption is essentially the same as the encryption process. Decryption is achieved by using the ciphertext as input to the same overall IDEA structure, as shown in Figure 4.4, but with a different selection of subkeys. The decryption subkeys  $U_1, \dots, U_{52}$  are derived from the encryption subkeys as follows:

1. The first four subkeys of decryption round  $i$  are derived from the first four subkeys of encryption round  $(10 - i)$ , where the transformation stage is counted as round 9. The first and fourth decryption subkeys are equal to the multiplicative inverse modulo  $(2^{16} + 1)$  of the corresponding first and fourth encryption subkeys. For rounds 2 through 8, the second and third decryption subkeys

are equal to the additive inverse modulo ( $2^{16}$ ) of the corresponding third and second encryption subkeys. For rounds 1 and 9, the second and third decryption subkeys are equal to the additive inverse modulo ( $2^{16}$ ) of the corresponding second and third encryption subkeys.

2. For the first eight rounds, the last two subkeys of decryption round  $i$  are equal to the last two subkeys of encryption round  $(9 - i)$ .

Table 4.2 summarizes these relationships. For the multiplicative inverse, the notation  $Z_j^{-1}$  is used, so that we have

$$Z_j \odot Z_j^{-1} = 1$$

Because  $2^{16} + 1$  is a prime number, each nonzero integer  $Z_j \leq 2^{16}$  has a unique multiplicative inverse modulo ( $2^{16} + 1$ ); see Chapter 7. For the additive inverse modulo  $2^{16}$ , the notation  $-Z_j$  is used, so that we have

$$-Z_j \boxplus Z_j = 0$$

To verify that the same algorithm with the decryption subkeys produces the correct result, consider Figure 4.8, which shows the encryption process going down the left-hand side and the decryption process going up the right-hand side. Each of the eight rounds is further shown broken up into the two substages of transformation and what is referred to as subencryption; the transformation substage corresponds to the upper, shaded rectangle in Figure 4.5, and the subencryption stage refers to the remainder of the processing of that round.

Consider the bottom box in both diagrams. On the encryption side, the following relationships hold for the output transformation:

$$\begin{aligned} Y_1 &= W_{81} \odot Z_{49} & Y_3 &= W_{82} \boxplus Z_{51} \\ Y_2 &= W_{83} \boxplus Z_{50} & Y_4 &= W_{84} \odot Z_{52} \end{aligned}$$

The first substage of the first round of the decryption process yields the following relationships:

$$\begin{aligned} J_{11} &= Y_1 \odot U_1 & J_{13} &= Y_3 \boxplus U_3 \\ J_{12} &= Y_2 \boxplus U_2 & J_{14} &= Y_4 \odot U_4 \end{aligned}$$

Substituting,

$$\begin{aligned} J_{11} &= Y_1 \odot Z_{49}^{-1} = W_{81} \odot Z_{49} \odot Z_{49}^{-1} = W_{81} \\ J_{12} &= Y_2 \boxplus -Z_{50} = W_{83} \boxplus Z_{50} \boxplus -Z_{50} = W_{83} \\ J_{13} &= Y_3 \boxplus -Z_{51} = W_{82} \boxplus Z_{51} \boxplus -Z_{51} = W_{82} \\ J_{14} &= Y_4 \odot Z_{52}^{-1} = W_{84} \odot Z_{52} \odot Z_{52}^{-1} = W_{84} \end{aligned}$$

**Table 4.2** Encryption and Decryption Subkeys

Stage	Encryption		Decryption	
	Designation	Equivalent to	Designation	Equivalent to
Round 1	$Z_1 Z_2 Z_3 Z_4 Z_5 Z_6$	$Z[1..96]$	$U_1 U_2 U_3 U_4 U_5 U_6$	$Z_{49}^{-1} - Z_{50} - Z_{51} Z_{52}^{-1} Z_{47} Z_{48}$
Round 2	$Z_7 Z_8 Z_9 Z_{10} Z_{11} Z_{12}$	$Z[97..128; 26..89]$	$U_7 U_8 U_9 U_{10} U_{11} U_{12}$	$Z_{43}^{-1} - Z_{45} - Z_{44} Z_{46}^{-1} Z_{41} Z_{42}$
Round 3	$Z_{13} Z_{14} Z_{15} Z_{16} Z_{17} Z_{18}$	$Z[90..128; 1..25; 51..82]$	$U_{13} U_{14} U_{15} U_{16} U_{17} U_{18}$	$Z_{37}^{-1} - Z_{39} - Z_{38} Z_{40}^{-1} Z_{35} Z_{36}$
Round 4	$Z_{19} Z_{20} Z_{21} Z_{22} Z_{23} Z_{24}$	$Z[83..128; 1..50]$	$U_{19} U_{20} U_{21} U_{22} U_{23} U_{24}$	$Z_{31}^{-1} - Z_{33} - Z_{32} Z_{34}^{-1} Z_{29} Z_{30}$
Round 5	$Z_{25} Z_{26} Z_{27} Z_{28} Z_{29} Z_{30}$	$Z[76..128; 1..43]$	$U_{25} U_{26} U_{27} U_{28} U_{29} U_{30}$	$Z_{25}^{-1} - Z_{27} - Z_{26} Z_{28}^{-1} Z_{23} Z_{24}$
Round 6	$Z_{31} Z_{32} Z_{33} Z_{34} Z_{35} Z_{36}$	$Z[44..75; 101..128; 1..36]$	$U_{31} U_{32} U_{33} U_{34} U_{35} U_{36}$	$Z_{19}^{-1} - Z_{21} - Z_{20} Z_{22}^{-1} Z_{17} Z_{18}$
Round 7	$Z_{37} Z_{38} Z_{39} Z_{40} Z_{41} Z_{42}$	$Z[37..100; 126..128; 1..29]$	$U_{37} U_{38} U_{39} U_{40} U_{41} U_{42}$	$Z_{13}^{-1} - Z_{15} - Z_{14} Z_{16}^{-1} Z_{11} Z_{12}$
Round 8	$Z_{43} Z_{44} Z_{45} Z_{46} Z_{47} Z_{48}$	$Z[30..125]$	$U_{43} U_{44} U_{45} U_{46} U_{47} U_{48}$	$Z_7^{-1} - Z_9 - Z_8 Z_{10}^{-1} Z_5 Z_6$
transformation	$Z_{49} Z_{50} Z_{51} Z_{52}$	$Z[23..86]$	$U_{49} U_{50} U_{51} U_{52}$	$Z_1^{-1} - Z_2 - Z_3 Z_4^{-1}$

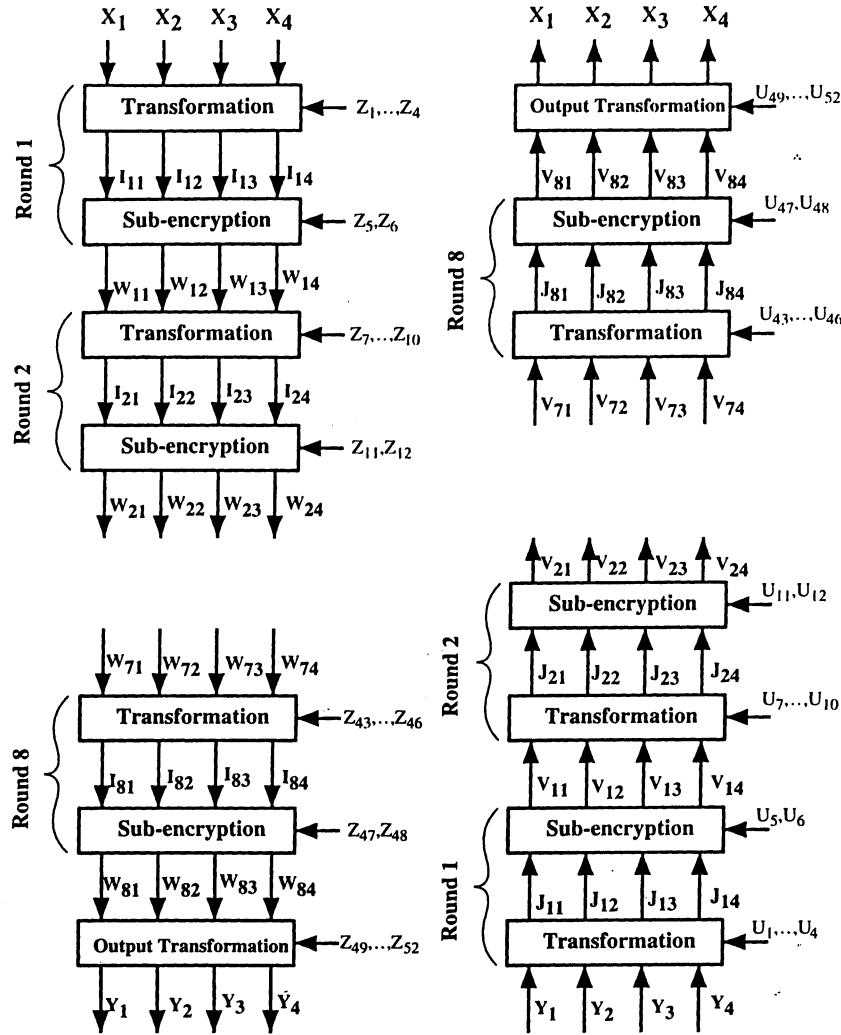


Figure 4.8 IDEA Encryption and Decryption.

Thus, the output of the first substage of the decryption process is equal to the input to the last stage of the encryption process except for an interchange of the second and third blocks. Now consider the following relationships, which can be derived from Figure 4.5:

$$W_{81} = I_{81} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})$$

$$W_{82} = I_{83} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})$$

$$W_{83} = I_{82} \oplus MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})$$

$$W_{84} = I_{84} \oplus MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})$$

where  $MA_R(X, Y)$  is the right-hand output of the MA structure (Figure 4.3) with inputs  $X$  and  $Y$ , and  $MA_L(X, Y)$  is the left-hand output of the MA structure with inputs  $X$  and  $Y$ . Now

$$\begin{aligned}
 V_{11} &= J_{11} \oplus MA_R(J_{11} \oplus J_{13}, J_{12} \oplus J_{14}) \\
 &= W_{81} \oplus MA_R(W_{81} \oplus W_{82}, W_{83} \oplus W_{84}) \\
 &= I_{81} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \oplus \\
 &\quad MA_R[I_{81} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \oplus I_{83} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}), \\
 &\quad I_{82} \oplus MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \oplus I_{84} \oplus MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})] \\
 &= I_{81} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84}) \\
 &= I_{81}
 \end{aligned}$$

Similarly, we have

$$\begin{aligned}
 V_{12} &= I_{83} \\
 V_{13} &= I_{82} \\
 V_{14} &= I_{84}
 \end{aligned}$$

So the output of the second substage of the decryption process is equal to the input to the next-to-last substage of the encryption process except for an interchange of the second and third blocks. Using the same derivation, this relationship can be shown to hold at each corresponding point in Figure 4.8, until we have

$$\begin{aligned}
 V_{81} &= I_{11} \\
 V_{82} &= I_{13} \\
 V_{83} &= I_{12} \\
 V_{84} &= I_{14}
 \end{aligned}$$

Finally, because the output transformation of the decryption process is equal to the first substage of the encryption process except for an interchange of the second and third blocks, we have the output of the entire encryption process equal to the input to the encryption process.

### 4.3 BLOWFISH

Blowfish is a symmetric block cipher developed by Bruce Schneier [SCHN93, SCHN94]. Blowfish was designed to have the following characteristics:

- **Fast:** Blowfish encrypts data on 32-bit microprocessors at a rate of 18 clock cycles per byte.
- **Compact:** Blowfish can run in less than 5K of memory.

- **Simple:** Blowfish's simple structure is easy to implement and eases the task of determining the strength of the algorithm.
- **Variably secure:** The key length is variable and can be as long as 448 bits. This allows a tradeoff between higher speed and higher security.

Blowfish encrypts 64-bit blocks of plaintext into 64-bit blocks of ciphertext. Blowfish is implemented in numerous products and has received a fair amount of scrutiny. So far, the security of Blowfish is unchallenged.

### Subkey and S-Box Generation

Blowfish makes use of a key that ranges from 32 bits to 448 bits (1 to 14 32-bit words). That key is used to generate 18 32-bit subkeys and four  $8 \times 32$  S-boxes containing a total of 1024 32-bit entries. The total is 1042 32-bit values, or 4168 bytes.

The keys are stored in a K-array:

$$K_1, K_2, \dots, K_j \quad 1 \leq j \leq 14$$

The subkeys are stored in the P-array:

$$P_1, P_2, \dots, P_{18}$$

There are four S-boxes, each with 256 32-bit entries:

$$S_{1,0}, S_{1,1}, \dots, S_{1,255}$$

$$S_{2,0}, S_{2,1}, \dots, S_{2,255}$$

$$S_{3,0}, S_{3,1}, \dots, S_{3,255}$$

$$S_{4,0}, S_{4,1}, \dots, S_{4,255}$$

The steps in generating the P-array and S-boxes are as follows:

1. Initialize first the P-array and then the four S-boxes in order using the bits of the fractional part of the constant  $\pi$ . Thus, the leftmost 32 bits of the fractional part of  $\pi$  become  $P_1$ , and so on. For example, in hexadecimal,

$$P_1 = 243F6A88$$

$$P_2 = 85A308D3$$

...

$$S_{4,254} = 578FD FE3$$

$$S_{4,255} = 3AC372E6$$

2. Perform a bitwise XOR of the P-array and the K-array, reusing words from the K-array as needed. For example, for the maximum length key (14 32-bit words),  $P_1 = P_1 \oplus K_1$ ,  $P_2 = P_2 \oplus K_2$ , ...,  $P_{14} = P_{14} \oplus K_{14}$ ,  $P_{15} = P_{15} \oplus K_1$ , ...,  $P_{18} = P_{18} \oplus K_4$ .

3. Encrypt the 64-bit block of all zeros using the current P- and S-arrays; replace  $P_1$  and  $P_2$  with the output of the encryption.
4. Encrypt the output of step 3 using the current P- and S-arrays and replace  $P_3$  and  $P_4$  with the resulting ciphertext.
5. Continue this process to update all elements of P and then, in order, all elements of S, using at each step the output of the continuously changing Blowfish algorithm.

The update process can be summarized as follows:

$$\begin{aligned}
 P_1, P_2 &= E_{P,S}[0] \\
 P_3, P_4 &= E_{P,S}[P_1 \parallel P_2] \\
 &\dots \\
 P_{17}, P_{18} &= E_{P,S}[P_{15} \parallel P_{16}] \\
 S_{1,0}, S_{1,1} &= E_{P,S}[P_{17} \parallel P_{18}] \\
 &\dots \\
 S_{4,254}, S_{4,255} &= E_{P,S}[S_{4,252} \parallel S_{4,253}]
 \end{aligned}$$

Where  $E_{P,S}[Y]$  is the ciphertext produced by encrypting Y using Blowfish with the arrays S and P.

A total of 521 executions of the Blowfish encryption algorithm are required to produce the final S- and P-arrays. Accordingly, Blowfish is not suitable for applications in which the secret key changes frequently. Further, for rapid execution, the P- and S-arrays can be stored rather than rederived from the key each time the algorithm is used. This requires over 4 kilobytes of memory. Thus, Blowfish is not appropriate for applications with limited memory, such as smart cards.

## Encryption and Decryption

Blowfish uses two primitive operations:

- **Addition:** Addition of words, denoted by +, is performed modulo  $2^{32}$ .
- **Bitwise exclusive-OR:** This operation is denoted by  $\oplus$ .

The important thing about these two operations is that they do not commute. This makes cryptanalysis more difficult.

Figure 4.9a depicts the encryption operation. The plaintext is divided into two 32-bit halves  $LE_0$  and  $RE_0$ . We use the variables  $LE_i$  and  $RE_i$  to refer to the left and right half of the data after round  $i$  has completed. The algorithm can be defined by the following pseudocode:

```

for  $i = 1$  to 16 do
     $RE_i = LE_{i-1} \oplus P_i;$ 
     $LE_i = F[RE_i] \oplus RE_{i-1};$ 
 $LE_{17} = RE_{16} \oplus P_{18};$ 
 $RE_{17} = LE_{16} \oplus P_{17};$ 

```

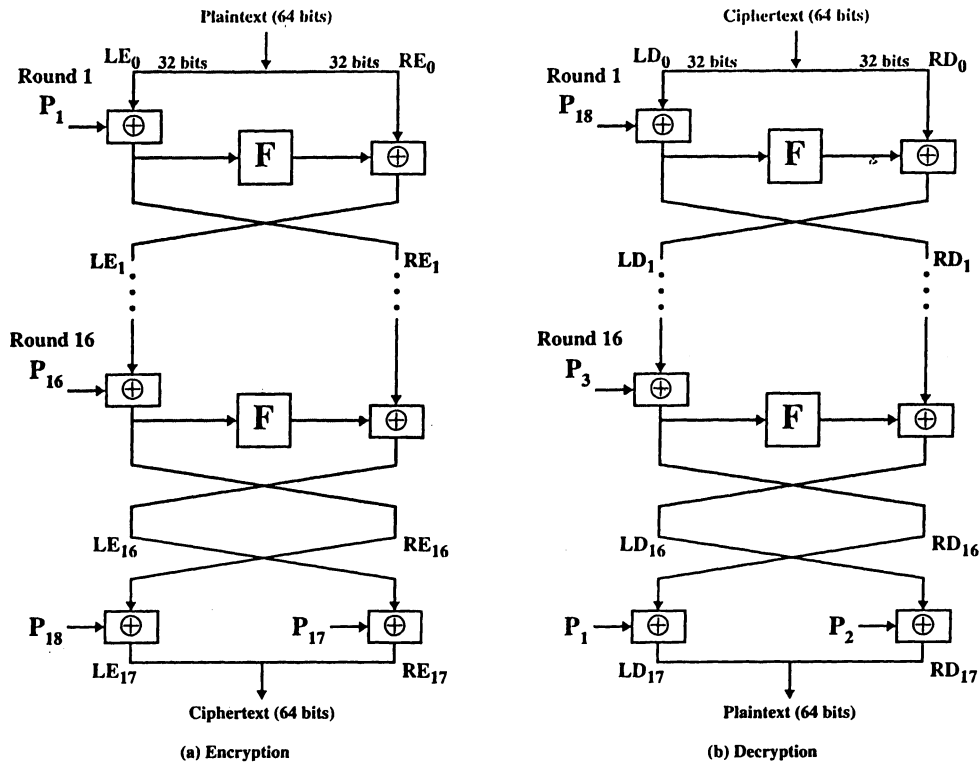


Figure 4.9 Blowfish Encryption and Decryption.

The resulting ciphertext is contained in the two variables  $LE_{17}$  and  $RE_{17}$ . The function  $F$  is shown in Figure 4.10. The 32-bit input to  $F$  is divided into 4 bytes. If we label those bytes  $a$ ,  $b$ ,  $c$ , and  $d$ , then the function can be defined as follows:

$$F[a, b, c, d] = ((S_{1,a} + S_{2,b}) \oplus S_{3,c}) + S_{4,d}$$

Thus, each round includes the complex use of addition modulo  $2^{32}$  and XOR, plus substitution using S-boxes.

Decryption, shown in Figure 4.9b, is easily derived from the encryption algorithm. In this case, the 64 bits of ciphertext are initially assigned to the two one-word variables  $LD_0$  and  $RD_0$ . We use the variables  $LD_i$  and  $RD_i$  to refer to the left and right half of the data after round  $i$ . As with most block ciphers, Blowfish decryption involves using the subkeys in reverse order. However, unlike most block ciphers, Blowfish decryption occurs in the same algorithmic direction as encryption, rather than the reverse. The algorithm can be defined as follows:

```

for  $i = 1$  to 16 do
     $RD_i = LD_{i-1} \oplus P_{19-i};$ 
     $LD_i = F[RD_i] \oplus RD_{i-1};$ 
 $LD_{17} = RD_{16} \oplus P_1;$ 
 $RD_{17} = LD_{16} \oplus P_2;$ 

```



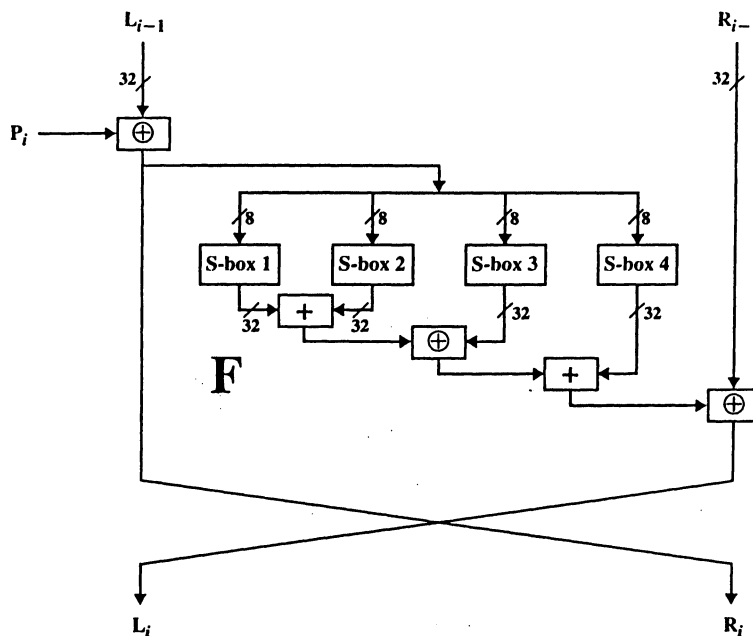


Figure 4.10 Detail of Single Blowfish Round.

### Discussion

Blowfish is perhaps the most formidable conventional encryption algorithm described in this book. Unlike DES, the S-boxes in Blowfish are key dependent. But more can be said. Some of the other algorithms, such as RC5, are designed so that one of the functions performed during a round is data dependent (in the case of RC5, the circular rotation). But in the case of Blowfish, both the subkeys and S-boxes are produced by a process of repeated applications of Blowfish itself. This thoroughly mangles the bits and makes cryptanalysis very difficult. So far, there have been a few published papers on Blowfish cryptanalysis, but no practical weaknesses have been found.

Another interesting aspect of the Blowfish design is that operations are performed on both halves of the data in each round, compared to performing an operation on just half the data in each round in the classic Feistel cipher. This should provide greater cryptographic strength, even though the additional operation is linear (XOR). In support of this belief, [HEYS95] notes that the inclusion of a linear transformation in each round of a substitution-permutation network (SPN) improves the avalanche characteristics of a block cipher.<sup>2</sup>

With regard to brute-force attacks, Blowfish is virtually invulnerable with suitable choice of key length, which can be as long as 448 bits.

Blowfish is also impressively fast to execute. Table 4.3, developed by Schneier, compares the number of clock cycles required on a Pentium for various algorithms implemented in C. Blowfish is clearly the fastest to execute.

<sup>2</sup>The authors were not looking at Blowfish but were analyzing SPNs in general.

**Table 4.3** Speed Comparisons of Block Ciphers on a Pentium

Algorithm	Clock cycles per round	# of rounds	# of clock cycles per byte encrypted
Blowfish	9	16	18
RC5	12	16	23
DES	18	16	45
IDEA	50	8	50
Triple-DES	18	48	108

In [SCHN93], Schneier provides an interesting account of the design decisions that went into the details of Blowfish. A few of the highlights of that discussion are as follows:

1. A brute-force attack is even more difficult than may be apparent from the key length because of the time-consuming subkey-generation process. A total of 522 executions of the encryption algorithm are required to test a single key.
2. The function  $F$  gives Blowfish the best possible avalanche affect for a Feistel network: In round  $i$ , every bit of  $L_{i-1}$  affects every bit of  $R_{i-1}$ . In addition, every subkey bit is affected by every key bit, and therefore  $F$  has a perfect avalanche effect between the key ( $P_i$ ) and the right half of the data ( $R_i$ ) after every round.
3. Every bit of the input to  $F$  is only used as input to one S-box. In contrast, in DES, many bits are used as inputs to two S-boxes, which strengthens the algorithm considerably against differential attacks. Schneier felt that this added complexity was not necessary with key-dependent S-boxes.
4. Unlike in CAST, the function  $F$  in Blowfish is not round dependent. Schneier felt that such dependency did not add any cryptographic merit, given that the P-array substitution is already round dependent.

## 4.4 RC5

RC5 is a symmetric encryption algorithm developed by Ron Rivest [RIVE94, RIVE95]. RC5 was designed to have the following characteristics:

- **Suitable for hardware or software:** RC5 uses only primitive computational operations commonly found on microprocessors.
- **Fast:** To achieve this, RC5 is a simple algorithm and is word oriented. The basic operations work on full words of data at a time.
- **Adaptable to processors of different word lengths:** The number of bits in a word is a parameter of RC5; different word lengths yield different algorithms.
- **Variable number of rounds:** The number of rounds is a second parameter of RC5. This parameter allows a tradeoff between higher speed and higher security.

- **Variable-length key:** The key length is a third parameter of RC5. Again, this allows a tradeoff between speed and security.
- **Simple:** RC5's simple structure is easy to implement and eases the task of determining the strength of the algorithm.
- **Low memory requirement:** A low memory requirement makes RC5 suitable for smart cards and other devices with restricted memory.
- **High security:** RC5 is intended to provide high security with suitable parameters.
- **Data-dependent rotations:** RC5 incorporates rotations (circular bit shifts) whose amount is data dependent. This appears to strengthen the algorithm against cryptanalysis.

RC5 has been incorporated into RSA Data Security, Inc.'s major products, including BSAFE, JSAFE, and S/MAIL.

### RC5 Parameters

RC5 is actually a family of encryption algorithms determined by three parameters, as follows:

Parameter	Definition	Allowable Values
$w$	Word size in bits. RC5 encrypts 2-word blocks	16, 32, 64
$r$	Number of rounds	0, 1, ..., 255
$b$	Number of 8-bit bytes (octets) in the secret key K	0, 1, ..., 255

Thus, RC5 encrypts blocks of plaintext of length 32, 64, or 128 bits into blocks of ciphertext of the same length. The key length ranges from 0 to 2040 bits. A specific version of RC5 is designated as RC5- $w/r/b$ . For example, RC5-32/12/16 has 32-bit words (64-bit plaintext and ciphertext blocks), 12 rounds in the encryption and decryption algorithms, and a key length of 16 bytes (128 bits). Rivest suggests the use of RC5-32/12/16 as the "nominal" version.

### Key Expansion

RC5 performs a complex set of operations on the secret key to produce a total of  $t$  subkeys. Two subkeys are used in each round, and two subkeys are used on an additional operation that is not part of any round, so  $t = 2r + 2$ . Each subkey is one word ( $w$  bits) in length.

Figure 4.11 illustrates the technique used to generate subkeys. The subkeys are stored in a  $t$ -word array labeled  $S[0], S[1], \dots, S[t - 1]$ . Using the parameters  $r$  and  $w$  as inputs, this array is initialized to a particular fixed pseudorandom bit pattern. Then the  $b$ -byte key,  $K[0 \dots b - 1]$ , is converted into a  $c$ -word array  $L[0 \dots c - 1]$ . On a little-endian machine, this is accomplished by zeroing out the array  $L$  and copying the string  $K$  directly into the memory positions represented by  $L$ . If  $b$  is not an integer multiple of  $w$ , then a portion of  $L$  at the right end remains zero. Finally, a mixing operation is performed that applies the contents of  $L$  to the initialized value of  $S$  to produce a final value for the array  $S$ .

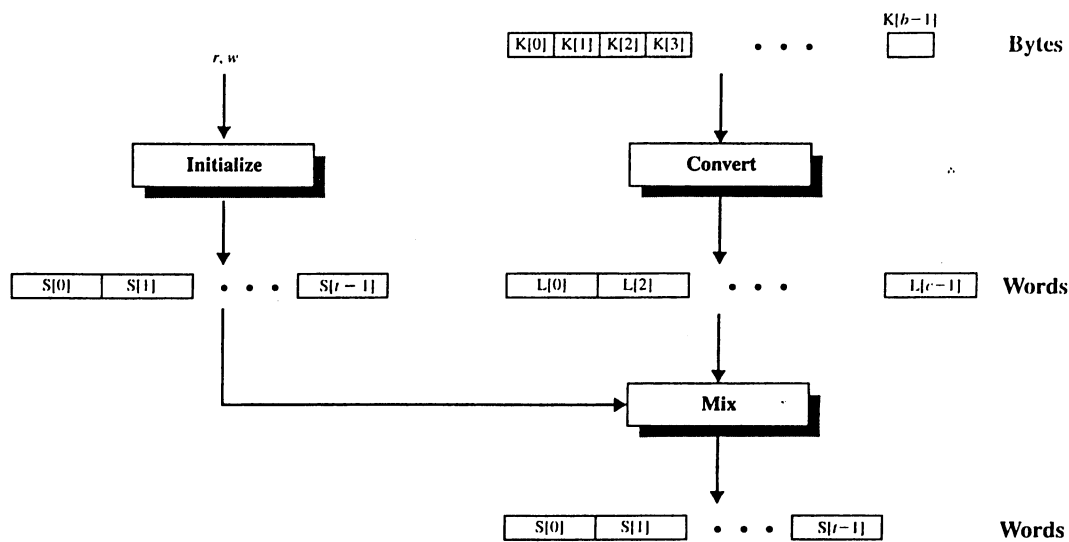


Figure 4.11 RC5 Key Expansion.

Let us look at these operations in more detail.

The initialize operation makes use of two word-length constants defined as follows:

$$P_w = \text{Odd}[(e-2)2^w]$$

$$Q_w = \text{Odd}[(\phi-1)2^w]$$

where

$$e = 2.718281828459 \dots \text{ (base of natural logarithms)}$$

$$\phi = 1.618033988749 \dots \text{ (golden ratio)}^3 = \left( \frac{1 + \sqrt{5}}{2} \right)$$

and  $\text{Odd}[x]$  is the odd integer nearest to  $x$  (rounded up to  $x$  if  $x$  is an even integer, although this will not happen here). For example,  $\text{Odd}[e] = 3$  and  $\text{Odd}[\phi] = 1$ . Using the allowable values of  $w$ , the constants are (in hexadecimal)

$w$	16	32	64
$P_w$	B7E1	B7E15163	B7E151628AED2A6B
$Q_w$	9E37	9E3779B9	9E3779B97F4A7C15

<sup>3</sup>This is one of the more ubiquitous numbers in mathematics. With the possible exception of  $\pi$ , no other number plays a more important role in our physical world. The ancient Greeks ascribed to it mystical characteristics and called it the divine proportion.

Using these two constants, the array  $S$  is initialized in the following manner:

```

 $S[0] = P_w;$ 
for  $i = 1$  to  $t - 1$  do
     $S[i] = S[i - 1] + Q_w;$ 

```

where addition is performed modulo  $2^w$ . The initialized array  $S$  is then mixed with the key array  $L$  to produce a final array  $S$  of subkeys. For this purpose, three passes are made through the larger of the two arrays; the smaller array may be handled more times:

```

 $i = j = X = Y = 0$ 
do  $3 \times \max(t, c)$  times:
     $S[i] = (S[i] + X + Y) \lll 3; X = S[i]; i = (i + 1) \bmod (t);$ 
     $L[j] = (L[j] + X + Y) \lll (X + Y); Y = L[j]; j = (j + 1) \bmod (c);$ 

```

Rivest [RIVE94] comments that the key expansion function has a certain amount of *one-wayness*: It is not so easy to determine  $K$  from  $S$ .

## Encryption

RC5 uses three primitive operations (and their inverses):

- **Addition:** Addition of words, denoted by  $+$ , is performed modulo  $2^w$ . The inverse operation, denoted by  $-$ , is subtraction modulo  $2^w$ .
- **Bitwise exclusive-OR:** This operation is denoted by  $\oplus$ .
- **Left circular rotation:** The cyclic rotation of word  $x$  left by  $y$  bits is denoted by  $x \lll y$ . The inverse is the right circular rotation of word  $x$  by  $y$  bits, denoted by  $x \ggg y$ .

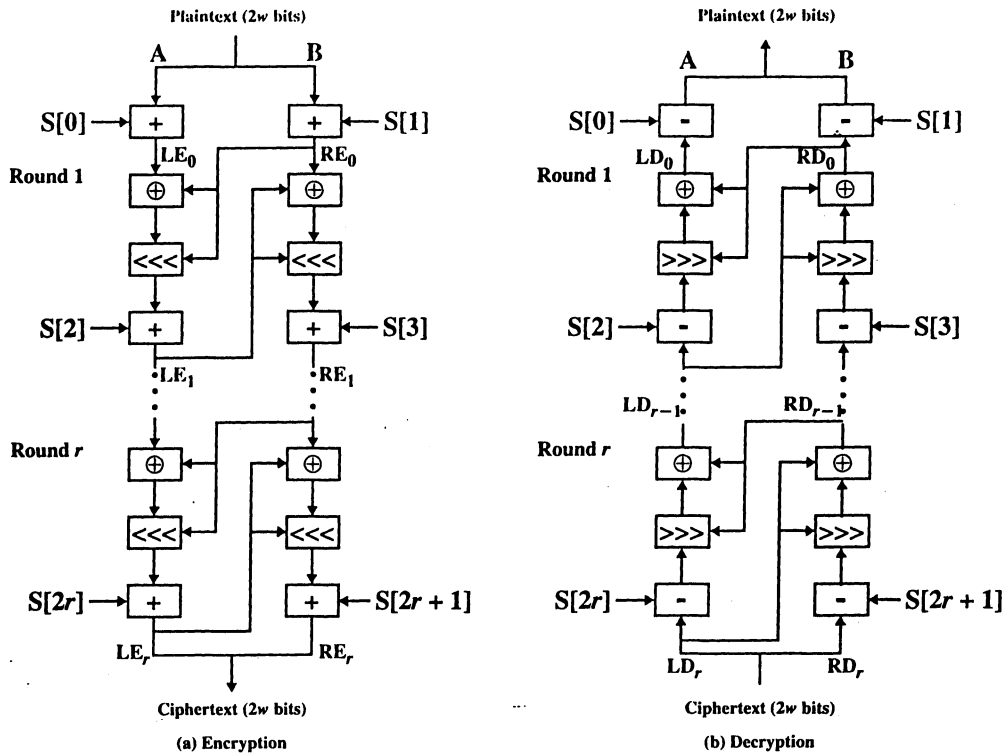
Figure 4.12a depicts the encryption operation. Note that this is not a classic Feistel structure. The plaintext is assumed to initially reside in the two  $w$ -bit registers  $A$  and  $B$ . We use the variables  $LE_i$  and  $RE_i$  to refer to the left and right half of the data after round  $i$  has completed. The algorithm can be defined by the following pseudocode:

```

 $LE_0 = A + S[0];$ 
 $RE_0 = B + S[1];$ 
for  $i = 1$  to  $r$  do
     $LE_i = ((LE_{i-1} \oplus RE_{i-1}) \lll RE_{i-1}) + S[2 \times i];$ 
     $RE_i = ((RE_{i-1} \oplus LE_i) \lll LE_i) + S[2 \times i + 1];$ 

```

The resulting ciphertext is contained in the two variables  $LE_r$  and  $RE_r$ . Each of the  $r$  rounds consists of a substitution using both words of data, a permutation using both words of data, and a substitution that depends on the key. Note the exceptional simplicity of the operation, which can be defined in five lines of code. Also note that both halves of the data are updated in each round. Thus, one round of RC5 is somewhat equivalent to two rounds of DES.



**Figure 4.12** RC5 Encryption and Decryption.

### Decryption

Decryption, shown in Figure 4.12b, is easily derived from the encryption algorithm. In this case, the  $2w$  bits of ciphertext are initially assigned to the two one-word variables  $LD_r$  and  $RD_r$ . We use the variables  $LD_i$  and  $RD_i$  to refer to the left and right half of the data before round  $i$  has begun, where the rounds are numbered from  $r$  down to 1.

```

for  $i = r$  down to 1 do
     $RD_{i-1} = ((RD_i - S[2 \times i + 1] \ggg LD_i) \oplus LD_i);$ 
     $LD_{i-1} = ((LD_i - S[2 \times i] \ggg RD_{i-1}) \oplus RD_{i-1});$ 
     $B = RD_0 - S[1];$ 
     $A = LD_0 - S[0];$ 
  
```

The two most striking features of RC5 are the simplicity of the algorithm and the use of data-dependent rotations. The rotations are the only nonlinear portion of the algorithm. Rivest feels that because the amount of rotation varies depending on the value of the data moving through the algorithm, linear and differential cryptanalysis should be more difficult. A number of studies have confirmed this supposition [YIN97].

## RC5 Modes

To enhance the effectiveness of RC5 in interoperable implementations, RFC 2040 [BALD96] defines four different modes of operation:

- **RC5 block cipher:** This is the raw encryption algorithm that takes a fixed-size input block ( $2w$  bits) and produces a ciphertext block of the same length using a transformation that depends on a key. This is often known as the electronic codebook (ECB) mode (see Figure 3.11).
- **RC5-CBC:** This is the cipher block chaining mode for RC5. CBC was discussed in Chapter 3 (see Figure 3.12). CBC processes messages whose length is a multiple of the RC5 block size (multiples of  $2w$  bits). As was discussed in Chapter 3, CBC provides enhanced security compared to ECB because repeated blocks of plaintext produce different blocks of ciphertext.
- **RC5-CBC-Pad:** This is a CBC style of algorithm that handles plaintext of any length. The ciphertext will be longer than the plaintext by at most the size of a single RC5 block.
- **RC5-CTS:** This is the ciphertext stealing mode, which is also a CBC style of algorithm. This mode handles plaintext of any length and produces ciphertext of equal length.

The last two modes in the preceding list warrant elaboration.

When a CBC mode is used to encrypt a message, some technique is needed to cope with messages that are not a multiple of the block length. The simplest approach is to use padding. In RC5, it is assumed that the message is an integer number of bytes. At the end of the message, from 1 to  $bb$  bytes of padding are added, where  $bb$  equals the block size for RC5 measured in bytes ( $bb = 2w/8$ ). The pad bytes are all the same and are set to a byte that represents the number of bytes of padding. For example, if there are 8 bytes of padding, each byte has the bit pattern 00001000.

Padding may not always be appropriate. For example, one might wish to store the encrypted data in the same memory buffer that originally contained the plaintext. In that case, the ciphertext must be the same length as the original plaintext. The RC5-CTS mode provides this capability (Figure 4.13). Assume that the last block of plaintext is only  $L$  bytes long, where  $L < 2w/8$ . The encryption sequence is as follows:<sup>4</sup>

1. Encrypt the first  $(N - 2)$  blocks using the traditional CBC technique.
2. Exclusive-OR  $P_{N-1}$  with the previous ciphertext block  $C_{N-2}$  to create  $Y_{N-1}$ .
3. Encrypt  $Y_{N-1}$  to create  $E_{N-1}$ .
4. Select the first  $L$  bytes of  $E_{N-1}$  to create  $C_N$ .
5. Pad  $P_N$  with zeros at the end and exclusive-OR with  $E_{N-1}$  to create  $Y_N$ .
6. Encrypt  $Y_N$  to create  $C_{N-1}$ .

The last two blocks of the ciphertext are  $C_{N-1}$  and  $C_N$ .

---

<sup>4</sup>The description in RFC 2040 has an error; the description here is correct.

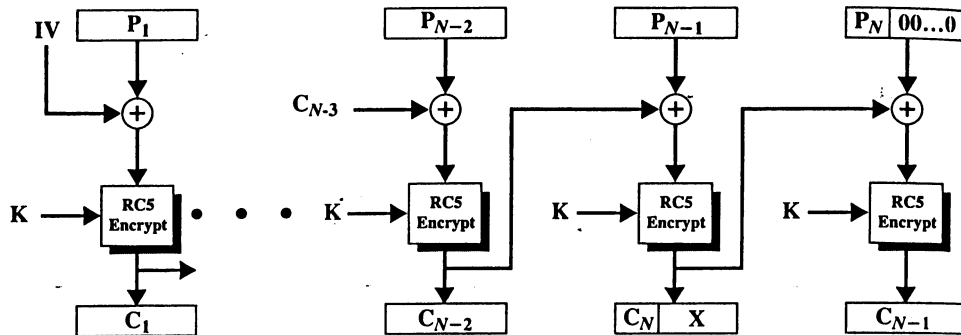


Figure 4.13 RC5 Ciphertext Stealing Mode.

## 4.5 CAST-128

CAST is a design procedure for symmetric encryption algorithms developed by Carlisle Adams and Stafford Tavares [ADAM97a]. In this section, we focus on the algorithm known as CAST-128 and defined in RFC 2144 [ADAM97b]. CAST-128 makes use of a key size that varies from 40 bits to 128 bits in 8-bit increments.

CAST has the structure of a classical Feistel network (see Figure 3.5) with 16 rounds and operating on 64-bit blocks of plaintext to produce 64-bit blocks of ciphertext. The two differences from Figure 3.5 are that (1) CAST employs two subkeys in each round: a 32-bit  $K_m$  and a 5-bit  $K_r$ , and (2) the function  $F$  depends on the round.

CAST is the result of a long process of research and development and has benefited from extensive review by cryptologists. It is beginning to be used in a number of products, including PGP.

We examine the details of the encryption algorithm first and then look at the subkey-generation process.

### CAST-128 Encryption

CAST-128 uses four primitive operations:

- **Addition and subtraction:** Addition of words, denoted by  $+$ , is performed modulo  $2^{32}$ . The inverse operation, denoted by  $-$ , is subtraction modulo  $2^{32}$ .
- **Bitwise exclusive-OR:** This operation is denoted by  $\oplus$ .
- **Left circular rotation:** The cyclic rotation of word  $x$  left by  $y$  bits is denoted by  $x \lll y$ .

The CAST-128 encryption algorithm can be defined by the following pseudocode. The plaintext is divided into two 32-bit halves  $L_0$  and  $R_0$ . We use the variables  $L_i$  and  $R_i$  to refer to the left and right half of the data after round  $i$  has completed. The ciphertext is formed by swapping the output of the sixteenth round; that is, the ciphertext is the concatenation of  $R_{16}$  and  $L_{16}$ .



$L_0 \parallel R_0 = \text{Plaintext}$   
**for**  $i = 1$  **to** 16 **do**  
      $L_i = R_{i-1};$   
      $R_i = L_{i-1} \oplus F_i[R_{i-1}, K_{m_i}, K_{r_i}];$   
**Ciphertext** =  $R_{16} \parallel L_{16}$

Decryption is the same as encryption, with the keys employed in reverse order.

Figure 4.14 depicts the details of a single round. The F function includes the use of four  $8 \times 32$  S-boxes, the left circular rotation function, and four functions that vary depending on the round number; we label these functions  $f1_i$ ,  $f2_i$ ,  $f3_i$ , and  $f4_i$ . We use  $I$  to refer to the intermediate 32-bit value after the left circular rotation function, and the labels  $Ia$ ,  $Ib$ ,  $Ic$ , and  $Id$  to refer to the 4 bytes of  $I$ , where  $Ia$  is the most significant and  $Id$  is the least significant. With these conventions,  $F$  is defined as follows:

Rounds 1, 4, 7, 10, 13, 16	$I = ((K_{m_i} + R_{i-1}) \lll K_{r_i})$ $F = ((S1[Ia] \oplus S2[Ib]) - S3[Ic]) + S4[Id]$
Rounds 2, 5, 8, 11, 14	$I = ((K_{m_i} \oplus R_{i-1}) \lll K_{r_i})$ $F = ((S1[Ia] - S2[Ib]) + S3[Ic]) \oplus S4[Id]$
Rounds 3, 6, 9, 12, 15	$I = ((K_{m_i} - R_{i-1}) \lll K_{r_i})$ $F = ((S1[Ia] + S2[Ib]) \oplus S3[Ic]) - S4[Id]$

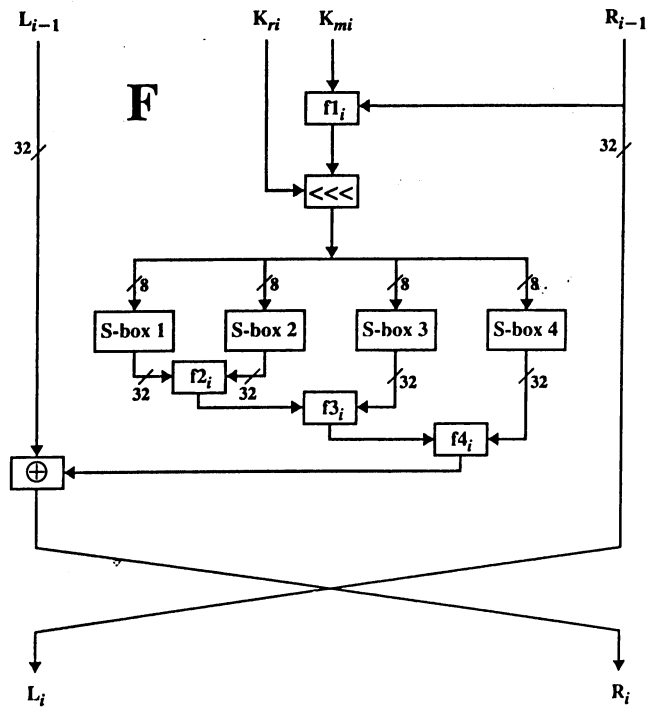


Figure 4.14 Detail of Single CAST-128 Round.

Note the similarity of the structure of the CAST-128 round function to that of Blowfish.

### Substitution Boxes

CAST-128 uses eight  $8 \times 32$  S-boxes. Four of these, S-box 1 through S-box 4, are used in the encryption and decryption process. The remaining four, S-box 5 through S-box 8, are used in subkey generation. Each S-box is an array of 32 columns by 256 rows. The 8-bit input selects a row in the array; the 32-bit value in that row is the output.

All of the S-boxes contain fixed values. The designers make use of the theory of bent functions, referred to briefly in Chapter 3 and described in Appendix 3A, to design boxes that have a high degree of nonlinearity.

### Subkey Generation

Subkey generation is a complex process. To begin, label the bytes of the 128-bit key as follows:

$x_0x_1x_2x_3x_4x_5x_6x_7x_8x_9x_Ax_Bx_Cx_Dx_Ex_F$

where  $x_0$  represents the most significant byte and  $x_F$  the least significant byte. Also use the following definitions:

$K_{m_1}, \dots, K_{m_{16}}$	Sixteen 32-bit masking subkeys (one per round)
$K_{r_1}, \dots, K_{r_{16}}$	Sixteen 32-bit rotate subkeys (one per round), of which only the least significant 5 bits of each are used
$z_0 \dots z_F$	Intermediate (temporary) bytes
$K_1, \dots, K_{32}$	Intermediate (temporary) 32-bit words

Figure 4.15 defines how the values  $K_1$  through  $K_{32}$  are calculated from the key using S-boxes 5 through 8. Then the subkeys are defined as

**for**  $i = 1$  **to** 16 **do**  
 $K_{mi} = K_i;$   
 $K_{ri} = K_{16+i};$

### Discussion

There are several notable features of CAST worthy of comment. First, as was mentioned, CAST makes use of fixed S-boxes. The designers felt that fixed S-boxes with good nonlinearity characteristics are preferable to random S-boxes as might be obtained if the S-boxes were key dependent. The procedure used for CAST is described in [ADAM97a] and, in more detail, in [MIST96]. The details are beyond the scope of this book. In essence, the procedure is as follows. Select 32 distinct binary bent vectors of length 256. Each column vector represents the value of one output bit for any given input. The vectors are chosen such that their sum (modulo 2) is highly nonlinear and so that they have good avalanche properties. This process involves choosing one column at a time, performing a test on the set of columns chosen so far, and then proceeding to make another choice or moving on to the next column.

```

z0z1z2z3 = x0x1x2x3 ⊕ S5[xD] ⊕ S6[xF] ⊕ S7[xC] ⊕ S8[xE] ⊕ S7[x8]
z4z5z6z7 = x8x9xAxB ⊕ S5[z0] ⊕ S6[z2] ⊕ S7[z1] ⊕ S8[z3] ⊕ S8[xA]
z8z9zAzB = xCxDxExF ⊕ S5[z7] ⊕ S6[z6] ⊕ S7[z5] ⊕ S8[z4] ⊕ S5[x9]
zCzDzEzF = x4x5x6x7 ⊕ S5[zA] ⊕ S6[z9] ⊕ S7[zB] ⊕ S8[z8] ⊕ S6[xB]

K1 = S5[z8] ⊕ S6[z9] ⊕ S7[z7] ⊕ S8[z6] ⊕ S5[z2]
K2 = S5[zA] ⊕ S6[zB] ⊕ S7[z5] ⊕ S8[z4] ⊕ S6[z6]
K3 = S5[zC] ⊕ S6[zD] ⊕ S7[z3] ⊕ S8[z2] ⊕ S7[z9]
K4 = S5[zE] ⊕ S6[zF] ⊕ S7[z1] ⊕ S8[z0] ⊕ S8[zC]

x0x1x2x3 = z8z9zAzB ⊕ S5[z5] ⊕ S6[z7] ⊕ S7[z4] ⊕ S8[z6] ⊕ S7[z0]
x4x5x6x7 = z0z1z2z3 ⊕ S5[x0] ⊕ S6[x2] ⊕ S7[x1] ⊕ S8[x3] ⊕ S8[z2]
x8x9xAxB = z4z5z6z7 ⊕ S5[x7] ⊕ S6[x6] ⊕ S7[x5] ⊕ S8[x4] ⊕ S5[z1]
xCxDxExF = zCzDzEzF ⊕ S5[xA] ⊕ S6[x9] ⊕ S7[xB] ⊕ S8[x8] ⊕ S6[z3]

K5 = S5[x3] ⊕ S6[x2] ⊕ S7[xC] ⊕ S8[xD] ⊕ S5[x8]
K6 = S5[x1] ⊕ S6[x0] ⊕ S7[xE] ⊕ S8[xF] ⊕ S6[xD]
K7 = S5[x7] ⊕ S6[x6] ⊕ S7[x8] ⊕ S8[x9] ⊕ S7[x3]
K8 = S5[x5] ⊕ S6[x4] ⊕ S7[xA] ⊕ S8[xB] ⊕ S8[x7]

z0z1z2z3 = x0x1x2x3 ⊕ S5[xD] ⊕ S6[xF] ⊕ S7[xC] ⊕ S8[xE] ⊕ S7[x8]
z4z5z6z7 = x8x9xAxB ⊕ S5[z0] ⊕ S6[z2] ⊕ S7[z1] ⊕ S8[z3] ⊕ S8[xA]
z8z9zAzB = xCxDxExF ⊕ S5[z7] ⊕ S6[z6] ⊕ S7[z5] ⊕ S8[z4] ⊕ S5[x9]
zCzDzEzF = x4x5x6x7 ⊕ S5[zA] ⊕ S6[z9] ⊕ S7[zB] ⊕ S8[z8] ⊕ S6[xB]

K9 = S5[z3] ⊕ S6[z2] ⊕ S7[zC] ⊕ S8[zD] ⊕ S5[z9]
K10 = S5[z1] ⊕ S6[z0] ⊕ S7[zE] ⊕ S8[zF] ⊕ S6[zC]
K11 = S5[z7] ⊕ S6[z6] ⊕ S7[z8] ⊕ S8[z9] ⊕ S7[z2]
K12 = S5[z5] ⊕ S6[z4] ⊕ S7[zA] ⊕ S8[zB] ⊕ S8[z6]

x0x1x2x3 = z8z9zAzB ⊕ S5[z5] ⊕ S6[z7] ⊕ S7[z4] ⊕ S8[z6] ⊕ S7[z0]
x4x5x6x7 = z0z1z2z3 ⊕ S5[x0] ⊕ S6[x2] ⊕ S7[x1] ⊕ S8[x3] ⊕ S8[z2]
x8x9xAxB = z4z5z6z7 ⊕ S5[x7] ⊕ S6[x6] ⊕ S7[x5] ⊕ S8[x4] ⊕ S5[z1]
xCxDxExF = zCzDzEzF ⊕ S5[xA] ⊕ S6[x9] ⊕ S7[xB] ⊕ S8[x8] ⊕ S6[z3]

K13 = S5[x8] ⊕ S6[x9] ⊕ S7[x7] ⊕ S8[x6] ⊕ S5[x3]
K14 = S5[xA] ⊕ S6[xB] ⊕ S7[x5] ⊕ S8[x4] ⊕ S6[x7]
K15 = S5[xC] ⊕ S6[xD] ⊕ S7[x3] ⊕ S8[x2] ⊕ S7[x8]
K16 = S5[xE] ⊕ S6[xF] ⊕ S7[x1] ⊕ S8[x0] ⊕ S8[xD]

z0z1z2z3 = x0x1x2x3 ⊕ S5[xD] ⊕ S6[xF] ⊕ S7[xC] ⊕ S8[xE] ⊕ S7[x8]
z4z5z6z7 = x8x9xAxB ⊕ S5[z0] ⊕ S6[z2] ⊕ S7[z1] ⊕ S8[z3] ⊕ S8[xA]
z8z9zAzB = xCxDxExF ⊕ S5[z7] ⊕ S6[z6] ⊕ S7[z5] ⊕ S8[z4] ⊕ S5[x9]
zCzDzEzF = x4x5x6x7 ⊕ S5[zA] ⊕ S6[z9] ⊕ S7[zB] ⊕ S8[z8] ⊕ S6[xB]

K17 = S5[z8] ⊕ S6[z9] ⊕ S7[z7] ⊕ S8[z6] ⊕ S5[z2]
K18 = S5[zA] ⊕ S6[zB] ⊕ S7[z5] ⊕ S8[z4] ⊕ S6[z6]
K19 = S5[zC] ⊕ S6[zD] ⊕ S7[z3] ⊕ S8[z2] ⊕ S7[z9]
K20 = S5[zE] ⊕ S6[zF] ⊕ S7[z1] ⊕ S8[z0] ⊕ S8[zC]

x0x1x2x3 = z8z9zAzB ⊕ S5[z5] ⊕ S6[z7] ⊕ S7[z4] ⊕ S8[z6] ⊕ S7[z0]
x4x5x6x7 = z0z1z2z3 ⊕ S5[x0] ⊕ S6[x2] ⊕ S7[x1] ⊕ S8[x3] ⊕ S8[z2]
x8x9xAxB = z4z5z6z7 ⊕ S5[x7] ⊕ S6[x6] ⊕ S7[x5] ⊕ S8[x4] ⊕ S5[z1]
xCxDxExF = zCzDzEzF ⊕ S5[xA] ⊕ S6[x9] ⊕ S7[xB] ⊕ S8[x8] ⊕ S6[z3]

K21 = S5[x3] ⊕ S6[x2] ⊕ S7[xC] ⊕ S8[xD] ⊕ S5[x8]
K22 = S5[x1] ⊕ S6[x0] ⊕ S7[xE] ⊕ S8[xF] ⊕ S6[xD]
K23 = S5[x7] ⊕ S6[x6] ⊕ S7[x8] ⊕ S8[x9] ⊕ S7[x3]
K24 = S5[x5] ⊕ S6[x4] ⊕ S7[xA] ⊕ S8[xB] ⊕ S8[x7]

z0z1z2z3 = x0x1x2x3 ⊕ S5[xD] ⊕ S6[xF] ⊕ S7[xC] ⊕ S8[xE] ⊕ S7[x8]
z4z5z6z7 = x8x9xAxB ⊕ S5[z0] ⊕ S6[z2] ⊕ S7[z1] ⊕ S8[z3] ⊕ S8[xA]
z8z9zAzB = xCxDxExF ⊕ S5[z7] ⊕ S6[z6] ⊕ S7[z5] ⊕ S8[z4] ⊕ S5[x9]
zCzDzEzF = x4x5x6x7 ⊕ S5[zA] ⊕ S6[z9] ⊕ S7[zB] ⊕ S8[z8] ⊕ S6[xB]

K25 = S5[z3] ⊕ S6[z2] ⊕ S7[zC] ⊕ S8[zD] ⊕ S5[z9]
K26 = S5[z1] ⊕ S6[z0] ⊕ S7[zE] ⊕ S8[zF] ⊕ S6[zC]
K27 = S5[z7] ⊕ S6[z6] ⊕ S7[z8] ⊕ S8[z9] ⊕ S7[z2]
K28 = S5[z5] ⊕ S6[z4] ⊕ S7[zA] ⊕ S8[zB] ⊕ S8[z6]

x0x1x2x3 = z8z9zAzB ⊕ S5[z5] ⊕ S6[z7] ⊕ S7[z4] ⊕ S8[z6] ⊕ S7[z0]
x4x5x6x7 = z0z1z2z3 ⊕ S5[x0] ⊕ S6[x2] ⊕ S7[x1] ⊕ S8[x3] ⊕ S8[z2]
x8x9xAxB = z4z5z6z7 ⊕ S5[x7] ⊕ S6[x6] ⊕ S7[x5] ⊕ S8[x4] ⊕ S5[z1]
xCxDxExF = zCzDzEzF ⊕ S5[xA] ⊕ S6[x9] ⊕ S7[xB] ⊕ S8[x8] ⊕ S6[z3]

K29 = S5[x8] ⊕ S6[x9] ⊕ S7[x7] ⊕ S8[x6] ⊕ S5[x3]
K30 = S5[xA] ⊕ S6[xB] ⊕ S7[x5] ⊕ S8[x4] ⊕ S6[x7]
K31 = S5[xC] ⊕ S6[xD] ⊕ S7[x3] ⊕ S8[x2] ⊕ S7[x8]
K32 = S5[xE] ⊕ S6[xF] ⊕ S7[x1] ⊕ S8[x0] ⊕ S8[xD]

```

Figure 4.15 CAST-128 Subkey Generation.

The subkey-generation process used in CAST-128 is different from that employed in other symmetric encryption algorithms described in the literature. The CAST designers were concerned to make subkeys as resistant to known cryptanalytic attacks as possible and felt that the use of highly nonlinear S-boxes provided

this strength. We have seen other approaches with the same goal. For example, Blowfish uses the encryption algorithm itself to generate the subkeys. RC5 uses a pseudorandom initialization sequence followed by a complex set of operations involving variable-length rotations and mod 2 additions. It is difficult to say which of these approaches is superior. However, any of these approaches would appear to offer greater cryptographic strength than the simple substitution-permutation schedule used in DES.

The function  $F$  is designed to have good confusion, diffusion, and avalanche properties. It uses S-box substitutions, mod 2 addition and subtraction, exclusive-OR operations, and key-dependent rotation. The strength of the  $F$  function is based primarily on the strength of the S-boxes, but the further use of these arithmetic, Boolean, and rotate operators adds to its strength. Finally,  $F$  is not uniform from round to round, as was described. This dependence of  $F$  on round number may provide further strength, although this has not been demonstrated.

## 4.6 RC2

RC2 is a symmetric encryption algorithm developed by Ron Rivest [RIVE97]. RC2 uses plaintext and ciphertext blocks of 64 bits and a key size that varies from 8 to 1024 bits. The algorithm is designed to be easy to implement on 16-bit microprocessors, reflecting its age. RC2 is used in S/MIME with 40-, 64-, and 128-bit key sizes.

### Key Expansion

RC2 performs a set of operations on the secret key to produce 128 bytes of subkey. The subkey is stored in a byte array labeled  $L[0]$ ,  $L[1]$ , ...,  $L[127]$ . For some operations, it is convenient to refer to the same subkey material as a 16-bit word array  $K[0]$ ,  $K[1]$ , ...,  $K[63]$ .

Let there be  $T$  bytes of key provided, with  $1 \leq T \leq 128$ . There is an optional key-weakening process, intended for export control use and ignored in the following discussion. Key generation begins by placing the  $T$  bytes of key into bytes  $L[0]$ , ...,  $L[T]$ . The  $L$  array is then computed making use of an auxiliary array of pseudorandom bytes,  $P[0]$ ,  $P[1]$ , ...,  $P[255]$ , which is based on the digits of  $\pi$ . The computation is described as follows:

```

for  $i = T$  to 127 do                      /* set  $L[T]$  through  $L[127]$  */
     $L[i] = P[L[i - 1] + L[i - T]]$ ;
 $L[128 - T] = P[L[128 - T]]$ 
for  $i = 127 - T$  down to 0 do             /* set  $L[0]$  through  $L[127 - T]$  */
     $L[i] = P[L[i + 1] \oplus L[i + T]]$ ;

```

In general, the first round sets each expanded subkey byte after the first  $T$  bytes to the sum of the previous subkey byte and the subkey byte  $T$  positions back. The second round sets each subkey byte except the last  $T$  bytes to the XOR of the next subkey byte and the subkey byte  $T$  positions ahead.

## Encryption

RC2 uses the following primitive operations:

- **Addition:** Addition of words, denoted by  $+$ , is performed modulo  $2^{32}$ . The inverse operation, denoted by  $-$ , is subtraction modulo  $2^n$ .
- **Bitwise exclusive-or:** This operation is denoted by  $\oplus$ .
- **Bitwise complement:** This operation is denoted by  $\sim$ .
- **Bitwise AND:** This operation is denoted by  $\&$ .
- **Left circular rotation:** The cyclic rotation of word  $x$  left by  $y$  bits is denoted by  $x \lll y$ .

Unlike the other symmetric block ciphers described in this book, RC2 does not use the classical Feistel structure. This makes it difficult to compare to other algorithms.

The encryption algorithm takes a 64-bit input stored in the 16-bit words  $R[0]$ ,  $R[1]$ ,  $R[2]$ ,  $R[3]$  and places the results back in  $R[0]$  through  $R[3]$ . The algorithm consists of a total of 18 rounds of two types: mixing and mashing. A **mixing round** is expressed as

```

R[0] = R[0] + K[j] + (R[3] & R[2]) + ((~R[3] & R[1]));
R[0] = R[0] <<< 1;
j = j + 1;
R[1] = R[1] + K[j] + (R[0] & R[3]) + ((~R[0] & R[2]));
R[1] = R[1] <<< 2;
j = j + 1;
R[2] = R[2] + K[j] + (R[1] & R[0]) + ((~R[1] & R[3]));
R[2] = R[2] <<< 3;
j = j + 2;
R[3] = R[3] + K[j] + (R[2] & R[1]) + ((~R[2] & R[0]));
R[3] = R[3] <<< 5;
j = j + 3;

```

In this formulation,  $K[j]$  is the first subkey word that has not yet been used. We can describe this operation in words as follows. For each word  $R[i]$ , these operations are performed: The next subkey word  $K[j]$  is added to  $R[i]$ . Then  $R[i - 1]$ , where the index into  $R$  is modulo 3, is used to create a “composite” word that is added to  $R[i]$ . The composite word consists of the bits of  $R[i - 2]$  in those bit positions where  $R[i - 1]$  is 1 and of the bits of  $R[i - 3]$  in those bit positions where  $R[i - 1]$  is 0. Then  $R[i]$  undergoes a circular left shift and  $j$  is incremented.

For a **mashing round**, we have

```

R[0] = R[0] + K[R[3] & 63];
R[1] = R[1] + K[R[0] & 63];
R[2] = R[2] + K[R[1] & 63];
R[3] = R[3] + K[R[2] & 63];

```

In words, for each  $R[i]$ , a subkey word is added to  $R[i]$ . The subkey array is indexed by the low-order 6 bits of  $R[i - 1]$ .

RC2 can now be defined as follows. The value of  $j$  after each step is indicated in parentheses.

1. Initialize  $j$  to zero.
2. Perform five mixing rounds ( $j = 20$ ).
3. Perform one mashing round.
4. Perform six mixing rounds ( $j = 44$ ).
5. Perform one mashing round.
6. Perform five mixing rounds ( $j = 64$ ).

Each mixing round uses four subkey words. There are 16 mixing rounds, so all subkeys are used once. In addition, subkeys are selected in a data-dependent manner for the mashing rounds.

Decryption is performed as the inverse operation of encryption, so that the rounds are performed and the keys are used in reverse order.

#### 4.7 CHARACTERISTICS OF ADVANCED SYMMETRIC BLOCK CIPHERS

Virtually all contemporary symmetric block ciphers are similar in many ways to DES and to the basic Feistel block cipher structure. This is a remarkable testament to those at IBM and NSA responsible for the design of DES. However, as knowledge of cryptanalysis has evolved and as the need for fast software encryption has emerged, advances have been made. This chapter illustrates those advances as embodied in some of the most important modern symmetric encryption algorithms. In this section, we highlight some of the key features found in some of these algorithms but not found in DES.

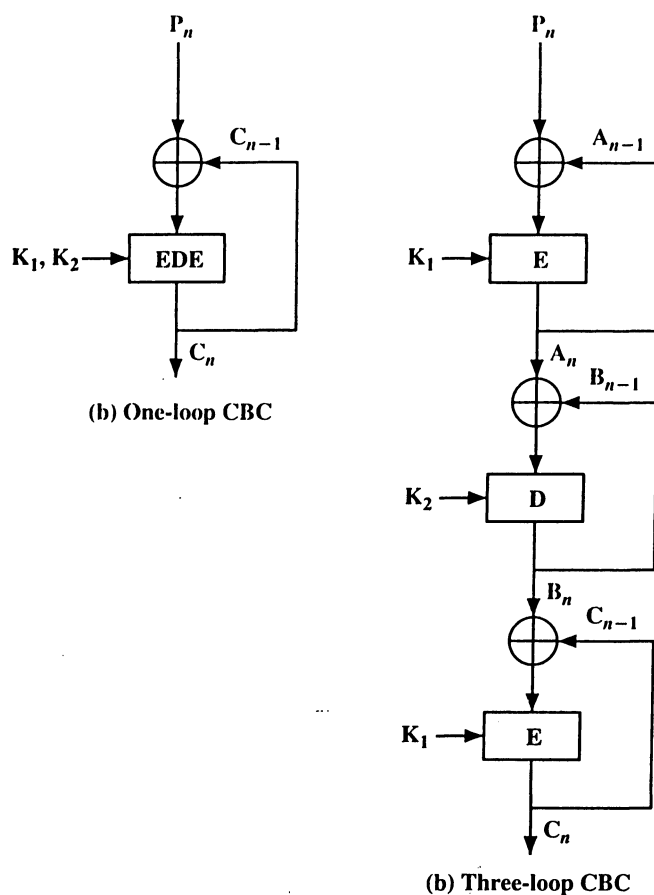
- **Variable key length:** If an encryption algorithm is designed to be extremely resistant to cryptanalysis, then its strength is determined by its key length: The longer the key, the longer it takes for a brute-force key search. Blowfish, RC5, CAST-128, and RC2 provide a variable key length.
- **Mixed operators:** The use of more than one arithmetic and/or Boolean operator complicates cryptanalysis, especially if these operators do not satisfy distributive and associative laws. This approach can provide nonlinearity as an alternative to S-boxes. All of the algorithms in this chapter, except triple DES, use mixed operators.
- **Data-dependent rotation:** Another intriguing alternative to S-boxes is to use rotations that depend on the data. With a sufficient number of rounds, this can provide excellent confusion and diffusion. Further, the rotations are dependent on the blocks of data moving through the rounds, rather than on the subkeys. This would appear to make recovery of the subkeys even more difficult. RC5 uses data-dependent rotations.

- **Key-dependent rotation:** A rotation can be used that depends on the key rather than on the data. This is done in CAST-128.
- **Key-dependent S-boxes:** Rather than attempt to design fixed S-boxes with desirable cryptographic features, such as is done in DES and CAST-128, the content of the S-boxes can be dependent on the key. A different key yields a different S-box. This approach, especially with larger S-boxes (e.g.,  $8 \times 32$ ), should yield highly nonlinear results and should be very difficult to cryptanalyze. Blowfish uses key-dependent S-boxes.
- **Lengthy key schedule algorithm:** This is an ingenious tactic employed in Blowfish. The generation of subkeys takes much longer than a single encryption or decryption. The result is that the effort for a brute-force attack is greatly magnified.
- **Variable F:** The use of a function  $F$  that varies from round to round may complicate the cryptanalysis problem. CAST-128 uses a variable  $F$ .
- **Variable plaintext/ciphertext block length:** A longer block length yields greater cryptographic strength. Also, a variable block length can provide a measure of convenience, allowing the algorithm to be tailored to the application. RC5 adopts this strategy.
- **Variable number of rounds:** Other things being equal, an increase in the number of rounds increases cryptanalytic strength. Of course, an increase in the number of rounds increases the encryption/decryption time. Allowing for a variable number of rounds allows the user to make a tradeoff between security and execution speed. RC5 provides for a variable number of rounds.
- **Operation on both data halves each round:** In the classic Feistel cipher, only one half of the data is altered in each round. If a simple operation were performed on the half that is otherwise not altered, security could be increased with a minimal increase in execution time. IDEA, Blowfish, and RC5 operate on both halves of the data each round.

The current thrust of research is along the lines illustrated in this chapter—namely, finding refinements to the Feistel cipher and DES rather than designing an entirely new structure. The reason for this is that the Feistel structure has been much studied and has revealed no inherent weaknesses.

## 4.8 PROBLEMS

- 4.1 You want to build a hardware device to do block encryption in the cipher block chaining (CBC) mode using an algorithm stronger than DES. Triple DES is a good candidate, but it is defined only in ECB mode. Although there is no established standard for CBC mode for triple DES, Figure 4.16 shows two contenders, both of which follow from the definition of CBC. Which of the two would you choose?
  - a. For security?
  - b. For performance?
- 4.2 Can you suggest a security improvement to either option in Figure 4.16, using only three DES chips and some number of XOR functions? Assume you are still limited to two keys.



**Figure 4.16** Use of Triple DES in CBC Mode.

- 4.3** The Merkle-Hellman attack on triple DES begins by assuming a value of  $A = 0$  (Figure 4.1b). Then, for each of the  $2^{56}$  possible values of  $K_1$ , the plaintext  $P$  that produces  $A = 0$  is determined. Describe the rest of the algorithm.
- 4.4** The most difficult part of the implementation of IDEA is multiplication modulo  $(2^{16} + 1)$ . The following equality suggests a way in which an efficient implementation can be achieved. Let  $a, b$  be two  $n$ -bit nonzero integers. Then

$$ab \bmod (2^n + 1) = \begin{cases} (ab \bmod 2^n) - (ab \operatorname{div} 2^n) & \text{if } (ab \bmod 2^n) \geq (ab \operatorname{div} 2^n) \\ (ab \bmod 2^n) - (ab \operatorname{div} 2^n) + 2^n + 1 & \text{if } (ab \bmod 2^n) < (ab \operatorname{div} 2^n) \end{cases}$$

Note that  $(ab \bmod 2^n)$  corresponds to the  $n$  least significant bits of  $ab$ , and  $(ab \operatorname{div} 2^n)$  is just the right shift of  $ab$  by  $n$  bits. The purpose of this problem is to prove that the foregoing equation is true.

- Show that there exist unique nonnegative integers  $q$  and  $r$  such that  $ab = q(2^n + 1) + r$ .
- What are the upper and lower bounds on  $q$  and  $r$ ?
- Prove that  $q + r < 2^{n+1}$ .
- Derive an expression for  $(ab \operatorname{div} 2^n)$  in terms of  $q$ .
- Derive an expression for  $(ab \bmod 2^n)$  in terms of  $q$  and  $r$ .
- Derive an expression for  $r$  using the results of (d) and (e).
- Demonstrate that  $r$  is equal to the right-hand side of the equation at the beginning of this problem.



- 4.5 IDEA uses four operations to provide complete diffusion (Figure 4.3). Prove that this is the minimum number of required operations. To do this, consider a function of the form

$$(Y_1, Y_2) = E(X_1, X_2, Z_1, Z_2) \quad 0 \leq X_i, Y_i \leq 2^m; 0 \leq Z_i \leq 2^k$$

such that for every choice of  $(Z_1, Z_2)$ ,  $E(\cdot, \cdot, Z_1, Z_2)$  is invertible. Such a function may be called a cipher function. A cipher function is said to have *complete diffusion* if each of its output variables depends on every input variable. We would like to prove that, if a cipher function is of the foregoing form and has complete diffusion, then the algorithm contains at least four operations.

- Show that the function must contain at least three operations.
  - Now suppose that  $E$  has exactly three operations and demonstrate that such a function cannot be invertible.
- 4.6
- Why is the multiplication operation of IDEA modulo  $(2^{16} + 1)$  instead of simply  $2^{16}$ ?
  - Why is the addition operation of IDEA modulo  $2^{16}$  instead of  $(2^{16} + 1)$ ?
- 4.7 The original proposal for an encryption algorithm by Lai and Massey, referred to as PES, differs from IDEA as follows:
- The four functions in the upper gray box of Figure 4.5 are in the order  $\odot, \oplus, \oplus, \odot$  for IDEA and in the order  $\odot, \odot, \oplus, \oplus$  for PES.
  - In IDEA, after each round, the second and third blocks are interchanged. In PES, after each round, the first and second blocks are interchanged with the third and fourth blocks.

It can be shown that the second change increases resistance to differential cryptanalysis (see [LAI91]). Demonstrate that the first change makes no difference in cryptographic strength, *Hint*: can you do some pre- and post-processing that provides the same effect?

- 4.8 Demonstrate that Blowfish decryption is the inverse of Blowfish encryption.
- 4.9 Demonstrate that RC5 decryption is the inverse of RC5 encryption.
- 4.10 In the RC5-CBC-Pad mode, there are from one to  $bb$  bytes of padding. Why not allow zero bytes of padding? That is, if the message to be encrypted is an integer multiple of the block size, why not refrain from padding?
- 4.11 Figure 4.17 shows an alternative to RC5-CTS for producing ciphertext of equal length to the plaintext when the plaintext is not an integer multiple of the block size.
- Explain the algorithm.
  - Explain why RC5-CTS is preferable to the approach illustrated in Figure 4.17.
- 4.12 Describe how to decrypt  $C_{n-1}$  and  $C_n$  in RC5-CTS mode.
- 4.13 Demonstrate that CAST-128 decryption is the inverse of CAST-128 encryption.

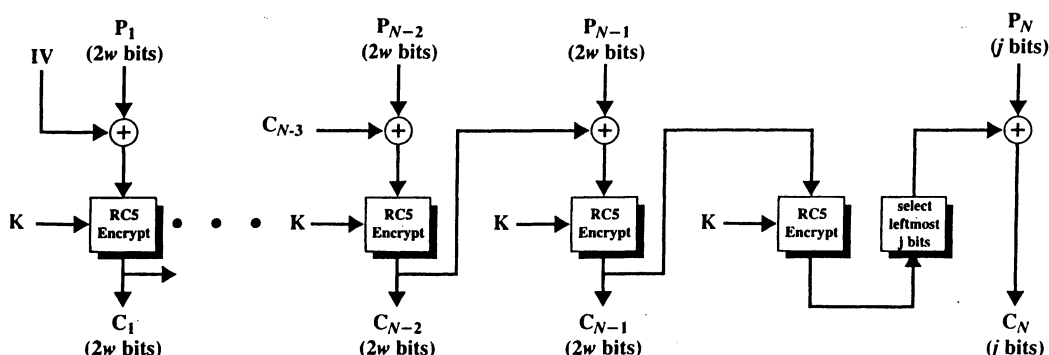


Figure 4.17 A Method of Encrypting the Last Short Block in RC5 CBC Mode.



## CHAPTER 5

# CONFIDENTIALITY USING CONVENTIONAL ENCRYPTION

*Amongst the tribes of Central Australia every man, woman, and child has a secret or sacred name which is bestowed by the older men upon him or her soon after birth, and which is known to none but the fully initiated members of the group. This secret name is never mentioned except upon the most solemn occasions; to utter it in the hearing of men of another group would be a most serious breach of tribal custom. When mentioned at all, the name is spoken only in a whisper, and not until the most elaborate precautions have been taken that it shall be heard by no one but members of the group. The native thinks that a stranger knowing his secret name would have special power to work him ill by means of magic.*

**—The Golden Bough, Sir James George Frazer**

*John wrote the letters of the alphabet under the letters in its first lines and tried it against the message. Immediately he knew that once more he had broken the code. It was extraordinary the feeling of triumph he had. He felt on top of the world. For not only had he done it, had he broken the July code, but he now had the key to every future coded message, since instructions as to the source of the next one must of necessity appear in the current one at the end of each month.*

**—Talking to Strange Men, Ruth Rendell**

**H**istorically, the focus of cryptology has been on the use of conventional encryption to provide confidentiality. It is only in the last several decades that other considerations, such as authentication,

integrity, digital signatures, and the use of public-key encryption, have been included in the theory and practice of cryptology.

Before examining some of these more recent topics, we concentrate in this chapter on the use of conventional encryption to provide confidentiality. This topic remains important in itself. In addition, an understanding of the issues involved here helps to motivate the development of public-key encryption and clarifies the issues involved in other applications of encryption, such as authentication.

We begin with a discussion of the location of encryption logic; the main choice here is between what are known as link and end-to-end encryption. Next, we look at the use of encryption to counter traffic analysis attacks. Then we discuss the difficult problem of key distribution. Finally, we discuss the principles underlying an important tool in providing a confidentiality facility: random number generation.

## 5.1 PLACEMENT OF ENCRYPTION FUNCTION

If encryption is to be used to counter attacks on confidentiality, we need to decide what to encrypt and where the encryption function should be located. To begin, this section examines the potential locations of security attacks and then looks at the two major approaches to encryption placement: link and end-to-end.

### Potential Locations for Confidentiality Attacks

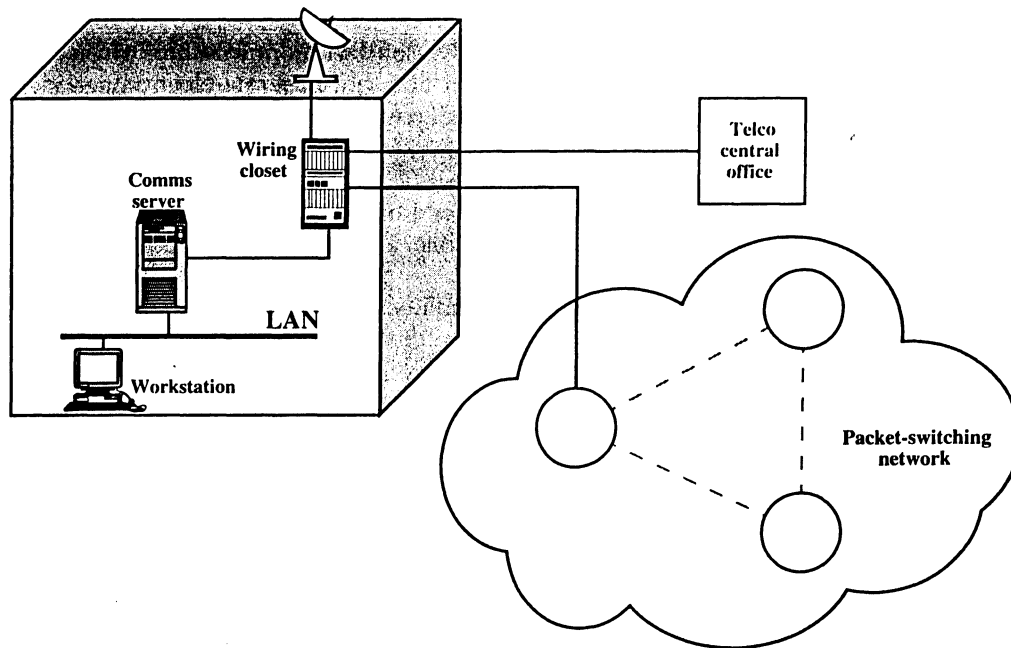
As an example, consider a user workstation in a typical business organization. Figure 5.1 suggests the types of communications facilities that might be employed by such a workstation and therefore gives an indication of the points of vulnerability.

In most organizations, workstations are attached to local area networks (LANs). Typically, the user can reach other workstations, hosts, and servers directly on the LAN or on other LANs in the same building that are interconnected with bridges and routers. Here, then, is the first point of vulnerability. In this case, the main concern is eavesdropping by another employee. Typically, a LAN is a broadcast network: Transmission from any station to any other station is visible on the LAN medium to all stations. Data are transmitted in the form of frames, with each frame containing the source and destination address. An eavesdropper can monitor the traffic on the LAN and capture any traffic desired on the basis of source and destination addresses.

Furthermore, the eavesdropper need not necessarily be an employee in the building. If the LAN, through a communications server or one of the hosts on the LAN, offers a dial-in capability, then it is possible for an intruder to gain access to the LAN and monitor traffic.

Access to the outside world from the LAN is almost always available in the form of a router, a bank of dial-out modems, or some other type of communications server. From the communications server, there is a line leading to a wiring closet. The wiring closet serves as a patch panel for interconnecting internal data and phone lines and for providing a staging point for external communications.

The wiring closet itself is vulnerable. If an intruder can penetrate to the closet, he or she can tap into each wire to determine which are used for data transmission.



**Figure 5.1** Points of Vulnerability.

After isolating one or more lines, the intruder can attach a low-power radio transmitter. The resulting signals can be picked up from a nearby location (e.g., a parked van or a nearby building).

Several routes out of the wiring closet are possible. A standard configuration provides access to the nearest central office of the local telephone company. Wires in the closet are gathered into a cable, which is usually consolidated with other cables in the basement of the building. From there, a larger cable runs underground to the central office.

In addition, the wiring closet may provide a link to a microwave antenna, either an earth station for a satellite link or a point-to-point terrestrial microwave link. The antenna link can be part of a private network, or it can be a local bypass to hook in to a long-distance carrier, such as AT&T or MCI.

The wiring closet may also provide a link to a node of a packet-switching network. This link can be a leased line, a direct private line, or a switched connection through a public telecommunications network such as ISDN. Inside the network, data pass through a number of nodes and links between nodes until the data arrive at the node to which the destination end system is connected.

An attack can take place at any of the communications links. For active attacks, the attacker needs to gain physical control of a portion of the link and be able to insert and capture transmissions. For a passive attack, the attacker merely needs to be able to observe transmissions. The communications links involved can be cable (telephone twisted pair, coaxial cable, or optical fiber), microwave links, or satellite channels. Twisted pair and coaxial cable can be attacked using either invasive taps or inductive devices that monitor electromagnetic emanation. Invasive taps

allow both active and passive attacks, whereas inductive taps are useful for passive attacks. Neither type of tap is particularly useful with optical fiber, which is one of the advantages of this medium. The fiber does not generate electromagnetic emanations and hence is not vulnerable to inductive taps. Physically breaking the cable seriously degrades signal quality and is therefore detectable. Microwave and satellite transmissions can be intercepted with little risk to the attacker. This is especially true of satellite transmissions, which cover a broad geographic area. Active attacks on microwave and satellite are also possible, although they are more difficult technically and can be quite expensive.

In addition to the potential vulnerability of the various communications links, the various processors along the path are themselves subject to attack. An attack can take the form of attempts to modify the hardware or software, to gain access to the memory of the processor, or to monitor the electromagnetic emanations. These attacks are less likely than those involving communications links but are nevertheless a source of risk.

Thus, there are a large number of locations at which an attack can occur. Furthermore, for wide area communications, many of these locations are not under the physical control of the end user. Even in the case of local area networks, in which physical security measures are possible, there is always the threat of the disgruntled employee.

### **Link versus End-to-End Encryption**

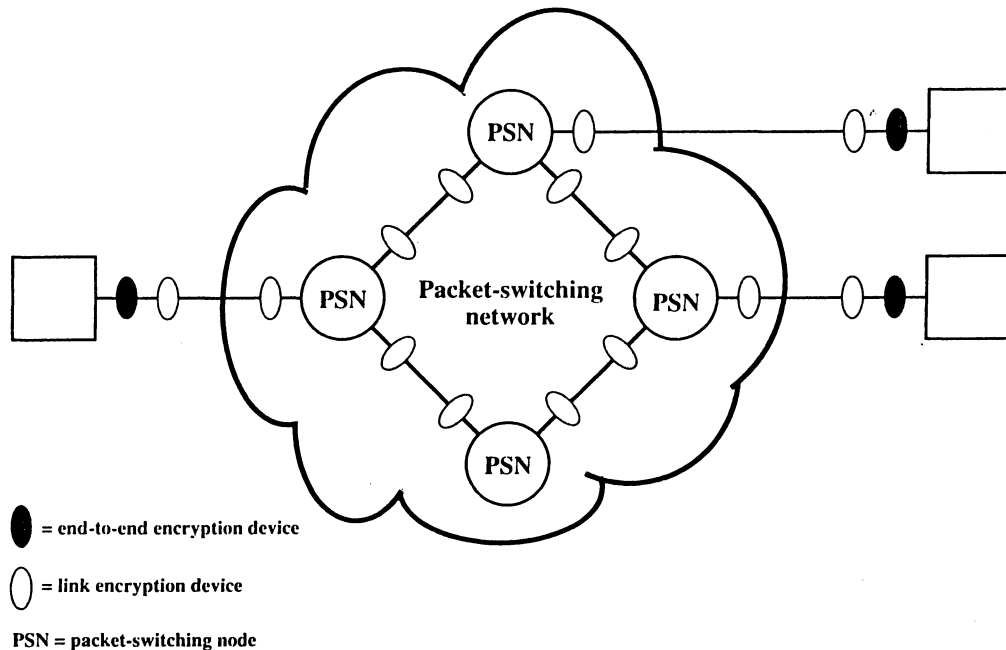
The most powerful and most common approach to securing the points of vulnerability highlighted in the preceding section is encryption. If encryption is to be used to counter these attacks, then we need to decide what to encrypt and where the encryption gear should be located. As Figure 5.2 indicates, there are two fundamental alternatives: link encryption and end-to-end encryption.

#### **Basic Approaches**

With link encryption, each vulnerable communications link is equipped on both ends with an encryption device. Thus, all traffic over all communications links is secured. Although this recourse requires a lot of encryption devices in a large network, its value is clear. One of its disadvantages is that the message must be decrypted each time it enters a packet switch because the switch must read the address (virtual circuit number) in the packet header in order to route the packet. Thus, the message is vulnerable at each switch. If working with a public packet-switching network, the user has no control over the security of the nodes.

Several implications of link encryption should be noted. For this strategy to be effective, all the potential links in a path from source to destination must use link encryption. Each pair of nodes that share a link should share a unique key, with a different key used on each link. Thus, many keys must be provided. However, each key must be distributed to only two nodes.

With end-to-end encryption, the encryption process is carried out at the two end systems. The source host or terminal encrypts the data. The data in encrypted form are then transmitted unaltered across the network to the destination terminal or host. The destination shares a key with the source and so is able to decrypt the



**Figure 5.2** Encryption Across a Packet-Switching Network.

data. This plan seems to secure the transmission against attacks on the network links or switches. Thus, end-to-end encryption relieves the end user of concerns about the degree of security of networks and links that support the communication. There is, however, still a weak spot.

Consider the following situation. A host connects to an X.25 packet-switching network, sets up a virtual circuit to another host, and is prepared to transfer data to that other host by using end-to-end encryption. Data are transmitted over such a network in the form of packets that consist of a header and some user data. What part of each packet will the host encrypt? Suppose that the host encrypts the entire packet, including the header. This will not work because, remember, only the other host can perform the decryption. The packet-switching node will receive an encrypted packet and be unable to read the header. Therefore, it will not be able to route the packet. It follows that the host may encrypt only the user data portion of the packet and must leave the header in the clear.

Thus, with end-to-end encryption, the user data are secure. However, the traffic pattern is not, because packet headers are transmitted in the clear. On the other hand, end-to-end encryption does provide a degree of authentication. If two end systems share an encryption key, then a recipient is assured that any message that it receives comes from the alleged sender, because only that sender shares the relevant key. Such authentication is not inherent in a link encryption scheme.

To achieve greater security, both link and end-to-end encryption are needed, as is shown in Figure 5.2. When both forms of encryption are employed, the host encrypts the user data portion of a packet using an end-to-end encryption key. The entire packet is then encrypted using a link encryption key. As the packet traverses

**Table 5.1** Characteristics of Link and End-to-End Encryption [PFLE97]

Link Encryption	End-to-End Encryption
<i>Security within End Systems and Intermediate Systems</i>	
Message exposed in sending host	Message encrypted in sending host
Message exposed in intermediate nodes	Message encrypted in intermediate nodes
<i>Role of User</i>	
Applied by sending host	Applied by sending process
Transparent to user	User applies encryption
Host maintains encryption facility	User must determine algorithm
One facility for all users	User selects encryption scheme
Can be done in hardware	Software implementation
All or no messages encrypted	User chooses to encrypt, or not, for each message
<i>Implementation Concerns</i>	
Requires one key per host/intermediate node	Requires one key per user pair
and intermediate node/intermediate node pair	Provides user authentication
Provides host authentication	

the network, each switch decrypts the packet, using a link encryption key to read the header, and then encrypts the entire packet again for sending it out on the next link. Now the entire packet is secure except for the time that the packet is actually in the memory of a packet switch, at which time the packet header is in the clear.

Table 5.1 summarizes the key characteristics of the two encryption strategies.

### Logical Placement of End-to-End Encryption Function

With link encryption, the encryption function is performed at a low level of the communications hierarchy. In terms of the open systems interconnection (OSI) model, link encryption occurs at either the physical or link layers.

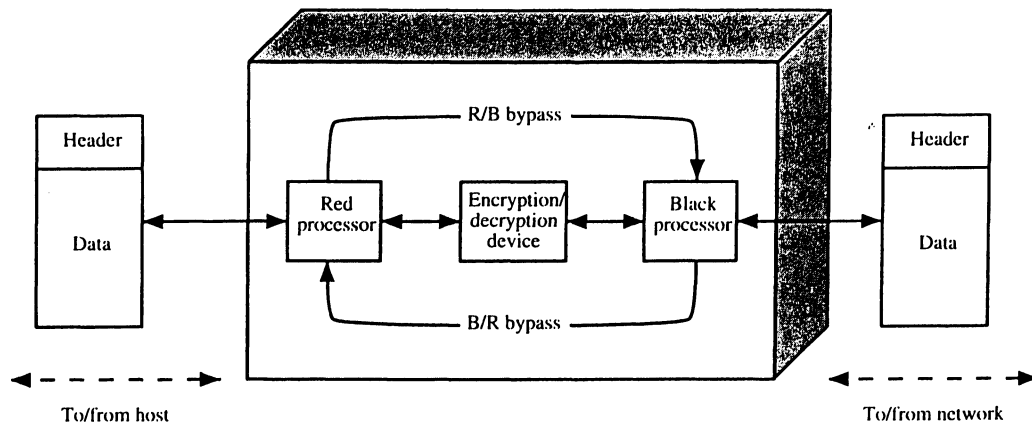
For end-to-end encryption, several choices are possible for the logical placement of the encryption function. At the lowest practical level, the encryption function could be performed at the network layer. Thus, for example, encryption could be associated with X.25, so that the user data portion of all X.25 packets is encrypted.

With network-layer encryption, the number of identifiable and separately protected entities corresponds to the number of end systems in the network. Each end system can engage in an encrypted exchange with another end system if the two share a secret key. All the user processes and applications within each end system would employ the same encryption scheme with the same key to reach a particular target end system. With this arrangement, it might be desirable to off-load the encryption function to some sort of front-end processor (typically a communications board in the end system).

Figure 5.3 shows the encryption function of the front-end processor (FEP). On the host side, the FEP accepts packets. The user data portion of the packet is encrypted, while the packet header bypasses the encryption process.<sup>1</sup> The resulting packet is delivered to the network. In the opposite direction, for packets arriving from the network, the user data portion is decrypted and the entire packet is deliv-

<sup>1</sup>The terms *red* and *black* are frequently used. Red data are sensitive or classified data in the clear. Black data are encrypted data.





**Figure 5.3** Front-End Processor Function.

ered to the host. If the transport layer functionality (e.g., the ISO transport protocol or TCP) is implemented in the front end, then the transport-layer header would also be left in the clear and the user data portion of the transport protocol data unit is encrypted.

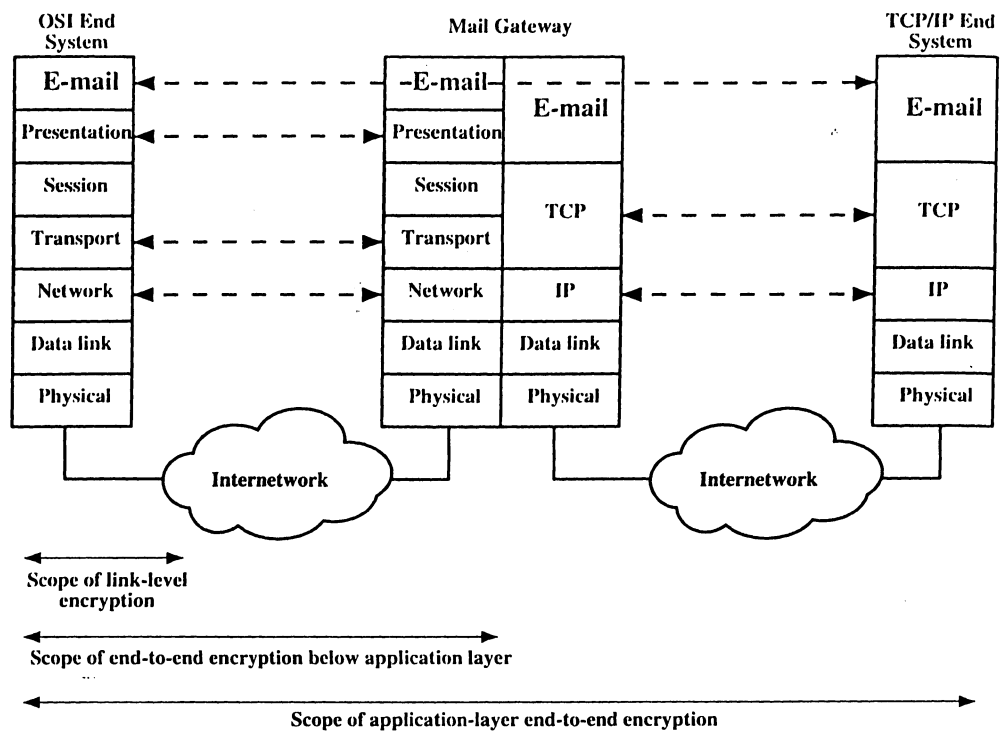
Deployment of encryption services on end-to-end protocols, such as a network-layer X.25 or TCP, provides end-to-end security for traffic within a fully integrated internetwork. However, such a scheme cannot deliver the necessary service for traffic that crosses internetwork boundaries, such as electronic mail, electronic data interchange (EDI), and file transfers.

Figure 5.4 illustrates the issues involved. In this example, an electronic mail gateway is used to interconnect an internetwork that uses an OSI-based architecture with one that uses a TCP/IP-based architecture. In such a configuration, there is no end-to-end protocol below the application layer. The transport and network connections from each end system terminate at the mail gateway, which sets up new transport and network connections to link to the other end system. Furthermore, such a scenario is not limited to the case of a gateway between two different architectures. Even if both end systems use TCP/IP or OSI, there are plenty of instances in actual configurations in which mail gateways sit between otherwise isolated internetworks. Thus, for applications like electronic mail that have a store-and-forward capability, the only place to achieve end-to-end encryption is at the application layer.

A drawback of application-layer encryption is that the number of entities to consider increases dramatically. A network that supports hundreds of hosts may support thousands of users and processes. Thus, many more secret keys need to be generated and distributed.

An interesting way of viewing the alternatives is to note that as we move up the communications hierarchy, less information is encrypted but it is more secure. Figure 5.5 highlights this point, using the TCP/IP architecture as an example. In the figure, an application-level gateway refers to a store-and-forward device that operates at the application level.<sup>2</sup>

<sup>2</sup>Unfortunately, most TCP/IP documents use the term *gateway* to refer to what is more commonly referred to as a *router*.



**Figure 5.4** Encryption Coverage Implications of Store-and-Forward Communications.

With application-level encryption (Figure 5.5a), only the user data portion of a TCP segment is encrypted. The TCP, IP, network-level, and link-level headers and link-level trailer are in the clear. By contrast, if encryption is performed at the TCP level (Figure 5.5b), then, on a single end-to-end connection, the user data and the TCP header are encrypted. The IP header remains in the clear because it is needed by routers to route the IP datagram from source to destination. Note, however, that if a message passes through a gateway, the TCP connection is terminated and a new transport connection is opened for the next hop. Furthermore, the gateway is treated as a destination by the underlying IP. Thus, the encrypted portions of the data unit are decrypted at the gateway. If the next hop is over a TCP/IP network, then the user data and TCP header are encrypted again before transmission. However, in the gateway itself the data unit is buffered entirely in the clear. Finally, for link-level encryption (Figure 5.5c), the entire data unit except for the link header and trailer is encrypted on each link, but the entire data unit is in the clear at each router and gateway.<sup>3</sup>

<sup>3</sup>The figure actually shows but one alternative. It is also possible to encrypt part or even all of the link header and trailer except for the starting and ending frame flags.

Link-H	Net-H	IP-H	TCP-H	Data	Link-T
--------	-------	------	-------	------	--------

(a) Application-Level Encryption (on links and at routers and gateways)

Link-H	Net-H	IP-H	TCP-H	Data	Link-T
--------	-------	------	-------	------	--------

On links and at routers

Link-H	Net-H	IP-H	TCP-H	Data	Link-T
--------	-------	------	-------	------	--------

In gateways

(b) TCP-Level Encryption

Link-H	Net-H	IP-H	TCP-H	Data	Link-T
--------	-------	------	-------	------	--------

On links

Link-H	Net-H	IP-H	TCP-H	Data	Link-T
--------	-------	------	-------	------	--------

In routers and gateways

(c) Link-Level Encryption

TCP-H	=	TCP header
IP-H	=	IP header
Net-H	=	Network-level header (e.g., X.25 packet header, LLC header)
Link-H	=	Data link control protocol header
Link-T	=	Data link control protocol trailer

Figure 5.5 Implications of Various Encryption Strategies.

## 5.2 TRAFFIC CONFIDENTIALITY

We mentioned in Chapter 1 that, in some cases, users are concerned about security from traffic analysis. Knowledge about the number and length of messages between nodes may enable an opponent to determine who is talking to whom. This can have obvious implications in a military conflict. Even in commercial applications, traffic analysis may yield information that the traffic generators would like to conceal. [MUFT89] lists the following types of information that can be derived from a traffic analysis attack:

- Identities of partners
- How frequently the partners are communicating
- Message pattern, message length, or quantity of messages that suggest important information is being exchanged
- The events that correlate with special conversations between particular partners

Another concern related to traffic is the use of traffic patterns to create a *covert channel*. A covert channel is a means of communication in a fashion unintended by the designers of the communications facility. Typically, the channel is used to transfer information in a way that violates a security policy. For example, an employee may wish to communicate information to an outsider in a way that is not detected by management and that requires simple eavesdropping on the part of the outsider. The two participants could set up a code in which an apparently legitimate message of a less than a certain length represents binary zero, whereas a longer message represents a binary one. Other such schemes are possible.

### Link Encryption Approach

With the use of link encryption, packet headers are encrypted, reducing the opportunity for traffic analysis. However, it is still possible in those circumstances for an attacker to assess the amount of traffic on a network and to observe the amount of traffic entering and leaving each end system. An effective countermeasure to this attack is traffic padding, illustrated in Figure 5.6.

Traffic padding produces ciphertext output continuously, even in the absence of plaintext. A continuous random data stream is generated. When plaintext is available, it is encrypted and transmitted. When input plaintext is not present, random data are encrypted and transmitted. This makes it impossible for an attacker to distinguish between true data flow and padding and therefore impossible to deduce the amount of traffic.

### End-to-End Encryption Approach

Traffic padding is essentially a link encryption function. If only end-to-end encryption is employed, then the measures available to the defender are more limited. For example, if encryption is implemented at the application layer, then an opponent can determine which transport entities are engaged in dialogue. If encryption techniques are housed at the transport layer, then network-layer addresses and traffic patterns remain accessible.

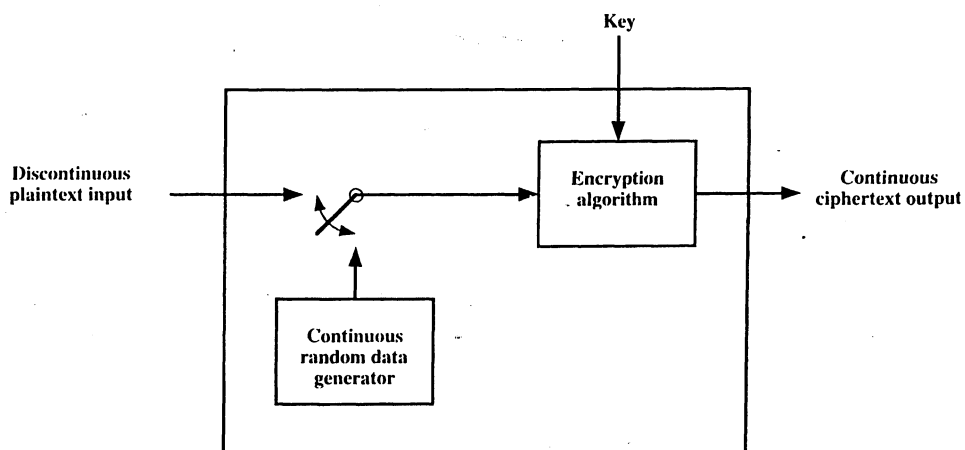


Figure 5.6 Traffic-Padding Encryption Device.

One technique that might prove useful is to pad out data units to a uniform length at either the transport or application level. In addition, null messages can be inserted randomly into the stream. These tactics deny an opponent knowledge about the amount of data exchanged between end users and obscure the underlying traffic pattern.

### 5.3 KEY DISTRIBUTION

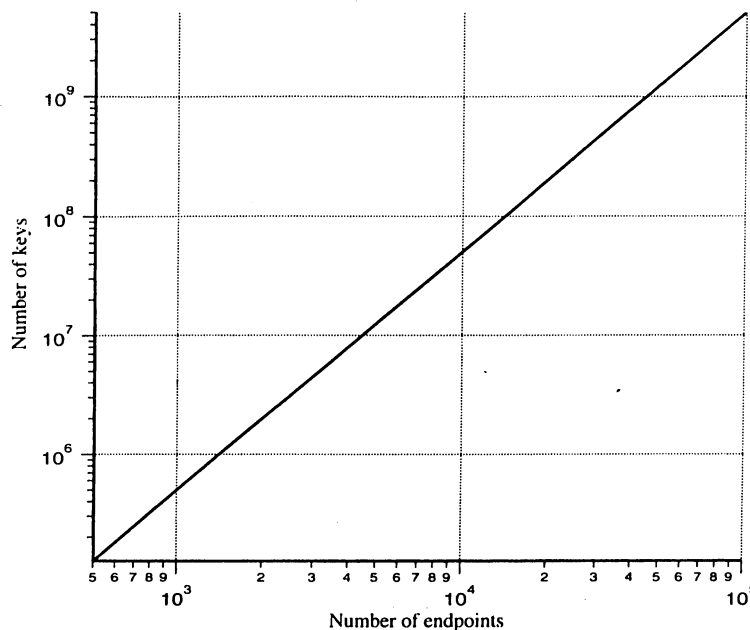
For conventional encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Furthermore, frequent key changes are usually desirable to limit the amount of data compromised if an attacker learns the key. Therefore, the strength of any cryptographic system rests with the *key distribution technique*, a term that refers to the means of delivering a key to two parties who wish to exchange data, without allowing others to see the key. For two parties A and B, key distribution can be achieved in a number of ways, as follows:

1. A key can be selected by A and physically delivered to B.
2. A third party can select the key and physically deliver it to A and B.
3. If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.
4. If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B.

Options 1 and 2 call for manual delivery of a key. For link encryption, this is a reasonable requirement, because each link encryption device is going to be exchanging data only with its partner on the other end of the link. However, for end-to-end encryption, manual delivery is awkward. In a distributed system, any given host or terminal may need to engage in exchanges with many other hosts and terminals over time. Thus, each device needs a number of keys supplied dynamically. The problem is especially difficult in a wide area distributed system.

The scale of the problem depends on the number of communicating pairs that must be supported. If end-to-end encryption is done at a network or IP level, then a key is needed for each pair of hosts on the network that wish to communicate. Thus, if there are  $N$  hosts, the number of required keys is  $[N(N - 1)]/2$ . If encryption is done at the application level, then a key is needed for every pair of users or processes that require communication. Thus, a network may have hundreds of hosts but thousands of users and processes. Figure 5.7 illustrates the magnitude of the key distribution task for end-to-end encryption. A network using node-level encryption with 1000 nodes would conceivably need to distribute as many as half a million keys. If that same network supported 10,000 applications, then as many as 50 million keys may be required for application-level encryption.

Returning to our list, option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then



**Figure 5.7** Number of Keys Required to Support Arbitrary Connections between Endpoints.

all subsequent keys will be revealed. Furthermore, the initial distribution of potentially millions of keys must still be made.

For end-to-end encryption, some variation on option 4 has been widely adopted. In this scheme, a key distribution center is responsible for distributing keys to pairs of users (hosts, processes, applications) as needed. Each user must share a unique key with the key distribution center for purposes of key distribution.

The use of a key distribution center is based on the use of a hierarchy of keys. At a minimum, two levels of keys are used (Figure 5.8). Communication between end systems is encrypted using a temporary key, often referred to as a *session key*. Typically, the session key is used for the duration of a logical connection, such as a virtual circuit or transport connection, and then discarded. Each session key is obtained from the key distribution center over the same networking facilities used for end-user communication. Accordingly, session keys are transmitted in encrypted form, using a *master key* that is shared by the key distribution center and an end system or user.

For each end system or user, there is a unique master key that it shares with the key distribution center. Of course, these master keys must be distributed in some fashion. However, the scale of the problem is vastly reduced. If there are  $N$  entities that wish to communicate in pairs, then, as was mentioned, as many as  $[N(N-1)]/2$  session keys are needed at any one time. However, only  $N$  master keys are required, one for each entity. Thus, master keys can be distributed in some noncryptographic way, such as physical delivery.

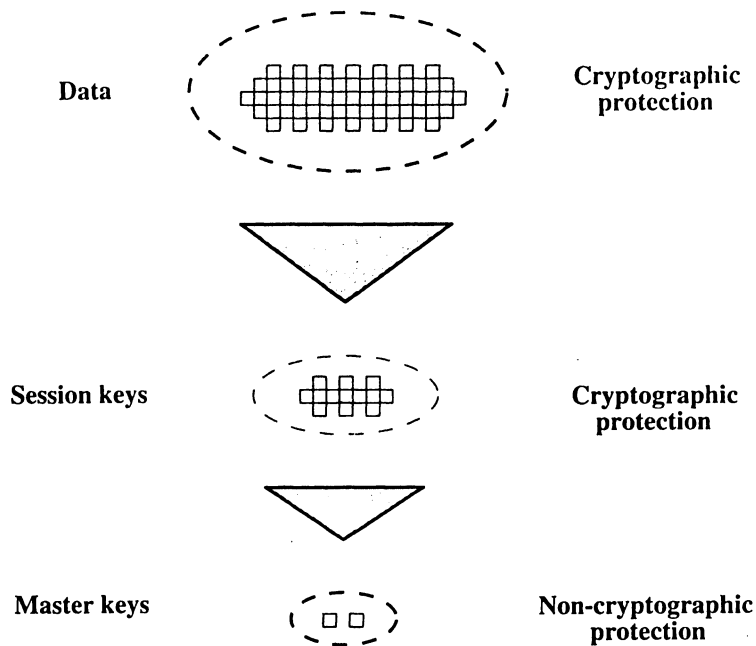


Figure 5.8 The Use of a Key Hierarchy.

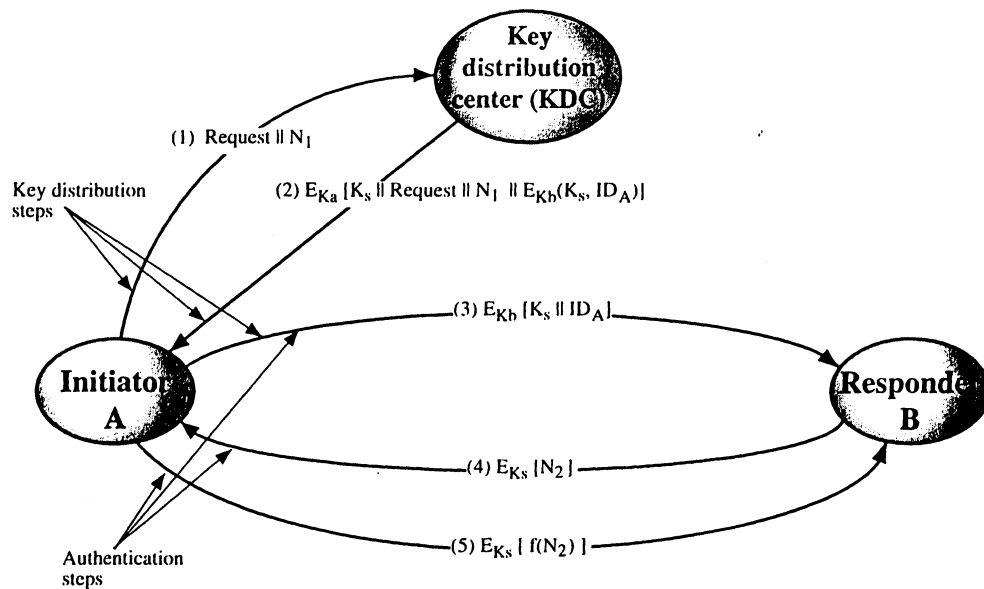
### A Key Distribution Scenario

The key distribution concept can be deployed in a number of ways. A typical scenario is illustrated in Figure 5.9, which is based on a figure in [POPE79]. The scenario assumes that each user shares a unique master key with the key distribution center (KDC).

Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection. A has a secret key,  $K_a$ , known only to itself and the KDC; similarly, B shares the master key  $K_b$  with the KDC. The following steps occur:

1. A issues a request to the KDC for a session key to protect a logical connection to B. The message includes the identity of A and B and a unique identifier,  $N_1$ , for this transaction, which we refer to as a *nonce*.<sup>4</sup> The nonce may be a timestamp, a counter, or a random number; the minimum requirement is that it differ with each request. Also, to prevent masquerade, it should be difficult for an opponent to guess the nonce. Thus, a random number is a good choice for a nonce.
2. The KDC responds with a message encrypted using  $K_a$ . Thus, A is the only one who can successfully receive the message, and A knows that it originated at the KDC. The message includes two items intended for A:

<sup>4</sup>The following definitions are useful in understanding the purpose of the nonce component. **Nonce:** The present or particular occasion. **Nonce word:** A word occurring, invented, or used just for a particular occasion. (From the *American Heritage Dictionary of the English Language*, 3rd ed.)



**Figure 5.9** Key Distribution Scenario.

- The one-time session key,  $K_s$ , to be used for the session
- The original request message, including the nonce, to enable A to match this response with the appropriate request

Thus, A can verify that its original request was not altered before reception by the KDC and, because of the nonce, that this is not a replay of some previous request.

In addition, the message includes two items intended for B:

- The one-time session key,  $K_s$ , to be used for the session
- An identifier of A (e.g., its network address),  $ID_A$

These last two items are encrypted with  $K_b$  (the master key that the KDC shares with B). They are to be sent to B to establish the connection and prove A's identity.

3. A stores the session key for use in the upcoming session and forwards to B the information that originated at the KDC for B—namely,  $E_{K_b}[K_s \parallel ID_A]$ . Because this information is encrypted with  $K_b$ , it is protected from eavesdropping. B now knows the session key ( $K_s$ ), knows that the other party is A (from  $ID_A$ ), and knows that the information originated at the KDC (because it is encrypted using  $E_{K_b}$ ).

At this point, a session key has been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

4. Using the newly minted session key for encryption, B sends a nonce,  $N_2$ , to A.
5. Also using  $K_s$ , A responds with  $f(N_2)$ , where  $f$  is a function that performs some transformation on  $N_2$  (e.g., adding one).



These steps assure B that the original message it received (step 3) was not a replay.

Note that the actual key distribution involves only steps 1 through 3 but that steps 4 and 5, as well as 3, perform an authentication function.

### **Hierarchical Key Control**

It is not necessary to limit the key distribution function to a single KDC. Indeed, for very large networks, it may not be practical to do so. As an alternative, a hierarchy of KDCs can be established. For example, there can be local KDCs, each responsible for a small domain of the overall internetwork, such as a single LAN or a single building. For communication among entities within the same local domain, the local KDC is responsible for key distribution. If two entities in different domains desire a shared key, then the corresponding local KDCs can communicate through a global KDC. In this case, any one of the three KDCs involved can actually select the key. The hierarchical concept can be extended to three or even more layers, depending on the size of the user population and the geographic scope of the internetwork.

A hierarchical scheme minimizes the effort involved in master key distribution, because most master keys are those shared by a local KDC with its local entities. Furthermore, such a scheme limits the damage of a faulty or subverted KDC to its local area only.

### **Session Key Lifetime**

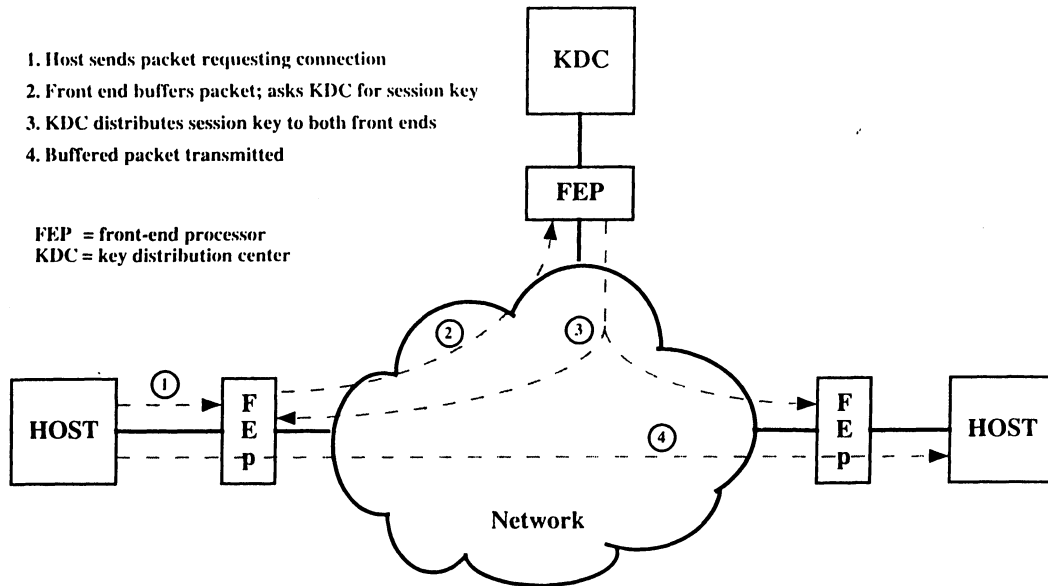
The more frequently session keys are exchanged, the more secure they are, because the opponent has less ciphertext to work with for any given session key. On the other hand, the distribution of session keys delays the start of any exchange and places a burden on network capacity. A security manager must try to balance these competing considerations in determining the lifetime of a particular session key.

For connection-oriented protocols, one obvious choice is to use the same session key for the length of time that the connection is open, using a new session key for each new session. If a logical connection has a very long lifetime, then it would be prudent to change the session key periodically, perhaps every time the PDU (protocol data unit) sequence number cycles.

For a connectionless protocol, such as a transaction-oriented protocol, there is no explicit connection initiation or termination. Thus, it is not obvious how often one needs to change the session key. The most secure approach is to use a new session key for each exchange. However, this negates one of the principal benefits of connectionless protocols, which is minimum overhead and delay for each transaction. A better strategy is to use a given session key for a certain fixed period only or for a certain number of transactions.

### **A Transparent Key Control Scheme**

The approach suggested in Figure 5.9 has many variations, one of which is described in this subsection. The scheme (Figure 5.10) is useful for providing end-to-end encryption at a network or transport level in a way that is transparent to the end users. The approach assumes that communication makes use of a connection-oriented end-to-end protocol, such as X.25 or TCP. The noteworthy element of



**Figure 5.10** Automatic Key Distribution for Connection-Oriented Protocol.

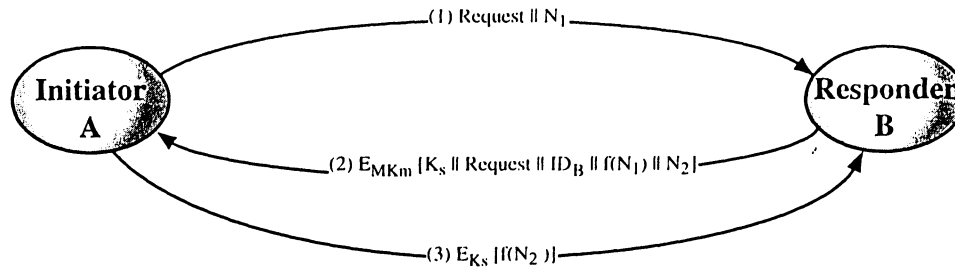
this approach is a front-end processor (such as shown in Figure 5.3) that performs end-to-end encryption and obtains session keys on behalf of its host or terminal.

Figure 5.10 shows the steps involved in establishing a connection. When one host wishes to set up a connection to another host, it transmits a connection-request packet (step 1). The front-end processor saves that packet and applies to the KDC for permission to establish the connection (step 2). The communication between the FEP and the KDC is encrypted using a master key shared only by the FEP and the KDC. If the KDC approves the connection request, it generates the session key and delivers it to the two appropriate front-end processors, using a unique permanent key for each front end (step 3). The requesting front-end processor can now release the connection request packet, and a connection is set up between the two end systems (step 4). All user data exchanged between the two end systems are encrypted by their respective front-end processors using the one-time session key.

The advantage of this approach is that it minimizes the impact on the end systems. From the host's point of view, the FEP appears to be a packet-switching node, and the host interface to the network is unchanged. From the network's point of view, the FEP appears to be a host, and the packet-switch interface to the host is unchanged.

### Decentralized Key Control

The use of a key distribution center imposes the requirement that the KDC be trusted and be protected from subversion. This requirement can be avoided if key distribution is fully decentralized. Although full decentralization is not practical for larger networks using conventional encryption only, it may be useful within a local context.



**Figure 5.11** Decentralized Key Distribution.

A decentralized approach requires that each end system be able to communicate in a secure manner with all potential partner end systems for purposes of session key distribution. Thus, there may need to be as many as  $[n(n - 1)]/2$  master keys for a configuration with  $n$  end systems.

A session key may be established with the following sequence of steps (Figure 5.11):

1. A issues a request to B for a session key and includes a nonce,  $N_1$ .
2. B responds with a message that is encrypted using the shared master key. The response includes the session key selected by B, an identifier of B, the value  $f(N_1)$ , and another nonce,  $N_2$ .
3. Using the new session key, A returns  $f(N_2)$  to B.

Thus, although each node must maintain at most  $(n - 1)$  master keys, as many session keys as required may be generated and used. Because the messages transferred using the master key are short, cryptanalysis is difficult. As before, session keys are used for only a limited time to protect them.

### Controlling Key Usage

The concept of a key hierarchy and the use of automated key distribution techniques greatly reduce the number of keys that must be manually managed and distributed. It may also be desirable to impose some control on the way in which automatically distributed keys are used. For example, in addition to separating master keys from session keys, we may wish to define different types of session keys on the basis of use, such as

- Data-encrypting key, for general communication across a network
- PIN-encrypting key, for personal identification numbers (PINs) used in electronic funds transfer and point-of-sale applications
- File-encrypting key, for encrypting files stored in publicly accessible locations

To illustrate the value of separating of keys by type, consider the risk that a master key is imported as a data-encrypting key into a device. Normally, the master key is physically secured within the cryptographic hardware of the key distribution

center and of the end systems. Session keys encrypted with this master key are available to application programs, as are the data encrypted with such session keys. However, if a master key is treated as a session key, it may be possible for an unauthorized application to obtain plaintext of session keys encrypted with that master key.

Thus, it may be desirable to institute controls in systems that limit the ways in which keys are used, based on characteristics associated with those keys. One simple plan is to associate a tag with each key ([JONE82]; see also [DAVI89]). The proposed technique is for use with DES and makes use of the extra 8 bits in each 64-bit DES key. That is, the 8 nonkey bits ordinarily reserved for parity checking form the key tag. The bits have the following interpretation:

- One bit indicates whether the key is a session key or a master key.
- One bit indicates whether the key can be used for encryption.
- One bit indicates whether the key can be used for decryption.
- The remaining bits are spares for future use.

Because the tag is embedded in the key, it is encrypted along with the key when that key is distributed, thus providing protection. The drawbacks of this scheme are that (1) the tag length is limited to 8 bits, limiting its flexibility and functionality; and (2) because the tag is not transmitted in clear form, it can be used only at the point of decryption, limiting the ways in which key use can be controlled.

A more flexible scheme, referred to as the control vector, is described in [MATY91a and b]. In this scheme, each session key has an associated control vector consisting of a number of fields that specify the uses and restrictions for that session key. The length of the control vector may vary.

The control vector is cryptographically coupled with the key at the time of key generation at the KDC. The coupling and decoupling processes are illustrated in Figure 5.12. As a first step, the control vector is passed through a hash function that produces a value whose length is equal to the encryption key length. Hash functions are discussed in detail in Chapter 8. In essence, a hash function maps values from a larger range into a smaller range, with a reasonably uniform spread. Thus, for example, if numbers in the range 1 to 100 are hashed into numbers in the range 1 to 10, approximately 10% of the source values should map into each of the target values.

The hash value is then XORed with the master key to produce an output that is used as the key input for encrypting the session key. Thus,

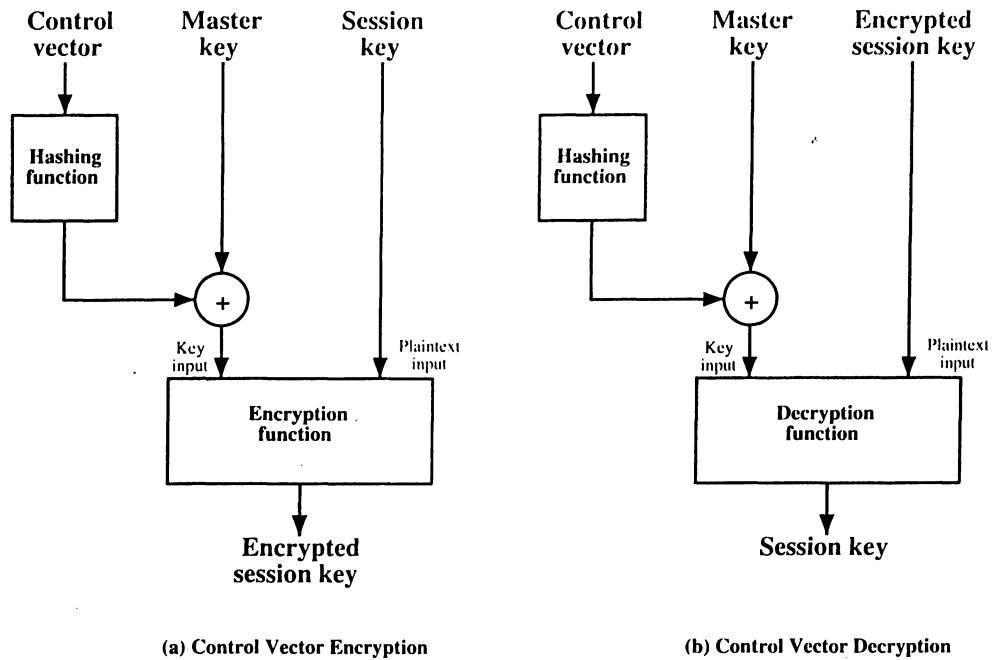
$$\text{Hash value} = H = h(\text{CV})$$

$$\text{Key input} = K_m \oplus H$$

$$\text{Ciphertext} = E_{K_m \oplus H}[K_s]$$

where  $K_m$  is the master key and  $K_s$  is the session key. The session key is recovered in plaintext by the reverse operation:

$$K_s = D_{K_m \oplus H}[E_{K_m \oplus H}[K_s]]$$



**Figure 5.12** Control Vector Encryption and Decryption.

When a session key is delivered to a user from the KDC, it is accompanied by the control vector in clear form. The session key can be recovered only by using both the master key that the user shares with the KDC and the control vector. Thus, the linkage between the session key and its control vector is maintained.

Use of the control vector has two advantages over use of an 8-bit tag. First, there is no restriction on length of the control vector, which enables arbitrarily complex controls to be imposed on key use. Second, the control vector is available in clear form at all stages of operation. Thus, control of key use can be exercised in multiple locations.

## 5.4 RANDOM NUMBER GENERATION

Random numbers play an important role in the use of encryption for various network security applications. In this section, we provide a brief overview of the use of random numbers in network security and then look at some approaches to generating random numbers.

### The Use of Random Numbers

A number of network security algorithms based on cryptography make use of random numbers. For example,

- Reciprocal authentication schemes, such as illustrated in Figures 5.9 and 5.11. In both of these key distribution scenarios, nonces are used for handshaking

to prevent replay attacks. The use of random numbers for the nonces frustrates opponents' efforts to determine or guess the nonce.

- Session key generation, whether done by a key distribution center or by one of the principals.
- Generation of keys for the RSA public-key encryption algorithm (described in Chapter 6).

These applications give rise to two distinct and not necessarily compatible requirements for a sequence of random numbers: randomness and unpredictability.

### Randomness

Traditionally, the concern in the generation of a sequence of allegedly random numbers has been that the sequence of numbers be random in some well-defined statistical sense. The following two criteria are used to validate that a sequence of numbers is random:

- **Uniform distribution:** The distribution of numbers in the sequence should be uniform; that is, the frequency of occurrence of each of the numbers should be approximately the same.
- **Independence:** No one value in the sequence can be inferred from the others.

Although there are well-defined tests for determining that a sequence of numbers matches a particular distribution, such as the uniform distribution, there is no such test to "prove" independence. Rather, a number of tests can be applied to demonstrate that a sequence does not exhibit independence. The general strategy is to apply a number of such tests until the confidence that independence exists is sufficiently strong.

In the context of our discussion, the use of a sequence of numbers that appear statistically random often occurs in the design of algorithms related to cryptography. For example, a fundamental requirement of the RSA public-key encryption scheme discussed in Chapter 6 is the ability to generate prime numbers. In general, it is difficult to determine if a given large number  $N$  is prime. A brute-force approach would be to divide  $N$  by every integer less than  $\sqrt{N}$ . If  $N$  is on the order, say, of  $10^{150}$ , a not uncommon occurrence in public-key cryptography, such a brute-force approach is beyond the reach of human analysts and their computers. However, a number of effective algorithms exist that test the primality of a number by using a sequence of randomly chosen integers as input to relatively simple computations. If the sequence is sufficiently long (but far, far less than  $\sqrt{10^{150}}$ ), the primality of a number can be determined with near certainty. This type of approach, known as randomization, crops up frequently in the design of algorithms. In essence, if a problem is too hard or time-consuming to solve exactly, a simpler, shorter approach based on randomization is used to provide an answer with any desired level of confidence.

### Unpredictability

In applications such as reciprocal authentication and session key generation, the requirement is not so much that the sequence of numbers be statistically ran-

dom but that the successive members of the sequence are unpredictable. With “true” random sequences, each number is statistically independent of other numbers in the sequence and therefore unpredictable. However, as is discussed shortly, true random numbers are rarely used; rather, sequences of numbers that appear to be random are generated by some algorithm. In this latter case, care must be taken that an opponent not be able to predict future elements of the sequence on the basis of earlier elements.

### Sources of Random Numbers

Sources of true random numbers are hard to come by. Physical noise generators, such as pulse detectors of ionizing radiation events, gas discharge tubes, and leaky capacitors, are one potential source. However, such devices are of limited utility in network security applications. There are problems both with the randomness and the precision of such numbers [BRIG79], to say nothing of the clumsy requirement of attaching one of these devices to every system in an internetwork. Another alternative is to dip into a collection of good-quality random numbers that have been published (e.g., [RAND55], [TIPP27]). However, these collections provide a very limited source of numbers compared to the potential requirements of a sizable network security application. Furthermore, although the numbers in these books do indeed exhibit statistical randomness, they are predictable, because an opponent who knows that the book is in use can obtain a copy.

Thus, cryptographic applications typically make use of algorithmic techniques for random number generation. These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random. However, if the algorithm is good, the resulting sequences will pass many reasonable tests of randomness. Such numbers are often referred to as pseudorandom numbers.

You may be somewhat uneasy about the concept of using numbers generated by a deterministic algorithm as if they were random numbers. Despite what might be called philosophical objections to such a practice, it generally works. As one expert on probability theory puts it [HAMM91],

For practical purposes we are forced to accept the awkward concept of “relatively random” meaning that with regard to the proposed use we can see no reason why they will not perform as if they were random (as the theory usually requires). This is highly subjective and is not very palatable to purists, but it is what statisticians regularly appeal to when they take “a random sample”—they hope that any results they use will have approximately the same properties as a complete counting of the whole sample space that occurs in their theory.

### Pseudorandom Number Generators

By far, the most widely used technique for pseudorandom number generation is an algorithm first proposed by Lehmer [LEHM51], which is known as the linear congruential method. The algorithm is parameterized with four numbers, as follows:

$m$	the modulus	$m > 0$
$a$	the multiplier	$0 \leq a < m$
$c$	the increment	$0 \leq c < m$
$X_0$	the starting value, or seed	$0 \leq X_0 < m$

The sequence of random numbers  $\{X_n\}$  is obtained via the following iterative equation:

$$X_{n+1} = (aX_n + c) \bmod m$$

If  $m$ ,  $a$ ,  $c$ , and  $X_0$  are integers, then this technique will produce a sequence of integers with each integer in the range  $0 \leq X_n < m$ .

The selection of values for  $a$ ,  $c$ , and  $m$  is critical in developing a good random number generator. For example, consider  $a = c = 1$ . The sequence produced is obviously not satisfactory. Now consider the values  $a = 7$ ,  $c = 0$ ,  $m = 32$ , and  $X_0 = 1$ . This generates the sequence  $\{7, 17, 23, 1, 7, \text{etc.}\}$ , which is also clearly unsatisfactory. Of the 32 possible values, only 4 are used; thus, the sequence is said to have a period of 4. If, instead, we change the value of  $a$  to 5, then the sequence is  $\{1, 5, 25, 29, 17, 21, 9, 13, 1, \text{etc.}\}$ , which increases the period to 8.

We would like  $m$  to be very large, so that there is the potential for producing a long series of distinct random numbers. A common criterion is that  $m$  be nearly equal to the maximum representable nonnegative integer for a given computer. Thus, a value of  $m$  near to or equal to  $2^{31}$  is typically chosen.

[PARK88] proposes three criteria to be used in evaluating a random number generator:

- T<sub>1</sub>: The function should be a full-period generating function. That is, the function should generate all the numbers between 0 and  $m$  before repeating.
- T<sub>2</sub>: The generated sequence should appear random. The sequence is not random because it is generated deterministically, but there is a variety of statistical tests that can be used to assess the degree to which a sequence exhibits randomness.
- T<sub>3</sub>: The function should implement efficiently with 32-bit arithmetic.

With appropriate values of  $a$ ,  $c$ , and  $m$ , these three tests can be passed. With respect to T<sub>1</sub>, it can be shown that if  $m$  is prime and  $c = 0$ , then for certain values of  $a$ , the period of the generating function is  $m - 1$ , with only the value 0 missing. For 32-bit arithmetic, a convenient prime value of  $m$  is  $2^{31} - 1$ . Thus, the generating function becomes:

$$X_{n+1} = (aX_n) \bmod (2^{31} - 1)$$

Of the more than 2 billion possible choices for  $a$ , only a handful of multipliers pass all three tests. One such value is  $a = 7^5 = 16807$ , which was originally designed for use in the IBM 360 family of computers [LEW169]. This generator is widely used and has been subjected to a more thorough testing than any other pseudorandom number generator. It is frequently recommended for statistical and simulation work (e.g., [JAIN91], [SAUE81]).

The strength of the linear congruential algorithm is that if the multiplier and modulus are properly chosen, the resulting sequence of numbers will be statistically indistinguishable from a sequence drawn at random (but without replacement) from the set  $1, 2, \dots, m-1$ . But there is nothing random at all about the algorithm, apart from the choice of the initial value  $X_0$ . Once that value is chosen, the remaining numbers in the sequence follow deterministically. This has implications for cryptanalysis.



If an opponent knows that the linear congruential algorithm is being used and if the parameters are known (e.g.,  $a = 7^5$ ,  $c = 0$ ,  $m = 2^{31} - 1$ ), then once a single number is discovered, all subsequent numbers are known. Even if the opponent knows only that a linear congruential algorithm is being used, knowledge of a small part of the sequence is sufficient to determine the parameters of the algorithm. Suppose that the opponent is able to determine values for  $X_0$ ,  $X_1$ ,  $X_2$ , and  $X_3$ . Then

$$X_1 = (aX_0 + c) \bmod m$$

$$X_2 = (aX_1 + c) \bmod m$$

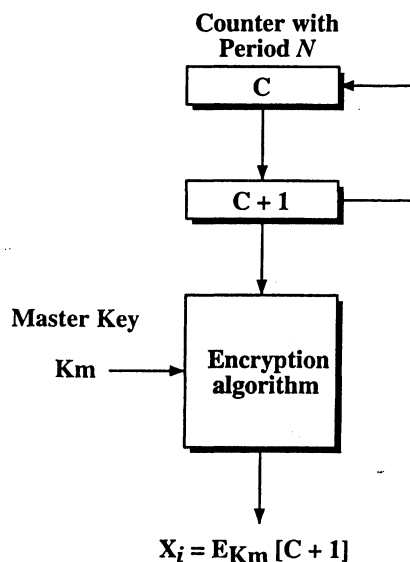
$$X_3 = (aX_2 + c) \bmod m$$

These equations can be solved for  $a$ ,  $c$ , and  $m$ .

Thus, although it is nice to be able to use a good pseudorandom number generator, it is desirable to make the actual sequence used nonreproducible, so that knowledge of part of the sequence on the part of an opponent is insufficient to determine future elements of the sequence. This goal can be achieved in a number of ways. For example, [BRIG79] suggests using an internal system clock to modify the random number stream. One way to use the clock would be to restart the sequence after every  $N$  numbers using the current clock value (mod  $m$ ) as the new seed. Another way would be simply to add the current clock value to each random number (mod  $m$ ).

### Cryptographically Generated Random Numbers

For cryptographic applications, it makes some sense to take advantage of the encryption logic available to produce random numbers. A number of means have been used, and in this subsection we look at three representative examples.



**Figure 5.13** Pseudorandom Number Generation from a Counter.

### Cyclic Encryption

Figure 5.13 illustrates an approach suggested in [MEYE82]. In this case, the procedure is used to generate session keys from a master key. A counter with period  $N$  provides input to the encryption logic. For example, if 56-bit DES keys are to be produced, then a counter with period  $2^{56}$  can be used. After each key is produced, the counter is incremented by one. Thus, the pseudorandom numbers produced by this scheme cycle through a full period: Each of the outputs  $X_0, X_1, \dots, X_{N-1}$  is based on a different counter value and therefore  $X_0 \neq X_1 \neq \dots \neq X_{N-1}$ . Because the master key is protected, it is not computationally feasible to deduce any of the secret keys through knowledge of one or more earlier keys.

To strengthen the algorithm further, the input could be the output of a full-period pseudorandom number generator rather than a simple counter.

### DES Output Feedback Mode

The output feedback (OFB) mode of DES, illustrated in Figure 3.14, can be used for key generation as well as for stream encryption. Notice that the output of each stage of operation is a 64-bit value, of which the  $j$  leftmost bits are fed back for encryption. Successive 64-bit outputs constitute a sequence of pseudorandom numbers with good statistical properties. Again, as with the approach suggested in the preceding subsection, the use of a protected master key protects the generated session keys.

### ANSI X9.17 Pseudorandom Number Generator

One of the strongest (cryptographically speaking) pseudorandom number generators is specified in ANSI X9.17. A number of applications employ this technique, including financial security applications and PGP (the latter described in Chapter 12).

Figure 5.14 illustrates the algorithm, which makes use of triple DES for encryption. The ingredients are as follows:

- **Input:** Two pseudorandom inputs drive the generator. One is a 64-bit representation of the current date and time, which is updated on each number gen-

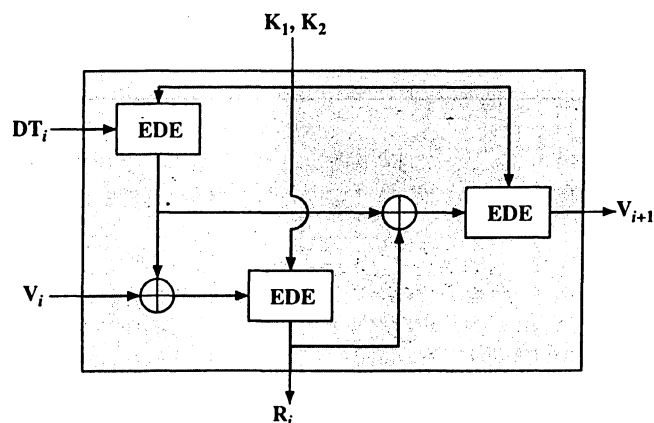


Figure 5.14 ANSI X9.17 Pseudorandom Number Generator.

eration. The other is a 64-bit seed value; this is initialized to some arbitrary value and is updated during the generation process.

- **Keys:** The generator makes use of three triple DES encryption modules. All three make use of the same pair of 56-bit keys, which must be kept secret and are used only for pseudorandom number generation.
- **Output:** The output consists of a 64-bit pseudorandom number and a 64-bit seed value.

Define the following quantities:

DT<sub>*i*</sub>: Date/time value at the beginning of *i*th generation stage  
 V<sub>*i*</sub>: Seed value at the beginning of *i*th generation stage  
 R<sub>*i*</sub>: Pseudorandom number produced by the *i*th generation stage  
 K<sub>1</sub>, K<sub>2</sub>: DES keys used for each stage

Then

$$R_i = \text{EDE}_{K_1, K_2}[V_i \oplus \text{EDE}_{K_1, K_2}[DT_i]]$$

$$V_{i+1} = \text{EDE}_{K_1, K_2}[R_i \oplus \text{EDE}_{K_1, K_2}[DT_i]]$$

where EDE refers to the sequence encrypt-decrypt-encrypt using two-key triple DES.

Several factors contribute to the cryptographic strength of this method. The technique involves a 112-bit key and three EDE encryptions for a total of nine DES encryptions. The scheme is driven by two pseudorandom inputs, the date and time value, and a seed produced by the generator that is distinct from the pseudorandom number produced by the generator. Thus, the amount of material that must be compromised by an opponent is overwhelming. Even if a pseudorandom number  $R_i$  were compromised, it would be impossible to deduce the  $V_{i+1}$  from the  $R_i$  because an additional EDE operation is used to produce the  $V_{i+1}$ .

### Blum Blum Shub Generator

A popular approach to generating secure pseudorandom number is known as the Blum, Blum, Shub (BBS) generator, named for its developers [BLUM86]. It has perhaps the strongest public proof of its cryptographic strength. The procedure is as follows. First, choose two large prime numbers,  $p$  and  $q$ , that both have a remainder of 3 when divided by 4. That is

$$p \equiv q \equiv 3 \pmod{4}$$

This notation, explained more fully in Chapter 7, simply means that  $(p \bmod 4) = (q \bmod 4) = 3$ . For example, the prime numbers 7 and 11 satisfy  $7 \equiv 11 \equiv 3 \pmod{4}$ . Let  $n = p \times q$ . Next, choose a random number  $s$ , such that  $s$  is relatively prime to  $n$ ; this is equivalent to saying that neither  $p$  nor  $q$  is a factor of  $s$ . Then the BBS generator produces a sequence of bits  $B_i$  according to the following algorithm:

$$\begin{aligned} X_0 &= s^2 \bmod n \\ \text{for } i &= 1 \text{ to } \infty \\ X_i &= (X_{i-1})^2 \bmod n \\ B_i &= X_i \bmod 2 \end{aligned}$$

**Table 5.2** Example Operation of BBS Generator

$s$	$X_i$	$B_i$
0	20749	
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0
9	186894	0
10	177046	0

$s$	$X_i$	$B_i$
11	137922	0
12	123175	1
13	8630	0
14	114386	0
15	14863	1
16	133015	1
17	106065	1
18	45870	0
19	137171	1
20	48060	0

Thus, the least significant bit is taken at each iteration. Table 5.2, taken from [STIN95], shows an example of BBS operation. Here,  $n = 192649 = 383 \times 503$  and the seed  $s = 101355$ .

The BBS is referred to as a **cryptographically secure pseudorandom bit generator** (CSPRNG). A CSPRNG is defined as one that passes the *next-bit test*, which in turn is defined as follows [MENE97]: “A pseudorandom bit generator is said to pass the next-bit test if there is not a polynomial-time algorithm that, on input of the first  $k$  bits of an output sequence,<sup>5</sup> can predict the  $(k + 1)$ <sup>st</sup> bit with probability significantly greater than  $1/2$ .” In other words, given the first  $k$  bits of the sequence, there is not a practical algorithm that can even allow you to state that the next bit will be 1 (or 0) with probability greater than  $1/2$ . For all practical purposes, the sequence is unpredictable. The security of BBS is based on the difficulty of factoring  $n$ . That is, given  $n$ , we need to determine its two prime factors  $p$  and  $q$ . For a discussion, see [STIN95].

## 5.5 RECOMMENDED READING

[FUMY93] is a good survey of key management principles. [MEYE82] contains a detailed description of a key distribution and key management facility. Perhaps the best treatment of pseudorandom number generators is found in [KNUT98]. An alternative to the standard linear congruential algorithm, known as the linear recurrence algorithm, is explained in some detail in [BRIG79]. [ZENG91] assesses various pseudorandom generation algorithms for use in generating variable-length keys for Vernam types of ciphers.

Good discussions of secure pseudorandom number generators are contained in [MENE97] and [STIN95]. Another good treatment, with an emphasis on practical implementation issues, is RFC 1750 [EAST94].

<sup>5</sup>A polynomial-time algorithm of order  $k$  is one whose running time is bounded by a polynomial of order  $k$ .

- BRIG79 Bright, H., and Enison, R. "Quasi-Random Number Sequences from Long-Period TLP Generator with Remarks on Application to Cryptography." *Computing Surveys*, December 1979.
- EAST94 Eastlake, D.; Crocker, S.; and Schiller, J. *Randomness Recommendations for Security*. RFC 1750, December 1994.
- FUMY93 Fumy, S., and Landrock, P. "Principles of Key Management." *IEEE Journal on Selected Areas in Communications*, June 1995.
- KNUT98 Knuth, D. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1998.
- MENE97 Menezes, A.; Oorschot, P.; and Vanstone, S. *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1997.
- MEYE82 Meyer, C., and Matyas, S. *Cryptography: A New Dimension in Computer Data Security*. New York: Wiley, 1982.
- STIN95 Stinson, D. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 1995.
- ZENG91 Zeng, K.; Yang, C.; Wei, D.; and Rao, T. "Pseudorandom Bit Generators in Stream-Cipher Cryptography." *Computer*, February 1991.

## 5.6 PROBLEMS

- 5.1 Electronic mail systems differ in the manner in which multiple recipients are handled. In some systems, the originating mail-handler makes all the necessary copies, and these are sent out independently. An alternative approach is to determine the route for each destination first. Then a single message is sent out on a common portion of the route, and copies are made only when the routes diverge; this process is referred to as *mail bagging*.
- Leaving aside considerations of security, discuss the relative advantages and disadvantages of the two methods.
  - Discuss the security requirements and implications of the two methods.
- 5.2 Section 5.2 describes the use of message length as a means of constructing a covert channel. Describe three additional schemes for using traffic patterns to construct a covert channel.
- 5.3 One local area network vendor provides a key distribution facility, as illustrated in Figure 5.15.
- Describe the scheme.
  - Compare this scheme to that of Figure 5.9. What are the pros and cons?
- 5.4 If we take the linear congruential algorithm with an additive component of 0,

$$X_{n+1} = (aX_n) \bmod m$$

then it can be shown that if  $m$  is prime, and if a given value of  $a$  produces the maximum period of  $m - 1$ , then  $a^k$  will also produce the maximum period, provided that  $k$  is less than  $m$  and that  $m - 1$  is not divisible by  $k$ . Demonstrate this by using  $X_0 = 1$  and  $m = 31$  and producing the sequences for  $a = 3, 3^2, 3^3$ , and  $3^4$ .

- 5.5 a. What is the maximum period obtainable from the following generator?

$$X_{n+1} = (aX_n) \bmod 2^4$$

- What should be the value of  $a$ ?
- What restrictions are required on the seed?

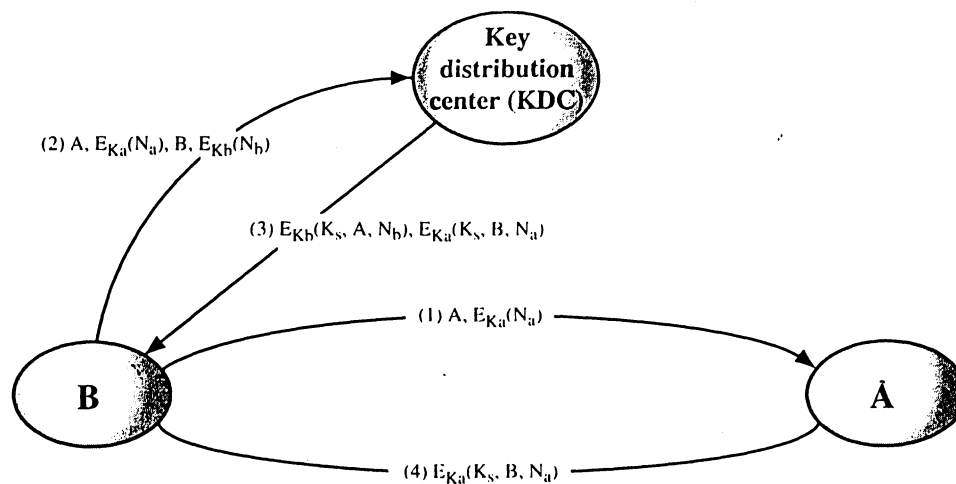


Figure 5.15 Figure for Problem 5.3.

- 5.6 You may wonder why the modulus  $m = 2^{31} - 1$  was chosen for the linear congruential method instead of simply  $2^{31}$ , because this latter number can be represented with no additional bits and the mod operation should be easier to perform. In general, the modulus  $2^k - 1$  is preferable to  $2^k$ . Why is this so?
- 5.7 With the linear congruential algorithm, a choice of parameters that provides a full period does not necessarily provide a good randomization. For example, consider the following two generators:

$$X_{n+1} = (6X_n) \bmod 13$$

$$X_{n+1} = (7X_n) \bmod 13$$

Write out the two sequences to show that both are full period. Which one appears more random to you?

- 5.8 In any use of pseudorandom numbers, whether for encryption, simulation, or statistical design, it is dangerous to trust blindly the random number generator that happens to be available in your computer's system library. [PARK88] found that many contemporary textbooks and programming packages make use of flawed algorithms for pseudorandom number generation. This exercise will enable you to test your system.

The test is based on a theorem attributed to Ernesto Cesaro (see [KNUT98] for a proof), which states that the probability that the greatest common divisor of two randomly chosen integers is 1 is equal to  $6/\pi^2$ . Use this theorem in a program to determine statistically the value of  $\pi$ . The main program should call three subprograms: the random number generator from the system library to generate the random integers; a subprogram to calculate the greatest common divisor of two integers using Euclid's algorithm; and a subprogram that calculates square roots. If these latter two programs are not available, you will have to write them as well. The main program should loop through a large number of random numbers to give an estimate of the aforementioned probability. From this, it is a simple matter to solve for your estimate of  $\pi$ .

If the result is close to 5.14, congratulations! If not, then the result is probably low, usually a value of around 2.7. Why would such an inferior result be obtained?

- 5.9 Suppose that someone suggests the following way to confirm that the two of you are both in possession of the same secret key. You create a random bit string the length of the key, XOR it with the key, and send the result over the channel. Your partner XORs the incoming block with the key (which should be the same as your key) and sends it

back. You check, and if what you receive is your original random string, you have verified that your partner has the same secret key, yet neither of you has ever transmitted the key. Is there a flaw in this scheme?

- 5.10 “We are under great pressure, Holmes.” Detective Lestrade looked nervous. “We have learned that copies of sensitive governmental documents are stored in computers of one foreign embassy here in London. Normally these documents exist in electronic form only on a selected few governmental computers that satisfy the most stringent security requirements. However, sometimes they must be sent through the network connecting all governmental computers. But all messages in this network are encrypted using top secret encryption algorithm certified by our best crypto experts. Even NSA and the KGB are unable to break it. And now these documents have appeared in hands of diplomats of a small, otherwise insignificant, country. And we have no idea how it could happen.”

“But you do have some suspicion who did it, do you?” asked Holmes.

“Yes, we did some routine investigation. There is a man who has legal access to one of the governmental computers and has frequent contacts with diplomats from the embassy. But the computer he has access to is not one of the trusted ones where these documents are normally stored. He is the suspect, but we have no idea how he could obtain copies of the documents. Even if he could obtain a copy of an encrypted document, he couldn’t decrypt it.”

“Hmm, please describe the communication protocol used on the network.” Holmes opened his eyes, thus proving that he had followed Lestrade’s talk with an attention that contrasted with his sleepy look.

“Well, the protocol is as follows. Each node of the network has been assigned a unique secret key  $K_n$ . This key is used to secure communication between the node and a trusted server. That is, all the keys are stored also on the server. User A, wishing to send a secret message  $M$  to user B, initiates the following protocol:

1. A generates a random number  $R$  and sends to the server his name A, destination B, and  $E_{K_n}[R]$ .
2. Server responds by sending  $E_{K_b}[R]$  to A.
3. A sends  $E_R[M]$  together with  $E_{K_b}[R]$  to B.
4. B knows  $K_b$ , thus decrypts  $E_{K_b}[R]$  to get  $R$ , and will subsequently use  $R$  to decrypt  $E_R[M]$  to get  $M$ .

You see that a random key is generated every time a message has to be sent. I admit the man could intercept messages sent between the top secret trusted nodes, but I see no way he could decrypt them.”

“Well, I think you have your man, Lestrade. The protocol isn’t secure because the server doesn’t authenticate users who send him a request. Apparently designers of the protocol have believed that sending  $E_{K_n}[R]$  implicitly authenticates user X as the sender, as only X (and the server) knows  $K_x$ . But you know that  $E_{K_n}[R]$  can be intercepted and later replayed. Once you understand where the hole is, you will be able to obtain enough evidence by monitoring the man’s use of the computer he has access to. Most likely he works as follows. After intercepting  $E_{K_n}[R]$  and  $E_R[M]$  (see steps 1 and 3 of the protocol), the man, let’s denote him as Z, will continue by pretending to be A and ...”





**PART  
TWO**

# Public-Key Encryption and Hash Functions





## CHAPTER 6

# PUBLIC-KEY CRYPTOGRAPHY

*Every Egyptian received two names, which were known respectively as the true name and the good name, or the great name and the little name; and while the good or little name was made public, the true or great name appears to have been carefully concealed.*

— *The Golden Bough*, Sir James George Frazer

**T**he development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography. From its earliest beginnings to modern times, virtually all cryptographic systems have been based on the elementary tools of substitution and permutation. After millennia of working with algorithms that could essentially be calculated by hand, a major advance in conventional cryptography occurred with the development of the rotor encryption/decryption machine. The electromechanical rotor enabled the development of fiendishly complex cipher systems. With the availability of computers, even more complex systems were devised, the most prominent of which was the Lucifer effort at IBM that culminated in the Data Encryption Standard (DES). But both rotor machines and DES, although representing significant advances, still relied on the bread-and-butter tools of substitution and permutation.

Public-key cryptography provides a radical departure from all that has gone before. For one thing, public-key algorithms are based on mathematical functions rather than on substitution and permutation. More important, public-key cryptography is asymmetric involving the use of two separate keys, in contrast to symmetric conventional encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication, as we shall see.

Before proceeding, we should mention several common misconceptions concerning public-key encryption. One such misconception is that public-key encryption is more secure from cryptanalysis than is conventional encryption. Such a claim was made, for example, in a famous article in *Scientific American* by Gardner [GARD77]. In fact, the security of any encryption scheme depends on the length of the key and the computational work involved in breaking a cipher. There is nothing in principle about either conventional or public-key encryption that makes one superior to another from the point of view of resisting cryptanalysis.

A second misconception is that public-key encryption is a general-purpose technique that has made conventional encryption obsolete. On the contrary, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that conventional encryption will be abandoned. As one of the inventors of public-key encryption has put it [DIFF88], “the restriction of public-key cryptography to key management and signature applications is almost universally accepted.”

Finally, there is a feeling that key distribution is trivial when using public-key encryption, compared to the rather cumbersome handshaking involved with key distribution centers for conventional encryption. In fact, some form of protocol is needed, generally involving a central agent, and the procedures involved are no simpler nor any more efficient than those required for conventional encryption (e.g., see analysis in [NEED78]).

This chapter provides an overview of public-key encryption. First, we look at its conceptual framework. Interestingly, the concept for this technique was developed and published before it was shown to be practical to adopt it. Next, we examine the RSA algorithm, which is the most important encryption/decryption algorithm that has been shown to be feasible for public-key encryption. Then we examine key distribution and management for public-key systems, including a discussion of Diffie-Hellman key exchange. Finally, we provide an introduction to elliptic curve cryptography.

Much of the theory of public-key cryptosystems is based on number theory. If one is prepared to accept the results given in this chapter, an understanding of number theory is not strictly necessary. However, to gain a full appreciation of public-key algorithms, some understanding of number theory is required. Chapter 7 provides an overview.

## 6.1 PRINCIPLES OF PUBLIC-KEY CRYPTOSYSTEMS

The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with conventional encryption. The first problem is that of key distribution, which was examined in some detail in Chapter 5.

As we have seen, key distribution under conventional encryption requires that two communicants either (1) already share a key, which somehow has been distributed to them; or (2) have the use of a key distribution center. Whitfield Diffie, one of the discoverers of public-key encryption (along with Martin Hellman, both at Stanford University at the time), reasoned that this second requirement negated the

very essence of cryptography: the ability to maintain total secrecy over your own communication. As Diffie put it [DIFF88], “What good would it do after all to develop impenetrable cryptosystems, if their users were forced to share their keys with a KDC that could be compromised by either burglary or subpoena?”

The second problem that Diffie pondered, and one that was apparently unrelated to the first, was that of “digital signatures.” If the use of cryptography was to become widespread, not just in military situations but for commercial and private purposes, then electronic messages and documents would need the equivalent of signatures used in paper documents. That is, could a method be devised that would stipulate, to the satisfaction of all parties, that a digital message had been sent by a particular person? This is a somewhat broader requirement than that of authentication, and its characteristics and ramifications are explored in Chapter 10.

Diffie and Hellman achieved an astounding breakthrough in 1976 [DIFF76a, b] by coming up with a method that addressed both problems and that was radically different from all previous approaches to cryptography, going back over four millennia.<sup>1</sup>

In the next subsection, we look at the overall framework for public-key cryptography. Then we examine the requirements for the encryption/decryption algorithm that is at the heart of the scheme.

### Public-Key Cryptosystems

Public-key algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic:

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristic:

- Either of the two related keys can be used for encryption, with the other used for decryption.

Figure 6.1a illustrates the public-key encryption process (compare with Figure 2.1). The essential steps are the following:

1. Each end system in a network generates a pair of keys to be used for encryption and decryption of messages that it will receive.
2. Each system publishes its encryption key by placing it in a public register or file. This is the public key. The companion key is kept private.
3. If A wishes to send a message to B, it encrypts the message using B's public key.
4. When B receives the message, B decrypts it using B's private key. No other recipient can decrypt the message because only B knows B's private key.

---

<sup>1</sup>Diffie and Hellman first *publicly* introduced the concepts of public-key cryptography in 1976. However, this is not the true beginning. Admiral Bobby Inman, while director of the NSA, claimed that public-key cryptography had been discovered at NSA in the mid-1960s [SIMM93]. The first *documented* introduction of these concepts came in 1970, from the Communications-Electronics Security Group, Britain's counterpart to NSA, in a classified report by James Ellis [ELLI70]. Ellis referred to the technique as non-secret encryption and described the discovery in the unclassified [ELLI87].

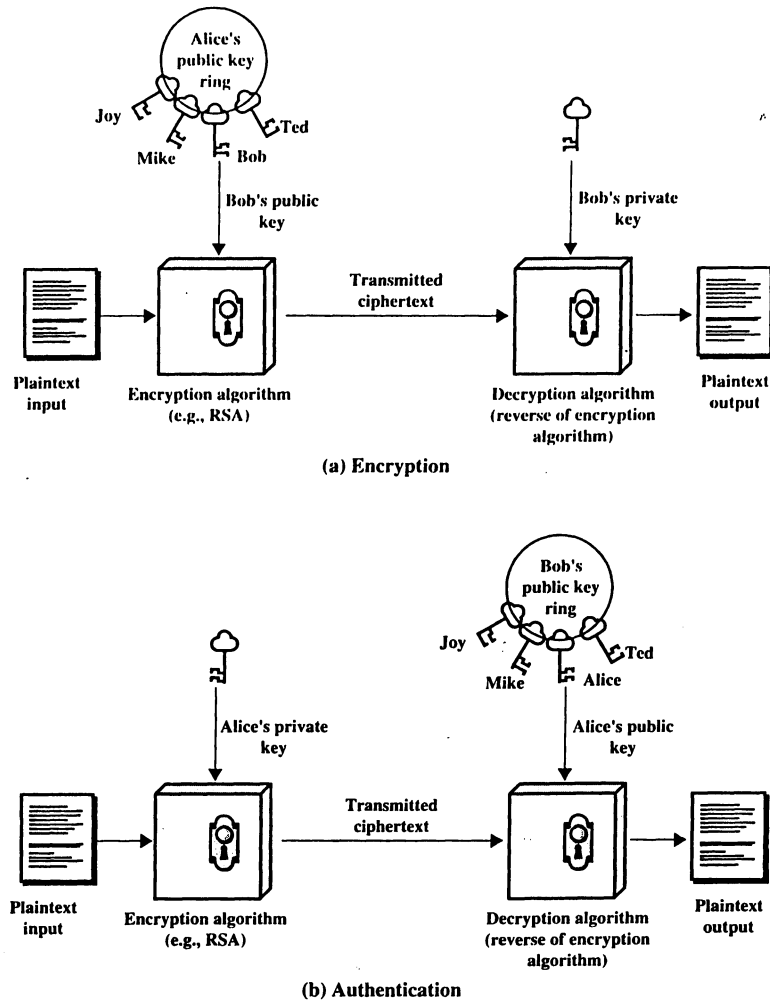


Figure 6.1 Public-Key Encryption.

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a system controls its private key, its incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

Table 6.1 summarizes some of the important aspects of conventional and public-key encryption. To discriminate between the two, we will generally refer to the key used in conventional encryption as a **secret key**. The two keys used for public-key encryption are referred to as the **public key** and the **private key**.<sup>2</sup> Invariably, the

<sup>2</sup>The following notation is used consistently throughout. A secret key is represented by  $K_m$ , where  $m$  is some modifier; for example,  $K_s$  is a session key. A public key is represented by  $KU_a$ , for user  $A$ , and the corresponding private key is  $KR_a$ . Encryption of plaintext  $P$  can be performed with a secret key, a public key, or a private key, denoted by  $E_{K_m}[P]$ ,  $E_{KU_a}[P]$ , and  $E_{KR_a}[P]$ , respectively. Similarly, decryption of

**Table 6.1** Conventional and Public-Key Encryption**Conventional Encryption***Needed to Work:*

1. The same algorithm with the same key is used for encryption and decryption.
2. The sender and receiver must share the algorithm and the key.

*Needed for Security:*

1. The key must be kept secret.
2. It must be impossible or at least impractical to decipher a message if no other information is available.
3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.

**Public-Key Encryption***Needed to Work:*

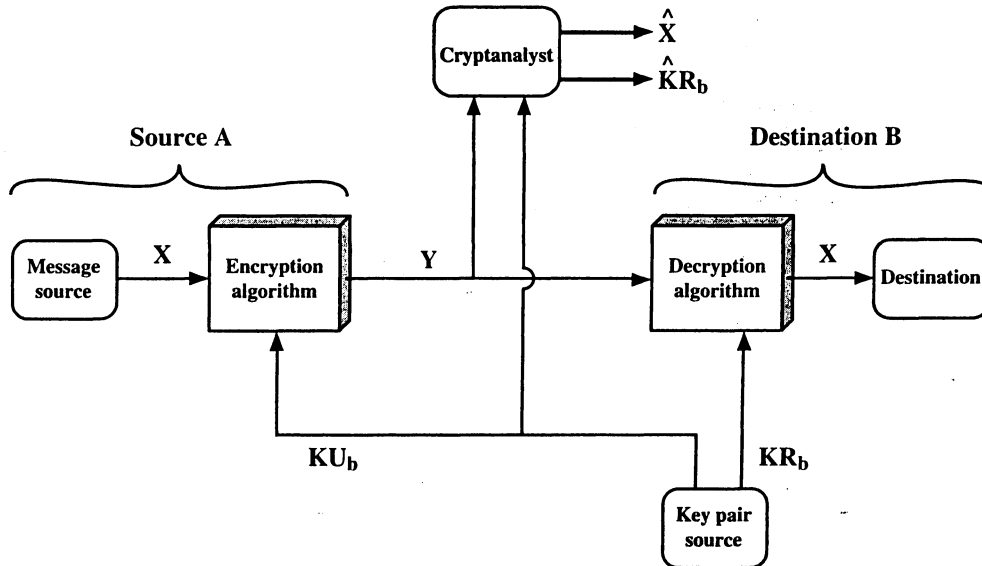
1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption.
2. The sender and receiver must each have one of the matched pair of keys (not the same one).

*Needed for Security:*

1. One of the two keys must be kept secret.
2. It must be impossible or at least impractical to decipher a message if no other information is available.
3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with conventional encryption.

Let us take a closer look at the essential elements of a public-key encryption scheme, using Figure 6.2 (compare with Figure 2.2). There is some source A for a message, which produces a message in plaintext,  $X = [X_1, X_2, \dots, X_M]$ . The  $M$  elements of  $X$  are letters in some finite alphabet. The message is intended for desti-

**Figure 6.2** Public-Key Cryptosystem: Secrecy.

ciphertext  $C$  can be performed with a secret key, a public key, or a private key, denoted by  $D_{K_m}[C]$ ,  $D_{K_{ub}}[C]$ , and  $D_{K_{rb}}[C]$ , respectively.

nation B. B generates a related pair of keys: a public key,  $KU_b$ , and a private key,  $KR_b$ .  $KR_b$  is known only to B, whereas  $KU_b$  is publicly available and therefore accessible by A.

With the message  $X$  and the encryption key  $KU_b$  as input, A forms the ciphertext  $Y = [Y_1, Y_2, \dots, Y_M]$ :

$$Y = E_{KU_b}(X)$$

The intended receiver, in possession of the matching private key, is able to invert the transformation:

$$X = D_{KR_b}(Y)$$

An opponent, observing  $Y$  and having access to  $KU_b$  but not having access to  $KR_b$  or  $X$ , must attempt to recover  $X$  and/or  $KR_b$ . It is assumed that the opponent does have knowledge of the encryption ( $E$ ) and decryption ( $D$ ) algorithms. If the opponent is interested only in this particular message, then the focus of effort is to recover  $X$ , by generating a plaintext estimate  $\hat{X}$ . Often, however, the opponent is interested in being able to read future messages as well, in which case an attempt is made to recover  $KR_b$  by generating an estimate  $\hat{K}R_b$ .

We mentioned earlier that either of the two related keys can be used for encryption, with the other being used for decryption. This enables a rather different cryptographic scheme to be implemented. Whereas the scheme illustrated in Figure 6.2 provides confidentiality, Figures 6.1b and 6.3 show the use of public-key encryption to provide authentication:

$$Y = E_{KR_a}(X)$$

$$X = D_{KU_a}(Y)$$

In this case, A prepares a message to B and encrypts it using A's private key before transmitting it. B can decrypt the message using A's public key. Because the message was encrypted using A's private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a *digital signature*. In addition, it is impossible to alter the message without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity.

In the preceding scheme, the entire message is encrypted, which, although validating both author and contents, requires a great deal of storage. Each document must be kept in plaintext to be used for practical purposes. A copy also must be stored in ciphertext so that the origin and contents can be verified in case of a dispute. A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted with the sender's private key, it serves as a signature that verifies origin, content, and sequencing. Chapter 10 examines this technique in detail.

It is important to emphasize that the encryption process just described does not provide confidentiality. That is, the message being sent is safe from alteration but not from eavesdropping. This is obvious in the case of a signature based on a



portion of the message, because the rest of the message is transmitted in the clear. Even in the case of complete encryption, as shown in Figure 6.3, there is no protection of confidentiality because any observer can decrypt the message by using the sender's public key.

It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme (Figure 6.4):

$$Z = E_{KU_b}[E_{KR_a}(X)]$$

$$X = D_{KU_a}[D_{KR_b}(Z)]$$

In this case, we begin as before by encrypting a message, using the sender's private key. This provides the digital signature. Next, we encrypt again, using the receiver's public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided. The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

### Applications for Public-Key Cryptosystems

Before proceeding, we need to clarify one aspect of public-key cryptosystems that is otherwise likely to lead to confusion. Public-key systems are characterized by the use of a cryptographic type of algorithm with two keys, one held private and one available publicly. Depending on the application, the sender uses either the sender's private key or the receiver's public key, or both, to perform some type of crypto-

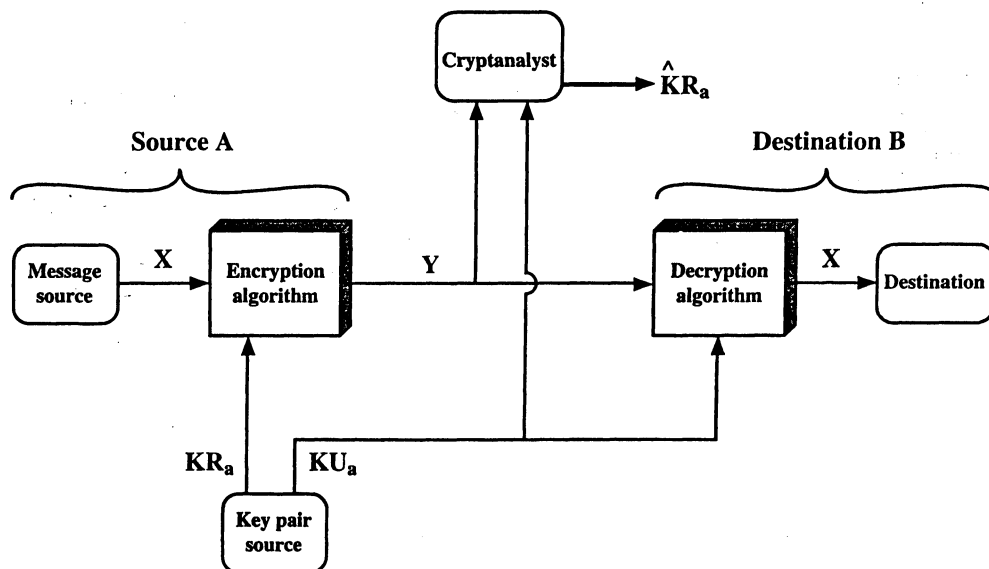


Figure 6.3 Public-Key Cryptosystem: Authentication.

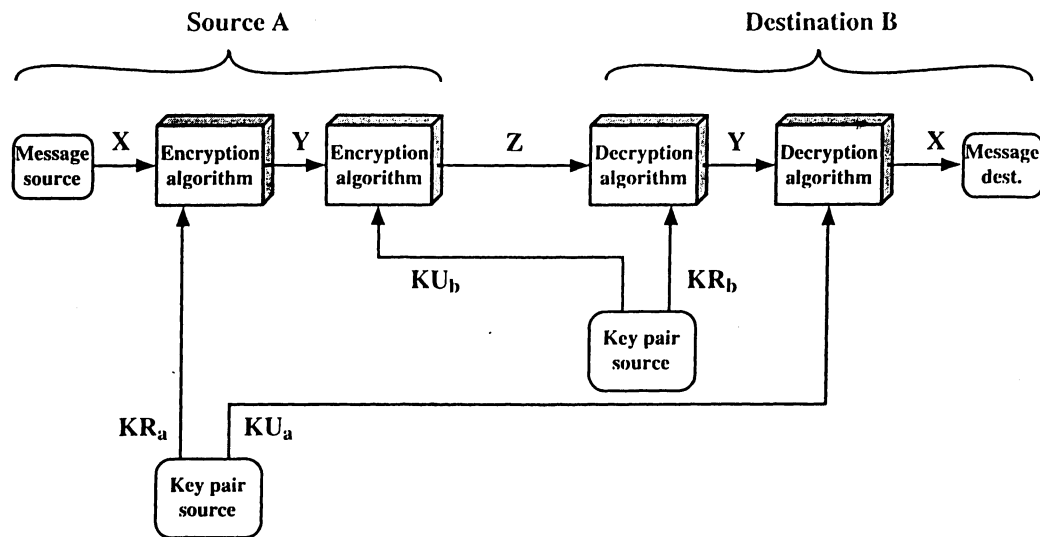


Figure 6.4 Public-Key Cryptosystem: Secrecy and Authentication.

graphic function. In broad terms, we can classify the use of public-key cryptosystems into three categories:

- **Encryption/decryption:** The sender encrypts a message with the recipient's public key.
- **Digital signature:** The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.
- **Key exchange:** Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. Table 6.2 indicates the applications supported by the algorithms discussed in this book.

### Requirements for Public-Key Cryptography

The cryptosystem illustrated in Figures 6.2 through 6.4 depends on a cryptographic algorithm based on two related keys. Diffie and Hellman postulated this system

Table 6.2 Applications for Public-Key Cryptosystems

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

without demonstrating that such algorithms exist. However, they did lay out the conditions that such algorithms must fulfill [DIFF76b]:

1. It is computationally easy for a party B to generate a pair (public key  $KU_b$ , private key  $KR_b$ ).
2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted,  $M$ , to generate the corresponding ciphertext:

$$C = E_{KU_b}(M)$$

3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

$$M = D_{KR_b}(C) = D_{KR_b}[E_{KU_b}(M)]$$

4. It is computationally infeasible for an opponent, knowing the public key,  $KU_b$ , to determine the private key,  $KR_b$ .
5. It is computationally infeasible for an opponent, knowing the public key,  $KU_b$ , and a ciphertext,  $C$ , to recover the original message,  $M$ .

We can add a sixth requirement that, although useful, is not necessary for all public-key applications:

6. The encryption and decryption functions can be applied in either order:

$$M = E_{KU_b}[D_{KR_b}(M)]$$

These are formidable requirements, as evidenced by the fact that only one such algorithm has received widespread acceptance in the several decades since the concept of public-key cryptography was proposed.

Before elaborating on why the requirements are so formidable, let us first recast them. The requirements boil down to the need for a trap-door one-way function. A *one-way function*<sup>3</sup> is one that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy whereas the calculation of the inverse is infeasible:

$$\begin{array}{ll} Y = f(X) & \text{easy} \\ X = f^{-1}(Y) & \text{infeasible} \end{array}$$

Generally, *easy* is defined to mean a problem that can be solved in polynomial time as a function of input length. Thus, if the length of the input is  $n$  bits, then the time to compute the function is proportional to  $n^a$ , where  $a$  is a fixed constant. Such algorithms are said to belong to the class **P**. The term *infeasible* is a much

---

<sup>3</sup>Not to be confused with a one-way hash function, which takes an arbitrarily large data field as its argument and maps it to a fixed output. Such functions are used for authentication (see Chapter 8).

fuzzier concept. In general, we can say a problem is infeasible if the effort to solve it grows faster than polynomial time as a function of input size. For example, if the length of the input is  $n$  bits and the time to compute the function is proportional to  $2^n$ , the problem is considered infeasible. Unfortunately, it is difficult to determine if a particular algorithm exhibits this complexity. Furthermore, traditional notions of computational complexity focus on the worst-case or average-case complexity of an algorithm. These measures are worthless for cryptography, which requires that it be infeasible to invert a function for virtually all inputs, not for the worst case or even average case. A brief introduction to some of these concepts is provided in Appendix 6A.

We now turn to the definition of a *trap-door one-way function*, which is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known. With the additional information the inverse can be calculated in polynomial time. We can summarize as follows: A trap-door one-way function is a family of invertible functions  $f_k$ , such that

$$\begin{array}{ll} Y = f_k(X) & \text{easy, if } k \text{ and } X \text{ are known} \\ X = f_k^{-1}(Y) & \text{easy, if } k \text{ and } Y \text{ are known} \\ X = f_k^{-1}(Y) & \text{infeasible, if } Y \text{ is known but } k \text{ is not known} \end{array}$$

Thus, the development of a practical public-key scheme depends on discovery of a suitable trap-door one-way function.

### Public-Key Cryptanalysis

As with conventional encryption, a public-key encryption scheme is vulnerable to a brute-force attack. The countermeasure is the same: Use large keys. However, there is a tradeoff to be considered. Public-key systems depend on the use of some sort of invertible mathematical function. The complexity of calculating these functions may not scale linearly with the number of bits in the key but grow more rapidly than that. Thus, the key size must be large enough to make brute-force attack impractical but small enough for practical encryption and decryption. In practice, the key sizes that have been proposed do make brute-force attack impractical but result in encryption/decryption speeds that are too slow for general-purpose use. Instead, as was mentioned earlier, public-key encryption is currently confined to key management and signature applications.

Another form of attack is to find some way to compute the private key given the public key. To date, it has not been mathematically proven that this form of attack is infeasible for a particular public-key algorithm. Thus, any given algorithm, including the widely used RSA algorithm, is suspect. The history of cryptanalysis shows that a problem that seems insoluble from one perspective can be found to have a solution if looked at in an entirely different way.

Finally, there is a form of attack that is peculiar to public-key systems. This is, in essence, a probable-message attack. Suppose, for example, that a message were to be sent that consisted solely of a 56-bit DES key. An opponent could encrypt all possible keys using the public key and could decipher any message by matching the

transmitted ciphertext. Thus, no matter how large the key size of the public-key scheme, the attack is reduced to a brute-force attack on a 56-bit key. This attack can be thwarted by appending some random bits to such simple messages.

## 6.2 THE RSA ALGORITHM

The pioneering paper by Diffie and Hellman [DIFF76b] introduced a new approach to cryptography and, in effect, challenged cryptologists to come up with a cryptographic algorithm that met the requirements for public-key systems. One of the first of the responses to the challenge was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978 [RIVE78].<sup>4</sup> The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the only widely accepted and implemented general-purpose approach to public-key encryption.

The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and  $n - 1$  for some  $n$ . We examine RSA in this section in some detail, beginning with an explanation of the algorithm.<sup>5</sup> Then we examine some of the computational and cryptanalytical implications of RSA.

### Description of the Algorithm

The scheme developed by Rivest, Shamir, and Adleman makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number  $n$ . That is, the block size must be less than or equal to  $\log_2(n)$ ; in practice, the block size is  $2^k$  bits, where  $2^k < n \leq 2^{k+1}$ . Encryption and decryption are of the following form, for some plaintext block  $M$  and ciphertext block  $C$ :

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Both sender and receiver must know the value of  $n$ . The sender knows the value of  $e$ , and only the receiver knows the value of  $d$ . Thus, this is a public-key encryption algorithm with a public key of  $KU = \{e, n\}$  and a private key of  $KR = \{d, n\}$ . For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:

1. It is possible to find values of  $e, d, n$  such that  $M^{ed} = M \bmod n$  for all  $M < n$ .
2. It is relatively easy to calculate  $M^e$  and  $C^d$  for all values of  $M < n$ .
3. It is infeasible to determine  $d$  given  $e$  and  $n$ .

<sup>4</sup>Apparently, the first workable public-key system for encryption/decryption was put forward by Clifford Cocks of Britain's CESG in 1973 [COCK73]; Cocks's method is virtually identical to RSA.

<sup>5</sup>For the discussion that follows, you should be familiar with the concepts of prime numbers, modular arithmetic, and other relevant aspects of number theory, which are discussed in Chapter 7.

For now, we focus on the first question and consider the other questions later. We need to find a relationship of the form

$$M^{ed} = M \pmod{n}$$

A corollary to Euler's theorem, presented in Chapter 7 [Equation (7.7)], fits the bill: Given two prime numbers,  $p$  and  $q$ , and two integers,  $n$  and  $m$ , such that  $n = pq$  and  $0 < m < n$ , and arbitrary integer  $k$ , the following relationship holds:

$$m^{k\phi(n)+1} = m^{k(p-1)(q-1)+1} \equiv m \pmod{n}$$

where  $\phi(n)$  is the Euler totient function, which is the number of positive integers less than  $n$  and relatively prime to  $n$ . It is shown in Chapter 7 that for  $p, q$  prime,  $\phi(pq) = (p-1)(q-1)$ . Thus, we can achieve the desired relationship if

$$ed = k\phi(n) + 1$$

This is equivalent to saying:

$$ed \equiv 1 \pmod{\phi(n)}$$

$$d \equiv e^{-1} \pmod{\phi(n)}$$

That is,  $e$  and  $d$  are multiplicative inverses mod  $\phi(n)$ . Note that, according to the rules of modular arithmetic, this is true only if  $d$  (and therefore  $e$ ) is relatively prime to  $\phi(n)$ . Equivalently,  $\gcd(\phi(n), d) = 1$ .

We are now ready to state the RSA scheme. The ingredients are the following:

$p, q$ , two prime numbers	(private, chosen)
$n = pq$	(public, calculated)
$e$ , with $\gcd(\phi(n), e) = 1$ ; $1 < e < \phi(n)$	(public, chosen)
$d \equiv e^{-1} \pmod{\phi(n)}$	(private, calculated)

The private key consists of  $\{d, n\}$  and the public key consists of  $\{e, n\}$ . Suppose that user A has published its public key and that user B wishes to send the message  $M$  to A. Then B calculates  $C = M^e \pmod{n}$  and transmits  $C$ . On receipt of this ciphertext, user A decrypts by calculating  $M = C^d \pmod{n}$ .

It is worthwhile to summarize the justification for this algorithm. We have chosen  $e$  and  $d$  such that

$$d \equiv e^{-1} \pmod{\phi(n)}$$

Therefore,

$$ed \equiv 1 \pmod{\phi(n)}$$

Therefore,  $ed$  is of the form  $k\phi(n) + 1$ . But by the corollary to Euler's theorem, provided in Chapter 7, given two prime numbers,  $p$  and  $q$ , and integers  $n = pq$  and  $M$ , with  $0 < M < n$ ,

$$M^{k\phi(n)+1} = M^{k(p-1)(q-1)+1} \equiv M \pmod{n}$$

So  $M^{ed} \equiv M \pmod{n}$ . Now

$$C = M^e \pmod{n}$$

$$M = C^d \pmod{n} \equiv (M^e)^d \pmod{n} \equiv M^{ed} \pmod{n} \equiv M \pmod{n}$$

Figure 6.5 summarizes the RSA algorithm. An example is shown in Figure 6.6. For this example, the keys were generated as follows:

1. Select two prime numbers,  $p = 7$  and  $q = 17$ .
2. Calculate  $n = pq = 7 \times 17 = 119$ .
3. Calculate  $\phi(n) = (p-1)(q-1) = 96$ .
4. Select  $e$  such that  $e$  is relatively prime to  $\phi(n) = 96$  and less than  $\phi(n)$ ; in this case,  $e = 5$ .
5. Determine  $d$  such that  $de = 1 \pmod{96}$  and  $d < 96$ . The correct value is  $d = 77$ , because  $77 \times 5 = 385 = 4 \times 96 + 1$ .

The resulting keys are public key  $KU = \{5, 119\}$  and private key  $KR = \{77, 119\}$ . The example shows the use of these keys for a plaintext input of  $M = 19$ . For

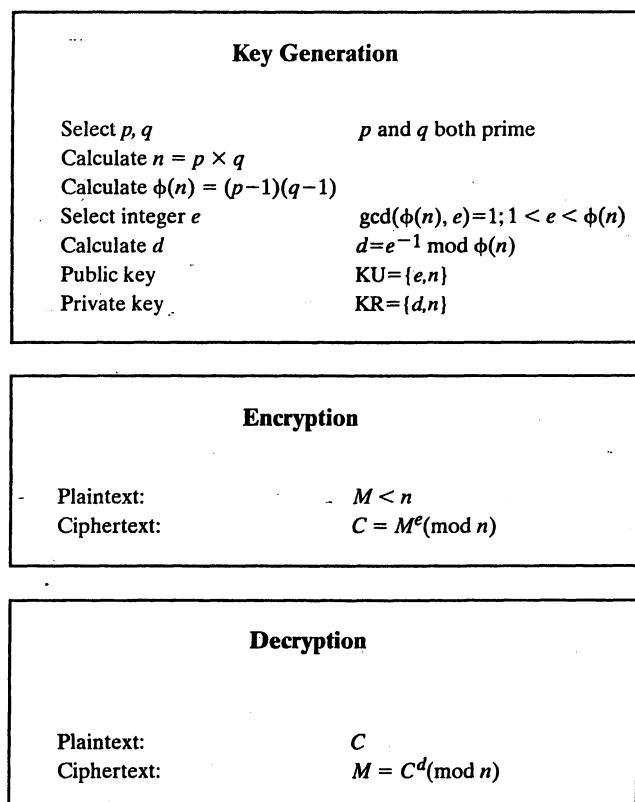


Figure 6.5 The RSA Algorithm.

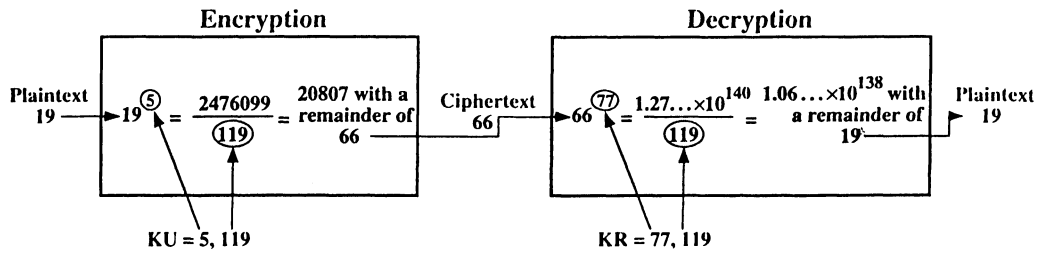


Figure 6.6 Example of RSA Algorithm.

encryption, 19 is raised to the fifth power, yielding 2476099. Upon division by 119, the remainder is determined to be 66. Hence  $19^5 \equiv 66 \pmod{119}$ , and the ciphertext is 66. For decryption, it is determined that  $66^{77} \equiv 19 \pmod{119}$ .

### Computational Aspects

We now turn to the issue of the complexity of the computation required to use RSA. There are actually two issues to consider: key generation and encryption/decryption. Let us look first at the process of encryption and decryption and then return to the issue of key generation.

#### Encryption and Decryption

Both encryption and decryption in RSA involve raising an integer to an integer power, mod  $n$ . If the exponentiation is done over the integers and then reduced modulo  $n$ , the intermediate values would be gargantuan. Fortunately, we can make use of a property of modular arithmetic:

$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

Thus, we can reduce intermediate results modulo  $n$ . This makes the calculation practical.

Another consideration is the efficiency of exponentiation, because with RSA we are dealing with potentially large exponents. To see how efficiency might be increased, consider that we wish to compute  $x^{16}$ . A straightforward approach requires 15 multiplications:

$$x^{16} = x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x$$

However, we can achieve the same final result with only four multiplications if we repeatedly take the square of each partial result, successively forming  $x^2, x^4, x^8, x^{16}$ .

More generally, suppose we wish to find the value  $a^m$ , with  $a$  and  $m$  positive integers. If we express  $m$  as a binary number  $b_k b_{k-1} \dots b_0$ , then we have

$$m = \sum_{b_i \neq 0} 2^i$$



```

c ← 0; d ← 1
for i ← k downto 0
  do c ← 2 × c
    d ← (d × d) mod n
    if bi = 1
      then c ← c + 1
        d ← (d × a) mod n
return d

```

**Figure 6.7** Algorithm for Computing  $a^b \bmod n$ .

Therefore,

$$a^m = a^{\left(\sum_{b_i \neq 0} 2^i\right)} = \prod_{b_i \neq 0} a^{(2^i)}$$

$$a^m \bmod n = \left[ \prod_{b_i \neq 0} a^{(2^i)} \right] \bmod n = \prod_{b_i \neq 0} \left[ a^{(2^i)} \bmod n \right]$$

We can therefore develop the algorithm<sup>6</sup> for computing  $a^b \bmod n$ , shown in Figure 6.7. Figure 6.8 shows an example of the execution of this algorithm. Note that the variable  $c$  is not needed; it is included for explanatory purposes. The final value of  $c$  is the value of the exponent.

### Key Generation

Before the application of the public-key cryptosystem, each participant must generate a pair of keys. This involves the following tasks:

- Determining two prime numbers,  $p$  and  $q$
- Selecting either  $e$  or  $d$  and calculating the other

First, consider the selection of  $p$  and  $q$ . Because the value of  $n = pq$  will be known to any potential opponent, to prevent the discovery of  $p$  and  $q$  by exhaustive methods, these primes must be chosen from a sufficiently large set (i.e.,  $p$  and  $q$  must

$i$	9	8	7	6	5	4	3	2	1	0
$b_i$	1	0	0	0	1	1	0	0	0	0
$c$	1	2	4	8	17	35	70	140	280	560
$d$	7	49	157	526	160	241	298	166	67	1

**Figure 6.8** Result of the Fast Modular Exponentiation Algorithm for  $a^b \bmod n$ , where  $a = 7$ ,  $b = 560 = 1000110000$ ,  $n = 561$ .

<sup>6</sup>The algorithm has a long history; this particular pseudocode expression is from [CORM90].

be large numbers). On the other hand, the method used for finding large primes must be reasonably efficient.

At present, there are no useful techniques that yield arbitrarily large primes, so some other means of tackling the problem is needed. The procedure that is generally used is to pick at random an odd number of the desired order of magnitude and test whether that number is prime. If not, pick successive random numbers until one is found that tests prime.

A variety of tests for primality have been developed (e.g., see [KNUT98] for a description of a number of such tests). Almost invariably, the tests are probabilistic. That is, the test will merely determine that a given integer is *probably* prime. Despite this lack of certainty, these tests can be run in such a way as to make the probability as close to 1.0 as desired. As an example, one of the more efficient and popular algorithms, the Miller-Rabin algorithm, is described in Chapter 7. With this algorithm and most such algorithms, the procedure for testing whether a given integer  $n$  is prime is to perform some calculation that involves  $n$  and a randomly chosen integer  $a$ . If  $n$  “fails” the test, then  $n$  is not prime. If  $n$  “passes” the test, then  $n$  may be prime or nonprime. If  $n$  passes many such tests with many different randomly chosen values for  $a$ , then we can have high confidence that  $n$  is, in fact, prime.

In summary, the procedure for picking a prime number is as follows:

1. Pick an odd integer  $n$  at random (e.g., using a pseudorandom number generator).
2. Pick an integer  $a < n$  at random.
3. Perform the probabilistic primality test, such as Miller-Rabin. If  $n$  fails the test, reject the value  $n$  and go to step 1.
4. If  $n$  has passed a sufficient number of tests, accept  $n$ ; otherwise, go to step 2.

This is a somewhat tedious procedure. However, remember that this process is performed relatively infrequently: only when a new pair (KU, KR) is needed.

It is worth noting how many numbers are likely to be rejected before a prime number is found. A result from number theory, known as the prime number theorem, states that the primes near  $N$  are spaced on the average one every  $(\ln N)$  integers. Thus, on average, one would have to test on the order of  $\ln(N)$  integers before a prime is found. Actually, because all even integers can be immediately rejected, the correct figure is  $\ln(N)/2$ . For example, if a prime on the order of magnitude of  $2^{200}$  were sought, then about  $\ln(2^{200})/2 = 70$  trials would be needed to find a prime.

Having determined prime numbers  $p$  and  $q$ , the process of key generation is completed by selecting a value of  $e$  and calculating  $d$  or, alternatively, selecting a value of  $d$  and calculating  $e$ . Assuming the former, then we need to select an  $e$  such that  $\gcd(\phi(n), e) = 1$  and then calculate  $d = e^{-1} \bmod \phi(n)$ . Fortunately, there is a single algorithm that will, at the same time, calculate the greatest common divisor of two integers and, if the gcd is 1, determine the inverse of one of the integers modulo the other. The algorithm, referred to as the extended Euclid's algorithm, is explained in Chapter 7. Thus, the procedure is to generate a series of random numbers, testing each against  $\phi(n)$  until a number relatively prime to  $\phi(n)$  is found. Again, we can ask the question, How many random numbers must we test to find a usable number — that is, a number relatively prime to  $\phi(n)$ ? It can be shown easily that the probability that two random numbers are relatively prime is about 0.6; thus, very few tests would be needed to find a suitable integer (see Problem 7.1).

## The Security of RSA

Three possible approaches to attacking the RSA algorithm are as follows:

- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effect to factoring the product of two primes.
- **Timing attacks:** These depend on the running time of the decryption algorithm.

The defense against the brute-force approach is the same for RSA as for other cryptosystems — namely, use a large key space. Thus, the larger the number of bits in  $e$  and  $d$ , the better. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run.

In this subsection, we provide an overview of mathematical and timing attacks.

### The Factoring Problem

We can identify three approaches to attacking RSA mathematically:

- Factor  $n$  into its two prime factors. This enables calculation of  $\phi(n) = (p - 1) \times (q - 1)$ , which, in turn, enables determination of  $d = e^{-1}(\text{mod } \phi(n))$ .
- Determine  $\phi(n)$  directly, without first determining  $p$  and  $q$ . Again, this enables determination of  $d = e^{-1}(\text{mod } \phi(n))$ .
- Determine  $d$  directly, without first determining  $\phi(n)$ .

Most discussions of the cryptanalysis of RSA have focused on the task of factoring  $n$  into its two prime factors. Determining  $\phi(n)$  given  $n$  is equivalent to factoring  $n$  [RIBE96]. With presently known algorithms, determining  $d$  given  $e$  and  $n$ , appears to be at least as time-consuming as the factoring problem [KALI95]. Hence, we can use factoring performance as a benchmark against which to evaluate the security of RSA.

For a large  $n$  with large prime factors, factoring is a hard problem, but not as hard as it used to be. A striking illustration of this is the following. In 1977, the three inventors of RSA dared *Scientific American* readers to decode a cipher they printed in Martin Gardner's "Mathematical Games" column. They offered a \$100 reward for the return of a plaintext sentence, an event they predicted might not occur for some 40 quadrillion years. In April of 1994, a group working over the Internet claimed the prize after only eight months of work. This challenge used a public key size (length of  $n$ ) of 129 decimal digits, or around 428 bits. In the meantime, just as they had done for DES, RSA Laboratories had issued challenges for ciphers with key sizes of 100, 110, 120, and so on, digits. The latest challenge to be met is the RSA-130 challenge with a key length of 130 decimal digits [COWE96]. Table 6.3 shows the results to date. The level of effort is measured in MIPS-years: a million-instructions-per-second processor running for one year, which is about  $3 \times 10^{13}$  instructions executed. A 200-MHz Pentium is about a 50-MIPS machine.

A striking fact about Table 6.3 concerns the method used. Until fairly recently, factoring attacks were made using an approach known as the quadratic sieve. The

Table 6.3 Progress in Factorization

Number of Decimal Digits	Approximate Number of Bits	Data Achieved	MIPS-Years	Algorithm
100	332	April 1991	7	Quadratic sieve
110	365	April 1992	75	Quadratic sieve
120	398	June 1993	830	Quadratic sieve
129	428	April 1994	5000	Quadratic sieve
130	431	April 1996	500	Generalized number field sieve

attack on RSA-130 used a newer algorithm, the generalized number field sieve (GNFS), and was able to factor a larger number than RSA-129 at only 10% of the computing effort.

The threat to larger key sizes is twofold: the continuing increase in computing power, and the continuing refinement of factoring algorithms. We have seen that the move to a different algorithm resulted in a tremendous speedup. We can expect further refinements in the GNFS, and the use of an even better algorithm is also a possibility. In fact, a related algorithm, the special number field sieve (SNFS), can factor numbers with a specialized form considerably faster than the generalized number field sieve. Figure 6.9 compares the performance of the two algorithms. It is reasonable to expect a breakthrough that would enable a general factoring performance in about the same time as SNFS, or even better [ODLY95]. Thus, we need to be careful in choosing a key size for RSA. For the near future, a key size in the range of 1024 to 2048 bits seems reasonable.

In addition to specifying the size of  $n$ , a number of other constraints have been suggested by researchers. To avoid values of  $n$  that may be factored more easily, the algorithm's inventors suggest the following constraints on  $p$  and  $q$ :

1.  $p$  and  $q$  should differ in length by only a few digits. Thus, both  $p$  and  $q$  should be on the order of  $10^{75}$  to  $10^{100}$ .
2. Both  $(p - 1)$  and  $(q - 1)$  should contain a large prime factor.
3.  $\gcd(p - 1, q - 1)$  should be small.

In addition, it has been demonstrated that if  $e < n$  and  $d < n^{1/4}$ , then  $d$  can be easily determined [WIEN90].

### Timing Attacks

If one needed yet another lesson about how difficult it is to assess the security of a cryptographic algorithm, the appearance of timing attacks provides a stunning one. Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages [KOCH96]. Timing attacks are applicable not just to RSA, but to other public-key cryptography systems. This attack is alarming for two reasons: It comes from a completely unexpected direction and it is a ciphertext-only attack.

A timing attack is somewhat analogous to a burglar guessing the combination of a safe by observing how long it takes for someone to turn the dial from number to number. We can explain the attack using the modular exponentiation algorithm

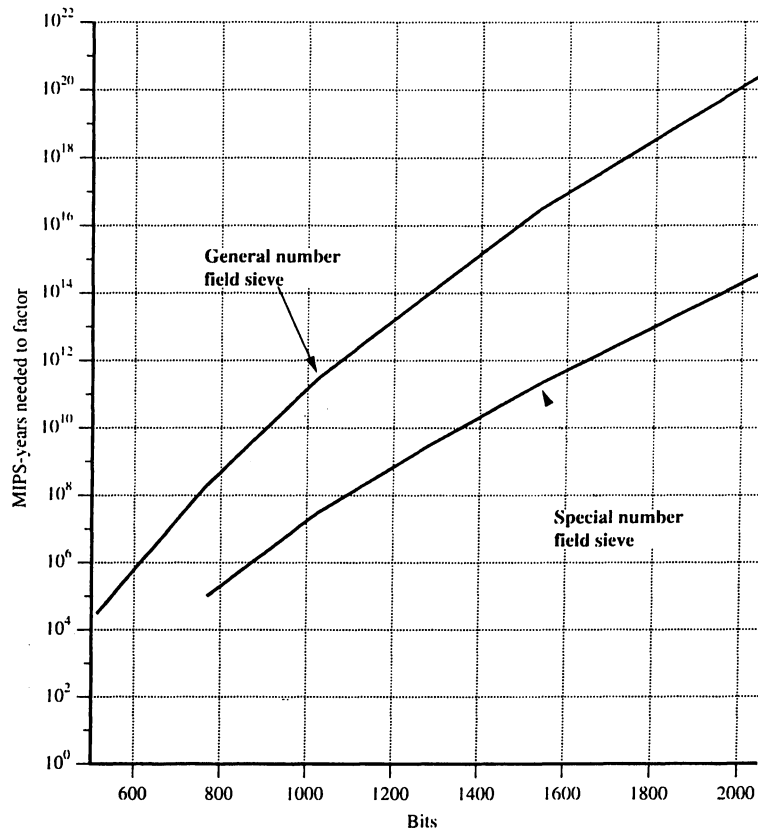


Figure 6.9 MIPS-years Needed to Factor.

of Figure 6.7, but the attack can be adapted to work with any implementation that does not run in fixed time. In this algorithm, modular exponentiation is accomplished bit by bit, with one modular multiplication performed at each iteration and an additional modular multiplication performed for each 1 bit.

As Kocher points out in his paper, the attack is simplest to understand in an extreme case. Suppose the target system uses a modular multiplication function that is very fast in almost all cases but in a few cases takes much more time than an entire average modular exponentiation. The attack proceeds bit by bit starting with the leftmost bit,  $b_k$ . Suppose that the first  $j$  bits are known. (To obtain the entire exponent, start with  $j = 0$  and repeat the attack until the entire exponent is known.) For a given ciphertext, the attacker can complete the first  $j$  iterations of the **for** loop. The operation of the subsequent step depends on the unknown exponent bit. If the bit is set,  $d \leftarrow (d \times a) \bmod n$  will be executed. For a few values of  $a$  and  $d$ , the modular multiplication will be extremely slow, and the attacker knows which these are. Therefore, if the observed time to execute the decryption algorithm is always slow when this particular iteration is slow with a 1 bit, then this bit is assumed to be 1. If a number of observed execution times for the entire algorithm are fast, then this bit is assumed to be 0.

In practice, modular exponentiation implementations do not have such extreme timing variations, in which the execution time of a single iteration can exceed the mean execution time of the entire algorithm. Nevertheless, there is enough variation to make this attack practical. For details, see [KOCH96].

Although the timing attack is a serious threat, there are simple countermeasures that can be used, including the following:

- **Constant exponentiation time:** Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.
- **Random delay:** Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. Kocher points out that if defenders do not add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays.
- **Blinding:** Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack.

RSA Data Security incorporates a blinding feature into some of its products. The private-key operation  $M = C^d \bmod n$  is implemented as follows:

1. Generate a secret random number  $r$  between 0 and  $n - 1$ .
2. Compute  $C' = C^e \bmod n$ , where  $e$  is the public exponent.
3. Compute  $M' = (C')^d \bmod n$  with the ordinary RSA implementation.
4. Compute  $M = M' r^{-1} \bmod n$ . In this equation,  $r^{-1}$  is the multiplicative inverse of  $r \bmod n$ ; see Chapter 7 for a discussion of this concept. It can be demonstrated that this is the correct result by observing that  $r^{ed} \bmod n = r \bmod n$ .

RSA Data Security reports a 2 to 10% performance penalty for blinding.

### 6.3 KEY MANAGEMENT

In Chapter 5, we examined the problem of the distribution of secret keys. One of the major roles of public-key encryption has been to address the problem of key distribution. There are actually two distinct aspects to the use of public-key encryption in this regard:

- The distribution of public keys
- The use of public-key encryption to distribute secret keys

We examine each of these areas in turn.

#### Distribution of Public Keys

Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:

- Public announcement
- Publicly available directory
- Public-key authority
- Public-key certificates

#### Public Announcement of Public Keys

On the face of it, the point of public-key encryption is that the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large (Figure 6.10). For example, because of the growing popularity of PGP (pretty good privacy, discussed in Chapter 12), which makes use of RSA, many PGP users have adopted the practice of appending their public key to messages that they send to public forums, such as USENET newsgroups and Internet mailing lists.

Although this approach is convenient, it has a major weakness: Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication (see Figure 6.3).

#### Publicly Available Directory

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization (Figure 6.11). Such a scheme would include the following elements:

1. The authority maintains a directory with a {name, public key} entry for each participant.
2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.



Figure 6.10 Uncontrolled Public-Key Distribution.

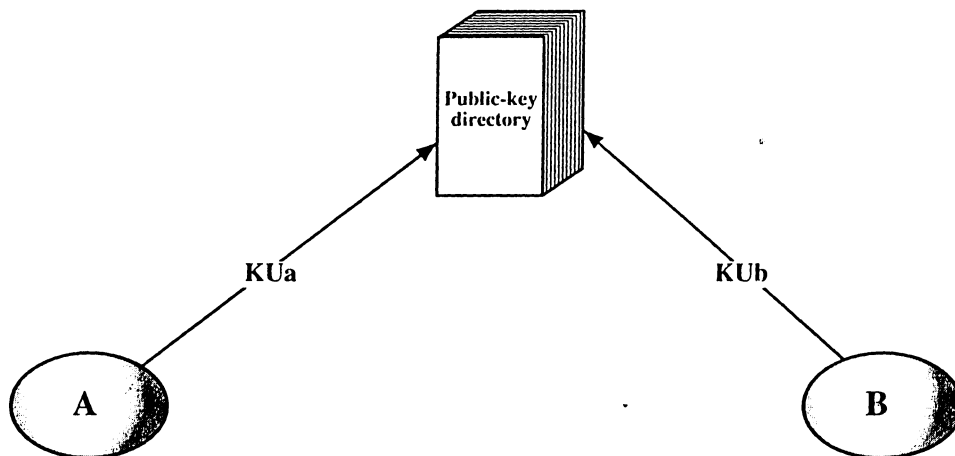


Figure 6.11 Public-Key Publication.

3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data or because the corresponding private key has been compromised in some way.
4. Periodically, the authority publishes the entire directory or updates to the directory. For example, a hard-copy version much like a telephone book could be published, or updates could be listed in a widely circulated newspaper.
5. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

This scheme is clearly more secure than individual public announcements but still has vulnerabilities. If an opponent succeeds in obtaining or computing the private key of the directory authority, the opponent could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Another way to achieve the same end is for the opponent to tamper with the records kept by the authority.

#### Public-Key Authority

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. A typical scenario is illustrated in Figure 6.12, which is based on a figure in [POPE79]. As before, the scenario assumes that a central authority maintains a dynamic directory of public keys of all participants. In addition, each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key. The following steps (matched by number to Figure 6.12) occur:

1. A sends a timestamped message to the public-key authority containing a request for the current public key of B.



2. The authority responds with a message that is encrypted using the authority's private key,  $KR_{auth}$ . Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:
  - B's public key,  $KU_b$ , which A can use to encrypt messages destined for B
  - The original request, to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority
  - The original timestamp, so A can determine that this is not an old message from the authority containing a key other than B's current public key
3. A stores B's public key and uses it to encrypt a message to B containing an identifier of A ( $ID_A$ ) and a nonce ( $N_1$ ), which is used to identify this transaction uniquely.
- 4, 5. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.

At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

6. B sends a message to A encrypted with  $KU_a$  and containing A's nonce ( $N_1$ ) as well as a new nonce generated by B ( $N_2$ ). Because only B could have decrypted message (3), the presence of  $N_1$  in message (6) assures A that the correspondent is B.
7. A returns  $N_2$ , encrypted using B's public key, to assure B that its correspondent is A.

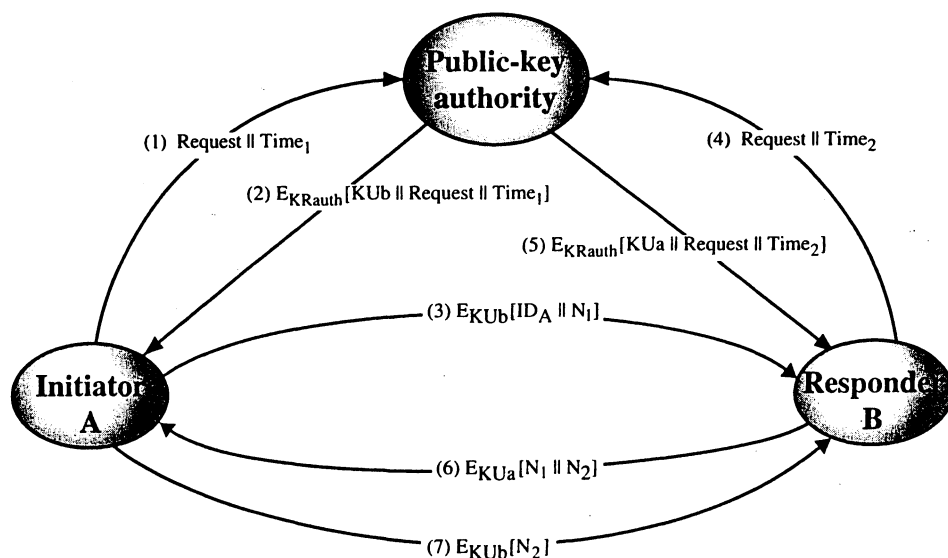


Figure 6.12. Public-Key Distribution Scenario.

Thus, a total of seven messages are required. However, the initial four messages need be used only infrequently because both A and B can save the other's public key for future use, a technique known as caching. Periodically, a user should request fresh copies of the public keys of its correspondents to ensure currency.

### Public-Key Certificates

The scenario of Figure 6.12 is attractive, yet it has some drawbacks. The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact. As before, the directory of names and public keys maintained by the authority is vulnerable to tampering.

An alternative approach, first suggested by Kohnfelder [KOH78], is to use **certificates** that can be used by participants to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. Each certificate contains a public key and other information, is created by a certificate authority, and is given to the participant with the matching private key. A participant conveys its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority. We can place the following requirements on this scheme:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.

These requirements are satisfied by the original proposal in [KOH78]. Denning [DEN83] added the following additional requirement:

4. Any participant can verify the currency of the certificate.

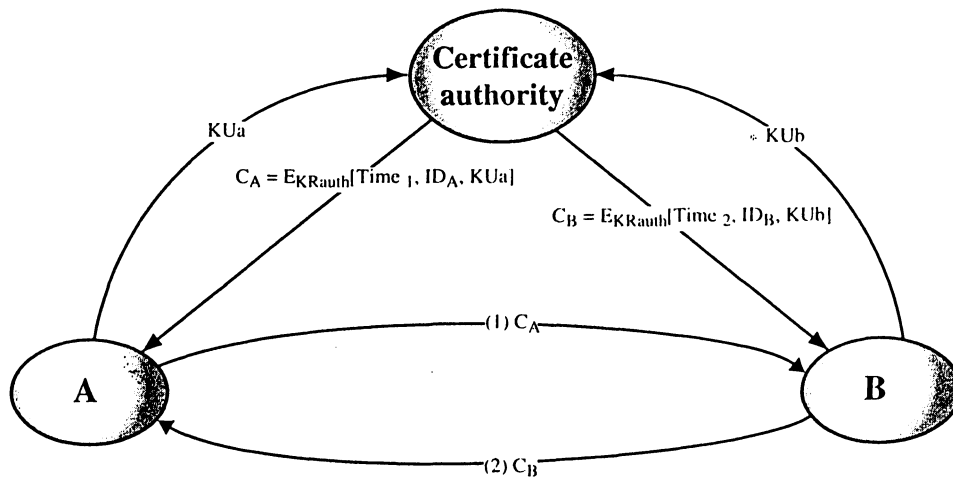
A certificate scheme is illustrated in Figure 6.13. Each participant applies to the certificate authority, supplying a public key and requesting a certificate. Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form

$$C_A = E_{KR_{auth}}[T, ID_A, KU_a]$$

where  $KR_{auth}$  is the private key used by the authority. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

$$D_{KU_{auth}}[C_A] = D_{KU_{auth}}[E_{KR_{auth}}[T, ID_A, KU_a]] = (T, ID_A, KU_a)$$

The recipient uses the authority's public key,  $KU_{auth}$ , to decrypt the certificate. Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority. The elements  $ID_A$  and  $KU_a$  provide the recipient with the name and public key of the certificate's holder. Finally, the timestamp  $T$  validates the currency of the certificate. The timestamp



**Figure 6.13** Exchange of Public-Key Certificates.

counters the following scenario: A's private key is learned by an opponent. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the opponent replays the old certificate to B. If B then encrypts messages using the compromised old public key, the opponent can read those messages.

In this context, the compromise of a private key is comparable to the loss of a credit card. The owner cancels the credit card number but is at risk until all possible communicants are aware that the old credit card is obsolete. Thus, the time-stamp serves as something like an expiration date. If a certificate is sufficiently old, it is assumed to be expired.

### Public-Key Distribution of Secret Keys

Once public keys have been distributed or have become accessible, secure communication that thwarts eavesdropping (Figure 6.2), tampering (Figure 6.3), or both (Figure 6.4) is possible. However, few users will wish to make exclusive use of public-key encryption for communication because of the relatively slow data rates that can be achieved. Accordingly, public-key encryption is more reasonably viewed as a vehicle for the distribution of secret keys to be used for conventional encryption.

#### Simple Secret Key Distribution

An extremely simple scheme was put forward by Merkle [MERK79], as illustrated in Figure 6.14. If A wishes to communicate with B, the following procedure is employed:

1. A generates a public/private key pair  $\{KU_a, KR_a\}$  and transmits a message to B consisting of  $KU_a$  and an identifier of A,  $ID_A$ .
2. B generates a secret key,  $K_s$ , and transmits it to A, encrypted with A's public key.

3. A computes  $D_{KR_a}[E_{KU_a}[K_s]]$  to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of  $K_s$ .
4. A discards  $KU_a$  and  $KR_a$  and B discards  $KU_a$ .

A and B can now securely communicate using conventional encryption and the session key  $K_s$ . At the completion of the exchange, both A and B discard  $K_s$ . Despite its simplicity, this is an attractive protocol. No keys exist before the start of the communication and none exist after the completion of communication. Thus, the risk of compromise of the keys is minimal. At the same time, the communication is secure from eavesdropping.

This protocol is vulnerable to an active attack. If an opponent, E, has control of the intervening communication channel, then E can compromise the communication in the following fashion without being detected:

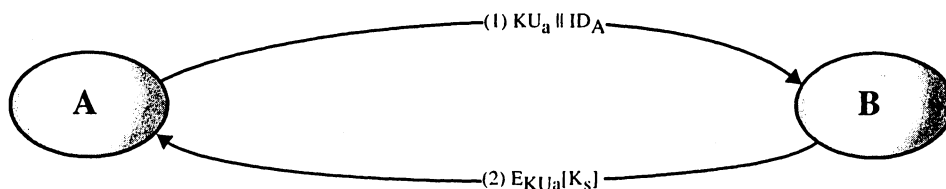
1. A generates a public/private key pair  $\{KU_a, KR_a\}$  and transmits a message intended for B consisting of  $KU_a$  and an identifier of A,  $ID_A$ .
2. E intercepts the message, creates its own public/private key pair  $\{KU_e, KR_e\}$ , and transmits  $KU_e \parallel ID_A$  to B.
3. B generates a secret key,  $K_s$ , and transmits  $E_{KU_e}[K_s]$ .
4. E intercepts the message and learns  $K_s$  by computing  $D_{KR_e}[E_{KU_e}[K_s]]$ .
5. E transmits  $E_{KU_a}[K_s]$  to A.

The result is that both A and B know  $K_s$  and are unaware that  $K_s$  has also been revealed to E. A and B can now exchange messages using  $K_s$ . E no longer actively interferes with the communications channel but simply eavesdrops. Knowing  $K_s$ , E can decrypt all messages, and both A and B are unaware of the problem. Thus, this simple protocol is only useful in an environment where the only threat is eavesdropping.

#### Secret Key Distribution with Confidentiality and Authentication

Figure 6.15, based on an approach suggested in [NEED78], provides protection against both active and passive attacks. We begin at a point when it is assumed that A and B have exchanged public keys by one of the schemes described earlier in this section. Then the following steps occur:

1. A uses B's public key to encrypt a message to B containing an identifier of A ( $ID_A$ ) and a nonce ( $N_1$ ), which is used to identify this transaction uniquely.
2. B sends a message to A encrypted with  $KU_a$  and containing A's nonce ( $N_1$ ) as well as a new nonce generated by B ( $N_2$ ). Because only B could have



**Figure 6.14** Simple Use of Public-Key Encryption to Establish a Session Key.

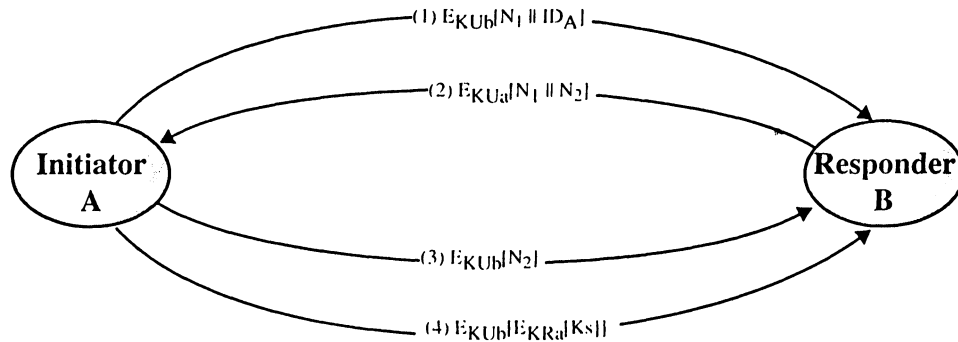


Figure 6.15 Public-Key Distribution of Secret Keys.

decrypted message (1), the presence of  $N_1$  in message (2) assures A that the correspondent is B.

3. A returns  $N_2$ , encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key  $K_s$  and sends  $M = E_{K_{Ub}}[E_{K_{Ra}}[K_s]]$  to B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.
5. B computes  $D_{K_{Ua}}[D_{K_{Rb}}[M]]$  to recover the secret key.

Notice that the first three steps of this scheme are the same as the last three steps of Figure 6.12. The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.

#### A Hybrid Scheme

Yet another way to use public-key encryption to distribute secret keys is a hybrid approach in use on IBM mainframes [LE93]. This scheme retains the use of a key distribution center (KDC) that shares a secret master key with each user and distributes secret session keys encrypted with the master key. A public-key scheme is used to distribute the master keys. The following rationale is provided for using this three-level approach:

- **Performance:** There are many applications, especially transaction-oriented applications, in which the session keys change frequently. Distribution of session keys by public-key encryption could degrade overall system performance because of the relatively high computational load of public-key encryption and decryption. With a three-level hierarchy, public-key encryption is used only occasionally to update the master key between a user and the KDC.
- **Backward compatibility:** The hybrid scheme is easily overlaid on an existing KDC scheme, with minimal disruption or software changes.

The addition of a public-key layer provides a secure, efficient means of distributing master keys. This is an advantage in a configuration in which a single KDC serves a widely distributed set of users.

## 6.4 DIFFIE-HELLMAN KEY EXCHANGE

The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography [DIFF76b] and is generally referred to as Diffie-Hellman key exchange<sup>7</sup>. A number of commercial products employ this key exchange technique.

The purpose of the algorithm is to enable two users to exchange a key securely that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of the keys.

The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. Briefly, we can define the discrete logarithm in the following way. First, we define a primitive root of a prime number  $p$  as one whose powers generate all the integers from 1 to  $p - 1$ . That is, if  $a$  is a primitive root of the prime number  $p$ , then the numbers

$$a \bmod p, a^2 \bmod p, \dots, a^{p-1} \bmod p$$

are distinct and consist of the integers from 1 through  $p-1$  in some permutation.

For any integer  $b$  and a primitive root  $a$  of prime number  $p$ , one can find a unique exponent  $i$  such that

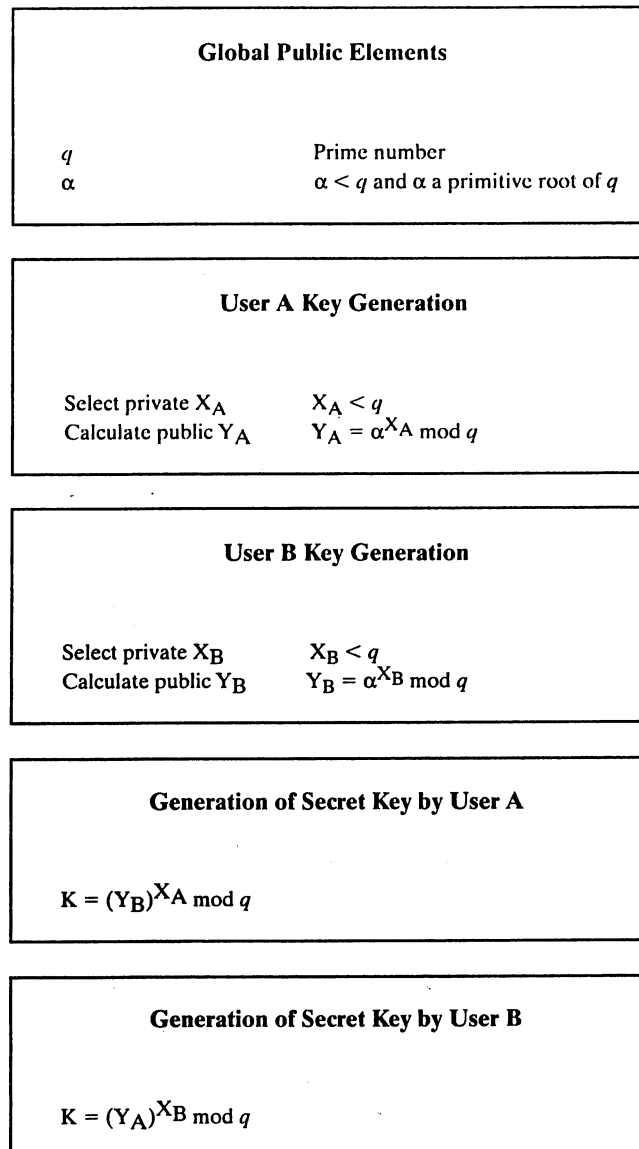
$$b = a^i \bmod p \quad \text{where } 0 \leq i \leq (p - 1)$$

The exponent  $i$  is referred to as the discrete logarithm, or index, of  $b$  for the base  $a$ , mod  $p$ . This value is denoted as  $\text{ind}_{a,p}(b)$ . See Chapter 7 for an extended discussion of discrete logarithms.

With this background we can define the Diffie-Hellman key exchange, which is summarized in Figure 6.16. For this scheme, there are two publicly known numbers: a prime number  $q$  and an integer  $\alpha$  that is a primitive root of  $q$ . Suppose the users A and B wish to exchange a key. User A selects a random integer  $X_A < q$  and computes  $Y_A = \alpha^{X_A} \bmod q$ . Similarly, user B independently selects a random integer  $X_B < q$  and computes  $Y_B = \alpha^{X_B} \bmod q$ . Each side keeps the  $X$  value private and makes the  $Y$  value available publicly to the other side. User A computes the key as  $K = (Y_B)^{X_A} \bmod q$  and user B computes the key as  $K = (Y_A)^{X_B} \bmod q$ . These two calculations produce identical results:

$$\begin{aligned} K &= (Y_B)^{X_A} \bmod q \\ &= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\ &= (\alpha^{X_B})^{X_A} \bmod q && \text{by the rules of modular arithmetic} \\ &= \alpha^{X_B X_A} \bmod q \\ &= (\alpha^{X_A})^{X_B} \bmod q \\ &= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\ &= (Y_A)^{X_B} \bmod q \end{aligned}$$

<sup>7</sup>Williamson of Britain's CESG published the identical scheme a few months earlier in a classified document [WILL76] and claims to have discovered it several years prior to that; see [ELLI87] for a discussion.



**Figure 6.16** The Diffie-Hellman Key Exchange Algorithm.

Thus, the two sides have exchanged a secret key. Furthermore, because  $X_A$  and  $X_B$  are private, an opponent only has the following ingredients to work with:  $q$ ,  $a$ ,  $Y_A$ , and  $Y_B$ . Thus, the opponent is forced to take a discrete logarithm to determine the key. For example, attacking the secret key of user B, the opponent must compute

$$X_B = \text{ind}_{\alpha, q}(Y_B)$$

The opponent can then calculate the key  $K$  in the same manner as user B calculates it.

The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

Here is an example, taken from [SEBE89]. Key exchange is based on the use of the prime number  $q = 97$  and a primitive root of 97, in this case  $\alpha = 5$ . A and B select secret keys  $X_A = 36$  and  $X_B = 58$ , respectively. Each computes its public key:

$$Y_A = 5^{36} = 50 \bmod 97$$

$$Y_B = 5^{58} = 44 \bmod 97$$

After they exchange public keys, each can compute the common secret key:

$$K = (Y_B)^{X_A} \bmod 97 = 44^{36} = 75 \bmod 97$$

$$K = (Y_A)^{X_B} \bmod 97 = 50^{58} = 75 \bmod 97$$

From  $\{50, 44\}$ , an attacker cannot easily compute 75.

Figure 6.17 shows a simple protocol that makes use of the Diffie-Hellman calculation. Suppose that user A wishes to set up a connection with user B and use a secret key to encrypt messages on that connection. User A can generate a one-time private key  $X_A$ , calculate  $Y_A$ , and send that to user B. User B responds by generating a private value  $X_B$ , calculating  $Y_B$ , and sending  $Y_B$  to user A. Both users can now calculate the key. The necessary public values  $q$  and  $\alpha$  would need to be known ahead of time. Alternatively, user A could pick values for  $q$  and  $\alpha$  and include those in the first message.

As an example of another use of the Diffie-Hellman algorithm, suppose that a group of users (e.g., all users on a LAN) each generate a long-lasting private value  $X_A$  and calculate a public value  $Y_A$ . These public values, together with global public values for  $q$  and  $\alpha$ , are stored in some central directory. At any time, user B can access user A's public value, calculate a secret key, and use that to send an encrypted message to user A. If the central directory is trusted, then this form of communica-

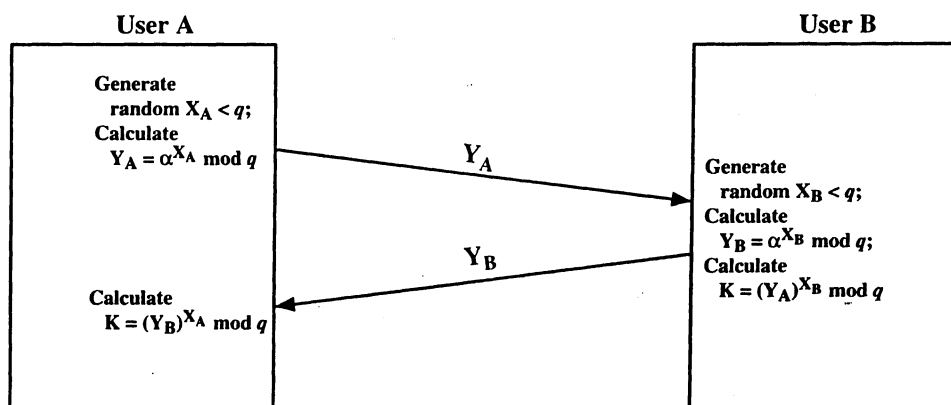


Figure 6.17 Diffie-Hellman Key Exchange.



tion provides both confidentiality and a degree of authentication. Because only A and B can determine the key, no other user can read the message (confidentiality). Recipient A knows that only user B could have created a message using this key (authentication). However, the technique does not protect against replay attacks.

## 6.5 ELLIPTIC CURVE CRYPTOGRAPHY

The vast majority of the products and standards that use public-key cryptography for encryption and digital signatures use RSA. As we have seen, the bit length for secure RSA use has increased over recent years, and this has put a heavier processing load on applications using RSA. This burden has ramifications, especially for electronic commerce sites that conduct large numbers of secure transactions. Recently, a competing system has begun to challenge RSA: elliptic curve cryptography (ECC). Already, ECC is showing up in standardization efforts, including the IEEE P1363 Standard for Public-Key Cryptography.

The principal attraction of ECC compared to RSA is that it appears to offer equal security for a far smaller bit size, thereby reducing processing overhead. On the other hand, although the theory of ECC has been around for some time, it is only recently that products have begun to appear and that there has been sustained cryptanalytic interest in probing for weaknesses. Thus, the confidence level in ECC is not yet as high as that in RSA.

ECC is fundamentally more difficult to explain than either RSA or Diffie-Hellman, and a full mathematical description is beyond the scope of this book. This section gives some background on elliptic curves and ECC.

### Elliptic Curves

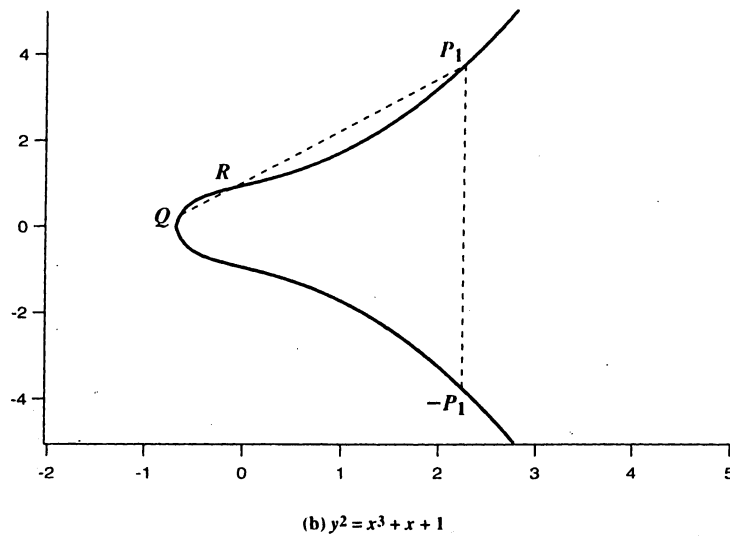
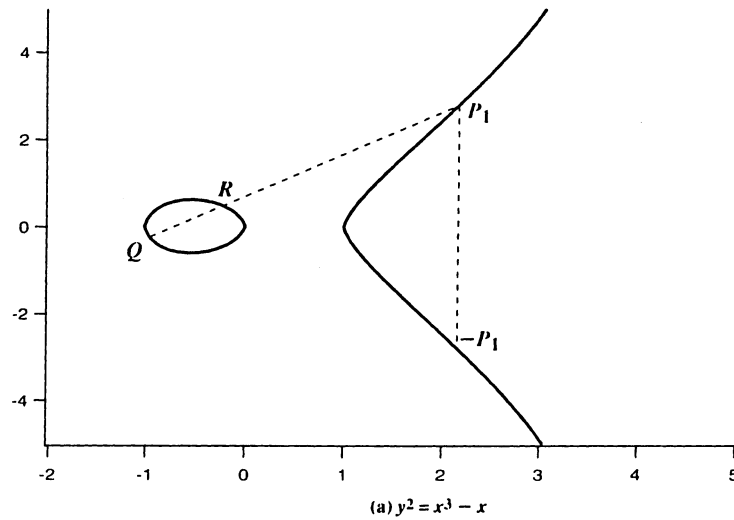
Elliptic curves are not ellipses. They are so named because they are described by cubic equations, similar to those used for calculating the circumference of an ellipse. In general, cubic equations for elliptic curves take the form

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

where  $a$ ,  $b$ ,  $c$ ,  $d$ , and  $e$  are real numbers that satisfy some simple conditions. Also included in the definition of any elliptic curve is a single element denoted  $O$  and called the *point at infinity* or the *zero point*, which we discuss subsequently. Such equations are said to be cubic, or of degree 3, because the highest exponent they contain is a 3. Figure 6.18 shows two examples of elliptic curves. As you can see, the formula sometimes produces weird-looking curves.

A form of addition can be defined for an elliptic curve, which in its short form is stated as follows: If three points on an elliptic curve lie on a straight line, their sum is  $O$ . From this definition, we can define the rules of addition over an elliptic curve:

1.  $O$  serves as the additive identity. Thus  $O = -O$ ; for any point  $P$  on the elliptic curve,  $P + O = P$ .



**Figure 6.18** Example of Elliptic Curves.

2. A vertical line meets the curve at two points with the same  $x$  coordinate—say,  $P_1 = (x, y)$  and  $P_2 = (x, -y)$ . It also meets the curve at the infinity point. Therefore,  $P_1 + P_2 + O = O$ , and  $P_1 = -P_2$ . Thus, the negative of a point is a point with the same  $x$  coordinate but negative  $y$  coordinate. Figure 6.18 illustrates this.
3. To add two points  $Q$  and  $R$  with different  $x$  coordinates, draw a straight line between them and find the third point of intersection  $P_1$ . It is easily seen that there is a unique point  $P_1$  that is the point of intersection (unless the line is tangent to the curve at either  $Q$  or  $R$ , in which case we take  $P_1 = Q$  or  $P_1 = R$ ,

respectively). Then note that  $Q + R + P_1 = O$  and so  $Q + R = -P_1$ . Figure 6.18 illustrates this construction.

4. To double a point  $Q$ , draw the tangent line and find the other point of intersection  $S$ . Then  $Q + Q = 2Q = -S$ .

The aforementioned rules of addition satisfy the normal properties of addition, such as commutative and associative laws. Multiplication of a point  $P$  on an elliptic curve by a positive integer  $k$  is defined, as you might expect, as the sum of  $k$  copies of  $P$ . Thus  $2P = P + P$ ,  $3P = P + P + P$ , and so on.

### Elliptic Curves over Finite Fields

For ECC, we are concerned with a restricted form of elliptic curve that is defined over a finite field. Of particular interest for cryptography is what is referred to as the elliptic group mod  $p$ , where  $p$  is a prime number. This is defined as follows. Choose two nonnegative integers,  $a$  and  $b$ , less than  $p$  that satisfy

$$4a^3 + 27b^2 \pmod{p} \neq 0$$

Then  $E_p(a, b)$  denotes the elliptic group mod  $p$  whose elements  $(x, y)$  are pairs of non-negative integers less than  $p$  satisfying

$$y^2 \equiv x^3 + ax + b \pmod{p} \quad (6.1)$$

together with the point at infinity  $O$ .

For example, let  $p = 23$  and consider the elliptic curve  $y^2 = x^3 + x + 1$ . In this case,  $a = b = 1$ . We have  $4 \times 1^3 + 27 \times 1^2 \pmod{23} = 8 \neq 0$ , which satisfies the condition for an elliptic group mod 23.

Note that the equation in the preceding paragraph is the same as that of Figure 6.18b. The figure shows a continuous curve with all of the real points that satisfy the equation. For the elliptic group, we are only interested in the nonnegative integers in the quadrant from  $(0, 0)$  to  $(p, p)$  that satisfy the equation mod  $p$ . Table 6.4 lists the points (other than  $O$ ) that are part of  $E_{23}(1, 1)$ . In general, the list is created in the following manner:

1. For each  $x$  such that  $0 \leq x < p$ , calculate  $x^3 + ax + b \pmod{p}$ .
2. For each result from the previous step, determine if it has a square root mod  $p$ . If not, there are no points in  $E_p(a, b)$  with this value of  $x$ . If so, there will be two values of  $y$  that satisfy the square root operation (unless the value is the single  $y$  value of 0). These  $(x, y)$  values are points in  $E_p(a, b)$ .

The rules for addition over  $E_p(a, b)$  correspond to the geometric technique illustrated in Figure 6.18. They can be stated as follows for all points  $P, Q \in E_p(a, b)$ :

1.  $P + O = P$ .
2. If  $P = (x, y)$ , then  $P + (x, -y) = O$ . The point  $(x, -y)$  is the negative of  $P$ , denoted as  $-P$ . Observe that  $(x, -y)$  is a point on the elliptic curve, as seen

**Table 6.4** Points on the Elliptic Curve  $E_{23}(1, 1)$ 

(0, 1)	(6, 4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)

graphically (Figure 6.18b) and in  $E_p(a, b)$ . For example, in  $E_{23}(1, 1)$ , for  $P = (13, 7)$ , we have  $-P = (13, -7)$ . But  $-7 \bmod 23 = 16$ . Therefore,  $-P = (13, 16)$ , which is also in  $E_{23}(1, 1)$ .

3. If  $P = (x_1, y_1)$  and  $Q = (x_2, y_2)$  with  $P \neq -Q$ , then  $P + Q = (x_3, y_3)$  is determined by the following rules:

$$x_3 \equiv \lambda^2 - x_1 - x_2 \pmod{p}$$

$$y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p}$$

where

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } P \neq Q \\ \frac{3x_1^2 + a}{2y_1} & \text{if } P = Q \end{cases}$$

We look at two examples, taken from [JURI97]. Let  $P = (3, 10)$  and  $Q = (9, 7)$ . Then

$$\lambda = \frac{7 - 10}{9 - 3} = \frac{-3}{6} = \frac{-1}{2} \equiv 11 \pmod{23}$$

$$x_3 = 11^2 - 3 - 9 = 109 \equiv 17 \pmod{23}$$

$$y_3 = 11(3 - (-6)) - 10 = 89 \equiv 20 \pmod{23}$$

So  $P + Q = (17, 20)$ . To find  $2P$ ,

$$\lambda = \frac{3(3^2) + 1}{2 \times 10} = \frac{5}{20} = \frac{1}{4} \equiv 6 \pmod{23}$$

$$x_3 = 6^2 - 3 - 3 = 30 \equiv 7 \pmod{23}$$

$$y_3 = 6(3 - 7) - 10 = -34 \equiv 12 \pmod{23}$$

and  $2P = (7, 12)$ .

Again, multiplication is defined as repeated addition; for example,  $4P = P + P + P + P$ .

### Cryptography with Elliptic Curves

The addition operation in ECC is the counterpart of modular multiplication in RSA, and multiple addition is the counterpart of modular exponentiation. To form a cryptographic system using elliptic curves, we need to find a “hard problem” corresponding to factoring the product of two primes or taking the discrete logarithm.

Consider the equation  $Q = kP$ , where  $Q, P \in E_p(a, b)$  and  $k < p$ . It is relatively easy to calculate  $Q$  given  $k$  and  $P$ , but it is relatively hard to determine  $k$  given  $Q$  and  $P$ .

In this subsection, we show two approaches to ECC that give the flavor of this technique.

#### Analog of Diffie-Hellman Key Exchange

Key exchange using elliptic curves can be done in the following manner. First pick a prime number  $p \approx 2^{180}$  and elliptic curve parameters  $a$  and  $b$  for Equation (6.1). This defines the elliptic group of points  $E_p(a, b)$ . Next, pick a *generator point*  $G = (x_1, y_1)$  in  $E_p(a, b)$ . The important criterion in selecting  $G$  is that the smallest value of  $n$  for which  $nG = O$  be a very large prime number.  $E_p(a, b)$  and  $G$  are parameters of the cryptosystem known to all participants.

A key exchange between users A and B can be accomplished as follows:

1. A selects an integer  $n_A$  less than  $n$ . This is A's private key. A then generates a public key  $P_A = n_A \times G$ ; the public key is a point in  $E_p(a, b)$ .
2. B similarly selects a private key  $n_B$  and computes a public key  $P_B$ .
3. A generates the secret key  $K = n_A \times P_B$ . B generates the secret key  $K = n_B \times P_A$ .

The two calculations in step 3 produce the same result because

$$n_A \times P_B = n_A \times (n_B \times G) = n_B \times (n_A \times G) = n_B \times P_A$$

To break this scheme, an attacker would need to be able to compute  $k$  given  $G$  and  $kG$ , which is assumed hard.

As an example,<sup>8</sup> take  $p = 211$ ;  $E_p(0, -4)$ , which is equivalent to the curve  $y^2 = x^3 - 4$ ; and  $G = (2, 2)$ . One can calculate that  $241G = O$ . A's private key is  $n_A = 121$ , so A's public key is  $P_A = 121(2, 2) = (115, 48)$ . B's private key is  $n_B = 203$ , so B's public key is  $203(2, 2) = (130, 203)$ . The shared secret key is  $121(130, 203) = 203(115, 48) = (161, 169)$ .

Note that the secret key is a pair of numbers. If this key is to be used as a session key for conventional encryption, then a single number must be generated. We could simply use the  $x$  coordinates or some simple function of the  $x$  coordinate.

<sup>8</sup>Provided by Ed Schaefer of Santa Clara University.

### Elliptic Curve Encryption/Decryption

Several approaches to encryption/decryption using elliptic curves have been analyzed in the literature. In this subsection we look at perhaps the simplest. The first task in this system is to encode the plaintext message  $m$  to be sent as an  $x$ - $y$  point  $P_m$ . It is the point  $P_m$  that will be encrypted as a ciphertext and subsequently decrypted. Note that we cannot simply encode the message as the  $x$  or  $y$  coordinate of a point, because not all such coordinates are in  $E_p(a, b)$ ; for example, see Table 6.4. Again, there are several approaches to this encoding, which we will not address here, but suffice it to say that there are relatively straightforward techniques that can be used.

As with the key exchange system, an encryption/decryption system requires a point  $G$  and an elliptic group  $E_p(a, b)$  as parameters. Each user  $A$  selects a private key  $n_A$  and generates a public key  $P_A = n_A \times G$ .

To encrypt and send a message  $P_m$  to  $B$ ,  $A$  chooses a random positive integer  $k$  and produces the ciphertext  $C_m$  consisting of the pair of points

$$C_m = \{kG, P_m + kP_B\}$$

Note that  $A$  has used  $B$ 's public key  $P_B$ . To decrypt the ciphertext,  $B$  multiplies the first point in the pair by  $B$ 's secret key and subtracts the result from the second point:

$$P_m + kP_B - n_B(kG) = P_m + k(n_B G) - n_B(kG) = P_m$$

$A$  has masked the message  $P_m$  by adding  $kP_B$  to it. Nobody but  $A$  knows the value of  $k$ , so even though  $P_B$  is a public key, nobody can remove the mask  $kP_B$ . However,  $A$  also includes a "clue," which is enough to remove the mask if one knows the private key  $n_B$ . For an attacker to recover the message, the attacker would have to compute  $k$  given  $G$  and  $kG$ , which is assumed hard.

As an example of the encryption process (taken from [KOBL94]), take  $p = 751$ ;  $E_p(-1, 188)$ , which is equivalent to the curve  $y^2 = x^3 - x + 188$ ; and  $G = (0, 376)$ . Suppose that  $A$  wishes to send a message to  $B$  that is encoded in the elliptic point  $P_m = (562, 201)$  and that  $A$  selects the random number  $k = 386$ .  $B$ 's public key is  $P_B = (201, 5)$ . We have  $386(0, 376) = (676, 558)$ , and  $(562, 201) + 386(201, 5) = (385, 328)$ . Thus  $A$  sends the cipher text  $\{(676, 558), (385, 328)\}$ .

### Security of Elliptic Curve Cryptography

The security of ECC depends on how difficult it is to determine  $k$  given  $kP$  and  $P$ . This is referred to as the elliptic curve logarithm problem. The fastest known technique for taking the elliptic curve logarithm is known as the Pollard rho method. Table 6.5 compares the efficiency of this method with factoring a number into two primes using the general number field sieve. As can be seen, a considerably smaller key size can be used for ECC compared to RSA. Furthermore, for equal key lengths, the computational effort required for ECC and RSA is comparable [JURI97]. Thus, there is a computational advantage to using ECC with a shorter key length than a comparably secure RSA.

**Table 6.5** Computational Effort for Cryptanalysis of Elliptic Curve Cryptography Compared to RSA

Key Size	MIPS-Years	Key Size	MIPS-Years
150	$3.8 \times 10^{10}$	512	$3 \times 10^4$
205	$7.1 \times 10^{18}$	768	$2 \times 10^8$
234	$1.6 \times 10^{28}$	1024	$3 \times 10^{11}$
		1280	$1 \times 10^{14}$
		1536	$3 \times 10^{16}$
		2048	$3 \times 10^{20}$

(a) Elliptic Curve Logarithms Using the Pollard rho Method

(b) Integer Factorization Using the General Number Field Sieve

## 6.6 RECOMMENDED READING

The recommended treatments of encryption provided in Chapter 3 cover public-key as well as conventional encryption.

[DIFF88] describes in detail the several attempts to devise secure two-key cryptoalgorithms and the gradual evolution of a variety of protocols based on them. A comprehensive treatment of contemporary public-key cryptology is given in [NECH92]. A good book-length treatment is [SALO96]. [CORM90] provides a concise but complete and readable summary of all of the algorithms relevant to the verification, computation, and cryptanalysis of RSA.

An excellent presentation of elliptic curve cryptography is [MENE93]. There are also good but more concise descriptions in [KUMA98], [STIN95], and [KOB94].

CORM90 Cormen, T.; Leiserson, C.; and Rivest, R. *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

DIFF88 Diffie, W. "The First Ten Years of Public-Key Cryptography." *Proceedings of the IEEE*, May 1988. Reprinted in [SIMM92].

KOB94 Koblitz, N. *A Course in Number Theory and Cryptography*. New York: Springer-Verlag, 1994.

KUMA98 Kumanduri, R., and Romero, C. *Number Theory with Computer Applications*. Upper Saddle River, NJ: Prentice Hall, 1998.

MENE93 Menezes, A. *Elliptic Curve Public Key Cryptosystems*. Boston: Kluwer Academic Publishers, 1993.

NECH92 Nechvatal, J. "Public Key Cryptography." In [SIMM92].

SALO96 Salomaa, A. *Public-Key Cryptography*. New York: Springer-Verlag, 1996.

SIMM92 Simmons, G., ed. *Contemporary Cryptology: The Science of Information Integrity*. Piscataway, NJ: IEEE Press, 1992.

STIN95 Stinson, D. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 1995.

## 6.7 PROBLEMS

- 6.1** Prior to the discovery of any specific public-key schemes, such as RSA, an existence proof was developed whose purpose was to demonstrate that public-key encryption is possible in theory. Consider the functions  $f_1(x_1) = z_1$ ;  $f_2(x_2, y_2) = z_2$ ;  $f_3(x_3, y_3) = z_3$ , where all values are integers with  $1 \leq x_i, y_i, z_i \leq N$ . Function  $f_1$  can be represented by a vector **M1** of length  $N$ , in which the  $k$ th entry is the value of  $f_1(k)$ . Similarly,  $f_2$  and  $f_3$  can be represented by  $N \times N$  matrices **M2** and **M3**. The intent is to represent the

encryption/decryption process by table lookups for tables with very large values of  $N$ . Such tables would be impractically huge but could, in principle, be constructed. The scheme works as follows: construct **M1** with a random permutation of all integers between 1 and  $N$ ; that is, each integer appears exactly once in **M1**. Construct **M2** so that each row contains a random permutation of the first  $N$  integers. Finally, fill in **M3** to satisfy the following condition:

$$f_3(f_2(f_1(k), p), k) = p \quad \text{for all } k, p \text{ with } 1 \leq k, p \leq N$$

In words:

1. **M1** takes an input  $k$  and produces an output  $x$ .
2. **M2** takes inputs  $x$  and  $p$  and gives output  $z$ .
3. **M3** takes inputs  $z$  and  $k$  and produces  $p$ .

The three tables, once constructed, are made public.

- a. It should be clear that it is possible to construct **M3** to satisfy the preceding condition. As an example, fill in **M3** for the following simple case:

<b>M1</b> =	5	<b>M2</b> =	5	2	3	4	1	<b>M3</b> =				
	4		4	2	5	1	3					
	2		1	3	2	4	5					
	3		3	1	4	2	5					
	1		2	5	3	4	1					

Convention: the  $i$ th element of **M1** corresponds to  $k = i$ . The  $i$ th row of **M2** corresponds to  $x = i$ ; the  $j$ th column of **M2** corresponds to  $p = j$ . The  $i$ th row of **M3** corresponds to  $z = i$ ; the  $j$ th column of **M3** corresponds to  $k = j$ .

- b. Describe the use of this set of tables to perform encryption and decryption between two users.
  - c. Argue that this is a secure scheme.
- 6.2 Perform encryption and decryption using the RSA algorithm, as in Figure 6.6, for the following:
- a.  $p = 3$ ;  $q = 11$ ,  $d = 7$ ;  $M = 5$
  - b.  $p = 5$ ;  $q = 11$ ,  $e = 3$ ;  $M = 9$
  - c.  $p = 7$ ;  $q = 11$ ,  $e = 17$ ;  $M = 8$
  - d.  $p = 11$ ;  $q = 13$ ,  $e = 11$ ;  $M = 7$
  - e.  $p = 17$ ;  $q = 31$ ,  $e = 7$ ;  $M = 2$ . *Hint:* Decryption is not as hard as you think; use some finesse.
- 6.3 In a public-key system using RSA, you intercept the ciphertext  $C = 10$  sent to a user whose public key is  $e = 5$ ,  $n = 35$ . What is the plaintext  $M$ ?
- 6.4 In an RSA system, the public key of a given user is  $e = 31$ ,  $n = 3599$ . What is the private key of this user?
- 6.5 In using the RSA algorithm, if a small number of repeated encodings give back the plaintext, what is the likely cause?
- 6.6 Suppose we have a set of blocks encoded with the RSA algorithm and we do not have the private key. Assume  $n = pq$ ,  $e$  is the public key. Suppose also that someone tells us they know one of the plaintext blocks has a common factor with  $n$ . Does this help us in any way?
- 6.7 In the RSA public-key encryption scheme, each user has an public key,  $e$ , and a private key,  $d$ . Suppose Bob leaks his private key. Rather than generating a new modulus, he decides to generate a new public and a new private key. Is this safe?
- 6.8 "I want to tell you, Holmes," Dr. Watson's voice was enthusiastic, "that your recent activities in network security have increased my interest in cryptography. And just yesterday I found a way to make one-time pad encryption practical."  
 "Oh, really?" Holmes's face lost its sleepy look. "So you have found a way to generate cryptographically strong sequences in a deterministic fashion, have you?"



"Yes, Holmes. The idea is really simple. For a given one-way function  $F$ , I generate a long pseudorandom sequence of elements by applying  $F$  to some standard sequence of arguments. The cryptanalyst is assumed to know  $F$  and the general nature of the sequence, which may be as simple as  $S, S + 1, S + 2, \dots$ , but not secret  $S$ . And due to the one-way nature of  $F$  no one is able to extract  $S$  given  $F(S + i)$  for some  $i$ ; thus even if he somehow obtains a certain segment of the sequence, he will not be able to determine the rest."

"I am afraid, Watson, that your proposal isn't without flaws and at least it needs some additional conditions to be satisfied by  $F$ . Let's consider, for instance, that the RSA encryption function, that is,  $F(M) = M^K \bmod N$ ,  $K$ , is secret. This function is believed to be one-way, but I wouldn't recommend its use, for example, to the sequence  $M = 2, 3, 4, 5, 6, \dots$ "

"But why, Holmes?" Dr. Watson apparently didn't understand. "Why do you think that the resulting sequence  $2^K \bmod N, 3^K \bmod N, 4^K \bmod N, \dots$  is not appropriate for one-time pad encryption if  $K$  is kept secret?"

"Because it is—at least partially—predictable, dear Watson, even if  $K$  is kept secret. You have said that the cryptanalyst is assumed to know  $F$  and the general nature of the sequence. Now let's assume that he will obtain somehow a short segment of the output sequence. In crypto circles this assumption is generally considered to be a viable one. And for this output sequence, knowledge of just the first two elements will allow him to predict quite a lot of the next elements of the sequence, even if not all of them, thus this sequence can't be considered to be cryptographically strong. And with the knowledge of a longer segment he could predict even more of the next elements of the sequence. Look, knowing the general nature of the sequence and its first two elements  $2^K \bmod N$  and  $3^K \bmod N$  you can easily compute its following elements ..."

**6.9** Show how RSA can be represented by matrices **M1**, **M2**, and **M3** of Problem 6.1.

**6.10** Consider the following scheme:

1. Pick an odd number,  $E$ .
2. Pick two prime numbers,  $P$  and  $Q$ , where  $(P - 1)(Q - 1) - 1$  is evenly divisible by  $E$ .
3. Multiply  $P$  and  $Q$  to get  $N$ .
4. Calculate  $D = \frac{(P-1)(Q-1)(E-1) + 1}{E}$ .

Is this scheme equivalent to RSA? Show why or why not.

**6.11** Consider the following scheme by which B encrypts a message for A.

1. A chooses two large primes  $P$  and  $Q$  that are also relatively prime to  $(P - 1)$  and  $(Q - 1)$ .
  2. A publishes  $N = PQ$  as its public key.
  3. A calculates  $P'$  and  $Q'$  such that  $PP' \equiv 1 \pmod{Q - 1}$  and  $QQ' \equiv 1 \pmod{P - 1}$ .
  4. B encrypts message  $M$  as  $C = M^N \pmod{N}$ .
  5. A finds  $M$  by solving  $M \equiv C^{P'} \pmod{Q}$  and  $M = C^{Q'} \pmod{P}$ .
- a. Explain how this scheme works.
  - b. How does it differ from RSA?
  - c. Is there any particular advantage to RSA compared to this scheme?
  - d. Show how this scheme can be represented by matrices **M1**, **M2**, and **M3** of Problem 6.1.

**6.12** "This is a very interesting case, Watson," Holmes said. "The young man loves a girl and she loves him too. However, her father is a strange fellow who insists that his would-be son-in-law must design a simple and secure protocol for an appropriate public-key cryptosystem he could use in his company's computer network. The young man came up with the following protocol for communication between two parties, for example, user A wishing to send message  $M$  to user B (messages exchanged are in the format [sender's name, text, receiver's name]):

1. A sends B (A,  $E_{K_{Ub}}[M, A]$ , B).
2. B acknowledges receipt by sending to A (B,  $E_{K_{Ua}}[M, B]$ , A).

"You can see that the protocol is really simple. But the girl's father claims that the young man has not satisfied his call for a simple protocol, because the proposal contains a certain redundancy and can be further simplified to the following:

1. A sends B ( $A, E_{K_{Ub}}[M], B$ ).
2. B acknowledges receipt by sending to A ( $B, E_{K_{Ua}}[M], A$ ).

"On the basis of that, the girl's father refuses to allow his daughter to marry the young man, thus making them both unhappy. The young man was just here to ask me for help."

"Hmm, I don't see how you can help him," Watson was visibly unhappy with the idea that the sympathetic young man has to lose his love.

"Well, I think I could help. You know, Watson, redundancy is sometimes good to ensure the security of protocol. Thus, the simplification the girl's father has proposed could make the new protocol vulnerable to an attack the original protocol was able to resist," mused Holmes. "Yes, it is so, Watson! Look, all an adversary needs is to be one of the users of the network and to be able to intercept messages exchanged between A and B. Being a user of the network, he has his own public encryption key and is able to send his own messages to A or to B and to receive theirs. With the help of the simplified protocol, he could then obtain message M user A has previously sent to B using the following procedure ..."

- 6.13** Here is another realization of the fast exponentiation algorithm. Demonstrate that it is equivalent to the one in Figure 6.7.

1.  $d \leftarrow 1; T \leftarrow a; E \leftarrow b$
2. **if** odd( $e$ ) **then**  $d \leftarrow d \times T$
3.  $E \leftarrow \lfloor E/2 \rfloor$
4.  $T \leftarrow T \times T$
5. **if**  $E > 0$  **then goto** 2
6. **output**  $d$

- 6.14** Consider a Diffie-Hellman scheme with a common prime  $q = 11$  and a primitive root  $\alpha = 2$ .

- a. If user A has public key  $Y_A = 9$ , what is A's private key  $X_A$ ?
- b. If user B has public key  $Y_B = 3$ , what is the shared secret key  $K$ ?

- 6.15** "But," said Dr. Watson, "your clients use Diffie-Hellman key exchange protocol in their network. It is based on discrete logarithm, and this is known to be a hard problem, isn't it?"

"Yes, Watson," nodded Holmes, "for appropriate choice of parameters the discrete logarithm problem is really hard. My clients know that and that's why they opted for this method of key distribution. Unfortunately their security consultants didn't realize that an active adversary might often be more successful than the passive one. An adversary also knows that he can't solve a discrete logarithm problem in a reasonable time, thus he has to try something else. And because I am sure Moriarty himself is interested in my clients' communications, I must suppose some kind of active attack on their network. Moriarty would never stay passive, Watson."

"Do you think, Holmes," Dr. Watson added, surprised, "that Moriarty could find a way to break the Diffie-Hellman key exchange scheme?"

"Oh, it is not so hard, Watson," smiled Holmes. "All that Moriarty needs is to place himself somewhere in the communication path to be able not only to intercept but also to change all the messages. I am sure this is completely within Moriarty's abilities. Now in this position he will ..."

- 6.16** Consider the elliptic curve  $E_{11}(1, 6)$ ; that is, the curve is defined by  $y^2 = x^3 + x + 6$  with a modulus of  $p = 11$ . Determine all of the points in  $E_{11}(1, 6)$ . *Hint:* Start by calculating the right-hand side of the equation for all values of  $x$ .

- 6.17** For  $E_{11}(1, 6)$ , consider the point  $G = (2, 7)$ . Compute the multiples of  $G$  from  $2G$  through  $13G$ .

- 6.18** This problem performs elliptic curve encryption/decryption using the scheme outlined in Section 6.5. The cryptosystem parameters are  $E_{11}(1, 6)$  and  $G = (2, 7)$ . B's secret key is  $n_B = 7$ .

- a. Find B's public key  $P_B$ .
  - b. A wishes to encrypt the message  $P_m = (10, 9)$  and chooses the random value  $k = 3$ . Determine the ciphertext  $C_m$ .
  - c. Show the calculation by which B recovers  $P_m$  from  $C_m$ .
- 6.19** In 1985, T. ElGamal announced a public-key scheme based on discrete logarithms, closely related to the Diffie-Hellman technique. As with Diffie-Hellman, the global elements of the ElGamal scheme are a prime number  $q$  and  $\alpha$ , a primitive root of  $q$ . A user A selects a private key  $X_A$  and calculates a public key  $Y_A$  as in Diffie-Hellman. User A encrypts a plaintext  $M < q$  intended for user B as follows:
1. Choose a random integer  $k$  such that  $1 \leq k \leq q - 1$ .
  2. Compute  $K = (Y_B)^k \pmod{q}$ .
  3. Encrypt  $M$  as the pair of integers  $(C_1, C_2)$  where  $C_1 = \alpha^k \pmod{q}$   $C_2 = KM \pmod{q}$
- User B recovers the plaintext as follows:
1. Compute  $K = (C_1)^{X_B} \pmod{q}$ .
  2. Compute  $M = (C_2 K^{-1}) \pmod{q}$ .
- Show that the system works; that is, show that the decryption process does recover the plaintext.
- 6.20** Consider an ElGamal scheme with a common prime  $q = 71$  and a primitive root  $\alpha = 7$ .
- a. If B has public key  $Y_B = 3$  and A chose the random integer  $k = 2$ , what is the ciphertext of  $M = 30$ ?
  - b. If A now chooses a different value of  $k$ , so that the encoding of  $M = 30$  is  $C = (59, C_2)$ , what is the integer  $C_2$ ?
- 6.21** Improve on algorithm P1 in appendix 6A.
- a. Develop an algorithm that requires  $2n$  multiplications and  $n + 1$  additions. *Hint:*  $x^{i+1} = x^i \times x$ .
  - b. Develop an algorithm that requires only  $n + 1$  multiplications and  $n + 1$  additions. *Hint:*  $P(x) = a_0 + x \times q(x)$ , where  $q(x)$  is a polynomial of degree  $(n - 1)$ .

## APPENDIX 6A THE COMPLEXITY OF ALGORITHMS

The central issue in assessing the resistance of an encryption algorithm to cryptanalysis is the amount of time that a given type of attack will take. Typically, one cannot be sure that one has found the most efficient attack algorithm. The most that one can say is that for a particular algorithm, the level of effort for an attack is of a particular order of magnitude. One can then compare that order of magnitude to the speed of current or predicted processors to determine the level of security of a particular algorithm.

A common measure of the efficiency of an algorithm is its time complexity. We define the time complexity of an algorithm to be  $f(n)$  if, for all  $n$  and all inputs of length  $n$ , the execution of the algorithm takes at most  $f(n)$  steps. Thus, for a given size of input and a given processor speed, the time complexity is an upper bound on the execution time.

There are several ambiguities here. First, the definition of a step is not precise. A step could be a single operation of a Turing machine, a single processor machine instruction, a single high-level language machine instruction, and so on. However, these various definitions of step should all be related by simple multiplicative constants. For very large values of  $n$ , these constants are not important. What is important is how fast the relative execution time is growing. For example, if we are concerned about whether to use 50-digit ( $n = 10^{50}$ ) or 100-digit ( $n = 10^{100}$ )

keys for RSA, it is not necessary (or really possible) to know exactly how long it would take to break each size of key. Rather, we are interested in ballpark figures for level of effort and in knowing how much extra relative effort is required for the larger key size.

A second issue is that, generally speaking, we cannot pin down an exact formula for  $f(n)$ . We can only approximate it. But again, we are primarily interested in the rate of change of  $f(n)$  as  $n$  becomes very large.

There is a standard mathematical notation, known as the “big-O” notation, for characterizing the time complexity of algorithms that is useful in this context. The definition is as follows:  $f(n) = O(g(n))$  if and only if there exist two numbers  $a$  and  $M$  such that

$$|f(n)| \leq a \times |g(n)|, \quad n \geq M \quad (6.2)$$

An example helps clarify the use of this notation. Suppose we wish to evaluate a general polynomial of the form

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

The following simple-minded algorithm is from [POHL81]:

```

algorithm P1;
  n, i, j: integer; x, polyval: real;
  a, S: array [0..100] of real;
  begin
    read(x, n);
    for i := 0 upto n do
      begin
        S[i] := 1; read(a[i]);
        for j := 1 upto i do S[i] := x × S[i];
        S[i] := a[i] × S[i]
      end;
    polyval := 0;
    for i := 0 upto n do polyval := polyval + S[i];
    write ('value at', x, 'is', polyval)
  end.

```

In this algorithm, each subexpression is evaluated separately. Each  $S[i]$  requires  $(i + 1)$  multiplications:  $i$  multiplications to compute  $S[i]$  and one to multiply by  $a[i]$ . Computing all  $n$  terms requires

$$\sum_{i=0}^n (i + 1) = \frac{(n + 2)(n + 1)}{2}$$

multiplications. There are also  $(n + 1)$  additions, which we can ignore relative to the much larger number of multiplications. Thus, the time complexity of this algorithm is  $f(n) = (n + 2)(n + 1)/2$ . We now show that  $f(n) = O(n^2)$ . From the definition of Equation (6.2), we want to show that for  $a = 1$  and  $M = 4$ , the relationship holds for  $g(n) = n^2$ . We do this by induction on  $n$ . The relationship holds for  $n = 4$  because

$(4 + 2)(4 + 1)/2 = 15 < 4^2 = 16$ . Now assume that it holds for all values of  $n$  up to  $k$ ; that is,  $(k + 2)(k + 1)/2 < k^2$ . Then, with  $n = k + 1$ ,

$$\begin{aligned}\frac{(n + 2)(n + 1)}{2} &= \frac{(k + 3)(k + 2)}{2} \\ &= \frac{(k + 2)(k + 1)}{2} + k + 2 \\ &\leq k^2 + k + 2 \\ &\leq k^2 + 2k + 2 = (k + 1)^2 = n^2\end{aligned}$$

Therefore, the result is true for  $n = k + 1$ .

In general, the big-O notation makes use of the term that grows the fastest. For example,

1.  $O(ax^7 + 3x^3 + \sin(x)) = O(ax^7) = O(x^7)$
2.  $O(e^n + an^{10}) = O(e^n)$
3.  $O(n! + n^{50}) = O(n!)$

There is much more to the big-O notation, with fascinating ramifications. For the interested reader, one of the best accounts is in [GRAH94].

An algorithm with an input of size  $n$  is said to be

- **Linear:** if the running time is  $O(n)$
- **Polynomial:** if the running time is  $O(n^t)$  for some constant  $t$
- **Exponential:** if the running time is  $O(t^{h(n)})$  for some constant  $t$  and polynomial  $h(n)$

Generally, a problem that can be solved in polynomial time is considered feasible, whereas anything worse than polynomial time, especially exponential time, is considered infeasible. But you must be careful with these terms. First, if the size of the input is small enough, even very complex algorithms become feasible. Suppose, for example, that you have a system that can execute  $10^{12}$  operations per unit time. Table 6.6 shows the size of the size of input that can be handled in one time unit for algorithms of various complexities. For algorithms of exponential or factorial time, only very small inputs can be accommodated.

**Table 6.6** Level of Effort for Various Levels of Complexity

Complexity	Size	Operations
$\log_2 n$	$2^{10^{12}} = 10^{3 \times 10^{11}}$	$10^{12}$
$n$	$10^{12}$	$10^{12}$
$n^2$	$10^6$	$10^{12}$
$n^6$	$10^2$	$10^{12}$
$2^n$	28	$10^{12}$
$n!$	15	$10^{12}$

The second thing to be careful about is the way in which the input is characterized. For example, the complexity of cryptanalysis of an encryption algorithm can be characterized equally well in terms of the number of possible keys or the length of the key. For DES, for example, the number of possible keys is  $2^{56}$ , and the length of the key is 56 bits. If we consider a single encryption to be a “step” and the number of possible keys to be  $N = 2^n$ , then the time complexity of the algorithm is linear in terms of the number of keys  $O(N)$  but exponential in terms of the length of the key  $O(2^n)$ .

# CHAPTER 7

## INTRODUCTION TO NUMBER THEORY

*The Devil said to Daniel Webster: "Set me a task I can't carry out, and I'll give you anything in the world you ask for."*

*Daniel Webster: "Fair enough. Prove that for  $n$  greater than 2, the equation  $a^n + b^n = c^n$  has no non-trivial solution in the integers."*

*They agreed on a three-day period for the labor, and the Devil disappeared.*

*At the end of three days, the Devil presented himself, haggard, jumpy, biting his lip. Daniel Webster said to him, "Well, how did you do at my task? Did you prove the theorem?"*

*"Eh? No ... no, I haven't proved it."*

*"Then I can have whatever I ask for? Money? The Presidency?"*

*"What? Oh, that—of course. But listen! If we could just prove the following two lemmas—"*

**—The Mathematical Magpie, Clifton Fadiman**

A number of concepts from number theory are essential in the design of public-key cryptographic algorithms. This chapter provides an overview of the concepts referred to in other chapters. The reader familiar with these topics can safely skip this chapter.

The concepts and techniques of number theory are quite abstract, and it is often difficult to grasp them intuitively without examples [RUBI97b]. Accordingly, this chapter is liberally endowed with examples, each of which is highlighted in a shaded box.

## 7.1 PRIME AND RELATIVELY PRIME NUMBERS<sup>1</sup>

A central concern of number theory are prime numbers. Indeed, whole books have been written on the subject (e.g., [RIBE96]). In this section we provide an overview relevant to the concerns of this book.

### Divisors

We say that  $b \neq 0$  divides  $a$  if  $a = mb$  for some  $m$ , where  $a$ ,  $b$ , and  $m$  are integers. That is,  $b$  divides  $a$  if there is no remainder on division. The notation  $b|a$  is commonly used to mean  $b$  divides  $a$ . Also, if  $b|a$ , we say that  $b$  is a *divisor* of  $a$ .

The positive divisors of 24 are 1, 2, 3, 4, 6, 8, 12, and 24.

The following relations hold:

- If  $a|1$ , then  $a = \pm 1$ .
- If  $a|b$  and  $b|a$ , then  $a = \pm b$ .
- Any  $b \neq 0$  divides 0.
- If  $b|g$  and  $b|h$ , then  $b|(mg + nh)$  for arbitrary integers  $m$  and  $n$ .

To see this last point, note that

if  $b|g$ , then  $g$  is of the form  $g = b \times g_1$  for some integer  $g_1$   
 if  $b|h$ , then  $h$  is of the form  $h = b \times h_1$  for some integer  $h_1$

so

$$mg + nh = mbg_1 + nbh_1 = b \times (mg_1 + nh_1)$$

and therefore  $b$  divides  $mg + nh$ .

$b = 7$ ;  $g = 14$ ;  $h = 63$ ;  $m = 3$ ;  $n = 2$   
 $7|14$  and  $7|63$ . To show  $7|(3 \times 14 + 2 \times 63)$   
 We have  $(3 \times 14 + 2 \times 63) = 7(3 \times 2 + 2 \times 9)$   
 And it is obvious that  $7|(7(3 \times 2 + 2 \times 9))$

### Prime Numbers

An integer  $p > 1$  is a prime number if its only divisors are  $\pm 1$  and  $\pm p$ . Prime numbers play a critical role in number theory and in the techniques discussed in this chapter.

Table 7.1 shows the primes under 2000.

<sup>1</sup>In this section, unless otherwise noted, we deal only with the nonnegative integers. The use of negative integers would introduce no essential differences.



P209

Table 7.1 Primes under 2000

2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59	61	67	71	73	79	83	89	97
101	103	107	109	113	127	131	137	139	149	151	157	163	167	173	179	181	191	193	197	199				
211	223	227	229	233	239	241	251	257	263	269	271	277	281	283	293									
307	311	313	317	331	337	347	349	353	359	367	373	379	383	389	397									
401	409	419	421	431	433	439	443	449	457	461	463	467	479	487	491	499								
503	509	521	523	541	547	557	563	569	571	577	587	593	599											
601	607	613	617	619	631	641	643	647	653	659	661	673	677	683	691									
701	709	719	727	733	739	743	751	757	761	769	773	787	797											
809	811	821	823	827	829	839	853	857	859	863	877	881	883	887										
907	911	919	929	937	941	947	953	967	971	977	983	991	997											
1009	1013	1019	1021	1031	1033	1039	1049	1051	1061	1063	1069	1087	1091	1093	1097									
1103	1109	1117	1123	1129	1151	1153	1163	1171	1181	1187	1193													
1201	1213	1217	1223	1229	1231	1237	1249	1259	1277	1279	1283	1289	1291	1297										
1301	1303	1307	1319	1321	1327	1361	1367	1373	1381	1399														
1409	1423	1427	1429	1433	1439	1447	1451	1453	1459	1471	1481	1483	1487	1489	1493	1499								
1511	1523	1531	1543	1549	1553	1559	1567	1571	1579	1583	1597													
1601	1607	1609	1613	1619	1621	1627	1637	1657	1663	1667	1669	1693	1697	1699										
1709	1721	1723	1733	1741	1747	1753	1759	1777	1783	1787	1789													
1801	1811	1823	1831	1847	1861	1867	1871	1873	1877	1879	1889													
1901	1907	1913	1931	1933	1949	1951	1973	1979	1987	1993	1997	1999												

Any integer  $a > 1$  can be factored in a unique way as

$$a = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_t^{\alpha_t}$$

where  $p_1 > p_2 > \dots > p_t$  are prime numbers and where each  $\alpha_i > 0$ .

$$91 = 7 \times 13; 11011 = 7 \times 11^2 \times 13$$

It is useful for what follows to cast this another way. If  $P$  is the set of all prime numbers, then any positive integer can be written uniquely in the following form:

$$a = \prod_p p^{a_p} \quad \text{where each } a_p \geq 0$$

The right-hand side is the product over all possible prime numbers  $p$ ; for any particular value of  $a$ , most of the exponents  $a_p$  will be 0.

$$3600 = 2^4 \times 3^2 \times 5^2$$

The value of any given positive integer can be specified by simply listing all the nonzero exponents in the foregoing formulation.

The integer 12 is represented by  $\{a_2 = 2, a_3 = 1\}$ .

The integer 18 is represented by  $\{a_2 = 1, a_3 = 2\}$ .

Multiplication of two numbers is equivalent to adding the corresponding exponents:

$$k = mn \rightarrow k_p = m_p + n_p \quad \text{for all } p$$

$$k = 12 \times 18 = 216$$

$$k_2 = 2 + 1 = 3; k_3 = 1 + 2 = 3$$

$$216 = 2^3 \times 3^3$$

What does it mean, in terms of these prime factors, to say that  $a|b$ ? Any integer of the form  $p^k$  can be divided only by an integer that is of a lesser or equal power of the same prime number,  $p^j$  with  $j \leq k$ . Thus, we can say

$$a|b \rightarrow a_p \leq b_p \quad \text{for all } p$$

$$a = 12; b = 36; 12|36; 12 = 2^2 \times 3; 36 = 2^2 \times 3^2$$

$$a_2 = 2 = b_2$$

$$a_3 = 1 \leq 2 = b_3$$

### Relatively Prime Numbers

We will use the notation  $\gcd(a, b)$  to mean the **greatest common divisor** of  $a$  and  $b$ . The positive integer  $c$  is said to be the greatest common divisor of  $a$  and  $b$  if

1.  $c$  is a divisor of  $a$  and of  $b$ ;
2. any divisor of  $a$  and  $b$  is a divisor of  $c$ .

An equivalent definition is the following:

$$\gcd(a, b) = \max\{k, \text{ such that } k|a \text{ and } k|b\}$$

Because we require that the greatest common divisor be positive,  $\gcd(a, b) = \gcd(a, -b) = \gcd(-a, b) = \gcd(-a, -b)$ . In general,  $\gcd(a, b) = \gcd(|a|, |b|)$ .

$$\gcd(60, 24) = \gcd(60, -24) = 12$$

Also, because all nonzero integers divide 0, we have  $\gcd(a, 0) = |a|$ .

It is easy to determine the greatest common divisor of two positive integers if we express each integer as the product of primes.

$$\begin{aligned} 300 &= 2^2 \times 3^1 \times 5^2 \\ 18 &= 2^1 \times 3^2 \\ \gcd(18, 300) &= 2^1 \times 3^1 \times 5^0 = 6 \end{aligned}$$

In general,

$$k = \gcd(a, b) \rightarrow k_p = \min(a_p, b_p) \quad \text{for all } p$$

Determining the prime factors of a large number is no easy task, so the preceding relationship does not directly lead to a way of calculating the greatest common divisor. We return to this topic in Section 7.5.

The integers  $a$  and  $b$  are relatively prime if they have no prime factors in common, that is, if their only common factor is 1. This is equivalent to saying that  $a$  and  $b$  are relatively prime if  $\gcd(a, b) = 1$ .

8 and 15 are relatively prime because the divisors of 8 are 1, 2, 4, and 8, and the divisors of 15 are 1, 3, 5, and 15, so 1 is the only number on both lists.

## 7.2 MODULAR ARITHMETIC

Given any positive integer  $n$  and any integer  $a$ , if we divide  $a$  by  $n$ , we get a quotient  $q$  and a remainder  $r$  that obey the following relationship:

$$a = qn + r \quad 0 \leq r < n; q = \lfloor a/n \rfloor$$

where  $\lfloor x \rfloor$  is the largest integer less than or equal to  $x$ .

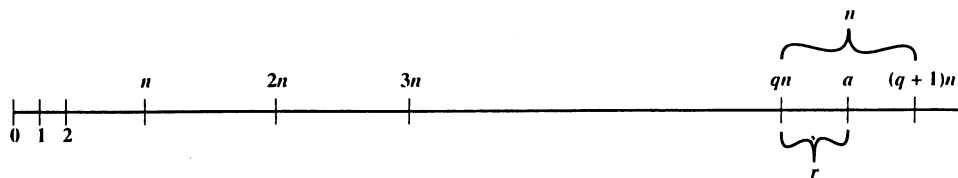


Figure 7.1 The Relationship  $a = qn + r$ ;  $0 \leq r < n$ .

Figure 7.1 demonstrates that, given  $a$  and positive  $n$ , it is always possible to find  $q$  and  $r$  that satisfy the preceding relationship. Represent the integers on the number line;  $a$  will fall somewhere on that line (positive  $a$  is shown, a similar demonstration can be made for negative  $a$ ). Starting at 0, proceed to  $n$ ,  $2n$ , up to  $qn$  such that  $qn \leq a$  and  $(q+1)n > a$ . The distance from  $qn$  to  $a$  is  $r$ , and we have found the unique values of  $q$  and  $r$ . The remainder  $r$  is often referred to as a **residue**.

$$\begin{array}{llll} a = 11; & n = 7; & 11 = 1 \times 7 + 4; & r = 4 \\ a = -11; & n = 7; & -11 = (-2) \times 7 + 3; & r = 3 \end{array}$$

If  $a$  is an integer and  $n$  is a positive integer, we define  $a \bmod n$  to be the remainder when  $a$  is divided by  $n$ . Thus, for any integer  $a$ , we can always write

$$a = \lfloor a/n \rfloor \times n + (a \bmod n)$$

$$11 \bmod 7 = 4; \quad -11 \bmod 7 = 3$$

Two integers  $a$  and  $b$  are said to be **congruent modulo  $n$**  if  $(a \bmod n) = (b \bmod n)$ . This is written  $a \equiv b \bmod n$ .

$$73 \equiv 4 \bmod 23; \quad 21 \equiv -9 \bmod 10$$

Note that if  $a \equiv 0 \bmod n$ , then  $n|a$ .

The modulo operator has the following properties:

1.  $a \equiv b \bmod n$  if  $n|(a - b)$ .
2.  $(a \bmod n) = (b \bmod n)$  implies  $a \equiv b \bmod n$ .
3.  $a \equiv b \bmod n$  implies  $b \equiv a \bmod n$ .
4.  $a \equiv b \bmod n$  and  $b \equiv c \bmod n$  imply  $a \equiv c \bmod n$ .

To demonstrate the first point, if  $n|(a - b)$  then  $(a - b) = kn$  for some  $k$ . So we can write  $a = b + kn$ . Therefore,  $(a \bmod n) = (\text{remainder when } b + kn \text{ is divided by } n) = (\text{remainder when } b \text{ is divided by } n) = (b \bmod n)$ .

$$\begin{array}{lll}
23 \equiv 8 \pmod{5} & \text{because} & 23 - 8 = 15 = 5 \times 3 \\
-11 \equiv 5 \pmod{8} & \text{because} & -11 - 5 = -16 = 8 \times (-2) \\
81 \equiv 0 \pmod{27} & \text{because} & 81 - 0 = 81 = 27 \times 3
\end{array}$$

The remaining points are as easily proved.

### Modular Arithmetic Operations

Note that, by definition (Figure 7.1), the  $(\text{mod } n)$  operator maps all integers into the set of integers  $\{0, 1, \dots, (n - 1)\}$ . This suggests the question, Can we perform arithmetic operations within the confines of this set? It turns out that we can; the technique is known as **modular arithmetic**.

Modular arithmetic exhibits the following properties:

1.  $[(a \text{ mod } n) + (b \text{ mod } n)] \text{ mod } n = (a + b) \text{ mod } n$
2.  $[(a \text{ mod } n) - (b \text{ mod } n)] \text{ mod } n = (a - b) \text{ mod } n$
3.  $[(a \text{ mod } n) \times (b \text{ mod } n)] \text{ mod } n = (a \times b) \text{ mod } n$

We demonstrate the first property. Define  $(a \text{ mod } n) = r_a$  and  $(b \text{ mod } n) = r_b$ . Then we can write  $a = r_a + jn$  for some integer  $j$  and  $b = r_b + kn$  for some integer  $k$ . Then

$$\begin{aligned}
(a + b) \text{ mod } n &= (r_a + jn + r_b + kn) \text{ mod } n \\
&= (r_a + r_b + (k + j)n) \text{ mod } n \\
&= (r_a + r_b) \text{ mod } n \\
&= [(a \text{ mod } n) + (b \text{ mod } n)] \text{ mod } n
\end{aligned}$$

The remaining properties are as easily proved. Here are examples of the three properties:

$$\begin{aligned}
&11 \text{ mod } 8 = 3; \quad 15 \text{ mod } 8 = 7 \\
&[(11 \text{ mod } 8) + (15 \text{ mod } 8)] \text{ mod } 8 = 10 \text{ mod } 8 = 2 \\
&(11 + 15) \text{ mod } 8 = 26 \text{ mod } 8 = 2 \\
&[(11 \text{ mod } 8) - (15 \text{ mod } 8)] \text{ mod } 8 = -4 \text{ mod } 8 = 4 \\
&(11 - 15) \text{ mod } 8 = -4 \text{ mod } 8 = 4 \\
&[(11 \text{ mod } 8) \times (15 \text{ mod } 8)] \text{ mod } 8 = 21 \text{ mod } 8 = 5 \\
&(11 \times 15) \text{ mod } 8 = 165 \text{ mod } 8 = 5
\end{aligned}$$

Exponentiation is performed by repeated multiplication, as in ordinary arithmetic. (We have more to say about exponentiation in Section 7.7.)

To find  $11^7 \bmod 13$ , we can proceed as follows:

$$11^2 = 121 \equiv 4 \bmod 13$$

$$11^4 \equiv 4^2 \equiv 3 \bmod 13$$

$$11^7 \equiv 11 \times 4 \times 3 \equiv 132 \equiv 2 \bmod 13$$

Thus, the rules for ordinary arithmetic involving addition, subtraction, and multiplication carry over into modular arithmetic.

Table 7.2 provides an illustration of modular addition and multiplication modulo 8. Looking at addition, the results are straightforward and there is a regular pattern to the matrix. Also, as in ordinary addition, there is an additive inverse, or negative, to each number in modular arithmetic. In this case, the negative of a number  $x$  is the number  $y$  such that  $x + y \equiv 0 \bmod 8$ . To find the additive inverse of a number in the left-hand column, scan across the corresponding row of the matrix to find the value 0; the number at the top of that column is the additive inverse; thus  $2 + 6 = 0 \bmod 8$ . Similarly, the entries in the multiplication table are straightforward. In ordinary arithmetic, there is a multiplicative inverse, or reciprocal, to each number. In modular arithmetic mod 8, the multiplicative inverse of  $x$  is the number  $y$  such that  $x \times y \equiv 1 \bmod 8$ . Now, to find the multiplicative inverse of a number from the multiplication table, scan across the matrix in the row for that number to find the value 1; the number at the top of that column is the multiplicative inverse; thus  $3 \times 3 = 1 \bmod 8$ . Note that not all numbers mod 8 have a multiplicative inverse; we will discuss this later.

**Table 7.2** Arithmetic Modulo 8

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

(a) Addition modulo 8

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

(b) Multiplication modulo 8

### Properties of Modular Arithmetic

Define the set  $Z_n$  as the set of nonnegative integers less than  $n$ :

$$Z_n = \{0, 1, \dots, (n - 1)\}$$

This is referred to as the set of residues modulo  $n$ . If we perform modular arithmetic within this set, the following properties hold for integers in  $Z_n$ :

Property	Expression
Commutative laws	$(w + x) \bmod n = (x + w) \bmod n$ $(w \times x) \bmod n = (x \times w) \bmod n$
Associative laws	$[(w + x) + y] \bmod n = [w + (x + y)] \bmod n$ $[(w \times x) \times y] \bmod n = [w \times (x \times y)] \bmod n$
Distributive law	$[w \times (x + y)] \bmod n = [(w \times x) + (w \times y)] \bmod n$
Identities	$(0 + w) \bmod n = w \bmod n$ $(1 \times w) \bmod n = w \bmod n$
Additive inverse ( $-w$ )	For each $w \in Z_n$ , there exists a $z$ such that $w + z \equiv 0 \bmod n$

There is one peculiarity of modular arithmetic that sets it apart from ordinary arithmetic. First, observe that, as in ordinary arithmetic, we can write the following:

$$\text{if } (a + b) \equiv (a + c) \bmod n \quad \text{then} \quad b \equiv c \bmod n \quad (7.1)$$

$$(5 + 23) \equiv (5 + 7) \bmod 8; \quad 23 \equiv 7 \bmod 8$$

Equation (7.1) is consistent with the existence of an additive inverse. Adding the additive inverse of  $a$  to both sides of Equation (7.1), we have

$$\begin{aligned} ((-a) + a + b) &\equiv ((-a) + a + c) \bmod n \\ b &\equiv c \bmod n \end{aligned}$$

However, the following statement is true only with the attached condition:

$$\text{if } (a \times b) \equiv (a \times c) \bmod n \quad \text{then} \quad b \equiv c \bmod n \quad \text{if } a \text{ is relatively prime to } n \quad (7.2)$$

To see this, consider an example in which the condition does not hold:

$$\begin{aligned} 6 \times 3 &= 18 \equiv 2 \bmod 8 \\ 6 \times 7 &= 42 \equiv 2 \bmod 8 \end{aligned}$$

Yet  $3 \not\equiv 7 \bmod 8$ .

The reason for this strange result is that for any general modulus  $n$ , a multiplier  $a$ , when applied in turn to the numbers 0 through  $(n - 1)$ , will fail to produce a complete set of residues if  $a$  and  $n$  have any factors in common.

With  $a = 6$  and  $n = 8$ :

$Z_8$	0	1	2	3	4	5	6	7
Multiply by 6	0	6	12	18	24	30	36	42
Residues	0	6	4	2	0	6	4	2

Because we do not have a complete set of residues when multiplying by 6, more than one number in  $Z_8$  maps into the same residue. Specifically,  $6 \times 0 \bmod 8 = 6 \times 4 \bmod 8$ ;  $6 \times 1 \bmod 8 = 6 \times 5 \bmod 8$ ; and so on. Because this is a many-to-one mapping, there is not a unique inverse to the multiply operation.

However, if we take  $a = 5$  and  $n = 8$ ,

$Z_8$	0	1	2	3	4	5	6	7
Multiply by 5	0	5	10	15	20	25	30	35
Residues	0	5	2	7	4	1	6	3

The line of residues contains all the numbers in  $Z_8$ , in a different order.

Finally, we can observe that if  $p$  is a prime number, then all the elements of  $Z_p$  are relatively prime to  $p$ . This enables us to add one property to those listed earlier:

---

Multiplicative inverse ( $w^{-1}$ )      For each  $w \in Z_p$ , there exists a  $z$  such that  $w \times z \equiv 1 \bmod p$ .

---

Because  $w$  is relatively prime to  $p$ , if we multiply all the elements of  $Z_p$  by  $w$ , the resulting residues include all of the elements of  $Z_p$  permuted. Thus, at least one of the residues has the value 1. Therefore, there is some number in  $Z_p$  that, when multiplied by  $w$ , yields the residue 1. That number is the multiplicative inverse of  $w$ , designated  $w^{-1}$ . Thus, Equation (7.2) is consistent with the existence of a multiplicative inverse. Multiplying to both sides of Equation (7.2) by the multiplicative inverse of  $a$ , we have

$$((a^{-1}) \times a \times b) \equiv ((a^{-1}) \times a \times c) \bmod n$$

$$b \equiv c \bmod n$$

A final point: Some, but not all, integers will have a multiplicative inverse even if the modulus is not a prime number. Specifically, if  $\gcd(a, n) = 1$ , one can find  $b$  in  $Z_n$  such that  $a \times b \equiv 1 \bmod n$ . The reasoning is the same as that of the preceding paragraph. Because  $a$  is relatively prime to  $n$ , if we multiply all the elements of  $Z_n$  by  $a$ , the resulting residues include all the elements of  $Z_n$  permuted. Therefore, there is some number  $b$  in  $Z_n$  such that  $a \times b \equiv 1 \bmod n$ .

Table 7.3 provides an example that illustrates the concepts of this section.



**Table 7.3** Arithmetic Modulo 7

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

(a) Addition modulo 7

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

(b) Multiplication modulo 7

w	-w	w <sup>-1</sup>
0	0	—
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

(c) Additive and multiplicative inverses modulo 7

## 7.3 FERMAT'S AND EULER'S THEOREMS

Two theorems that play important roles in public-key cryptography are Fermat's theorem and Euler's theorem.

### Fermat's Theorem

Fermat's theorem states the following: If  $p$  is prime and  $a$  is a positive integer not divisible by  $p$ , then

$$a^{p-1} \equiv 1 \pmod{p} \quad (7.3)$$

**Proof:** From our previous discussion, we know that if all the elements of  $Z_p$  are multiplied by  $a$ , modulo  $p$ , the result consists of the elements of  $Z_p$  in some order. Furthermore,  $a \times 0 \equiv 0 \pmod{p}$ . Therefore, the  $(p - 1)$  numbers  $\{a \pmod{p}, 2a \pmod{p}, \dots,$

$(p-1)a \bmod p$  are just the numbers  $\{1, 2, \dots, (p-1)\}$  in some order. Multiply these numbers together:

$$a \times 2a \times \dots \times ((p-1)a) \equiv [(a \bmod p) \times (2a \bmod p) \times \dots \times ((p-1)a \bmod p)] \bmod p \\ \equiv (p-1)! \bmod p$$

But

$$a \times 2a \times \dots \times ((p-1)a) = (p-1)!a^{p-1}$$

Therefore,

$$(p-1)!a^{p-1} \equiv (p-1)! \bmod p$$

We can cancel the  $(p-1)!$  term because it is relatively prime to  $p$  [see Equation (7.2)]. This yields Equation (7.3).

$$\begin{aligned} a &= 7, p = 19 \\ 7^2 &= 49 \equiv 11 \bmod 19 \\ 7^4 &\equiv 121 \equiv 7 \bmod 19 \\ 7^8 &\equiv 49 \equiv 11 \bmod 19 \\ 7^{16} &\equiv 121 \equiv 7 \bmod 19 \\ a^{p-1} &= 7^{18} = 7^{16} \times 7^2 \equiv 7 \times 11 \equiv 1 \bmod 19 \end{aligned}$$

An alternative form of the theorem is also useful: If  $p$  is prime and  $a$  is any positive integer, then

$$a^p \equiv a \bmod p \quad (7.4)$$

$$\begin{aligned} p = 5, a = 3, 3^5 &= 243 \equiv 3 \bmod 5 \\ p = 5, a = 10, 10^5 &= 100000 \equiv 10 \bmod 5 \equiv 0 \bmod 5 \end{aligned}$$

### Euler's Totient Function

Before presenting Euler's theorem, we need to introduce an important quantity in number theory, referred to as Euler's totient function and written  $\phi(n)$ , where  $\phi(n)$  is the number of positive integers less than  $n$  and relatively prime to  $n$ .

Table 7.4 lists the first 30 values of  $\phi(n)$ . The value  $\phi(1)$  is without meaning but is defined to have the value 1.

It should be clear that for a prime number  $p$ ,

$$\phi(p) = p - 1$$

Now suppose that we have two prime numbers  $p$  and  $q$ . Then, for  $n = pq$ ,

$$\phi(n) = \phi(pq) = \phi(p) \times \phi(q) = (p-1) \times (q-1)$$

**Table 7.4** Some Values of Euler's Totient Function  $\phi(n)$ 

$n$	$\phi(n)$	$n$	$\phi(n)$	$n$	$\phi(n)$
1	1	11	10	21	12
2	1	12	4	22	10
3	2	13	12	23	22
4	2	14	6	24	8
5	4	15	8	25	20
6	2	16	8	26	12
7	6	17	16	27	18
8	4	18	6	28	12
9	6	19	18	29	28
10	4	20	8	30	8

To see this, consider that the set of residues in  $Z_n$  is  $\{0, 1, \dots, (pq - 1)\}$ . The residues that are not relatively prime to  $n$  are the set  $\{p, 2p, \dots, (q - 1)p\}$ , the set  $\{q, 2q, \dots, (p - 1)q\}$ , and 0. Accordingly,

$$\begin{aligned}
 \phi(n) &= pq - [(q - 1) + (p - 1) + 1] \\
 &= pq - (p + q) + 1 \\
 &= (p - 1) \times (q - 1) \\
 &= \phi(p) \times \phi(q)
 \end{aligned}$$

$$\begin{aligned}
 \phi(21) &= 12 = \phi(3) \times \phi(7) = 2 \times 6 = (3 - 1) \times (7 - 1) \\
 &\text{where the 12 integers are } \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}
 \end{aligned}$$

### Euler's Theorem

Euler's theorem states that for every  $a$  and  $n$  that are relatively prime,

$$a^{\phi(n)} \equiv 1 \pmod{n} \quad (7.5)$$

$$\begin{aligned}
 a &= 3; n = 10; \phi(10) = 4; 3^4 = 81 \equiv 1 \pmod{10} \\
 a &= 2; n = 11; \phi(11) = 10; 2^{10} = 1024 \equiv 1 \pmod{11}
 \end{aligned}$$

**Proof:** Equation (7.5) is true if  $n$  is prime, because in that case  $\phi(n) = (n - 1)$ , and Fermat's theorem holds. However, it also holds for any integer  $n$ . Recall that  $\phi(n)$  is the number of positive integers less than  $n$  that are relatively prime to  $n$ . Consider the set of such integers, labeled as follows:

$$R = \{x_1, x_2, \dots, x_{\phi(n)}\}$$

Now multiply each element by  $a$ , modulo  $n$ :

$$S = \{(ax_1 \pmod{n}), (ax_2 \pmod{n}), \dots, (ax_{\phi(n)} \pmod{n})\}$$

This set is a permutation of  $R$ , by the following line of reasoning:

1. Because  $a$  is relatively prime to  $n$  and  $x_i$  is relatively prime to  $n$ ,  $ax_i$  must also be relatively prime to  $n$ . Thus, all the members of  $S$  are integers less than  $n$  that are relatively prime to  $n$ .
2. There are no duplicates in  $S$ . Refer to Equation (7.2). If  $ax_i \bmod n = ax_j \bmod n$ , then  $x_i = x_j$ .

Therefore,

$$\begin{aligned} \prod_{i=1}^{\phi(n)} (ax_i \bmod n) &= \prod_{i=1}^{\phi(n)} x_i \\ \prod_{i=1}^{\phi(n)} ax_i &\equiv \prod_{i=1}^{\phi(n)} x_i \pmod{n} \\ a^{\phi(n)} + \left[ \prod_{i=1}^{\phi(n)} x_i \right] &\equiv \prod_{i=1}^{\phi(n)} x_i \pmod{n} \\ a^{\phi(n)} &\equiv 1 \pmod{n} \end{aligned}$$

An alternative form of the theorem is also useful:

$$a^{\phi(n)+1} \equiv a \pmod{n} \quad (7.6)$$

We can develop a corollary to Euler's theorem that is useful in demonstrating the validity of the RSA algorithm. Given two prime numbers,  $p$  and  $q$ , and integers  $n = pq$  and  $m$ , with  $0 < m < n$ , the following relationship holds:

$$m^{\phi(n)+1} = m^{(p-1)(q-1)+1} \equiv m \pmod{n} \quad (7.7)$$

If  $\gcd(m, n) = 1$ —that is, if  $m$  and  $n$  are relatively prime—then the relationship holds by virtue of Euler's theorem [Equation (7.5)]. Suppose  $\gcd(m, n) \neq 1$ . What does this mean? Because  $n = pq$ , the equality  $\gcd(m, n) = 1$  is equivalent to the logical expression ( $m$  is not a multiple of  $p$ ) AND ( $m$  is not a multiple of  $q$ ). If  $m$  is a multiple of  $p$ , then  $n$  and  $m$  share the prime factor  $p$  and are not relatively prime, and if  $m$  is a multiple of  $q$ , then  $n$  and  $m$  share the prime factor  $q$  and are not relatively prime. Therefore, the expression  $\gcd(m, n) \neq 1$  must be equivalent to the negation of the foregoing logical expression. Therefore,  $\gcd(m, n) \neq 1$  is equivalent to the logical expression ( $m$  is a multiple of  $p$ ) OR ( $m$  is a multiple of  $q$ ).

Let us look at the case in which  $m$  is a multiple of  $p$ , so that the relationship  $m = cp$  holds for some positive integer  $c$ . In this case, we must have  $\gcd(m, q) = 1$ . Otherwise, we have  $m$  a multiple of  $p$  and  $m$  a multiple of  $q$  and yet  $m < pq$ . If  $\gcd(m, q) = 1$ , then Euler's theorem holds and

$$m^{\phi(q)} \equiv 1 \pmod{q}$$

But then, by the rules of modular arithmetic,

$$\begin{aligned} [m^{\phi(q)}]^{\phi(p)} &\equiv 1 \pmod{q} \\ m^{\phi(n)} &\equiv 1 \pmod{q} \end{aligned}$$

Therefore, there is some integer  $k$  such that

$$m^{\phi(n)} = 1 + kq$$

Multiplying each side by  $m = cp$ ,

$$m^{\phi(n)+1} = m + kcpq = m + kcn$$

$$m^{\phi(n)+1} \equiv m \pmod{n}$$

A similar line of reasoning is used for the case in which  $m$  is a multiple of  $q$ . Thus, Equation (7.7) is proven. An alternative form of this corollary is also useful.

$$[m^{\phi(n)}]^k \equiv 1 \pmod{n}$$

$$m^{k\phi(n)} \equiv 1 \pmod{n} \tag{7.8}$$

$$m^{k\phi(n)+1} = m^{k(p-1)(q-1)+1} \equiv m \pmod{n}$$

## 7.4 TESTING FOR PRIMALITY

There is no simple yet efficient means of determining whether a large number is prime. In this section, we present one attractive approach. First, we need to derive some results. The first of these is as follows:

If  $p$  is an odd prime, then the equation

$$x^2 \equiv 1 \pmod{p}$$

has only two solutions—namely,  $x \equiv 1$  and  $x \equiv -1$ .

**Proof:** We have

$$x^2 - 1 \equiv 0 \pmod{p}$$

$$(x + 1)(x - 1) \equiv 0 \pmod{p}$$

By the rules of modular arithmetic, the latter equality requires that  $p$  divide  $(x + 1)$  or  $p$  divide  $(x - 1)$ , or both. Suppose that  $p$  divides both  $(x + 1)$  and  $(x - 1)$ . Then we can say that  $(x + 1) = kp$  and  $(x - 1) = jp$  for some integers  $k$  and  $j$ . Subtracting the two equations yields  $2 = (k - j)p$ . This equation can be true only for  $p = 2$ . By the terms of the theorem, we are concerned only with odd primes. Therefore, for a given solution  $x$ , either  $p|(x + 1)$  or  $p|(x - 1)$  but not both. Suppose  $p|(x - 1)$ . Then

$$x - 1 = kp \quad \text{for some } k$$

Therefore,  $x \equiv 1 \pmod{p}$ . By similar reasoning we arrive at the other solution of  $x \equiv -1 \pmod{p}$ .

$$x^2 \equiv 1 \pmod{7}$$

Using Table 7.3b:

$$1^2 \equiv 1 \pmod{7}$$

$$6^2 \equiv 36 \pmod{7} \equiv 1 \pmod{7}; 6 \equiv -1 \pmod{7}$$

Solutions: 1, -1

$$x^2 \equiv 1 \pmod{8}$$

Using Table 7.2b:

$$1^2 \equiv 1 \pmod{8}$$

$$3^2 \equiv 9 \pmod{8} \equiv 1 \pmod{8}$$

$$5^2 \equiv 25 \pmod{8} \equiv 1 \pmod{8}; 5 \equiv -3 \pmod{8}$$

$$7^2 \equiv 49 \pmod{8} \equiv 1 \pmod{8}; 7 \equiv -1 \pmod{8}$$

Solutions: 1, -1, 3, -3

The theorem can be stated the opposite way: If there exist solutions to  $x^2 \equiv 1 \pmod{n}$  other than  $\pm 1$ , then  $n$  is not a prime number.

We can now state an algorithm, due to Miller and Rabin [MILL75, RAB180], for testing whether a number is prime. The core algorithm, called WITNESS, is defined as follows:

```

WITNESS (a, n)
1. let  $b_k b_{k-1} \dots b_0$  be the binary representation of  $(n-1)$ 
2.  $d \leftarrow 1$ 
3. for  $i \leftarrow k$  downto 0
4.   do  $x \leftarrow d$ 
5.    $d \leftarrow (d \times d) \pmod{n}$ 
6.   if  $d = 1$  and  $x \neq 1$  and  $x \neq n-1$ 
7.     then return TRUE
8.   if  $b_i = 1$ 
9.     then  $d \leftarrow (d \times a) \pmod{n}$ 
10.  if  $d \neq 1$ 
11.    then return TRUE
12.  return FALSE

```

The inputs to WITNESS are the number  $n$ , to be tested for primality, and some integer  $a$  less than  $n$ . The purpose is to test whether  $n$  is prime. If WITNESS returns TRUE, then  $n$  is definitely not prime; if WITNESS returns FALSE, then  $n$  may be prime.

A comparison of WITNESS to the algorithm of Figure 6.7 for computing a power in modular arithmetic shows that lines 3 to 9 compute  $d$  as the value  $a^{n-1} \pmod{n}$ . We know from Fermat's theorem [Equation (7.3)] that  $a^{n-1} \equiv 1 \pmod{n}$  if  $n$  is prime. Thus, if the final result for  $d$  is not equal to 1, we know that  $n$  is not prime, and return TRUE. Now consider the test at line 6. Because  $(n-1) \equiv -1 \pmod{n}$ , this line tests whether  $x^2 \equiv 1 \pmod{n}$  with a root other than  $\pm 1$ . By the theorem stated earlier, this condition holds only if  $n$  is not prime. Thus, if this test is passed, WITNESS returns TRUE.

As an example, consider again Figure 6.8. In this case,  $n = 561$  and  $a = 7$ . WITNESS discovers a square root other than  $\pm 1$  in the last squaring step, because  $a^{280} \equiv 67 \pmod{561}$  and  $a^{560} \equiv 1 \pmod{561}$ . At this point, WITNESS returns TRUE.

So if WITNESS returns TRUE, the number  $n$  is not prime. It can be shown (e.g., see [CORM90]) that given an odd number  $n$  that is not prime and a randomly

chosen integer  $a < n$ , the probability that WITNESS will return FALSE (i.e., fail to detect that  $n$  is not prime) is less than 0.5. This gives us a basis for determining whether an odd integer  $n$  is prime with a reasonable degree of confidence. The procedure is as follows: Repeatedly invoke WITNESS ( $a, n$ ) using randomly chosen values for  $a$ . If, at any point, WITNESS returns TRUE, then  $n$  is determined to be nonprime. If WITNESS returns FALSE  $s$  times in succession, then the probability that  $n$  is prime is at least  $1 - 2^{-s}$  (see [CORM90], p. 843). Thus, for a sufficiently large value of  $s$ , we can be confident that  $n$  is prime.

## 7.5 EUCLID'S ALGORITHM

One of the basic techniques of number theory is Euclid's algorithm, which is a simple procedure for determining the greatest common divisor of two positive integers. An extended form of Euclid's algorithm determines the greatest common divisor of two positive integers and, if those numbers are relatively prime, the multiplicative inverse of one with respect to the other.

### Finding the Greatest Common Divisor

Euclid's algorithm is based on the following theorem: For any nonnegative integer  $a$  and any positive integer  $b$ ,

$$\gcd(a, b) = \gcd(b, a \bmod b) \quad (7.9)$$

$$\gcd(55, 22) = \gcd(22, 55 \bmod 22) = \gcd(22, 11) = 11$$

To see this, consider if  $d = \gcd(a, b)$ . Then, by the definition of  $\gcd$ ,  $d|a$  and  $d|b$ . For any positive integer  $b$ ,  $a$  can be expressed in the form

$$\begin{aligned} a &= kb + r \equiv r \bmod b \\ a \bmod b &= r \end{aligned}$$

Therefore,  $(a \bmod b) = a - kb$  for some integer  $k$ . But because  $d|b$ , it also divides  $kb$ . We also have  $d|a$ . Therefore,  $d|(a \bmod b)$ . This shows that  $d$  is a common divisor of  $b$  and  $(a \bmod b)$ . Conversely, if  $d$  is a common divisor of  $b$  and  $(a \bmod b)$ , then  $d|kb$  and thus  $d|[kb + (a \bmod b)]$ , which is equivalent to  $d|a$ . Thus, the set of common divisors of  $a$  and  $b$  is equal to the set of common divisors of  $b$  and  $(a \bmod b)$ . Therefore, the  $\gcd$  of one is the same as the  $\gcd$  of the other, proving the theorem.

Equation (7.9) can be used repetitively to determine the greatest common divisor.

$$\begin{aligned} \gcd(18, 12) &= \gcd(12, 6) = \gcd(6, 0) = 6 \\ \gcd(11, 10) &= \gcd(10, 1) = \gcd(1, 0) = 1 \end{aligned}$$

Euclid's algorithm makes repeated use of Equation (7.9) to determine the greatest common divisor, as follows. The algorithm assumes  $d > f > 0$ . It is acceptable to restrict the algorithm to positive integers because  $\gcd(a, b) = \gcd(|a|, |b|)$ .

EUCLID( $d, f$ )

1.  $X \leftarrow f; Y \leftarrow d$
2. **if**  $Y = 0$  **return**  $X = \gcd(d, f)$
3.  $R = X \bmod Y$
4.  $X \leftarrow Y$
5.  $Y \leftarrow R$
6. **goto** 2

To find the  $\gcd(1970, 1066)$ ,

$1970 = 1 \times 1066 + 904$	$\gcd(1066, 904)$
$1066 = 1 \times 904 + 162$	$\gcd(904, 162)$
$904 = 5 \times 162 + 94$	$\gcd(162, 94)$
$162 = 1 \times 94 + 68$	$\gcd(94, 68)$
$94 = 1 \times 68 + 26$	$\gcd(68, 26)$
$68 = 2 \times 26 + 16$	$\gcd(26, 16)$
$26 = 1 \times 16 + 10$	$\gcd(16, 10)$
$16 = 1 \times 10 + 6$	$\gcd(10, 6)$
$10 = 1 \times 6 + 4$	$\gcd(6, 4)$
$6 = 2 \times 2 + 2$	$\gcd(4, 2)$
$2 = 2 \times 2 + 0$	$\gcd(2, 0)$

Therefore,  $\gcd(1970, 1066) = 2$ .

The alert reader may ask how we can be sure that this process terminates. That is, how can we be sure that at some point  $Y$  divides  $X$ ? If not, we would get an endless sequence of positive integers, each one strictly smaller than the one before, and this is clearly impossible.

### Finding the Multiplicative Inverse

If  $\gcd(d, f) = 1$ , then  $d$  has a multiplicative inverse modulo  $f$ . That is, for positive integer  $d < f$ , there exists a  $d^{-1} < f$  such that  $dd^{-1} = 1 \bmod f$ . Euclid's algorithm can be extended so that, in addition to finding  $\gcd(d, f)$ , if the gcd is 1, the algorithm returns the multiplicative inverse of  $d$ .

EXTENDED EUCLID( $d, f$ )

1.  $(X1, X2, X3) \leftarrow (1, 0, f); (Y1, Y2, Y3) \leftarrow (0, 1, d)$
2. **if**  $Y3 = 0$  **return**  $X3 = \gcd(d, f)$ ; no inverse
3. **if**  $Y3 = 1$  **return**  $Y3 = \gcd(d, f); Y2 = d^{-1} \bmod f$
4.  $Q = \left\lfloor \frac{X3}{Y3} \right\rfloor$
5.  $(T1, T2, T3) \leftarrow (X1 - QY1, X2 - QY2, X3 - QY3)$
6.  $(X1, X2, X3) \leftarrow (Y1, Y2, Y3)$
7.  $(Y1, Y2, Y3) \leftarrow (T1, T2, T3)$
8. **goto** 2



Throughout the computation, the following relationships hold:

$$fT1 + dT2 = T3 \quad fX1 + dX2 = X3 \quad fY1 + dY2 = Y3$$

To see that this algorithm correctly returns  $\gcd(d, f)$ , note that if we equate  $X$  and  $Y$  in Euclid's algorithm with  $X3$  and  $Y3$  in the extended Euclid's algorithm, then the treatment of the two variables is identical. At each iteration of Euclid's algorithm,  $X$  is set equal to the previous value of  $Y$  and  $Y$  is set equal to the previous value of  $X \bmod Y$ . Similarly, at each step of the extended Euclid's algorithm,  $X3$  is set equal to the previous value of  $Y3$ , and  $Y3$  is set equal to the previous value of  $X3$  minus the quotient of  $X3$  divided by  $Y3$ . This latter value is simply the remainder of  $X3$  divided by  $Y3$ , which is  $X3 \bmod Y3$ .

Note also that if  $\gcd(d, f) = 1$ , then on the final step we would have  $Y3 = 0$  and  $X3 = 1$ . Therefore, on the preceding step,  $Y3 = 1$ . But if  $Y3 = 1$ , then we can say the following:

$$fY1 + dY2 = Y3$$

$$fY1 + dY2 = 1$$

$$dY2 = 1 + (-Y1) \times f$$

$$dY2 \equiv 1 \pmod f$$

And  $Y2$  is the multiplicative inverse of  $d$ , modulo  $f$ .

Table 7.5 is an example of the execution of the algorithm. It shows that  $\gcd(550, 1769) = 1$  and that the multiplicative inverse of 550 is itself; that is,  $550 \times 550 \equiv 1 \pmod{1769}$ .

For a more detailed proof of this algorithm, see [KNUT97].

**Table 7.5** Extended Euclid (550, 1769)

Q	X1	X2	X3	Y1	Y2	Y3
—	1	0	1769	0	1	550
3	0	1	550	1	-3	119
4	1	-3	119	-4	13	74
1	-4	13	74	5	-16	45
1	5	-16	45	-9	29	29
1	-9	29	29	14	-45	16
1	14	-45	16	-23	74	13
1	-23	74	13	37	-119	3
4	37	-119	3	-171	550	1

## 7.6 THE CHINESE REMAINDER THEOREM

One of the most useful elements of number theory is the Chinese remainder theorem (CRT).<sup>2</sup> In essence, the CRT says it is possible to reconstruct integers in a certain range from their residues modulo a set of pairwise relatively prime moduli.

The 10 integers in  $\mathbb{Z}_{10}$  (0, 1, ..., 9) can be reconstructed from their two residues modulo 2 and 5 (the relatively prime factors of 10). Say the known residues of a decimal digit  $x$  are  $r_2 = 0$  and  $r_5 = 3$ ; that is,  $x \bmod 2 = 0$  and  $x \bmod 5 = 3$ . Therefore,  $x$  is an even integer in  $\mathbb{Z}_{10}$  whose remainder, on division by 5, is 3. The unique solution is  $x = 8$ .

The CRT can be stated in several ways. We present here a formulation that is most useful from the point of view of this text. An alternative formulation is explored in Problem 7.13. Let

$$M = \prod_{i=1}^k m_i$$

where the  $m_i$  are pairwise relatively prime; that is,  $\gcd(m_i, m_j) = 1$  for  $1 \leq i, j \leq k$ , and  $i \neq j$ . We can represent any integer in  $\mathbb{Z}_M$  by a  $k$ -tuple whose elements are in  $\mathbb{Z}_{m_i}$  using the following correspondence:

$$A \leftrightarrow (a_1, a_2, \dots, a_k) \quad (7.10)$$

where  $A \in \mathbb{Z}_M$ ,  $a_i \in \mathbb{Z}_{m_i}$ , and  $a_i = A \bmod m_i$  for  $1 \leq i \leq k$ . The CRT makes two assertions:

1. The mapping of Equation (7.10) is a one-to-one correspondence (called a bijection) between  $\mathbb{Z}_M$  and the Cartesian product  $\mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2} \times \dots \times \mathbb{Z}_{m_k}$ . That is, for every integer  $A$  such that  $0 \leq A < M$ , there is a unique  $k$ -tuple  $(a_1, a_2, \dots, a_k)$  with  $0 \leq a_i < m_i$  that represents it, and for every such  $k$ -tuple  $(a_1, a_2, \dots, a_k)$  there is a unique  $A$  in  $\mathbb{Z}_M$ .
2. Operations performed on the elements of  $\mathbb{Z}_M$  can be equivalently performed on the corresponding  $k$ -tuples by performing the operation independently in each coordinate position in the appropriate system.

The second statement above can be stated as follows: If

$$A \leftrightarrow (a_1, a_2, \dots, a_k); B \leftrightarrow (b_1, b_2, \dots, b_k)$$

<sup>2</sup>The CRT is so called because it is believed to have been discovered by the Chinese mathematician Sun-Tse in around 100 A.D.

then

$$(A + B) \bmod M \leftrightarrow ((a_1 + b_1) \bmod m_1, \dots, (a_k + b_k) \bmod m_k)$$

$$(A - B) \bmod M \leftrightarrow ((a_1 - b_1) \bmod m_1, \dots, (a_k - b_k) \bmod m_k)$$

$$(A \times B) \bmod M \leftrightarrow ((a_1 \times b_1) \bmod m_1, \dots, (a_k \times b_k) \bmod m_k)$$

Let us demonstrate the first assertion. The transformation from  $A$  to  $(a_1, a_2, \dots, a_k)$  is obviously unique; just take  $a_i = A \bmod m_i$ . Computing  $A$  from  $(a_1, a_2, \dots, a_k)$  can be done as follows. Let  $M_i = M/m_i$  for  $1 \leq i \leq k$ . Note that  $M_i = m_1 \times m_2 \times \dots \times m_{i-1} \times m_{i+1} \times \dots \times m_k$ , so that  $M_i \equiv 0 \pmod{m_j}$  for all  $j \neq i$ . Then let

$$c_i = M_i \times (M_i^{-1} \bmod m_i) \quad \text{for } 1 \leq i \leq k \quad (7.11)$$

By the definition of  $M_i$ , it is relatively prime to  $m_i$  and therefore has a unique multiplicative inverse mod  $m_i$ . So Equation (7.11) is well defined and produces a unique value  $c_i$ . We can now compute

$$A \equiv \left( \sum_{i=1}^k a_i c_i \right) \bmod M \quad (7.12)$$

To show that the value of  $A$  produced by Equation (7.12) is correct, we must show that  $a_i = A \bmod m_i$  for  $1 \leq i \leq k$ . Note that  $c_j \equiv M_j \equiv 0 \pmod{m_i}$  if  $j \neq i$  and that  $c_i \equiv 1 \pmod{m_i}$ . It follows that  $a_i = A \bmod m_i$ .

The second assertion of the CRT, concerning arithmetic operations, follows from the rules for modular arithmetic.

One of the useful features of the Chinese remainder theorem is that it provides a way to manipulate (potentially very large) numbers mod  $M$  in terms of tuples of smaller numbers. This can be useful when  $M$  is 150 digits or more.

To represent  $973 \bmod 1813$  as a pair of numbers mod 37 and 49, define<sup>3</sup>

$$\begin{aligned} m_1 &= 37 \\ m_2 &= 49 \\ M &= 1813 \\ A &= 973 \end{aligned}$$

We also have  $M_1 = 49$  and  $M_2 = 37$ . Using the extended Euclid's algorithm, we compute  $M_1^{-1} = 34 \bmod m_1$  and  $M_2^{-1} = 4 \bmod m_2$ . (Note that we only need to compute each  $M_i$  and each  $M_i^{-1}$  once for all.) Taking residues modulo 37 and 49, our representation of 973 is  $(11, 42)$ , because  $973 \bmod 37 = 11$  and  $973 \bmod 49 = 42$ .

Now suppose we want to add 678 to 973. What do we do to  $(11, 42)$ ? First we compute  $(678) \leftrightarrow (678 \bmod 37, 678 \bmod 49) = (12, 41)$ . Then we add the

<sup>3</sup>This example was provided by Professor Ken Calvert of Georgia Tech.

tuples element-wise and reduce  $(11 + 12 \bmod 37, 42 + 41 \bmod 49) = (23, 34)$ . To verify that this has the correct effect, we compute

$$\begin{aligned}(23, 34) &\leftrightarrow a_1 M_1 M_1^{-1} + a_2 M_2 M_2^{-1} \bmod M \\ &= [(23)(49)(34) + (34)(37)(4)] \bmod 1813 \\ &= 43350 \bmod 1813 \\ &= 1651\end{aligned}$$

and check that it is equal to  $(973 + 678) \bmod 1813 = 1651$ .

Suppose we want to multiply  $1651 \bmod 1813$  by 73. We multiply  $(23, 34)$  by 73 and reduce to get  $(23 \times 73 \bmod 37, 34 \times 73 \bmod 49) = (14, 32)$ . It is easily verified that

$$\begin{aligned}(14, 32) &\leftrightarrow [(14)(49)(34) + (32)(37)(4)] \bmod 1813 \\ &= 865 \\ &= 1651 \times 73 \bmod 1813\end{aligned}$$

## 7.7 DISCRETE LOGARITHMS

Discrete logarithms are fundamental to a number of public-key algorithms, including Diffie-Hellman key exchange and the digital signature algorithm (DSA). This section provides a brief overview of discrete logarithms. For the interested reader, more detailed developments of this topic can be found in [ORE76] and [LEVE90].

### The Powers of an Integer, Modulo $n$

Recall from Euler's theorem [Equation (7.5)] that, for every  $a$  and  $n$  that are relatively prime,

$$a^{\phi(n)} \equiv 1 \bmod n$$

where  $\phi(n)$ , Euler's totient function, is the number of positive integers less than  $n$  and relatively prime to  $n$ . Now consider the more general expression

$$a^m \equiv 1 \bmod n \tag{7.13}$$

If  $a$  and  $n$  are relatively prime, then there is at least one integer  $m$  that satisfies Equation (7.13)—namely,  $m = \phi(n)$ . The least positive exponent  $m$  for which Equation (7.13) holds is referred to in several ways:

- the order of  $a \bmod n$
- the exponent to which  $a$  belongs  $\bmod n$
- the length of the period generated by  $a$

To see this last point consider the powers of 7, modulo 19:

$$\begin{array}{rcl} 7^1 & = & 7 \bmod 19 \\ 7^2 = 49 & = 2 \times 19 + 11 = & 11 \bmod 19 \\ 7^3 = 343 & = 18 \times 19 + 1 = & 1 \bmod 19 \\ 7^4 = 2401 & = 126 \times 19 + 7 = & 7 \bmod 19 \\ 7^5 = 16807 & = 884 \times 19 + 11 = & 11 \bmod 19 \end{array}$$

There is no point in continuing because the sequence is repeating. This can be proven by noting that  $7^3 \equiv 1 \pmod{19}$  and therefore  $7^{3+j} \equiv 7^3 7^j \equiv 7^j \pmod{19}$ , and hence any two powers of 7 whose exponents differ by 3 (or a multiple of 3) are congruent to each other (mod 19). In other words, the sequence is periodic, and the length of the period is the smallest positive exponent  $m$  such that  $7^m \equiv 1 \pmod{19}$ .

Table 7.6 shows all the powers of  $a$ , modulo 19 for all positive  $a < 19$ . The length of the sequence for each base value is indicated by shading. Note the following:

1. All sequences end in 1. This is consistent with the reasoning of the preceding few paragraphs.
2. The length of a sequence divides  $\phi(19) = 18$ . That is, an integral number of sequences occur in each row of the table.
3. Some of the sequences are of length 18. In this case, it is said that the base integer  $a$  generates (via powers) the set of nonzero integers modulo 19. Each such integer is called a primitive root of the modulus 19.

**Table 7.6** Powers of Integers, Modulo 19

[illegible]

More generally, we can say that the highest possible exponent to which a number can belong (mod  $n$ ) is  $\phi(n)$ . If a number is of this order, it is referred to as a primitive root of  $n$ . The importance of this notion is that if  $a$  is a primitive root of  $n$ , then its powers

$$a, a^2, \dots, a^{\phi(n)}$$

are distinct (mod  $n$ ) and are all relatively prime to  $n$ . In particular, for a prime number  $p$ , if  $a$  is a primitive root of  $p$ , then

$$a, a^2, \dots, a^{p-1}$$

are distinct (mod  $p$ ). For the prime number 19, its primitive roots are 2, 3, 10, 13, 14, and 15.

Not all integers have primitive roots. In fact, the only integers with primitive roots are those of the form 2, 4,  $p^a$ , and  $2p^a$ , where  $p$  is any odd prime.

### Indices

With ordinary positive real numbers, the logarithm function is the inverse of exponentiation. An analogous function exists for modular arithmetic.

Let us briefly review the properties of ordinary logarithms. The logarithm of a number is defined to be the power to which some positive base (except 1) must be raised in order to equal the number. That is, for base  $x$  and for a value  $y$ ,

$$y = x^{\log_x(y)}$$

The properties of logarithms include the following:

$$\log_x(1) = 0 \quad (7.14)$$

$$\log_x(x) = 1 \quad (7.15)$$

$$\log_x(yz) = \log_x(y) + \log_x(Z) \quad (7.16)$$

$$\log_x(y^r) = r \times \log_x(y) \quad (7.17)$$

Consider a primitive root  $a$  for some prime number  $p$  (the argument can be developed for nonprimes as well). Then we know that the powers of  $a$  from 1 through  $(p - 1)$  produce each integer from 1 through  $(p - 1)$  exactly once. We also know that any integer  $b$  can be expressed in the form

$$b \equiv r \pmod{p} \quad \text{where } 0 \leq r \leq (p - 1)$$

by the definition of modular arithmetic. It follows that for any integer  $b$  and a primitive root  $a$  of prime number  $p$ , one can find a unique exponent  $i$  such that

$$b \equiv a^i \pmod{p} \quad \text{where } 0 \leq i \leq (p - 1)$$

This exponent  $i$  is referred to as the index of the number  $b$  for the base  $a \pmod{p}$ . We denote this value as  $\text{ind}_{a,p}(b)$ .

Note the following:

$$\text{ind}_{a,p}(1) = 0, \text{ because } a^0 \pmod{p} = 1 \pmod{p} = 1 \quad (7.18)$$

$$\text{ind}_{a,p}(a) = 1, \text{ because } a^1 \pmod{p} = a \quad (7.19)$$

Here is an example using a nonprime modulus,  $n = 9$ . Here  $\phi(n) = 6$  and  $a = 2$  is a primitive root. We compute the various powers of  $a$  and find

$$\begin{array}{ll} 2^0 = 1 & 2^4 = 7 \\ 2^1 = 2 & 2^5 = 5 \\ 2^2 = 4 & 2^6 = 1 \\ 2^3 = 8 & \end{array} \pmod{9}$$

This gives us the following table of the numbers with given indices  $\pmod{9}$  for the root  $a = 2$ :

Index	0	1	2	3	4	5
Number	1	2	4	8	7	5

To obtain the index of a given number, one rearranges the table to make the remainders relatively prime to 9 the primary entry:

Number	1	2	4	5	7	8
Index	0	1	2	5	4	3

Now consider

$$\begin{aligned} x &= a^{\text{ind}_{a,p}(x)} \pmod{p} & y &= a^{\text{ind}_{a,p}(y)} \pmod{p} \\ xy &= a^{\text{ind}_{a,p}(xy)} \pmod{p} \end{aligned}$$

Using the rules of modular multiplication,

$$\begin{aligned} a^{\text{ind}_{a,p}(xy)} \pmod{p} &= (a^{\text{ind}_{a,p}(x)} \pmod{p}) (a^{\text{ind}_{a,p}(y)} \pmod{p}) \\ &= (a^{\text{ind}_{a,p}(x) + \text{ind}_{a,p}(y)}) \pmod{p} \end{aligned}$$

But now consider Euler's theorem, which states that, for every  $a$  and  $n$  that are relatively prime,

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

Any positive integer  $z$  can be expressed in the form  $z = q + k\phi(n)$ . Therefore, by Euler's theorem,

$$a^z = a^q \pmod{n} \quad \text{if } z \equiv q \pmod{\phi(n)}$$

Applying this to the foregoing equality, we have

$$\text{ind}_{a,p}(xy) = [\text{ind}_{a,p}(x) + \text{ind}_{a,p}(y)] \bmod \phi(p)$$

and, generalizing,

$$\text{ind}_{a,p}(y^r) = [r \times \text{ind}_{a,p}(y)] \bmod \phi(p)$$

This demonstrates the analogy between true logarithms and indices. For this reason, the latter are often referred to as discrete logarithms.

Keep in mind that unique discrete logarithms mod  $m$  to some base  $a$  exist only if  $a$  is a primitive root of  $m$ .

Table 7.7, which is directly derived from Table 7.6, shows the sets of discrete logarithms that can be defined for modulus 19.

**Table 7.7** Tables of Discrete Logarithms, Modulo 19

(a) Discrete logarithms to the base 2, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\text{Ind}_{2,19}(a)$	18	1	13	2	16	14	6	3	8	17	12	15	5	7	11	4	10	9

(b) Discrete logarithms to the base 3, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\text{Ind}_{3,19}(a)$	18	7	1	14	4	8	6	3	2	11	12	15	17	13	5	10	16	9

(c) Discrete logarithms to the base 10, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\text{Ind}_{10,19}(a)$	18	17	5	16	2	4	12	15	10	1	6	3	13	11	7	14	8	9

(d) Discrete logarithms to the base 13, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\text{Ind}_{13,19}(a)$	18	11	17	4	14	10	12	15	16	7	6	3	1	5	13	8	2	9

(e) Discrete logarithms to the base 14, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\text{Ind}_{14,19}(a)$	18	13	7	8	10	2	6	3	14	5	12	15	11	1	17	16	14	9

(f) Discrete logarithms to the base 15, modulo 19

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\text{Ind}_{15,19}(a)$	18	5	11	10	8	16	12	15	4	13	6	3	7	17	1	2	12	9



### Calculation of Discrete Logarithms

Consider the equation

$$y = g^x \bmod p$$

Given  $g$ ,  $x$ , and  $p$ , it is a straightforward matter to calculate  $y$ . At the worst, one must perform  $x$  repeated multiplications, and algorithms exist for achieving greater efficiency.

However, given  $y$ ,  $g$ , and  $p$ , it is, in general, very difficult to calculate  $x$  (take the discrete logarithm). The difficulty seems to be on the same order of magnitude as that of factoring primes required for RSA. At the time of this writing, the asymptotically fastest known algorithm for taking discrete logarithms modulo a prime number is on the order of [BETH91]

$$e^{((\ln p)^{1/4} \ln(\ln p))^{2/3}}$$

which is not feasible for large primes.

## 7.8 RECOMMENDED READING

There are many basic texts on the subject of number theory that provide far more detail than most readers of this book will desire. An elementary but nevertheless useful short introduction is [ORE67]. For the reader interested in a more in-depth treatment, two excellent textbooks on the subject are [KUMA98] and [ROSE93]. [LEVE90] is a readable and detailed account as well. All of these books include problems with solutions, enhancing their value for self-study.

For readers willing to commit the time, perhaps the best way to get a solid grasp of the fundamentals of number theory is to work their way through [BURN97], which consists solely of a series of exercises with solutions that lead the student step by step through the concepts of number theory; working through all of the exercises is equivalent to completing an undergraduate course in number theory.

**BURN97** Burn, R. *A Pathway to Number Theory*. Cambridge, England: Cambridge University Press, 1997.

**KUMA98** Kumanduri, R., and Romero, C. *Number Theory with Computer Applications*. Upper Saddle River, NJ: Prentice Hall, 1998.

**LEVE90** Leveque, W. *Elementary Theory of Numbers*. New York: Dover, 1990.

**ORE67** Ore, O. *Invitation to Number Theory*. Washington, DC: The Mathematical Association of America, 1967.

**ROSE93** Rosen, K. *Elementary Number Theory and its Applications*. Reading, MA: Addison-Wesley, 1993.

## 7.9 PROBLEMS

- 7.1** The purpose of this problem is to justify the assertion, made in Section 6.2, that the probability that two random numbers are relatively prime is about 0.6.

- a. Let  $P = \Pr[\gcd(a, b) = 1]$ . Show that  $\Pr[\gcd(a, b = d)] = P/d^2$ . *Hint:* Consider the

$$\text{quantity } \gcd \begin{pmatrix} a & b \\ b & d \end{pmatrix}$$

- b. The sum of the result of part (a) over all possible values of  $d$  is 1. That is,

$$\sum_{d=1}^{\infty} \Pr[\gcd(a, b) = d] = 1$$

Use this equality to determine the value of  $P$ . *Hint:* Use the identity  $\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$ .

- 7.2** Why is  $\gcd(n, n+1) = 1$  for two consecutive integers  $n$  and  $n+1$ ?
- 7.3** In Section 7.2, we define the congruence relationship as follows: Two integers  $a$  and  $b$  are said to be congruent modulo  $n$  if  $(a \bmod n) = (b \bmod n)$ . We then proved that  $a \equiv b \bmod n$  if  $n \mid (a - b)$ . Some texts on number theory use this latter relationship as the definition of congruence: Two integers  $a$  and  $b$  are said to be congruent modulo  $n$ , if  $n \mid (a - b)$ . Using this latter definition as the starting point, prove that if  $(a \bmod n) = (b \bmod n)$  then  $n$  divides  $(a - b)$ .
- 7.4** Prove the following:
- $(a \bmod n) = (b \bmod n)$  implies  $a \equiv b \bmod n$
  - $a \equiv b \bmod n$  implies  $b \equiv a \bmod n$
  - $a \equiv b \bmod n$  and  $b \equiv c \bmod n$  imply  $a \equiv c \bmod n$
- 7.5** Prove the following:
- $[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$
  - $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$
- 7.6** Using Fermat's theorem, find  $3^{201} \bmod 11$ .
- 7.7** Although ancient Chinese mathematicians did good work coming up with their remainder theorem, they did not always get it right. They had a test for primality. The test said that  $n$  is prime if and only if  $n$  divides  $(2^n - 2)$ .
- Give an example that satisfies the condition using an odd prime.
  - The condition is obviously true for  $n = 2$ . Prove that the condition is true if  $n$  is an odd prime (proving the **if** condition).
  - Give an example of an odd  $n$  that is not prime and that does not satisfy the condition. You can do this with nonprime numbers up to a very large value. This misled the Chinese mathematicians into thinking that if the condition is true, then  $n$  is prime.
  - Unfortunately, the ancient Chinese never tried  $n = 341$ , which is nonprime ( $341 = 11 \times 31$ ) and yet 341 divides  $2^{341} - 2$  without remainder. Demonstrate that  $2^{341} \equiv 2 \bmod 341$  (disproving the **only if** condition). *Hint:* It is not necessary to calculate  $2^{341}$ ; play around with the congruences instead.
- 7.8** One notices in Table 7.4 that  $\phi(n)$  is even for  $n > 2$ . This is true for all  $n > 2$ . Give a concise argument why this is so.
- 7.9**
- Determine  $\gcd(24140, 16762)$ .
  - Determine  $\gcd(4655, 12075)$ .
- 7.10** The purpose of this problem is to set an upper bound on the number of iterations of Euclid's algorithm.
- Suppose that  $m = qn + r$  with  $q$  and  $r$  nonnegative integers and  $0 \leq r < n$ . Show that  $m/2 > r$ .
  - Let  $X_i$  be the value of  $X$  in Euclid's algorithm after the  $i$ th iteration. Show that

$$X_{i+2} < \frac{X_i}{2}$$

- Show that if  $m$ ,  $n$ , and  $N$  are integers with  $1 \leq m, n, \leq 2^N$ , then Euclid's algorithm takes at most  $2N$  steps to find  $\gcd(m, n)$ .

- 7.11** Euclid's algorithm has been known for over 2000 years and has always been a favorite among number theorists. After these many years, there is now a potential competitor, invented by J. Stein in 1961. Stein's algorithm is as follows. Determine  $\gcd(A, B)$  with  $A, B \geq 1$ .

**STEP 1** Set  $A_1 = A, B_1 = B, C_1 = C$

**STEP n** (1) If  $A_n = B_n$  stop.  $\gcd(A, B) = A_n C_n$   
 (2) If  $A_n$  and  $B_n$  are both even, set  $A_{n+1} = A_n/2, B_{n+1} = B_n/2, C_{n+1} = 2C_n$   
 (3) If  $A_n$  is even and  $B_n$  is odd, set  $A_{n+1} = A_n/2, B_{n+1} = B_n, C_{n+1} = C_n$   
 (4) If  $A_n$  is odd and  $B_n$  is even, set  $A_{n+1} = A_n, B_{n+1} = B_n/2, C_{n+1} = C_n$   
 (5) If  $A_n$  and  $B_n$  are both odd, set  $A_{n+1} = |A_n - B_n|, B_{n+1} = \min(B_n, A_n), C_{n+1} = C_n$

Continue to step  $n + 1$

- a. To get a feel for the two algorithms, compute  $\gcd(2152, 764)$  using both Euclid's and Stein's algorithm.  
 b. What is the apparent advantage of Stein's algorithm over Euclid's algorithm?
- 7.12** a. Show that if Stein's algorithm does not stop before the  $n$ th step, then

$$C_{n+1} \times \gcd(A_{n+1}, B_{n+1}) = C_n \times \gcd(A_n, B_n)$$

- b. Show that if the algorithm does not stop before step  $(n - 1)$ , then

$$A_{n+2} B_{n+2} \leq \frac{A_n B_n}{2}$$

- c. Show that if  $1 \leq A, B \leq 2^N$ , then Stein's algorithm takes at most  $4N$  steps to find  $\gcd(m, n)$ . Thus, Stein's algorithm works in roughly the same number of steps as Euclid's algorithm.  
 d. Demonstrate that Stein's algorithm does indeed return  $\gcd(A, B)$ .
- 7.13** A common formulation of the Chinese remainder theorem goes like this: Let  $m_1, \dots, m_k$  be integers that are pairwise relatively prime for  $1 \leq i, j \leq k$ , and  $i \neq j$ . Define  $M$  to be the product of all the  $m_i$ 's. Let  $a_1, \dots, a_k$  be integers. Then the set of congruences

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

has a unique solution modulo  $M$ . Show that the theorem stated in this form is true.

- 7.14** The example used by Sun-Tse to illustrate the CRT was

$$x \equiv 2 \pmod{3}; x \equiv 3 \pmod{5}; x \equiv 2 \pmod{7}$$

Solve for  $x$ .

- 7.15** Six professors begin courses on Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday, respectively, and announce their intentions of lecturing at intervals of 2, 3, 4, 1, 6, and 5 days, respectively. The regulations of the university forbid Sunday lectures (so that a Sunday lecture must be omitted). When first will all six professors find themselves compelled to omit a lecture? *Hint:* Use the CRT.
- 7.16** Find all primitive roots of 25.
- 7.17** Given 2 as a primitive root of 29, construct a table of indices, and use it to solve the following congruences:
- $17x^2 \equiv 10 \pmod{29}$
  - $x^2 - 4x - 16 \equiv 0 \pmod{29}$
  - $x^7 \equiv 17 \pmod{29}$



## CHAPTER 8

# MESSAGE AUTHENTICATION AND HASH FUNCTIONS

*At cats' green on the Sunday he took the message from the inside of the pillar and added Peter Moran's name to the two names already printed there in the "Brontosaur" code. The message now read: "Leviathan to Dragon: Martin Hillman, Trevor Allan, Peter Moran: observe and tail." What was the good of it John hardly knew. He felt better, he felt that at last he had made an attack on Peter Moran instead of waiting passively and effecting no retaliation. Besides, what was the use of being in possession of the key to the codes if he never took advantage of it?*

—*Talking to Strange Men*, Ruth Rendell

Perhaps the most confusing area of network security is that of message authentication and the related topic of digital signatures. The attacks and countermeasures become so convoluted that practitioners in this area begin to remind one of the astronomers of old, who built epicycles on top of epicycles in an attempt to account for all contingencies. Fortunately, it appears that today's designers of cryptographic protocols, unlike those long-forgotten astronomers, are working from a fundamentally sound model.

It would be impossible, in anything less than book length, to exhaust all the cryptographic functions and protocols that have been proposed or implemented for message authentication and digital signatures. Instead, the purpose of this chapter and the next two is to provide a broad overview of the subject and to develop a systematic means of describing the various approaches.

This chapter begins with an introduction to the requirements for authentication and digital signature and the types of attacks to be countered.

Then the basic approaches are surveyed, including the increasingly important area of secure hash functions. Specific hash functions are examined in Chapter 9.

## 8.1 AUTHENTICATION REQUIREMENTS

In the context of communications across a network, the following attacks can be identified:

1. **Disclosure:** Release of message contents to any person or process not possessing the appropriate cryptographic key.
2. **Traffic analysis:** Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.
3. **Masquerade:** Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or nonreceipt by someone other than the message recipient.
4. **Content modification:** Changes to the contents of a message, including insertion, deletion, transposition, and modification.
5. **Sequence modification:** Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.
6. **Timing modification:** Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g., datagram) could be delayed or replayed.
7. **Repudiation:** Denial of receipt of message by destination or denial of transmission of message by source.

Measures to deal with the first two attacks are in the realm of message confidentiality and are dealt with in Part One. Measures to deal with items 3 through 6 in the foregoing list are generally regarded as message authentication. Mechanisms for dealing specifically with item 7 come under the heading of digital signatures. Generally, a digital signature technique will also counter some or all the attacks listed under items 3 through 6.

In summary, message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. Message authentication may also verify sequencing and timeliness. A digital signature is an authentication technique that also includes measures to counter repudiation by either source or destination.

## 8.2 AUTHENTICATION FUNCTIONS

Any message authentication or digital signature mechanism can be viewed as having fundamentally two levels. At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower-level function is then used as primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message.

This section is concerned with the types of functions that may be used to produce an authenticator. These may be grouped into three classes, as follows:

- **Message encryption:** The ciphertext of the entire message serves as its authenticator
- **Message authentication code (MAC):** A public function of the message and a secret key that produces a fixed-length value that serves as the authenticator
- **Hash function:** A public function that maps a message of any length into a fixed-length hash value, which serves as the authenticator

We now briefly examine each of these topics; MACs and hash functions are examined in greater detail in Sections 8.3 and 8.4.

### Message Encryption

Message encryption by itself can provide a measure of authentication. The analysis differs for conventional and public-key encryption schemes.

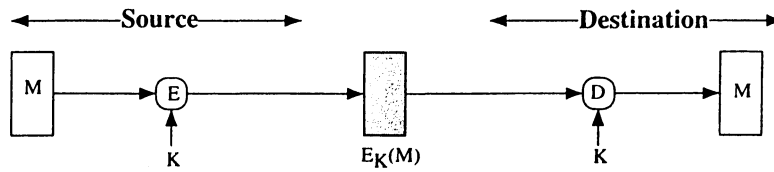
#### Conventional Encryption

Consider the straightforward use of conventional encryption (Figure 8.1a). A message transmitted from source A to destination B is encrypted using a secret key K shared by A and B. If no other party knows the key, then confidentiality is provided: No other party can recover the plaintext of the message.

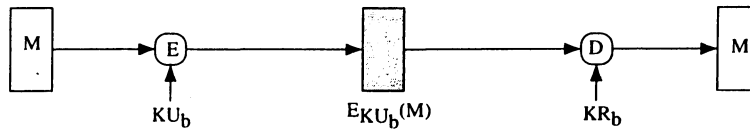
In addition, we may say that B is assured that the message came was generated by A. Why? The message must have come from A because A is the only other party that possesses K and therefore the only other party with the information necessary to construct ciphertext that can be decrypted with K. Furthermore, if M is recovered, B knows that none of the bits of M have been altered, because an opponent that does not know K would not know how to alter bits in the ciphertext to produce desired changes in the plaintext.

So we may say that conventional encryption provides authentication as well as confidentiality. However, this flat statement needs to be qualified. Consider exactly what is happening at B. Given a decryption function D and a secret key K, the destination will accept *any* input X and produce output  $Y = D_K(X)$ . If X is the ciphertext of a legitimate message M produced by the corresponding encryption function, then Y is some plaintext message M. Otherwise, Y will be a meaningless sequence of bits. There may need to be some automated means of determining at B whether Y is legitimate plaintext and therefore must have come from A.

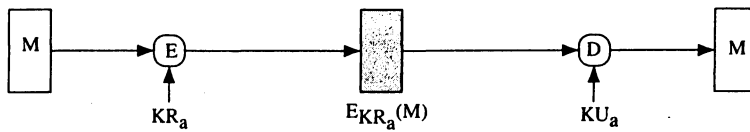
The implications of the line of reasoning in the preceding paragraph are profound from the point of view of authentication. Suppose the message M can be any



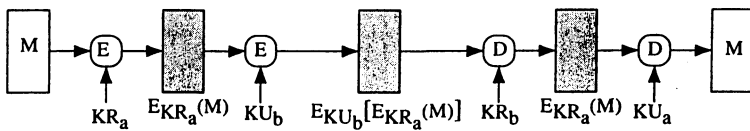
(a) Conventional encryption: confidentiality and authentication



(b) Public-key encryption: confidentiality



(c) Public-key encryption: authentication and signature



(d) Public-key encryption: confidentiality, authentication, and signature

**Figure 8.1** Basic Uses of Message Encryption.

arbitrary bit pattern. In that case, there is no way to determine automatically, at the destination, whether an incoming message is the ciphertext of a legitimate message. This conclusion is incontrovertible: If  $M$  can be any bit pattern, then regardless of the value of  $X$ ,  $Y = D_K(X)$  is *some* bit pattern and therefore must be accepted as authentic plaintext.

Thus, in general, we require that only a small subset of all possible bit patterns is considered legitimate plaintext. In that case, any spurious ciphertext is unlikely to produce legitimate plaintext. For example, suppose that only one bit pattern in  $10^6$  is legitimate plaintext. Then the probability that any randomly chosen bit pattern, treated as ciphertext, will produce a legitimate plaintext message is only  $10^{-6}$ .

For a number of applications and encryption schemes, the desired conditions prevail as a matter of course. For example, suppose that we are transmitting English-language messages using a Caesar cipher with a shift of one ( $K = 1$ ). A sends the following legitimate ciphertext:



nbsftfbupbutboeepftfbupbutboemjuumfmbnctfbujwz

B decrypts to produce the following plaintext:

mareseatoatsanddoseatoatsandlittlelambseativy

A simple frequency analysis confirms that this message has the profile of ordinary English. On the other hand, if an opponent generates the following random sequence of letters:

zuvrsoevgqxlzwigamdvnmhpmccxiuureosfbcebtqxsxq

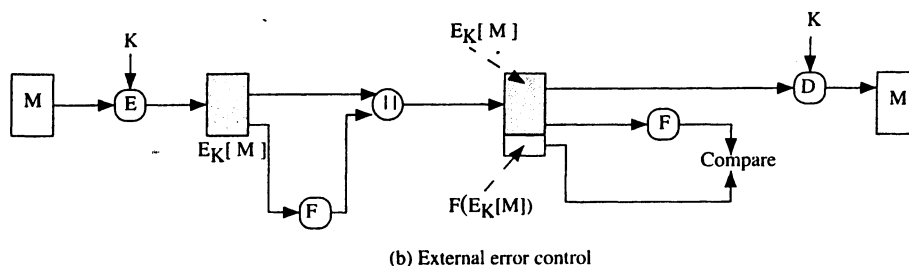
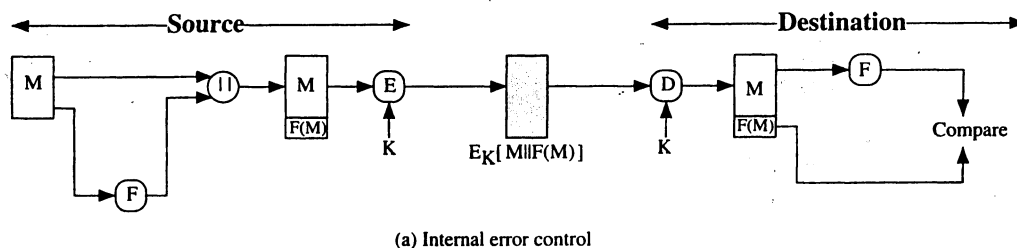
this decrypts to

ytuqrndufpkyvhfzlcumlgolbbwhttqdnreabdasprwrp

which does not fit the profile of ordinary English.

It may be difficult to determine *automatically* if incoming ciphertext decrypts to intelligible plaintext. If the plaintext is, say, a binary object file or digitized X-rays, determination of properly formed and therefore authentic plaintext may be difficult. Thus, an opponent could achieve a certain level of disruption simply by issuing messages with random content purporting to come from a legitimate user.

One solution to this problem is to force the plaintext to have some structure that is easily recognized but that cannot be replicated without recourse to the encryption function. We could, for example, append an error-detecting code, also known as a frame check sequence (FCS) or checksum, to each message before encryption, as illustrated in Figure 8.2a. A prepares a plaintext message  $M$  and then provides this as input to a function  $F$  that produces an FCS. The FCS is appended to  $M$  and the entire block is then encrypted. At the destination, B decrypts the incoming block and treats the results as a message with an appended FCS. B applies



**Figure 8.2** Internal and External Error Control.

the same function  $F$  to attempt to reproduce the FCS. If the calculated FCS is equal to the incoming FCS, then the message is considered authentic. It is unlikely that any random sequence of bits would exhibit the desired relationship.

Note that the order in which the FCS and encryption functions are performed is critical. The sequence illustrated in Figure 8.2a is referred to in [DIFF79] as internal error control, which the authors contrast with external error control (Figure 8.2b). With internal error control, authentication is provided because an opponent would have difficulty generating ciphertext that, when decrypted, would have valid error control bits. If instead the FCS is the outer code, an opponent can construct messages with valid error-control codes. Although the opponent cannot know what the decrypted plaintext will be, he or she can still hope to create confusion and disrupt operations.

An error-control code is just one example; in fact, any sort of structuring added to the transmitted message serves to strengthen the authentication capability. Such structure is provided by the use of a communications architecture consisting of layered protocols. As an example, consider the structure of messages transmitted using the TCP/IP protocol architecture. Figure 8.3 shows the format of a TCP segment, illustrating the TCP header. Now suppose that each pair of hosts shared a unique secret key, so that all exchanges between a pair of hosts used the same key, regardless of application. Then one could simply encrypt all of the datagram except the IP header (see Figure 5.5). Again, if an opponent substituted some arbitrary bit pattern for the encrypted TCP segment, the resulting plaintext would not include a meaningful header. In this case, the header includes not only a checksum (which covers the header) but other useful information, such as the sequence number. Because successive TCP segments on a given connection are numbered sequentially, encryption assures that an opponent does not delay, misorder, or delete any segments.

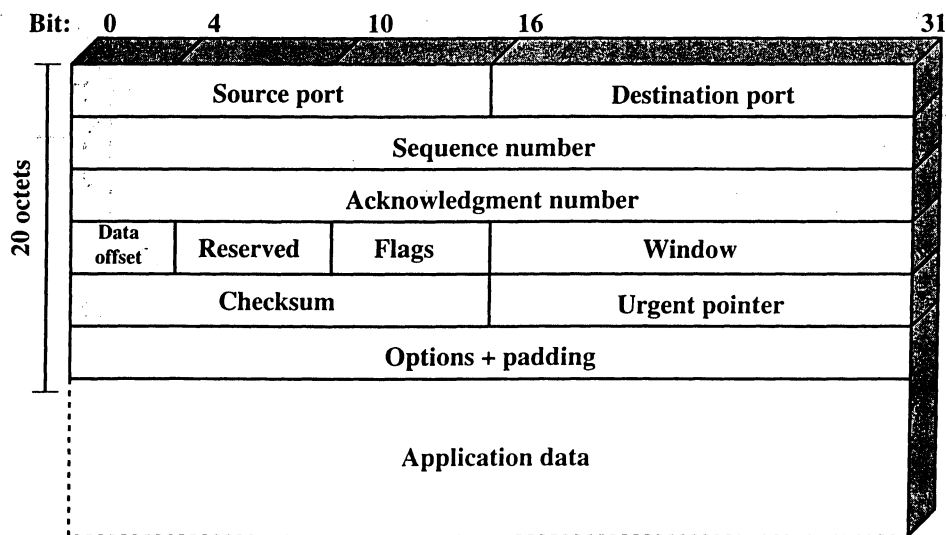


Figure 8.3 TCP Segment.

### Public-Key Encryption

The straightforward use of public-key encryption (Figure 8.1b) provides confidentiality but not authentication. The source (A) uses the public key  $KU_b$  of the destination (B) to encrypt  $M$ . Because only B has the corresponding private key  $KR_b$ , only B can decrypt the message. This scheme provides no authentication because any opponent could also use B's public key to encrypt a message, claiming to be A.

To provide authentication, A uses its private key to encrypt the message, and B uses A's public key to decrypt (Figure 8.1c). This provides a measure of authentication using the same type of reasoning as in the conventional encryption case: The message must have come from A because A is the only party that possesses  $KR_a$  and therefore the only party with the information necessary to construct ciphertext that can be decrypted with  $KU_a$ . Again, the same reasoning as before applies: There must be some internal structure to the plaintext so that the receiver can distinguish between well-formed plaintext and random bits.

Assuming there is such structure, then the scheme of Figure 8.1c does provide authentication. It also provides what is known as digital signature.<sup>1</sup> Only A could have constructed the ciphertext because only A possesses  $KR_a$ . Not even B, the recipient, could have constructed the ciphertext. Therefore, if B is in possession of the ciphertext, B has the means to prove that the message must have come from A. In effect, A has "signed" the message by using its private key to encrypt.

Note that this scheme does not provide confidentiality. Anyone in possession of A's public key can decrypt the ciphertext.

To provide both confidentiality and authentication, A can encrypt  $M$  first using its private key, which provides the digital signature, and then using B's public key, which provides confidentiality (Figure 8.1d). The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

Table 8.1 summarizes the confidentiality and authentication implications of these various approaches to message encryption.

### Message Authentication Code

An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a cryptographic checksum or MAC, that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key  $K$ . When A has a message to send to B, it calculates the MAC as a function of the message and the key:  $MAC = C_K(M)$ . The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC (Figure 8.4a). If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

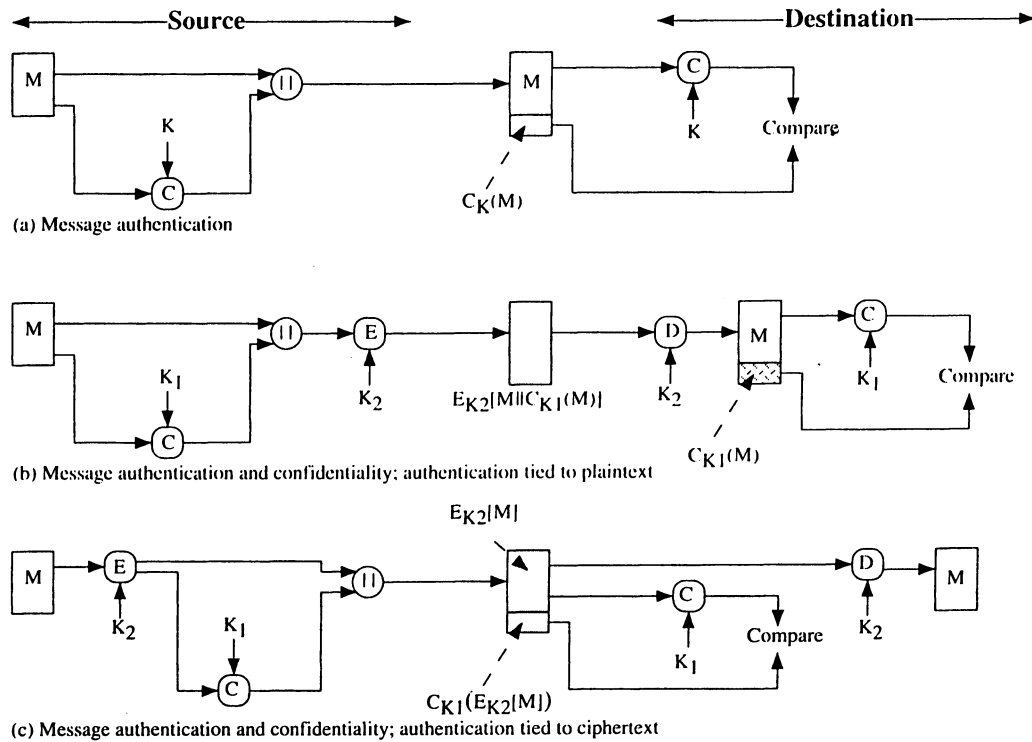
<sup>1</sup>This is not the way in which digital signatures are constructed, as we shall see, but the principle is the same.

**Table 8.1** Confidentiality and Authentication Implications of Message Encryption

(a) Conventional (symmetric) Encryption
$A \rightarrow B: E_K[M]$ <ul style="list-style-type: none"> <li>• Provides confidentiality <ul style="list-style-type: none"> <li>—Only A and B share K</li> </ul> </li> <li>• Provides a degree of authentication <ul style="list-style-type: none"> <li>—Could come only from A</li> <li>—Has not been altered in transit</li> <li>—Requires some formatting/redundancy</li> </ul> </li> <li>• Does not provide signature <ul style="list-style-type: none"> <li>—Receiver could forge message</li> <li>—Sender could deny message</li> </ul> </li> </ul>
(b) Public-Key (asymmetric) Encryption
$A \rightarrow B: E_{KU_b}[M]$ <ul style="list-style-type: none"> <li>• Provides confidentiality <ul style="list-style-type: none"> <li>—Only B has <math>KR_b</math> to decrypt</li> </ul> </li> <li>• Provides no authentication <ul style="list-style-type: none"> <li>—Any party could use <math>KU_b</math> to encrypt message and claim to be A</li> </ul> </li> </ul> $A \rightarrow B: E_{KR_a}[M]$ <ul style="list-style-type: none"> <li>• Provides authentication and signature <ul style="list-style-type: none"> <li>—Only A has <math>KR_a</math> to encrypt</li> <li>—Has not been altered in transit</li> <li>—Requires some formatting/redundancy</li> <li>—Any party can use <math>KU_a</math> to verify signature</li> </ul> </li> </ul> $A \rightarrow B: E_{KU_b}[E_{KR_a}(M)]$ <ul style="list-style-type: none"> <li>• Provides confidentiality because of <math>KU_b</math></li> <li>• Provides authentication and signature because of <math>KR_a</math></li> </ul>

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.
2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
3. If the message includes a sequence number (such as is used with HDLC, X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must for decryption. It turns out that because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption.



**Figure 8.4** Basic Uses of Message Authentication Code (MAC).

The process just described provides authentication but not confidentiality, because the message as a whole is transmitted in the clear. Confidentiality can be provided by performing message encryption either after (Figure 8.4b) or before (Figure 8.4c) the MAC algorithm. In both these cases, two separate keys are needed, each of which is shared by the sender and the receiver. In the first case, the MAC is calculated with the message as input and is then concatenated to the message. The entire block is then encrypted. In the second case, the message is encrypted first. Then the MAC is calculated using the resulting ciphertext and is concatenated to the ciphertext to form the transmitted block. Typically, it is preferable to tie the authentication directly to the plaintext, so the method of Figure 8.4b is used.

Because conventional encryption will provide authentication and because it is widely used with readily available products, why not simply use this instead of a separate message authentication code? [DAVI89] suggests three situations in which a message authentication code is used:

1. There are a number of applications in which the same message is broadcast to a number of destinations. Examples are notification to users that the network is now unavailable or an alarm signal in a military control center. It is cheaper and more reliable to have only one destination responsible for monitoring authenticity. Thus, the message must be broadcast in plaintext with an associ-

ated message authentication code. The responsible system has the secret key and performs authentication. If a violation occurs, the other destination systems are alerted by a general alarm.

2. Another possible scenario is an exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages. Authentication is carried out on a selective basis, messages being chosen at random for checking.
3. Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having to decrypt it every time, which would be wasteful of processor resources. However, if a message authentication code were attached to the program, it could be checked whenever assurance was required of the integrity of the program.

Three other rationales may be added, as follows:

4. For some applications, it may not be of concern to keep messages secret, but it is important to authenticate messages. An example is the Simple Network Management Protocol Version 3 (SNMPv3), which separates the functions of confidentiality and authentication. For this application, it is usually important for a managed system to authenticate incoming SNMP messages, particularly if the message contains a command to change parameters at the managed system. On the other hand, it may not be necessary to conceal the SNMP traffic.
5. Separation of authentication and confidentiality functions affords architectural flexibility. For example, it may be desired to perform authentication at the application level but to provide confidentiality at a lower level, such as the transport layer.
6. A user may wish to prolong the period of protection beyond the time of reception and yet allow processing of message contents. With message encryption, the protection is lost when the message is decrypted, so the message is protected against fraudulent modifications only in transit but not within the target system.

Finally, note that the MAC does not provide a digital signature because both sender and receiver share the same key.

Table 8.2 summarizes the confidentiality and authentication implications of the approaches illustrated in Figure 8.4.

### Hash Function

A variation on the message authentication code is the one-way hash function. As with the message authentication code, a hash function accepts a variable-size message  $M$  as input and produces a fixed-size hash code  $H(M)$ , sometimes called a message digest, as output. The hash code is a function of all the bits of the message and provides an error-detection capability: A change to any bit or bits in the message results in a change to the hash code.

Figure 8.5 illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows:

**Table 8.2** Basic Uses of Message Authentication Code C

<p>(a) <math>A \rightarrow B: M \parallel C_K(M)</math></p> <ul style="list-style-type: none"> <li>• Provides authentication —Only A and B share K</li> </ul>
<p>(b) <math>A \rightarrow B: E_{K_2} [M \parallel C_{K_1}(M)]</math></p> <ul style="list-style-type: none"> <li>• Provides authentication —Only A and B share <math>K_1</math></li> <li>• Provides confidentiality —Only A and B share <math>K_2</math></li> </ul>
<p>(c) <math>A \rightarrow B: E_{K_2} [M] \parallel C_{K_1}(E_{K_2}[M])</math></p> <ul style="list-style-type: none"> <li>• Provides authentication —Using <math>K_1</math></li> <li>• Provides confidentiality —Using <math>K_2</math></li> </ul>

- a. The message plus concatenated hash code is encrypted using conventional encryption. This is identical in structure to the internal error-control strategy shown in Figure 8.2a. The same line of reasoning applies: Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.
- b. Only the hash code is encrypted, using conventional encryption. This reduces the processing burden for those applications that do not require confidentiality. Note that the combination of hashing and encryption results in an overall function that is, in fact, a MAC (Figure 8.4a). That is,  $E_K[H(M)]$  is a function of a variable-length message  $M$  and a secret key  $K$ , and it produces a fixed-size output that is secure against an opponent who does not know the secret key.
- c. Only the hash code is encrypted, using public-key encryption and using the sender's private key. As with (b), this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.
- d. If confidentiality as well as a digital signature is desired, then the message plus the public-key-encrypted hash code can be encrypted using a conventional secret key.
- e. This technique uses a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value  $S$ . A computes the hash value over the concatenation of  $M$  and  $S$  and appends the resulting hash value to  $M$ . Because B possesses  $S$ , it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

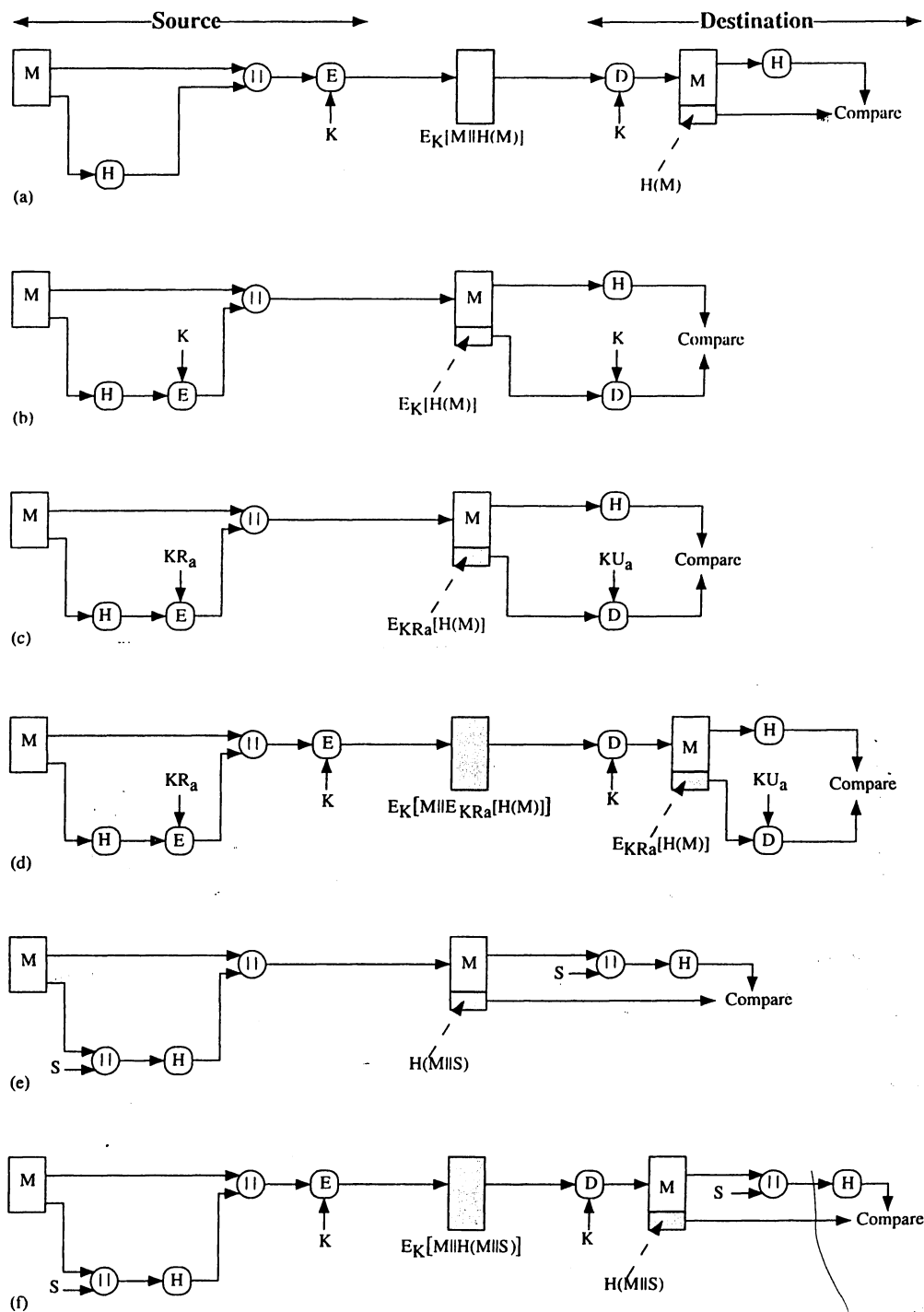


Figure 8.5 Basic Uses of Hash Function.



- f. Confidentiality can be added to the approach of (e) by encrypting the entire message plus the hash code.

When confidentiality is not required, methods (b) and (c) have an advantage over those that encrypt the entire message in that less computation is required. Nevertheless, there has been growing interest in techniques that avoid encryption (Figure 8.5e). Several reasons for this interest are pointed out in [TSUD92]:

- Encryption software is quite slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
- Encryption hardware costs are not negligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.
- Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
- Encryption algorithms may be covered by patents. Some encryption algorithms, such as the RSA public-key algorithm, are patented and must be licensed, adding a cost.
- Encryption algorithms are subject to U.S. export control.

Table 8.3 summarizes the confidentiality and authentication implications of the approaches illustrated in Figure 8.5.

### 8.3 MESSAGE AUTHENTICATION CODES

A MAC, also known as a cryptographic checksum, is generated by a function  $C$  of the form

$$\text{MAC} = C_K(M)$$

**Table 8.3** Basic Uses of Hash Function  $H$

<p>(a) <math>A \rightarrow B: E_K[M \parallel H(M)]</math></p> <ul style="list-style-type: none"> <li>• Provides confidentiality —Only A and B share <math>K</math></li> <li>• Provides authentication —<math>H(M)</math> is cryptographically protected</li> </ul>	<p>(d) <math>A \rightarrow B: E_K[M \parallel E_{KRa}[H(M)]]</math></p> <ul style="list-style-type: none"> <li>• Provides authentication and digital signature</li> <li>• Provides confidentiality —Only A and B share <math>K</math></li> </ul>
<p>(b) <math>A \rightarrow B: M \parallel E_K[H(M)]</math></p> <ul style="list-style-type: none"> <li>• Provides authentication —<math>H(M)</math> is cryptographically protected</li> </ul>	<p>(e) <math>A \rightarrow B: M \parallel H(M \parallel S)</math></p> <ul style="list-style-type: none"> <li>• Provides authentication —Only A and B share <math>S</math></li> </ul>
<p>(c) <math>A \rightarrow B: M \parallel E_{KRa}[H(M)]</math></p> <ul style="list-style-type: none"> <li>• Provides authentication and digital signature —<math>H(M)</math> is cryptographically protected —Only A could create <math>E_{KRa}[H(M)]</math></li> </ul>	<p>(f) <math>A \rightarrow B: E_K[M \parallel H(M) \parallel S]</math></p> <ul style="list-style-type: none"> <li>• Provides authentication —Only A and B share <math>S</math></li> <li>• Provides confidentiality —Only A and B share <math>K</math></li> </ul>

where  $M$  is a variable-length message,  $K$  is a secret key shared only by sender and receiver, and  $C_K(M)$  is the fixed-length authenticator. The MAC is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the MAC.

In this section, we review the requirements for the function  $C$  and then examine a specific example. Another example is discussed in Chapter 9.

### Requirements for MACs

When an entire message is encrypted for confidentiality, using either symmetric or asymmetric encryption, the security of the scheme generally depends on the bit length of the key. Barring some weakness in the algorithm, the opponent must resort to a brute-force attack using all possible keys. On average, such an attack will require  $2^{(k-1)}$  attempts for a  $k$ -bit key. In particular, for a ciphertext-only attack, the opponent, given ciphertext  $C$ , would perform  $P_i = D_{K_i}(C)$  for all possible key values  $K_i$  until a  $P_i$  was produced that matched the form of acceptable plaintext.

In the case of a MAC, the considerations are entirely different. In general, the MAC function is a many-to-one function. The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys. If an  $n$ -bit MAC is used, then there are  $2^n$  possible MACs, whereas there are  $N$  possible messages with  $N \gg 2^n$ . Furthermore, with a  $k$ -bit key, there are  $2^k$  possible keys.

Using brute-force methods, how would an opponent attempt to discover a key? If confidentiality is not employed, the opponent has access to plaintext messages and their associated MACs. Suppose  $k > n$ ; that is, suppose that the key size is greater than the MAC size. Then, given a known  $M_1$  and  $MAC_1$ , with  $MAC_1 = C_{K_1}(M_1)$ , the cryptanalyst can perform  $MAC_i = C_{K_i}(M_1)$  for all possible key values  $K_i$ . At least one key is guaranteed to produce a match of  $MAC_i = MAC_1$ . Note that a total of  $2^k$  MACs will be produced but there are only  $2^n < 2^k$  different MAC values. Thus, a number of keys will produce the correct MAC, and the opponent has no way of knowing which is the correct key. On average, a total of  $2^k/2^n = 2^{(k-n)}$  keys will produce a match. Thus, the opponent must iterate the attack:

- **Round 1**
  - Given:  $M_1$ ,  $MAC_1 = C_K(M_1)$
  - Compute  $MAC_i = C_{K_i}(M_1)$  for all  $2^k$  keys
  - Number of matches  $\approx 2^{(k-n)}$
- **Round 2**
  - Given:  $M_2$ ,  $MAC_2 = C_K(M_2)$
  - Compute  $MAC_i = C_{K_i}(M_2)$  for the remaining  $2^{(k-n)}$  keys
  - Number of matches  $\approx 2^{(k-2 \times n)}$

and so on. On average,  $\alpha$  rounds will be needed if  $k = \alpha \times n$ . For example, if an 80-bit key is used and the MAC is 32 bits long, then the first round will produce about  $2^{48}$  possible keys. The second round will narrow the possible keys to about  $2^{16}$  possibilities. The third round should produce only a single key, which must be the one used by the sender.

If the key length is less than or equal to the MAC length, then it is likely that a first round will produce a single match. It is possible that more than one key will produce such a match, in which case the opponent would need to perform the same test on a new (message, MAC) pair.

Thus, a brute-force attempt to discover the authentication key is no less effort and may be more effort than that required to discover a decryption key of the same length. However, other attacks that do not require the discovery of the key are possible.

Consider the following MAC algorithm. Let  $M = (X_1 || X_2 || \dots || X_m)$  be a message that is treated as a concatenation of 64-bit blocks  $X_i$ . Then define

$$\begin{aligned}\Delta(M) &= X_1 \oplus X_2 \oplus \dots \oplus X_m \\ C_K(M) &= E_K[\Delta(M)]\end{aligned}$$

where  $\oplus$  is the exclusive-OR (XOR) operation and the encryption algorithm is DES in electronic codebook mode. Thus, the key length is 56 bits and the MAC length is 64 bits. If an opponent observes  $\{M || C_K(M)\}$ , a brute-force attempt to determine  $K$  will require at least  $2^{56}$  encryptions. But the opponent can attack the system by replacing  $X_1$  through  $X_{m-1}$  with any desired values  $Y_1$  through  $Y_{m-1}$  and replacing  $X_m$  with  $Y_m$ , where  $Y_m$  is calculated as follows:

$$Y_m = Y_1 \oplus Y_2 \oplus \dots \oplus Y_{m-1} \oplus \Delta(M)$$

The opponent can now concatenate the new message, which consists of  $Y_1$  through  $Y_m$ , with the original MAC to form a message that will be accepted as authentic by the receiver. With this tactic, any message of length  $64 \times (m-1)$  bits can be fraudulently inserted.

Thus, in assessing the security of a MAC function, we need to consider the types of attacks that may be mounted against it. With that in mind, let us state the requirements for the function. Assume that an opponent knows the MAC function  $C$  but does not know  $K$ . Then the MAC function should have the following properties:

1. If an opponent observes  $M$  and  $C_K(M)$ , it should be computationally infeasible for the opponent to construct a message  $M'$  such that  $C_K(M') = C_K(M)$ .
2.  $C_K(M)$  should be uniformly distributed in the sense that for randomly chosen messages,  $M$  and  $M'$ , the probability that  $C_K(M) = C_K(M')$  is  $2^{-n}$ , where  $n$  is the number of bits in the MAC.
3. Let  $M'$  be equal to some known transformation on  $M$ . That is,  $M' = f(M)$ . For example,  $f$  may involve inverting one or more specific bits. In that case,  $\Pr[C_K(M) = C_K(M')] = 2^{-n}$ .

The first requirement speaks to the earlier example, in which an opponent is able to construct a new message to match a given MAC, even though the opponent does not know and does not learn the key. Requirement (2) deals with the need to thwart a brute-force attack based on chosen plaintext. That is, if we assume that the opponent does not know  $K$  but does have access to the MAC function and can present messages for MAC generation, then the opponent could try various messages

until finding one that matches a given MAC. If the MAC function exhibits uniform distribution, then a brute-force method would require, on average,  $2^{(n-1)}$  attempts before finding a message that fits a given MAC.

The final requirement dictates that the authentication algorithm should not be weaker with respect to certain parts or bits of the message than others. If this were not the case, then an opponent who had  $M$  and  $C_K(M)$  could attempt variations on  $M$  at the known “weak spots” with a likelihood of early success at producing a new message that matched the old MAC.

### Message Authentication Code Based on DES

One of the most widely used MACs, referred to as the Data Authentication Algorithm, is based on DES. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17).

The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero. The data (e.g., message, record, file, or program) to be authenticated is grouped into contiguous 64-bit blocks:  $D_1, D_2, \dots, D_N$ . If necessary, the final block is padded on the right with zeroes to form a full 64-bit block. Using the DES encryption algorithm,  $E$ , and a secret key,  $K$ , a data authentication code (DAC) is calculated as follows (Figure 8.6):

$$\begin{aligned} O_1 &= E_K(D_1) \\ O_2 &= E_K(D_2 \oplus O_1) \\ O_3 &= E_K(D_3 \oplus O_2) \\ &\vdots \\ O_N &= E_K(D_N \oplus O_{N-1}) \end{aligned}$$

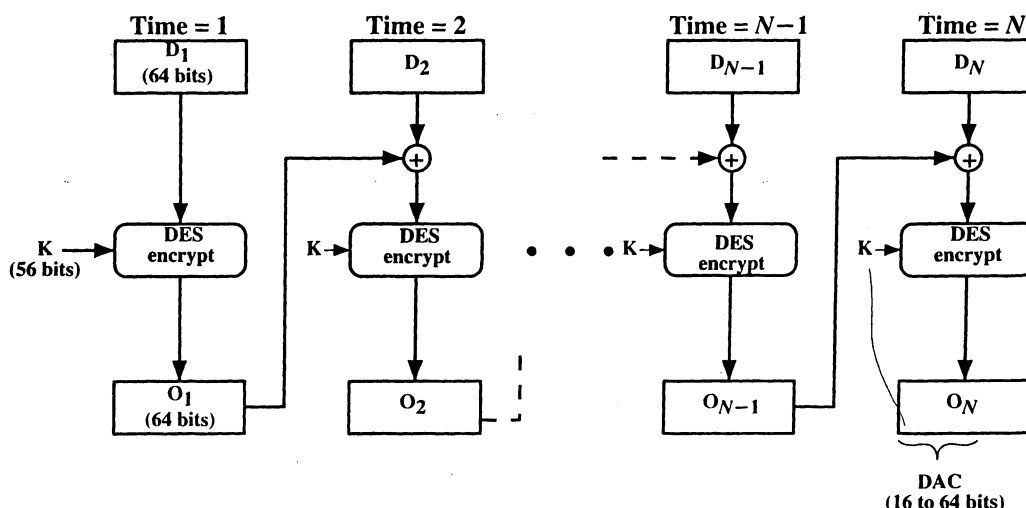


Figure 8.6 Data Authentication Algorithm (FIPS PUB 113).

The DAC consists of either the entire block  $O_N$  or the leftmost  $M$  bits of the block, with  $16 \leq M \leq 64$ .

This algorithm appears to meet the requirements specified earlier.

## 8.4 HASH FUNCTIONS

A hash value is generated by a function  $H$  of the form

$$h = H(M)$$

where  $M$  is a variable-length message and  $H(M)$  is the fixed-length hash value. The hash value is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the hash value. Because the hash function itself is not considered to be secret, some means is required to protect the hash value (Figure 8.5).

We begin by examining the requirements for a hash function to be used for message authentication. Because hash functions are, typically, quite complex, it is useful to examine next some very simple hash functions to get a feel for the issues involved. We then look at several approaches to hash function design.

### Requirements for a Hash Function

The purpose of a hash function is to produce a “fingerprint” of a file, message, or other block of data. To be useful for message authentication, a hash function  $H$  must have the following properties (adapted from a list in [NECH92]):

1.  $H$  can be applied to a block of data of any size.
2.  $H$  produces a fixed-length output.
3.  $H(x)$  is relatively easy to compute for any given  $x$ , making both hardware and software implementations practical.
4. For any given code  $h$ , it is computationally infeasible to find  $x$  such that  $H(x) = h$ . This is sometimes referred to in the literature as the **one-way** property.
5. For any given block  $x$ , it is computationally infeasible to find  $y \neq x$  with  $H(y) = H(x)$ . This is sometimes referred to as **weak collision resistance**.
6. It is computationally infeasible to find any pair  $(x, y)$  such that  $H(x) = H(y)$ . This is sometimes referred to as **strong collision resistance**.<sup>2</sup>

The first three properties are requirements for the practical application of a hash function to message authentication.

The fourth property is the one-way property: It is easy to generate a code given a message but virtually impossible to generate a message given a code. This

<sup>2</sup>Unfortunately, these terms are not used consistently. Alternate terms used in the literature include *one-way hash function* (properties 4 and 5); *collision-resistant hash function* (properties 4, 5, and 6); *weak one-way hash function* (properties 4 and 5); *strong one-way hash function* (properties 4, 5, and 6). The reader must take care in reading the literature to determine the meaning of the particular terms used.

property is important if the authentication technique involves the use of a secret value (Figure 8.5e). The secret value itself is not sent; however, if the hash function is not one way, an attacker can easily discover the secret value: If the attacker can observe or intercept a transmission, the attacker obtains the message  $M$  and the hash code  $C = H(S_{AB}||M)$ . The attacker then inverts the hash function to obtain  $S_{AB}||M = H^{-1}(C)$ . Because the attacker now has both  $M$  and  $S_{AB}||M$ , it is a trivial matter to recover  $S_{AB}$ .

The fifth property guarantees that an alternative message hashing to the same value as a given message cannot be found. This prevents forgery when an encrypted hash code is used (Figure 8.5b and 8.5c). For these cases, the opponent can read the message and therefore generate its hash code. However, because the opponent does not have the secret key, the opponent should not be able to alter the message without detection. If this property were not true, an attacker would be capable of the following sequence: First, observe or intercept a message plus its encrypted hash code; second, generate an unencrypted hash code from the message; third, generate an alternate message with the same hash code.

The sixth property refers to how resistant the hash function is to a class of attack known as the birthday attack, which we examine shortly.

### Simple Hash Functions

All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of  $n$ -bit blocks. The input is processed one block at a time in an iterative fashion to produce an  $n$ -bit hash function.

One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as follows:

$$C_i = b_{i1} \oplus b_{i2} \oplus \dots \oplus b_{im}$$

where

$$\begin{aligned} C_i &= i\text{th bit of the hash code, } 1 \leq i \leq n \\ m &= \text{number of } n\text{-bit blocks in the input} \\ b_{ij} &= i\text{th bit in } j\text{th block} \\ \oplus &= \text{XOR operation} \end{aligned}$$

Figure 8.7 illustrates this operation; it produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check. Each  $n$ -bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is  $2^{-n}$ . With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of  $2^{-128}$ , the hash function on this type of data has an effectiveness of  $2^{-112}$ .

A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows:

	Bit 1	Bit 2	• • •	Bit $n$
Block 1	$b_{11}$	$b_{21}$		$b_{n1}$
Block 2	$b_{12}$	$b_{22}$		$b_{n2}$
	•	•	•	•
	•	•	•	•
	•	•	•	•
Block $m$	$b_{1m}$	$b_{2m}$		$b_{nm}$
Hash code	$C_1$	$C_2$		$C_n$

Figure 8.7 Simple Hash Function Using Bitwise XOR.

1. Initially set the  $n$ -bit hash value to zero.
2. Process each successive  $n$ -bit block of data as follows:
  - a. Rotate the current hash value to the left by one bit.
  - b. XOR the block into the hash value.

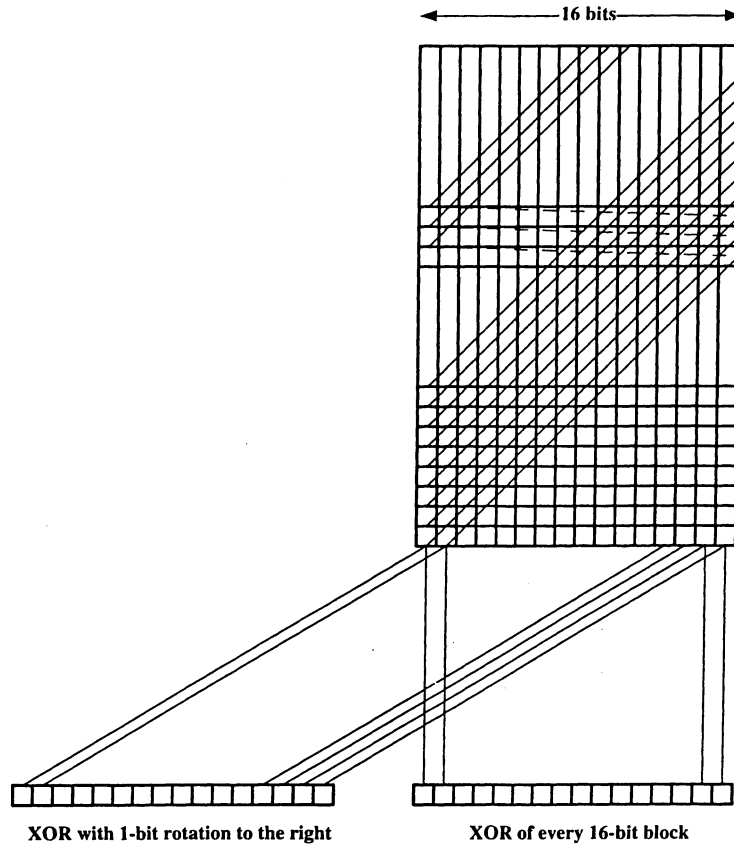
This has the effect of “randomizing” the input more completely and overcoming any regularities that appear in the input. Figure 8.8 illustrates these two types of hash functions for 16-bit hash values.

Although the second procedure provides a good measure of data integrity, it is virtually useless for data security when an encrypted hash code is used with a plaintext message, as in Figures 8.5b and 8.5c. Given a message, it is an easy matter to produce a new message that yields that hash code: Simply prepare the desired alternate message and then append an  $n$ -bit block that forces the new message plus block to yield the desired hash code.

Although a simple XOR or rotated XOR (RXOR) is insufficient if only the hash code is encrypted, you may still feel that such a simple function could be useful when the message as well as the hash code are encrypted (Figure 8.5a). But one must be careful. A technique originally proposed by the National Bureau of Standards used the simple XOR applied to 64-bit blocks of the message and then an encryption of the entire message that used the cipher block chaining (CBC) mode. We can define the scheme as follows: Given a message consisting of a sequence of 64-bit blocks  $X_1, X_2, \dots, X_N$ , define the hash code  $C$  as the block-by-block XOR of all blocks and append the hash code as the final block:

$$C = X_{N+1} = X_1 \oplus X_2 \oplus \dots \oplus X_N$$

Next, encrypt the entire message plus hash code, using CBC mode to produce the encrypted message  $Y_1, Y_2, \dots, Y_{N+1}$ . [JUEN85] points out several ways in which the ciphertext of this message can be manipulated in such a way that it is not detectable by the hash code. For example, by the definition of CBC (Figure 3.12), we have



**Figure 8.8** Two Simple Hash Functions.

$$X_1 = IV \oplus D_K(Y_1)$$

$$X_i = Y_{i-1} \oplus D_K(Y_i)$$

$$X_{N+1} = Y_N \oplus D_K(Y_{N+1})$$

But  $X_{N+1}$  is the hash code:

$$\begin{aligned} X_{N+1} &= X_1 \oplus X_2 \oplus \dots \oplus X_N \\ &= (IV \oplus D_K(Y_1)) \oplus (Y_1 \oplus D_K(Y_2)) \oplus \dots \oplus (Y_{N-1} \oplus D_K(Y_N)) \end{aligned}$$

Because the terms in the preceding equation can be XORed in any order, it follows that the hash code would not change if the ciphertext blocks were permuted.

### Birthday Attacks

Suppose that a 64-bit hash code is used. One might think that this is quite secure. For example, if an encrypted hash code  $C$  is transmitted with the corresponding



unencrypted message  $M$  (Figure 8.5b or 8.5c), then an opponent would need to find an  $M'$  such that  $H(M') = H(M)$  to substitute another message and fool the receiver. On average, the opponent would have to try about  $2^{63}$  messages to find one that matches the hash code of the intercepted message [see Appendix 8A; Equation (8.1)].

However, a different sort of attack is possible, based on the birthday paradox (Appendix 8A). Yuval proposed the following strategy [YUVA79]:

1. The source, A, is prepared to “sign” a message by appending the appropriate  $m$ -bit MAC and encrypting that MAC with A’s private key (Figure 8.5c).
2. The opponent generates  $2^{m/2}$  variations on the message, all of which convey essentially the same meaning. The opponent prepares an equal number of messages, all of which are variations on the fraudulent message to be substituted for the real one.
3. The two sets of messages are compared to find a pair of messages that produces the same hash code. The probability of success, by the birthday paradox, is greater than 0.5. If no match is found, additional valid and fraudulent messages are generated until a match is made.
4. The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known.

Thus, if a 64-bit hash code is used, the level of effort required is only on the order of  $2^{32}$  [see Appendix 8A; Equation (8.7)].

The generation of many variations that convey the same meaning is not difficult. For example, the opponent could insert a number of “space-space-backspace” character pairs between words throughout the document. Variations could then be generated by substituting “space-backspace-space” in selected instances. Alternatively, the opponent could simply reword the message but retain the meaning. Figure 8.9 [DAVI89] provides an example.

The conclusion to be drawn from this is that the length of the hash code should be substantial. We discuss this further in Section 8.5.

### Block Chaining Techniques

A number of proposals have been made for hash functions based on using a cipher block chaining technique, but without the secret key. One of the first such proposals was that of Rabin [RABI78]. Divide a message  $M$  into fixed-size blocks  $M_1, M_2, \dots, M_N$  and use a conventional encryption system such as DES to compute the hash code  $G$  as follows:

$$H_0 = \text{initial value}$$

$$H_i = E_{M_i}[H_{i-1}]$$

$$G = H_N$$

Dear Anthony,

{ This letter is } to introduce { you to } { Mr. } Alfred { P. }  
 { I am writing } { to you } { -- }  
 Barton, the { new } { chief } jewellery buyer for { our }  
 { newly appointed } { senior } { the }  
 Northern { European } { area } . He { will take } over { the }  
 { Europe } { division } { has taken }  
 responsibility for { the whole of } our interests in { watches and jewellery }  
 { jewellery and watches }  
 in the { area } . Please { afford } him { every } help he { may need }  
 { region } { give } { all the } { needs }  
 to { seek out } the most { modern } lines for the { top } end of the  
 { find } { up to date } { high }  
 market. He is { empowered } to receive on our behalf { samples } of the  
 { authorized } { specimens }  
 { latest } { watch and jewellery } products, { up } to a { limit }  
 { newest } { jewellery and watch } { subject } { maximum }  
 of ten thousand dollars. He will { carry } a signed copy of this { letter }  
 { hold } { document }  
 as proof of identity. An order with his signature, which is { appended }  
 { attached }  
 { authorizes } you to charge the cost to this company at the { above }  
 { allows } { head office }  
 address. We { fully } expect that our { level } of orders will increase in  
 { -- } { volume }  
 the { following } year and { trust } that the new appointment will { be }  
 { next } { hope } { prove }  
 { advantageous } to both our companies.  
 { an advantage }

Figure 8.9 A Letter in  $2^{37}$  Variations [DAVI89].

This is similar to the CBC technique, but in this case there is no secret key. As with any hash code, this scheme is subject to the birthday attack, and if the encryption algorithm is DES and only a 64-bit hash code is produced, then the system is vulnerable.

Furthermore, another version of the birthday attack can be used even if the opponent has access to only one message and its valid signature and cannot obtain multiple signings. Here is the scenario; we assume that the opponent intercepts a message with a signature in the form of an encrypted hash code and that the unencrypted hash code is  $m$  bits long:

1. Use the algorithm defined at the beginning of this subsection to calculate the unencrypted hash code  $G$ .
2. Construct any desired message in the form  $Q_1, Q_2, \dots, Q_{N-2}$ .
3. Compute  $H_i = E_{Q_i}[H_{i-1}]$  for  $1 \leq i \leq (N-2)$ .
4. Generate  $2^{m/2}$  random blocks; for each block  $X$ , compute  $E_X[H_{N-2}]$ . Generate an additional  $2^{m/2}$  random blocks; for each block  $Y$ , compute  $D_Y[G]$ , where  $D$  is the decryption function corresponding to  $E$ .

5. Based on the birthday paradox, with high probability there will be an  $X$  and  $Y$  such that  $E_X[H_{V-2}] = D_Y[G]$ .
6. Form the message  $Q_1, Q_2, \dots, Q_{N-2}, X, Y$ . This message has the hash code  $G$  and therefore can be used with the intercepted encrypted signature.

This form of attack is known as a “meet in the middle” attack. A number of researchers have proposed refinements intended to strengthen the basic block chaining approach. For example, Davies and Price [DAV89] describe the following variation:

$$H_i = E_{M_i}[H_{i-1}] \oplus H_{i-1}$$

Another variation, proposed in [MEYE88], is as follows:

$$H_i = E_{H_{i-1}}[M_i] \oplus M_i$$

However, both of these schemes have been shown to be vulnerable to a variety of attacks [MIYA90]. More generally, it can be shown that some form of birthday attack will succeed against any hash scheme involving the use of cipher block chaining without a secret key provided that either the resulting hash code is small enough (e.g., 64 bits or less) or that a larger hash code can be decomposed into independent subcodes [JUN87].

Thus, attention has been directed at finding other approaches to hashing. Many of these have also been shown to have weaknesses [MITC92]. We examine some strong hash functions in Chapter 9.

## 8.5 SECURITY OF HASH FUNCTIONS AND MACS

Just as with conventional and public-key encryption, we can group attacks on hash functions and MACs into two categories: brute-force attacks and cryptanalysis.

### Brute-Force Attacks

The nature of brute-force attacks differs somewhat for hash functions and MACs.

#### Hash Functions

The strength of a hash function against brute-force attacks depends solely on the length of the hash code produced by the algorithm. Recall from our discussion of hash functions that there are three desirable properties:

- **One-way:** For any given code  $h$ , it is computationally infeasible to find  $x$  such that  $H(x) = h$ .
- **Weak collision resistance:** For any given block  $x$ , it is computationally infeasible to find  $y \neq x$  with  $H(y) = H(x)$ .

- **Strong collision resistance:** It is computationally infeasible to find any pair  $(x, y)$  such that  $H(x) = H(y)$ .

For a code of length  $n$ , the level of effort required, as we have seen, is proportional to the following:

One way	$2^n$
Weak collision resistance	$2^n$
Strong collision resistance	$2^{n/2}$

If strong collision resistance is required (and this is desirable for a general-purpose secure hash code), then the value  $2^{n/2}$  determines the strength of the hash code against brute-force attacks. Oorschot and Wiener [OORS94] presented a design for a \$10 million collision search machine for MD5, which has a 128-bit hash length, that could find a collision in 24 days. Thus a 128-bit code may be viewed as inadequate. The next step up, if a hash code is treated as a sequence of 32 bits, is a 160-bit hash length. With a hash length of 160 bits, the same search machine would require over four thousand years to find a collision. Currently, the two most popular hash codes, SHA-1 and RIPEMD-160, discussed in Chapter 9, provide a 160-bit hash code length.

#### Message Authentication Codes

A brute-force attack on a MAC is a more difficult undertaking because it requires known message-MAC pairs. Let us see why this is so. To attack a hash code, one can proceed in the following way. Given a fixed message  $x$  with  $n$ -bit hash code  $h = H(x)$ , a brute-force method of finding a collision is to pick a random bit string  $y$  and check if  $H(y) = H(x)$ . The attacker can do this repeatedly off line. Whether an off-line attack can be used on a MAC algorithm depends on the relative size of the key and the MAC.

To proceed, we need to state the desired security property of a MAC algorithm, which can be expressed as follows:

- **Computation resistance:** Given one or more text-MAC pairs  $(x_i, C_K(x_i))$ , it is computationally infeasible to compute any text-MAC pair  $(x, C_K(x))$  for any new input  $x \neq x_i$ .

In other words, the attacker would like to come up with the valid MAC code for a given message  $x$ . There are two lines of attack possible: Attack the key space and attack the MAC value. We examine each of these in turn.

If an attacker can determine the MAC key, then it is possible to generate a valid MAC value for any input  $x$ . Suppose the key size is  $k$  bits and that the attacker has one known text-MAC pair. Then the attacker can compute the  $n$ -bit MAC on the known text for all possible keys. At least one key is guaranteed to produce the correct MAC — namely, the valid key that was initially used to produce the known text-MAC pair. This phase of the attack takes a level of effort proportional to  $2^k$  (that is, one operation for each of the  $2^k$  possible key values). However, as was described earlier, because the MAC is a many-to-one mapping, there may be other

keys that produce the correct value. Thus, if more than one key is found to produce the correct value, additional text-MAC pairs must be tested. It can be shown that the level of effort drops off rapidly with each additional text-MAC pair and that the overall level of effort is roughly  $2^k$  [MENE97].

An attacker can also work on the MAC value without attempting to recover the key. Here, the objective is to generate a valid MAC value for a given message or to find a message that matches a given MAC value. In either case, the level of effort is comparable to that for attacking the one-way or weak collision resistant property of a hash code, or  $2^n$ . In the case of the MAC, the attack cannot be conducted off line without further input; the attacker will require chosen text-MAC pairs or knowledge of the key.

To summarize, the level of effort for brute-force attack on a MAC algorithm can be expressed as  $\min(2^k, 2^n)$ . The assessment of strength is similar to that for symmetric encryption algorithms. It would appear reasonable to require that the key length and MAC length satisfy a relationship such as  $\min(k, n) \geq N$ , where  $N$  is perhaps in the range of 128 bits.

### Cryptanalysis

As with encryption algorithms, cryptanalytic attacks on hash functions and MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. The way to measure the resistance of a hash or MAC algorithm to cryptanalysis is to compare its strength to the effort required for a brute-force attack. That is, an ideal hash or MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

#### Hash Functions

In recent years, there has been considerable effort, and some successes, in developing cryptanalytic attacks on hash functions. To understand these, we need to look at the overall structure of a typical secure hash function, indicated in Figure 8.10.

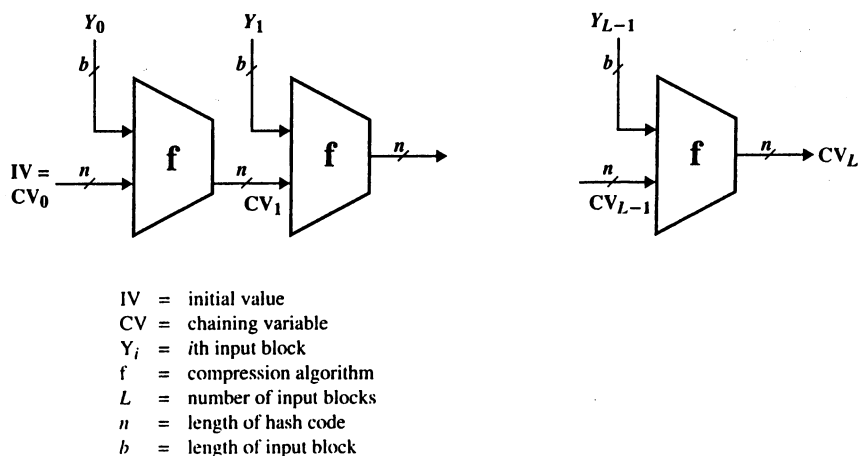


Figure 8.10 General Structure of Secure Hash Code.

This structure, referred to as an iterated hash function, was proposed by Merkle [MERK79, MERK89] and is the structure of most hash functions in use today, including MD5, SHA-1, and RIPEMD-160, all of which are described in Chapter 9. The hash function takes an input message and partitions it into  $L - 1$  fixed-sized blocks of  $b$  bits each. If necessary, the final block is padded to  $b$  bits. The final block also includes the value of the total length of the input to the hash function. The inclusion of the length makes the job of the opponent more difficult. Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the same value.

The hash algorithm involves repeated use of a *compression function*,  $f$ , that takes two inputs (an  $n$ -bit input from the previous step, called the *chaining variable*, and a  $b$ -bit block) and produces an  $n$ -bit output. At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Usually,  $b > n$ ; hence the term *compression*. The hash function can be summarized as follows:

$$\begin{aligned} CV_0 &= IV = \text{initial } n\text{-bit value} \\ CV_i &= f(CV_{i-1}, Y_{i-1}) \quad 1 \leq i \leq L \\ H(M) &= CV_L \end{aligned}$$

where the input to the hash function is a message  $M$  consisting of the blocks  $Y_0, Y_1, \dots, Y_{L-1}$ .

The motivation for this iterative structure stems from the observation by Merkle [MERK89] and Damgård [DAMG89] that if the compression function is collision resistant, then so is the resultant iterated hash function.<sup>3</sup> Therefore, the structure can be used to produce a secure hash function to operate on a message of any length. The problem of designing a secure hash function reduces to that of designing a collision-resistant compression function that operates on inputs of some fixed size.

Cryptanalysis of hash functions focuses on the internal structure of  $f$  and is based on attempts to find efficient techniques for producing collisions for a single execution of  $f$ . Once that is done, the attack must take into account the fixed value of  $IV$ . The attack on  $f$  depends on exploiting its internal structure. Typically, as with symmetric block ciphers,  $f$  consists of a series of rounds of processing, so that the attack involves analysis of the pattern of bit changes from round to round.

Keep in mind that for any hash function there must exist collisions, because we are mapping a message of length at least equal to the block size  $b$  into a hash code of length  $n$ , where  $b < n$ . What is required is that it is computationally infeasible to find collisions.

The attacks that have been mounted on hash functions are rather complex and beyond our scope here. For the interested reader, [DOBB96a] and [BELL97] are recommended.

<sup>3</sup>The converse is not necessarily true.

### Message Authentication Codes

There is much more variety in the structure of MACs than in hash functions, so it is difficult to generalize about the cryptanalysis of MACs. Further, far less work has been done on developing such attacks. A useful recent survey of some methods for specific MACs is [PREN96].

## 8.6 RECOMMENDED READING

[JUE85] and [JUE87] provide a good background on message authentication, with a focus on cryptographic MACs and hash functions. Solid treatments of hash functions and message authentication codes are found in [STIN95] and [MENE97].

JUE85 Jueneman, R.; Matyas, S.; and Meyer, C. "Message Authentication." *IEEE Communications Magazine*, September 1988.

JUE87 Jueneman, R. "Electronic Document Authentication." *IEEE Network Magazine*, April 1987.

MENE97 Menezes, A.; Oorschot, P.; and Vanstone, S. *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1997.

STIN95 Stinson, D. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 1995.

## 8.7 PROBLEMS

- 8.1 If  $F$  is an error-detection function, either internal or external use (Figure 8.2) will provide error-detection capability. If any bit of the transmitted message is altered, this will be reflected in a mismatch of the received FCS and the calculated FCS, whether the FCS function is performed inside or outside the encryption function. Some codes also provide an error-correction capability. Depending on the nature of the function, if one or a small number of bits is altered in transit, the error-correction code contains sufficient redundant information to determine the errored bit or bits and correct them. Clearly, an error-correction code will provide error-correction capability when used external to the encryption function. Will it also provide this capability if used internal to the encryption function?
- 8.2 The data authentication algorithm, described in Section 8.3, can be defined as using the cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero (Figure 8.6). Show that the same result can be produced using the cipher feedback mode.
- 8.3 The high-speed transport protocol XTP (Xpress Transfer Protocol) uses a 32-bit checksum function defined as the concatenation of two 16-bit functions: XOR and RXOR, defined in Section 8.4 as "two simple hash functions" and illustrated in Figure 8.8.
  - a. Will this checksum detect all errors caused by an odd number of error bits? Explain.
  - b. Will this checksum detect all errors caused by an even number of error bits? If not, characterize the error patterns that will cause the checksum to fail.
  - c. Comment on the effectiveness of this function for use as a hash function for authentication.
- 8.4 a. Consider the Davies and Price hash code scheme described in Section 8.4 and assume that DES is used as the encryption algorithm:

$$H_i = E_M[H_{i-1}] \oplus H_{i-1}$$

and recall the complementarity property of DES (Problem 3.10): If  $Y = \text{DES}_K(X)$ , then  $Y' = \text{DES}_K(X')$ . Use this property to show how a message consisting of blocks  $M_1, M_2, \dots, M_N$  can be altered without altering its hash code.

b. Show that a similar attack will succeed against the scheme proposed in [MEYE88]:

$$H_i = E_{H_i}[M_i] \oplus M_i$$

8.5 It is possible to use a hash function to construct a block cipher with a structure similar to DES. Because a hash function is one way and a block cipher must be reversible (to decrypt), how is it possible?

8.6 Now consider the opposite problem: using an encryption algorithm to construct a one-way hash function. Consider using RSA with a known key. Then process a message consisting of a sequence of blocks as follows: Encrypt the first block, XOR the result with the second block and encrypt again, etc. Show that this scheme is not secure by solving the following problem. Given a two-block message  $B1, B2$ , and its hash

$$\text{RSAH}(B1, B2) = \text{RSA}(\text{RSA}(B1) \oplus B2)$$

Given an arbitrary block  $C1$ , choose  $C2$  so that  $\text{RSAH}(C1, C2) = \text{RSAH}(B1, B2)$ .

## APPENDIX 8A MATHEMATICAL BASIS OF THE BIRTHDAY ATTACK

In this appendix, we derive the mathematical justification for the birthday attack. We begin with a related problem and then look at the problem from which the name "birthday attack" is derived.

### Related Problem

A general problem relating to hash functions is the following. Given a hash function  $H$ , with  $n$  possible outputs and a specific value  $H(x)$ , if  $H$  is applied to  $k$  random inputs, what must be the value of  $k$  so that the probability that at least one input  $y$  satisfies  $H(y) = H(x)$  is 0.5?

For a single value of  $y$ , the probability that  $H(y) = H(x)$  is just  $1/n$ . Conversely, the probability that  $H(y) \neq H(x)$  is  $[1 - (1/n)]$ . If we generate  $k$  random values of  $y$ , then the probability that none of them match is just the product of the probabilities that each individual value does not match, or  $[1 - (1/n)]^k$ . Thus, the probability that there is at least one match is  $1 - [1 - (1/n)]^k$ .

Now the binomial theorem can be stated as follows:

$$(1-a)^k = 1 - ka + \frac{k(k-1)}{2!} a^2 - \frac{k(k-1)(k-2)}{3!} a^3 \dots$$

For very small values of  $a$ , this can be approximated as  $(1 - ka)$ . Thus, the probability of at least one match is approximated as  $1 - [1 - (1/n)]^k \approx 1 - [1 - (k/n)] = k/n$ . For a probability of 0.5, we have  $k = n/2$ .

In particular, for an  $m$ -bit hash code, the number of possible codes is  $2^m$  and the value of  $k$  that produces a probability of one-half is

$$k = 2^{(m-1)} \quad (8.1)$$



### The Birthday Paradox

The birthday paradox is often presented in elementary probability courses to demonstrate that probability results are sometimes counterintuitive. The problem can be stated as follows: What is the minimum value of  $k$  such that the probability is greater than 0.5 that at least two people in a group of  $k$  people have the same birthday? Ignore February 29 and assume that each birthday is equally likely. To answer, let us define

$P(n, k) = \text{Pr}[\text{at least one duplicate in } k \text{ items, with each item able to take on one of } n \text{ equally likely values between 1 and } n]$

Thus, we are looking for the smallest value of  $k$  such that  $P(365, k) \geq 0.5$ . It is easier first to derive the probability that there are no duplicates, which we designate as  $Q(365, k)$ . If  $k > 365$ , then it is impossible for all values to be different. So we assume  $k \leq 365$ . Now consider the number of different ways,  $N$ , that we can have  $k$  values with no duplicates. We may choose any of the 365 values for the first item, any of the remaining 364 numbers for the second item, and so on. Hence, the number of different ways is

$$N = 365 \times 364 \times \cdots (365 - k + 1) = \frac{365!}{(365 - k)!} \quad (8.2)$$

If we remove the restriction that there are no duplicates, then each item can be any of 365 values, and the total number of possibilities is  $365^k$ . So the probability of no duplicates is simply the fraction of sets of values that have no duplicates out of all possible sets of values:

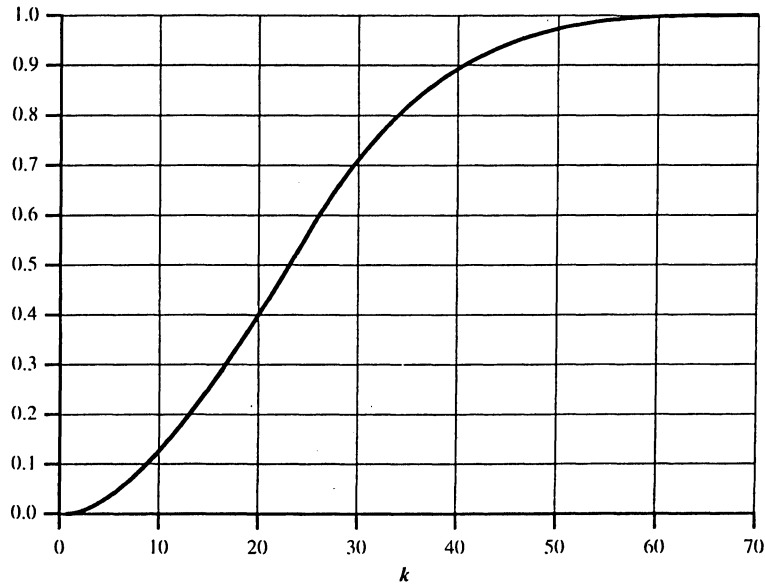
$$Q(365, k) = \frac{365! / (365 - k)!}{(365)^k} = \frac{365!}{(365 - k)! (365)^k}$$

and

$$P(365, k) = 1 - Q(365, k) = 1 - \frac{365!}{(365 - k)! (365)^k} \quad (8.3)$$

This function is plotted in Figure 8.11. The probabilities may seem surprisingly large to anyone who has not considered the problem before. Many people would guess that to have a probability greater than 0.5 that there is at least one duplicate, the number of people in the group would have to be about 100. In fact, the number is 23, with  $P(365, 23) = 0.5073$ . For  $k = 100$ , the probability of at least one duplicate is 0.9999997.

Perhaps the reason that the result seems so surprising is that if you consider a particular person in a group, the probability that some other person in the group has the same birthday is small. But the probability that we are concerned with is the probability that *any* pair of people in the group has the same birthday. In a group of 23, there are  $\frac{23(23 - 1)}{2} = 253$  different pairs of people—hence the high probabilities.



**Figure 8.11** The Birthday Paradox.

### Useful Inequality

Before developing a generalization of the birthday problem, we derive an inequality that will be needed:

$$(1-x) \leq e^{-x} \quad \text{for all } x \geq 0 \quad (8.4)$$

Figure 8.12 illustrates the inequality. To see that the inequality holds, note that the lower line is the tangent to  $e^{-x}$  at  $x = 0$ . The slope of that line is just the derivative of  $e^{-x}$  at  $x = 0$ :

$$f(x) = e^{-x}$$

$$f'(x) = \frac{d}{dx} e^{-x} = -e^{-x}$$

$$f'(0) = -1$$

The tangent is a straight line of the form  $ax + b$ , with  $a = -1$ , and the tangent at  $x = 0$  must equal  $e^{-0} = 1$ . Thus, the tangent is the function  $(1 - x)$ , confirming the inequality of Equation (8.4). Further, note that for small  $x$ , we have  $(1 - x) \approx e^{-x}$ .

### The General Case of Duplications

The birthday problem can be generalized to the following problem: Given a random variable that is an integer with uniform distribution between 1 and  $n$  and a selection of  $k$  instances ( $k \leq n$ ) of the random variable, what is the probability,  $P(n, k)$ , that

there is at least one duplicate? The birthday problem is just the special case with  $n = 365$ . By the same reasoning as before, we have the following generalization of Equation (8.3):

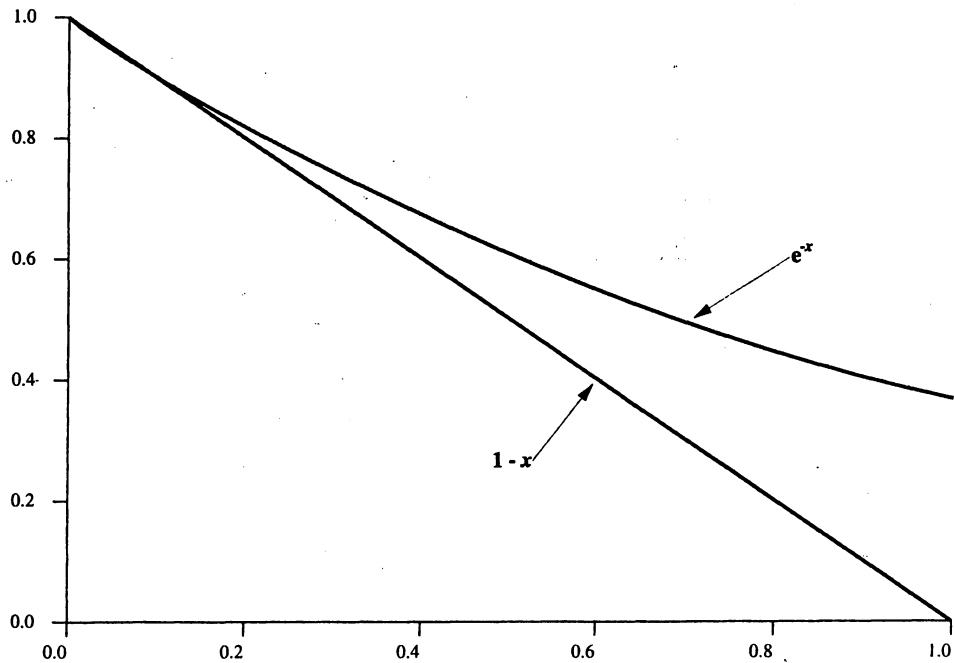
$$P(n, k) = 1 - \frac{n!}{(n-k)!n^k} \quad (8.5)$$

We can rewrite as

$$\begin{aligned} P(n, k) &= 1 - \frac{n \times (n-1) \times \cdots \times (n-k+1)}{n^k} \\ &= 1 - \left[ \frac{n-1}{n} \times \frac{n-2}{n} \times \cdots \times \frac{n-k+1}{n} \right] \\ &= 1 - \left[ \left(1 - \frac{1}{n}\right) \times \left(1 - \frac{2}{n}\right) \times \cdots \times \left(1 - \frac{k-1}{n}\right) \right] \end{aligned}$$

Using the inequality of Equation (8.4),

$$\begin{aligned} P(n, k) &> 1 - \left[ \left(e^{-1/n}\right) \times \left(e^{-2/n}\right) \times \cdots \times \left(e^{-(k-1)/n}\right) \right] \\ &> 1 - e^{-[(1/n) + (2/n) + \cdots + ((k-1)/n)]} \\ &> 1 - e^{-k \times (k-1) / 2n} \end{aligned}$$



**Figure 8.12** A Useful Inequality.

Now let us pose the question, What value of  $k$  is required such that  $P(n, k) > 0.5$ ? To satisfy the requirement, we have

$$\begin{aligned} 1/2 &= 1 - e^{-(k \times (k-1))/2n} \\ 2 &= e^{(k \times (k-1))/2n} \\ \ln(2) &= \frac{k \times (k-1)}{2n} \end{aligned}$$

For large  $k$ , we can replace  $k \times (k-1)$  by  $k^2$ , and we get

$$k = \sqrt{2(\ln 2)n} = 1.18\sqrt{n} \approx \sqrt{n} \quad (8.6)$$

As a reality check, for  $n = 365$ , we get  $k = 1.18 \times \sqrt{365} = 22.54$ , which is very close to the correct answer of 23.

We can now state the basis of the birthday attack in the following terms. Suppose we have a function  $H$ , with  $2^m$  possible outputs (i.e., an  $m$ -bit output). If  $H$  is applied to  $k$  random inputs, what must be the value of  $k$  so that there is the probability of at least one duplicate [i.e.,  $H(x) = H(y)$  for some inputs  $x, y$ ]? Using the approximation in Equation (8.6),

$$k = \sqrt{2^m} = 2^{m/2} \quad (8.7)$$

### Overlap between Two Sets

There is a problem related to the general case of duplications that is also of relevance for our discussions. The problem is this: Given a random variable that is an integer with uniform distribution between 1 and  $n$  and two sets of  $k$  instances ( $k \leq n$ ) of the random variable, what is the probability,  $R(n, k)$ , that the two sets are not disjoint; that is, what is the probability that there is at least one value found in both sets?

Let us call the two sets  $X$  and  $Y$ , with elements  $\{x_1, x_2, \dots, x_k\}$  and  $\{y_1, y_2, \dots, y_k\}$ , respectively. Given the value of  $x_1$ , the probability that  $y_1 = x_1$  is just  $1/n$ , and therefore the probability that  $y_1$  does not match  $x_1$  is  $[1 - (1/n)]$ . If we generate the  $k$  random values in  $Y$ , the probability that none of these values is equal to  $x_1$  is  $[1 - (1/n)]^k$ . Thus, the probability that there is at least one match to  $x_1$  is  $1 - [1 - (1/n)]^k$ .

To proceed, let us make the assumption that all the elements of  $X$  are distinct. If  $n$  is large and if  $k$  is also large (e.g., on the order of  $\sqrt{n}$ ), then this is a good approximation. In fact, there may be a few duplications, but most of the values will be distinct. With that assumption, we can make the following derivation:

$$\begin{aligned} \text{Pr[no match in } Y \text{ to } x_1] &= \left(1 - \frac{1}{n}\right)^k \\ \text{Pr[no match in } Y \text{ to } X] &= \left(\left(1 - \frac{1}{n}\right)^k\right)^k = \left(1 - \frac{1}{n}\right)^{k^2} \\ R(n, k) &= \text{Pr[at least one match in } Y \text{ to } X] = 1 - \left(1 - \frac{1}{n}\right)^{k^2} \end{aligned}$$

Using the inequality of Equation (8.4),

$$R(n, k) > 1 - (e^{-1/n})^{k^2}$$

$$R(n, k) > 1 - (e^{-k^2/n})$$

Let us pose the question, What value of  $k$  is required such that  $R(n, k) > 0.5$ ? To satisfy the requirement, we have

$$1/2 = 1 - (e^{-k^2/n})$$

$$2 = e^{k^2/n} \tag{8.8}$$

$$\ln(2) = \frac{k^2}{n}$$

$$k = \sqrt{(\ln(2))n} = 0.83\sqrt{n} \approx \sqrt{n}$$

We can state this in terms related to birthday attacks as follows. Suppose we have a function  $H$ , with  $2^m$  possible outputs (i.e., an  $m$ -bit output). Apply  $H$  to  $k$  random inputs to produce the set  $X$  and again to  $k$  additional random inputs to produce the set  $Y$ . What must be the value of  $k$  so that there is the probability of at least one match between the two sets [i.e.,  $H(x) = H(y)$  for some inputs  $x \in X, y \in Y$ ]? Using the approximation in Equation (8.8),

$$k = \sqrt{2^m} = 2^{m/2}$$



# CHAPTER 9

## HASH ALGORITHMS

*Each of the messages, like each one he had ever read of Stern's commands, began with a number and ended with a number or row of numbers. No efforts on the part of Mungo or any of his experts had been able to break Stern's code, nor was there any clue as to what the preliminary number and those ultimate numbers signified.*

—*Talking to Strange Men*, Ruth Rendell

*The Douglas Squirrel has a distinctive eating habit. It usually eats pine cones from the bottom end up. Partially eaten cones can indicate the presence of these squirrels if they have been attacked from the bottom first. If, instead, the cone has been eaten from the top end down, it is more likely to have been a crossbill finch that has been doing the dining.*

—*Squirrels: A Wildlife Handbook*, Kim Long

There are several similarities in the evolution of hash functions and that of symmetric block ciphers. We have seen that the increasing power of brute-force attacks and advances in cryptanalysis have led to the decline in the popularity of DES and in the design of newer algorithms with longer key lengths and with features designed to resist specific cryptanalytic attacks. Similarly, advances in computing power and hash function cryptanalysis have led to the decline in the popularity of first MD4 and then MD5, two very popular hash functions. In response, newer hash algorithms have been developed with longer hash code length and with features designed to resist specific cryptanalytic attacks. Another point of similarity is the reluctance to depart from a proven structure. DES is based on the Feistel cipher, which, in turn, is based on the substitution-permutation network proposal of Shannon. Virtually all important subsequent block ciphers follow the Feistel design because the design can be adapted to resist newly

discovered cryptanalytic threats. If, instead, an entirely new design were used for a symmetric block cipher, there would be concern that the structure itself opened up new avenues of attack not yet thought of. Similarly, most important modern hash functions follow the basic structure of Figure 8.10. Again, this has proved to be a fundamentally sound structure, and newer designs simply refine the structure and add to the hash code length.

In this chapter, we look at three important hash functions: MD5, SHA-1, and RIPEMD-160. We then look at an Internet-standard message authentication code, HMAC, that is based on the use of a hash function.

## 9.1 MD5 MESSAGE DIGEST ALGORITHM

The MD5 message-digest algorithm (RFC 1321) was developed by Ron Rivest at MIT (the “R” in the RSA [Rivest-Shamir-Adleman] public-key encryption algorithm). Until the last few years, when both brute-force and cryptanalytic concerns have arisen, MD5 was the most widely used secure hash algorithm.

### MD5 Logic

The algorithm takes as input a message of arbitrary length and produces as output a 128-bit message digest. The input is processed in 512-bit blocks.

Figure 9.1 depicts the overall processing of a message to produce a digest. This follows the general structure depicted in Figure 8.10. The processing consists of the following steps:

- **Step 1: Append padding bits.** The message is padded so that its length in bits is congruent to 448 modulo 512 ( $\text{length} \equiv 448 \pmod{512}$ ). That is, the length of the padded message is 64 bits less than an integer multiple of 512 bits. Padding is always added, even if the message is already of the desired length. For example, if the message is 448 bits long, it is padded by 512 bits to a length of 960 bits. Thus, the number of padding bits is in the range of 1 to 512.

The padding consists of a single 1-bit followed by the necessary number of 0-bits.

- **Step 2: Append length.** A 64-bit representation of the length in bits of the original message (before the padding) is appended to the result of step 1 (least significant byte first). If the original length is greater than  $2^{64}$ , then only the low-order 64 bits of the length are used. Thus, the field contains the length of the original message, modulo  $2^{64}$ .

The outcome of the first two steps yields a message that is an integer multiple of 512 bits in length. In Figure 9.1, the expanded message is represented as the sequence of 512-bit blocks  $Y_0, Y_1, \dots, Y_{L-1}$ , so that the total length of the expanded message is  $L \times 512$  bits. Equivalently, the result is a multiple of 16 32-bit words. Let  $M[0 \dots N - 1]$  denote the words of the resulting message, with  $N$  an integer multiple of 16. Thus,  $N = L \times 16$ .

- **Step 3: Initialize MD buffer.** A 128-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as four 32-bit



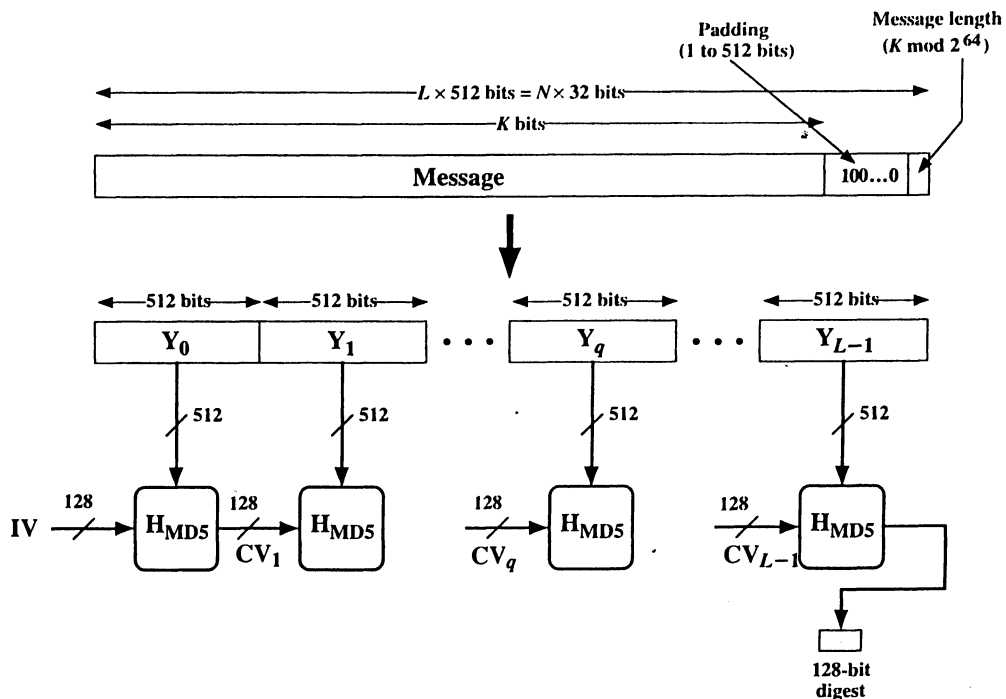


Figure 9.1 Message Digest Generation Using MD5.

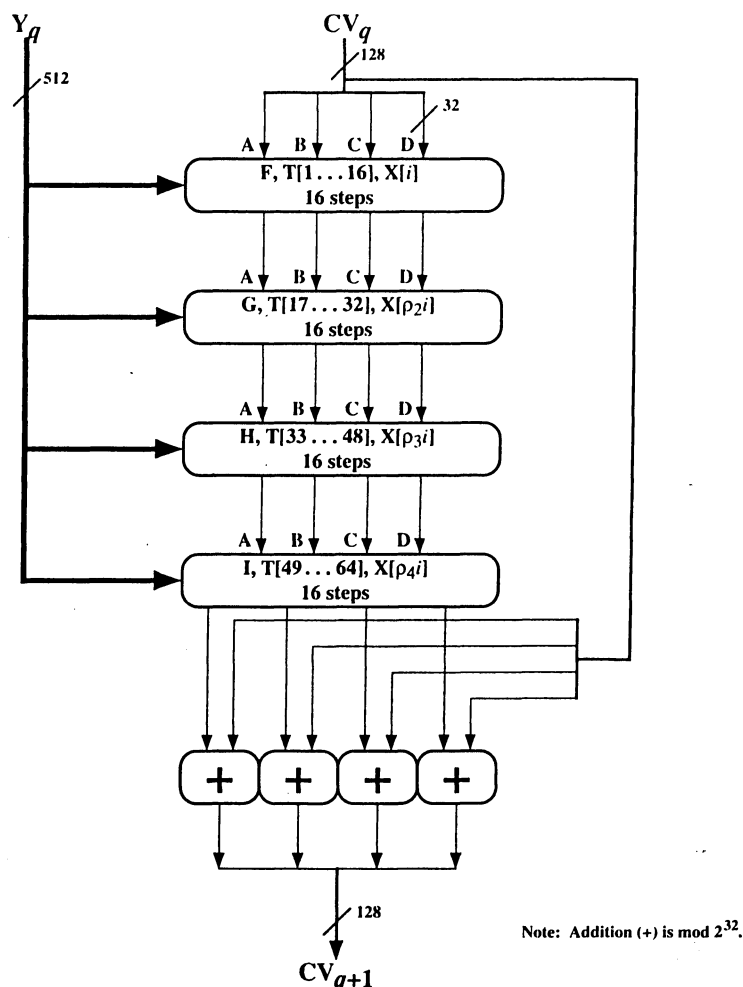
registers (A, B, C, D). These registers are initialized to the following 32-bit integers (hexadecimal values):

A = 67452301  
 B = EFCDAB89  
 C = 98BADCFE  
 D = 10325476

These values are stored in little-endian format, which is the least significant byte of a word in the low-address byte position. As 32-bit strings, the initialization-values (in hexadecimal) appear as follows:

word A: 01 23 45 67  
 word B: 89 AB CD EF  
 word C: FE DC BA 98  
 word D: 76 54 32 10

- **Step 4: Process message in 512-bit (16-word) blocks.** The heart of the algorithm is a compression function that consists of four “rounds” of processing; this module is labeled  $H_{MD5}$  in Figure 9.1, and its logic is illustrated in Figure 9.2. The four rounds have a similar structure, but each uses a different primitive logical function, referred to as F, G, H, and I in the specification.



**Figure 9.2** MD5 Processing of a Single 512-bit Block (MD5 compression function).

Each round takes as input the current 512-bit block being processed ( $Y_q$ ) and the 128-bit buffer value  $ABCD$  and updates the contents of the buffer. Each round also makes use of one-fourth of a 64-element table  $T[1 \dots 64]$ , constructed from the sine function. The  $i$ th element of  $T$ , denoted  $T[i]$ , has the value equal to the integer part of  $2^{32} \times \text{abs}(\sin(i))$ , where  $i$  is in radians. Because  $\text{abs}(\sin(i))$  is a number between 0 and 1, each element of  $T$  is an integer that can be represented in 32 bits. The table provides a “randomized” set of 32-bit patterns, which should eliminate any regularities in the input data. Table 9.1b lists the values of  $T$ .

The output of the fourth round is added to the input to the first round ( $CV_q$ ) to produce  $CV_{q+1}$ . The addition is done independently for each of the four words in the buffer with each of the corresponding words in  $CV_q$ , using addition modulo  $2^{32}$ .

275

**Table 9.1** Key Elements of MD5

(a) Truth table of logical functions

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

(b) Table T, constructed from the sine function

T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B756	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 265E5A51	T[35] = 699D6122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57C0FAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECF A9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEFF47D
T[8] = FD469501	T[24] = E7D3FBC8	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7BE	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F7537E82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438E	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

- **Step 5: Output.** After all  $L$  512-bit blocks have been processed, the output from the  $L$ th stage is the 128-bit message digest.

We can summarize the behavior of MD5 as follows:

$$\begin{aligned} CV_0 &= IV \\ CV_{q+1} &= \text{SUM}_{32}(CV_q, \text{RF}_I[Y_q, \text{RF}_H[Y_q, \text{RF}_G[Y_q, \text{RF}_F[Y_q, CV_q]]]]) \\ MD &= CV_L \end{aligned}$$

where

$IV$  = initial value of the ABCD buffer, defined in step 3  
 $Y_q$  = the  $q$ th 512-bit block of the message  
 $L$  = the number of blocks in the message (including padding and length fields)  
 $CV_q$  = chaining variable processed with the  $q$ th block of the message  
 $\text{RF}_x$  = round function using primitive logical function  $x$   
 $MD$  = final message digest value  
 $\text{SUM}_{32}$  = Addition modulo  $2^{32}$  performed separately on each word of the pair of inputs

### MD5 Compression Function

Let us look in more detail at the logic in each of the four rounds of the processing of one 512-bit block. Each round consists of a sequence of 16 steps operating on the buffer ABCD. Each step is of the form

$$a \leftarrow b + ((a + g(b, c, d) + X[k] + T[i]) \lll s)$$

where

$a, b, c, d$  = the four words of the buffer, in a specified order that varies across steps  
 $g$  = one of the primitive functions F, G, H, I  
 $\lll s$  = circular left shift (rotation) of the 32-bit argument by  $s$  bits  
 $X[k]$  =  $M[q \times 16 + k]$  = the  $k$ th 32-bit word in the  $q$ th 512-bit block of the message  
 $T[i]$  = the  $i$ th 32-bit word in matrix T  
 $+$  = addition modulo  $2^{32}$

Figure 9.3 illustrates the step operation. The order in which the four words ( $a, b, c, d$ ) are used produces a word-level circular right shift of one word for each step.

One of the four primitive logical functions is used for each of the four rounds of the algorithm. Each primitive function takes three 32-bit words as input and produces a 32-bit word output. Each function performs a set of bitwise logical operations; that is, the  $n$ th bit of the output is a function of the  $n$ th bit of the three inputs. The functions can be summarized as follows:

Round	Primitive function $g$	$g(b, c, d)$
1	$F(b, c, d)$	$(b \wedge c) \vee (b \wedge d)$
2	$G(b, c, d)$	$(b \wedge d) \vee (c \wedge d)$
3	$H(b, c, d)$	$b \oplus c \oplus d$
4	$I(b, c, d)$	$c \oplus (b \vee d)$

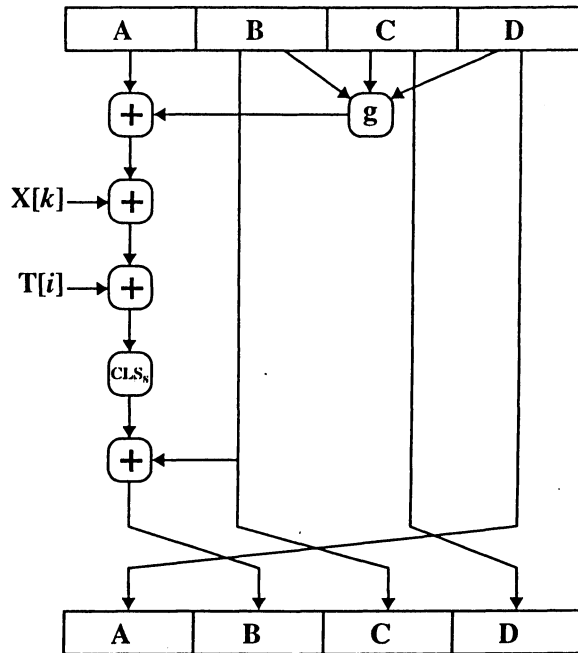


Figure 9.3 Elementary MD5 Operation (single step).

The logical operators (AND, OR, NOT, XOR) are represented by the symbols ( $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\oplus$ ). Function  $F$  is a conditional function: If  $b$  then  $c$  else  $d$ . Similarly,  $G$  can be stated as follows: If  $d$  then  $b$  else  $c$ . Function  $H$  produces a parity bit. Table 9.1a is a truth table of the four functions.

Figure 9.4, adapted from RFC 1321, defines the processing algorithm of step 4. The array of 32-bit words  $X[0..15]$  holds the value of the current 512-bit input block being processed. Within a round, each of the 16 words of  $X[i]$  is used exactly once, during one step; the order in which these words is used varies from round to round. In the first round, the words are used in their original order. The following permutations are defined for rounds 2 through 4:

$$\rho_2(i) = (1 + 5i) \bmod 16$$

$$\rho_3(i) = (5 + 3i) \bmod 16$$

$$\rho_4(i) = 7i \bmod 16$$

Each of the 64 32-bit word elements of  $T$  is used exactly once, during one step of one round. Also, note that for each step, only one of the 4 bytes of the ABCD buffer is updated. Hence, each byte of the buffer is updated four times during the round and then a fifth time at the end to produce the final output for this block. Finally, note that four different circular left shift amounts are used each round and are different from round to round. The point of all this complexity is to make it very difficult to generate collisions (two 512-bit blocks that produce the same output).

78

```

/* Process each 16-word (512-bit) block. */
For q = 0 to (N/16) - 1 do
  /* Copy block q into X. */
  For j = 0 to 15 do
    Set X[j] to M[q*16 + j].
  end /* of loop on j */

  /* Save A as AA, B as BB, C as CC, and
  D as DD. */
  AA = A
  BB = B
  CC = C
  DD = D

  /* Round 1. */
  /* Let [abcd k s i] denote the operation
  a = b + ((a + F(b,c,d) + X[k] + T[i]) <<<s).
  Do the following 16 operations. */
  [ABCD 0 7 1]
  [DABC 1 12 2]
  [CDAB 2 17 3]
  [BCDA 3 22 4]
  [ABCD 4 7 5]
  [DABC 5 12 6]
  [CDAB 6 17 7]
  [BCDA 7 22 8]
  [ABCD 8 7 9]
  [DABC 9 12 10]
  [CDAB 10 17 11]
  [BCDA 11 22 12]
  [ABCD 12 7 13]
  [DABC 13 12 14]
  [CDAB 14 17 15]
  [BCDA 15 22 16]

  /* Round 2. */
  /* Let [abcd k s i] denote the operation
  a = b + ((a + G(b,c,d) + X[k] + T[i]) <<<s).
  Do the following 16 operations. */
  [ABCD 1 5 17]
  [DABC 6 9 18]
  [CDAB 11 14 19]
  [BCDA 0 20 20]
  [ABCD 5 5 21]
  [DABC 10 9 22]
  [CDAB 15 14 23]
  [BCDA 4 20 24]
  [ABCD 9 5 25]
  [DABC 14 9 26]
  [CDAB 3 14 27]
  [BCDA 8 20 28]
  [ABCD 13 5 29]
  [DABC 2 9 30]
  [CDAB 7 14 31]
  [BCDA 12 20 32]

  /* Round 3. */
  /* Let [abcd k s i] denote the operation
  a = b + ((a + H(b,c,d) + X[k] + T[i]) <<<s).
  Do the following 16 operations. */
  [ABCD 5 4 33]
  [DABC 8 11 34]
  [CDAB 11 16 35]
  [BCDA 14 23 36]
  [ABCD 1 4 37]
  [DABC 4 11 38]
  [CDAB 7 16 39]
  [BCDA 10 23 40]
  [ABCD 13 4 41]
  [DABC 0 11 42]
  [CDAB 3 16 43]
  [BCDA 6 23 44]
  [ABCD 9 4 45]
  [DABC 12 11 46]
  [CDAB 15 16 47]
  [BCDA 2 23 48]

  /* Round 4. */
  /* Let [abcd k s i] denote the operation
  a = b + ((a + I(b,c,d) + X[k] + T[i]) <<<s).
  Do the following 16 operations. */
  [ABCD 0 6 49]
  [DABC 7 10 50]
  [CDAB 14 15 51]
  [BCDA 5 21 52]
  [ABCD 12 6 53]
  [DABC 3 10 54]
  [CDAB 10 15 55]
  [BCDA 1 21 56]
  [ABCD 8 6 57]
  [DABC 15 10 58]
  [CDAB 6 15 59]
  [BCDA 13 21 60]
  [ABCD 4 6 61]
  [DABC 11 10 62]
  [CDAB 2 15 63]
  [BCDA 9 21 64]

  /* Then increment each of the four registers by the
  value it had before this block was started. */
  A = A + AA
  B = B + BB
  C = C + CC
  D = D + DD
end /* of loop on q */

```

Figure 9.4 Basic MD5 Update Algorithm (RFC 1321).

## MD4

MD4 is a precursor to MD5 developed by the same designer, Ron Rivest. It was originally published as an RFC in October 1990. A slightly revised version was published as RFC 1320 in April 1992, the same date as MD5. It is worth briefly discussing MD4 because MD5 shares the design goals of MD4, which were documented in a paper by Rivest [RIVE90]. The following goals were listed:

- **Security:** There is the usual requirement for a hash code—namely, that it be computationally infeasible to find two messages that have the same message digest. At the time of its publication, MD4 was secure against brute-force attacks because of the length of the digest. Rivest also felt it to be secure against cryptanalytic attacks based on the state of the art and on the complexity of MD4.
- **Speed:** The algorithm should lend itself to implementations in software that executes rapidly. In particular, the algorithm is intended to be fast on 32-bit architectures. Thus, the algorithm is based on a simple set of primitive operations on 32-bit words.
- **Simplicity and compactness:** The algorithm should be simple to describe and simple to program, without requiring large programs or substitution tables. These characteristics not only have obvious programming advantages but are desirable from a security point of view, because a simple algorithm is more likely to receive the necessary critical review.
- **Favor little-endian architecture:** Some processor architectures (such as the Intel 80xxx and Pentium line) store the least significant byte of a word in the low-address byte position (little endian). Others (such as a SUN Sparcstation) store the most significant byte of a word in the low-address byte position (big endian). This distinction is significant when treating a message as a sequence of 32-bit words, because one of the two architectures will have to byte reverse each word for processing. Rivest chose to use a little-endian scheme for interpreting a message as a sequence of 32-bit words. This choice was made on the basis of Rivest's observation that big-endian processors are generally faster and can therefore better afford the processing penalty.

These design goals carried over to MD5. MD5 is somewhat more complex and hence somewhat slower to execute than MD4. Rivest felt that the added complexity was justified by the increased level of security afforded. The following are the main differences between the two:

1. MD4 uses three rounds of 16 steps each, whereas MD5 uses four rounds of 16 steps each.
2. In MD4, no additive constant is used in the first round. The same additive constant is used for each of the steps of the second round. Another additive constant is used for each of the steps of the third round. In MD5, a different additive constant,  $T[i]$ , is used for each of the 64 steps.
3. MD5 uses four primitive logical functions, one for each round, compared to three for MD4, again one for each round.

4. In MD5, each step adds in the result of the preceding step. For example, the step 1 result updates word A. The step 2 result, which is stored in D, is formed by adding A to the circular left shift result. Similarly, the step 3 result is stored in C and is formed by adding D to the circular left shift result. MD4 did not include this final addition. Rivest feels that the inclusion of the previous step's result promotes a greater avalanche effect.

### Strength of MD5

The MD5 algorithm has the property that every bit of the hash code is a function of every bit in the input. The complex repetition of the basic functions (F, G, H, I) produces results that are well mixed; that is, it is unlikely that two messages chosen at random, even if they exhibit similar regularities, will have the same hash code. Rivest conjectures in the RFC that MD5 is as strong as possible for a 128-bit hash code; namely, the difficulty of coming up with two messages having the same message digest is on the order of  $2^{64}$  operations, whereas the difficulty of finding a message with a given digest is on the order of  $2^{128}$  operations.

As of this writing, no analysis has been done to disprove these conjectures. However, there has been an ominous trend in the attacks on MD5:

1. Berson [BERS92] showed, using differential cryptanalysis, that it is possible in reasonable time to find two messages that produce the same digest for a single-round MD5. The result was demonstrated for each of the four rounds. However, the author has not been able to show how to generalize the attack to the full four-round MD5.
2. Boer and Bosselaers [BOER93] showed how to find a message block X and two related chaining variables that yield the same output state. That is, execution of MD5 on a single block of 512 bits will yield the same output for two different input values in buffer ABCD. This is referred to as a *pseudocollision*. At present, there does not seem to be any way to extend this approach to a successful attack on MD5.
3. The most serious attack on MD5 has been developed by Dobbertin [DOBB96a]. His technique enables the generation of a collision for the MD5 compression function; that is, the attack works on the operation of MD5 on a single 512-bit block of input by finding another block that produces the same 128-bit output. As of this writing, no way has been found to generalize this attack to a full message using the MD5 initial value (IV). Nevertheless, the success of this attack is too close for comfort.

Thus we see that from a cryptanalytic point of view, MD5 must now be considered vulnerable. Further, from the point of view of brute-force attack, MD5 is now vulnerable to birthday attacks that require on the order of effort of  $2^{64}$ . As a result, there is a need to replace the popular MD5 with a hash function that has a longer hash code and is more resistant to known methods of cryptanalysis. Two candidates have emerged: SHA-1 and RIPEMD-160. These are examined in the next two sections.



## 9.2 SECURE HASH ALGORITHM

The secure hash algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS PUB 180) in 1993; a revised version was issued as FIPS PUB 180-1 in 1995 and is generally referred to as SHA-1. SHA is based on the MD4 algorithm, and its design closely models MD4.

### SHA-1 Logic

The algorithm takes as input a message with a maximum length of less than  $2^{64}$  bits and produces as output a 160-bit message digest. The input is processed in 512-bit blocks.

The overall processing of a message follows the structure shown for MD5 in Figure 9.1, with a block length of 512 bits and a hash length and chaining variable length of 160 bits. The processing consists of the following steps:

- **Step 1: Append padding bits.** The message is padded so that its length is congruent to 448 modulo 512 ( $\text{length} \equiv 448 \pmod{512}$ ). Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 512. The padding consists of a single 1-bit followed by the necessary number of 0-bits.
- **Step 2: Append length.** A block of 64 bits is appended to the message. This block is treated as an unsigned 64-bit integer (most significant byte first) and contains the length of the original message (before the padding).
- **Step 3: Initialize MD buffer.** A 160-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as five 32-bit registers (A, B, C, D, E). These registers are initialized to the following 32-bit integers (hexadecimal values):

```
A = 67452301
B = EFCDA889
C = 98BADCFE
D = 10325476
E = C3D2E1F0
```

Note that the first four values are the same as those used in MD5. However, in the case of SHA-1, these values are stored in big-endian format, which is the most significant byte of a word in the low-address byte position. As 32-bit strings, the initialization values (in hexadecimal) appear as follows:

```
word A: 67 45 23 01
word B: EF CD AB 89
word C: 98 BA DC FE
word D: 10 32 54 76
word E: C3 D2 E1 F0
```

- **Step 4: Process message in 512-bit (16-word) blocks.** The heart of the algorithm is a module that consists of four rounds of processing of 20 steps each.

The logic is illustrated in Figure 9.5. The four rounds have a similar structure, but each uses a different primitive logical function, which we refer to as  $f_1$ ,  $f_2$ ,  $f_3$ , and  $f_4$ .

Each round takes as input the current 512-bit block being processed ( $Y_q$ ) and the 160-bit buffer value ABCDE and updates the contents of the buffer. Each round also makes use of an additive constant  $K_t$ , where  $0 \leq t \leq 79$  indicates one of the 80 steps across four rounds. In fact, only four distinct constants are used. The values, in hexadecimal and decimal, are as follows:

Step Number	Hexadecimal	Take Integer Part of:
$0 \leq t \leq 19$	$K_t = 5A827999$	$\lfloor 2^{30} \times \sqrt{2} \rfloor$
$20 \leq t \leq 39$	$K_t = 6ED9EBA1$	$\lfloor 2^{30} \times \sqrt{3} \rfloor$
$40 \leq t \leq 59$	$K_t = 8F1BBCDC$	$\lfloor 2^{30} \times \sqrt{5} \rfloor$
$60 \leq t \leq 79$	$K_t = CA62C1D6$	$\lfloor 2^{30} \times \sqrt{10} \rfloor$

The output of the fourth round (eightieth step) is added to the input to the first round ( $CV_q$ ) to produce  $CV_{q+1}$ . The addition is done independently for each of the five words in the buffer with each of the corresponding words in  $CV_q$ , using addition modulo  $2^{32}$ .

- **Step 5: Output.** After all  $L$  512-bit blocks have been processed, the output from the  $L$ th stage is the 160-bit message digest.

We can summarize the behavior of SHA-1 as follows:

$$\begin{aligned} CV_0 &= IV \\ CV_{q+1} &= \text{SUM}_{32}(CV_q, ABCDE_q) \\ MD &= CV_L \end{aligned}$$

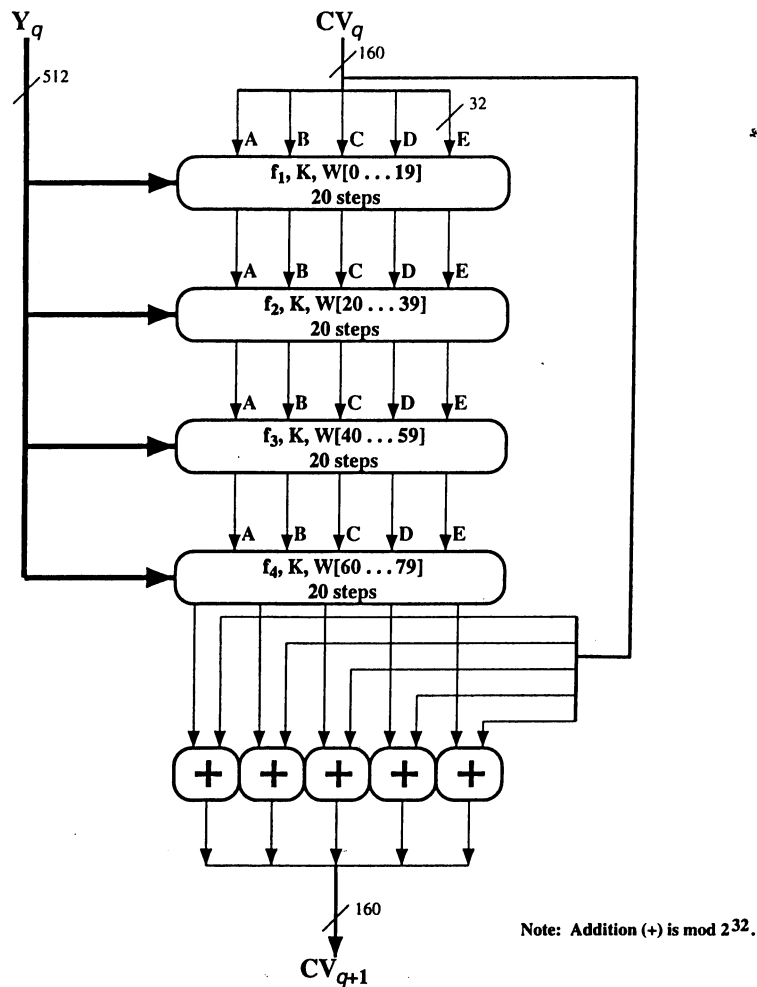
where

- IV = initial value of the ABCDE buffer, defined in step 3
- ABCDE<sub>q</sub> = the output of the last round of processing of the  $q$ th message block
- $L$  = the number of blocks in the message (including padding and length fields)
- SUM<sub>32</sub> = Addition modulo  $2^{32}$  performed separately on each word of the pair of inputs
- MD = final message digest value

### SHA-1 Compression Function

Let us look in more detail at the logic in each of the 80 rounds of the processing of one 512-bit block. Each round is of the form (Figure 9.6)

$$A, B, C, D, E \leftarrow (E + f(t, B, C, D) + S^5(A) + W_t + K_t), A, S^{30}(B), C, D$$



**Figure 9.5** SHA-1 Processing of a Single 512-bit Block (SHA-1 compression function).

where

- $A, B, C, D, E$  = the five words of the buffer
- $t$  = step number;  $0 \leq t \leq 79$
- $f(t, B, C, D)$  = primitive logical function for step  $t$
- $S^k$  = circular left shift (rotation) of the 32-bit argument by  $k$  bits
- $W_t$  = a 32-bit word derived from the current 512-bit input block
- $K_t$  = an additive constant; four distinct values are used, as defined previously
- $+$  = addition modulo  $2^{32}$

Each primitive function takes three 32-bit words as input and produces a 32-bit word output. Each function performs a set of bitwise logical operations; that is,

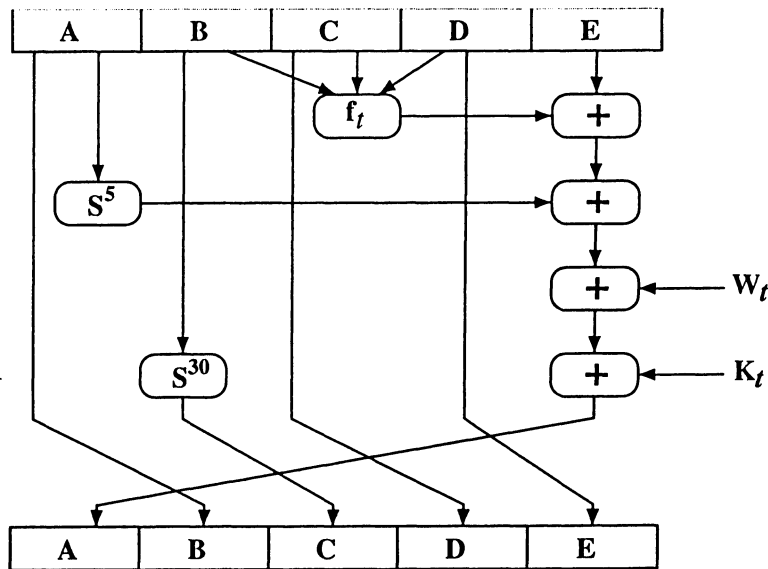


Figure 9.6 Elementary SHA Operation (single step).

the  $n$ th bit of the output is a function of the  $n$ th bit of the three inputs. The functions can be summarized as follows:

Step	Function Name	Function Value
$(0 \leq t \leq 19)$	$f_1 = f(t, B, C, D)$	$(B \wedge C) \vee (B \wedge D)$
$(20 \leq t \leq 39)$	$f_2 = f(t, B, C, D)$	$B \oplus C \oplus D$
$(40 \leq t \leq 59)$	$f_3 = f(t, B, C, D)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$(60 \leq t \leq 79)$	$f_4 = f(t, B, C, D)$	$B \oplus C \oplus D$

The logical operators (AND, OR, NOT, XOR) are represented by the symbols ( $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\oplus$ ). As can be seen, only three different functions are used. For  $0 \leq t \leq 19$ , the function is the conditional function: If B then C else D. For  $20 \leq t \leq 39$  and  $60 \leq t \leq 79$ , the function produces a parity bit. For  $40 \leq t \leq 59$ , the function is true if two or three of the arguments are true. Table 9.2 is a truth table of these functions.

It remains to indicate how the 32-bit word values  $W_t$  are derived from the 512-bit message. Figure 9.7 illustrates the mapping. The first 16 values of  $W_t$  are taken directly from the 16 words of the current block. The remaining values are defined as follows:

$$W_t = S^1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3})$$

Thus, in the first 16 steps of processing, the value of  $W_t$  is equal to the corresponding word in the message block. For the remaining 64 steps, the value of  $W_t$  consists of the circular left shift by one bit of the XOR of four of the preceding val-

**Table 9.2** Truth Table of Logical Functions for SHA-1

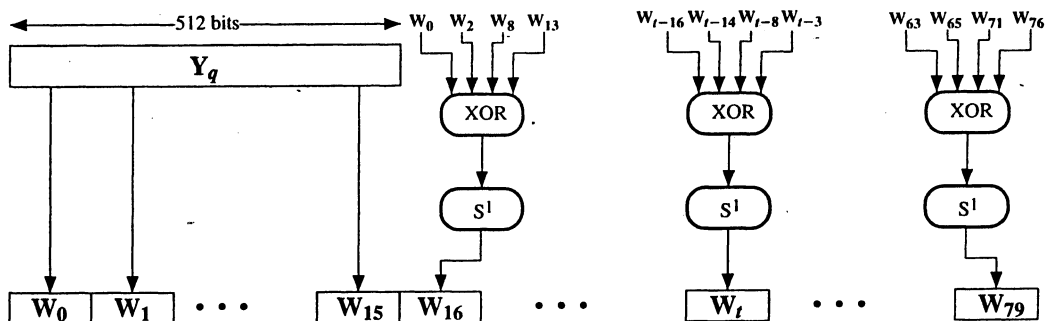
B	C	D	$f_{0..19}$	$f_{20..39}$	$f_{40..59}$	$f_{60..79}$
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

ues of  $W_t$ . This is a notable difference from MD5 and RIPEMD-160, both of which use one of the 16 words of a message block directly as input to each step function; only the order of the words is permuted from round to round. SHA-1 expands the 16 block words to 80 words for use in the compression function. This introduces a great deal of redundancy and interdependence to the message blocks that are compressed, which complicates the task of finding a different message block that maps to the same compression function output.

### Comparison of SHA-1 and MD5

Because both are derived from MD4, SHA-1 and MD5 are quite similar to one another. Accordingly, their strengths and other characteristics should be similar. We compare the two algorithms using the design goals cited earlier for MD4.

- **Security against brute-force attacks:** The most obvious and most important difference is that the SHA-1 digest is 32 bits longer than the MD5 digest. Using a brute-force technique, the difficulty of producing any message having a given message digest is on the order of  $2^{128}$  operations for MD5 and  $2^{160}$  for SHA-1. Again, using a brute-force technique, the difficulty of producing two messages having the same message digest is on the order of  $2^{64}$  operations for MD5 and  $2^{80}$  for SHA-1. Thus, SHA-1 is considerably stronger against brute-force attacks.

**Figure 9.7** Creation of 80-word Input Sequence for SHA-1 Processing of Single Block.

- **Security against cryptanalysis:** As was discussed in the previous section, MD5 is vulnerable to cryptanalytic attacks discovered since its design. SHA-1 appears not to be vulnerable to such attacks. However, little is publicly known about the design criteria for SHA-1, so its strength is more difficult to judge than would otherwise be the case.
- **Speed:** Because both algorithms rely heavily on addition modulo  $2^{32}$ , both do well on a 32-bit architecture. SHA-1 involves more steps (80 versus 64) and must process a 160-bit buffer compared to MD5's 128-bit buffer. Thus, SHA-1 should execute more slowly than MD5 on the same hardware.
- **Simplicity and compactness:** Both algorithms are simple to describe and simple to implement and do not require large programs or substitution tables.
- **Little-endian versus big-endian architecture:** MD5 uses a little-endian scheme for interpreting a message as a sequence of 32-bit words, whereas SHA-1 uses a big-endian scheme.<sup>1</sup> There appears to be no strong advantage to either approach.

### 9.3 RIPEMD-160

The RIPEMD-160 message-digest algorithm [DOBB96b, BOSS97] was developed under the European RACE Integrity Primitives Evaluation (RIPE) project, by a group of researchers that launched partially successful attacks on MD4 and MD5. The group originally developed a 128-bit version of RIPEM. After the end of the RIPE project, H. Dobbertin (who was not a part of the RIPE project) found attacks on two rounds of RIPEMD, and later on MD4 and MD5. Because of these attacks, some members of the RIPE consortium decided to upgrade RIPEMD. The design work was done by them and by Dobbertin.

#### RIPEMD-160 Logic

The algorithm takes as input a message of arbitrary length and produces as output a 160-bit message digest. The input is processed in 512-bit blocks.

The overall processing of a message follows the structure shown for MD5 in Figure 9.1, with a block length of 512 bits and a hash length and chaining variable length of 160 bits. The processing consists of the following steps:

- **Step 1: Append padding bits.** The message is padded so that its length is congruent to 448 modulo 512 ( $\text{length} \equiv 448 \pmod{512}$ ). Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 512. The padding consists of a single 1-bit followed by the necessary number of 0-bits.
- **Step 2: Append length.** A block of 64 bits is appended to the message. This block is treated as an unsigned 64-bit integer (least significant byte first) and contains the length of the original message (before the padding). Note that as with MD5, and in contrast to SHA-1, RIPEMD-160 uses a little-endian convention.

<sup>1</sup>This is probably because the NSA designers of SHA used a Sun for the prototype implementation.

- **Step 3: Initialize MD buffer.** A 160-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as five 32-bit registers (A, B, C, D, E). These registers are initialized to the following hexadecimal values:

A = 67452301  
 B = EFCDAB89  
 C = 98BADCFE  
 D = 10325476  
 E = C3D2E1F0

These are the same values used in SHA-1, and the first four are the same values used in MD5. As in MD5, these values are stored in little-endian format.

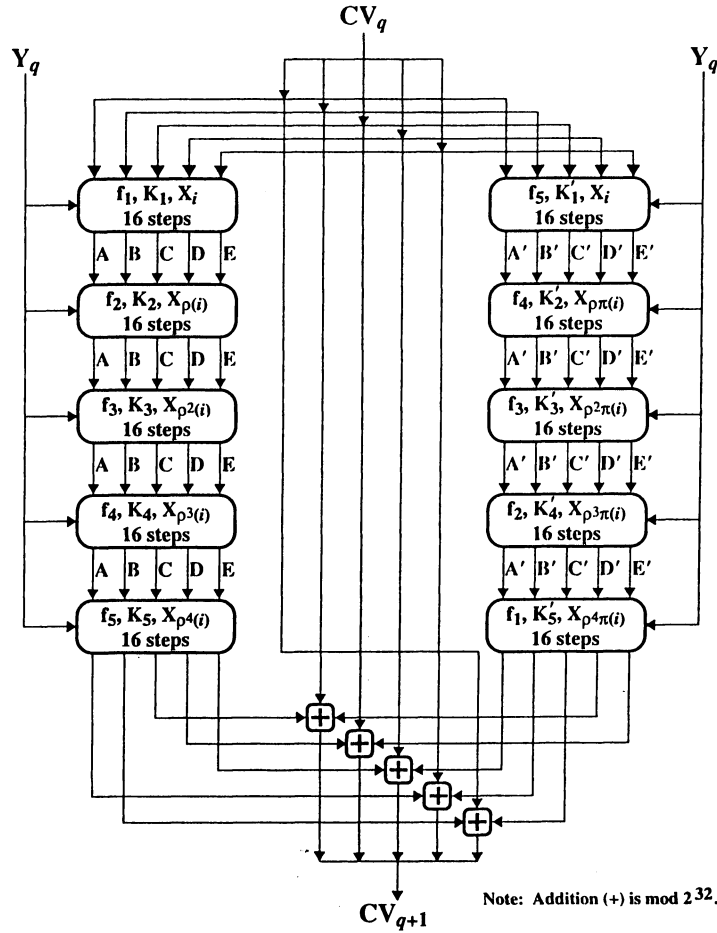
- **Step 4: Process message in 512-bit (16-word) blocks.** The heart of the algorithm is a module that consists of 10 rounds of processing of 16 steps each. The 10 rounds are arranged as two parallel lines of five rounds. The logic is illustrated in Figure 9.8. The 10 rounds have a similar structure, but each uses a different primitive logical function, which we refer to as f1, f2, f3, f4, and f5. Note that the functions are used in reverse order in the right line.

Each round takes as input the current 512-bit block being processed ( $Y_q$ ) and the 160-bit buffer value ABCDE (left line) or A'B'C'D'E' (right line) and updates the contents of the buffer. Each round also makes use of an additive constant. In fact, only nine distinct constants are used, one of which is zero. The values, in hexadecimal and decimal, are shown in Table 9.3.

The output of the fifth round (eightieth step) is added to the chaining variable input to the first round ( $CV_q$ ) to produce  $CV_{q+1}$ . The addition is done independently for each of the five words in the buffer of each line with each of the words in  $CV_q$ , using addition modulo  $2^{32}$ . The addition involves a rotation of the words of each of the three inputs in the following fashion:

**Table 9.3** RIPEMD-160 Constants

Step Number	Left Half		Right Half	
	Hexadecimal	Integer part of	Hexadecimal	Integer part of
$0 \leq j \leq 15$	$K_1 = K(j) =$ 00000000	0	$K'_1 = K'(j) =$ 50A28BE6	$2^{30} \times \sqrt[3]{2}$
$16 \leq j \leq 31$	$K_2 = K(j) =$ 5A827999	$2^{30} \times \sqrt{2}$	$K'_2 = K'(j) =$ 5C4DD124	$2^{30} \times \sqrt[3]{3}$
$32 \leq j \leq 47$	$K_3 = K(j) =$ 6ED9EBA1	$2^{30} \times \sqrt{3}$	$K'_3 = K'(j) =$ 6D703EF3	$2^{30} \times \sqrt[3]{5}$
$48 \leq j \leq 63$	$K_4 = K(j) =$ 8F1BBCDC	$2^{30} \times \sqrt{5}$	$K'_4 = K'(j) =$ 7A6D76E9	$2^{30} \times \sqrt[3]{7}$
$64 \leq j \leq 79$	$K_5 = K(j) =$ A953FD4E	$2^{30} \times \sqrt{7}$	$K'_5 = K'(j) =$ 00000000	0



**Figure 9.8** RIPEMD-160 Processing of a Single 512-bit Block (RIPEMD-160 compression function).

$$CV_{q+1}(0) = CV_q(1) + C + D'$$

$$CV_{q+1}(1) = CV_q(2) + D + E'$$

$$CV_{q+1}(2) = CV_q(3) + E + A'$$

$$CV_{q+1}(3) = CV_q(4) + A + B'$$

$$CV_{q+1}(4) = CV_q(0) + B + C'$$

- **Step 5: Output.** After all  $L$  512-bit blocks have been processed, the output from the  $L$ th stage is the 160-bit message digest.

### RIPEMD-160 Compression Function

Let us look in more detail at the logic in each of the 10 rounds of the processing of one 512-bit block. Each round consists of a sequence of 16 steps. Figure 9.9 illus-



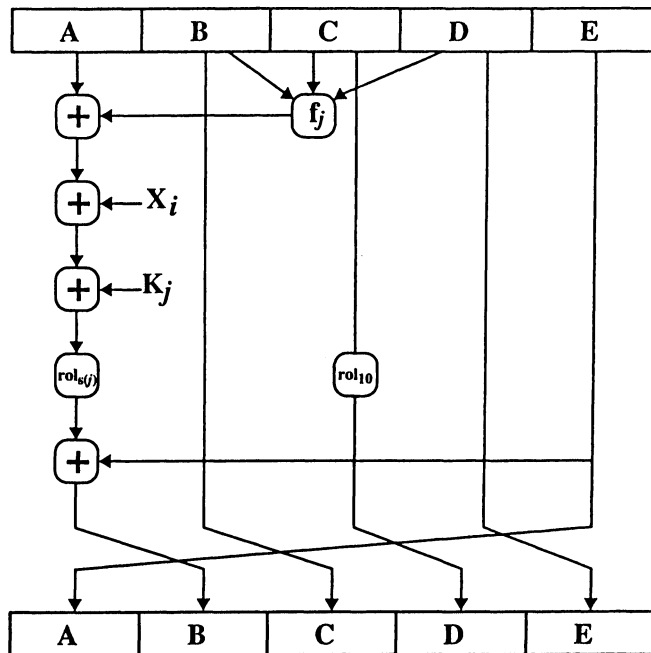


Figure 9.9 Elementary RIPEMD-160 Operation (single step).

trates the step operation. The order in which the five words (A, B, C, D, E) are used produces a word-level circular right shift of one word for each step. Note that this structure is almost identical to that of MD5.

One of the five primitive logical functions is used for each of the five rounds of the algorithm, with the functions used in reverse order on the right line (Figure 9.8). Each primitive function takes three 32-bit words as input and produces a 32-bit word output. Each function performs a set of bitwise logical operations; that is, the  $n$ th bit of the output is a function of the  $n$ th bit of the three inputs. The functions can be summarized as follows:

Step	Function Name	Function Value
$0 \leq j \leq 15$	$f_1 = f(j, B, C, D)$	$B \oplus C \oplus D$
$16 \leq j \leq 31$	$f_2 = f(j, B, C, D)$	$(B \wedge C) \vee (B \wedge D)$
$32 \leq j \leq 47$	$f_3 = f(j, B, C, D)$	$(B \vee C) \oplus D$
$48 \leq j \leq 63$	$f_4 = f(j, B, C, D)$	$(B \wedge D) \vee (C \wedge D)$
$64 \leq j \leq 79$	$f_5 = f(j, B, C, D)$	$B \oplus (C \vee D)$

The logical operators (AND, OR, NOT, XOR) are represented by the symbols ( $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\oplus$ ). Function  $f_1$  produces a parity bit. Function  $f_2$  is a conditional function: If B then C else D. Similarly,  $f_4$  can be stated as follows: If D then B else C. Table 9.4 is a truth table of the five functions.

**Table 9.4** Truth Table of Logical Functions for RIPEMD-160

B	C	D	f1	f2	f3	f4	f5
0	0	0	0	0	1	0	1
0	0	1	1	1	0	0	0
0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	1
1	0	0	1	0	1	0	0
1	0	1	0	0	0	1	1
1	1	0	0	1	1	1	0
1	1	1	1	1	0	1	0

The following pseudocode, adapted from [DOBB96b], defines the processing algorithm of one round.

```

A := CVq(0); B := CVq(1); C := CVq(2); D := CVq(3); E := CVq(4);
A' := CVq(0); B' := CVq(1); C' := CVq(2); D' := CVq(3); E' := CVq(4);
for j := 0 to 79 do
    T := rols(j)(A + f(j, B, C, D) + Xr(j) + K(j)) + E;
    A := E; E := D; D := rol10(C); C := B; B := T;
    T := rols'(j)(A' + f(79 - j, B', C', D') + Xr'(j) + K'(j)) + E';
    A' := E'; E' := D'; D' := rol10(C'); C' := B'; B' := T';
enddo
CVq+1(0) = CVq(1) + C + D'; CVq+1(1) = CVq(2) + D + E'; CVq+1(2) = CVq(3) + E + A';
CVq+1(3) = CVq(4) + A + B'; CVq+1(4) = CVq(0) + B + C';
where

```

A, B, C, D, E     = the five words of the buffer for the left line  
A', B', C', D', E' = the five words of the buffer for the right line  
j                    = step number;  $0 \leq j \leq 79$   
f(j, B, C, D)      = primitive logical function used in step j of the left line and  
                         step 79 - j of the right line  
rol<sub>s(j)</sub>             = circular left shift (rotation) of the 32-bit argument; s(j) is a  
                         function that determines the amount of rotation for a  
                         particular step  
X<sub>r(j)</sub>                = a 32-bit word from the current 512-bit input block; r(j) is a  
                         permutation function that selects a particular word  
K(j)                = additive constant used in step j  
+                    = addition modulo  $2^{32}$

The array of 32-bit words X[0..15] holds the value of the current 512-bit input block being processed. Within a round, each of the 16 words of X[i] is used exactly twice, during one step on each line; the order in which these words is used varies from round to round. Table 9.5a indicates the permutation used for each round in each line. The permutation  $\pi$  can be expressed as  $\pi(i) = 9i + 5 \pmod{16}$ . Table 9.5b defines the circular left shifts used in each round. The table shows the amount of

shift used for the step when a particular 32-bit word is being processed. This is not the order in which the shifts are used; the order depends on the order in which the words are used.

### RIPEMD-160 Design Decision

It is instructive to look at some of the design decisions made by the developers of RIPEMD-160 to get some idea of the level of detail that must be considered in designing a strong cryptographic hash function. The following points are reported in [DOBB96a]:

1. Two parallel lines of five rounds each are used to increase the complexity of finding collisions between rounds, which can be used as a starting point for finding a collision of the compression function.
2. For simplicity, the two lines use essentially the same logic, but the designers felt that it was necessary to introduce as many differences between the two lines as possible. The designers envisage that in the foreseeable future, it will become possible to attack one of the two lines and up to three rounds of the two parallel lines, but that the combination of the two parallel lines will resist attacks because of their differences. The differences are as follows:
  - a. The additive constants for the two lines are different (Table 9.3).
  - b. The order of the primitive logical functions ( $f_1$  through  $f_5$ ) is reversed.
  - c. The order of processing of the 32-bit words in the message block is different (Table 9.5a).
3. Except for the use of five words rather than four, the step operation for RIPEMD-160 (Figure 9.9) is identical to that of MD5 (Figure 9.3), with one additional exception: the rotation of the C word by 10 bit positions. This rotation avoids an MD5 attack that focuses on the most significant bit [BOER93]. The value of 10 was chosen because it is not used for the other rotations.
4. The permutation  $\rho$  (Table 9.5a) has the effect that two message words that are close in one round are relatively far apart in the next. The permutation  $\pi$  (Table 9.5a) was chosen such that two message words that are close in the left line will always be at least seven positions apart in the right line.
5. The circular left shifts (Table 9.5b) were chosen based on the following design criteria:
  - a. The shifts range from 5 to 15; a shift smaller than 5 was considered weak.
  - b. Every message word is rotated over different amounts for the five rounds, not all having the same parity.
  - c. The shifts applied to each word should not have a special pattern (e.g., the total should not be divisible by 32).
  - d. Not too many shift constants should be divisible by 4.

### Comparison with MD5 and SHA-1

RIPEMD-160 is similar in many ways to both MD5 and SHA-1, which is to be expected because all three algorithms derive from MD4. Table 9.6 highlights some of the differences and similarities. We can comment on several aspects of the design:

**Table 9.5** Elements of RIPEMD-160

(a) Permutations of Message Words

$i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho(i)$	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
$\pi(i)$	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12

Line	Round 1	Round 2	Round 3	Round 4	Round 5
Left	Identity	$\rho$	$\rho^2$	$\rho^3$	$\rho^4$
Right	$\pi$	$\rho\pi$	$\rho^2\pi$	$\rho^3\pi$	$\rho^4\pi$

(b) Circular Left Shift of Message Words (both lines)

Round	$X_0$	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$	$X_{11}$	$X_{12}$	$X_{13}$	$X_{14}$	$X_{15}$
1	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8
2	12	13	11	15	6	9	9	7	12	15	11	13	7	8	7	7
3	13	15	14	11	7	7	6	8	13	14	13	12	5	5	6	9
4	14	11	12	14	8	6	5	5	14	12	15	14	9	9	8	6
5	15	12	13	13	9	5	8	6	15	11	12	11	8	6	5	5

- **Resistance to brute-force attack:** All three algorithms are invulnerable to attacks against weak collision resistance. At 128 bits, MD5 is highly vulnerable to a birthday attack on strong collision resistance, whereas both SHA-1 and RIPEMD-160 are safe for the foreseeable future.
- **Resistance to cryptanalysis:** Significant progress has been made in the cryptanalysis of MD5, as discussed earlier. RIPEMD-160 was designed specifically to resist known cryptanalytic attacks. Although little is known about the

**Table 9.6** A Comparison of MD5, SHA-1, and RIPEMD-160

	MD5	SHA-1	RIPEMD-160
Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)	160 (5 paired rounds of 16)
Maximum message size	$\infty$	$2^{64} - 1$ bits	$2^{64} - 1$ bits
Primitive logical functions	4	4	5
Additive constants used	64	4	9
Endianness	Little-endian	Big-endian	Little-endian

**Table 9.7** Relative Performance of Several Hash Functions  
(coded in C++ on a 266-MHz Pentium.)

Algorithm	Mbps
MD5	32.4
SHA-1	14.4
RIPEMD-160	13.6

*Note:* Coded by Wei Dai; results are posted at <http://www.eskimo.com/~weidai/benchmarks.txt>

design principles for SHA-1, it also appears to be highly resistant to known cryptanalytic attacks. The use of two lines of processing, which doubles the number of steps performed, gives RIPEMD-160 added complexity, which should make cryptanalysis more difficult compared to SHA-1.

- **Speed:** All three algorithms rely on addition modulo  $2^{32}$  and simple bitwise logical operations, all of which should perform well on a 32-bit architecture. The added complexity and number of steps of SHA-1 and RIPEMD-160 does lead to a slowdown compared to MD5. Table 9.7 shows one set of results achieved on a 266-MHz Pentium. Similar relative results are reported in [BOSS96].
- **Little-endian versus big-endian architecture:** MD5 and RIPEMD-160 use a little-endian scheme for interpreting a message as a sequence of 32-bit words, whereas SHA-1 uses a big-endian scheme. There is no strong advantage to either approach.

## 9.4 HMAC

In Chapter 8, we looked at an example of a message authentication code (MAC) based on the use of a symmetric block cipher—namely, the data authentication algorithm defined in FIPS PUB 113. This has traditionally been the most common approach to constructing a MAC. In recent years, there has been increased interest in developing a MAC derived from a cryptographic hash code. The motivations for this interest are as follows:

1. Cryptographic hash functions such as MD5 and SHA-1 generally execute faster in software than symmetric block ciphers such as DES.
2. Library code for cryptographic hash functions is widely available.
3. There are no export restrictions from the United States or other countries for cryptographic hash functions, whereas symmetric block ciphers, even when used for MACs, are restricted.

A hash function such as MD5 was not designed for use as a MAC and cannot be used directly for that purpose because it does not rely on a secret key. There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm. The approach that has received the most support is HMAC [BELL96a, BELL96b]. HMAC has been issued as RFC 2104, has been chosen as

the mandatory-to-implement MAC for IP security, and is used in other Internet protocols, such as SSL.

### HMAC Design Objectives

RFC 2104 lists the following design objectives for HMAC:

- To use, without modifications, available hash functions. In particular, hash functions that perform well in software, and for which code is freely and widely available.
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions on the embedded hash function.

The first two objectives are important to the acceptability of HMAC. HMAC treats the hash function as a “black box.” This has two benefits. First, an existing implementation of a hash function can be used as a module in implementing HMAC. In this way, the bulk of the HMAC code is prepackaged and ready to use without modification. Second, if it is ever desired to replace a given hash function in an HMAC implementation, all that is required is to remove the existing hash function module and drop in the new module. This could be done if a faster hash function were desired. More important, if the security of the embedded hash function were compromised, the security of HMAC could be retained simply by replacing the embedded hash function with a more secure one (e.g., replacing MD5 with RIPEMD-160).

The last design objective in the preceding list is, in fact, the main advantage of HMAC over other proposed hash-based schemes. HMAC can be proven secure provided that the embedded hash function has some reasonable cryptographic strengths. We return to this point later in this section, but first we examine the structure of HMAC.

### HMAC Algorithm

Figure 9.10 illustrates the overall operation of HMAC. Define the following terms:

- $H$  = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)
- $M$  = message input to HMAC (including the padding specified in the embedded hash function)
- $Y_i$  =  $i$ th block of  $M$ ,  $0 \leq i \leq L - 1$
- $L$  = number of blocks in  $M$
- $b$  = number of bits in a block
- $n$  = length of hash code produced by embedded hash function
- $K$  = secret key; if key length is greater than  $b$ , the key is input to the hash function to produce an  $n$ -bit key; recommended length is  $\geq n$

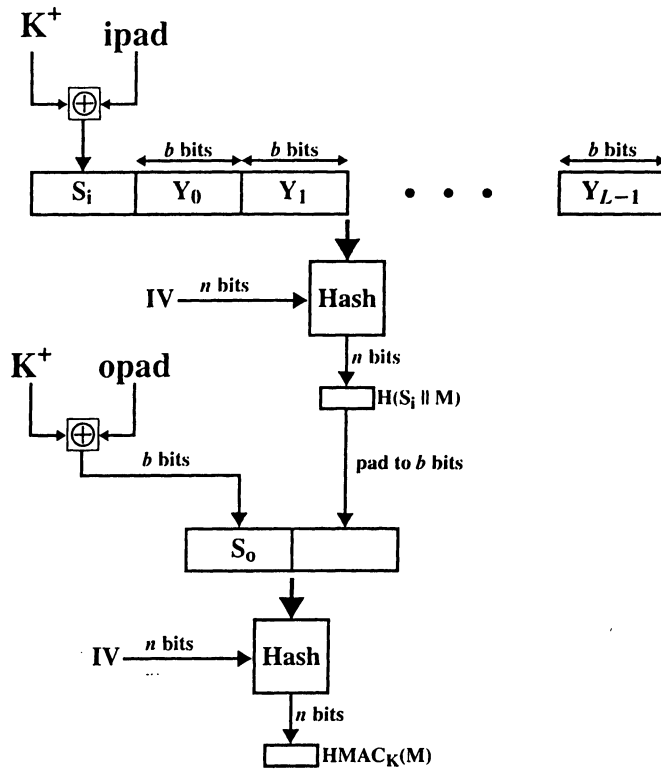


Figure 9.10 HMAC Structure.

$K^+$  =  $K$  padded with zeros on the left so that the result is  $b$  bits in length

ipad = 00110110 repeated  $b/8$  times

opad = 01011010 repeated  $b/8$  times

Then HMAC can be expressed as follows:

$$HMAC_K = H[(K^+ \oplus opad) \parallel H[(K^+ \oplus ipad) \parallel M]]$$

In words:

1. Append zeros to the left end of  $K$  to create a  $b$ -bit string  $K^+$  (e.g., if  $K$  is of length 160 bits and  $b = 512$ , then  $K$  will be appended with 44 zero bytes 0x00).
2. XOR (bitwise exclusive-OR)  $K^+$  with ipad to produce the  $b$ -bit block  $S_i$ .
3. Append  $M$  to  $S_i$ .
4. Apply  $H$  to the stream generated in step 3.
5. XOR  $K^+$  with opad to produce the  $b$ -bit block  $S_o$ .
6. Append the hash result from step 4 to  $S_o$ .
7. Apply  $H$  to the stream generated in step 6 and output the result.

Note that the XOR with *ipad* results in flipping one-half of the bits of *K*. Similarly, the XOR with *opad* results in flipping one-half of the bits of *K*, but a different set of bits. In effect, by passing  $S_i$  and  $S_o$  through the compression function of the hash algorithm, we have pseudorandomly generated two keys from *K*.

HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the hash compression function (for  $S_i$ ,  $S_o$ , and the block produced from the inner hash).

A more efficient implementation is possible, as shown in Figure 9.11. Two quantities are precomputed:

$$f(\text{IV}, (K^+ \oplus \text{ipad}))$$

$$f(\text{IV}, (K^+ \oplus \text{opad}))$$

where  $f(\text{cv}, \text{block})$  is the compression function for the hash function, which takes as arguments a chaining variable of  $n$  bits and a block of  $b$  bits and produces a chaining variable of  $n$  bits. These quantities only need to be computed initially and every time the key changes. In effect, the precomputed quantities substitute for the initial

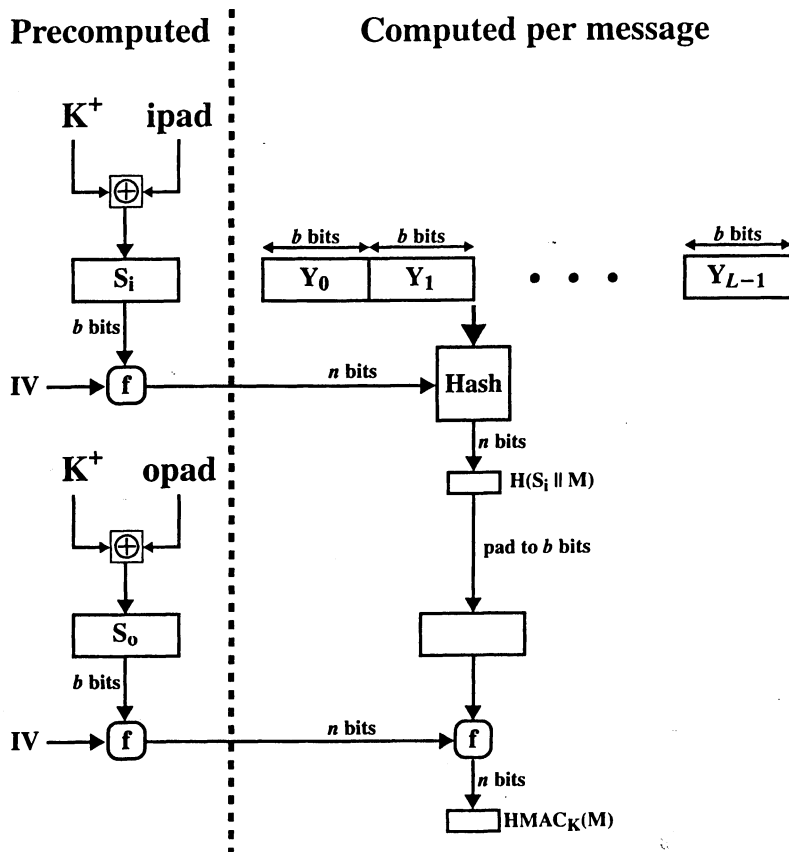


Figure 9.11 Efficient Implementation of HMAC.



value (IV) in the hash function. With this implementation, only one additional instance of the compression function is added to the processing normally produced by the hash function. This more efficient implementation is especially worthwhile if most of the messages for which a MAC is computed are short.

### Security of HMAC

The security of any MAC function based on an embedded hash function depends in some way on the cryptographic strength of the underlying hash function. The appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC.

The security of a MAC function is generally expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message-MAC pairs created with the same key. In essence, it is proved in [BELL96a] that for a given level of effort (time, message-MAC pairs), by a legitimate user and seen by an attacker, the probability of successful attack on HMAC is equivalent to one of the following attacks on the embedded hash function:

1. The attacker is able to compute an output of the compression function even with an IV that is random, secret, and unknown to the attacker.
2. The attacker finds collisions in the hash function even when the IV is random and secret.

In the first attack, we can view the compression function as equivalent to the hash function applied to a message consisting of a single  $b$ -bit block. For this attack, the IV of the hash function is replaced by a secret, random value of  $n$  bits. An attack on this hash function requires either a brute-force attack on the key, which is a level of effort on the order of  $2^n$ , or a birthday attack, which is a special case of the second attack, discussed next.

In the second attack, the attacker is looking for two messages  $M$  and  $M'$  that produce the same hash:  $H(M) = H(M')$ . This is the birthday attack discussed in Chapter 8. We have shown that this requires a level of effort of  $2^{n/2}$  for a hash length of  $n$ . On this basis, the security of MD5 is called into question, because a level of effort of  $2^{64}$  looks feasible with today's technology. Does this mean that a 128-bit hash function such as MD5 is unsuitable for HMAC? The answer is no, because of the following argument: To attack MD5, the attacker can choose any set of messages and work on these off line on a dedicated computing facility to find a collision. Because the attacker knows the hash algorithm and the default IV, the attacker can generate the hash code for each of the messages that the attacker generates. However, when attacking HMAC, the attacker cannot generate message/code pairs off line because the attacker does not know  $K$ . Therefore, the attacker must observe a sequence of messages generated by HMAC under the same key and perform the attack on these known messages. For a hash code length of 128 bits, this requires  $2^{64}$  observed blocks ( $2^{73}$  bits) generated using the same key. On a 1-Gbps link, one would need to observe a continuous stream of messages with no change in key for about 250,000 years in order to succeed. Thus, if speed is a concern, it is fully acceptable to use MD5 rather than SHA-1 or RIPEMD-160 as the embedded hash function for HMAC.

## 9.5 PROBLEMS

- 9.1 Comment on the differences between MD4 and MD5. Specifically, to what extent do you think that MD5 is stronger than MD4, and why?
- 9.2 In Figure 9.7, it is assumed that an array of 80 32-bit words is available to store the values of  $W_i$ , so that they can be precomputed at the beginning of the processing of a block. Now assume that space is at a premium. As an alternative, consider the use of a 16-word circular buffer that is initially loaded with  $W_0$  through  $W_{15}$ . Design an algorithm that, for each step  $t$ , computes the required input value  $W_t$ .
- 9.3 For SHA-1, show the values of  $W_{16}$ ,  $W_{17}$ ,  $W_{18}$ ,  $W_{19}$ .
- 9.4 Suppose  $a_1a_2a_3a_4$  are the 4 bytes in a 32-bit word. Each  $a_i$  can be viewed as an integer in the range 0 to 255, represented in binary. In a big-endian architecture, this word represents the integer

$$a_12^{24} + a_22^{16} + a_32^8 + a_4$$

In a little-endian architecture, this word represents the integer

$$a_42^{24} + a_32^{16} + a_22^8 + a_1$$

- a. MD5 and RIPEMD-160 assume a little-endian architecture. It is important that the message digest be independent of the underlying architecture. Therefore, to perform the modulo 2 addition operation of MD5 or RIPEMD-160 on a big-endian architecture, an adjustment must be made. Suppose  $X = x_1x_2x_3x_4$  and  $Y = y_1y_2y_3y_4$ . Show how the MD5 addition operation  $(X + Y)$  would be carried out on a big-endian machine.
- b. SHA-1 assumes a big-endian architecture. Show how the operation  $(X + Y)$  for SHA-1 would be carried out on a little-endian machine.

# CHAPTER 10

## DIGITAL SIGNATURES AND AUTHENTICATION PROTOCOLS

*To guard against the baneful influence exerted by strangers is therefore an elementary dictate of savage prudence. Hence before strangers are allowed to enter a district, or at least before they are permitted to mingle freely with the inhabitants, certain ceremonies are often performed by the natives of the country for the purpose of disarming the strangers of their magical powers, or of disinfecting, so to speak, the tainted atmosphere by which they are supposed to be surrounded.*

—*The Golden Bough*, Sir James George Frazer

The most important development from the work on public-key cryptography is the digital signature. The digital signature provides a set of security capabilities that would be difficult to implement in any other way. We begin this chapter with an overview of digital signatures. Then we look at authentication protocols, many of which depend on the use of the digital signature. Finally, we introduce the Digital Signature Standard (DSS).

### 10.1 DIGITAL SIGNATURES

#### Requirements

Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. Several forms of dispute between the two are possible.

For example, suppose that John sends an authenticated message to Mary, using one of the schemes of Figure 8.4. Consider the following disputes that could arise:

1. Mary may forge a different message and claim that it came from John. Mary would simply have to create a message and append an authentication code using the key that John and Mary share.
2. John can deny sending the message. Because it is possible for Mary to forge a message, there is no way to prove that John did in fact send the message.

Both scenarios are of legitimate concern. Here is an example of the first scenario: An electronic funds transfer takes place, and the receiver increases the amount of funds transferred and claims that the larger amount had arrived from the sender. An example of the second scenario is that an electronic mail message contains instructions to a stockbroker for a transaction that subsequently turns out badly. The sender pretends that the message was never sent.

In situations where there is not complete trust between sender and receiver, something more than authentication is needed. The most attractive solution to this problem is the digital signature. The digital signature is analogous to the handwritten signature. It must have the following properties:

- It must be able to verify the author and the date and time of the signature.
- It must be able to authenticate the contents at the time of the signature.
- The signature must be verifiable by third parties, to resolve disputes.

Thus, the digital signature function includes the authentication function.

On the basis of these properties, we can formulate the following requirements for a digital signature:

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information unique to the sender, to prevent both forgery and denial.
- It must be relatively easy to produce the digital signature.
- It must be relatively easy to recognize and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to retain a copy of the digital signature in storage.

A secure hash function, embedded in a scheme such as that of Figure 8.5c or 8.5d, satisfies these requirements.

A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated.

### **Direct Digital Signature**

The direct digital signature involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source. A dig-

ital signature may be formed by encrypting the entire message with the sender's private key (Figure 8.1c) or by encrypting a hash code of the message with the sender's private key (Figure 8.5c).

Confidentiality can be provided by further encrypting the entire message plus signature with either the receiver's public key (public-key encryption) or a shared secret key (conventional encryption); for example, see Figures 8.1d and 8.5d. Note that it is important to perform the signature function first and then an outer confidentiality function. In case of dispute, some third party must view the message and its signature. If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message. However, if the signature is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.

All direct schemes described so far share a common weakness: The validity of the scheme depends on the security of the sender's private key. If a sender later wishes to deny sending a particular message, the sender can claim that the private key was lost or stolen and that someone else forged his or her signature. Administrative controls relating to the security of private keys can be employed to thwart or at least weaken this ploy, but the threat is still there, at least to some degree. One example is to require every signed message to include a timestamp (date and time) and to require prompt reporting of compromised keys to a central authority.

Another threat is that some private key might actually be stolen from X at time T. The opponent can then send a message signed with X's signature and stamped with a time before or equal to T.

### Arbitrated Digital Signature

The problems associated with direct digital signatures can be addressed by using an arbiter.

As with direct signature schemes, there is a variety of arbitrated signature schemes. In general terms, they all operate as follows: Every signed message from a sender X to a receiver Y goes first to an arbiter A, who subjects the message and its signature to a number of tests to check its origin and content. The message is then dated and sent to Y with an indication that it has been verified to the satisfaction of the arbiter. The presence of A solves the problem faced by direct signature schemes: that X might disown the message.

The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly. The use of a trusted system, described in Chapter 16, might satisfy this requirement.

Table 10.1, based on scenarios described in [AKL83] and [MITC92], gives several examples of arbitrated digital signatures. In the first, conventional encryption is used. It is assumed that the sender X and the arbiter A share a secret key  $K_{xa}$  and that A and Y share secret key  $K_{ay}$ . X constructs a message M and computes its hash value  $H(M)$ . Then X transmits the message plus a signature to A. The signature consists of an identifier of X plus the hash value, all encrypted using  $K_{xa}$ . A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y, encrypted with  $K_{ay}$ . The message includes  $ID_X$ , the original message

**Table 10.1** Arbitrated Digital Signature Techniques

(a) Conventional Encryption, Arbiter Sees Message	
(1) $X \rightarrow A: M \parallel E_{K_{xa}} [ID_X \parallel H(M)]$	
(2) $A \rightarrow Y: E_{K_{ay}}[ID_X \parallel M \parallel E_{K_{xa}} [ID_X \parallel H(M)]] \parallel T$	
(b) Conventional Encryption, Arbiter Does Not See Message	
(1) $X \rightarrow A: ID_X \parallel E_{K_{xy}} [M] \parallel E_{K_{xa}} [ID_X \parallel H(E_{K_{xy}}[M])]$	
(2) $A \rightarrow Y: E_{K_{ay}}[ID_X \parallel E_{K_{xy}}[M] \parallel E_{K_{xa}} [ID_X \parallel H(E_{K_{xy}}[M])]] \parallel T$	
(c) Public-Key Encryption, Arbiter Does Not See Message	
(1) $X \rightarrow A: ID_X \parallel E_{K_{Rx}}[ID_X \parallel E_{K_{Uy}}(E_{K_{Rx}}[M])]$	
(2) $A \rightarrow Y: E_{K_{Ra}}[ID_X \parallel E_{K_{Uy}}[E_{K_{Rx}}[M]] \parallel T]$	

Notation: X = sender  
 Y = recipient  
 A = Arbiter  
 M = Message

from X, the signature, and a timestamp. Y can decrypt this to recover the message and the signature. The timestamp informs Y that this message is timely and not a replay. Y can store M and the signature. In case of dispute, Y, who claims to have received M from X, sends the following message to A:

$$E_{K_{ay}}[ID_X \parallel M \parallel E_{K_{xa}}[ID_X \parallel H(M)]]$$

The arbiter uses  $K_{ay}$  to recover  $ID_X$ , M, and the signature, and then uses  $K_{xa}$  to decrypt the signature and verify the hash code. In this scheme, Y cannot directly check X's signature; the signature is there solely to settle disputes. Y considers the message from X authentic because it comes through A. In this scenario, both sides must have a high degree of trust in A:

- X must trust A not to reveal  $K_{xa}$  and not to generate false signatures of the form  $E_{K_{xa}} [ID_X \parallel H(M)]$ .
- Y must trust A to send  $E_{K_{ay}}[ID_X \parallel M \parallel E_{K_{xa}}[ID_X \parallel H(M)] \parallel T]$  only if the hash value is correct and the signature was generated by X.
- Both sides must trust A to resolve disputes fairly.

If the arbiter does live up to this trust, then X is assured that no one can forge his signature and Y is assured that X cannot disavow his signature.

The preceding scenario also implies that A is able to read messages from X to Y and, indeed, that any eavesdropper is able to do so. Table 10.1b shows a scenario that provides the arbitration as before but also assures confidentiality. In this case it is assumed that X and Y share the secret key  $K_{xy}$ . Now X transmits an identifier, a copy of the message encrypted with  $K_{xy}$ , and a signature to A. The signature consists of the identifier plus the hash value of the encrypted message, all encrypted using  $K_{xa}$ . As before, A decrypts the signature and checks the hash value to validate

the message. In this case, A is working only with the encrypted version of the message and is prevented from reading it. A then transmits everything that it received from X, plus a timestamp, all encrypted with  $K_{ay}$ , to Y.

Although unable to read the message, the arbiter is still in a position to prevent fraud on the part of either X or Y. A remaining problem, one shared with the first scenario, is that the arbiter could form an alliance with the sender to deny a signed message, or with the receiver to forge the sender's signature.

All the problems just discussed can be resolved by going to a public-key scheme, one version of which is shown in Table 10.1c. In this case, X double encrypts a message M first with X's private key,  $KR_x$ , and then with Y's public key,  $KU_y$ . This is a signed, secret version of the message. This signed message, together with X's identifier, is encrypted again with  $KR_x$  and, together with  $ID_x$ , is sent to A. The inner, double-encrypted message is secure from the arbiter (and everyone else except Y). However, A can decrypt the outer encryption to assure that the message must have come from X (because only X has  $KR_x$ ). A checks to make sure that X's private/public key pair is still valid and, if so, verifies the message. Then A transmits a message to Y, encrypted with  $KR_y$ . The message includes  $ID_x$ , the double-encrypted message, and a timestamp.

This scheme has a number of advantages over the preceding two schemes. First, no information is shared among the parties before communication, preventing alliances to defraud. Second, no incorrectly dated message can be sent, even if  $KR_x$  is compromised, assuming that  $KR_y$  is not compromised. Finally, the content of the message from X to Y is secret from A and anyone else.

## 10.2 AUTHENTICATION PROTOCOLS

The basic tools described in Chapter 8 are used in a variety of applications, including the digital signature discussed in Section 10.1. Other uses are numerous and growing. In this section, we focus on two general areas (mutual authentication and one-way authentication) and examine some of the implications of authentication techniques in both.

### Mutual Authentication

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. This topic was examined in Section 5.3 (conventional techniques) and Section 6.3 (public-key techniques). There, the focus was key distribution. We return to this topic here to consider the wider implications of authentication.

Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness. To prevent masquerade and to prevent compromise of session keys, essential identification and session key information must be communicated in encrypted form. This requires the prior existence of secret or public keys that can be used for this purpose. The second issue, timeliness, is important

because of the threat of message replays. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

[GONG93] lists the following examples of replay attacks:

- **Simple replay:** The opponent simply copies a message and replays it later.
- **Repetition that can be logged:** An opponent can replay a timestamped message within the valid time window.
- **Repetition that cannot be detected:** This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only the replay message arrives.
- **Backward replay without modification:** This is a replay back to the message sender. This attack is possible if conventional encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order. The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. Instead, one of the following two general approaches is used:

- **Timestamps:** Party A accepts a message as fresh only if the message contains a timestamp that, in A's judgment, is close enough to A's knowledge of current time. This approach requires that clocks among the various participants be synchronized.
- **Challenge/response:** Party A, expecting a fresh message from B, first sends B a nonce (challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

It can be argued (e.g., [LAM92a]) that the timestamp approach should not be used for connection-oriented applications because of the inherent difficulties with this technique. First, some sort of protocol is needed to maintain synchronization among the various processor clocks. This protocol must be both fault tolerant, to cope with network errors, and secure, to cope with hostile attacks. Second, the opportunity for a successful attack will arise if there is a temporary loss of synchronization resulting from a fault in the clock mechanism of one of the parties. Finally, because of the variable and unpredictable nature of network delays, distributed clocks cannot be expected to maintain precise synchronization. Therefore, any timestamp-based procedure must allow for a window of time sufficiently large to accommodate network delays yet sufficiently small to minimize the opportunity for attack.

On the other hand, the challenge-response approach is unsuitable for a connectionless type of application because it requires the overhead of a handshake before any connectionless transmission, effectively negating the chief characteristic



of a connectionless transaction. For such applications, reliance on some sort of secure time server and a consistent attempt by each party to keep its clocks in synchronization may be the best approach (e.g., [LAM92b]).

### Conventional Encryption Approaches

As was discussed in Section 5.3, a two-level hierarchy of conventional encryption keys can be used to provide confidentiality for communication in a distributed environment. In general, this strategy involves the use of a trusted key distribution center (KDC). Each party in the network shares a secret key, known as a master key, with the KDC. The KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys, and for distributing those keys using the master keys to protect the distribution. This approach is quite common. As an example, we look at the Kerberos system in Chapter 11. The discussion in this subsection is relevant to an understanding of the Kerberos mechanisms.

Figure 5.9 illustrates a proposal initially put forth by Needham and Schroeder [NEED78] for secret key distribution using a KDC that, as was mentioned in Chapter 5, includes authentication features. The protocol<sup>1</sup> can be summarized as follows:

1.  $A \rightarrow KDC:$   $ID_A || ID_B || N_1$
2.  $KDC \rightarrow A:$   $E_{K_a}[K_s || ID_B || N_1 || E_{K_b}[K_s || ID_A]]$
3.  $A \rightarrow B:$   $E_{K_b}[K_s || ID_A]$
4.  $B \rightarrow A:$   $E_{K_s}[N_2]$
5.  $A \rightarrow B:$   $E_{K_s}[f(N_2)]$

Secret keys  $K_a$  and  $K_b$  are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key  $K_s$  to A and B. A securely acquires a new session key in step 2. The message in step 3 can be decrypted, and hence understood, only by B. Step 4 reflects B's knowledge of  $K_s$ , and step 5 assures B of A's knowledge of  $K_s$  and assures B that this is a fresh message because of the use of the nonce  $N_2$ . Recall from our discussion in Chapter 5 that the purpose of steps 4 and 5 is to prevent a certain type of replay attack. In particular, if an opponent is able to capture the message in step 3 and replay it, this might in some fashion disrupt operations at B.

Despite the handshake of steps 4 and 5, the protocol is still vulnerable to a form of replay attack. Suppose that an opponent, X, has been able to compromise an old session key. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3. Nevertheless, it is a potential security risk. X can impersonate A and trick B into using the old key by simply replaying step 3. Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay. If C can intercept the handshake message, step 4, then it can impersonate A's response, step 5. From this point on, C can send bogus messages to B that appear to B to come from A using an authenticated session key.

<sup>1</sup>The following format is used. A communication step in which P sends a message M to Q is represented as  $P \rightarrow Q: M$ .

Denning [DENN81, DENN82] proposes to overcome this weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3. Her proposal assumes that the master keys,  $K_a$  and  $K_b$ , are secure, and it consists of the following steps:

1.  $A \rightarrow KDC:$   $ID_A || ID_B$
2.  $KDC \rightarrow A:$   $E_{K_a}[K_s || ID_B || T || E_{K_b}[K_s || ID_A || T]]$
3.  $A \rightarrow B:$   $E_{K_b}[K_s || ID_A || T]$
4.  $B \rightarrow A:$   $E_{K_s}[N_1]$
5.  $A \rightarrow B:$   $E_{K_s}[f(N_1)]$

$T$  is a timestamp that assures  $A$  and  $B$  that the session key has only just been generated. Thus, both  $A$  and  $B$  know that the key distribution is a fresh exchange.  $A$  and  $B$  can verify timeliness by checking that

$$|Clock - T| < \Delta t_1 + \Delta t_2$$

where  $\Delta t_1$  is the estimated normal discrepancy between the KDC's clock and the local clock (at  $A$  or  $B$ ) and  $\Delta t_2$  is the expected network delay time. Each node can set its clock against some standard reference source. Because the timestamp  $T$  is encrypted using the secure master keys, an opponent, even with knowledge of an old session key, cannot succeed because a replay of step 3 will be detected by  $B$  as untimely.

A final point: Steps 4 and 5 were not included in the original presentation [DENN81] but were added later [DENN82]. These steps confirm the receipt of the session key at  $B$ .

The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network. [GONG92] points out a risk involved. The risk is based on the fact that the distributed clocks can become unsynchronized as a result of sabotage or faults in the clocks or the synchronization mechanism.<sup>2</sup> The problem occurs when a sender's clock is ahead of the intended recipient's clock. In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site. This replay could cause unexpected results. Gong refers to such attacks as suppress-replay attacks.

One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC's clock. The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonces. This latter alternative is not vulnerable to a suppress-replay attack because the nonces the recipient will choose in the future are unpredictable to the sender. The Needham/Schroeder protocol relies on nonces only but, as we have seen, has other vulnerabilities.

In [KEHN92], an attempt is made to respond to the concerns about suppress-replay attacks and at the same time fix the problems in the Needham/Schroeder pro-

<sup>2</sup>Such things can and do happen. In recent years, flawed chips were used in a number of computers and other electronic systems to track the time and date. The chips had a tendency to skip forward one day [NEUM90].

tol. Subsequently, an inconsistency in this latter protocol was noted and an improved strategy was presented in [NEUM93a].<sup>3</sup> The protocol is as follows:

1.  $A \rightarrow B$ :  $ID_A || N_a$
2.  $B \rightarrow KDC$ :  $ID_B || N_b || E_{K_b}[ID_A || N_a || T_b]$
3.  $KDC \rightarrow A$ :  $E_{K_a}[ID_B || N_a || K_s || T_b] || E_{K_b}[ID_A || K_s || T_b] || N_b$
4.  $A \rightarrow B$ :  $E_{K_b}[ID_A || K_s || T_b] || E_{K_s}[N_b]$

Let us follow this exchange step by step.

1. A initiates the authentication exchange by generating a nonce,  $N_a$ , and sending that plus its identifier to B in plaintext. This nonce will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.
2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce,  $N_b$ . This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness. B's message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDC to issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.
3. The KDC passes on to A B's nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a "ticket" that can be used by A for subsequent authentications, as will be seen. The KDC also sends A a block encrypted with the secret key shared by A and the KDC. This block verifies that B has received A's initial message ( $ID_B$ ) and that this is a timely message and not a replay ( $N_a$ ), and it provides A with a session key ( $K_s$ ) and the time limit on its use ( $T_b$ ).
4. A transmits the ticket to B, together with the B's nonce, the latter encrypted with the session key. The ticket provides B with the secret key that is used to decrypt  $E_{K_s}[N_b]$  to recover the nonce. The fact that B's nonce is encrypted with the session key authenticates that the message came from A and is not a replay.

This protocol provides an effective, secure means for A and B to establish a session with a secure session key. Furthermore, the protocol leaves A in possession of a key that can be used for subsequent authentication to B, avoiding the need to contact the authentication server repeatedly. Suppose that A and B establish a session using the aforementioned protocol and then conclude that session. Subsequently, but within the time limit established by the protocol, A desires a new session with B. The following protocol ensues:

1.  $A \rightarrow B$ :  $E_{K_b}[ID_A || K_s || T_b], N_a'$
2.  $B \rightarrow A$ :  $N_b', E_{K_s}[N_a']$
3.  $A \rightarrow B$ :  $E_{K_s}[N_b']$

<sup>3</sup>It really is hard to get these things right.

When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces  $N_a'$  and  $N_b'$  assure each party that there is no replay attack.

In all the foregoing, the time specified in  $T_b$  is a time relative to B's clock. Thus, this timestamp does not require synchronized clocks because B checks only self-generated timestamps.

### Public-Key Encryption Approaches

In Chapter 6, we presented one approach to the use of public-key encryption for the purpose of session key distribution (Figure 6.15). This protocol assumes that each of the two parties is in possession of the current public key of the other. It may not be practical to require this assumption.

A protocol using timestamps is provided in [DENN81]:

1.  $A \rightarrow AS:$   $ID_A || ID_B$
2.  $AS \rightarrow A:$   $E_{KR_{as}}[ID_A || KU_a || T] || E_{KR_{as}}[ID_B || KU_b || T]$
3.  $A \rightarrow B:$   $E_{KR_{as}}[ID_A || KU_a || T] || E_{KR_{as}}[ID_B || KU_b || T] || E_{KU_b}[E_{KR_a}[K_s || T]]$

In this case, the central system is referred to as an authentication server (AS), because it is not actually responsible for secret key distribution. Rather, the AS provides public-key certificates. The session key is chosen and encrypted by A; hence, there is no risk of exposure by the AS. The timestamps protect against replays of compromised keys.

This protocol is compact but, as before, requires synchronization of clocks. Another approach, proposed by Woo and Lam [WOO92a], makes use of nonces. The protocol consists of the following steps:

1.  $A \rightarrow KDC:$   $ID_A || ID_B$
2.  $KDC \rightarrow A:$   $E_{KR_{auth}}[ID_B || KU_b]$
3.  $A \rightarrow B:$   $E_{KU_b}[N_a || ID_A]$
4.  $B \rightarrow KDC:$   $ID_B || ID_A || E_{KU_{auth}}[N_a]$
5.  $KDC \rightarrow B:$   $E_{KR_{auth}}[ID_A || KU_a] || E_{KU_b}[E_{KR_{auth}}[N_a || K_s || ID_B]]$
6.  $B \rightarrow A:$   $E_{KU_a}[E_{KR_{auth}}[N_a || K_s || ID_B] || N_b]$
7.  $A \rightarrow B:$   $E_{K_s}[N_b]$

In step 1, A informs the KDC of its intention to establish a secure connection with B. The KDC returns to A a copy of B's public-key certificate (step 2). Using B's public key, A informs B of its desire to communicate and sends a nonce  $N_a$  (step 3). In step 4, B asks the KDC for A's public-key certificate and requests a session key; B includes A's nonce so that the KDC can stamp the session key with that nonce. The nonce is protected using the KDC's public key. In step 5, the KDC returns to B a copy of A's public-key certificate, plus the information  $\{N_a, K_s, ID_B\}$ . This information basically says that  $K_s$  is a secret key generated by the KDC on behalf of B and tied to  $N_a$ ; the binding of  $K_s$  and  $N_a$  will assure A that  $K_s$  is fresh. This triple is

encrypted, using the KDC's private key, to allow B to verify that the triple is, in fact, from the KDC. It is also encrypted using B's public key, so that no other entity may use the triple in an attempt to establish a fraudulent connection with A. In step 6, the triple  $\{N_a, K_s, ID_B\}$ , still encrypted with the KDC's private key, is relayed to A, together with a nonce  $N_b$  generated by B. All the foregoing are encrypted using A's public key. A retrieves the session key  $K_s$  and uses it to encrypt  $N_b$  and return it to B. This last message assures B of A's knowledge of the session key.

This seems to be a secure protocol that takes into account the various attacks. However, the authors themselves spotted a flaw and submitted a revised version of the algorithm in [WOO92b]:

1.  $A \rightarrow KDC:$   $ID_A || ID_B$
2.  $KDC \rightarrow A:$   $E_{KR_{auth}}[ID_B || KU_b]$
3.  $A \rightarrow B:$   $E_{KU_b}[N_a || ID_A]$
4.  $B \rightarrow KDC:$   $ID_B || ID_A || E_{KU_{auth}}[N_a]$
5.  $KDC \rightarrow B:$   $E_{KR_{auth}}[ID_A || KU_a] || E_{KU_b}[E_{KR_{auth}}[N_a || K_s || ID_A || ID_B]]$
6.  $B \rightarrow A:$   $E_{KU_a}[E_{KR_{auth}}[N_a || K_s || ID_A || ID_B] || N_b]$
7.  $A \rightarrow B:$   $E_{K_s}[N_b]$

The identifier of A,  $ID_A$ , is added to the set of items encrypted with the KDC's private key in steps 5 and 6. This binds the session key  $K_s$  to the identities of the two parties that will be engaged in the session. This inclusion of  $ID_A$  accounts for the fact that the nonce value  $N_a$  is considered unique only among all nonces generated by A, not among all nonces generated by all parties. Thus, it is the pair  $\{ID_A, N_a\}$  that uniquely identifies the connection request of A.

In both this example and the protocols described earlier, protocols that appeared secure were revised after additional analysis. These examples highlight the difficulty of getting things right the first time in the area of authentication.

### One-Way Authentication

One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be on line at the same time. Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.

The "envelope" or header of the e-mail message must be in the clear so that the message can be handled by the store-and-forward e-mail protocol, such as the Simple Mail Transfer Protocol (SMTP) or X.400. However, it is often desirable that the mail-handling protocol not require access to the plaintext form of the message, because that would require trusting the mail-handling mechanism. Accordingly, the e-mail message should be encrypted such that the mail-handling system is not in possession of the decryption key.

A second requirement is that of authentication. Typically, the recipient wants some assurance that the message is from the alleged sender.

### Conventional Encryption Approach

Using conventional encryption, the decentralized key distribution scenario illustrated in Figure 5.11 is impractical. This scheme requires the sender to issue a request to the intended recipient, await a response that includes a session key, and only then send the message.

With some refinement, the KDC strategy illustrated in Figure 5.9 is a candidate for encrypted electronic mail. Because we wish to avoid requiring that the recipient (B) be on line at the same time as the sender (A), steps 4 and 5 must be eliminated. For a message with content M, the sequence is as follows:

1.  $A \rightarrow KDC: ID_A || ID_B || N_1$
2.  $KDC \rightarrow A: E_{K_a}[K_s || ID_B || N_1 || E_{K_b}[K_s || ID_A]]$
3.  $A \rightarrow B: E_{K_b}[K_s, ID_A] || E_{K_s}[M]$

This approach guarantees that only the intended recipient of a message will be able to read it. It also provides a level of authentication that the sender is A. As specified, the protocol does not protect against replays. Some measure of defense could be provided by including a timestamp with the message. However, because of the potential delays in the e-mail process, such timestamps may have limited usefulness.

### Public-Key Encryption Approaches

We have already presented public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for confidentiality (Figure 8.1b), authentication (Figure 8.1c), or both (Figure 8.1d). These approaches require that either the sender know the recipient's public key (confidentiality) or that the recipient know the sender's public key (authentication) or both (confidentiality plus authentication). In addition, the public-key algorithm must be applied once or twice to what may be a long message.

If confidentiality is the primary concern, then the following may be more efficient:

$$A \rightarrow B: E_{K_{ub}}[K_s] || E_{K_s}[M]$$

In this case, the message is encrypted with a one-time secret key. A also encrypts this one-time key with B's public key. Only B will be able to use the corresponding private key to recover the one-time key and then use that key to decrypt the message. This scheme is more efficient than simply encrypting the entire message with B's public key.

If authentication is the primary concern, then a digital signature may suffice, as was illustrated in Figure 8.5c:

$$A \rightarrow B: M || E_{K_{Ra}}[H(M)]$$

This method guarantees that A cannot later deny having sent the message. However, this technique is open to another kind of fraud. Bob composes a message to his boss Alice that contains an idea that will save the company money. He appends his digital signature and sends it into the e-mail system. Eventually, the message will

get delivered to Alice's mailbox. But suppose that Max has heard of Bob's idea and gains access to the mail queue before delivery. He finds Bob's message, strips off his signature, appends his, and requeues the message to be delivered to Alice. Max gets credit for Bob's idea.

To counter such a scheme, both the message and signature can be encrypted with the recipient's public key:

$$A \rightarrow B: E_{KU_b}[M||E_{KR_a}[H(M)]]$$

The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate, described in Chapter 6. Now we have

$$A \rightarrow B: M||E_{KR_a}[H(M)]||E_{KR_{as}}[T||ID_A||KU_a]$$

In addition to the message, A sends B the signature, encrypted with A's private key, and A's certificate, encrypted with the private key of the authentication server. The recipient of the message first uses the certificate to obtain the sender's public key and verify that it is authentic and then uses the public key to verify the message itself. If confidentiality is required, then the entire message can be encrypted with B's public key. Alternatively, the entire message can be encrypted with a one-time secret key; the secret key is also transmitted, encrypted with B's public key. This approach is explored in Chapter 12.

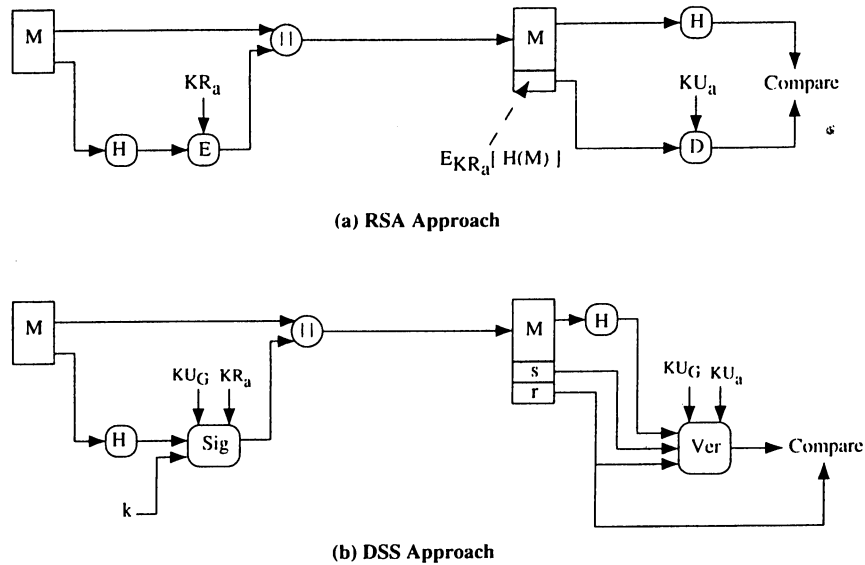
## 10.3 DIGITAL SIGNATURE STANDARD

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS PUB 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Hash Algorithm (SHA) described in Chapter 9 and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme. A further minor revision occurred in 1996.

### The DSS Approach

The DSS uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

Figure 10.1 contrasts the DSS approach for generating digital signatures to that used with RSA. In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signa-



**Figure 10.1** Two Approaches to Digital Signatures.

ture is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.

The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number  $k$  generated for this particular signature. The signature function also depends on the sender's private key ( $KR_a$ ) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key ( $KU_G$ ).<sup>4</sup> The result is a signature consisting of two components, labeled  $s$  and  $r$ .

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender's public key ( $KU_a$ ), which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component  $r$  if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

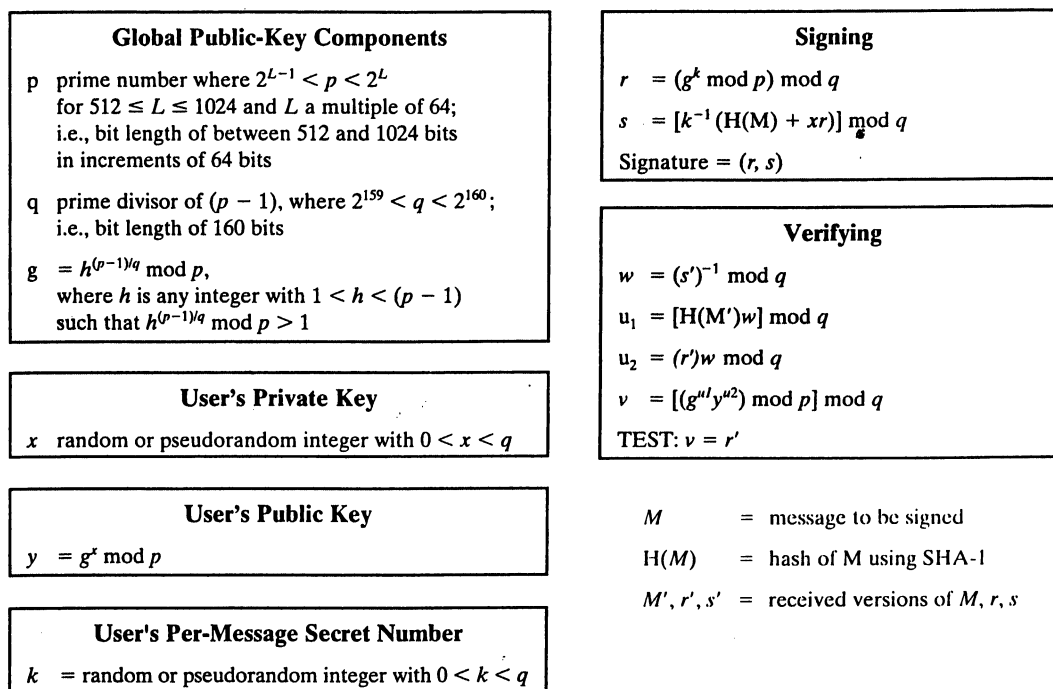
We turn now to the details of the algorithm.

### The Digital Signature Algorithm

The DSA is based on the difficulty of computing discrete logarithms (see Chapter 7) and is based on schemes originally presented by ElGamal [ELGA85] and Schnorr [SCHN91].

<sup>4</sup>It is also possible to allow these additional parameters to vary with each user so that they are a part of a user's public key. In practice, it is more likely that a global public key will be used that is separate from each user's public key.





**Figure 10.2** The Digital Signature Algorithm (DSA).

Figure 10.2 summarizes the algorithm. There are three parameters that are public and can be common to a group of users. A 160-bit prime number  $q$  is chosen. Next, a prime number  $p$  is selected with a length between 512 and 1024 bits such that  $q$  divides  $(p - 1)$ . Finally,  $g$  is chosen to be of the form  $h^{(p-1)/q} \bmod p$ , where  $h$  is an integer between 1 and  $(p - 1)$  with the restriction that  $g$  must be greater than 1.<sup>5</sup>

With these numbers in hand, each user selects a private key and generates a public key. The private key  $x$  must be a number from 1 to  $(q-1)$  and should be chosen randomly or pseudorandomly. The public key is calculated from the private key as  $y = g^x \bmod p$ . The calculation of  $y$  given  $x$  is relatively straightforward. However, given the public key  $y$ , it is believed to be computationally infeasible to determine  $x$ , which is the discrete logarithm of  $y$  to the base  $g$ , mod  $p$  (see Chapter 7).

To create a signature, a user calculates two quantities,  $r$  and  $s$ , that are functions of the public key components  $(p, q, g)$ , the user's private key  $(x)$ , the hash code of the message,  $H(M)$ , and an additional integer  $k$  that should be generated randomly or pseudorandomly and be unique for each signing.

At the receiving end, verification is performed using the formulas shown in Figure 10.2. The receiver generates a quantity  $v$  that is a function of the public-key components, the sender's public key, and the hash code of the incoming message. If this quantity matches the  $r$  component of the signature, then the signature is validated.

<sup>5</sup>In number-theoretic terms,  $g$  is of order  $q \bmod p$ ; see Chapter 7.

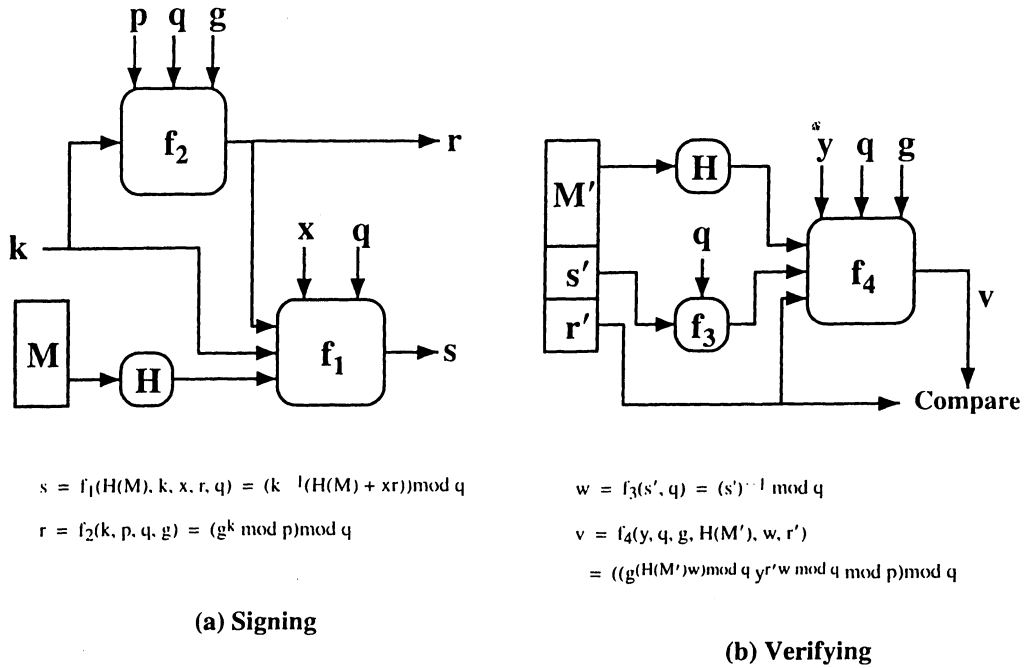
**Figure 10.3** DSS Signing and Verifying.

Figure 10.3 depicts the functions of signing and verifying.

The structure of the algorithm, as revealed in Figure 10.3, is quite interesting. Note that the test at the end is on the value  $r$ , which does not depend on the message at all. Instead,  $r$  is a function of  $k$  and the three global public-key components. The multiplicative inverse of  $k \pmod{p}$  is passed to a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover  $r$  using the incoming message and signature, the public key of the user, and the global public key. It is certainly not obvious from Figure 10.2 or Figure 10.3 that such a scheme would work. A proof is provided in Appendix 10A.

Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover  $k$  from  $r$  or to recover  $x$  from  $s$ .

Another point worth noting is that the only computationally demanding task in signature generation is the exponential calculation  $g^k \bmod p$ . Because this value does not depend on the message to be signed, it can be computed ahead of time. Indeed, a user could precalculate a number of values of  $r$  to be used to sign documents as needed. The only other somewhat demanding task is the determination of a multiplicative inverse,  $k^{-1}$ . Again, a number of these values can be precalculated.

## 10.4 RECOMMENDED READING

[AKL83] is the classic paper on digital signatures and is still highly relevant. A more recent, and excellent, survey is [MITC92].

- AKL83 Akl, S. "Digital Signatures: A Tutorial Survey." *Computer*, February 1983.  
 MITC92 Mitchell, C.; Piper, F.; and Wild, P. "Digital Signatures." In [SIMM92a].

## 10.5 PROBLEMS

- 10.1** In Section 10.2, we outlined the public-key scheme proposed in [WOO92a] for the distribution of secret keys. The revised version includes  $ID_A$  in steps 5 and 6. What attack, specifically, is countered by this revision?
- 10.2** The protocol referred to in Problem 10.1 can be reduced from seven steps to five, having the following sequence:
1.  $A \rightarrow B:$
  2.  $B \rightarrow KDC:$
  3.  $KDC \rightarrow B:$
  4.  $B \rightarrow A:$
  5.  $A \rightarrow B:$
- Show the message transmitted at each step. *Hint:* The final message in this protocol is the same as the final message in the original protocol.
- 10.3** Modify the digital signature techniques of Table 10.1a and 10.1b to enable the receiver to verify the signature.
- 10.4** Modify the digital signature technique of Table 10.1c to avoid triple encryption of the entire message.
- 10.5** In discussing Table 10.1c, it was stated that alliances to defraud were impossible. In fact, there is one possibility. Describe it and explain why it would have so little credibility that we can safely ignore it.
- 10.6** With reference to the suppress-replay attack described in Section 10.2,
- a. Give an example of an attack when a party's clock is ahead of that of the KDC.
  - b. Give an example of an attack when a party's clock is ahead of that of another party.
- 10.7** There are three typical ways to use nonces as challenges. Suppose  $N_a$  is a nonce generated by A, A and B share key K, and  $f()$  is a function such as an increment. The three usages are as follows:

Usage 1	Usage 2	Usage 3
(1) $A \rightarrow B: N_a$ (2) $A \rightarrow B: E_K[N_a]$	(1) $A \rightarrow B: E_K[N_a]$ (2) $A \rightarrow B: N_a$	(1) $A \rightarrow B: E_K[N_a]$ (2) $A \rightarrow B: E_K[f(N_a)]$

Describe situations for which each usage is appropriate.

- 10.8** Dr. Watson patiently waited until Sherlock Holmes finished. "Some interesting problem to solve, Holmes?" he asked when Holmes finally logged out.

"Oh, not exactly. I merely checked my e-mail and then made a couple of network experiments instead of my usual chemical ones. I have only one client now and I have already solved his problem. If I remember correctly, you once mentioned cryptography among your other hobbies, so it may interest you."

"Well, I am only an amateur cryptologist, Holmes. But of course I am interested in the problem. What is it about?"

"My client is Mr. Hosgrave, director of a small but progressive bank. The bank is fully computerized and of course uses network communications extensively. The bank already uses RSA to protect its data and to digitally sign documents that are communicated. Now the bank wants to introduce some changes in its procedures; in particular, it needs to digitally sign some documents by *two* signatories so that

1. The first signatory prepares the document, forms its signature, and passes the document to the second signatory.

2. The second signatory as a first step must verify that the document was really signed by the first signatory. She then incorporates her signature into the document's signature so that the recipient, as well as any member of the public, may verify that the document was indeed signed by both signatories. In addition, only the second signatory has to be able to verify the document's signature after the step (1); that is, the recipient (or any member of the public) should be able to verify only the complete document with signatures of both signatories, but not the document in its intermediate form, where only one signatory has signed it. Moreover, the bank would like to make use of its existing modules that support RSA-style digital signatures."

"Hm, I understand how RSA can be used to digitally sign documents by *one* signatory, Holmes. I guess you have solved the problem of Mr. Hosgrave by appropriate generalization of RSA digital signatures."

"Exactly, Watson," nodded Sherlock Holmes. "Originally, the RSA digital signature was formed by encrypting the document by the signatory's private decryption key 'd', and the signature could be verified by anyone through its decryption using publicly known encryption key 'e'. One can verify that the signature S was formed by the person who knows d, which is supposed to be the only signatory. Now the problem of Mr. Hosgrave can be solved in the same way by slight generalization of the process, that is ..."

- 10.9 DSA specifies that if the signature-generation process results in a value of  $s = 0$ , a new value of  $k$  should be generated and the signature should be recalculated. Why?
- 10.10 What happens if a  $k$  value used in creating a DSA signature is compromised?
- 10.11 The DSS document includes a recommended algorithm for testing a number for primality, as follows:
- (1) [**Choose  $w$** ] Let  $w$  be a random odd integer. Then  $(w - 1)$  is even and can be expressed in the form  $2^a m$  with  $m$  odd. That is,  $2^a$  is the largest power of 2 that divides  $(w - 1)$ .
  - (2) [**Generate  $b$** ] Let  $b$  be a random integer in the range  $1 < b < w$ .
  - (3) [**Exponentiate**] Set  $j = 0$  and  $z = b^m \bmod w$ .
  - (4) [**Done?**] If  $j = 0$  and  $z = 1$ , or if  $z = w - 1$ , then  $w$  passes the test and may be prime; go to step 8.
  - (5) [**Terminate?**] If  $j > 0$  and  $z = 1$ , then  $w$  is not prime; terminate algorithm for this  $w$ .
  - (6) [**Increase  $j$** ] Set  $j = j + 1$ . If  $j < a$ , set  $z = z^2 \bmod w$  and go to step 4.
  - (7) [**Terminate**]  $w$  is not prime; terminate algorithm for this  $w$ .
  - (8) [**Test again?**] If enough random values of  $b$  have been tested, then accept  $w$  as prime and terminate algorithm; otherwise, go to step 2.
- a. Explain how the algorithm works.
  - b. Show that it is equivalent to the Miller-Rabin test described in Chapter 7.
- 10.12 With DSS, because the value of  $k$  is generated for each signature, even if the same message is signed twice on different occasions, the signatures will differ. This is not true of RSA signatures. What is the practical implication of this difference?
- 10.13 It is tempting to try to develop a variation on Diffie-Hellman that could be used as a digital signature. Here is one that is simpler than DSA and that does not require a secret random number in addition to the private key:

**Public elements:**

- $q$  prime number
- $\alpha$   $\alpha < q$  and  $\alpha$  is a primitive root of  $q$

**Private key:**

- $X$   $X < q$

**Public key:**

- $Y = \alpha^X \bmod q$

To sign a message  $M$ , compute  $h = H(M)$ , the hash code of the message. We require that  $\gcd(h, p-1) = 1$ . If not, append the hash to the message and calculate a new hash.

Continue this process until a hash code is produced that is relatively prime to  $(p - 1)$ . Then calculate  $Z$  to satisfy  $Z \times h \equiv X \pmod{(p - 1)}$ . The signature of the message is  $\alpha^Z$ . To verify the signature, a user verifies that  $(\alpha^Z)^h = \alpha^X \pmod{q}$ .

- a. Show that this scheme works. That is, show that the verification process produces an equality if the signature is valid.
- b. Show that the scheme is unacceptable by describing a simple technique for forging a user's signature on an arbitrary message.

## APPENDIX 10A PROOF OF THE DIGITAL SIGNATURE ALGORITHM

The purpose of this appendix is to provide a proof that in the signature verification we have  $v = r$  if the signature is valid. The following proof is based on that which appears in the FIPS standard, but it includes additional details to make the derivation clearer.

---

**LEMMA 1.** For any integer  $t$ ,

$$\begin{array}{ll} \text{if} & g = h^{(p-1)/q} \pmod{p} \\ \text{then} & g^t \pmod{p} = g^{t \pmod{q}} \pmod{p} \end{array}$$

**Proof:** By Fermat's theorem (Chapter 7), because  $h$  is relatively prime to  $p$ , we have  $h^{p-1} \pmod{p} = 1$ . Hence, for any nonnegative integer  $n$ ,

$$\begin{aligned} g^{nq} \pmod{p} &= (h^{(p-1)/q} \pmod{p})^{nq} \pmod{p} \\ &= h^{((p-1)/q)nq} \pmod{p} && \text{by the rules of modular arithmetic} \\ &= h^{(p-1)n} \pmod{p} \\ &= ((h^{p-1} \pmod{p})^n) \pmod{p} && \text{by the rules of modular arithmetic} \end{aligned}$$

So, for nonnegative integers  $n$  and  $z$ , we have

$$\begin{aligned} g^{nq+z} \pmod{p} &= (g^{nq} g^z) \pmod{p} \\ &= ((g^{nq} \pmod{p})(g^z \pmod{p})) \pmod{p} \\ &= g^z \pmod{p} \end{aligned}$$

Any nonnegative integer  $t$  can be represented uniquely as  $t = nq + z$ , where  $n$  and  $z$  are nonnegative integers and  $0 < z < q$ . So  $z = t \pmod{q}$ . The result follows. **QED.**

---

**LEMMA 2.** For nonnegative integers  $a$  and  $b$ ,  $g^{(a \pmod{q} + b \pmod{q})} \pmod{p} = g^{(a+b) \pmod{q}} \pmod{p}$

**Proof:** By Lemma 1, we have

$$\begin{aligned} g^{(a \pmod{q} + b \pmod{q})} \pmod{p} &= g^{(a \pmod{q} + b \pmod{q}) \pmod{q}} \pmod{p} \\ &= g^{(a+b) \pmod{q}} \pmod{p} \end{aligned}$$

**QED.**

**LEMMA 3.**

$$y^{(rw) \bmod q} \bmod p = g^{(xrw) \bmod q} \bmod p$$

**Proof:** By definition (Figure 10.2),  $y = g^x \bmod p$ . Then

$$\begin{aligned} y^{(rw) \bmod q} \bmod p &= (g^x \bmod p)^{(rw) \bmod q} \bmod p \\ &= g^{x((rw) \bmod q)} \bmod p && \text{by the rules of modular arithmetic} \\ &= g^{(x((rw) \bmod q)) \bmod q} \bmod p && \text{by Lemma 1} \\ &= g^{(xrw) \bmod q} \bmod p \end{aligned}$$

**QED.**

**LEMMA 4.**

$$((H(M) + xr)w) \bmod q = k$$

**Proof:** By definition (Figure 10.2),  $s = (k^{-1}(H(M) + xr)) \bmod q$ . Also, because  $q$  is prime, any nonnegative integer less than  $q$  has a multiplicative inverse (Chapter 7). So  $(kk^{-1}) \bmod q = 1$ . Then

$$\begin{aligned} (ks) \bmod q &= (k((k^{-1}(H(M) + xr)) \bmod q)) \bmod q \\ &= (((k(k^{-1}(H(M) + xr)))) \bmod q \\ &= (((kk^{-1}) \bmod q)((H(M) + xr) \bmod q)) \bmod q \\ &= ((H(M) + xr) \bmod q \end{aligned}$$

By definition,  $w = s^{-1} \bmod q$  and therefore  $(ws) \bmod q = 1$ . Therefore,

$$\begin{aligned} ((H(M) + xr)w) \bmod q &= (((H(M) + xr) \bmod q)(w \bmod q)) \bmod q \\ &= (((ks) \bmod q)(w \bmod q)) \bmod q \\ &= (kws) \bmod q \\ &= ((k \bmod q)((ws) \bmod q)) \bmod q \\ &= k \bmod q \end{aligned}$$

Because  $0 < k < q$ , we have  $k \bmod q = k$ . **QED.**

---

**THEOREM:** Using the definitions of Figure 10.2,  $v = r$ .

$$\begin{aligned}
 v &= ((g^{u_1} y^{u_2}) \bmod p) \bmod q && \text{by definition} \\
 &= ((g^{(H(M)w \bmod q) y^{(rw \bmod q)}} \bmod p) \bmod q && * \\
 &= ((g^{(H(M)w \bmod q) g^{(xrw \bmod q)}} \bmod p) \bmod q && \text{by Lemma 3} \\
 &= ((g^{(H(M)w \bmod q + (xrw \bmod q))} \bmod p) \bmod q \\
 &= ((g^{(H(M)w + xrw \bmod q)} \bmod p) \bmod q && \text{by Lemma 2} \\
 &= ((g^{((H(M) + xr)w \bmod q)} \bmod p) \bmod q \\
 &= (g^k \bmod p) \bmod q && \text{by Lemma 4} \\
 &= r && \text{by definition}
 \end{aligned}$$

**QED.**





**PART  
THREE**

# Network Security Practice



# CHAPTER 11

## AUTHENTICATION APPLICATIONS

*We cannot enter into alliance with neighboring princes until we are acquainted with their designs.*

—*The Art of War*, Sun Tzu

**T**his chapter examines some of the authentication functions that have been developed to support application-level authentication and digital signatures.

We begin by looking at one of the earliest and also one of the most widely used services, which is known as Kerberos. Next, we examine the X.509 directory authentication service. This standard is important as part of the directory service that it supports but is also a basic building block used in other standards, such as S/MIME, discussed in Chapter 12.

### 11.1 KERBEROS

Kerberos<sup>1</sup> is an authentication service developed as part of Project Athena at MIT. The problem that Kerberos addresses is this: Assume an open dis-

---

<sup>1</sup>"In Greek mythology, a many headed dog, commonly three, perhaps with a serpent's tail, the guardian of the entrance of Hades." From *Dictionary of Subjects and Symbols in Art*, by James Hall, Harper & Row, 1979. Just as the Greek Kerberos has three heads, the modern Kerberos was intended to have three components to guard a network's gate: authentication, accounting, and audit. The last two heads were never implemented.

tributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services. In particular, the following three threats exist:

- A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
- A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
- A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building in elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes described in this book, Kerberos relies exclusively on conventional encryption, making no use of public-key encryption.

Two versions of Kerberos are in common use. Version 4 [MILL88, STEI88] is still widely used. Version 5 [KOHL94] corrects some of the security deficiencies of version 4 and has been issued as a draft Internet Standard (RFC 1510).<sup>2</sup>

We begin this section with a brief discussion of the motivation for the Kerberos approach. Then, because of the complexity of Kerberos, it is best to start with a description of the authentication protocol used in version 4. This enables us to see the essence of the Kerberos strategy without considering some of the details required to handle subtle security threats. Finally, we examine version 5.

### Motivation

If a set of users is provided with dedicated personal computers that have no network connections, then a user's resources and files can be protected by physically securing each personal computer. When these users instead are served by a centralized time-sharing system, the time-sharing operating system must provide the security. The operating system can enforce access control policies based on user identity and use the logon procedure to identify users.

Today, neither of these scenarios is typical. More common is a distributed architecture consisting of dedicated user workstations (clients) and distributed or centralized servers. In this environment, three approaches to security can be envisioned:

1. Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).

<sup>2</sup> Versions 1 through 3 were internal development versions. Version 4 is the "original" Kerberos.

2. Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.
3. Require the user to prove identity for each service invoked. Also require that servers prove their identity to clients.

In a small, closed environment, in which all systems are owned and operated by a single organization, the first or perhaps the second strategy may suffice.<sup>3</sup> But in a more open environment, in which network connections to other machines are supported, the third approach is needed to protect user information and resources housed at the server. This third approach is supported by Kerberos. Kerberos assumes a distributed client/server architecture and employs one or more Kerberos servers to provide an authentication service.

The first published report on Kerberos [STEI88] listed the following requirements for Kerberos:

- **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- **Reliable:** For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.
- **Transparent:** Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.
- **Scalable:** The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture.

To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78], which was discussed in Chapter 5. It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.<sup>4</sup>

### Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. Viewing the protocol as a whole, it is difficult to see the need for the many elements contained therein. Therefore, we adopt a strategy used by Bill Bryant of Project Athena [BRYA88] and build up to the full protocol by look-

<sup>3</sup>However, even a closed environment faces the threat of attack by a disgruntled employee.

<sup>4</sup>Remember that the security of the Kerberos server should not automatically be assumed but must be guarded carefully (e.g., in a locked room). It is well to remember the fate of the Greek Kerberos, whom Hercules was ordered by Eurystheus to capture as his Eleventh Labor: "Hercules found the great dog on its chain and seized it by the throat. At once the three heads tried to attack, and Kerberos lashed about with his powerful tail. Hercules hung on grimly, and Kerberos relaxed into unconsciousness. Eurystheus may have been surprised to see Hercules alive—when he saw the three slaving heads and the huge dog they belonged to he was frightened out of his wits, and leapt back into the safety of his great bronze jar." From *The Hamlyn Concise Dictionary of Greek and Roman Mythology*, by Michael Stapleton, Hamlyn, 1982.

ing first at several hypothetical dialogues. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue.

After examining the protocol, we look at some other aspects of version 4.

### A Simple Authentication Dialogue

In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers must be able to confirm the identities of clients who request service. Each server can be required to undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. Consider the following hypothetical dialogue:

- (1)  $C \rightarrow AS: ID_C || P_C || ID_V$
  - (2)  $AS \rightarrow C: Ticket$
  - (3)  $C \rightarrow V: ID_C || Ticket$
- $$Ticket = E_{K_V}[ID_C || AD_C || ID_V]$$

where

C	=	client
AS	=	authentication server
V	=	server
$ID_C$	=	identifier of user on C
$ID_V$	=	identifier of V
$P_C$	=	password of user on C
$AD_C$	=	network address of C
$K_V$	=	secret encryption key shared by AS and V
	=	concatenation

In this scenario, the user logs on to a workstation and requests access to server V. The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password. The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V. If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic. To do so, the AS creates a ticket that contains the user's ID and network address and the server's ID. This ticket is encrypted using the secret key shared by the AS and this server. This ticket is then sent back to C. Because the ticket is encrypted, it cannot be altered by C or by an opponent.

With this ticket, C can now apply to V for service. C sends a message to V containing C's ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message. If these two match, the server considers the user authenticated and grants the requested service.

Each of the ingredients of message (3) is significant. The ticket is encrypted to prevent alteration or forgery. The server's ID ( $ID_V$ ) is included in the ticket so that the server can verify that it has decrypted the ticket properly.  $ID_C$  is included in the ticket to indicate that this ticket has been issued on behalf of C. Finally,  $AD_C$  serves to counter the following threat. An opponent could capture the ticket transmitted in message (2), then use the name  $ID_C$  and transmit a message of form (3) from another workstation. The server would receive a valid ticket that matches the user ID and grant access to the user on that other workstation. To prevent this attack, the AS includes in the ticket the network address from which the original request came. Now the ticket is valid only if it is transmitted from the same workstation that initially requested the ticket.

### A More Secure Authentication Dialogue

Although the foregoing scenario solves some of the problems of authentication in an open network environment, problems remain. Two in particular stand out. First, we would like to minimize the number of times that a user has to enter a password. Suppose each ticket can be used only once. If user C logs on to a workstation in the morning and wishes to check his or her mail at a mail server, C must supply a password to get a ticket for the mail server. If C wishes to check the mail several times during the day, each attempt requires reentering the password. We can improve matters by saying that tickets are reusable. For a single logon session, the workstation can store the mail server ticket after it is received and use it on behalf of the user for multiple accesses to the mail server.

However, under this scheme it remains the case that a user would need a new ticket for every different service. If a user wished to access a print server, a mail server, a file server, and so on, the first instance of each access would require a new ticket and hence require the user to enter the password.

The second problem is that the earlier scenario involved a plaintext transmission of the password (message 1). An eavesdropper could capture the password and use any service accessible to the victim.

To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the ticket-granting server (TGS). The new but still hypothetical scenario is as follows:

#### Once per user logon session:

- (1)  $C \rightarrow AS: ID_C || ID_{TGS}$
- (2)  $AS \rightarrow C: E_{K_C}[Ticket_{TGS}]$

#### Once per type of service:

- (3)  $C \rightarrow TGS: ID_C || ID_V || Ticket_{TGS}$
- (4)  $TGS \rightarrow C: Ticket_V$

#### Once per service session:

- (5)  $C \rightarrow V: ID_C || Ticket_V$

$$Ticket_{TGS} = E_{K_{TGS}}[ID_C || AD_C || ID_{TGS} || TS_1 || Lifetime_1]$$

$$Ticket_V = E_{K_V}[ID_C || AD_C || ID_V || TS_2 || Lifetime_2]$$

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ( $\text{Ticket}_{\text{TGS}}$ ) from the AS. This ticket is saved by the client module in the user workstation. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested. Let us look at the details of this scheme:

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID to the AS, together with the TGS ID, indicating a request to use the TGS service.
2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password. When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.

Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext. The ticket itself consists of the ID and network address of the user, and the ID of the TGS. This corresponds to the first scenario. The idea is that this ticket can be used by the client to request multiple service-granting tickets. So the ticket-granting ticket is to be reusable. However, we do not wish an opponent to be able to capture the ticket and use it. Consider the following scenario: An opponent captures the ticket and waits until the user has logged off his or her workstation. The opponent either gains access to that workstation or configures his workstation with the same network address as that of the victim. The opponent would be able to reuse the ticket to spoof the TGS. To counter this, the ticket includes a timestamp, indicating the date and time at which the ticket was issued, and a lifetime, indicating the length of time for which the ticket is valid (e.g., eight hours). Thus, the client now has a reusable ticket and need not bother the user for a password for each new service request. Finally, note that the ticket-granting ticket is encrypted with a secret key known only to the AS and the TGS. This prevents alteration of the ticket. The ticket is reencrypted with a key based on the user's password. This assures that the ticket can be recovered only by the correct user, providing the authentication.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4:

3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.
4. The TGS decrypts the incoming ticket and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to V, the TGS issues a ticket to grant access to the requested service.



The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key ( $K_v$ ) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5:

5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service-granting ticket. The server authenticates by using the contents of the ticket.

This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

#### **The Version 4 Authentication Dialogue**

Although the foregoing scenario enhances security compared to the first attempt, two additional problems remain. The heart of the first problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay. An opponent could eavesdrop on the network and capture a copy of the ticket-granting ticket and then wait for the legitimate user to log out. Then the opponent could forge the legitimate user's network address and send the message of step (3) to the TGS. This would give the opponent unlimited access to the resources and files available to the legitimate user.

Similarly, if an opponent captures a service-granting ticket and uses it before it expires, the opponent has access to the corresponding service.

Thus, we arrive at an additional requirement. A network service (the TGS or an application service) must be able to prove that the person using a ticket is the same person to whom that ticket was issued.

The second problem is that there may be a requirement for servers to authenticate themselves to users. Without such authentication, an opponent could sabotage the configuration so that messages to a server were directed to another location. The false server would then be in a position to act as a real server and capture any information from the user and deny the true service to the user.

We examine these problems in turn and refer to Table 11.1, which shows the actual Kerberos protocol.

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. The threat is that an opponent will steal the ticket and use it before it expires. To get around this problem, let us have the AS provide both the client and the TGS with a secret piece of information in a secure manner. Then the client can prove its identity to the TGS by revealing the secret information, again in a secure manner.

**Table 11.1** Summary of Kerberos Version 4 Message Exchanges

<b>(a) Authentication Service Exchange: to obtain ticket-granting ticket</b>	
(1) $C \rightarrow AS:$	$ID_c    ID_{tgs}    TS_1$
(2) $AS \rightarrow C:$	$E_{K_c} [K_{c,tgs}    ID_{tgs}    TS_2    Lifetime_2    Ticket_{tgs}]$
	$Ticket_{tgs} = E_{K_{tgs}} [K_{c,tgs}    ID_c    AD_c    ID_{tgs}    TS_2    Lifetime_2]$
<b>(b) Ticket-Granting Service Exchange: to obtain service-granting ticket</b>	
(3) $C \rightarrow TGS:$	$ID_v    Ticket_{tgs}    Authenticator_c$
(4) $TGS \rightarrow C:$	$E_{K_{c,tgs}} [K_{c,v}    ID_v    TS_4    Ticket_v]$
	$Ticket_{tgs} = E_{K_{tgs}} [K_{c,tgs}    ID_c    AD_c    ID_{tgs}    TS_2    Lifetime_2]$
	$Ticket_v = E_{K_v} [K_{c,v}    ID_c    AD_c    ID_v    TS_4    Lifetime_4]$
	$Authenticator_c = E_{K_{c,tgs}} [ID_c    AD_c    TS_3]$
<b>(c) Client/Server Authentication Exchange: to obtain service</b>	
(5) $C \rightarrow K:$	$Ticket_v    Authenticator_c$
(6) $K \rightarrow C:$	$E_{K_{c,v}} [TS_5 + 1]$ (for mutual authentication)
	$Ticket_v = E_{K_v} [K_{c,v}    ID_c    AD_c    ID_v    TS_4    Lifetime_4]$
	$Authenticator_c = E_{K_{c,v}} [ID_c    AD_c    TS_3]$

An efficient way of accomplishing this is to use an encryption key as the secure information; this is referred to as a session key in Kerberos.

Table 11.1a shows the technique for distributing the key, known as a session key. As before, the client sends a message to the AS requesting access to the TGS. The AS responds with a message, encrypted with a key derived from the user's password ( $K_c$ ), that contains the ticket, encrypted with a key derived from the user's password ( $K_c$ ). The encrypted message also contains a copy of the session key,  $K_{c,tgs}$ , where the subscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with  $K_c$ , only the user's client can read it. The same session key is included in the ticket, which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS.

Before proceeding, note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

Armed with the ticket and the session key, C is ready to approach the TGS. As before, C sends the TGS a message that includes the ticket plus the ID of the requested service (message (3) in Table 11.1b). In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key

$K_{c,tgs}$ . In effect, the ticket says, "Anyone who uses  $K_{c,tgs}$  must be C." The TGS uses the session key to decrypt the authenticator. The TGS can then check the name and address from the authenticator with that of the ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner. In effect, the authenticator says, "At time  $TS_3$ , I hereby use  $K_{c,tgs}$ ." Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered.

The reply from the TGS, in message (4), follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V, the ID of V, and the timestamp of the ticket. The ticket itself includes the same session key.

C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator.

If mutual authentication is required, the server can reply as shown in message (6) of Table 11.1. The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assures C that this is not a replay of an old reply.

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.

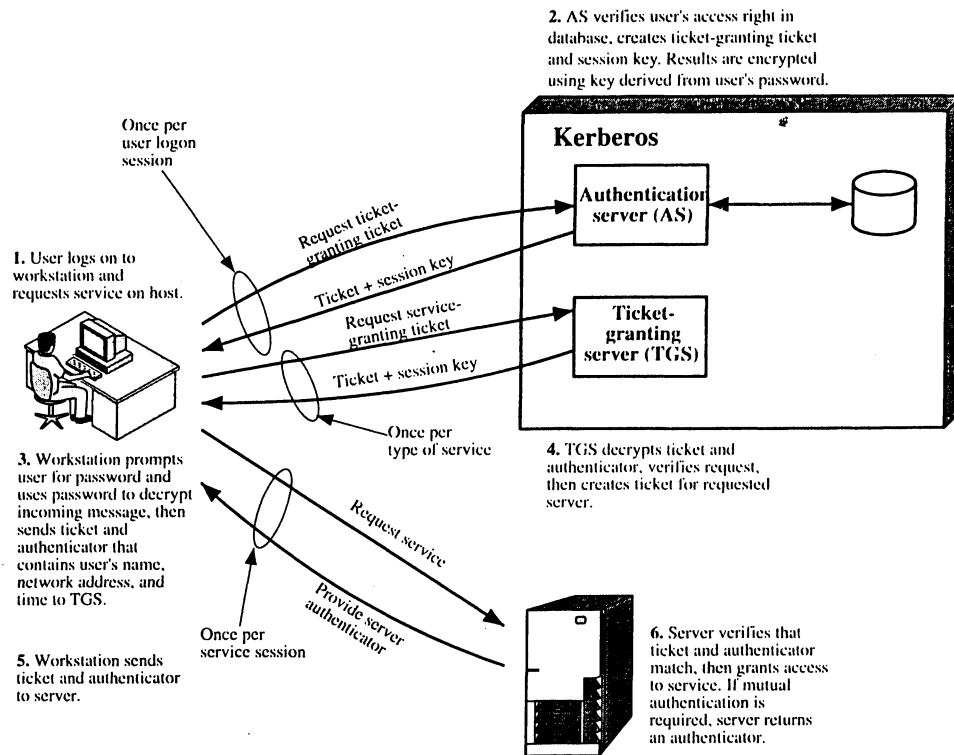
Table 11.2 summarizes the justification for each of the elements in the Kerberos protocol, and Figure 11.1 provides a simplified overview of the action.

**Table 11.2** Rationale for the Elements of the Kerberos Version 4 Protocol

(a) Authentication Service Exchange	
Message (1)	Client requests ticket-granting ticket
$ID_C$ :	Tells AS identity of user from this client
$ID_{tgs}$ :	Tells AS that user requests access to TGS
$TS_1$ :	Allows AS to verify that client's clock is synchronized with that of AS
Message (2)	AS returns ticket-granting ticket
$E_{K_C}$ :	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2)
$K_{c,tgs}$ :	Copy of session key accessible to client; created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key
$ID_{tgs}$ :	Confirms that this ticket is for the TGS
$TS_2$ :	Informs client of time this ticket was issued
Lifetime <sub>2</sub> :	Informs client of the lifetime of this ticket
Ticket <sub>tgs</sub> :	Ticket to be used by client to access TGS

*continued*

(b) Ticket-Granting Service Exchange	
Message (3)	Client requests service-granting ticket
$ID_V$ :	Tells TGS that user requests access to server V
$Ticket_{TGS}$ :	Assures TGS that this user has been authenticated by AS
$Authenticator_c$ :	Generated by client to validate ticket
Message (4)	TGS returns service-granting ticket
$E_{K_{c,TGS}}$ :	Key shared only by C and TGS; protects contents of message (4)
$K_{c,TGS}$ :	Copy of session key accessible to client; created by TGS to permit secure exchange between client and server without requiring them to share a permanent key
$ID_V$ :	Confirms that this ticket is for server V
$TS_4$ :	Informs client of time this ticket was issued
$Ticket_V$ :	Ticket to be used by client to access server V
$Ticket_{TGS}$	Reusable so that user does not have to reenter password
$E_{K_{TGS}}$ :	Ticket is encrypted with key known only to AS and TGS, to prevent tampering
$K_{c,TGS}$ :	Copy of session key accessible to TGS; used to decrypt authenticator, thereby authenticating ticket
$ID_c$ :	Indicates the rightful owner of this ticket
$AD_c$ :	Prevents use of ticket from workstation other than one that initially requested the ticket
$ID_{TGS}$ :	Assures server that it has decrypted ticket properly
$TS_2$ :	Informs TGS of time this ticket was issued
$Lifetime_2$ :	Prevents replay after ticket has expired
$Authenticator_c$ :	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay
$E_{K_{c,TGS}}$ :	Authenticator is encrypted with key known only to client and TGS, to prevent tampering
$ID_c$ :	Must match ID in ticket to authenticate ticket
$AD_c$ :	Must match address in ticket to authenticate ticket
$TS_2$ :	Informs TGS of time this authenticator was generated
(c) Client/Server Authentication Exchange	
Message (5)	Client requests service
$Ticket_V$ :	Assures server that this user has been authenticated by AS
$Authenticator_c$ :	Generated by client to validate ticket
Message (6)	Optional authentication of server to client
$E_{K_{c,V}}$ :	Assures C that this message is from V
$TS_5 + 1$ :	Assures C that this is not a replay of an old reply
$Ticket_V$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server
$E_{K_V}$ :	Ticket is encrypted with key known only to TGS and server, to prevent tampering
$K_{c,V}$ :	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket
$ID_c$ :	Indicates the rightful owner of this ticket
$AD_c$ :	Prevents use of ticket from workstation other than one that initially requested the ticket
$ID_V$ :	Assures server that it has decrypted ticket properly
$TS_4$ :	Informs server of time this ticket was issued
$Lifetime_4$ :	Prevents replay after ticket has expired
$Authenticator_c$ :	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay
$E_{K_{c,V}}$ :	Authenticator is encrypted with key known only to client and server, to prevent tampering
$ID_c$ :	Must match ID in ticket to authenticate ticket
$AD_c$ :	Must match address in ticket to authenticate ticket
$TS_5$ :	Informs server of time this authenticator was generated



**Figure 11.1** Overview of Kerberos.

### Kerberos Realms and Multiple Kerber

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID (UID) and hashed password of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a **realm**. Networks of clients and servers under different administrative organizations typically constitute different realms. That is, it generally is not practical, or does not conform to administrative policy, to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such inter-realm authentication. For two realms to support inter-realm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

With these ground rules in place, we can describe the mechanism as follows (Figure 11.2): A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm). The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS.

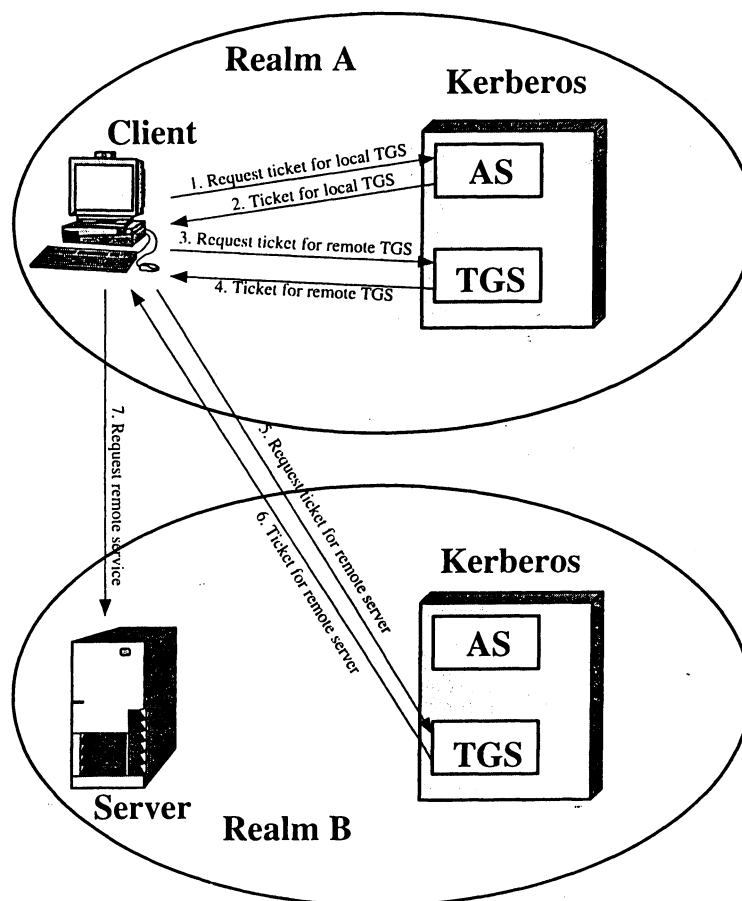


Figure 11.2 Request for Service in Another Realm.

The details of the exchanges illustrated in Figure 11.2 are as follows (compare Table 11.1):

- (1)  $C \rightarrow AS:$   $ID_c || ID_{tgs} || TS_1$
- (2)  $AS \rightarrow C:$   $E_{K_c}[K_{c,tgs} || ID_{tgs} || TS_2 || Lifetime_2 || Ticket_{tgs}^w]$
- (3)  $C \rightarrow TGS:$   $ID_{tgsrem} || Ticket_{tgs} || Authenticator_c$
- (4)  $TGS \rightarrow C:$   $E_{K_{c,tgs}}[K_{c,tgsrem} || ID_{tgsrem} || TS_4 || Ticket_{tgsrem}]$
- (5)  $C \rightarrow TGS_{rem}:$   $ID_{vrem} || Ticket_{tgsrem} || Authenticator_c$
- (6)  $TGS \rightarrow C:$   $E_{K_{c,tgsrem}}[K_{c,vrem} || ID_{vrem} || TS_b || Ticket_{vrem}]$
- (7)  $C \rightarrow V_{rem}:$   $Ticket_{vrem} || Authenticator_c$

The ticket presented to the remote server ( $V_{rem}$ ) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are  $N$  realms, then there must be  $N(N-1)/2$  secure key exchanges so that each Kerberos realm can interoperate with all other Kerberos realms.

## Kerberos Version 5

Version 5 of Kerberos is specified in RFC 1510 and provides a number of improvements over version 4 [KOHL94]. To begin, we provide an overview of the changes from version 4 to version 5 and then look at the version 5 protocol.

### Differences Between Versions 4 and 5

Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies. Let us briefly summarize the improvements in each area.<sup>5</sup>

Version 4 of Kerberos was developed for use within the Project Athena environment and, accordingly, did not fully address the need to be of general purpose. This led to the following **environmental shortcomings**:

- 1. Encryption system dependence:** Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES are thus of concern. In version 5, ciphertext is tagged with an encryption type identifier so that any encryption technique may be used. Encryption keys are tagged with a type and a length, allowing the same key to be used in different algorithms and allowing the specification of different variations on a given algorithm.
- 2. Internet protocol dependence:** Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.
- 3. Message byte ordering:** In version 4, the sender of a message employs a byte

<sup>5</sup>The following discussion follows the presentation in [KOHL94].

ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This technique works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.

4. **Ticket lifetime:** Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is  $2^8 \times 5 = 1280$  minutes, or a little over 21 hours. This may be inadequate for some applications (e.g., a long-running simulation that requires valid Kerberos credentials throughout execution). In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.
5. **Authentication forwarding:** Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.
6. **Inter-realm authentication:** In version 4, interoperability among  $N$  realms requires on the order of  $N^2$  Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

Apart from these environmental limitations, there are **technical deficiencies** in the version 4 protocol itself. Most of these deficiencies were documented in [BELL90], and version 5 attempts to address these. The deficiencies are the following:

1. **Double encryption:** Note in Table 11.1 (messages 2 and 4) that tickets provided to clients are encrypted twice, once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.
2. **PCBC encryption:** Encryption in version 4 makes use of a nonstandard mode of DES known as propagating cipher block chaining (PCBC).<sup>6</sup> It has been demonstrated that this mode is vulnerable to an attack involving the interchange of ciphertext blocks [KOHL89]. PCBC was intended to provide an integrity check as part of the encryption operation. Version 5 provides explicit integrity mechanisms, allowing the standard CBC mode to be used for encryption.
3. **Session keys:** Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket. In addition, the session key may subsequently be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a

<sup>6</sup>This is described in Appendix 11A.



subsession key, which is to be used only for that one connection. A new access by the client would result in the use of a new subsession key.

4. **Password attacks:** Both versions are vulnerable to a password attack. The message from the AS to the client includes material encrypted with a key based on the client's password.<sup>7</sup> An opponent can capture this message and attempt to decrypt it by trying various passwords. If the result of a test decryption is of the proper form, then the opponent has discovered the client's password and may subsequently use it to gain authentication credentials from Kerberos. This is the same type of password attack described in Chapter 15, with the same kinds of countermeasures being applicable. Version 5 does provide a mechanism known as preauthentication, which should make password attacks more difficult, but it does not prevent them.

### The Version 5 Authentication Dialogue

Table 11.3 summarizes the basic version 5 dialogue. This is best explained by comparison with version 4 (Table 11.1).

First, consider the **authentication service exchange**. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:

- **Realm:** Indicates realm of user
- **Options:** Used to request that certain flags be set in the returned ticket, as explained below
- **Times:** Used by the client to request the following time settings in the ticket:

**Table 11.3** Summary of Kerberos Version 5 Message Exchanges

<b>(a) Authentication Service Exchange: to obtain ticket-granting ticket</b>	
(1) C → AS:	Options    ID <sub>c</sub>    Realm <sub>c</sub>    ID <sub>tgs</sub>    Times    Nonce <sub>1</sub>
(2) AS → C:	Realm <sub>c</sub>    ID <sub>c</sub>    Ticket <sub>tgs</sub>    E <sub>K<sub>c</sub></sub> [K <sub>c,tgs</sub>    Times    Nonce <sub>1</sub>    Realm <sub>tgs</sub>    ID <sub>tgs</sub> ] Ticket <sub>tgs</sub> = E <sub>K<sub>tgs</sub></sub> [Flags    K <sub>c,tgs</sub>    Realm <sub>c</sub>    ID <sub>c</sub>    AD <sub>c</sub>    Times]
<b>(b) Ticket-Granting Service Exchange: to obtain service-granting ticket</b>	
(3) C → TGS:	Options    ID <sub>v</sub>    Times    Nonce <sub>2</sub>    Ticket <sub>tgs</sub>    Authenticator <sub>c</sub>
(4) TGS → C:	Realm <sub>c</sub>    ID <sub>c</sub>    Ticket <sub>v</sub>    E <sub>K<sub>c,tgs</sub></sub> [K <sub>c,v</sub>    Times    Nonce <sub>2</sub>    Realm <sub>v</sub>    ID <sub>v</sub> ] Ticket <sub>tgs</sub> = E <sub>K<sub>tgs</sub></sub> [Flags    K <sub>c,tgs</sub>    Realm <sub>c</sub>    ID <sub>c</sub>    AD <sub>c</sub>    Times] Ticket <sub>v</sub> = E <sub>K<sub>v</sub></sub> [Flags    K <sub>c,v</sub>    Realm <sub>c</sub>    ID <sub>c</sub>    AD <sub>c</sub>    Times] Authenticator <sub>c</sub> = E <sub>K<sub>c,tgs</sub></sub> [ID <sub>c</sub>    Realm <sub>c</sub>    TS <sub>1</sub> ]
<b>(c) Client/Server Authentication Exchange: to obtain service</b>	
(5) C → TGS:	Options    Ticket <sub>v</sub>    Authenticator <sub>c</sub>
(6) TGS → C:	E <sub>K<sub>c,v</sub></sub> [TS <sub>2</sub>    Subkey    Seq#] Ticket <sub>v</sub> = E <sub>K<sub>v</sub></sub> [Flags    K <sub>c,v</sub>    Realm <sub>c</sub>    ID <sub>c</sub>    AD <sub>c</sub>    Times] Authenticator <sub>c</sub> = E <sub>K<sub>c,v</sub></sub> [ID <sub>c</sub>    Realm <sub>c</sub>    TS <sub>2</sub>    Subkey    Seq#]

<sup>7</sup>Appendix 11A describes the mapping of passwords to encryption keys.

- from: the desired start time for the requested ticket
  - till: the requested expiration time for the requested ticket
  - rtime: requested renew-till time
- **Nonce:** A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent.

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the **ticket-granting service exchange** for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2), returning a ticket plus information needed by the client, the latter encrypted with the session key now shared by the client and the TGS.

Finally, for the **client/server authentication exchange**, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields as follows:

- **Subkey:** The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ( $K_{c,v}$ ) is used.
- **Sequence number:** An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5 because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys. The subkey field, if present, overrides the subkey field, if present, in message (5). The optional sequence number field specifies the starting sequence number to be used by the client.

**Table 11.4** Kerberos Version 5 Flags

INITIAL	This ticket was issued using the AS protocol, and not issued based on a ticket-granting ticket.
PRE-AUTHENT	During initial authentication, the client was authenticated by the KDC before a ticket was issued.
HW-AUTHENT	The protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.
RENEWABLE	Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
MAY-POSTDATE	Tells TGS that a postdated ticket may be issued based on this ticket-granting ticket.
POSTDATED	Indicates that this ticket has been postdated; the end server can check the authtime field to see when the original authentication occurred.
INVALID	This ticket is invalid and must be validated by the KDC before use.
PROXIABLE	Tells TGS that a new service-granting ticket with a different network address may be issued based on the presented ticket.
PROXY	Indicates that this ticket is a proxy.
FORWARDABLE	Tells TGS that a new ticket-granting ticket with a different network address may be issued based on this ticket-granting ticket.
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.

### Ticket Flags

The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4. Table 11.4 summarizes the flags that may be included in a ticket.

The INITIAL flag indicates that this ticket was issued by the AS, not by the TGS. When a client requests a service-granting ticket from the TGS, it presents a ticket-granting ticket obtained from the AS. In version 4, this was the only way to obtain a service-granting ticket. Version 5 provides the additional capability that the client can get a service-granting ticket directly from the AS. The utility of this is as follows: A server, such as a password-changing server, may wish to know that the client's password was recently tested.

The PRE-AUTHENT flag, if set, indicates that when the AS received the initial request (message 1), it authenticated the client before issuing a ticket. The exact form of this preauthentication is left unspecified. As an example, the MIT implementation of version 5 has encrypted timestamp preauthentication, enabled by default. When a user wants to get a ticket, it has to send to the AS a preauthentication block containing a random confounder, a version number, and a timestamp, encrypted in the client's password-based key. The AS decrypts the block and will not send a ticket-granting ticket back unless the timestamp in the preauthentication block is within the allowable time skew (time interval to account for clock drift and network delays). Another possibility is the use of a smart card that generates continually changing passwords that are included in the preauthenticated messages. The passwords generated by the card can be based on a user's password but be trans-

formed by the card so that, in effect, arbitrary passwords are used. This prevents an attack based on easily guessed passwords. If a smart card or similar device was used, this is indicated by the HW-AUTHENT flag.

When a ticket has a long lifetime, there is the potential for it to be stolen and used by an opponent for a considerable period. If a short lifetime is used to lessen the threat, then overhead is involved in acquiring new tickets. In the case of a ticket-granting ticket, the client would either have to store the user's secret key, which is clearly risky, or repeatedly ask the user for a password. A compromise scheme is the use of renewable tickets. A ticket with the RENEWABLE flag set includes two expiration times: one for this specific ticket and one that is the latest permissible value for an expiration time. A client can have the ticket renewed by presenting it to the TGS with a requested new expiration time. If the new time is within the limit of the latest permissible value, the TGS can issue a new ticket with a new session time and a later specific expiration time. The advantage of this mechanism is that the TGS may refuse to renew a ticket reported as stolen.

A client may request that the AS provide a ticket-granting ticket with the MAY-POSTDATE flag set. The client can then use this ticket to request a ticket that is flagged as POSTDATED and INVALID from the TGS. Subsequently, the client may submit the postdated ticket for validation. This scheme can be useful for running a long batch job on a server that requires a ticket periodically. The client can obtain a number of tickets for this session at once, with spread-out time values. All but the first ticket are initially invalid. When the execution reaches a point in time when a new ticket is required, the client can get the appropriate ticket validated. With this approach, the client does not have to repeatedly use its ticket-granting ticket to obtain a service-granting ticket.

In version 5 it is possible for a server to act as a proxy on behalf of a client, in effect adopting the credentials and privileges of the client to request a service from another server. If a client wishes to use this mechanism, it requests a ticket-granting ticket with the PROXIABLE flag set. When this ticket is presented to the TGS, the TGS is permitted to issue a service-granting ticket with a different network address; this latter ticket will have its PROXY flag set. An application receiving such a ticket may accept it or require additional authentication to provide an audit trail.<sup>8</sup>

The proxy concept is a limited case of the more powerful forwarding procedure. If a ticket is set with the FORWARDABLE flag, a TGS can issue to the requestor a ticket-granting ticket with a different network address and the FORWARDED flag set. This ticket can then be presented to a remote TGS. This capability allows a client to gain access to a server on another realm without requiring that each Kerberos maintain a secret key with Kerberos servers in every other realm. For example, realms could be structured hierarchically. Then a client could walk up the tree to a common node and then back down to reach a target realm. Each step of the walk would involve forwarding a ticket-granting ticket to the next TGS in the path.

---

<sup>8</sup>For a discussion of some of the possible uses of the proxy capability, see [NEUM93b].

## 11.2 X.509 AUTHENTICATION SERVICE

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users.

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates of the type discussed in Chapter 6. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME (Chapter 12), IP Security (Chapter 13), and SSL/TLS and SET (Chapter 14).

X.509 was initially issued in 1988. The standard was subsequently revised to address some of the security concerns documented in [IANS90] and [MITC90]; a revised recommendation was issued in 1993. A third version was drafted in 1995.

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function. Again, the standard does not dictate a specific hash algorithm. The 1988 recommendation included the description of a recommended hash algorithm; this algorithm has since been shown to be insecure and has been dropped from the 1993 recommendation.

### Certificates

The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

Figure 11.3a shows the general format of a certificate, which includes the following elements:

- **Version:** Differentiates among successive versions of the certificate format; the default is version 1. If the Initiator Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.
- **Serial number:** An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.
- **Signature algorithm identifier:** The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated

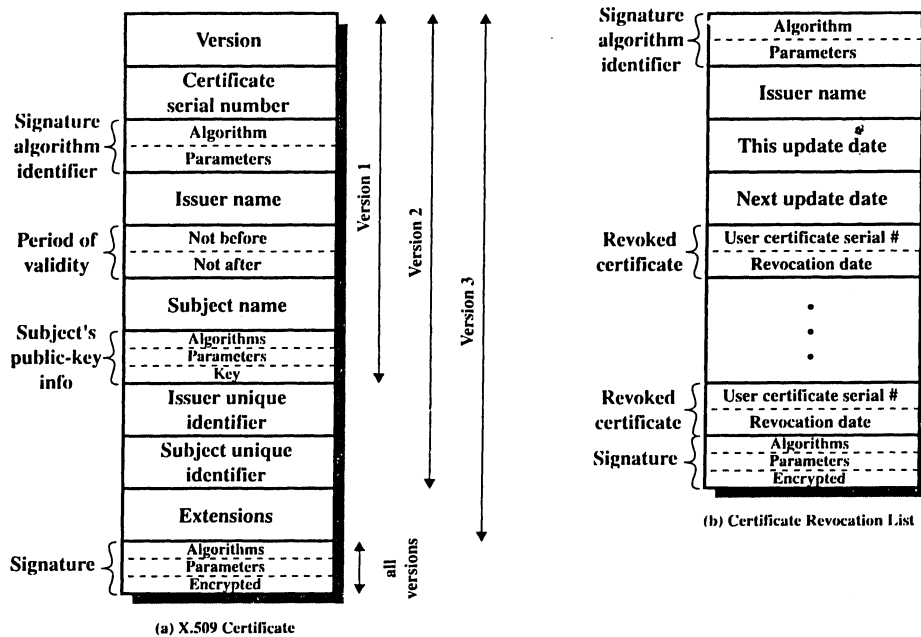


Figure 11.3 X.509 Formats.

in the Signature field at the end of the certificate, this field has little, if any, utility.

- **Issuer name:** X.500 name of the CA that created and signed this certificate.
- **Period of validity:** Consists of two dates: the first and last on which the certificate is valid.
- **Subject name:** The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.
- **Subject's public-key information:** The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
- **Issuer unique identifier:** An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.
- **Subject unique identifier:** An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
- **Extensions:** A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.
- **Signature:** Covers all of the other fields of the certificate; it contains the hash code of the other fields, encrypted with the CA's private key. This field includes the signature algorithm identifier.

The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. These fields are rarely used.

The standard uses the following notation to define a certificate:

$$CA\langle\langle A \rangle\rangle = CA \{V, SN, AI, CA, T_A, A, Ap\}$$

where

$Y\langle\langle X \rangle\rangle$  = the certificate of user X issued by certification authority Y  
 $Y \{I\}$  = the signing of I by Y. It consists of I with an encrypted hash code appended.

The CA signs the certificate with its secret key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid. This is the typical digital signature approach illustrated in Figure 8.5c.

### Obtaining a User's Certificate

User certificates generated by a CA have the following characteristics:

- Any user with access to the public key of the CA can recover the user public key that was certified.
- No party other than the certification authority can modify the certificate without this being detected.

Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users. In addition, a user can transmit his or her certificate directly to other users. In either case, once B is in possession of A's certificate, B has confidence that messages it encrypts with A's public key will be secure from eavesdropping and that messages signed with A's private key are unforgeable.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority  $X_1$  and B has obtained a certificate from CA  $X_2$ . If A does not securely know the public key of  $X_2$ , then B's certificate, issued by  $X_2$ , is useless to A. A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key:

1. A obtains, from the directory, the certificate of  $X_2$  signed by  $X_1$ . Because A securely knows  $X_1$ 's public key, A can obtain  $X_2$ 's public key from its certificate and verify it by means of  $X_1$ 's signature on the certificate.

2. A then goes back to the directory and obtains the certificate of B signed by  $X_2$ . Because A now has a trusted copy of  $X_2$ 's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

$$X_1 \langle\langle X_2 \rangle\rangle X_2 \langle\langle B \rangle\rangle$$

In the same fashion, B can obtain A's public key with the reverse chain:

$$X_2 \langle\langle X_1 \rangle\rangle X_1 \langle\langle A \rangle\rangle$$

This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with  $N$  elements would be expressed as

$$X_1 \langle\langle X_2 \rangle\rangle X_2 \langle\langle X_3 \rangle\rangle \dots X_N \langle\langle B \rangle\rangle$$

In this case, each pair of CAs in the chain  $(X_i, X_{i+1})$  must have created certificates for each other.

All these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

Figure 11.4, taken from X.509, is an example of such a hierarchy. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

- **Forward certificates:** Certificates of  $X$  generated by other CAs
- **Reverse certificates:** Certificates generated by  $X$  that are the certificates of other CAs

In this example, user A can acquire the following certificates from the directory to establish a certification path to B:

$$X \langle\langle W \rangle\rangle W \langle\langle V \rangle\rangle V \langle\langle Y \rangle\rangle Y \langle\langle Z \rangle\rangle Z \langle\langle B \rangle\rangle$$

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

$$Z \langle\langle Y \rangle\rangle Y \langle\langle V \rangle\rangle V \langle\langle W \rangle\rangle W \langle\langle X \rangle\rangle X \langle\langle A \rangle\rangle$$

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.



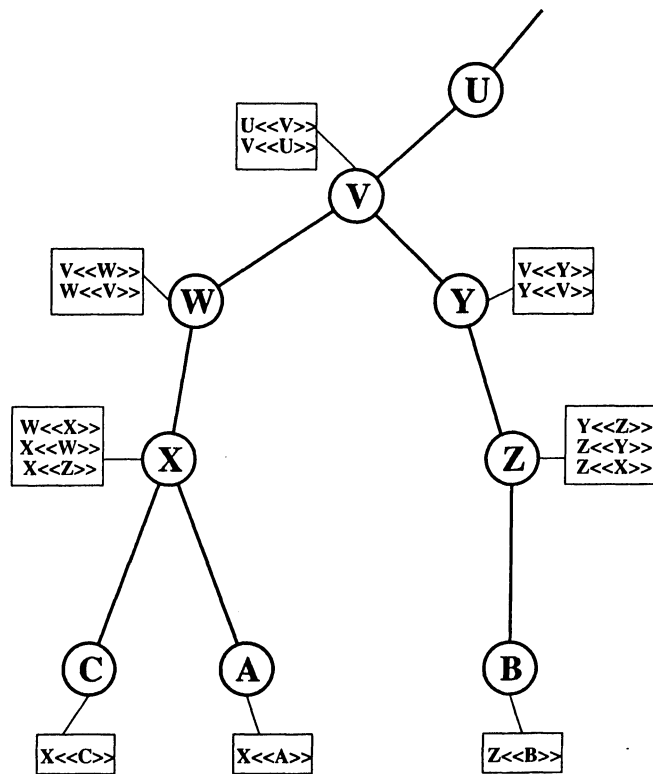


Figure 11.4 X.509 Hierarchy: A Hypothetical Example.

### Revocation of Certificates

Recall from Figure 11.3 that each certificate includes a period of validity, much like a credit card. Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

1. The user's secret key is assumed to be compromised.
2. The user is no longer certified by this CA.
3. The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory.

Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (Figure 11.3b) the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

When a user receives a certificate in a message, the user must determine whether the certificate has been revoked. The user could check the directory each time a certificate is received. To avoid the delays (and possible costs) associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certificates.

### Authentication Procedures

X.509 also includes three alternative authentication procedures that are intended for use across a variety of applications. All these procedures make use of public-key signatures. It is assumed that the two parties know each other's public key, either by obtaining each other's certificates from the directory or because the certificate is included in the initial message from each side.

Figure 11.5 illustrates the three procedures.

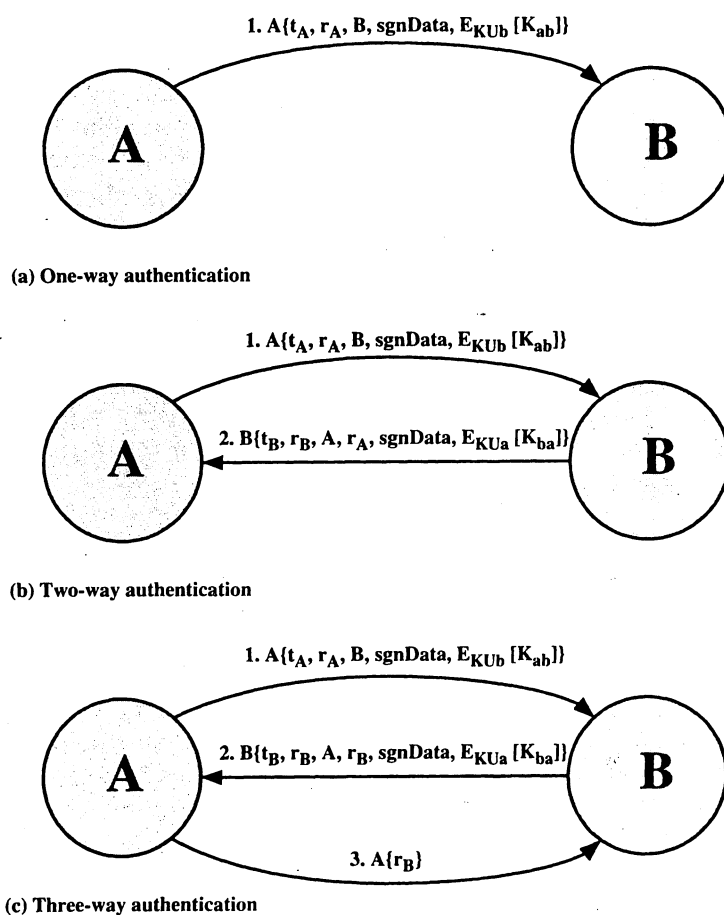


Figure 11.5 X.509 Strong Authentication Procedures.

### One-Way Authentication

One-way authentication involves a single transfer of information from one user (A) to another (B) and establishes the following:

1. The identity of A and that the message was generated by A
2. That the message was intended for B
3. The integrity and originality (it has not been sent multiple times) of the message

Note that only the identity of the initiating entity is verified in this process, not that of the responding entity.

At a minimum, the message includes a timestamp  $t_A$ , a nonce  $r_A$ , and the identity of B and is signed with A's public key. The timestamp consists of an optional generation time and an expiration time. This prevents delayed delivery of messages. The nonce can be used to detect replay attacks. The nonce value must be unique within the expiration time of the message. Thus, B can store the nonce until it expires and reject any new messages with the same nonce.

For pure authentication, the message is used simply to present credentials to B. The message may also include information to be conveyed. This information, *sgn-Data*, is included within the scope of the signature, guaranteeing its authenticity and integrity. The message may also be used to convey a session key to B, encrypted with B's public key.

### Two-Way Authentication

In addition to the three elements just listed, two-way authentication establishes the following elements:

4. The identity of B and that the reply message was generated by B
5. That the message was intended for A
6. The integrity and originality of the reply

Two-way authentication thus permits both parties in a communication to verify the identity of the other.

The reply message includes the nonce from A, to validate the reply. It also includes a timestamp and nonce generated by B. As before, the message may include signed additional information and a session key encrypted with A's public key.

### Three-Way Authentication

In three-way authentication, a final message from A to B is included, which contains a signed copy of the nonce  $r_B$ . The intent of this design is that timestamps need not be checked: Because both nonces are echoed back by the other side, each side can check the returned nonce to detect replay attacks. This approach is needed when synchronized clocks are not available.

## X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed. [FORD95] lists the following requirements not satisfied by version 2:

1. The Subject field is inadequate to convey the identity of a key owner to a public-key user. X.509 names may be relatively short and lacking in obvious identification details that may be needed by the user.
2. The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet-related identification.
3. There is a need to indicate security policy information. This enables a security application or function, such as IPSec, to relate an X.509 certificate to a given policy.
4. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.
5. It is important to be able to identify separately different keys used by the same owner at different times. This feature supports key life cycle management—in particular, the ability to update key pairs for users and CAs on a regular basis or under exceptional circumstances.

Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. Thus, version 3 includes a number of optional extensions that may be added to the version 2 format. Each extension consists of an extension identifier, a criticality indicator, and an extension value. The criticality indicator indicates whether an extension can be safely ignored. If the indicator has a value of TRUE and an implementation does not recognize the extension, it must treat the certificate as invalid.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

#### **Key and Policy Information**

These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy. A certificate policy is a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes the following:

- **Authority key identifier:** Identifies the public key to be used to verify the signature on this certificate or CRL. Enables distinct keys of the same CA to be differentiated. One use of this field is to handle CA key pair updating.
- **Subject key identifier:** Identifies the public key being certified. Useful for subject key pair updating. Also, a subject may have multiple key pairs and, correspondingly, different certificates for different purposes (e.g., digital signature and encryption key agreement).
- **Key usage:** Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used. May indicate one or more of the following: digital signature, nonrepudiation, key encryption, data encryption, key agreement, CA signature verification on certificates, CA signature verification on CRLs.

- **Private-key usage period:** Indicates the period of use of the private key corresponding to the public key. Typically, the private key is used over a different period from the validity of the public key. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.
- **Certificate policies:** Certificates may be used in environments where multiple policies apply. This extension lists policies that the certificate is recognized as supporting, together with optional qualifier information.
- **Policy mappings:** Used only in certificates for CAs issued by other CAs. Policy mappings allow an issuing CA to indicate that one or more of that issuer's policies can be considered equivalent to another policy used in the subject CA's domain.

### Certificate Subject and Issuer Attributes

These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject, to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required.

The extension fields in this area include the following:

- **Subject alternative name:** Contains one or more alternative names, using any of a variety of forms. This field is important for supporting certain applications, such as electronic mail, EDI, and IPsec, which may employ their own name forms.
- **Issuer alternative name:** Contains one or more alternative names, using any of a variety of forms.
- **Subject directory attributes:** Conveys any desired X.500 directory attribute values for the subject of this certificate.

### Certification Path Constraints

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs. The constraints may restrict the types of certificates that can be issued by the subject CA or that may occur subsequently in a certification chain.

The extension fields in this area include the following:

- **Basic constraints:** Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.
- **Name constraints:** Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.
- **Policy constraints:** Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

### 11.3 RECOMMENDED READING

A painless way to get a grasp of Kerberos concepts is found in [BRAY88]. One of the best treatments of Kerberos is [KOHL94].

**BRYA88** Bryant, W. *Designing an Authentication System: A Dialogue in Four Scenes*. Project Athena document, February 1988. Available at <http://web.mit.edu/kerberos/www/dialogue.html>.

**KOHL94** Kohl, J.; Neuman, B.; and Ts'o, T. "The Evolution of the Kerberos Authentication Service." In Brazier, F., and Johansen, D. *Distributed Open Systems*. Los Alamitos, CA: IEEE Computer Society Press, 1994. Available at <http://web.mit.edu/kerberos/www/papers.html>.

Recommended Web Sites:

- **MIT Kerberos Site:** Information about Kerberos, including the FAQ, papers and documents, and pointers to commercial product sites.
- **USC/ISI Kerberos Page:** Another good source of Kerberos material.
- **Public-Key Infrastructure Working Group:** IETF group developing standards based on X.509v3.
- **Verisign:** A leading commercial vendor of X.09-related products; white papers and other worthwhile material at this site.

### 11.4 PROBLEMS

- 11.1** Show that a random error in one block of ciphertext is propagated to all subsequent blocks of plaintext in PCBC mode (Figure 11.7).
- 11.2** Suppose that, in PCBC mode, blocks  $C_i$  and  $C_{i+1}$  are interchanged during transmission. Show that this affects only the decrypted blocks  $P_i$  and  $P_{i+1}$  but not subsequent blocks.
- 11.3** The original three-way authentication procedure for X.509 illustrated in Figure 11.5c contains a security flaw. The essence of the protocol is as follows:

$$\begin{aligned} A \rightarrow B: & \quad A \{t_A, r_A, B\} \\ B \rightarrow A: & \quad B \{t_B, r_B, A, r_A\} \\ A \rightarrow B: & \quad A \{r_B\} \end{aligned}$$

The text of X.509 states that checking timestamps  $t_A$  and  $t_B$  is optional for three-way authentication. But consider the following example. Suppose A and B have used the preceding protocol on some previous occasion, and that opponent C has intercepted the preceding three messages. In addition, suppose that timestamps are not used and are all set to 0. Finally, suppose C wishes to impersonate A to B. C initially sends the first captured message to B:

$$C \rightarrow B: \quad A \{0, r_A, B\}$$

B responds, thinking it is talking to A but is actually talking to C:

$$B \rightarrow C: \quad B \{0, r'_B, A, r_A\}$$

C, meanwhile, causes A to initiate authentication with C, by some means. As a result, A sends C the following:

$$A \rightarrow C: \quad A \{0, r'_A, B\}$$

C responds to A, using the same nonce provided to C by B.

$C \rightarrow A: \quad C \{0, r'_B, A, r'_A\}$

A responds with

$A \rightarrow C: \quad A \{r'_B\}$

This is exactly what C needs to convince B that it is talking to A, so C now repeats the incoming message back out to B.

$C \rightarrow B: \quad A \{r'_B\}$

So B will believe it is talking to A whereas it is actually talking to C. Suggest a simple solution to this problem that does not involve the use of timestamps.

- 11.4** The 1988 version of X.509 lists properties that RSA keys must satisfy to be secure, given current knowledge about the difficulty of factoring large numbers. The discussion concludes with a constraint on the public exponent and the modulus  $n$ :

It must be ensured that  $e > \log_2(n)$  to prevent attack by taking the  $e$ th root mod  $n$  to disclose the plaintext.

Although the constraint is correct, the reason given for requiring it is incorrect. What is wrong with the reason given, and what is the correct reason?

## APPENDIX 11A: KERBEROS ENCRYPTION TECHNIQUES

Kerberos includes an encryption library that supports various encryption-related operations.

### Password-to-Key Transformation

In Kerberos, passwords are limited to the use of the characters that can be represented in a 7-bit ASCII format. This password, of arbitrary length, is converted into an encryption key that is stored in the Kerberos database. Figure 11.6 illustrates the procedure.

First, the character string,  $s$ , is packed into a bit string,  $b$ , such that the first character is stored in the first 7 bits, the second character in the second 7 bits, and so on. This can be expressed as

$b[0] \quad = \text{bit 0 of } s[0]$

...

$b[6] \quad = \text{bit 6 of } s[0]$

$b[7] \quad = \text{bit 0 of } s[1]$

...

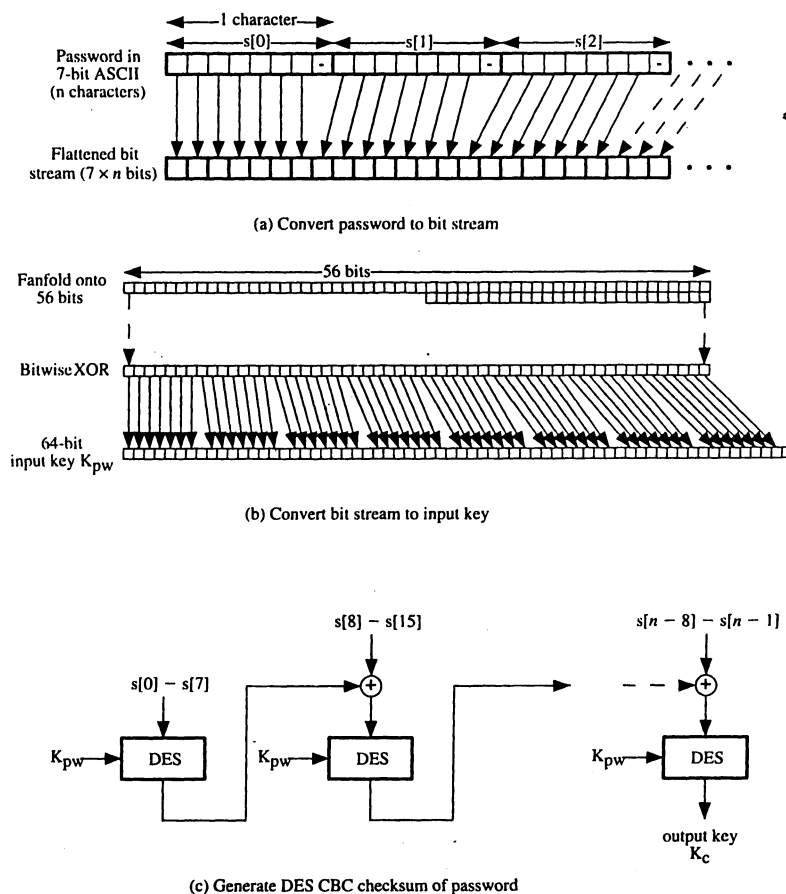
$b[7i + m] = \text{bit } m \text{ of } s[i] \quad 0 \leq m \leq 6$

Next, the bit string is compacted to 56 bits by aligning the bits in “fanfold” fashion and performing a bitwise XOR. For example, if the bit string is of length 59, then

$b[55] = b[55] \oplus b[56]$

$b[54] = b[54] \oplus b[57]$

$b[53] = b[53] \oplus b[58]$



**Figure 11.6** Generation of Encryption Key from Password.

This creates a 56-bit DES key. To conform to the expected 64-bit key format, the string is treated as a sequence of eight 7-bit blocks and is mapped into eight 8-bit blocks to form an input key  $K_{pw}$ .

Finally, the original password is encrypted using the cipher block chaining (CBC) mode of DES with key  $K_{pw}$ . The last 64-bit block returned from this process, known as the CBC checksum, is the output key associated with this password.

The entire algorithm can be viewed as a hash function that maps an arbitrary password into a 64-bit hash code.

### Propagating Cipher Block Chaining Mode

Recall from Chapter 3 that, in the CBC mode of DES, the input to the DES algorithm at each stage consists of the XOR of the current plaintext block and the preceding ciphertext block, with the same key used for each block (Figure 3.12). The advantage of this mode over the electronic codebook (ECB) mode, in which



each plaintext block is independently encrypted, is this: With CBC, the same plaintext block, if repeated, produces different ciphertext blocks.

CBC has the property that if an error occurs in transmission of ciphertext block  $C_i$ , then this error propagates to the recovered plaintext blocks  $P_i$  and  $P_{i+1}$ .

Version 4 of Kerberos uses an extension to CBC, called the propagating CBC (PCBC) mode [MEYE82]. This mode has the property that an error in one ciphertext block is propagated to all subsequent decrypted blocks of the message, rendering each block useless. Thus, data encryption and integrity are combined in one operation.

PCBC is illustrated in Figure 11.7. In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block, the preceding cipher text block, and the preceding plaintext block:

$$C_n = E_K[C_{n-1} \oplus P_{n-1} \oplus P_n]$$

On decryption, each ciphertext block is passed through the decryption algorithm. Then the output is XORed with the preceding ciphertext block and the preceding plaintext block. We can demonstrate that this scheme works, as follows:

$$\begin{aligned} D_K[C_n] &= D_K[E_K[C_{n-1} \oplus P_{n-1} \oplus P_n]] \\ &= C_{n-1} \oplus P_{n-1} \oplus P_n \\ C_{n-1} \oplus P_{n-1} \oplus D_K[C_n] &= P_n \end{aligned}$$

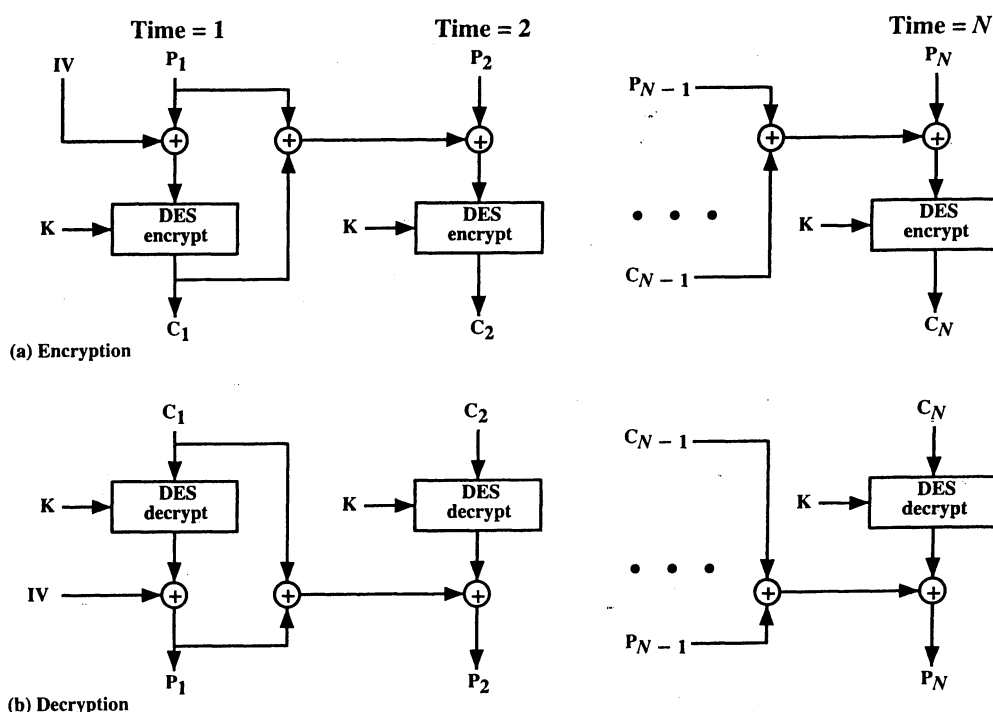


Figure 11.7 Propagating Cipher Block Chaining (PCBC) Mode.



# CHAPTER 12

## ELECTRONIC MAIL SECURITY

*Despite the refusal of VADM Poindexter and LtCol North to appear, the Board's access to other sources of information filled much of this gap. The FBI provided documents taken from the files of the National Security Advisor and relevant NSC staff members, including messages from the PROF system between VADM Poindexter and LtCol North. The PROF messages were conversations by computer, written at the time events occurred and presumed by the writers to be protected from disclosure. In this sense, they provide a first-hand, contemporaneous account of events.*

—The Tower Commission Report  
to President Reagan on the Iran-Contra Affair, 1987

*Bless the man who made it,  
And pray that he ain't dead.  
He could've made a million  
If he'd sold it to the feds,  
But he was hot for freedom;  
He gave it out for free.  
Now every common citizen's got PGP.*

—From the song "P.G.P.,"  
by Leslie Fish

**I**n virtually all distributed environments, electronic mail is the most heavily used network-based application. It is also the only distributed application that is widely used across all architectures and vendor platforms. Users expect to be able to, and do, send mail to others who are connected directly or indirectly to the Internet, regardless of host operating system or communications suite.

With the explosively growing reliance on electronic mail for every conceivable purpose, there grows a demand for authentication and confidential-

ity services. Two schemes stand out as approaches that are likely to enjoy widespread use in the next few years: pretty good privacy (PGP) and S/MIME. Both are examined in this chapter.

## 12.1 PRETTY GOOD PRIVACY

PGP is a remarkable phenomenon. Largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. In essence, Zimmermann has done the following:

1. Selected the best available cryptographic algorithms as building blocks
2. Integrated these algorithms into a general-purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands
3. Made the package and its documentation, including the source code, freely available via the Internet, bulletin boards, and commercial networks such as CompuServe
4. Entered into an agreement with a company (Viacrypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP

PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth:

1. It is available free worldwide in versions that run on a variety of platforms, including DOS/Windows, UNIX, Macintosh, and many more. In addition, the commercial version satisfies users who want a product that comes with vendor support.
2. It is based on algorithms that have survived extensive public review and are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES for conventional encryption; and SHA-1 for hash coding.
3. It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.
4. It was not developed by, nor is it controlled by, any governmental or standards organization. For those with an instinctive distrust of "the establishment," this makes PGP attractive.

We begin with an overall look at the operation of PGP. Next, we examine how cryptographic keys are created and stored. Then we address the vital issue of public key management.

## Notation

Most of the notation used in this chapter has been used before, but a few terms are new. It is perhaps best to summarize those at the beginning. The following symbols are used:

$K_s$	=	session key used in conventional encryption scheme
$KR_a$	=	private key of user A, used in public-key encryption scheme
$KU_a$	=	public key of user A, used in public-key encryption scheme
EP	=	public-key encryption
DP	=	public-key decryption
EC	=	conventional encryption
DC	=	conventional decryption
H	=	hash function
	=	concatenation
Z	=	compression using ZIP algorithm
R64	=	conversion to radix 64 ASCII format

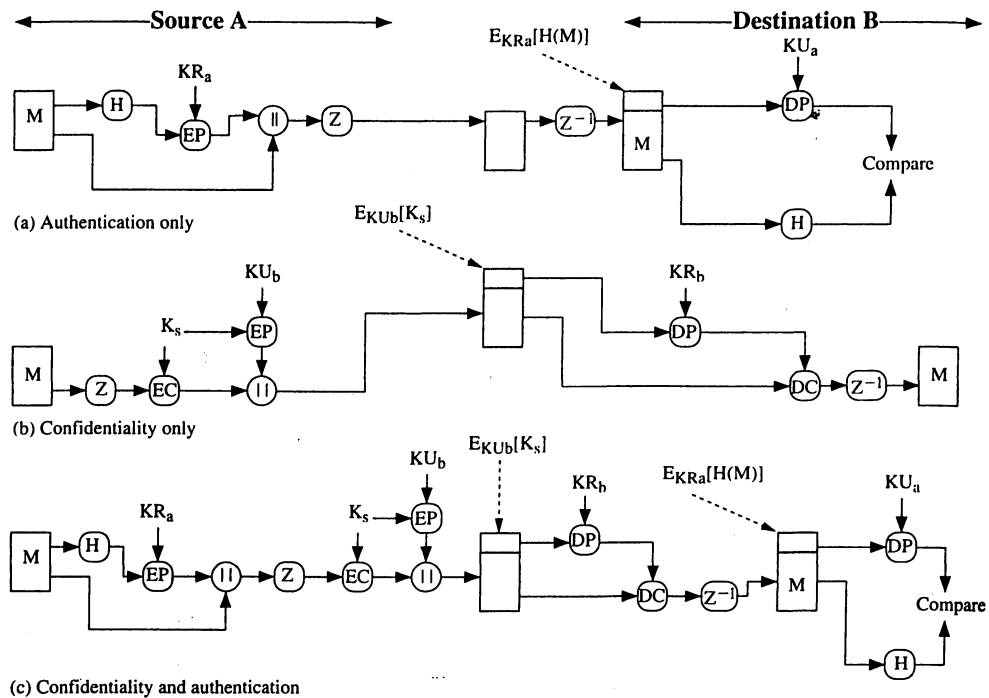
The PGP documentation often uses the term *secret key* to refer to a key paired with a public key in a public-key encryption scheme. As was mentioned earlier, this practice risks confusion with a secret key used for conventional encryption. Hence, we will use the term *private key* instead.

## Operational Description

The actual operation of PGP, as opposed to the management of keys, consists of five services: authentication, confidentiality, compression, e-mail compatibility, and segmentation (Table 12.1). We examine each of these in turn.

**Table 12.1** Summary of PGP Services

Function	Algorithms Used	Description
Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key, and included with the message.
Message encryption	CAST or IDEA or Three-Key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie Hellman or RSA with the recipient's public key, and included with the message.
Compression	ZIP	A message may be compressed, for storage or transmission, using ZIP.
Email compatibility	Radix 64-conversion	To provide transparency for e-mail applications, an encrypted message may be converted to an ASCII string using radix-64 conversion.
Segmentation	—	To accommodate maximum message size limitations, PGP performs segmentation and reassembly.



**Figure 12.1** PGP Cryptographic Functions.

### Authentication

Figure 12.1a illustrates the digital signature service provided by PGP. This is the digital signature scheme discussed in Chapter 10 and illustrated in Figure 8.5c. The sequence is as follows:

1. The sender creates a message.
2. SHA-1 is used to generate a 160-bit hash code of the message.
3. The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.
4. The receiver uses RSA with the sender's public key to decrypt and recover the hash code.
5. The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.

The combination of SHA-1 and RSA provides an effective digital signature scheme. Because of the strength of RSA, the recipient is assured that only the possessor of the matching private key can generate the signature. Because of the strength of SHA-1, the recipient is assured that no one else could generate a new message that matches the hash code and, hence, the signature of the original message.

As an alternative, signatures can be generated using DSS/SHA-1.

Although signatures normally are found attached to the message or file that they sign, this is not always the case: Detached signatures are supported. A detached

signature may be stored and transmitted separately from the message it signs. This is useful in several contexts. A user may wish to maintain a separate signature log of all messages sent or received. A detached signature of an executable program can detect subsequent virus infection. Finally, detached signatures can be used when more than one party must sign a document, such as a legal contract.\*Each person's signature is independent and therefore is applied only to the document. Otherwise, signatures would have to be nested, with the second signer signing both the document and the first signature, and so on.

### Confidentiality

Another basic service provided by PGP is confidentiality, which is provided by encrypting messages to be transmitted or to be stored locally as files. In both cases, the conventional encryption algorithm CAST-128 may be used. Alternatively, IDEA or 3DES may be used. The 64-bit cipher feedback (CFB) mode is used.

As always, one must address the problem of key distribution. In PGP, each conventional key is used only once. That is, a new key is generated as a random 128-bit number for each message. Thus, although this is referred to in the documentation as a session key, it is in reality a one-time key. Because it is to be used only once, the session key is bound to the message and transmitted with it. To protect the key, it is encrypted with the receiver's public key. Figure 12.1b illustrates the sequence, which can be described as follows:

1. The sender generates a message and a random 128-bit number to be used as a session key for this message only.
2. The message is encrypted, using CAST-128 (or IDEA or 3DES) with the session key.
3. The session key is encrypted with RSA, using the recipient's public key, and is prepended to the message.
4. The receiver uses RSA with its private key to decrypt and recover the session key.
5. The session key is used to decrypt the message.

As an alternative to the use of RSA for key encryption, PGP provides an option referred to as *Diffie-Hellman*. As was explained in Chapter 6, Diffie-Hellman is a key exchange algorithm. In fact, PGP uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal (see Problem 6.19).

Several observations may be made. First, to reduce encryption time the combination of conventional and public-key encryption is used in preference to simply using RSA or ElGamal to encrypt the message directly: CAST-128 and the other conventional algorithms are substantially faster than RSA or ElGamal. Second, the use of the public-key algorithm solves the session key distribution problem, because only the recipient is able to recover the session key that is bound to the message. Note that we do not need a session key exchange protocol of the type discussed in Chapter 6, because we are not beginning an ongoing session. Rather, each message is a one-time independent event with its own key. Furthermore, given the store-and-forward nature of electronic mail, the use of handshaking to assure that both sides have the same session key is not practical. Finally, the use of one-time conventional

keys strengthens what is already a strong conventional encryption approach. Only a small amount of plaintext is encrypted with each key, and there is no relationship among the keys. Thus, to the extent that the public-key algorithm is secure, the entire scheme is secure. To this end, PGP provides the user with a range of key size options from 768 to 3072 bits (the DSS key for signatures is limited to 1024 bits).

### Confidentiality and Authentication

As Figure 12.1c illustrates, both services may be used for the same message. First, a signature is generated for the plaintext message and prepended to the message. Then the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES), and the session key is encrypted using RSA (or ElGamal). This sequence is preferable to the opposite: encrypting the message and then generating a signature for the encrypted message. It is generally more convenient to store a signature with a plaintext version of a message. Furthermore, for purposes of third-party verification, if the signature is performed first, a third party need not be concerned with the conventional key when verifying the signature.

In summary, when both services are used, the sender first signs the message with its own private key, then encrypts the message with a session key, and then encrypts the session key with the recipient's public key.

### Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.

The placement of the compression algorithm, indicated by Z for compression and  $Z^{-1}$  for decompression in Figure 12.1, is critical:

1. The signature is generated before compression for two reasons:
  - a. It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification. If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.
  - b. Even if one were willing to generate dynamically a recompressed message for verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic; various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and, as a result, produce different compressed forms. However, these different compression algorithms are interoperable because any version of the algorithm can correctly decompress the output of any other version. Applying the hash function and signature after compression would constrain all PGP implementations to the same compression algorithm.
2. Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult.

The compression algorithm used is ZIP, which is described in Appendix 12A.



### E-mail Compatibility

When PGP is used, at least part of the block to be transmitted is encrypted. If only the signature service is used, then the message digest is encrypted (with the sender's private key). If the confidentiality service is used, the message plus signature (if present) are encrypted (with a one-time symmetric key). Thus, part or all of the resulting block consists of a stream of arbitrary 8-bit octets. However, many electronic mail systems only permit the use of blocks consisting of ASCII text. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters.

The scheme used for this purpose is radix-64 conversion. Each group of three octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors. See Appendix 12B for a description.

The use of radix 64 expands a message by 33%. Fortunately, the session key and signature portions of the message are relatively compact, and the plaintext message has been compressed. In fact, the compression should be more than enough to compensate for the radix-64 expansion. For example, [HELD96] reports an average compression ratio of about 2.0 using ZIP. If we ignore the relatively small signature and key components, the typical overall effect of compression and expansion of a file of length  $X$  would be  $1.33 \times 0.5 \times X = 0.665 \times X$ . Thus, there is still an overall compression of about one-third.

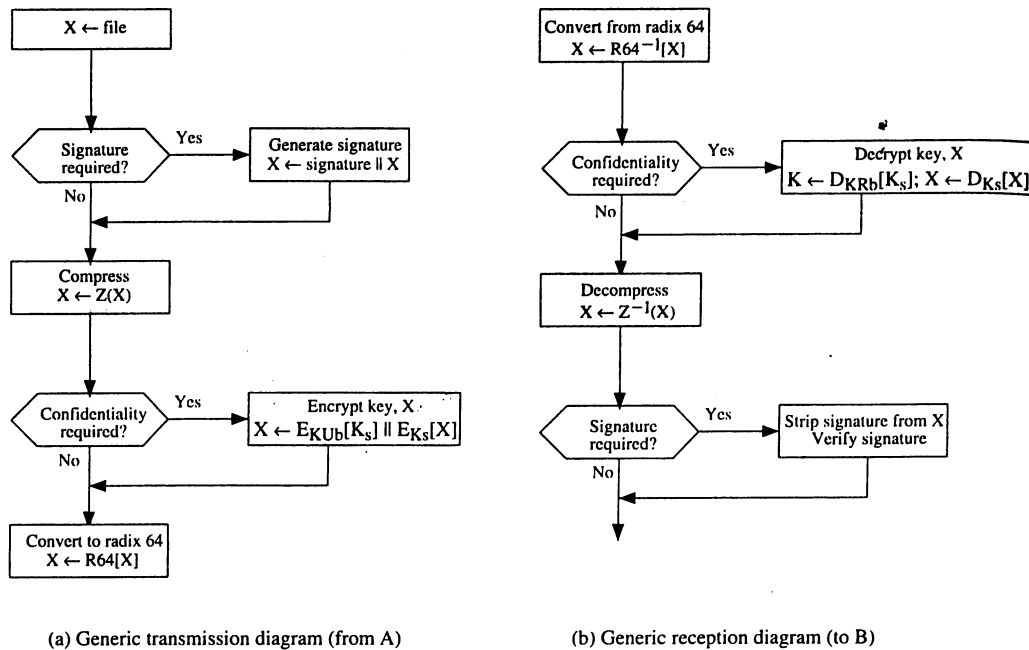
One noteworthy aspect of the radix-64 algorithm is that it blindly converts the input stream to radix-64 format regardless of content, even if the input happens to be ASCII text. Thus, if a message is signed but not encrypted and the conversion is applied to the entire block, the output will be unreadable to the casual observer, which provides a certain level of confidentiality. As an option, PGP can be configured to convert to radix-64 format only the signature portion of signed plaintext messages. This enables the human recipient to read the message without using PGP. PGP would still have to be used to verify the signature.

Figure 12.2 shows the relationship among the four services so far discussed. On transmission, if it is required, a signature is generated using a hash code of the compressed plaintext. Then the plaintext, plus signature if present, is compressed. Next, if confidentiality is required, the block (compressed plaintext or compressed signature plus plaintext) is encrypted and prepended with the public-key-encrypted conventional encryption key. Finally, the entire block is converted to radix-64 format.

On reception, the incoming block is first converted back from radix-64 format to binary. Then, if the message is encrypted, the recipient recovers the session key and decrypts the message. The resulting block is then decompressed. If the message is signed, the recipient recovers the transmitted hash code and compares it to its own calculation of the hash code.

### Segmentation and Reassembly

E-mail facilities often are restricted to a maximum message length. For example, many of the facilities accessible through the Internet impose a maximum length of 50,000 octets. Any message longer than that must be broken up into smaller segments, each of which is mailed separately.



**Figure 12.2** Transmission and Reception of PGP Messages.

To accommodate this restriction, PGP automatically subdivides a message that is too large into segments that are small enough to send via e-mail. The segmentation is done after all of the other processing, including the radix-64 conversion. Thus, the session key component and signature component appear only once, at the beginning of the first segment. At the receiving end, PGP must strip off all e-mail headers and reassemble the entire original block before performing the steps illustrated in Figure 12.2b.

### Cryptographic Keys and Key Rings

PGP makes use of four types of keys: one-time session conventional keys, public keys, private keys, and passphrase-based conventional keys (explained subsequently). Three separate requirements can be identified with respect to these keys:

1. A means of generating unpredictable session keys is needed.
2. We would like to allow a user to have multiple public-key/private-key pairs. One reason is that the user may wish to change his or her key pair from time to time. When this happens, any messages in the pipeline will be constructed with an obsolete key. Furthermore, recipients will know only the old public key until an update reaches them. In addition to the need to change keys over time, a user may wish to have multiple key pairs at a given time to interact with different groups of correspondents or simply to enhance security by limiting the amount of material encrypted with any one key. The upshot of all this is

that there is not a one-to-one correspondence between users and their public keys. Thus, some means is needed for identifying particular keys.

3. Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

We examine each of these requirements in turn.

### **Session Key Generation**

Each session key is associated with a single message and is used only for the purpose of encrypting and decrypting that message. Recall that message encryption/decryption is done with a symmetric encryption algorithm. CAST-128 and IDEA use 128-bit keys; 3DES uses a 168-bit key. For the following discussion, we assume CAST-128.

Random 128-bit numbers are generated using CAST-128 itself. The input to the random number generator consists of a 128-bit key and two 64-bit blocks that are treated as plaintext to be encrypted. Using cipher feedback mode, the CAST-128 encrypter produces two 64-bit cipher text blocks, which are concatenated to form the 128-bit session key. The algorithm that is used is based on the one specified in ANSI X12.17.

The “plaintext” input to the random number generator, consisting of two 64-bit blocks, is itself derived from a stream of 128-bit randomized numbers. These numbers are based on keystroke input from the user. Both the keystroke timing and the actual keys struck are used to generate the randomized stream. Thus, if the user hits arbitrary keys at his or her normal pace, a reasonably “random” input will be generated. This random input is also combined with previous session key output from CAST-128 to form the key input to the generator. The result, given the effective scrambling of CAST-128, is to produce a sequence of session keys that is effectively unpredictable.

Appendix 12C discusses PGP random number generation techniques in more detail.

### **Key Identifiers**

As we have discussed, an encrypted message is accompanied by an encrypted form of the session key that was used. The session key itself is encrypted with the recipient’s public key. Hence, only the recipient will be able to recover the session key and therefore recover the message. If each user employed a single public/private key pair, then the recipient would automatically know which key to use to decrypt the session key: the recipient’s unique private key. However, we have stated a requirement that any given user may have multiple public/private key pairs.

How, then, does the recipient know which of its public keys was used to encrypt the session key? One simple solution would be to transmit the public key with the message. The recipient could then verify that this is indeed one of its public keys, and proceed. This scheme would work, but it is unnecessarily wasteful of space. An RSA public key may be hundreds of decimal digits in length. Another solution would be to associate an identifier with each public key that is unique at least within one user. That is, the combination of user ID and key ID would be sufficient to identify a key uniquely. Then only the much shorter key ID would need

to be transmitted. This solution, however, raises a management and overhead problem: Key IDs must be assigned and stored so that both sender and recipient could map from key ID to public key. This seems unnecessarily burdensome.

The solution adopted by PGP is to assign a key ID to each public key that is, with very high probability, unique within a user ID.<sup>1</sup> The key ID associated with each public key consists of its least significant 64 bits. That is, the key ID of public key  $KU_a$  is  $(KU_a \bmod 2^{64})$ . This is a sufficient length that the probability of duplicate key IDs is very small.

A key ID is also required for the PGP digital signature. Because a sender may use one of a number of private keys to encrypt the message digest, the recipient must know which public key is intended for use. Accordingly, the digital signature component of a message includes the 64-bit key ID of the required public key. When the message is received, the recipient verifies that the key ID is for a public key that it knows for that sender and then proceeds to verify the signature.

Now that the concept of key ID has been introduced, we can take a more detailed look at the format of a transmitted message, which is shown in Figure 12.3. A message consists of three components: the message component, a signature (optional), and a session key component (optional).

The **message component** includes the actual data to be stored or transmitted, as well as a filename and a timestamp that specifies the time of creation.

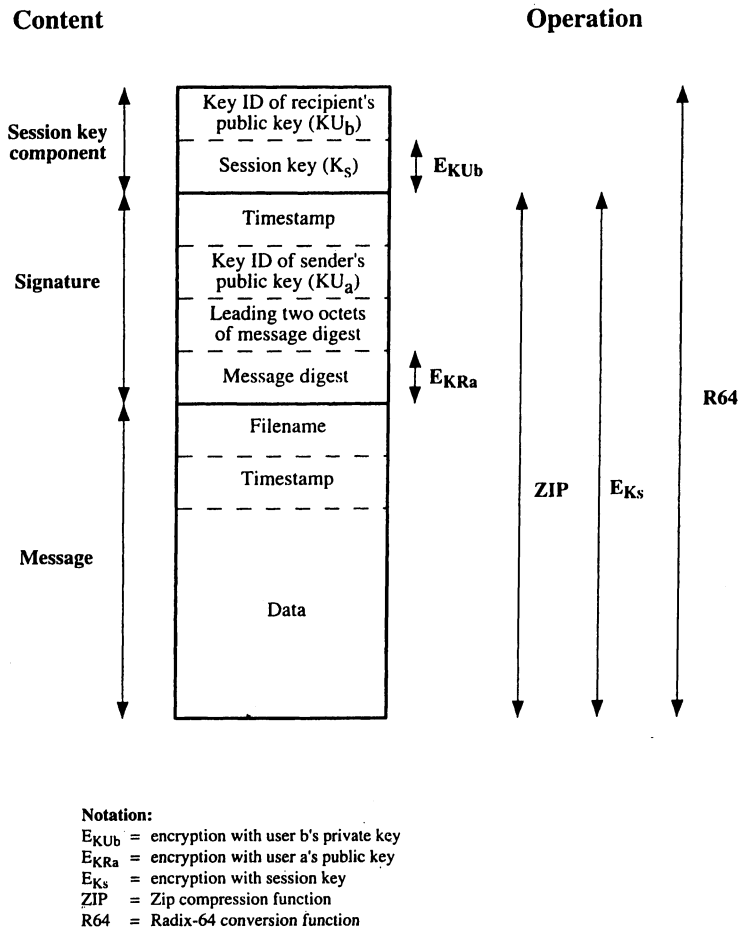
The **signature component** includes the following components:

- **Timestamp:** The time at which the signature was made.
- **Message digest:** The 160-bit SHA-1 digest, encrypted with the sender's private signature key. The digest is calculated over the signature timestamp concatenated with the data portion of the message component. The inclusion of the signature timestamp in the digest assures against replay types of attacks. The exclusion of the filename and timestamp portions of the message component ensures that detached signatures are exactly the same as attached signatures prefixed to the message. Detached signatures are calculated on a separate file that has none of the message component header fields.
- **Leading two octets of message digest:** To enable the recipient to determine if the correct public key was used to decrypt the message digest for authentication, by comparing this plaintext copy of the first two octets with the first two octets of the decrypted digest. These octets also serve as a 16-bit frame-check sequence for the message.
- **Key ID of sender's public key:** Identifies the public key that should be used to decrypt the message digest and, hence, identifies the private key that was used to encrypt the message digest.

The message component and optional signature component may be compressed using ZIP and may be encrypted using a session key.

---

<sup>1</sup>We have seen this introduction of probabilistic concepts before, in Section 7.4, for determining whether a number is prime. It is often the case in designing algorithms that the use of probabilistic techniques results in a less time-consuming or less complex solution, or both.



**Figure 12.3** General Format of PGP Message (from A to B).

The **session key component** includes the session key and the identifier of the recipient's public key that was used by the sender to encrypt the session key.

The entire block is usually encoded with radix-64 encoding.

### Key Rings

We have seen how key IDs are critical to the operation of PGP and that two key IDs are included in any PGP message that provides both confidentiality and authentication. These keys need to be stored and organized in a systematic way for efficient and effective use by all parties. The scheme used in PGP is to provide a pair of data structures at each node, one to store the public/private key pairs owned by that node and one to store the public keys of other users known at this node. These data structures are referred to, respectively, as the private-key ring and the public-key ring.

Private-Key Ring				
Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•
$T_i$	$KU_i \bmod 2^{64}$	$KU_i$	$E_{H(P_i)}[KR_i]$	User $i$
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•

Public-Key Ring							
Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
$T_i$	$KU_i \bmod 2^{64}$	$KU_i$	$trust\_flag_i$	User $i$	$trust\_flag_i$		
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•

\* = field used to index table

**Figure 12.4** General Structure of Private- and Public-Key Rings.

Figure 12.4 shows the general structure of a **private-key ring**. We can view the ring as a table, in which each row represents one of the public/private key pairs owned by this user. Each row contains the following entries:

- **Timestamp:** The date/time when this key pair was generated.
- **Key ID:** The least significant 64 bits of the public key for this entry.
- **Public key:** The public-key portion of the pair.
- **Private key:** The private-key portion of the pair; this field is encrypted.
- **User ID:** Typically, this will be the user's e-mail address (e.g., stallings@acm.org). However, the user may choose to associate a different name with each pair (e.g., Stallings, WStallings, WilliamStallings, etc.) or to reuse the same user ID more than once.

The private-key ring can be indexed by either User ID or Key ID; later we will see the need for both means of indexing.

Although it is intended that the private-key ring be stored only on the machine of the user that created and owns the key pairs, and that it be accessible only to that user, it makes sense to make the value of the private key as secure as possible. Accordingly, the private key itself is not stored in the key ring. Rather, this key is encrypted using CAST-128 (or IDEA or 3DES). The procedure is as follows:

1. The user selects a passphrase to be used for encrypting private keys.
2. When the system generates a new public/private key pair using RSA, it asks the user for the passphrase. Using SHA-1, a 160-bit hash code is generated from the passphrase, and the passphrase is discarded.
3. The system encrypts the private key using CAST-128 with the 128 bits of the hash code as the key. The hash code is then discarded, and the encrypted private key is stored in the private-key ring.

Subsequently, when a user accesses the private-key ring to retrieve a private key, he or she must supply the passphrase. PGP will retrieve the encrypted private key, generate the hash code of the passphrase, and decrypt the encrypted private key using CAST-128 with the hash code.

This is a very compact and effective scheme. As in any system based on passwords, the security of this system depends on the security of the password. To avoid the temptation to write it down, the user should use a passphrase that is not easily guessed but that is easily remembered.

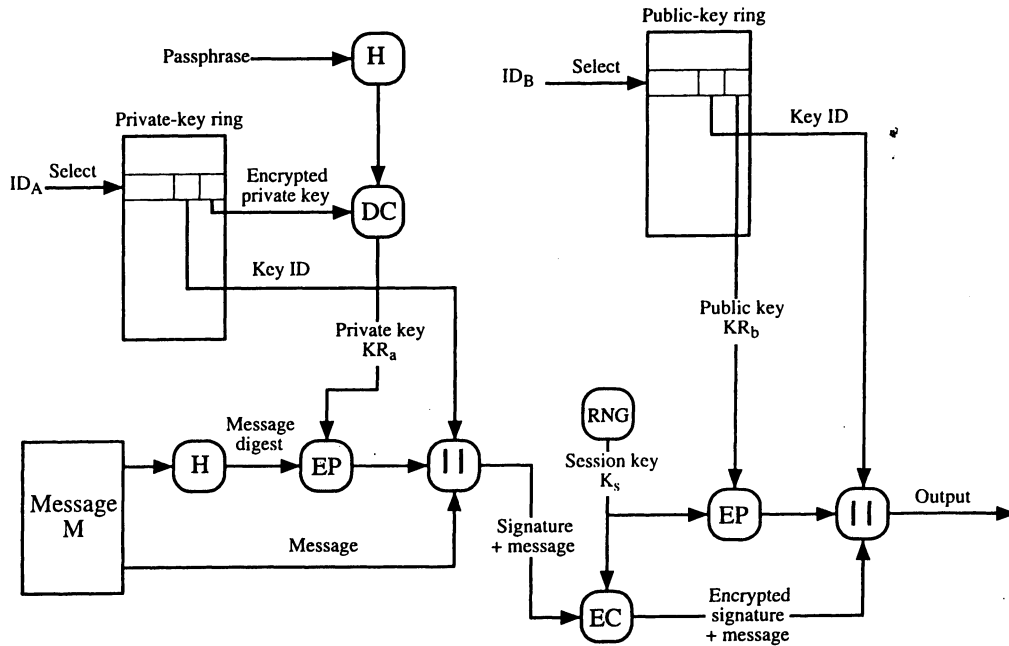
Figure 12.4 also shows the general structure of a **public-key ring**. This data structure is used to store public keys of other users that are known to this user. For the moment, let us ignore some fields shown in the table and describe the following fields:

- **Timestamp:** The date/time when this entry was generated.
- **Key ID:** The least significant 64 bits of the public key for this entry.
- **Public key:** The public key for this entry.
- **User ID:** The owner of this key. Multiple user IDs may be associated with a single public key.

The public-key ring can be indexed by either User ID or Key ID; we will see the need for both means of indexing later.

We are now in a position to show how these key rings are used in message transmission and reception. For simplicity, we ignore compression and radix-64 conversion in the following discussion. First consider message transmission (Figure 12.5) and assume that the message is to be both signed and encrypted. The sending PGP entity performs the following steps:

1. Signing the message
  - a. PGP retrieves the sender's private key from the private-key ring using `your_userid` as an index. If `your_userid` was not provided in the command, the first private key on the ring is retrieved.
  - b. PGP prompts the user for the passphrase to recover the unencrypted private key.
  - c. The signature component of the message is constructed.
2. Encrypting the message
  - a. PGP generates a session key and encrypts the message.



**Figure 12.5** PGP Message Generation (from user A to user B; no compression or radix-64 conversion).

- b. PGP retrieves the recipient's public key from the public-key ring using her\_userid as an index.
- c. The session key component of the message is constructed.

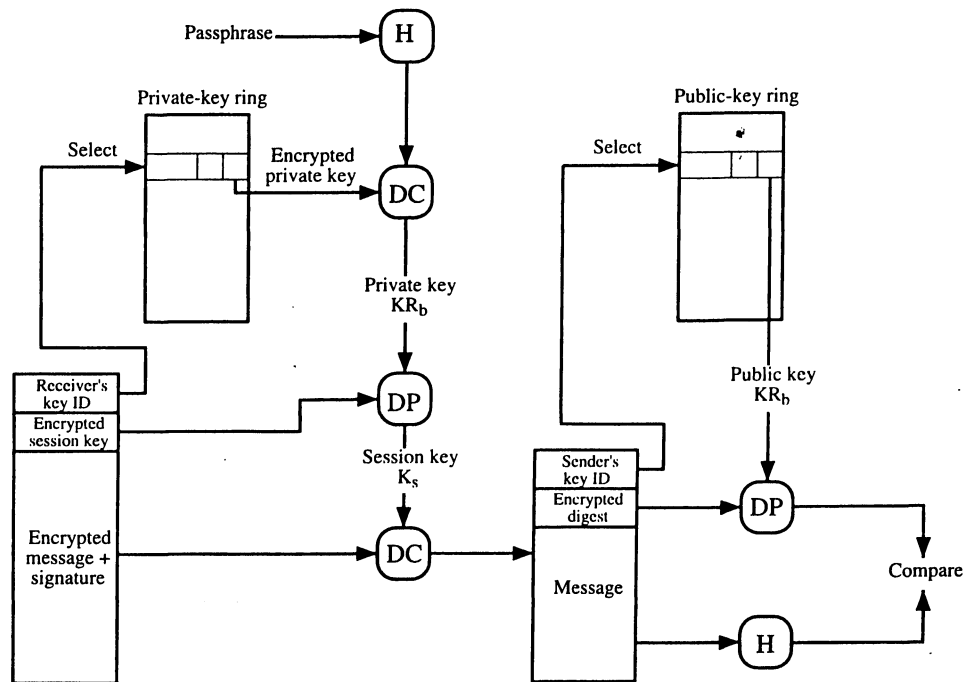
The receiving PGP entity performs the following steps (Figure 12.6):

1. Decrypting the message
  - a. PGP retrieves the receiver's private key from the private-key ring, using the Key ID field in the session key component of the message as an index.
  - b. PGP prompts the user for the passphrase to recover the unencrypted private key.
  - c. PGP then recovers the session key and decrypts the message.
2. Authenticating the message
  - a. PGP retrieves the sender's public key from the public-key ring, using the Key ID field in the signature key component of the message as an index.
  - b. PGP recovers the transmitted message digest.
  - c. PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

### Public-Key Management

As can be seen from the discussion so far, PGP contains a clever, efficient, interlocking set of functions and formats to provide an effective confidentiality and authentication service. To complete the system, one final area needs to be addressed,





**Figure 12.6** PGP Message Reception (from user A to user B; no compression or radix-64 conversion).

that of public-key management. The PGP documentation captures the importance of this area:

This whole business of protecting public keys from tampering is the single most difficult problem in practical public key applications. It is the “Achilles heel” of public key cryptography, and a lot of software complexity is tied up in solving this one problem.

PGP provides a structure for solving this problem, with several suggested options that may be used. Because PGP is intended for use in a variety of formal and informal environments, no rigid public-key management scheme is set up, such as we will see in our discussion of S/MIME later in this chapter.

### Approaches to Public-Key Management

The essence of the problem is this: User A must build up a public-key ring containing the public keys of other users to interoperate with them using PGP. Suppose that A’s key ring contains a public key attributed to B but that the key is, in fact, owned by C. This could happen if, for example, A got the key from a bulletin board system (BBS) that was used by B to post the public key but that has been compromised by C. The result is that two threats now exist. First, C can send messages to A and forge B’s signature, so that A will accept the message as coming from B. Second, any encrypted message from A to B can be read by C.

A number of approaches are possible for minimizing the risk that a user’s public-key ring contains false public keys. Suppose that A wishes to obtain a reliable public key for B. The following are some approaches that could be used:

1. Physically get the key from B. B could store her public key ( $KU_b$ ) on a floppy disk and hand it to A. A could then load the key into his system from the floppy disk. This is a very secure method but has obvious practical limitations.
2. Verify a key by telephone. If A can recognize B on the phone, A could call B and ask her to dictate the key, in radix-64 format, over the phone. As a more practical alternative, B could transmit her key in an e-mail message to A. A could have PGP generate a 160-bit SHA-1 digest of the key and display it in hexadecimal format; this is referred to as the "fingerprint" of the key. A could then call B and ask her to dictate the fingerprint over the phone. If the two fingerprints match, the key is verified.
3. Obtain B's public key from a mutual trusted individual D. For this purpose, the introducer, D, creates a signed certificate. The certificate includes B's public key, the time of creation of the key, and a validity period for the key. D generates an SHA-1 digest of this certificate, encrypts it with her private key, and attaches the signature to the certificate. Because only D could have created the signature, no one else can create a false public key and pretend that it is signed by D. The signed certificate could be sent directly to A by B or D, or could be posted on a bulletin board.
4. Obtain B's public key from a trusted certifying authority. Again, a public-key certificate is created and signed by the authority. A could then access the authority, providing a user name and receiving a signed certificate.

For cases 3 and 4, A would already have to have a copy of the introducer's public key and trust that this key is valid. Ultimately, it is up to A to assign a level of trust to anyone who is to act as an introducer.

### The Use of Trust

Although PGP does not include any specification for establishing certifying authorities or for establishing trust, it does provide a convenient means of using trust, associating trust with public keys, and exploiting trust information.

The basic structure is as follows. Each entry in the public-key ring is a public-key certificate, as described in the preceding subsection. Associated with each such entry is a **key legitimacy field** that indicates the extent to which PGP will trust that this is a valid public key for this user; the higher the level of trust, the stronger is the binding of this user ID to this key. This field is computed by PGP. Also associated with the entry are zero or more signatures that the key ring owner has collected that sign this certificate. In turn, each signature has associated with it a **signature trust field** that indicates the degree to which this PGP user trusts the signer to certify public keys. The key legitimacy field is derived from the collection of signature trust fields in the entry. Finally, each entry defines a public key associated with a particular owner, and an **owner trust field** is included that indicates the degree to which this public key is trusted to sign other public-key certificates; this level of trust is assigned by the user. We can think of the signature trust fields as cached copies of the owner trust field from another entry.

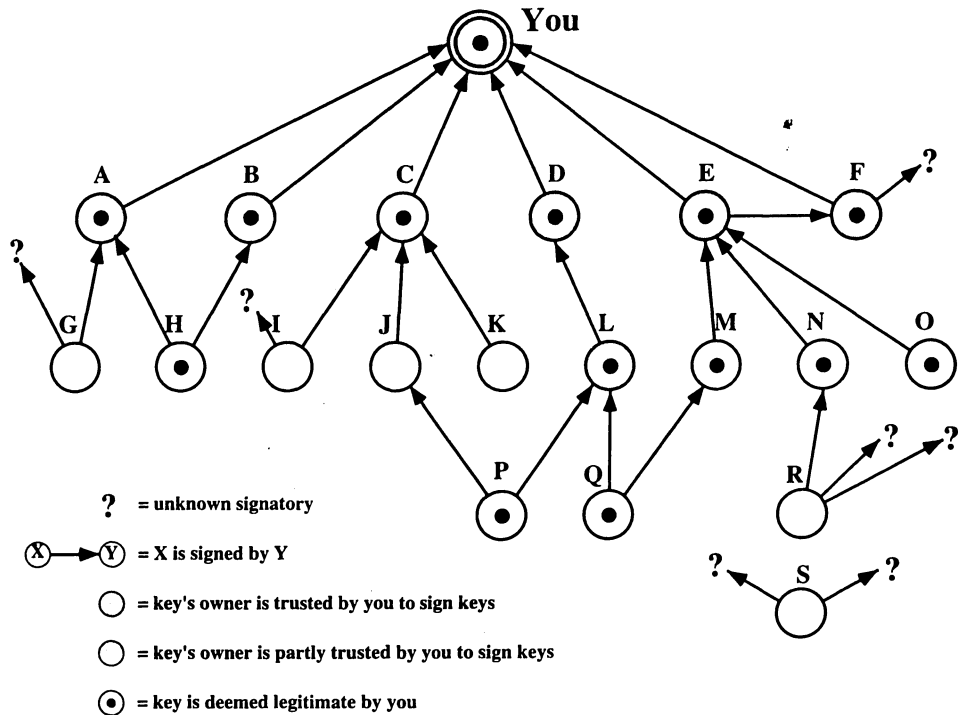
The three fields mentioned in the previous paragraph are each contained in a structure referred to as a trust flag byte. The content of this trust flag for each of

**Table 12.2** Contents of Trust Flag Byte

(a) Trust Assigned to Public-Key Owner (appears after key packet; user defined)	(b) Trust Assigned to Public Key/User ID Pair (appears after User ID packet; computed by PGP)	(c) Trust Assigned to Signature (appears after signature packet; cached copy of OWNERTRUST for this signator)
<b>OWNERTRUST Field</b> —undefined trust —unknown user —usually not trusted to sign other keys —usually trusted to sign other keys —always trusted to sign other keys —this key is present in secret key ring (ultimate trust) <b>BUCKSTOP bit</b> —set if this key appears in secret key ring	<b>KEYLEGIT Field</b> —unknown or undefined trust —key ownership not trusted —marginal trust in key ownership —complete trust in key ownership <b>WARNONLY bit</b> —set if user wants only to be warned when key that is not fully validated is used for encryption	<b>SIGTRUST Field</b> —undefined trust —unknown user —usually not trusted to sign other keys —usually trusted to sign other keys —always trusted to sign other keys —this key is present in secret key ring (ultimate trust) <b>CONTIG bit</b> —set if signature leads up a contiguous trusted certification path back to the ultimately trusted key ring owner

these three uses is shown in Table 12.2. Suppose that we are dealing with the public-key ring of user A. We can describe the operation of the trust processing as follows:

1. When A inserts a new public key on the public-key ring, PGP must assign a value to the trust flag that is associated with the owner of this public key. If the owner is A, and therefore this public key also appears in the private-key ring, then a value of *ultimate trust* is automatically assigned to the trust field. Otherwise, PGP asks A for his assessment of the trust to be assigned to the owner of this key, and A must enter the desired level. The user can specify that this owner is unknown, untrusted, marginally trusted, or completely trusted.
2. When the new public key is entered, one or more signatures may be attached to it. More signatures may be added later. When a signature is inserted into the entry, PGP searches the public-key ring to see if the author of this signature is among the known public-key owners. If so, the OWNERTRUST value for this owner is assigned to the SIGTRUST field for this signature. If not, an *unknown user* value is assigned.
3. The value of the key legitimacy field is calculated on the basis of the signature trust fields present in this entry. If at least one signature has a signature trust value of *ultimate*, then the key legitimacy value is set to complete. Otherwise, PGP computes a weighted sum of the trust values. A weight of  $1/X$  is given to signatures that are always trusted and  $1/Y$  to signatures that are usually trusted, where X and Y are user-configurable parameters. When the total of weights of the introducers of a key/UserID combination reaches 1, the binding is considered to be trustworthy, and the key legitimacy value is set to complete. Thus, in the absence of ultimate trust, at least X signatures that are always trusted or Y signatures that are usually trusted or some combination is needed.



**Figure 12.7** PGP Trust Model Example.

Periodically, PGP processes the public-key ring to achieve consistency. In essence, this is a top-down process. For each OWNERTRUST field, PGP scans the ring for all signatures authored by that owner and updates the SIGTRUST field to equal the OWNERTRUST field. This process starts with keys for which there is ultimate trust. Then all KEYLEGIT fields are computed on the basis of the attached signatures.

Figure 12.7 provides an example of the way in which signature trust and key legitimacy are related.<sup>2</sup> The figure shows the structure of a public-key ring. The user has acquired a number of public keys, some directly from their owners and some from a third party such as a key server.

The node labeled “You” refers to the entry in the public-key ring corresponding to this user. This key is legitimate and the OWNERTRUST value is ultimate trust. Each other node in the key ring has an OWNERTRUST value of undefined unless some other value is assigned by the user. In this example, this user has specified that it always trusts the following users to sign other keys: D, E, F, L. This user partially trusts users A and B to sign other keys.

So the shading, or lack thereof, of the nodes in Figure 12.7 indicates the level of trust assigned by this user. The tree structure indicates which keys have been signed by which other users. If a key is signed by a user whose key is also in this key

<sup>2</sup>Figure provided to the author by Phil Zimmermann.

ring, the arrow joins the signed key to the signatory. If the key is signed by a user whose key is not present in this key ring, the arrow joins the signed key to a question mark, indicating that the signatory is unknown to this user.

Several points are illustrated in Figure 12.7:

1. Note that all keys whose owners are fully or partially trusted by this user have been signed by this user, with the exception of node L. Such a user signature is not always necessary, as the presence of node L indicates, but in practice, most users are likely to sign the keys for most owners that they trust. So, for example, even though E's key is already signed by trusted introducer F, the user chose to sign E's key directly.
2. We assume that two partially trusted signatures are sufficient to certify a key. Hence, the key for user H is deemed legitimate by PGP because it is signed by A and B, both of whom are partially trusted.
3. A key may be determined to be legitimate because it is signed by one fully trusted or two partially trusted signatories, but its user may not be trusted to sign other keys. For example, N's key is legitimate because it is signed by E, whom this user trusts, but N is not trusted to sign other keys because this user has not assigned N that trust value. Therefore, although R's key is signed by N, PGP does not consider R's key legitimate. This situation makes perfect sense. If you wish to send a private message to some individual, it is not necessary that you trust that individual in any respect. It is only necessary that you are sure that you have the correct public key for that individual.
4. Figure 12.7 also shows an example of a detached "orphan" node S, with two unknown signatures. Such a key may have been acquired from a key server. PGP cannot assume that this key is legitimate simply because it came from a reputable server. The user must declare the key legitimate by signing it or by telling PGP that it is willing to trust fully one of the key's signatories.

A final point: Earlier it was mentioned that multiple user IDs may be associated with a single public key on the public-key ring. This could be because a person has changed names or has been introduced via signature under multiple names, indicating different e-mail addresses for the same person, for example. So we can think of a public key as the root of a tree. A public key has a number of user IDs associated with it, with a number of signatures below each user ID. The binding of a particular user ID to a key depends on the signatures associated with that user ID and that key, whereas the level of trust in this key (for use in signing other keys) is a function of all the dependent signatures.

### Revoking Public Keys

A user may wish to revoke his or her current public key either because compromise is suspected or simply to avoid the use of the same key for an extended period. Note that a compromise would require that an opponent somehow had obtained a copy of your unencrypted private key or that the opponent had obtained both the private key from your private-key ring and your passphrase.

The convention for revoking a public key is for the owner to issue a key revocation certificate, signed by the owner. This certificate has the same form as a normal signature certificate but includes an indicator that the purpose of this certificate is to revoke the use of this public key. Note that the corresponding private key must be used to sign a certificate that revokes a public key. The owner should then attempt to disseminate this certificate as widely and as quickly as possible to enable potential correspondents to update their public-key rings.

Note that an opponent who has compromised the private key of an owner can also issue such a certificate. However, this would deny the opponent as well as the legitimate owner the use of the public key, and therefore it seems a much less likely threat than the malicious use of a stolen private key.

## 12.2 S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet e-mail format standard, based on technology from RSA Data Security. Although both PGP and S/MIME are on an IETF standards track, it appears likely that S/MIME will emerge as the industry standard for commercial and organizational use, while PGP will remain the choice for personal e-mail security for many users.

To understand S/MIME, we need first to have a general understanding of the underlying e-mail format that it uses (namely, MIME). But to understand the significance of MIME, we need to go back to the traditional e-mail format standard, RFC 822, which is still in common use. Accordingly, this section first provides an introduction to these two earlier standards and then moves on to a discussion of S/MIME.

### RFC 822

RFC 822 defines a format for text messages that are sent using electronic mail. It has been the standard for Internet-based text mail message and remains in common use. In the RFC 822 context, messages are viewed as having an envelope and contents. The envelope contains whatever information is needed to accomplish transmission and delivery. The contents compose the object to be delivered to the recipient. The RFC 822 standard applies only to the contents. However, the content standard includes a set of header fields that may be used by the mail system to create the envelope, and the standard is intended to facilitate the acquisition of such information by programs.

The overall structure of a message that conforms to RFC 822 is very simple. A message consists of some number of header lines (*the header*) followed by unrestricted text (*the body*). The header is separated from the body by a blank line. Put differently, a message is ASCII text, and all lines up to the first blank line are assumed to be header lines used by the user agent part of the mail system.

A header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines. The most frequently used keywords are *From*, *To*, *Subject*, and *Date*. Here is an example message:

Date: Tue, 16 Jan 1998 10:37:17 (EST)  
 From: "William Stallings" <ws@shore.net>  
 Subject: The Syntax in RFC 822  
 To: Smith@Other-host.com  
 Cc: Jones@Yet-Another-Host.com

Hello. This section begins the actual message body, which is delimited from the message heading by a blank line.

Another field that is commonly found in RFC 822 headers is *Message-ID*. This field contains a unique identifier associated with this message.

### Multipurpose Internet Mail Extensions

MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol) or some other mail transfer protocol and RFC 822 for electronic mail. [MURP95] lists the following limitations of the SMTP/822 scheme:

1. SMTP cannot transmit executable files or other binary objects. A number of schemes are in use for converting binary files into a text form that can be used by SMTP mail systems, including the popular UNIX UUencode/UUdecode scheme. However, none of these is a standard or even a de facto standard.
2. SMTP cannot transmit text data that includes national language characters because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.
3. SMTP servers may reject mail message over a certain size.
4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
5. SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included in X.400 messages.
6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include the following:
  - Deletion, addition, or reordering of carriage return and linefeed
  - Truncating or wrapping lines longer than 76 characters
  - Removal of trailing white space (tab and space characters)
  - Padding of lines in a message to the same length
  - Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations. The specification is provided in RFCs 2045 through 2049.

#### Overview

The MIME specification includes the following elements:

1. Five new message header fields are defined, which may be included in an RFC 822 header. These fields provide information about the body of the message.

2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. Transfer encodings are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

In this subsection, we introduce the five message header fields. The next two subsections deal with content formats and transfer encodings.

The five header fields defined in MIME are as follows:

- **MIME-Version:** Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.
- **Content-Type:** Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to represent the data to the user or otherwise deal with the data in an appropriate manner.
- **Content-Transfer-Encoding:** Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.
- **Content-ID:** Used to identify MIME entities uniquely in multiple contexts.
- **Content-Description:** A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

Any or all of these fields may appear in a normal RFC 822 header. A compliant implementation must support the MIME-Version, Content-Type, and Content-Transfer-Encoding fields; the Content-ID and Content-Description fields are optional and may be ignored by the recipient implementation.

### MIME Content Types

The bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment.

Table 12.3 lists the content types specified in RFC 2046. There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data.

For the **text type** of body, no special software is required to get the full meaning of the text, aside from support of the indicated character set. The primary subtype is *plain text*, which is simply a string of ASCII characters or ISO 8859 characters. The *enriched* subtype allows greater formatting flexibility.

The **multipart type** indicates that the body contains multiple, independent parts. The Content-Type header field includes a parameter, called *boundary*, that defines the delimiter between body parts. This boundary should not appear in any parts of the message. Each boundary starts on a new line and consists of two hyphens followed by the boundary value. The final boundary, which indicates the end of the last part, also has a suffix of two hyphens. Within each part, there may be an optional ordinary MIME header.



**Table 12.3** MIME Content Types

Type	Subtype	Description
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
	Enriched	Provides greater format flexibility.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the "best" version to the user.
	Digest	Similar to Mixed, but the default type/subtype of each part is message/rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript
	octet-stream	General binary data consisting of 8-bit bytes.

Here is a simple example of a multipart message, containing two parts both consisting of simple text (taken from RFC 2046):

```

From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Sample message
MIME-Version: 1.0
Content-type: multipart/mixed; boundary="simple boundary"

```

This is the preamble. It is to be ignored, though it is a handy place for mail composers to include an explanatory note to non-MIME conformant readers.

--simple boundary

This is implicitly typed plain ASCII text. It does NOT end with a linebreak.

--simple boundary

Content-type: text/plain; charset=us-ascii

This is explicitly typed plain ASCII text. It DOES end with a linebreak.

--simple boundary--

This is the epilogue. It is also to be ignored.

There are four subtypes of the multipart type, all of which have the same overall syntax. The **multipart/mixed subtype** is used when there are multiple independent body parts that need to be bundled in a particular order. For the **multipart/parallel subtype**, the order of the parts is not significant. If the recipient's system is appropriate, the multiple parts can be presented in parallel. For example, a picture or text part could be accompanied by a voice commentary that is played while the picture or text is displayed.

For the **multipart/alternative subtype**, the various parts are different representations of the same information. The following is an example:

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: multipart/alternative; boundary=boundary42
```

--boundary42

Content-Type: text/plain; charset=us-ascii

... plain text version of message goes here ....

--boundary42

Content-Type: text/enriched

.... RFC 1896 text/enriched version of same message goes here ...

--boundary42--

In this subtype, the body parts are ordered in terms of increasing preference. For this example, if the recipient system is capable of displaying the message in the text/enriched format, this is done; otherwise, the plain text format is used.

The **multipart/digest subtype** is used when each of the body parts is interpreted as an RFC 822 message with headers. This subtype enables the construction of a message whose parts are individual messages. For example, the moderator of a group might collect e-mail messages from participants, bundle these messages, and send them out in one encapsulating MIME message.

The **message type** provides a number of important capabilities in MIME. The **message/rfc822 subtype** indicates that the body is an entire message, including header and body. Despite the name of this subtype, the encapsulated message may be not only a simple RFC 822 message, but any MIME message.

The **message/partial subtype** enables fragmentation of a large message into a number of parts, which must be reassembled at the destination. For this subtype, three parameters are specified in the Content-Type: Message/Partial field: an *id* common to all fragments of the same message, a *sequence number* unique to each fragment, and the *total* number of fragments.

The **message/external-body subtype** indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data. As with the other message types, the message/external-body subtype has an outer header and an encapsulated message with its own header. The only necessary field in the outer header is the Content-Type field, which identifies this as a message/external-body subtype. The inner header is the message header for the encapsulated message. The Content-Type field in the outer header must include an access-type parameter, which indicates the method of access, such as FTP (file transfer protocol).

The **application type** refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application.

### MIME Transfer Encodings

The other major component of the MIME specification, in addition to content-type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

The MIME standard defines two methods of encoding data. The Content-Transfer-Encoding field can actually take on six values, as listed in Table 12.4. However, three of these values (7bit, 8bit, and binary) indicate that no encoding has been done, but provide some information about the nature of the data. For SMTP transfer, it is safe to use the 7bit form. The 8bit and binary forms may be usable in other mail transport contexts. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used, for which a name is to be supplied. This could be a vendor-specific or application-specific scheme. The two actual encoding schemes defined are quoted-printable and base64. Two schemes are defined to provide a choice between a transfer technique that is essentially human readable and one that is safe for all types of data in a way that is reasonably compact.

**Table 12.4** MIME Transfer Encodings

7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
binary	Not only may non-ASCII characters be present, but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by humans.
base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

The **quoted-printable** transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents unsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters.

The **base64 transfer encoding**, also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail transport programs. It is also used in PGP and is described in Appendix 12B.

### A Multipart Example

Figure 12.8, taken from RFC 1521, is the outline of a complex multipart message. The message has five parts to be displayed serially: two introductory plain text parts, an embedded multipart message, a rich text part, and a closing encapsulated text message in a non-ASCII character set. The embedded multipart message has two parts to be displayed in parallel, a picture and an audio fragment.

### Canonical Form

An important concept in MIME and S/MIME is that of canonical form. Canonical form is a format, appropriate to the content type, that is standardized for use between systems. This is in contrast to native form, which is a format that may be peculiar to a particular system. Table 12.5, from RFC 2049, should help clarify this matter.

### S/MIME Functionality

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability. We then look in more detail at this capability by examining message formats and message preparation.

#### Functions

S/MIME provides the following functions:

- **Enveloped data:** This consists of encrypted content of any type and encrypted-content encryption keys for one or more recipients.
- **Signed data:** A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.
- **Clear-signed data:** As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.
- **Signed and enveloped data:** Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

From: Nathaniel Borenstein <nsb@bellcore.com>  
To: Ned Freed <ned@innosoft.com>  
Subject: A multipart example  
Content-Type: multipart/mixed;  
    boundary=unique-boundary-1

This is the preamble area of a multipart message. Mail readers that understand multipart format should ignore this preamble. If you are reading this text, you might want to consider changing to a mail reader that understands how to properly display multipart messages.

--unique-boundary-1

...Some text appears here...

[Note that the preceding blank line means no header fields were given and this is text, with charset US ASCII. It could have been done with explicit typing as in the next part.]

--unique-boundary-1

Content-type: text/plain; charset=US-ASCII

This could have been part of the previous part, but illustrates explicit versus implicit typing of body parts.

--unique-boundary-1

Content-Type: multipart/parallel; boundary=unique-boundary-2

--unique-boundary-2

Content-Type: audio/basic

Content-Transfer-Encoding: base64

... base64-encoded 8000 Hz single-channel mu-law-format audio data goes here....

--unique-boundary-2

Content-Type: image/jpeg

Content-Transfer-Encoding: base64

... base64-encoded image data goes here....

--unique-boundary-2--

--unique-boundary-1

Content-type: text/enriched

This is <bold><italic>richtext.</italic></bold> <smaller>as defined in RFC 1896</smaller>

Isn't it <bigger><bigger>cool?</bigger></bigger>

--unique-boundary-1

Content-Type: message/rfc822

From: (mailbox in US-ASCII)

To: (address in US-ASCII)

Subject: (subject in US-ASCII)

Content-Type: Text/plain; charset=ISO-8859-1

Content-Transfer-Encoding: Quoted-printable

... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--

**Figure 12.8** Example MIME Message Structure.

**Table 12.5** Native and Canonical Form

Native Form	The body to be transmitted is created in the system's native format. The native character set is used and, where appropriate, local end-of-line conventions are used as well. The body may be a UNIX-style text file, or a Sun raster image, or a VMS indexed file, or audio data in a system-dependent format stored only in memory, or anything else that corresponds to the local model for the representation of some form of information. Fundamentally, the data is created in the "native" form that corresponds to the type specified by the media type.
Canonical Form	The entire body, including "out-of-band" information such as record lengths and possibly file attribute information, is converted to a universal canonical form. The specific media type of the body as well as its associated attributes dictate the nature of the canonical form that is used. Conversion to the proper canonical form may involve character set conversion, transformation of audio data, compression, or various other operations specific to the various media types. If character set conversion is involved, however, care must be taken to understand the semantics of the media type, which may have strong implications for any character set conversion (e.g., with regard to syntactically meaningful characters in a text subtype other than "plain").

### Cryptographic Algorithms

Table 12.6 summarizes the cryptographic algorithms used in S/MIME. S/MIME uses the following terminology, taken from RFC 2119 to specify the requirement level:<sup>3</sup>

- **MUST:** The definition is an absolute requirement of the specification. An implementation must include this feature or function to be in conformance with the specification.
- **SHOULD:** There may exist valid reasons in particular circumstances to ignore this feature or function, but it is recommended that an implementation include the feature or function.

S/MIME incorporates three public-key algorithms. The Digital Signature Standard (DSS) described in Chapter 10 is the preferred algorithm for digital signature. S/MIME lists Diffie-Hellman as the preferred algorithm for encrypting session keys; in fact, S/MIME uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal (see Problem 6.19). As an alternative, RSA, described in Chapter 6, can be used for both signatures and session key encryption. These are the same algorithms used in PGP and provide a high level of security. For the hash function used to create the digital signature, the specification recommends the 160-bit SHA-1 but requires support for the 128-bit MD5. As we discussed in Chapter 9, there is justifiable concern about the security of MD5, so SHA-1 is clearly the preferred alternative. However, MD5 is widely implemented, so support is provided.

For message encryption, three-key triple DES (tripleDES) is recommended, but compliant implementations must support 40-bit RC2. The latter is a weak encryption algorithm but allows compliance with U.S. export controls.

<sup>3</sup>RFC 2119, *Key Words for Use in RFCs to Indicate Requirement Levels*, March 1997.

**Table 12.6** Cryptographic Algorithms Used in S/MIME

Function	Requirement
Create a message digest to be used in forming a digital signature.	MUST support SHA-1 and MD5. SHOULD use SHA-1.
Encrypt message digest to form digital signature.	Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of RSA signatures with key sizes 512 bits to 1024 bits.
Encrypt session key for transmission with message.	Sending and receiving agents MUST support Diffie-Hellman. Sending agent SHOULD support RSA encryption with key sizes 512 bits to 1024 bits. Receiving agent SHOULD support RSA decryption.
Encrypt message for transmission with one-time session key.	Sending agents SHOULD support encryption with tripleDES and RC2/40. Receiving agents MUST support decryption using tripleDES and SHOULD support decryption with RC2/40.

The S/MIME specification includes a discussion of the procedure for deciding which content encryption algorithm to use. In essence, a sending agent has two decisions to make. First, the sending agent must determine if the receiving agent is capable of decrypting using a given encryption algorithm. Second, if the receiving agent is only capable of accepting weakly encrypted content, the sending agent must decide if it is acceptable to send using weak encryption. To support this decision process, a sending agent may announce its decrypting capabilities in order of preference any message that it sends out. A receiving agent may store that information for future use.

The following rules, in the following order, should be followed by a sending agent:

1. If the sending agent has a list of preferred decrypting capabilities from an intended recipient, it SHOULD choose the first (highest preference) capability on the list that it is capable of using.
2. If the sending agent has no such list of capabilities from an intended recipient but has received one or more messages from the recipient, then the outgoing message SHOULD use the same encryption algorithm as was used on the last signed and encrypted message received from that intended recipient.
3. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is willing to risk that the recipient may not be able to decrypt the message, then the sending agent SHOULD use tripleDES.
4. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is not willing to risk that the recipient may not be able to decrypt the message, then the sending agent MUST use RC2/40.

If a message is to be sent to multiple recipients and a common encryption algorithm cannot be selected for all, then the sending agent will need to send two messages. However, in that case, it is important to note that the security of the message is made vulnerable by the transmission of one copy with lower security.

### S/MIME Messages

S/MIME makes use of a number of new MIME content types, which are shown in the Table 12.7. All of the new application types use the designation PKCS. This refers to a set of public-key cryptography specifications issued by RSA Laboratories and made available for the S/MIME effort.

We examine each of these in turn after first looking at the general procedures for S/MIME message preparation.

#### Securing a MIME Entity

S/MIME secures a MIME entity with a signature, encryption, or both. A MIME entity may be an entire message (except for the RFC 822 headers), or if the MIME content type is multipart, then a MIME entity is one or more of the subparts of the message. The MIME entity is prepared according to the normal rules for MIME message preparation. Then the MIME entity plus some security-related data, such as algorithm identifiers and certificates, are processed by S/MIME to produce what is known as a PKCS object. A PKCS object is then treated as message content and wrapped in MIME (provided with appropriate MIME headers). This process should become clear as we look at specific objects and provide examples.

In all cases, the message to be sent is converted to canonical form. In particular, for a given type and subtype, the appropriate canonical form is used for the message content. For a multipart message, the appropriate canonical form is used for each subpart.

The use of transfer encoding requires special attention. For most cases, the result of applying the security algorithm will be to produce an object that is partially or totally represented in arbitrary binary data. This will then be wrapped in an outer MIME message and transfer encoding can be applied at that point, typically base64. However, in the case of a multipart signed message, described in more detail later,

**Table 12.7** S/MIME Content Types

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: One is the message and the other is the signature.
Application	pkcs7-mime	signedData	A signed S/MIME entity.
	pkcs7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs7-mime	degenerate signedData	An entity containing only public-key certificates.
	pkcs7-signature	—	The content type of the signature subpart of a multipart/signed message.
	pkcs10-mime	—	A certificate registration request message.



the message content in one of the subparts is unchanged by the security process. Unless that content is 7bit, it should be transfer encoded using base64 or quoted printable, so that there is no danger of altering the content to which the signature was applied.

We now look at each of the S/MIME content types. \*

### Enveloped Data

An application/pkcs7-mime subtype is used for one of four categories of S/MIME processing, each with a unique smime-type parameter. In all cases, the resulting entity, referred to as an *object*, is represented in a form known as Basic Encoding Rules (BER), which is defined in ITU-T Recommendation X.209. The BER format consists of arbitrary octet strings and is therefore binary data. Such an object should be transfer encoded with base64 in the outer MIME message. We first look at envelopedData.

The steps for preparing an envelopedData MIME entity are as follows:

1. Generate a pseudorandom session key for a particular symmetric encryption algorithm (RC2/40 or tripleDES).
2. For each recipient, encrypt the session key with the recipient's public RSA key.
3. For each recipient, prepare a block known as RecipientInfo that contains the sender's public-key certificate,<sup>4</sup> an identifier of the algorithm used to encrypt the session key, and the encrypted session key.
4. Encrypt the message content with the session key.

The RecipientInfo blocks followed by the encrypted content constitute the envelopedData. This information is then encoded into base64. A sample message (excluding the RFC 822 headers) is the following:

```
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
      name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGT6rfvbnjT6jH7756tbB9H
f8HHGT6rfvHJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpF4
0GhIGfHfQbnj756YT64V
```

To recover the encrypted message, the recipient first strips off the base64 encoding. Then the recipient's private key is used to recover the session key. Finally, the message content is decrypted with the session key.

---

<sup>4</sup>This is an X.509 certificate, discussed later in this section.

**SignedData**

The signedData smime-type can actually be used with one or more signers. For clarity, we confine our description to the case of a single digital signature. The steps for preparing an envelopedData MIME entity are as follows:

1. Select a message digest algorithm (SHA or MD5).
2. Compute the message digest, or hash function, of the content to be signed.
3. Encrypt the message digest with the signer's private key.
4. Prepare a block known as SignerInfo that contains the signer's public-key certificate, an identifier of the message digest algorithm, an identifier of the algorithm used to encrypt the message digest, and the encrypted message digest.

The signedData entity consists of a series of blocks, including a message digest algorithm identifier, the message being signed, and SignerInfo. The signedData entity may also include a set of public-key certificates sufficient to constitute a chain from a recognized root or top-level certification authority to the signer. This information is then encoded into base64. A sample message (excluding the RFC 822 headers) is the following:

```
Content-Type: application/pkcs7-mime; smime-type=signed-data;
           name=smime.p7m
```

```
Content-Transfer-Encoding: base64
```

```
Content-Disposition: attachment; filename=smime.p7m
```

```
567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj7
77n8HHGT9HG4VQpfyF467GhIGfHfYT6rfvbnj756tbBghyHhHUujhJhjH
HUujhJh4VQpfyF467GhIGfHfYGT6rfvbnjT6jH7756tbB9H7n8HHGghyHh
6YT64V0GhIGfHfQbnj75
```

To recover the signed message and verify the signature, the recipient first strips off the base64 encoding. Then the signer's public key is used to decrypt the message digest. The recipient independently computes the message digest and compares it to the decrypted message digest to verify the signature.

**Clear Signing**

Clear signing is achieved using the multipart content type with a signed subtype. As was mentioned, this signing process does not involve transforming the message to be signed, so that the message is sent "in the clear." Thus, recipients with MIME capability but not S/MIME capability are able to read the incoming message.

A multipart/signed message has two parts. The first part can be any MIME type but must be prepared so that it will not be altered during transfer from source to destination. This means that if the first part is not 7bit, then it needs to be encoded using base64 or quoted-printable. Then this part is processed in the same manner as signedData, but in this case an object with signedData format is created

that has an empty message content field. This object is a detached signature. It is then transfer encoded using base64 to become the second part of the multipart/signed message. This second part has a MIME content type of application and a subtype of pkcs7-signature. Here is a sample message:

```
Content-Type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=sha1; boundary=boundary42

--boundary42
Content-Type: text/plain

This is a clear-signed message.

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpFyF4
7GhIGfHfYT64VQbnj756
--boundary42--
```

The protocol parameter indicates that this is a two-part clear-signed entity. The micalg parameter indicates the type of message digest used. The receiver can verify the signature by taking the message digest of the first part and comparing this to the message digest recovered from the signature in the second part.

### Registration Request

Typically, an application or user will apply to a certification authority for a public-key certificate. The application/pkcs10 S/MIME entity is used to transfer a certification request. The certification request includes certificationRequestInfo block, followed by an identifier of the public-key encryption algorithm, followed by the signature of the certificationRequestInfo block, made using the sender's private key. The certificationRequestInfo block includes a name of the certificate subject (the entity whose public key is to be certified) and a bit-string representation of the user's public key.

### Certificates-Only Message

A message containing only certificates or a certificate revocation list (CRL) can be sent in response to a registration request. The message is an application/pkcs7-mime type/subtype with an smime-type parameter of degenerate. The steps involved are the same as those for creating a signedData message, except that there is no message content and the signerInfo field is empty.

### S/MIME Certificate Processing

S/MIME uses public-key certificates that conform to version 3 of X.509 (see Chapter 11). The key-management scheme used by S/MIME is in some ways a hybrid between a strict X.509 certification hierarchy and PGP's web of trust. As with the PGP model, S/MIME managers and/or users must configure each client with a list of trusted keys and with certificate revocation lists. That is, the responsibility is local for maintaining the certificates needed to verify incoming signatures and to encrypt outgoing messages. On the other hand, the certificates are signed by certification authorities.

#### User Agent Role

An S/MIME user has several key-management functions to perform:

- **Key generation:** The user of some related administrative utility (e.g., one associated with LAN management) **MUST** be capable of generating separate Diffie-Hellman and DSS key pairs and **SHOULD** be capable of generating RSA key pairs. Each key pair **MUST** be generated from a good source of non-deterministic random input and be protected in a secure fashion. A user agent **SHOULD** generate RSA key pairs with a length in the range of 768 to 1024 bits and **MUST NOT** generate a length of less than 512 bits.
- **Registration:** A user's public key must be registered with a certification authority in order to receive an X.509 public-key certificate.
- **Certificate storage and retrieval:** A user requires access to a local list of certificates in order to verify incoming signatures and to encrypt outgoing messages. Such a list could be maintained by the user or by some local administrative entity on behalf of a number of users.

#### VeriSign Certificates

There are several companies that provide certification authority (CA) services. For example, Nortel has designed an enterprise CA solution and can provide S/MIME support within an organization. There are a number of Internet-based CAs, including VeriSign, GTE, and the U.S. Postal Service. Of these, the most widely used is the VeriSign CA service, a brief description of which we now provide.

VeriSign provides a CA service that is intended to be compatible with S/MIME and a variety of other applications. VeriSign issues X.509 certificates with the product name VeriSign Digital ID. As of early 1998, over 35,000 commercial Web sites were using VeriSign Server Digital IDs, and over a million consumer Digital IDs had been issued to users of Netscape and Microsoft browsers.

The information contained in a Digital ID depends on the type of Digital ID and its use. At a minimum, each Digital ID contains the following:

- Owner's public key
- Owner's name or alias
- Expiration date of the Digital ID
- Serial number of the Digital ID

- Name of the certification authority that issued the Digital ID
- Digital signature of the certification authority that issued the Digital ID

Digital IDs can also contain other user-supplied information, including

- Address
- E-mail address
- Basic registration information (country, zip code, age, and gender)

VeriSign provides three levels, or classes, of security for public-key certificates, as summarized in Table 12.8. A user requests a certificate on line at VeriSign's Web site or other participating Web sites. Class 1 and Class 2 requests are processed on

**Table 12.8** VeriSign Public-Key Certificate Classes

	Summary of Confirmation of Identity	IA Private-Key Protection	Certificate Applicant and Subscriber Private-Key Protection	Applications Implemented or Contemplated by Users
Class 1	Automated unambiguous name and e-mail address search	PCA: trustworthy hardware; CA: trustworthy software or trustworthy hardware	Encryption software (PIN protected) recommended but not required	Web browsing and certain e-mail usage
Class 2	Same as Class 1, plus automated enrollment information check plus automated address check	PCA and CA: trustworthy hardware	Encryption software (PIN protected) required	Individual and intra- and intercompany e-mail, on-line subscriptions, password replacement, and software validation
Class 3	Same as Class 1, plus personal presence and ID documents plus Class 2 automated ID check for individuals; business records (or filings) for organizations	PCA and CA: trustworthy hardware	Encryption software (PIN protected) required; hardware token recommended but not required	e-banking, corp. database access, personal banking, membership-based on-line services, content integrity services, e-commerce server, software validation; authentication of LRAAs; and strong encryption for certain servers

IA: Issuing Authority  
CA: Certification Authority  
PCA: VeriSign Public Primary Certification Authority  
PIN: Personal Identification Number  
LRAA: Local Registration Authority Administrator

line, and in most cases take only a few seconds to approve. Briefly, the following procedures are used:

- For Class 1 Digital IDs, VeriSign confirms the user's e-mail address by sending a PIN and Digital ID pick-up information to the e-mail address provided in the application.
- For Class 2 Digital IDs, VeriSign verifies the information in the application through an automated comparison with a consumer database in addition to performing all of the checking associated with a Class 1 Digital ID. Finally, confirmation is sent to the specified postal address alerting the user that a Digital ID has been issued in his or her name.
- For Class 3 Digital IDs, VeriSign requires a higher level of identity assurance. An individual must prove his or her identity by providing notarized credentials or applying in person.

### Enhanced Security Services

As of this writing, three enhanced security services have been proposed in an Internet draft. The details of these may change, and additional services may be added. The three services are as follows:

- **Signed receipts:** A signed receipt may be requested in a SignedData object. Returning a signed receipt provides proof of delivery to the originator of a message and allows the originator to demonstrate to a third party that the recipient received the message. In essence, the recipient signs the entire original message plus original (sender's) signature and appends the new signature to form a new S/MIME message.
- **Security labels:** A security label may be included in the authenticated attributes of a SignedData object. A security label is a set of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation. The labels may be used for access control, by indicating which users are permitted access to an object. Other uses include priority (secret, confidential, restricted, and so on) or role based, describing which kind of people can see the information (e.g., patient's health-care team, medical billing agents, etc.).
- **Secure mailing lists:** When a user sends a message to multiple recipients, a certain amount of per-recipient processing is required, including the use of each recipient's public key. The user can be relieved of this work by employing the services of an S/MIME Mail List Agent (MLA). An MLA can take a single incoming message, perform the recipient-specific encryption for each recipient, and forward the message. The originator of a message need only send the message to the MLA, with encryption performed using the MLA's public key.

## 12.3 RECOMMENDED READING

The following are recommended Web sites:

- **PGP Home Page:** PGP Web site by Network Associates, the leading PGP commercial vendor.

- **MIT Distribution Site for PGP:** Leading distributor of freeware PGP. Contains FAQ, other information, and links to other PGP sites.
- **S/MIME Charter:** Latest RFCs and internet drafts for S/MIME.
- **S/MIME Central:** RSA Inc.'s Web site for S/MIME. Includes FAQ and other useful information.

## 12.4 PROBLEMS

- 12.1** PGP makes use of the cipher feedback (CFB) mode of CAST-128, whereas most conventional encryption applications (other than key encryption) use the cipher block chaining (CBC) mode. We have

$$\text{CBC: } C_i = E_K[C_{i-1} \oplus P_i]; \quad P_i = C_{i-1} \oplus D_K[C_i]$$

$$\text{CFB: } C_i = P_i \oplus E_K[C_{i-1}]; \quad P_i = C_i \oplus E_K[C_{i-1}]$$

These two appear to provide equal security. Suggest a reason why PGP uses the CFB mode.

- 12.2** In the PGP scheme, what is the expected number of session keys generated before a previously created session key is produced?
- 12.3** In PGP, what is the probability that a user with  $N$  public keys will have at least one duplicate key ID?
- 12.4** The first 16 bits of the 128-bit message digest in a PGP signature are translated in the clear.
- To what extent does this compromise the security of the hash algorithm?
  - To what extent does it, in fact, perform its intended function—namely, to help determine if the correct RSA key was used to decrypt the digest?
- 12.5** In Figure 12.4, each entry in the public-key ring contains an owner trust field that indicates the degree of trust associated with this public-key owner. Why is that not enough? That is, if this owner is trusted and this is supposed to be the owner's public key, why is not that trust enough to permit PGP to use this public key?
- 12.6** Consider radix-64 conversion as a form of encryption. In this case, there is no key. But suppose that an opponent knew only that some form of substitution algorithm was being used to encrypt English text. How effective would this algorithm be against cryptanalysis?
- 12.7** Phil Zimmermann chose IDEA, three-key triple DES, and CAST-128 as conventional encryption algorithms for PGP. Give reasons why each of the other conventional encryption algorithms described in this book is suitable or unsuitable for PGP: DES, two-key triple DES, Blowfish, RC2, and RC5.

## APPENDIX 12A: DATA COMPRESSION USING ZIP

PGP makes use of a compression package called ZIP, written by Jean-loup Gailly, Mark Adler, and Richard Wales. ZIP is a freeware package written in C that runs as a utility on UNIX and some other systems. ZIP is functionally equivalent to PKZIP, a widely available shareware package for Windows systems developed by PKWARE, Inc. The zip algorithm is perhaps the most commonly used cross-platform compression technique; freeware and shareware versions are available for Macintosh and other systems as well as Windows and UNIX systems.

Zip and similar algorithms stem from research by Jacob Ziv and Abraham Lempel. In 1977, they described a technique based on a sliding window buffer that

holds the most recently processed text [ZIV77]. This algorithm is generally referred to as LZ77. A version of this algorithm is used in the zip compression scheme (PKZIP, gzip, zipit, etc.).

LZ77 and its variants exploit the fact that words and phrases within a text stream (image patterns in the case of GIF) are likely to be repeated. When a repetition occurs, the repeated sequence can be replaced by a short code. The compression program scans for such repetitions and develops codes on the fly to replace the repeated sequence. Over time, codes are reused to capture new sequences. The algorithm must be defined in such a way that the decompression program is able to deduce the current mapping between codes and sequences of source data.

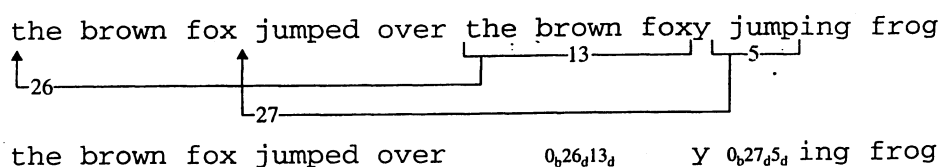
Before looking at the details of LZ77, let us look at a simple example.<sup>5</sup> Consider the nonsense phrase

the brown fox jumped over the brown foxy jumping frog

which is 53 octets = 424 bits long. The algorithm processes this text from left to right. Initially, each character is mapped into a 9-bit pattern consisting of a binary 1 followed by the 8-bit ASCII representation of the character. As the processing proceeds, the algorithm looks for repeated sequences. When a repetition is encountered, the algorithm continues scanning until the repetition ends. In other words, each time a repetition occurs, the algorithm includes as many characters as possible. The first such sequence encountered is the brown fox. This sequence is replaced by a pointer to the prior sequence and the length of the sequence. In this case the prior sequence of the brown fox occurs 26 character positions before and the length of the sequence is 13 characters. For this example, assume two options for encoding: an 8-bit pointer and a 4-bit length, or a 12-bit pointer and a 6-bit length; a 2-bit header indicates which option is chosen, with 00 indicating the first option and 01 the second option. Thus, the second occurrence of the brown fox is encoded as  $\langle 00_b \rangle \langle 26_d \rangle \langle 13_d \rangle$ , or 00 00011010 1101.

The remaining parts of the compressed message are the letter y; the sequence  $\langle 00_b \rangle \langle 27_d \rangle \langle 5_d \rangle$ , which replaces the sequence consisting of the space character followed by jump; and the character sequence ing frog.

Figure 12.9 illustrates the compression mapping. The compressed message consists of 35 9-bit characters and two codes, for a total of  $35 \times 9 + 2 \times 14 = 343$  bits. This compares with 424 bits in the uncompressed message for a compression ratio of 1.24.



**Figure 12.9** Example of LZ77 Scheme.

<sup>5</sup>Based on an example in [WEIS93].



### Compression Algorithm

The compression algorithm for LZ77 and its variants makes use of two buffers. A **sliding history buffer** contains the last  $N$  characters of source that have been processed, and a **look-ahead buffer** contains the next  $L$  characters to be processed (Figure 12.10a). The algorithm attempts to match two or more characters from the beginning of the look-ahead buffer to a string in the sliding history buffer. If no match is found, the first character in the look-ahead buffer is output as a 9-bit character and is also shifted into the sliding window, with the oldest character in the sliding window shifted out. If a match is found, the algorithm continues to scan for the longest match. Then the matched string is output as a triplet (indicator, pointer, length). For a  $K$ -character string, the  $K$  oldest characters in the sliding window are shifted out, and the  $K$  characters of the encoded string are shifted into the window.

Figure 12.10b shows the operation of this scheme on our example sequence. The illustration assumes a 39-character sliding window and a 13-character look-ahead buffer. In the upper part of the example, the first 40 characters have been processed and the uncompressed version of the most recent 39 of these characters is in the sliding window. The remaining source is in the look-ahead window. The compression algorithm determines the next match, shifts five characters from the look-ahead buffer into the sliding window, and outputs the code for this string. The state of the buffer after these operations is shown in the lower part of the example.

While LZ77 is effective and does adapt to the nature of the current input, it has some drawbacks. The algorithm uses a finite window to look for matches in previous text. For a very long block of text, compared to the size of the window, many potential matches are eliminated. The window size can be increased, but this imposes two penalties: (1) The processing time of the algorithm increases because

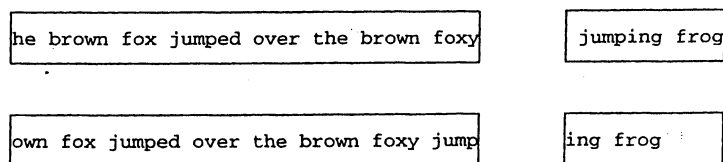
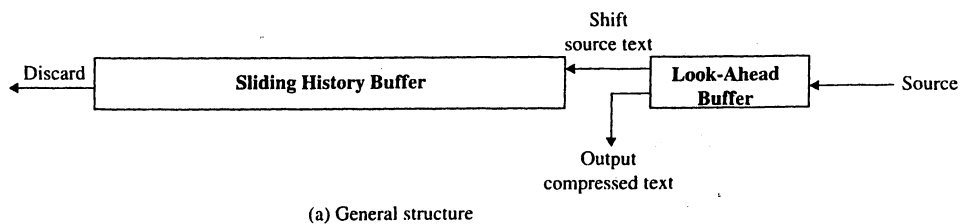


Figure 12.10 LZ77 Scheme.

it must perform a string comparison against the look-ahead buffer for every position in the sliding window, and (2) the <pointer> field must be larger to accommodate the longer jumps.

### Decompression Algorithm

Decompression of LZ77-compressed text is simple. The decompression algorithm must save the last  $N$  characters of decompressed output. When an encoded string is encountered, the decompression algorithm uses the <pointer> and <length> fields to replace the code with the actual text string.

## APPENDIX 12B: RADIX-64 CONVERSION

Both PGP and S/MIME make use of an encoding technique referred to as radix-64 conversion. This technique maps arbitrary binary input into printable character output. The form of encoding has the following relevant characteristics:

1. The range of the function is a character set that is universally representable at all sites, not a specific binary encoding of that character set. Thus, the characters themselves can be encoded into whatever form is needed by a specific system. For example, the character "E" is represented in an ASCII-based system as hexadecimal 45 and in an EBCDIC-based system as hexadecimal C5.
2. The character set consists of 65 printable characters, one of which is used for padding. With  $2^6 = 64$  available characters, each character can be used to represent 6 bits of input.
3. No control characters are included in the set. Thus, a message encoded in radix-64 can traverse mail-handling systems that scan the data stream for control characters.
4. The hyphen character ("-") is not used. This character has significance in the RFC 822 format and should therefore be avoided.

Table 12.9 shows the mapping of 6-bit input values to characters. The character set consists of the alphanumeric characters plus "+" and "/". The "=" character is used as the padding character.

Figure 12.11 illustrates the simple mapping scheme. Binary input is processed in blocks of 3 octets, or 24 bits. Each set of 6 bits in the 24-bit block is mapped into a character. In the figure, the characters are shown encoded as 8-bit quantities. In this typical case, each 24-bit input is expanded to 32 bits of output.

For example, consider the 24-bit raw text sequence 00100011 01011100 10010001, which can be expressed in hexadecimal as 235C91. We arrange this input in blocks of 6 bits:

```
001000 110101 110010 010001
```

The extracted 6-bit decimal values are 8, 53, 50, 17. Looking these up in Table 12.9 yields the radix-64 encoding as the following characters: IlyR. If these characters are stored in 8-bit ASCII format with parity bit set to zero, we have

**Table 12.9** Radix-64 Encoding

6-Bit value	Character encoding	6-Bit value	Character encoding	6-Bit value	Character encoding	6-Bit value	Character encoding
0	A	16	Q	32	g	48	* w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/
						(pad)	=

01001001 00110001 01111001 01010010

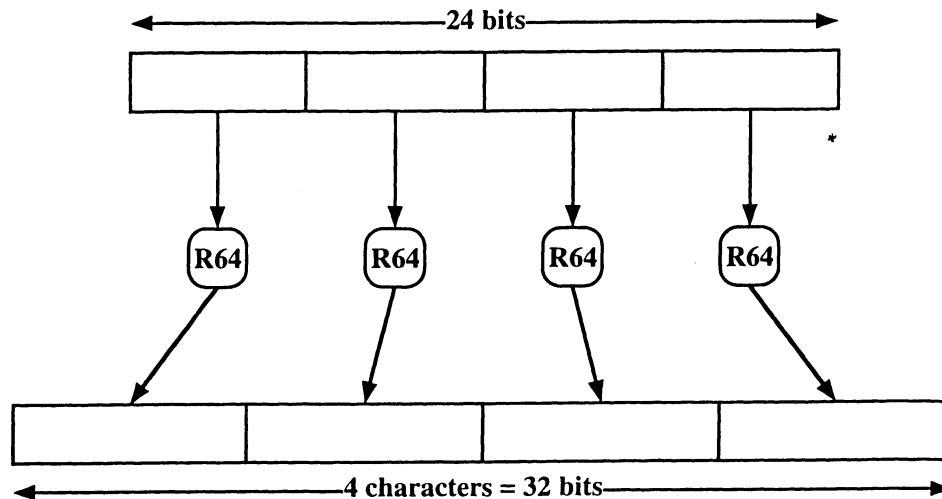
In hexadecimal, this is 49317952. To summarize,

Input Data	
Binary representation	00100011 01011100 10010001
Hexadecimal representation	235C91
Radix-64 Encoding of Input Data	
Character representation	IlyR
ASCII code (8 bit, zero parity)	01001001 00110001 01111001 01010010
Hexadecimal representation	49317952

**APPENDIX 12C: PGP RANDOM NUMBER GENERATION**

PGP uses a complex and powerful scheme for generating random numbers and pseudorandom numbers, for a variety of purposes. PGP generates random numbers from the content and timing of user keystrokes, and pseudorandom numbers using an algorithm based on the one in ANSI X12.17. PGP uses these numbers for the following purposes:

- True random numbers:
  - used to generate RSA key pairs
  - provides the initial seed for the pseudorandom number generator
  - provides additional input during pseudorandom number generation



**Figure 12.11** Printable Encoding of Binary Data into Radix-64 Format.

- Pseudorandom numbers:
  - used to generate session keys
  - used to generate initialization vectors (IVs) for use with the session key in CFB mode encryption

### True Random Numbers

PGP maintains a 256-byte buffer of random bits. Each time PGP expects a keystroke, it records the time, in 32-bit format, at which it starts waiting. When it receives the keystroke, it records the time the key was pressed and the 8-bit value of the keystroke. The time and keystroke information are used to generate a key, which is, in turn, used to encrypt the current value of the random-bit buffer.

### Pseudorandom Numbers

Pseudorandom number generation makes use of a 24-octet seed and produces a 16-octet session key, an 8-octet initialization vector, and a new seed to be used for the next pseudorandom number generation. The algorithm is based on the X12.17 algorithm described in Chapter 5 (see Figure 5.14) but uses CAST-128 instead of triple DES for encryption. The algorithm uses the following data structures:

#### 1. Input

- **randseed.bin** (24 octets): If this file is empty, it is filled with 24 true random octets.
- **message**: The session key and IV that will be used to encrypt a message are themselves a function of that message. This further contributes to the randomness of the key and IV, and if an opponent already knows the plaintext content of the message, there is no apparent need for capturing the one-time session key.



- G3. [Prepare to generate random octets]** Set  $\text{rcount} \leftarrow 0$  and  $k \leftarrow 23$ . The loop of steps G4–G7 will be executed 24 times ( $k = 23 \dots 0$ ), once for each random octet produced and placed in  $K$ . The variable  $\text{rcount}$  is the number of unused random octets in  $\text{rbuf}$ . It will count down from 8 to 0 three times to generate the 24 octets. \*
- G4. [Bytes available?]** If  $\text{rcount} = 0$  goto G5 else goto G7. Steps G5 and G6 perform one instance of the X12.17 algorithm to generate a new batch of eight random octets.
- G5. [Generate new random octets]**
- a.  $\text{rseed} \leftarrow \text{rseed} \oplus \text{dtbuf}$
  - b.  $\text{rbuf} \leftarrow E_{\text{rkey}}[\text{rseed}]$  in ECB mode
- G6. [Generate next seed]**
- a.  $\text{rseed} \leftarrow \text{rbuf} \oplus \text{dtbuf}$
  - b.  $\text{rseed} \leftarrow E_{\text{rkey}}[\text{rseed}]$  in ECB mode
  - c. Set  $\text{rcount} \leftarrow 8$
- G7 [Transfer one byte at a time from rbuf to K]**
- a. Set  $\text{rcount} \leftarrow \text{rcount} - 1$ .
  - b. Generate a true random byte  $b$ , and set  $K[k] \leftarrow \text{rbuf}[\text{rcount}] \oplus b$ .
- G8 [Done?]** If  $k = 0$  goto G9 else set  $k \leftarrow k - 1$  and goto G4.
- G9 [Postwash seed and return result]**
- a. Generate 24 more bytes by the method of steps G4–G7, except do not XOR in a random byte in G7. Place the result in buffer  $K'$ .
  - b. Encrypt  $K'$  with key  $K[0 \dots 15]$  and IV  $K[16 \dots 23]$  in CFB mode; store result in  $\text{randseed.bin}$ .
  - c. Return  $K$ .

It should not be possible to determine the session key from the 24 new octets generated in step G12.a. However, to make sure that the stored  $\text{randseed.bin}$  file provides no information about the most recent session key, the 24 new octets are encrypted and the result is stored as the new seed.

This elaborate algorithm should provide cryptographically strong pseudorandom numbers. A preliminary analysis of the algorithm indicates that there are no internal dependencies among the bits of a single session key and that successive session keys are independent [NEUH93].

# CHAPTER 13

## IP SECURITY

*If a secret piece of news is divulged by a spy before the time is ripe, he must be put to death, together with the man to whom the secret was told.*

—*The Art of War*, Sun Tzu

The Internet community has developed application-specific security mechanisms in a number of application areas, including electronic mail (S/MIME, PGP), client/server (Kerberos), Web access (Secure Sockets Layer), and others. However, users have some security concerns that cut across protocol layers. For example, an enterprise can run a secure, private TCP/IP network by disallowing links to untrusted sites, encrypting packets that leave the premises, and authenticating packets that enter the premises. By implementing security at the IP level, an organization can ensure secure networking not only for applications that have security mechanisms but for the many security-ignorant applications.

IP-level security encompasses three functional areas: authentication, confidentiality, and key management. The authentication mechanism assures that a received packet was, in fact, transmitted by the party identified as the source in the packet header. In addition, this mechanism assures that the packet has not been altered in transit. The confidentiality facility enables communicating nodes to encrypt messages to prevent eavesdropping by third parties. The key management facility is concerned with the secure exchange of keys.

We begin this chapter with an overview of IP security (IPSec) and an introduction to the IPSec architecture. We then look at each of the three functional areas in detail.<sup>1</sup> The appendix to this chapter reviews internet protocols.

---

<sup>1</sup>As of this writing, many of the IPSec specifications are in Internet Draft form and subject to change. Most of these documents have been through numerous iterations and should be quite stable, but some of the details may change.

## 13.1 IP SECURITY OVERVIEW

In 1994, the Internet Architecture Board (IAB) issued a report entitled “Security in the Internet Architecture” (RFC 1636). The report stated the general consensus that the Internet needs more and better security, and it identified key areas for security mechanisms. Among these were the need to secure the network infrastructure from unauthorized monitoring and control of network traffic and the need to secure end-user-to-end-user traffic using authentication and encryption mechanisms.

These concerns are fully justified. As confirmation, the 1997 annual report from the Computer Emergency Response Team (CERT) lists over 2500 reported security incidents affecting nearly 150,000 sites. The most serious types of attacks included IP spoofing, in which intruders create packets with false IP addresses and exploit applications that use authentication based on IP; and various forms of eavesdropping and packet sniffing, in which attackers read transmitted information, including logon information and database contents.

In response to these issues, the IAB included authentication and encryption as necessary security features in the next-generation IP, which has been issued as IPv6. Fortunately, these security capabilities were designed to be usable both with the current IPv4 and the future IPv6. This means that vendors can begin offering these features now, and many vendors do now have some IPSec capability in their products.

### Applications of IPSec<sup>2</sup>

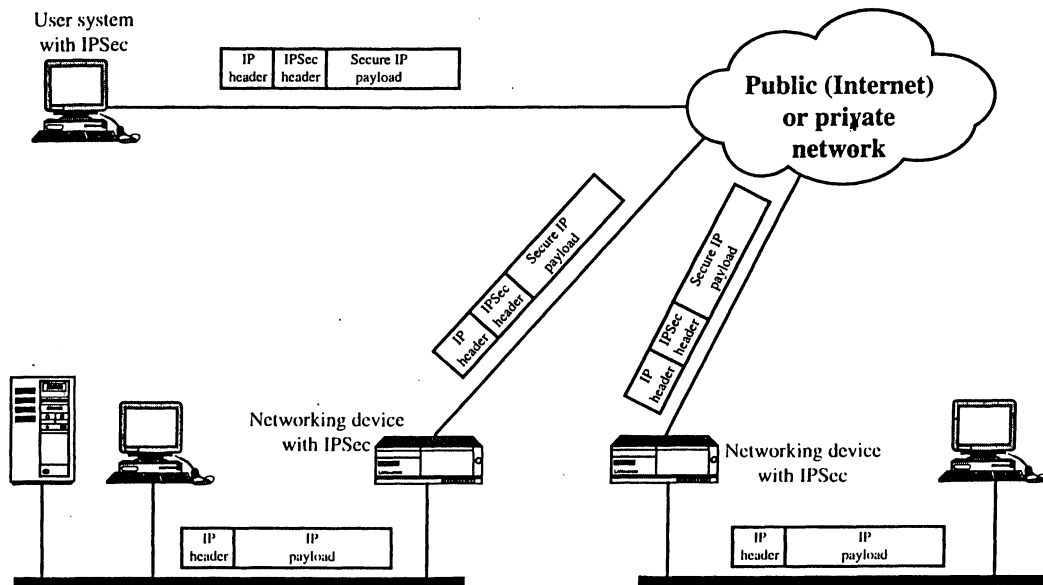
IPSec provides the capability to secure communications across a LAN, across private and public wide area networks (WANs), and across the Internet. Examples of its use include the following:

- **Secure branch office connectivity over the Internet:** A company can build a secure virtual private network over the Internet or over a public WAN. This enables a business to rely heavily on the Internet and reduce its need for private networks, saving costs and network management overhead.
- **Secure remote access over the Internet:** An end user whose system is equipped with IP security protocols can make a local call to an Internet service provider (ISP) and gain secure access to a company network. This reduces the cost of toll charges for traveling employees and telecommuters.
- **Establishing extranet and intranet connectivity with partners:** IPSec can be used to secure communication with other organizations, ensuring authentication and confidentiality and providing a key exchange mechanism.
- **Enhancing electronic commerce security:** Even though some Web and electronic commerce applications have built-in security protocols, the use of IPSec enhances that security.

The principal feature of IPSec that enables it to support these varied applications is that it can encrypt and/or authenticate *all* traffic at the IP level. Thus, all

<sup>2</sup>This subsection is based on material in *IP Security Whitepaper*, from CyLAN Technologies, <http://www.cylan.com/files/whpaper.htm>, 1997.





**Figure 13.1** An IP Security Scenario.

distributed applications, including remote logon, client/server, e-mail, file transfer, Web access, and so on, can be secured.

Figure 13.1 is a typical scenario of IPSec usage. An organization maintains LANs at dispersed locations. Nonsecure IP traffic is conducted on each LAN. For traffic off site, through some sort of private or public WAN, IPSec protocols are used. These protocols operate in networking devices, such as a router or firewall, that connect each LAN to the outside world. The IPSec networking device will typically encrypt and compress all traffic going into the WAN, and decrypt and decompress traffic coming from the WAN; these operations are transparent to workstations and servers on the LAN. Secure transmission is also possible with individual users who dial into the WAN. Such user workstations must implement the IPSec protocols to provide security.

### Benefits of IPSec

[MARK97] lists the following benefits of IPSec:

- When IPSec is implemented in a firewall or router, it provides strong security that can be applied to all traffic crossing the perimeter. Traffic within a company or workgroup does not incur the overhead of security-related processing.
- IPSec in a firewall is resistant to bypass if all traffic from the outside must use IP and the firewall is the only means of entrance from the Internet into the organization.
- IPSec is below the transport layer (TCP, UDP) and so is transparent to applications. There is no need to change software on a user or server system when

IPSec is implemented in the firewall or router. Even if IPSec is implemented in end systems, upper-layer software, including applications, is not affected.

- IPSec can be transparent to end users. There is no need to train users on security mechanisms, issue keying material on a per-user basis, or revoke keying material when users leave the organization.
- IPSec can provide security for individual users if needed. This is useful for off-site workers and for setting up a secure virtual subnetwork within an organization for sensitive applications.

### Routing Applications

In addition to supporting end users and protecting premises systems and networks, IPSec can play a vital role in the routing architecture required for internetworking. [HUIT98] lists the following examples of the use of IPSec. IPSec can assure that

- A router advertisement (a new router advertises its presence) comes from an authorized router.
- A neighbor advertisement (a router seeks to establish or maintain a neighbor relationship with a router in another routing domain) comes from an authorized router.
- A redirect message comes from the router to which the initial packet was sent.
- A routing update is not forged.

Without such security measures, an opponent can disrupt communications or divert some traffic. Routing protocols such as OSPF should be run on top of security associations between routers that are defined by IPSec.

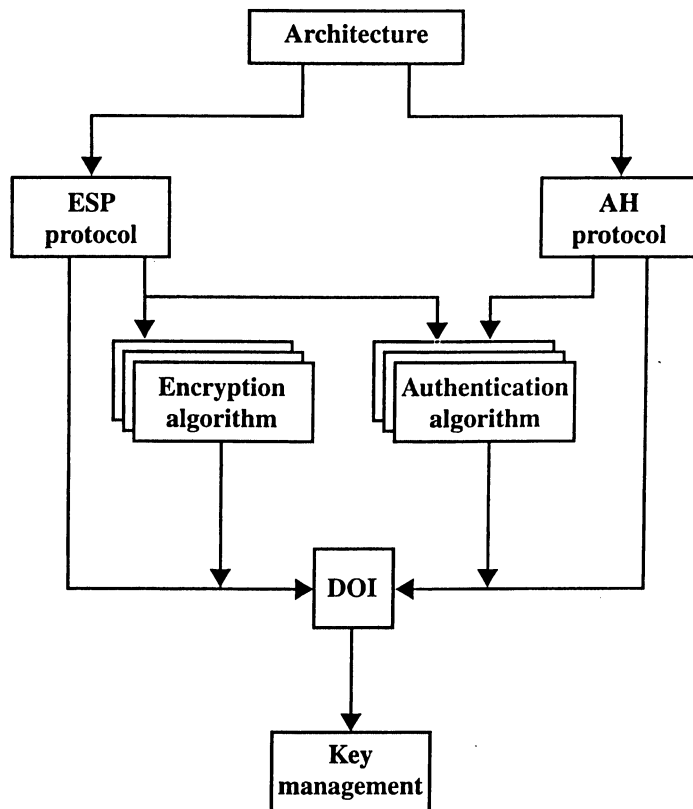
## 13.2 IP SECURITY ARCHITECTURE

The IPSec specification has become quite complex. To get a feel for the overall architecture, we begin with a look at the documents that define IPSec. Then we discuss IPSec services and introduce the concept of security association.

### IPSec Documents

In August 1995, the IETF published five security-related Proposed Standards that define a security capability at the internet level:

- RFC 1825: An overview of a security architecture
- RFC 1826: Description of a packet authentication extension to IP
- RFC 1828: A specific authentication mechanism
- RFC 1827: Description of a packet encryption extension to IP
- RFC 1829: A specific encryption mechanism



**Figure 13.2** IPsec Document Overview.

Support for these features is mandatory for IPv6 and optional for IPv4. In both cases, the security features are implemented as extension headers that follow the main IP header. The extension header for authentication is known as the Authentication header; that for encryption is known as the Encapsulating Security Payload (ESP) header.

Since this initial set of documents, a great deal of work has been done within the IP Security Protocol Working Group set up by the IETF. The documents are divided into seven groups, as depicted in Figure 13.2:

- **Architecture:** Covers the general concepts, security requirements, definitions, and mechanisms defining IPsec technology.
- **Encapsulating Security Payload (ESP):** Covers the packet format and general issues related to the use of the ESP for packet encryption and, optionally, authentication.
- **Authentication Header (AH):** Covers the packet format and general issues related to the use of AH for packet authentication.
- **Encryption Algorithm:** A set of documents that describe how various encryption algorithms are used for ESP.

- **Authentication Algorithm:** A set of documents that describe how various authentication algorithms are used for AH and for the authentication option of ESP.
- **Key Management:** Documents that describe key management schemes.
- **Domain of Interpretation (DOI):** Contains values needed for the other documents to relate to each other. These include identifiers for approved encryption and authentication algorithms, as well as operational parameters such as key lifetime.

### IPSec Services

IPSec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. Two protocols are used to provide security: an authentication protocol designated by the header of the protocol, Authentication Header (AH); and a combined encryption/authentication protocol designated by the format of the packet for that protocol, Encapsulating Security Payload (ESP). The services are as follows:

- Access control
- Connectionless integrity
- Data origin authentication
- Rejection of replayed packets (a form of partial sequence integrity)
- Confidentiality (encryption)
- Limited traffic flow confidentiality

Table 13.1 shows which services are provided by the AH and ESP protocols. For ESP, there are two cases: with and without the authentication option. Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols.

### Security Associations

A key concept that appears in both the authentication and confidentiality mechanisms for IP is the security association (SA). An association is a one-way relationship between a sender and a receiver that affords security services to the traffic

**Table 13.1** IPSec Services

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✓	✓	✓
Connectionless integrity	✓		✓
Data origin authentication	✓		✓
Rejection of replayed packets	✓	✓	✓
Confidentiality		✓	✓
Limited traffic flow confidentiality		✓	✓

carried on it. If a peer relationship is needed, for two-way secure exchange, then two security associations are required. Security services are afforded to an SA for the use of AH or ESP, but not both.

A security association is uniquely identified by three parameters:

- **Security Parameters Index (SPI):** A bit string assigned to this SA and having local significance only. The SPI is carried in AH and ESP headers to enable the receiving system to select the SA under which a received packet will be processed.
- **IP Destination Address:** Currently, only unicast addresses are allowed; this is the address of the destination endpoint of the SA, which may be an end user system or a network system such as a firewall or router.
- **Security Protocol Identifier:** This indicates whether the association is an AH or ESP security association.

Hence, in any IP packet,<sup>3</sup> the security association is uniquely identified by the Destination Address in the IPv4 or IPv6 header and the SPI in the enclosed extension header (AH or ESP).

#### SA Parameters

In each IPsec implementation, there is a nominal<sup>4</sup> Security Association Database that defines the parameters associated with each SA. A security association is normally defined by the following parameters:

- **Sequence Number Counter:** A 32-bit value used to generate the Sequence Number field in AH or ESP headers, described in Section 13.3 (required for all implementations).
- **Sequence Counter Overflow:** A flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA (required for all implementations).
- **Anti-Replay Window:** Used to determine whether an inbound AH or ESP packet is a replay, described in Section 13.3 (required for all implementations).
- **AH Information:** Authentication algorithm, keys, key lifetimes, and related parameters being used with AH (required for AH implementations).
- **ESP Information:** Encryption and authentication algorithm, keys, initialization values, key lifetimes, and related parameters being used with ESP (required for ESP implementations).
- **Lifetime of this Security Association:** A time interval or byte count after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur (required for all implementations).

<sup>3</sup>In this chapter, the term *IP packet* refers to either an IPv4 datagram or an IPv6 packet.

<sup>4</sup>Nominal in the sense that the functionality provided by a Security Association Database must be present in any IPsec implementation, but the way in which that functionality is provided is up to the implementor.

- **IPSec Protocol Mode:** Tunnel, transport, or wildcard (required for all implementations). These modes are discussed later in this section.
- **Path MTU:** Any observed path maximum transmission unit (maximum size of a packet that can be transmitted without fragmentation) and aging variables (required for all implementations).

The key management mechanism that is used to distribute keys is coupled to the authentication and privacy mechanisms only by way of the Security Parameters Index. Hence, authentication and privacy have been specified independent of any specific key management mechanism.

### SA Selectors

IPSec provides the user with considerable flexibility in the way in which IPSec services are applied to IP traffic. As we will see later, SAs can be combined in a number of ways to yield the desired user configuration. Furthermore, IPSec provides a high degree of granularity in discriminating between traffic that is afforded IPSec protection and traffic that is allowed to bypass IPSec, in the former case relating IP traffic to specific SAs.

The means by which IP traffic is related to specific SAs (or no SA in the case of traffic allowed to bypass IPSec) is the nominal Security Policy Database (SPD). In its simplest form, an SPD contains entries, each of which defines a subset of IP traffic and points to an SA for that traffic. In more complex environments, there may be multiple entries that potentially relate to a single SA or multiple SAs associated with a single SPD entry. The reader is referred to the relevant IPSec documents for a full discussion.

Each SPD entry is defined by a set of IP and upper-layer protocol field values, called *selectors*. In effect, these selectors are used to filter outgoing traffic in order to map it into a particular SA. Outbound processing obeys the following general sequence for each IP packet:

1. Compare the values of the appropriate fields in the packet (the selector fields) against the SPD to find a matching SPD entry, which will point to zero or more SAs.
2. Determine the SA if any for this packet and its associated SPI.
3. Do the required IPSec processing (i.e., AH or ESP processing).

The following selectors determine an SPD entry:

- **Destination IP Address:** This may be a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address. The latter two are required to support more than one destination system sharing the same SA (e.g., behind a firewall).
- **Source IP Address:** This may be a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address. The latter two are required to support more than one source system sharing the same SA (e.g., behind a firewall).

- **UserID:** A user identifier from the operating system. This is not a field in the IP or upper-layer headers but is available if IPsec is running on the same operating system as the user.
- **Data Sensitivity Level:** Used for systems providing information flow security (e.g., Secret or Unclassified).
- **Transport Layer Protocol:** Obtained from the IPv4 Protocol or IPv6 Next Header field. This may be an individual protocol number, a list of protocol numbers, or a range of protocol numbers.
- **IPsec Protocol (AH or ESP or AH/ESP):** If present, this is obtained from the IPv4 Protocol or IPv6 Next Header field.
- **Source and Destination Ports:** These may be individual TCP or UDP port values, an enumerated list of ports, or a wildcard port.
- **IPv6 Class:** Obtained from the IPv6 header. This may be a specific IPv6 Class value or a wildcard value.
- **IPv6 Flow Label:** Obtained from the IPv6 header. This may be a specific IPv6 Flow Label value or a wildcard value.
- **IPv4 Type of Service (TOS):** Obtained from the IPv4 header. This may be a specific IPv4 TOS value or a wildcard value.

### Transport and Tunnel Modes

Both AH and ESP support two modes of use: transport and tunnel mode. The operation of these two modes is best understood in the context of a description of AH and ESP, which are covered in Sections 13.3 and 13.4, respectively. Here we provide a brief overview.

#### Transport Mode

Transport mode provides protection primarily for upper-layer protocols. That is, transport mode protection extends to the payload of an IP packet. Examples include a TCP or UDP segment or an Internet Control Message Protocol (ICMP) packet, all of which operate directly above IP in a host protocol stack. Typically, transport mode is used for end-to-end communication between two hosts (e.g., a client and a server, or two workstations). When a host runs AH or ESP over IPv4, the payload is the data that normally follow the IP header. For IPv6, the payload is the data that normally follow both the IP header and any IPv6 extensions headers that are present, with the possible exception of the destination options header, which may be included in the protection.

ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header. AH in transport mode authenticates the IP payload and selected portions of the IP header.

#### Tunnel Mode

Tunnel mode provides protection to the entire IP packet. To achieve this, after the AH or ESP fields are added to the IP packet, the entire packet plus security fields is treated as the payload of new “outer” IP packet with a new outer IP header. The entire original, or inner, packet travels through a “tunnel” from one point of an

**Table 13.2** Tunnel Mode and Transport Mode Functionality

	Transport Mode SA	Tunnel Mode SA
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts inner IP packet. Authenticates inner IP packet.

IP network to another; no routers along the way are able to examine the inner IP header. Because the original packet is encapsulated, the new, larger packet may have totally different source and destination addresses, adding to the security. Tunnel mode is used when one or both ends of an SA is a security gateway, such as a firewall or router that implements IPSec. With tunnel mode, a number of hosts on networks behind firewalls may engage in secure communications without implementing IPSec. The unprotected packets generated by such hosts are tunneled through external networks by tunnel mode SAs set up by the IPSec software in the firewall or secure router at the boundary of the local network.

Here is an example of how tunnel mode IPSec operates. Host A on a network generates an IP packet with the destination address of host B on another network. This packet is routed from the originating host to a firewall or secure router at the boundary of A's network. The firewall filters all outgoing packets to determine the need for IPSec processing. If this packet from A to B requires IPSec, the firewall performs IPSec processing and encapsulates the packet in an outer IP header. The source IP address of this outer IP packet is this firewall, and the destination address may be a firewall that forms the boundary to B's local network. This packet is now routed to B's firewall, with intermediate routers examining only the outer IP header. At B's firewall, the outer IP header is stripped off, and the inner packet is delivered to B.

ESP in tunnel mode encrypts and optionally authenticates the entire inner IP packet, including the inner IP header. AH in tunnel mode authenticates the entire inner IP packet and selected portions of the outer IP header.

Table 13.2 summarizes transport and tunnel mode functionality.

### 13.3 AUTHENTICATION HEADER

The Authentication Header provides support for data integrity and authentication of IP packets. The data integrity feature ensures that undetected modification to a packet's content in transit is not possible. The authentication feature enables an end



system or network device to authenticate the user or application and filter traffic accordingly; it also prevents the address spoofing attacks observed in today's Internet. The AH also guards against the replay attack described later in this section.

Authentication is based on the use of a message authentication code (MAC), as described in Chapter 8; hence the two parties must share a secret key.\*

The Authentication Header consists of the following fields (Figure 13.3):

- **Next Header (8 bits):** Identifies the type of header immediately following this header.
- **Payload Length (8 bits):** Length of Authentication Header in 32-bit words, minus 2. For example, the default length of the authentication data field is 96 bits, or three 32-bit words. With a three-word fixed header, there are a total of six words in the header, and the Payload Length field has a value of 4.
- **Reserved (16 bits):** For future use.
- **Security Parameters Index (32 bits):** Identifies a security association.
- **Sequence Number (32 bits):** A monotonically increasing counter value, discussed later.
- **Authentication Data (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value (ICV), or MAC, for this packet, discussed later.

### Anti-Replay Service

A replay attack is one in which an attacker obtains a copy of an authenticated packet and later transmits it to the intended destination. The receipt of duplicate, authenticated IP packets may disrupt service in some way or may have some other undesired consequence. The Sequence Number field is designed to thwart such attacks. First, we discuss sequence number generation by the sender, and then we look at how it is processed by the recipient.

When a new SA is established, the **sender** initializes a sequence number counter to 0. Each time that a packet is sent on this SA, the sender increments the counter and places the value in the Sequence Number field. Thus, the first value to

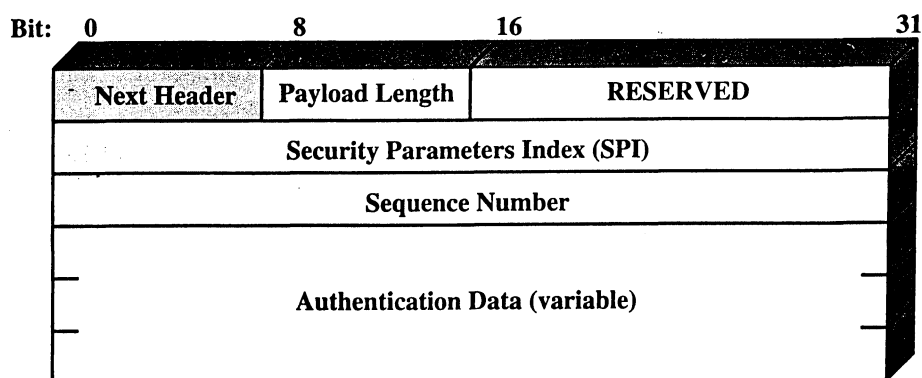
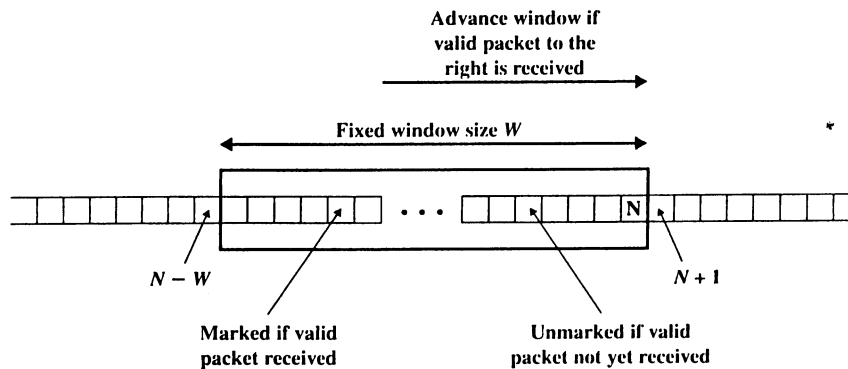


Figure 13.3 IPsec Authentication Header.



**Figure 13.4** Anti-Replay Mechanism.

be used is 1. If anti-replay is enabled (the default), the sender must not allow the sequence number to cycle past  $2^{32} - 1$  back to zero. Otherwise, there would be multiple valid packets with the same sequence number. If the limit of  $2^{32} - 1$  is reached, the sender should terminate this SA and negotiate a new SA with a new key.

Because IP is a connectionless, unreliable service, the protocol does not guarantee that packets will be delivered in order and does not guarantee that all packets will be delivered. Therefore, the IPSec authentication document dictates that the **receiver** should implement a window of size  $W$ , with a default of  $W = 64$ . The right edge of the window represents the highest sequence number,  $N$ , so far received for a valid packet. For any packet with a sequence number in the range from  $N - W + 1$  to  $N$  that has been correctly received (i.e., properly authenticated), the corresponding slot in the window is marked (Figure 13.4). Inbound processing proceeds as follows when a packet is received:

1. If the received packet falls within the window and is new, the MAC is checked. If the packet is authenticated, the corresponding slot in the window is marked.
2. If the received packet is to the right of the window and is new, the MAC is checked. If the packet is authenticated, the window is advanced so that this sequence number is the right edge of the window, and the corresponding slot in the window is marked.
3. If the received packet is to the left of the window, or if authentication fails, the packet is discarded; this is an auditable event.

### Integrity Check Value

The Authentication Data field holds a value referred to as the Integrity Check Value. The ICV is a message authentication code or a truncated version of a code produced by a MAC algorithm. The current specification dictates that a compliant implementation must support

- HMAC-MD5-96
- HMAC-SHA-1-96

Both of these use the HMAC algorithm, the first with the MD5 hash code and the second with the SHA-1 hash code (all of these algorithms are described in Chapter 9). In both cases, the full HMAC value is calculated but then truncated by using the first 96 bits, which is the default length for the Authentication Data field.

The MAC is calculated over the following:

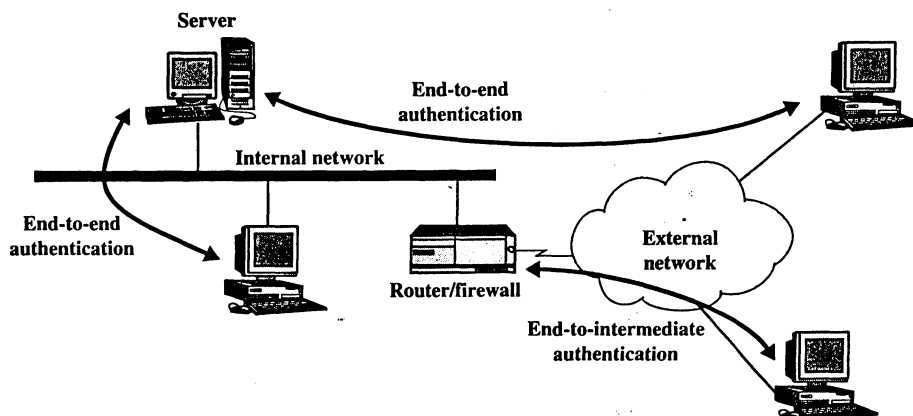
- IP header fields that either do not change in transit (immutable) or that are predictable in value upon arrival at the endpoint for the AH SA. Fields that may change in transit and whose value on arrival are unpredictable are set to zero for purposes of calculation at both source and destination.
- The AH header other than the Authentication Data field. The Authentication Data field is set to zero for purposes of calculation at both source and destination.
- The entire upper-level protocol data, which is assumed to be immutable in transit (e.g., a TCP segment or an inner IP packet in tunnel mode).

For IPv4, examples of immutable fields are Internet Header Length and Source Address. An example of a mutable but predictable field is the Destination Address (with loose or strict source routing). Examples of mutable fields that are zeroed prior to ICV calculation are the Time to Live and Header Checksum fields. Note that both source and destination address fields are protected, so that address spoofing is prevented.

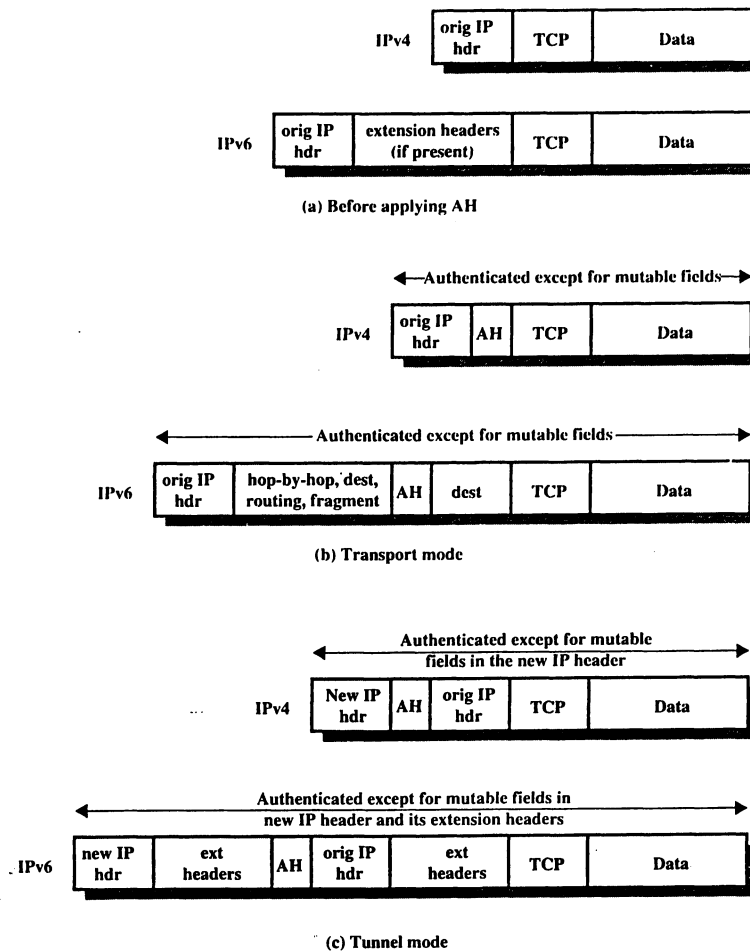
For IPv6, examples in the base header are Version (immutable), Destination Address (mutable but predictable), and Flow Label (mutable and zeroed for calculation).

### Transport and Tunnel Modes

Figure 13.5 shows two ways in which the IPSec authentication service can be used. In one case, authentication is provided directly between a server and client workstations; the workstation can be either on the same network as the server or on an



**Figure 13.5** End-to-end versus End-to-intermediate Authentication.



**Figure 13.6** Scope of AH Authentication.

external network. As long as the workstation and the server share a protected secret key, the authentication process is secure. This case uses a transport mode SA. In the other case, a remote workstation authenticates itself to the corporate firewall, either for access to the entire internal network or because the requested server does not support the authentication feature. This case uses a tunnel mode SA.

In this subsection, we look at the scope of authentication provided by AH and the authentication header location for the two modes. The considerations are somewhat different for IPv4 and IPv6. Figure 13.6a shows typical IPv4 and IPv6 packets. In this case, the IP payload is a TCP segment; it could also be a data unit for any other protocol that uses IP, such as UDP or ICMP.

For **transport mode AH** using IPv4, the AH is inserted after the original IP header and before the IP payload (e.g., a TCP segment); this is shown in the upper

part of Figure 13.6b. Authentication covers the entire packet, excluding mutable fields in the IPv4 header that are set to zero for MAC calculation.

In the context of IPv6, AH is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the AH appears after the IPv6 base header and the hop-by-hop, routing, and fragment extension headers. The destination options extension header could appear before or after the AH header, depending on the semantics desired. Again, authentication covers the entire packet, excluding mutable fields that are set to zero for MAC calculation.

For **tunnel mode AH**, the entire original IP packet is authenticated, and the AH is inserted between the original IP header and a new outer IP header (Figure 13.6c). The inner IP header carries the ultimate source and destination addresses, while an outer IP header may contain different IP addresses (e.g., addresses of firewalls or other security gateways).

With tunnel mode, the entire inner IP packet, including the entire inner IP header, is protected by AH. The outer IP header (and, in the case of IPv6, the outer IP extension headers) is protected except for mutable and unpredictable fields.

## 13.4 ENCAPSULATING SECURITY PAYLOAD

The Encapsulating Security Payload provides confidentiality services, including confidentiality of message contents and limited traffic flow confidentiality. As an optional feature, ESP can also provide the same authentication services as AH.

### ESP Format

Figure 13.7 shows the format of an ESP packet. It contains the following fields:

- **Security Parameters Index (32 bits):** Identifies a security association.
- **Sequence Number (32 bits):** A monotonically increasing counter value; this provides an anti-replay function, as discussed for AH.
- **Payload Data (variable):** This is a transport-level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption.
- **Padding (0–255 bytes):** The purpose of this field is discussed later.
- **Pad Length (8 bits):** Indicates the number of pad bytes immediately preceding this field.
- **Next Header (8 bits):** Identifies the type of data contained in the payload data field by identifying the first header in that payload (for example, an extension header in IPv6, or an upper-layer protocol such as TCP).
- **Authentication Data (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value computed over the ESP packet minus the Authentication Data field.

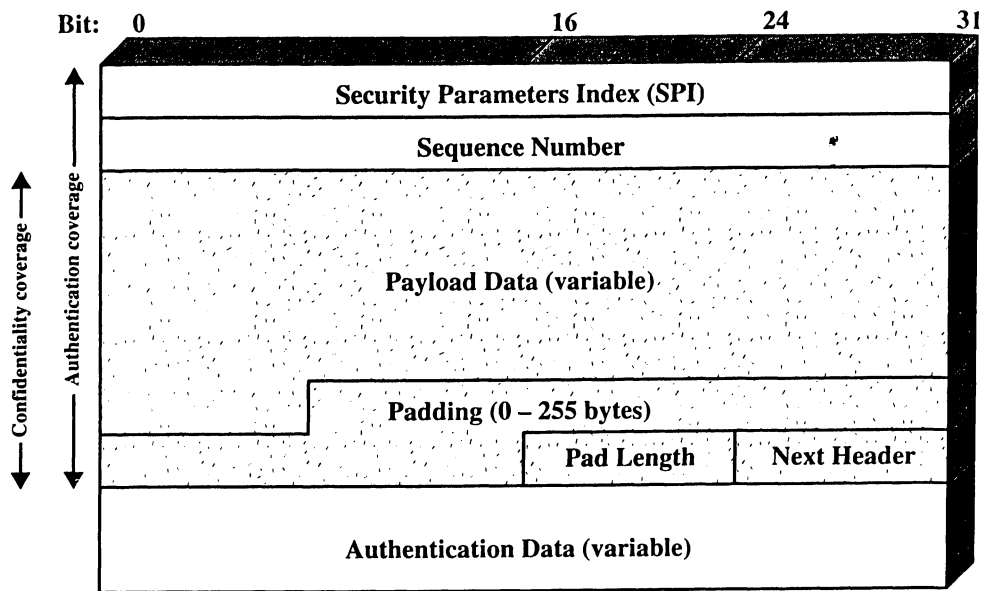


Figure 13.7 IPSec ESP Format.

### Encryption and Authentication Algorithms

The Payload Data, Padding, Pad Length, and Next Header fields are encrypted by the ESP service. If the algorithm used to encrypt the payload requires cryptographic synchronization data, such as an initialization vector (IV), then this data may be carried explicitly at the beginning of the Payload Data field. If included, an IV is usually not encrypted, although it is often referred to as being part of the ciphertext.

The current specification dictates that a compliant implementation must support DES in cipher block chaining (CBC) mode (described in Chapter 3). A number of other algorithms have been assigned identifiers in the DOI document and could therefore easily be used for encryption. These include the following:

- Three-key triple DES
- RC5
- IDEA
- Three-key triple IDEA
- CAST
- Blowfish

All of these algorithms are described in Chapter 4.

As with AH, ESP supports the use of a MAC with a default length of 96 bits. Also as with AH, the current specification dictates that a compliant implementation must support HMAC-MD5-96 and HMAC-SHA-1-96.

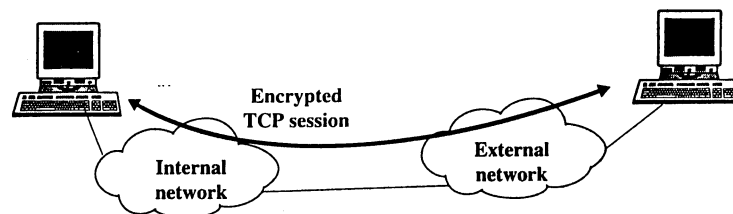
## Padding

The Padding field serves several purposes:

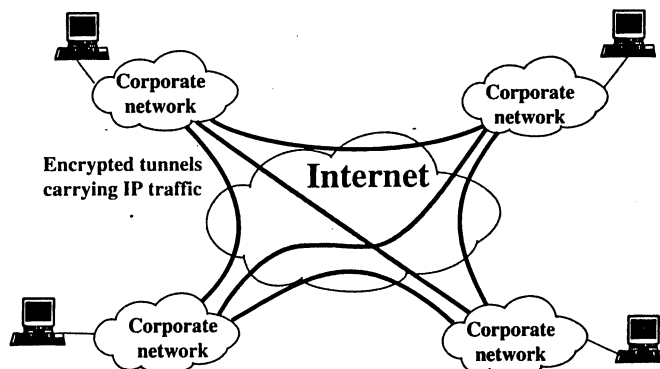
- If an encryption algorithm requires the plaintext to be a multiple of some number of bytes (e.g., the multiple of a single block for a block cipher), the Padding field is used to expand the plaintext (consisting of the Payload Data, Padding, Pad Length, and Next Header fields) to the required length.
- The ESP format requires that the Pad Length and Next Header fields be right aligned within a 32-bit word. Equivalently, the ciphertext must be an integer multiple of 32 bits. The Padding field is used to assure this alignment.
- Additional padding may be added to provide partial traffic flow confidentiality by concealing the actual length of the payload.

## Transport and Tunnel Modes

Figure 13.8 shows two ways in which the IPSec ESP service can be used. In the upper part of the figure, encryption (and optionally authentication) is provided directly between two hosts. Figure 13.8b shows how tunnel mode operation can be



(a) Transport-level security



(b) A virtual private network via tunnel mode

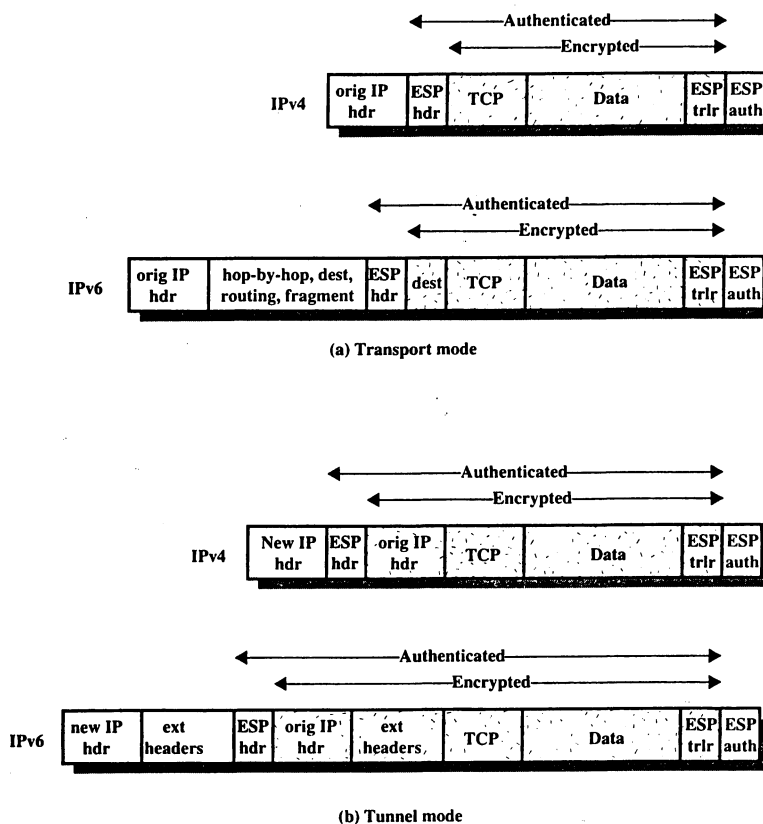
**Figure 13.8** Transport Mode versus Tunnel Mode Encryption.

used to set up a *virtual private network*. In this example, an organization has four private networks interconnected across the Internet. Hosts on the internal networks use the Internet for transport of data but do not interact with other Internet-based hosts. By terminating the tunnels at the security gateway to each internal network, the configuration allows the hosts to avoid implementing the security capability. The former technique is support by a transport mode SA, while the latter technique uses a tunnel mode SA.

In this section, we look at the scope of ESP for the two modes. The considerations are somewhat different for IPv4 and IPv6. As with our discussion of AH scope, we will use the packet formats of Figure 13.6a as a starting point.

### Transport Mode ESP

Transport mode ESP is used to encrypt and optionally authenticate the data carried by IP (e.g., a TCP segment), as shown in Figure 13.9a. For this mode using IPv4, the ESP header is inserted into the IP packet immediately prior to the transport-layer header (e.g., TCP, UDP, ICMP) and an ESP trailer (Padding, Pad Length, and Next Header fields) is placed after the IP packet; if authentication is selected, the ESP Authentication Data field is added after the ESP trailer. The



**Figure 13.9** Scope of ESP Encryption and Authentication.



entire transport-level segment plus the ESP trailer are encrypted. Authentication covers all of the ciphertext plus the ESP header.

In the context of IPv6, ESP is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the ESP header appears after the IPv6 base header and the hop-by-hop, routing, and fragment extension headers. The destination options extension header could appear before or after the ESP header, depending on the semantics desired. For IPv6, encryption covers the entire transport-level segment plus the ESP trailer plus the destination options extension header if it occurs after the ESP header. Again, authentication covers the ciphertext plus the ESP header.

Transport mode operation may be summarized as follows:

1. At the source, the block of data consisting of the ESP trailer plus the entire transport-layer segment is encrypted and the plaintext of this block is replaced with its ciphertext to form the IP packet for transmission. Authentication is added if this option is selected.
2. The packet is then routed to the destination. Each intermediate router needs to examine and process the IP header plus any plaintext IP extension headers, but does not need to examine the ciphertext.
3. The destination node examines and processes the IP header plus any plaintext IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext transport-layer segment.

Transport mode operation provides confidentiality for any application that uses it, thus avoiding the need to implement confidentiality in every individual application. This mode of operation is also reasonably efficient, adding little to the total length of the IP packet. One drawback to this mode is that it is possible to do traffic analysis on the transmitted packets.

#### **Tunnel Mode ESP**

Tunnel mode ESP is used to encrypt an entire IP packet (Figure 13.9b). For this mode, the ESP header is prefixed to the packet and then the packet plus the ESP trailer is encrypted. This method can be used to counter traffic analysis.

Because the IP header contains the destination address and possibly source routing directives and hop-by-hop option information, it is not possible simply to transmit the encrypted IP packet prefixed by the ESP header. Intermediate routers would be unable to process such a packet. Therefore, it is necessary to encapsulate the entire block (ESP header plus ciphertext plus Authentication Data, if present) with a new IP header that will contain sufficient information for routing but not for traffic analysis.

Whereas the transport mode is suitable for protecting connections between hosts that support the ESP feature, the tunnel mode is useful in a configuration that includes a firewall or other sort of security gateway that protects a trusted network from external networks. In this latter case, encryption occurs only between an external host and the security gateway or between two security gateways. This relieves hosts on the internal network of the processing burden of encryption and simplifies

the key distribution task by reducing the number of needed keys. Further, it thwarts traffic analysis based on ultimate destination.

Consider a case in which an external host wishes to communicate with a host on an internal network protected by a firewall, and in which ESP is implemented in the external host and the firewalls. The following steps occur for transfer of a transport-layer segment from the external host to the internal host:

1. The source prepares an inner IP packet with a destination address of the target internal host. This packet is prefixed by an ESP header; then the packet and ESP trailer are encrypted and Authentication Data may be added. The resulting block is encapsulated with a new IP header (base header plus optional extensions such as routing and hop-by-hop options for IPv6) whose destination address is the firewall; this forms the outer IP packet.
2. The outer packet is routed to the destination firewall. Each intermediate router needs to examine and process the outer IP header plus any outer IP extension headers but does not need to examine the ciphertext.
3. The destination firewall examines and processes the outer IP header plus any outer IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext inner IP packet. This packet is then transmitted in the internal network.
4. The inner packet is routed through zero or more routers in the internal network to the destination host.

### 13.5 COMBINING SECURITY ASSOCIATIONS

An individual SA can implement either the AH or ESP protocol but not both. Sometimes a particular traffic flow will call for the services provided by both AH and ESP. Further, a particular traffic flow may require IPsec services between hosts and, for that same flow, separate services between security gateways, such as firewalls. In all of these cases, multiple SAs must be employed for the same traffic flow to achieve the desired IPsec services. The term *security association bundle* refers to a sequence of SAs through which traffic must be processed to provide a desired set of IPsec services. The SAs in a bundle may terminate at different endpoints or at the same endpoints.

Security associations may be combined into bundles in two ways:

- **Transport adjacency:** Refers to applying more than one security protocol to the same IP packet, without invoking tunneling. This approach to combining AH and ESP allows for only one level of combination; further nesting yields no added benefit since the processing is performed at one IPsec instance: the (ultimate) destination.
- **Iterated tunneling:** Refers to the application of multiple layers of security protocols effected through IP tunneling. This approach allows for multiple levels of nesting, since each tunnel can originate or terminate at a different IPsec site along the path.

The two approaches can be combined (for example, by having a transport SA between hosts travel part of the way through a tunnel SA between security gateways).

One interesting issue that arises when considering SA bundles is the order in which authentication and encryption may be applied between a given pair of endpoints and the ways of doing so. We examine that issue next. Then we look at combinations of SAs that involve at least one tunnel.

### Authentication Plus Confidentiality

Encryption and authentication can be combined in order to transmit an IP packet that has both confidentiality and authentication between hosts. We look at several approaches.

#### ESP with Authentication Option

This approach is illustrated in Figure 13.9. In this approach, the user first applies ESP to the data to be protected and then appends the authentication data field. There are actually two subcases:

- **Transport mode ESP:** Authentication and encryption apply to the IP payload delivered to the host, but the IP header is not protected.
- **Tunnel mode ESP:** Authentication applies to the entire IP packet delivered to the outer IP destination address (e.g., a firewall), and authentication is performed at that destination. The entire inner IP packet is protected by the privacy mechanism, for delivery to the inner IP destination.

For both cases, authentication applies to the ciphertext rather than the plaintext.

#### Transport Adjacency

Another way to apply authentication after encryption is to use two bundled transport SAs, with the inner being an ESP SA and the outer being an AH SA. In this case ESP is used without its authentication option. Because the inner SA is a transport SA, encryption is applied to the IP payload. The resulting packet consists of an IP header (and possibly IPv6 header extensions) followed by an ESP. AH is then applied in transport mode, so that authentication covers the ESP plus the original IP header (and extensions) except for mutable fields. The advantage of this approach over simply using a single ESP SA with the ESP authentication option is that the authentication covers more fields, including the source and destination IP addresses. The disadvantage is the overhead of two SAs versus one SA.

#### Transport-Tunnel Bundle

The use of authentication prior to encryption might be preferable for several reasons. First, because the authentication data are protected by encryption, it is impossible for anyone to intercept the message and alter the authentication data without detection. Second, it may be desirable to store the authentication information with the message at the destination for later reference. It is more convenient to do this if the authentication information applies to the unencrypted

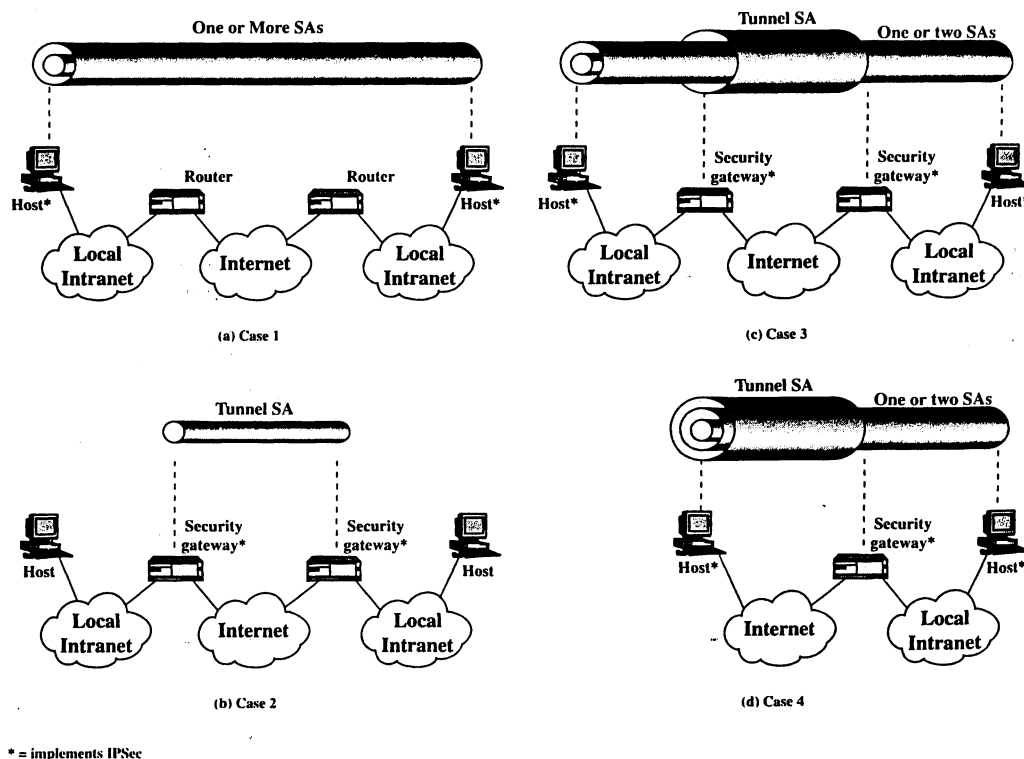
message; otherwise the message would have to be reencrypted to verify the authentication information.

One approach to applying authentication before encryption between two hosts is to use a bundle consisting of an inner AH transport SA and an outer ESP tunnel SA. In this case, authentication is applied to the IP payload plus the IP header (and extensions) except for mutable fields. The resulting IP packet is then processed in tunnel mode by ESP; the result is that the entire, authenticated inner packet is encrypted and a new outer IP header (and extensions) is added.

### Basic Combinations of Security Associations

The IPsec Architecture document lists four examples of combinations of SAs that must be supported by compliant IPsec hosts (e.g., workstation, server) or security gateways (e.g., firewall, router). These are illustrated in Figure 13.10. The lower part of each case in the figure represents the physical connectivity of the elements; the upper part represents logical connectivity via one or more nested SAs. Each SA can be either AH or ESP. For host-to-host SAs, the mode may be either transport or tunnel; otherwise it must be tunnel mode.

In **Case 1**, all security is provided between end systems that implement IPsec. For any two end systems to communicate via an SA, they must share the appropriate secret keys. Possible combinations include the following:



**Figure 13.10.** Basic Combinations of Security Associations.

- a. AH in transport mode
- b. ESP in transport mode
- c. AH followed by ESP in transport mode (an AH SA inside an ESP SA)
- d. Any one of a, b, or c inside an AH or ESP in tunnel mode

We have already discussed how these various combinations can be used to support authentication, encryption, authentication before encryption, and authentication after encryption.

For **Case 2**, security is provided only between gateways (routers, firewalls, etc.) and no hosts implement IPSec. This case illustrates simple virtual private network support. The security architecture document specifies that only a single tunnel SA is needed for this case. The tunnel could support AH, ESP, or ESP with the authentication option. Nested tunnels are not required because the IPSec services apply to the entire inner packet.

**Case 3** builds on case 2 by adding end-to-end security. The same combinations discussed for cases 1 and 2 are allowed here. The gateway-to-gateway tunnel provides either authentication or confidentiality or both for all traffic between end systems. When the gateway-to-gateway tunnel is ESP, it also provides a limited form of traffic confidentiality. Individual hosts can implement any additional IPSec services required for given applications or given users by means of end-to-end SAs.

**Case 4** provides support for a remote host that uses the Internet to reach an organization's firewall and then to gain access to some server or workstation behind the firewall. Only tunnel mode is required between the remote host and the firewall. As in Case 1, one or two SAs may be used between the remote host and the local host.

## 13.6 KEY MANAGEMENT

The key management portion of IPSec involves the determination and distribution of secret keys. A typical requirement is four keys for communication between two applications: transmit and receive pairs for both AH and ESP. The IPSec Architecture document mandates support for two types of key management:

- **Manual:** A system administrator manually configures each system with its own keys and with the keys of other communicating systems. This is practical for small, relatively static environments.
- **Automated:** An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration.

The default automated key management protocol for IPSec is referred to as ISAKMP/Oakley and consists of the following elements:

- **Oakley Key Determination Protocol:** Oakley is a key exchange protocol based on the Diffie-Hellman algorithm but providing added security. Oakley is generic in that it does not dictate specific formats.

- **Internet Security Association and Key Management Protocol (ISAKMP):** ISAKMP provides a framework for Internet key management and provides the specific protocol support, including formats, for negotiation of security attributes.

ISAKMP by itself does not dictate a specific key exchange algorithm; rather, ISAKMP consists of a set of message types that enable the use of a variety of key exchange algorithms. Oakley is the specific key exchange algorithm mandated for use with the initial version of ISAKMP.

We begin with an overview of Oakley and then look at ISAKMP.

### Oakley Key Determination Protocol

Oakley is a refinement of the Diffie-Hellman key exchange algorithm. Recall that Diffie-Hellman involves the following interaction between users A and B. There is prior agreement on two global parameters:  $q$ , a large prime number; and  $\alpha$ , a primitive root of  $q$ . A selects a random integer  $X_A$  as its private key and transmits to B its public key  $Y_A = \alpha^{X_A}$ . Similarly, B selects a random integer  $X_B$  as its private key, and transmits to A its public key  $Y_B = \alpha^{X_B}$ . Each side can now compute the secret session key:

$$K = (Y_B)^{X_A} \bmod q = (Y_A)^{X_B} \bmod q = \alpha^{X_A X_B} \bmod q$$

The Diffie-Hellman algorithm has two attractive features:

- Secret keys are created only when needed. There is no need to store secret keys for a long period of time, exposing them to increased vulnerability.
- The exchange requires no preexisting infrastructure other than an agreement on the global parameters.

However, there are a number of weaknesses to Diffie-Hellman, as pointed out in [HUIT98]:

- It does not provide any information about the identities of the parties.
- It is subject to a man-in-the-middle attack, in which a third party C impersonates B while communicating with A and impersonates A while communicating with C. Both A and B end up negotiating a key with C, which can then listen to and pass on traffic. The man-in-the-middle attack proceeds as follows:
  1. B sends his public key  $Y_B$  in a message addressed to A (see Figure 6.16).
  2. The enemy (E) intercepts this message. E saves B's public key and sends a message to A that has B's User ID but E's public key  $Y_E$ . This message is sent in such a way that it appears as though it was sent from B's host system. A receives E's message and stores E's public key with B's User ID. Similarly, E sends a message to B with E's public key, purporting to come from A.
  3. B computes a secret key  $K_1$  based on B's private key and  $Y_E$ . A computes a secret key  $K_2$  based on A's private key and  $Y_E$ . E computes  $K_1$  using E's secret key  $X_E$  and  $Y_B$  and computes  $K_2$  using  $X_E$  and  $Y_A$ .

4. From now on E is able to relay messages from A to B and from B to A, appropriately changing their encipherment en route in such a way that neither A nor B will know that they share their communication with E.
- It is computationally intensive. As a result, it is vulnerable to a clogging attack, in which an opponent requests a high number of keys. The victim spends considerable computing resources doing useless modular exponentiation rather than real work.

Oakley is designed to retain the advantages of Diffie-Hellman while counteracting its weaknesses.

### Features of Oakley

The Oakley algorithm is characterized by five important features:

1. It employs a mechanism known as cookies to thwart clogging attacks.
2. It enables the two parties to negotiate a *group*; this, in essence, specifies the global parameters of the Diffie-Hellman key exchange.
3. It uses nonces to ensure against replay attacks.
4. It enables the exchange of Diffie-Hellman public key values.
5. It authenticates the Diffie-Hellman exchange to thwart man-in-the-middle attacks.

We have already discussed Diffie-Hellman. Let us look the remainder of these elements in turn. First, consider the problem of clogging attacks. In this attack, an opponent forges the source address of a legitimate user and sends a public Diffie-Hellman key to the victim. The victim then performs a modular exponentiation to compute the secret key. Repeated messages of this type can *clog* the victim's system with useless work. The **cookie exchange** requires that each side send a pseudorandom number, the cookie, in the initial message, which the other side acknowledges. This acknowledgment must be repeated in the first message of the Diffie-Hellman key exchange. If the source address was forged, the opponent gets no answer. Thus, an opponent can only force a user to generate acknowledgments and not to perform the Diffie-Hellman calculation.

ISAKMP mandates that cookie generation satisfy three basic requirements:

1. The cookie must depend on the specific parties. This prevents an attacker from obtaining a cookie using a real IP address and UDP port, and then using it to swamp the victim with requests from randomly chosen IP addresses or ports.
2. It must not be possible for anyone other than the issuing entity to generate cookies that will be accepted by that entity. This implies that the issuing entity will use local secret information in the generation and subsequent verification of a cookie. It must not be possible to deduce this secret information from any particular cookie. The point of this requirement is that the issuing entity need not save copies of its cookies, which are then more vulnerable to discovery, but can verify an incoming cookie acknowledgment when it needs to.
3. The cookie generation and verification methods must be fast to thwart attacks intended to sabotage processor resources.

The recommended method for creating the cookie is to perform a fast hash (e.g., MD5) over the IP Source and Destination addresses, the UDP Source and Destination ports, and a locally generated secret value.

Oakley supports the use of different **groups** for the Diffie-Hellman key exchange. Each group includes the definition of the two global parameters and the identity of the algorithm. The current specification includes the following groups:

- Modular exponentiation with a 768-bit modulus

$$q = 2^{768} - 2^{704} - 1 + 2^{64} \times (\lfloor 2^{638} \times \pi \rfloor + 149686)$$

$$\alpha = 2$$

- Modular exponentiation with a 1024-bit modulus

$$q = 2^{1024} - 2^{960} - 1 + 2^{64} \times (\lfloor 2^{894} \times \pi \rfloor + 129093)$$

$$\alpha = 2$$

- Modular exponentiation with a 1024-bit modulus
  - Parameters to be determined
- Elliptic curve group over  $2^{155}$ 
  - Generator (hexadecimal): X = 7B, Y = 1C8
  - Elliptic curve parameters (hexadecimal): A = 0, Y = 7338F
- Elliptic curve group over  $2^{185}$ 
  - Generator (hexadecimal): X = 18, Y = D
  - Elliptic curve parameters (hexadecimal): A = 0, Y = 1EE9

The first three groups are the classic Diffie-Hellman algorithm using modular exponentiation. The last two groups use the elliptic curve analog to Diffie-Hellman, which was described in Chapter 6.

Oakley employs **nonces** to ensure against replay attacks. Each nonce is a locally generated pseudorandom number. Nonces appear in responses and are encrypted during certain portions of the exchange to secure their use.

Three different **authentication** methods can be used with Oakley:

- **Digital signatures:** The exchange is authenticated by signing a mutually obtainable hash; each party encrypts the hash with its private key. The hash is generated over important parameters, such as user IDs and nonces.
- **Public-key encryption:** The exchange is authenticated by encrypting parameters, such as IDs and nonces, with the sender's private key.
- **Symmetric-key encryption:** A key derived by some out-of-band mechanism can be used to authenticate the exchange by symmetric encryption of exchange parameters.

#### Oakley Exchange Example

The Oakley specification includes a number of examples of exchanges that are allowable under the protocol. To give a flavor of Oakley, we present one example, called aggressive key exchange in the specification, so called because only three messages are exchanged.



Figure 13.11 shows the aggressive key exchange protocol. In the first step, the initiator (I) transmits a cookie, the group to be used, and I's public Diffie-Hellman key for this exchange. I also indicates the offered public-key encryption, hash, and authentication algorithms to be used in this exchange. Also included in this message are the identifiers of I and the responder (R) and I's nonce for this exchange. Finally, I appends a signature using I's private key that signs the two identifiers, the nonce, the group, the Diffie-Hellman public key, and the offered algorithms.

When R receives the message, R verifies the signature using I's public signing key. R acknowledges the message by echoing back I's cookie, identifier, and nonce, as well as the group. R also includes in the message a cookie, R's Diffie-Hellman public key, the selected algorithms (which must be among the offered algorithms), R's identifier, and R's nonce for this exchange. Finally, R appends a signature using R's private key that signs the two identifiers, the two nonces, the group, the two Diffie-Hellman public keys, and the selected algorithms.

When I receives the second message, I verifies the signature using R's public key. The nonce values in the message assure that this is not a replay of an old message. To complete the exchange, I must send a message back to R to verify that I has received R's public key.

## ISAKMP

ISAKMP defines procedures and packet formats to establish, negotiate, modify, and delete security associations. As part of SA establishment, ISAKMP defines payloads for exchanging key generation and authentication data. These payload formats provide a consistent framework independent of the specific key exchange protocol, encryption algorithm, and authentication mechanism.

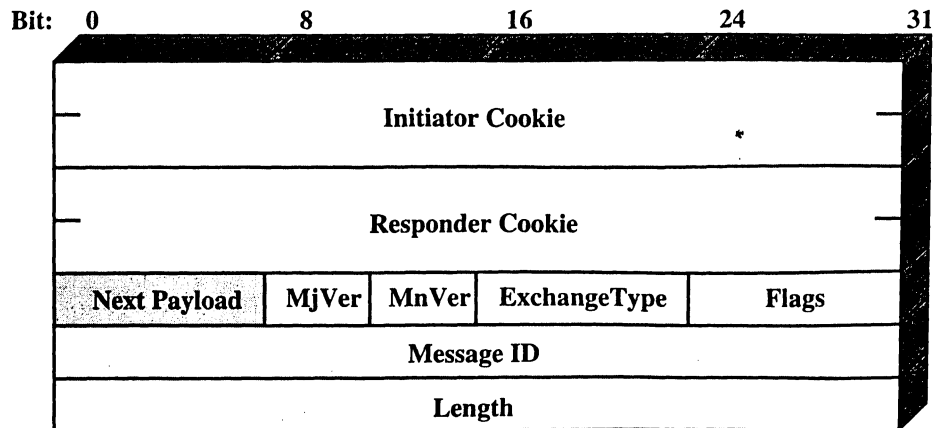
### ISAKMP Header Format

An ISAKMP message consists of an ISAKMP header followed by one or more payloads. All of this is carried in a transport protocol. The specification dictates that implementations must support the use of UDP for the transport protocol.

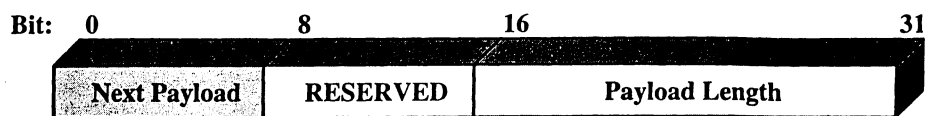
Figure 13.12a shows the header format for an ISAKMP message. It consists of the following fields:

- **Initiator Cookie (64 bits):** Cookie of entity that initiated SA establishment, SA notification, or SA deletion.
- **Responder Cookie (64 bits):** Cookie of responding entity; null in first message from initiator.
- **Next Payload (8 bits):** Indicates the type of the first payload in the message; payloads are discussed in the next subsection.
- **Major Version (4 bits):** Indicates major version of ISAKMP in use.
- **Minor Version (4 bits):** Indicates minor version in use.
- **Exchange Type (8 bits):** Indicates the type of exchange; these are discussed later in this section.
- **Flags (8 bits):** Indicates specific options set for this ISAKMP exchange. Two bits so far defined: The Encryption bit is set if all payloads following the





(a) ISAKMP header



(b) Generic Payload header

Figure 13.12 ISAKMP Formats.

header are encrypted using the encryption algorithm for this SA. The Commit bit is used to ensure that encrypted material is not received prior to completion of SA establishment.

- **Message ID (32 bits):** Unique ID for this message.
- **Length (32 bits):** Length of total message (header plus all payloads) in octets.

#### ISAKMP Payload Types

All ISAKMP payloads begin with the same generic payload header shown in Figure 13.12b. The Next Payload field has a value of 0 if this is the last payload in the message; otherwise its value is the type of the next payload. The Payload Length field indicates the length in octets of this payload, including the generic payload header.

Table 13.3 summarizes the payload types defined for ISAKMP and lists the fields, or parameters, that are part of each payload. The **SA payload** is used to begin the establishment of an SA. In this payload, the Domain of Interpretation parameter identifies the DOI under which negotiation is taking place. The IPsec DOI is one example, but ISAKMP can be used in other contexts. The Situation parameter defines the security policy for this negotiation; in essence, the levels of security required for encryption and confidentiality are specified (e.g., sensitivity level, security compartment).

The **Proposal payload** contains information used during SA negotiation. The payload indicates the protocol for this SA (ESP or AH) for which services and

Table 13.3 ISAKMP Payload Types

Type	Parameters	Description
Security Association (SA)	Domain of Interpretation, Situation	Used to negotiate security attributes and indicate the DOI and Situation under which negotiation is taking place.
Proposal (P)	Proposal #, Protocol-ID, SPI Size, # of Transforms, SPI	Used during SA negotiation; indicates protocol to be used and number of transforms.
Transform (T)	Transform #, Transform-ID, SA Attributes	Used during SA negotiation; indicates transform and related SA attributes.
Key Exchange (KE)	Key Exchange Data	Supports a variety of key exchange techniques.
Identification (ID)	ID Type, ID Data	Used to exchange identification information.
Certificate (CERT)	Cert Encoding, Certificate Data	Used to transport certificates and other certificate-related information.
Certificate Request (CR)	# Cert Types, Certificate Types, # Cert Auths, Certificate Authorities	Used to request certificates; indicates the types of certificates requested and the acceptable certificate authorities.
Hash (HASH)	Hash Data	Contains data generated by a hash function.
Signature (SIG)	Signature Data	Contains data generated by a digital signature function.
Nonce (NONCE)	Nonce Data	Contains a nonce.
Notification (N)	DOI, Protocol-ID, SPI Size, Notify Message Type, SPI, Notification Data	Used to transmit notification data, such as an error condition.
Delete (D)	DOI, Protocol-ID, SPI Size, # of SPIs, SPI (one or more)	Indicates an SA that is no longer valid.

mechanisms are being negotiated. The payload also includes the sending entity's SPI and the number of transforms. Each transform is contained in a transform payload. The use of multiple transform payloads enables the initiator to offer several possibilities, of which the responder must choose one or reject the offer.

The **Transform payload** defines a security transform to be used to secure the communications channel for the designated protocol. The Transform # parameter serves to identify this particular payload so that the responder may use it to indicate acceptance of this transform. The Transform-ID and Attributes fields identify a specific transform (e.g., 3DES for ESP, HMAC-SHA-1-96 for AH) with its associated attributes (e.g., hash length).

The **Key Exchange payload** can be used for a variety of key exchange techniques, including Oakley, Diffie-Hellman, and the RSA-based key exchange used by PGP. The Key Exchange data field contains the data required to generate a session key and is dependent on the key exchange algorithm used.

The **Identification payload** is used to determine the identity of communicating peers and may be used for determining authenticity of information. Typically, the ID Data field will contain an IPv4 or IPv6 address.

The **Certificate payload** transfers a public-key certificate. The Certificate Encoding field indicates the type of certificate or certificate-related information, which may include the following:

- KCS #7 wrapped X.509 certificate
- PGP certificate
- DNS signed key
- X.509 certificate—signature
- X.509 certificate—key exchange
- Kerberos tokens
- Certificate Revocation List (CRL)
- Authority Revocation List (ARL)
- SPKI certificate

At any point in an ISAKMP exchange, the sender may include a **Certificate Request payload** to request the certificate of the other communicating entity. The payload may list more than one certificate type that is acceptable and more than one certificate authority that is acceptable.

The **Hash payload** contains data generated by a hash function over some part of the message and/or ISAKMP state. This payload may be used to verify the integrity of the data in a message or to authenticate the negotiating entities.

The **Signature payload** contains data generated by a digital signature function over some part of the message and/or ISAKMP state. This payload is used to verify the integrity of the data in a message and may be used for nonrepudiation services.

The **Nonce payload** contains random data used to guarantee liveness during an exchange and protect against replay attacks.

The **Notification payload** contains either error or status information associated with this SA or this SA negotiation. The following ISAKMP error messages have been defined:

Invalid Payload Type	Invalid Protocol ID	Invalid Certificate
DOI Not Supported	Invalid SPI	Bad Cert Request Syntax
Situation Not Supported	Invalid Transform ID	Invalid Cert Authority
Invalid Cookie	Attributes Not Supported	Invalid Hash Information
Invalid Major Version	No Proposal Chosen	Authentication Failed
Invalid Minor Version	Bad Proposal Syntax	Invalid Signature
Invalid Exchange Type	Payload Malformed	Address Notification
Invalid Flags	Invalid Key Information	
Invalid Message ID	Invalid Cert Encoding	

The only ISAKMP status message so far defined is Connected. In addition to these ISAKMP notifications, DOI-specific notifications are used. For IPsec, the following additional status messages are defined:

- **Responder-Lifetime:** Communicates the SA lifetime chosen by the responder.
- **Replay-Status:** Used for positive confirmation of the responder's election of whether or not the responder will perform anti-replay detection.
- **Initial-Contact:** Informs the other side that this is the first SA being established with the remote system. The receiver of this notification might then delete any existing SAs it has for the sending system under the assumption that the sending system has rebooted and no longer has access to those SAs.

The **Delete payload** indicates one or more SAs that the sender has deleted from its database and that therefore are no longer valid.

### ISAKMP Exchanges

ISAKMP provides a framework for message exchange, with the payload types serving as the building blocks. The specification identifies five default exchange types that should be supported; these are summarized in Table 13.4. In the table, SA refers to an SA payload with associated Protocol and Transform payloads.

The **Base Exchange** allows key exchange and authentication material to be transmitted together. This minimizes the number of exchanges at the expense of not providing identity protection. The first two messages provide cookies and establish an SA with agreed protocol and transforms; both sides use a nonce to ensure against replay attacks. The last two messages exchange the key material and user IDs, with the AUTH payload used to authenticate keys, identities, and the nonces from the first two messages.

The **Identity Protection Exchange** expands the Base Exchange to protect the users' identities. The first two messages establish the SA. The next two messages perform key exchange, with nonces for replay protection. Once the session key has been computed, the two parties exchange encrypted messages that contain authentication information, such as digital signatures and optionally certificates validating the public keys.

The **Authentication Only Exchange** is used to perform mutual authentication, without a key exchange. The first two messages establish the SA. In addition, the responder uses the second message to convey its ID and uses authentication to protect the message. The initiator sends the third message to transmit its authenticated ID.

Table 13.4 ISAKMP Exchange Types

Exchange	Note
<b>(a) Base Exchange</b>	
(1) I → R: SA; NONCE	Begin ISAKMP-SA negotiation
(2) R → I: SA; NONCE	Basic SA agreed on
(3) I → R: KE; ID <sub>I</sub> ; AUTH	Key generated; Initiator identity verified by responder
(4) R → I: KE; ID <sub>R</sub> ; AUTH	Responder identity verified by initiator; Key generated; SA established
<b>(b) Identity Protection Exchange</b>	
(1) I → R: SA	Begin ISAKMP-SA negotiation
(2) R → I: SA	Basic SA agreed on
(3) I → R: KE; NONCE	Key generated
(4) R → I: KE; NONCE	Key generated
(5)* I → R: ID <sub>I</sub> ; AUTH	Initiator identity verified by responder
(6)* R → I: ID <sub>R</sub> ; AUTH	Responder identity verified by initiator; SA established
<b>(c) Authentication Only Exchange</b>	
(1) I → R: SA; NONCE	Begin ISAKMP-SA negotiation
(2) R → I: SA; NONCE; ID <sub>R</sub> ; AUTH	Basic SA agreed on; Responder identity verified by initiator
(3) I → R: ID <sub>I</sub> ; AUTH	Initiator identity verified by responder; SA established
<b>(d) Aggressive Exchange</b>	
(1) I → R: SA; KE; NONCE; ID <sub>I</sub>	Begin ISAKMP-SA negotiation and key exchange
(2) R → I: SA; KE; NONCE; ID <sub>R</sub> ; AUTH	Initiator identity verified by responder; Key generated; Basic SA agreed upon
(3)* I → R: AUTH	Responder identity verified by initiator; SA established
<b>(e) Informational Exchange</b>	
(1)* I → R: N/D	Error or status notification or deletion

Notation:

I = initiator

R = responder

\* = payload encryption after the ISAKMP header

The **Aggressive Exchange** minimizes the number of exchanges at the expense of not providing identity protection. In the first message, the initiator proposes an SA with associated offered protocol and transform options. The initiator also begins the key exchange and provides its ID. In the second message, the responder indicates its acceptance of the SA with a particular protocol and transform, completes the key exchange, and authenticates the transmitted information. In the third message, the initiator transmits an authentication result that covers the previous information, encrypted using the shared secret session key.

The **Informational Exchange** is used for one-way transmittal of information for SA management.

### 13.7 RECOMMENDED READING

IPv6 and IPv4 are covered in more detail in [STAL97]. Good coverage of internetworking and IPv4 can be found in [COME95] and [STEV94]. [BRAD96] is a thorough treatment of IPv6-related issues; the book provides a relatively nontechnical discussion of the requirements for IPv6 and the history of the IPv6 development. [HUIT98] is a straightforward technical description of the various RFCs that together make up the IPv6 specification; the book provides a discussion of the purpose of various features and of the operation of the protocol. [CHEN98] provides a good discussion of an IPsec design.

**BRAD96** Bradner, S., and Mankin, A. *IPng: Internet Protocol Next Generation*. Reading, MA: Addison-Wesley, 1996.

**CHEN98** Cheng, P., et al. "A Security Architecture for the Internet Protocol." *IBM Systems Journal*, Number 1, 1998.

**COME95a** Comer, D. *Internetworking with TCP/IP, Volume I: Principles, Protocols and Architecture*. Upper Saddle River, NJ: Prentice Hall, 1995.

**HUIT98** Huitema, C. *IPv6: The New Internet Protocol*. Upper Saddle River, NJ: Prentice Hall, 1998.

**STAL97** Stallings, W. *Data and Computer Communications, Fifth Edition*. Upper Saddle River, NJ: Prentice Hall, 1997.

**STEV94** Stevens, W. *TCP/IP Illustrated, Volume 1: The Protocols*. Reading, MA: Addison-Wesley, 1994.

Recommended Web sites:

- **IP Security Protocol (ipsec) Charter:** Latest RFCs and internet drafts for IPsec
- **IP Security Working Group News:** Working group documents, mail archives, related technical papers, and other useful material
- **IP Security (IPSEC) Resources:** List of companies implementing IPsec, implementation survey, and other useful material

### 13.8 PROBLEMS

- 13.1** In discussing AH processing, it was mentioned that not all of the fields in an IP header are included in MAC calculation.
  - a. For each of the fields in the IPv4 header, indicate whether the field is immutable, mutable but predictable, or mutable (zeroed prior to ICV calculation).
  - b. Do the same for the IPv6 header.
  - c. Do the same for the IPv6 extension headers.
 In each case, justify your decision for each field.
- 13.2** When tunnel mode is used, a new outer IP header is constructed. For both IPv4 and IPv6, indicate the relationship of each outer IP header field and each extension header in the outer packet to the corresponding field or extension header of the inner IP packet. That is, indicate which outer values are derived from inner values and which are constructed independently of the inner values.
- 13.3** End-to-end authentication and encryption are desired between two hosts. Draw figures similar to Figures 13.6 and 13.9 that show
  - a. Transport adjacency, with encryption applied before authentication
  - b. A transport SA bundled inside a tunnel SA, with encryption applied before authentication
  - c. A transport SA bundled inside a tunnel SA, with authentication applied before encryption



- 13.4 The IPSec architecture document states that when two transport mode SA's are bundled to allow both AH and ESP protocols on the same end-to-end flow, only one ordering of security protocols seems appropriate: performing the ESP protocol before performing the AH protocol. Why is this approach recommended rather than authentication before encryption?
- 13.5 a. Which of the ISAKMP Exchange Types (Table 13.4) corresponds to the aggressive Oakley key exchange (Figure 13.11)?
- b. For the Oakley aggressive key exchange, indicate which parameters in each message go in which ISAKMP payload types.

## APPENDIX 13A: INTERNETWORKING AND INTERNET PROTOCOLS

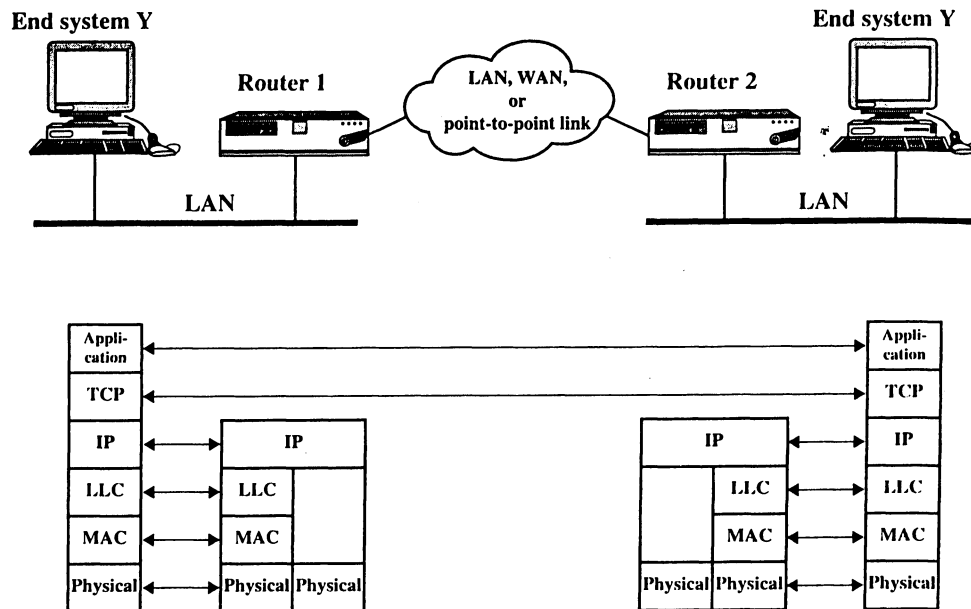
This appendix provides an overview of internet protocols. We begin with a summary of the role of an internet protocol in providing internetworking. Then the two main internet protocols, IPv4 and IPv6, are introduced.

### The Role of an Internet Protocol

An internet protocol (IP) provides the functionality for interconnecting end systems across multiple networks. For this purpose, IP is implemented in each end system and in routers, which are devices that provide connection between networks. Higher-level data at a source end system are encapsulated in an IP protocol data unit (PDU) for transmission. This PDU is then passed through one or more networks and connecting routers to reach the destination end system.

The router must be able to cope with a variety of differences among networks, including the following:

- **Addressing schemes:** The networks may use different schemes for assigning addresses to devices. For example, an IEEE 802 LAN uses either 16-bit or 48-bit binary addresses for each attached device; an X.25 public packet-switching network uses 12-digit decimal addresses (encoded as 4 bits per digit for a 48-bit address). Some form of global network addressing must be provided, as well as a directory service.
- **Maximum packet sizes:** Packets from one network may have to be broken into smaller pieces to be transmitted on another network, a process known as **fragmentation**. For example, Ethernet imposes a maximum packet size of 1500 bytes; a maximum packet size of 1000 bytes is common on X.25 networks. A packet that is transmitted on an Ethernet system and picked up by a router for retransmission on an X.25 network may have to fragment the incoming packet into two smaller ones.
- **Interfaces:** The hardware and software interfaces to various networks differ. The concept of a router must be independent of these differences.
- **Reliability:** Various network services may provide anything from a reliable end-to-end virtual circuit to an unreliable service. The operation of the routers should not depend on an assumption of network reliability.



**Figure 13.13** Configuration for TPC/IP Example.

The operation of the router, as Figure 13.13 indicates, depends on an internet protocol. In this example, the Internet Protocol (IP) of the TCP/IP protocol suite performs that function. IP must be implemented in all end systems on all networks as well as on the routers. In addition, each end system must have compatible protocols above IP to communicate successfully. The intermediate routers need only have up through IP.

Consider the transfer of a block of data from end system *X* to end system *Y* in Figure 13.13. The IP layer at *X* receives blocks of data to be sent to *Y* from TCP in *X*. The IP layer attaches a header that specifies the global internet address of *Y*. That address is in two parts: network identifier and end system identifier. Let us refer to this block as the IP packet. Next, IP recognizes that the destination (*Y*) is on another subnetwork. So the first step is to send the packet to a router, in this case router 1. To accomplish this, IP hands its data unit down to LLC with the appropriate addressing information. LLC creates an LLC PDU, which is handed down to the MAC layer. The MAC layer constructs a MAC packet whose header contains the address of router 1.

Next, the packet travels through LAN to router 1. The router removes the packet and LLC headers and trailers and analyzes the IP header to determine the ultimate destination of the data, in this case *Y*. The router must now make a routing decision. There are two possibilities:

1. The destination end system *Y* is connected directly to one of the subnetworks to which the router is attached.
2. To reach the destination, one or more additional routers must be traversed.

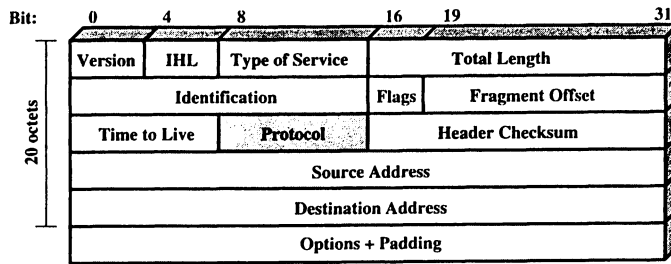
In this example, the packet must be routed through router 2 before reaching the destination. So router 1 passes the IP packet to router 2 via the intermediate network. For this purpose, the protocols of that network are used. For example, if the intermediate network is an X.25 network, the IP data unit is wrapped in an X.25 packet with appropriate addressing information to reach router 2. When this packet arrives at router 2, the packet header is stripped off. The router determines that this IP packet is destined for Y, which is connected directly to a subnetwork to which the router is attached. The router therefore creates a packet with a destination address of Y and sends it out onto the LAN. The data finally arrive at Y, where the packet, LLC, and internet headers and trailers can be stripped off.

This service offered by IP is an unreliable one. That is, IP does not guarantee that all data will be delivered or that the data that are delivered will arrive in the proper order. It is the responsibility of the next higher layer, in this case TCP, to recover from any errors that occur. This approach provides for a great deal of flexibility. Because delivery is not guaranteed, there is no particular reliability requirement on any of the subnetworks. Thus, the protocol will work with any combination of subnetwork types. Because the sequence of delivery is not guaranteed, successive packets can follow different paths through the internet. This allows the protocol to react to congestion and failure in the internet by changing routes.

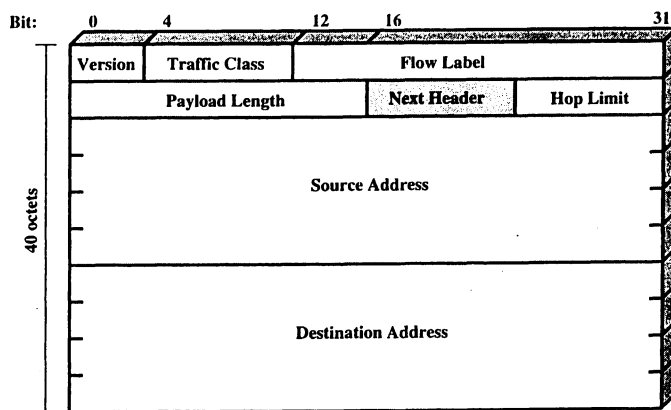
## IPv4

For decades, the keystone of the TCP/IP protocol architecture has been the Internet Protocol (IP) version 4. Figure 13.14a shows the IP header format, which is a minimum of 20 octets, or 160 bits. The fields are as follows:

- **Version (4 bits):** Indicates version number, to allow evolution of the protocol; the value is 4.
- **Internet Header Length (IHL) (4 bits):** Length of header in 32-bit words. The minimum value is five, for a minimum header length of 20 octets.
- **Type of Service (8 bits):** Provides guidance to end system IP modules and to routers along the packet's path, in terms of the packet's relative priority.
- **Total Length (16 bits):** Total IP packet length, in octets.
- **Identification (16 bits):** A sequence number that, together with the source address, destination address, and user protocol, is intended to identify a packet uniquely. Thus, the identifier should be unique for the packet's source address, destination address, and user protocol for the time during which the packet will remain in the internet.
- **Flags (3 bits):** Only two of the bits are currently defined. When a packet is fragmented, the More bit indicates whether this is the last fragment in the original packet. The Don't Fragment bit prohibits fragmentation when set. This bit may be useful if it is known that the destination does not have the capability to reassemble fragments. However, if this bit is set, the packet will be discarded if it exceeds the maximum size of an en route subnetwork. Therefore, if the bit is set, it may be advisable to use source routing to avoid subnetworks with small maximum packet size.



(a) IPv4 header



(b) IPv6 header

Figure 13.14 IP Headers.

- **Fragment Offset (13 bits):** Indicates where in the original packet this fragment belongs, measured in 64-bit units. This implies that fragments other than the last fragment must contain a data field that is a multiple of 64 bits in length.
- **Time to Live (8 bits):** Specifies how long, in seconds, a packet is allowed to remain in the internet. Every router that processes a packet must decrease the TTL by at least one, so the TTL is somewhat similar to a hop count.
- **Protocol (8 bits):** Indicates the next higher level protocol, which is to receive the data field at the destination; thus, this field identifies the type of the next header in the packet after the IP header.
- **Header Checksum (16 bits):** An error-detecting code applied to the header only. Because some header fields may change during transit (e.g., time to live, segmentation-related fields), this is reverified and recomputed at each router. The checksum field is the 16-bit one's complement addition of all 16-bit words in the header. For purposes of computation, the checksum field is itself initialized to a value of zero.
- **Source Address (32 bits):** Coded to allow a variable allocation of bits to specify the network and the end system attached to the specified network (7 and 24 bits, 14 and 16 bits, or 21 and 8 bits).

- **Destination Address (32 bits):** Same characteristics as source address.
- **Options (variable):** Encodes the options requested by the sending user; these may include security label, source routing, record routing, and timestamping.
- **Padding (variable):** Used to ensure that the packet header is a multiple of 32 bits in length.

## IPv6

In 1995, the Internet Engineering Task Force (IETF), which develops protocol standards for the Internet, issued a specification for a next-generation IP, known then as IPng. This specification was turned into a standard in 1996 known as IPv6. IPv6 provides a number of functional enhancements over the existing IP (known as IPv4), designed to accommodate the higher speeds of today's networks and the mix of data streams, including graphic and video, that are becoming more prevalent. But the driving force behind the development of the new protocol was the need for more addresses. IPv4 uses a 32-bit address to specify a source or destination. With the explosive growth of the Internet and of private networks attached to the Internet, this address length became insufficient to accommodate all systems needing addresses. As Figure 13.14b shows, IPv6 includes 128-bit source and destination address fields. Ultimately, all installations using TCP/IP are expected to migrate from the current IP to IPv6, but this process will take many years, if not decades.

### IPv6 Header

The IPv6 header has a fixed length of 40 octets, consisting of the following fields (Figure 13.14b):

- **Version (4 bits):** Internet Protocol version number; the value is 6.
- **Traffic Class (8 bits):** Available for use by originating nodes and/or forwarding routers to identify and distinguish between different classes or priorities of IPv6 packets. The use of this field is still under study.
- **Flow Label (20 bits):** May be used by a host to label those packets for which it is requesting special handling by routers within a network. Flow labeling may assist resource reservation and real-time traffic processing.
- **Payload Length (16 bits):** Length of the remainder of the IPv6 packet following the header, in octets. In other words, this is the total length of all of the extension headers plus the transport-level PDU.
- **Next Header (8 bits):** Identifies the type of header immediately following the IPv6 header; this will either be an IPv6 extension header or a higher-layer header, such as TCP or UDP.
- **Hop Limit (8 bits):** The remaining number of allowable hops for this packet. The hop limit is set to some desired maximum value by the source and decremented by 1 by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero.
- **Source Address (128 bits):** The address of the originator of the packet.
- **Destination Address (128 bits):** The address of the intended recipient of the packet. This may not, in fact, be the intended ultimate destination if a Routing extension header is present, as explained later.

Although the IPv6 header is longer than the mandatory portion of the IPv4 header (40 octets versus 20 octets), it contains fewer fields (8 versus 12). Thus, routers have less processing to do per header, which should speed up routing.

#### IPv6 Extension Headers

An IPv6 packet includes the IPv6 header, just discussed, and zero or more extension headers. Outside of IPsec, the following extension headers have been defined:

- **Hop-by-Hop Options header:** Defines special options that require hop-by-hop processing.
- **Routing header:** Provides extended routing, similar to IPv4 source routing.
- **Fragment header:** Contains fragmentation and reassembly information.
- **Authentication header:** Provides packet integrity and authentication.
- **Encapsulating Security Payload header:** Provides privacy.
- **Destination Options header:** Contains optional information to be examined by the destination node.

The IPv6 standard recommends that, when multiple extension headers are used, the IPv6 headers appear in the following order.

1. IPv6 header: Mandatory, must always appear first
2. Hop-by-Hop Options header
3. Destination Options header: For options to be processed by the first destination that appears in the IPv6 Destination Address field plus subsequent destinations listed in the Routing header
4. Routing header
5. Fragment header
6. Authentication header
7. Encapsulating Security Payload header
8. Destination Options header: For options to be processed only by the final destination of the packet

Figure 13.15 shows an example of an IPv6 packet that includes an instance of each nonsecurity header. Note that the IPv6 header and each extension header include a Next Header field. This field identifies the type of the immediately following header. If the next header is an extension header, then this field contains the type identifier of that header. Otherwise, this field contains the protocol identifier of the upper-layer protocol using IPv6 (typically a transport-level protocol), using the same values as the IPv4 Protocol field. In Figure 13.15, the upper-layer protocol is TCP, so the upper-layer data carried by the IPv6 packet consist of a TCP header followed by a block of application data.

The **Hop-by-Hop Options header** carries optional information that, if present, must be examined by every router along the path. The header consists of the following fields:

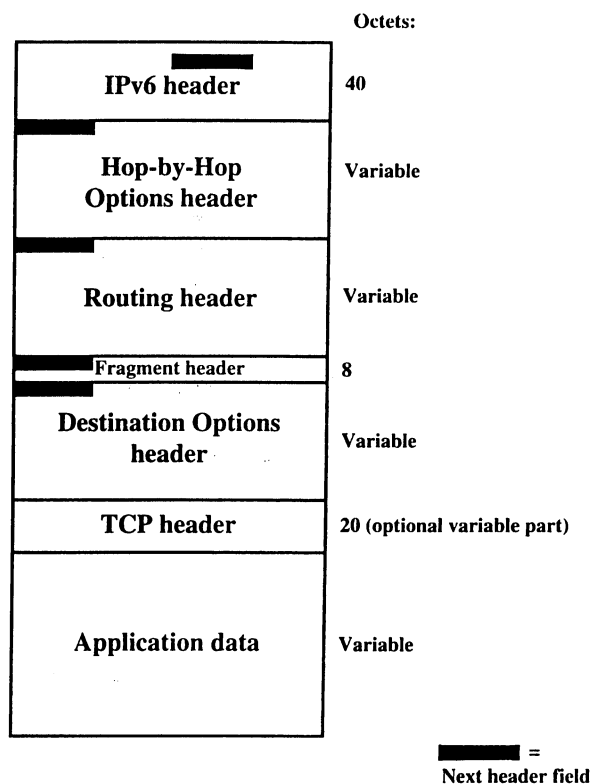
- **Next Header (8 bits):** Identifies the type of header immediately following this header.

- **Header Extension Length (8 bits):** Length of this header in 64-bit units, not including the first 64 bits.
- **Options:** Contains one or more options. Each option consists of three sub-fields: a tag, indicating the option type; a length, and a value.

Only one option has so far been defined: the Jumbo Payload option, used to send IPv6 packets with payloads longer than  $2^{16} - 1 = 65,535$  octets. The Option Data field of this option is 32 bits long and gives the length of the packet in octets, excluding the IPv6 header. For such packets, the Payload Length field in the IPv6 header must be set to zero, and there must be no Fragment header. With this option, IPv6 supports packet sizes up to more than 4 billion octets. This facilitates the transmission of large video packets and enables IPv6 to make the best use of available capacity over any transmission medium.

The **Routing header** contains a list of one or more intermediate nodes to be visited on the way to a packet's destination. All routing headers start with a 32-bit block consisting of four 8-bit fields, followed by routing data specific to a given routing type. The four 8-bit fields are Next Header, Header Extension Length, and

- **Routing Type:** Identifies a particular Routing header variant. If a router does not recognize the Routing Type value, it must discard the packet.



**Figure 13.15** IPv6 Packet with Extension Headers (containing a TCP segment).

- **Segments Left:** Number of explicitly listed intermediate nodes still to be visited before reaching the final destination.

In addition to this general header definition, the IPv6 specification defines the Type 0 Routing header. When using the Type 0 Routing header, the source node does not place the ultimate destination address in the IPv6 header. Instead, that address is the last address listed in the Routing header, and the IPv6 header contains the destination address of the first desired router on the path. The Routing header will not be examined until the packet reaches the node identified in the IPv6 header. At that point, the IPv6 and Routing header contents are updated and the packet is forwarded. The update consists of placing the next address to be visited in the IPv6 header and decrementing the Segments Left field in the Routing header.

IPv6 requires an IPv6 node to reverse routes in a packet it receives containing a Routing header, to return a packet to the sender.

The **Fragment header** is used by a source when fragmentation is required. In IPv6, fragmentation may only be performed by source nodes, not by routers along a packet's delivery path. To take full advantage of the internetworking environment, a node must perform a path discovery algorithm that enables it to learn the smallest maximum transmission unit (MTU) supported by any subnetwork on the path. In other words, the path discovery algorithm enables a node to learn the MTU of the "bottleneck" subnetwork on the path. With this knowledge, the source node will fragment, as required, for each given destination address. Otherwise the source must limit all packets to 1280 octets, which is the minimum MTU that must be supported by each subnetwork.

In addition to the Next Header field, the fragment header includes the following fields:

- **Fragment Offset (13 bits):** Indicates where in the original packet the payload of this fragment belongs. It is measured in 64-bit units. This implies that fragments (other than the last fragment) must contain a data field that is a multiple of 64 bits long.
- **Res (2 bits):** Reserved for future use.
- **M Flag (1 bit):** 1 = more fragments; 0 = last fragment.
- **Identification (32 bits):** Intended to identify uniquely the original packet. The identifier must be unique for the packet's source address and destination address for the time during which the packet will remain in the internet. All fragments with the same identifier, source address, and destination address are reassembled to form the original packet.

The **Destination Options header** carries optional information that, if present, is examined only by the packet's destination node. The format of this header is the same as that of the Hop-by-Hop Options header.



# CHAPTER 14

---

## WEB SECURITY

*Use your mentality  
Wake up to reality*

—From the song “I’ve Got You under My Skin,”  
by Cole Porter

Virtually all businesses, most government agencies, and many individuals now have Web sites. The number of individuals and companies with Internet access is expanding rapidly, and all of these have graphical Web browsers. As a result, businesses are enthusiastic about setting up facilities on the Web for electronic commerce. But the reality is that the Internet and the Web are extremely vulnerable to compromises of various sorts. As businesses wake up to this reality, the demand for secure Web services grows.

The topic of Web security is a broad one and can easily fill a book (several are recommended at the end of this chapter). In this chapter, we begin with a discussion of the general requirements for Web security and then focus on two standardized schemes that are becoming increasingly important as part of Web commerce: SSL/TLS and SET.

### 14.1 WEB SECURITY CONSIDERATIONS

The World Wide Web is fundamentally a client/server application running over the Internet and TCP/IP intranets. As such, the security tools and approaches discussed so far in this book are relevant to the issue of Web security. But, as pointed out in [GARF97], the Web presents new challenges not generally appreciated in the context of computer and network security:

- The Internet is two way. Unlike traditional publishing environments, even electronic publishing systems involving teletext, voice response, or fax-back, the Web is vulnerable to attacks on the Web servers over the Internet.
- The Web is increasingly serving as a highly visible outlet for corporate and product information and as the platform for business transactions. Reputations can be damaged and money can be lost if the Web servers are subverted.
- Although Web browsers are very easy to use, Web servers are relatively easy to configure and manage, and Web content is increasingly easy to develop, the underlying software is extraordinarily complex. This complex software may hide many potential security flaws. The short history of the Web is filled with examples of new and upgraded systems, properly installed, that are vulnerable to a variety of security attacks.
- A Web server can be exploited as a launching pad into the corporation's or agency's entire computer complex. Once the Web server is subverted, an attacker may be able to gain access to data and systems not part of the Web itself but connected to the server at the local site.
- Casual and untrained (in security matters) users are common clients for Web-based services. Such users are not necessarily aware of the security risks that exist and do not have the tools or knowledge to take effective countermeasures.

### Web Security Threats

Table 14.1 provides a summary of the types of security threats faced in using the Web. One way to group these threats is in terms of passive and active attacks. Passive attacks include eavesdropping on network traffic between browser and server and gaining access to information on a Web site that is supposed to be restricted. Active attacks include impersonating another user, altering messages in transit between client and server, and altering information on a Web site.

Another way to classify Web security threats is in terms of the location of the threat: Web server, Web browser, and network traffic between browser and server. Issues of server and browser security fall into the category of computer system security; Part Four of this book addresses the issue of system security in general but is also applicable to Web system security. Issues of traffic security fall into the category of network security and are addressed in this chapter.

### Web Traffic Security Approaches

A number of approaches to providing Web security are possible. The various approaches that have been considered are similar in the services they provide and, to some extent, in the mechanisms that they use, but they differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack.

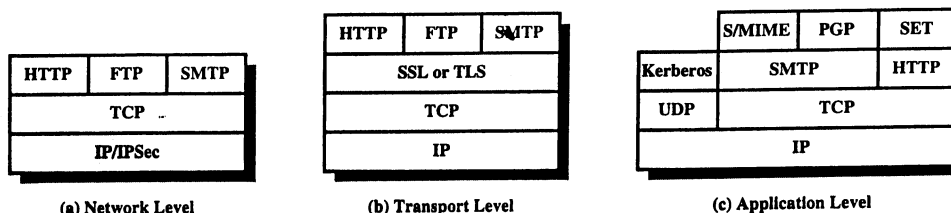
Figure 14.1 illustrates this difference. One way to provide Web security is to use IP Security (Figure 14.1a). The advantage of using IPSec is that it is transparent to end users and applications and provides a general-purpose solution. Further, IPSec includes a filtering capability so that only selected traffic need incur the overhead of IPSec processing.

**Table 14.1** A Comparison of Threats on the Web [RUBI97]

	Threats	Consequences	Countermeasures
Integrity	<ul style="list-style-type: none"> <li>• Modification of user data</li> <li>• Trojan horse browser</li> <li>• Modification of memory</li> <li>• Modification of message traffic in transit</li> </ul>	<ul style="list-style-type: none"> <li>• Loss of information</li> <li>• Compromise of machine</li> <li>• Vulnerability to all other threats</li> </ul>	Cryptographic checksums
Confidentiality	<ul style="list-style-type: none"> <li>• Eavesdropping on the Net</li> <li>• Theft of info from server</li> <li>• Theft of data from client</li> <li>• Info about network configuration</li> <li>• Info about which client talks to server</li> </ul>	<ul style="list-style-type: none"> <li>• Loss of information</li> <li>• Loss of privacy</li> </ul>	Encryption, Web proxies
Denial of Service	<ul style="list-style-type: none"> <li>• Killing of user threads</li> <li>• Flooding machine with bogus threats</li> <li>• Filling up disk or memory</li> <li>• Isolating machine by DNS attacks</li> </ul>	<ul style="list-style-type: none"> <li>• Disruptive</li> <li>• Annoying</li> <li>• Prevent user from getting work done</li> </ul>	Difficult to prevent
Authentication	<ul style="list-style-type: none"> <li>• Impersonation of legitimate users</li> <li>• Data forgery</li> </ul>	<ul style="list-style-type: none"> <li>• Misrepresentation of user</li> <li>• Belief that false information is valid</li> </ul>	Cryptographic techniques

Another relatively general-purpose solution is to implement security just above TCP (Figure 14.1b). The foremost example of this approach is the Secure Sockets Layer (SSL) and the follow-on Internet standard of SSL known as Transport Layer Security (TLS). At this level, there are two implementation choices. For full generality, SSL (or TLS) could be provided as part of the underlying protocol suite and therefore be transparent to applications. Alternatively, SSL can be embedded in specific packages. For example, Netscape and Microsoft Explorer browsers come equipped with SSL, and most Web servers have implemented the protocol.

Application-specific security services are embedded within the particular application. Figure 14.1c shows examples of this architecture. The advantage of this

**Figure 14.1** Relative Location of Security Facilities in the TCP/IP Protocol Stack.

approach is that the service can be tailored to the specific needs of a given application. In the context of Web security, an important example of this approach is Secure Electronic Transaction (SET).<sup>1</sup>

The remainder of this chapter is devoted to a discussion of SSL/TLS and SET.

## 14.2 SECURE SOCKET LAYER AND TRANSPORT LAYER SECURITY

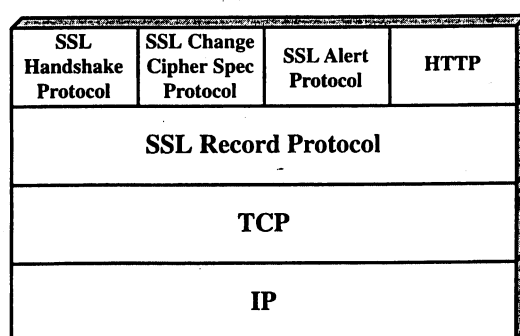
SSL was originated by Netscape. Version 3 of the protocol was designed with public review and input from industry and was published as an Internet draft document. Subsequently, when a consensus was reached to submit the protocol for Internet standardization, the TLS working group was formed within IETF to develop a common standard. The current work on TLS is aimed at producing an initial version as an Internet Standard. This first version of TLS can be viewed as essentially an SSLv3.1 and is very close to and backward compatible with SSLv3.

The bulk of this section is devoted to a discussion of SSLv3. At the end of the section, the key differences between SSLv3 and TLS are described.

### SSL Architecture

SSL is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol but rather two layers of protocols, as illustrated in Figure 14.2.

The SSL Record Protocol provides basic security services to various higher-layer protocols. In particular, the hypertext transfer protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL. Three higher-layer protocols are defined as part of SSL: the Handshake Protocol, the Change Cipher Spec Protocol, and the Alert Protocol. These SSL-



**Figure 14.2** SSL Protocol Stack.

<sup>1</sup>Figure 14 shows SET on top of HTTP; this is a common implementation. In some implementations, SET makes use of TCP directly.

specific protocols are used in the management of SSL exchanges and are examined later in this section.

Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows:

- **Connection:** A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
- **Session:** An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

Between any pair of parties (applications such as HTTP on client and server), there may be multiple secure connections. In theory, there may also be multiple simultaneous sessions between parties, but this feature is not used in practice.

There are actually a number of states associated with each session. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created. Upon successful conclusion of the Handshake Protocol, the pending states become the current states.

A session state is defined by the following parameters (definitions from the SSL specification):

- **Session identifier:** An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- **Peer certificate:** An X509.v3 certificate of the peer. This element of the state may be null.
- **Compression method:** The algorithm used to compress data prior to encryption.
- **Cipher spec:** Specifies the bulk data encryption algorithm (such as null, DES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash\_size.
- **Master secret:** Forty-eight-byte secret shared between the client and server.
- **Is resumable:** A flag indicating whether the session can be used to initiate new connections.

A connection state is defined by the following parameters:

- **Server and client random:** Byte sequences that are chosen by the server and client for each connection.
- **Server write MAC secret:** The secret key used in MAC operations on data sent by the server.
- **Client write MAC secret:** The secret key used in MAC operations on data sent by the client.

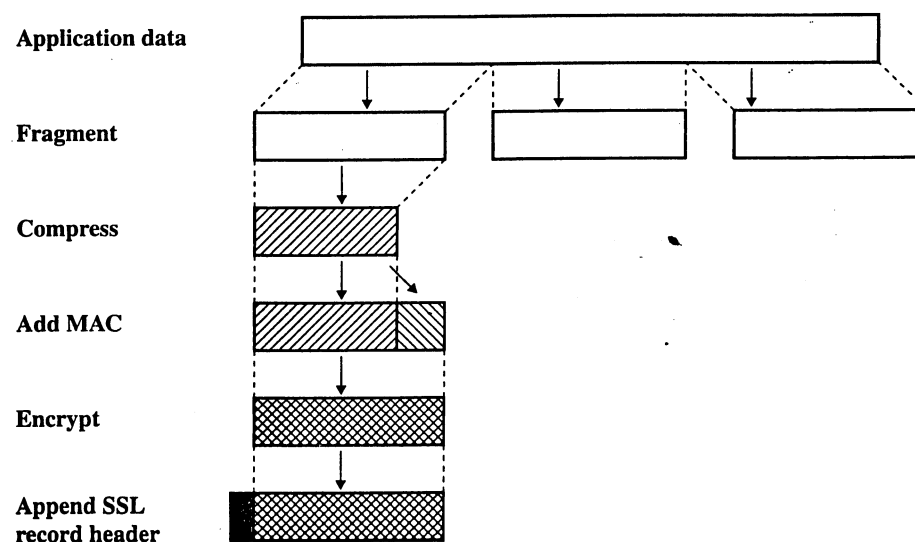
- **Server write key:** The conventional encryption key for data encrypted by the server and decrypted by the client.
- **Client write key:** The conventional encryption key for data encrypted by the client and decrypted by the server.
- **Initialization vectors:** When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter the final ciphertext block from each record is preserved for use as the IV with the following record.
- **Sequence numbers:** Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed  $2^{64} - 1$ .

### SSL Record Protocol

The SSL Record Protocol provides two services for SSL connections:

- **Confidentiality:** The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads.
- **Message Integrity:** The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).

Figure 14.3 indicates the overall operation of the SSL Record Protocol. The Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment. Received data are



**Figure 14.3** SSL Record Protocol Operation.

decrypted, verified, decompressed, and reassembled and then delivered to higher-level users.

The first step is **fragmentation**. Each upper-layer message is fragmented into blocks of  $2^{14}$  bytes (16384 bytes) or less. Next, **compression** is optionally applied. Compression must be lossless and may not increase the content length by more than 1024 bytes.<sup>2</sup> In SSLv3 (as well as the current version of TLS), no compression algorithm is specified, so the default compression algorithm is null.

The next step in processing is to compute a **message authentication code** over the compressed data. For this purpose, a shared secret key is used. The calculation is defined as follows:

```
hash(MAC_write_secret || pad_2 ||
      hash(MAC_write_secret || pad_1 || seq_num || SSLCompressed.type ||
           SSLCompressed.length || SSLCompressed.fragment))
```

where

	=	concatenation
MAC_write_secret	=	shared secret key
hash	=	cryptographic hash algorithm; either MD5 or SHA-1
pad_1	=	the byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times (320 bits) for SHA-1
pad_2	=	the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1
seq_num	=	the sequence number for this message
SSLCompressed.type	=	the higher-level protocol used to process this fragment
SSLCompressed.length	=	the length of the compressed fragment
SSLCompressed.fragment	=	the compressed fragment (if compression is not used, the plaintext fragment)

Note that this is very similar to the HMAC algorithm defined in Chapter 9. The difference is that the two pads are concatenated in SSLv3 and are XORed in HMAC. The SSLv3 MAC algorithm is based on the original internet draft for HMAC, which used concatenation. The final version of HMAC, defined in RFC 2104, uses the XOR.

Next, the compressed message plus the MAC are **encrypted** using symmetric encryption. Encryption may not increase the content length by more than 1024 bytes, so that the total length may not exceed  $2^{14} + 2048$ . The following encryption algorithms are permitted:

---

<sup>2</sup>Of course, one hopes that compression shrinks rather than expands the data. However, for very short blocks, it is possible, because of formatting conventions, that the compression algorithm will actually provide output that is longer than the input.

Block Cipher		Stream Cipher	
Algorithm	Key Size	Algorithm	Key Size
IDEA	128	RC4-40	40
RC2-40	40	RC4-128	128
DES-40	40		
DES	56		
3DES	168		
Fortezza	80		

Fortezza can be used in a smart card encryption scheme.

For stream encryption, the compressed message plus the MAC are encrypted. Note that the MAC is computed before encryption takes place and that the MAC is then encrypted along with the plaintext or compressed plaintext.

For block encryption, padding may be added after the MAC prior to encryption. The padding is in the form of a number of padding bytes followed by a 1-byte indication of the length of the padding. The total amount of padding is the smallest amount such that the total size of the data to be encrypted (plaintext plus MAC plus padding) is a multiple of the cipher's block length. An example is a plaintext (or compressed text if compression is used) of 58 bytes, with a MAC of 20 bytes (using SHA-1), that is encrypted using a block length of 8 bytes (e.g., DES). With the padding.length byte, this yields a total of 79 bytes. To make the total an integer multiple of 8, one byte of padding is added.

The final step of SSL Record Protocol processing is to prepend a header, consisting of the following fields:

- **Content Type (8 bits):** The higher-layer protocol used to process the enclosed fragment.
- **Major Version (8 bits):** Indicates major version of SSL in use. For SSLv3, the value is 3.
- **Minor Version (8 bits):** Indicates minor version in use. For SSLv3, the value is 0.
- **Compressed length (16 bits):** The length in bytes of the plaintext fragment (or compressed fragment if compression is used). The maximum value is  $2^{14} + 2048$ .

The content types that have been defined are change\_cipher\_spec, alert, handshake, and application\_data. The first three are the SSL-specific protocols, discussed next. Note that no distinction is made among the various applications (e.g., HTTP) that might use SSL; the content of the data created by such applications is opaque to SSL.

Figure 14.4 illustrates the SSL record format.

### Change Cipher Spec Protocol

The Change Cipher Spec Protocol is one of the three SSL-specific protocols that use the SSL Record Protocol, and it is the simplest. This protocol consists of a single message (Figure 14.5a), which consists of a single byte with the value 1. The sole purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.



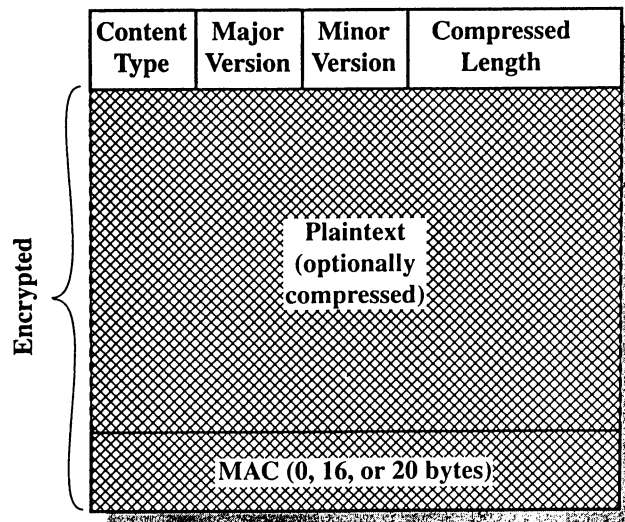


Figure 14.4 SSL Record Format.

### Alert Protocol

The Alert Protocol is used to convey SSL-related alerts to the peer entity. As with other applications that use SSL, alert messages are compressed and encrypted, as specified by the current state.

Each message in this protocol consists of two bytes (Figure 14.5b). The first byte takes the value warning(1) or fatal(2) to convey the severity of the message. If the level is fatal, SSL immediately terminates the connection. Other connections on the same session may continue, but no new connections on this session may be established. The second byte contains a code that indicates the specific alert. First, we list those alerts that are always fatal (as defined in the SSL specification):

- **unexpected\_message:** An inappropriate message was received.
- **bad\_record\_mac:** An incorrect MAC was received.

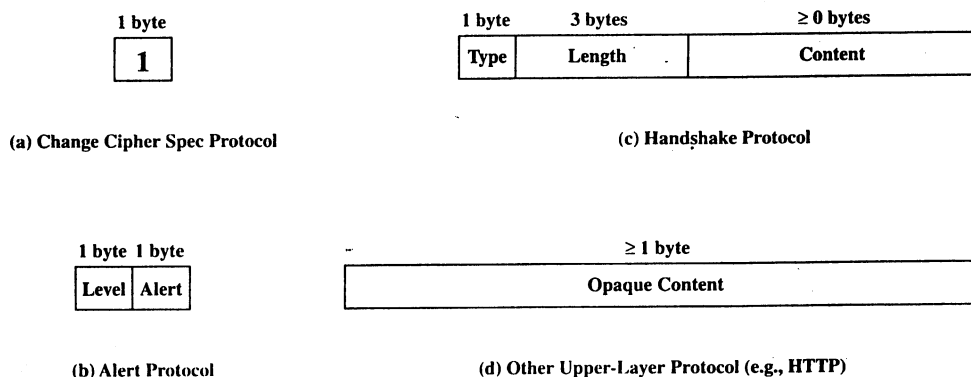


Figure 14.5 SSL Record Protocol Payload.

- **decompression\_failure:** The decompression function received improper input (e.g., unable to decompress or decompress to greater than maximum allowable length).
- **handshake\_failure:** Sender was unable to negotiate an acceptable set of security parameters given the options available.
- **illegal\_parameter:** A field in a handshake message was out of range or inconsistent with other fields.

The remainder of the alerts are the following:

- **close\_notify:** Notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send a close\_notify alert before closing the write side of a connection.
- **no\_certificate:** May be sent in response to a certificate request if no appropriate certificate is available.
- **bad\_certificate:** A received certificate was corrupt (e.g., contained a signature that did not verify).
- **unsupported\_certificate:** The type of the received certificate is not supported.
- **certificate\_revoked:** A certificate has been revoked by its signer.
- **certificate\_expired:** A certificate has expired.
- **certificate\_unknown:** Some other unspecified issue arose in processing the certificate, rendering it unacceptable.

## Handshake Protocol

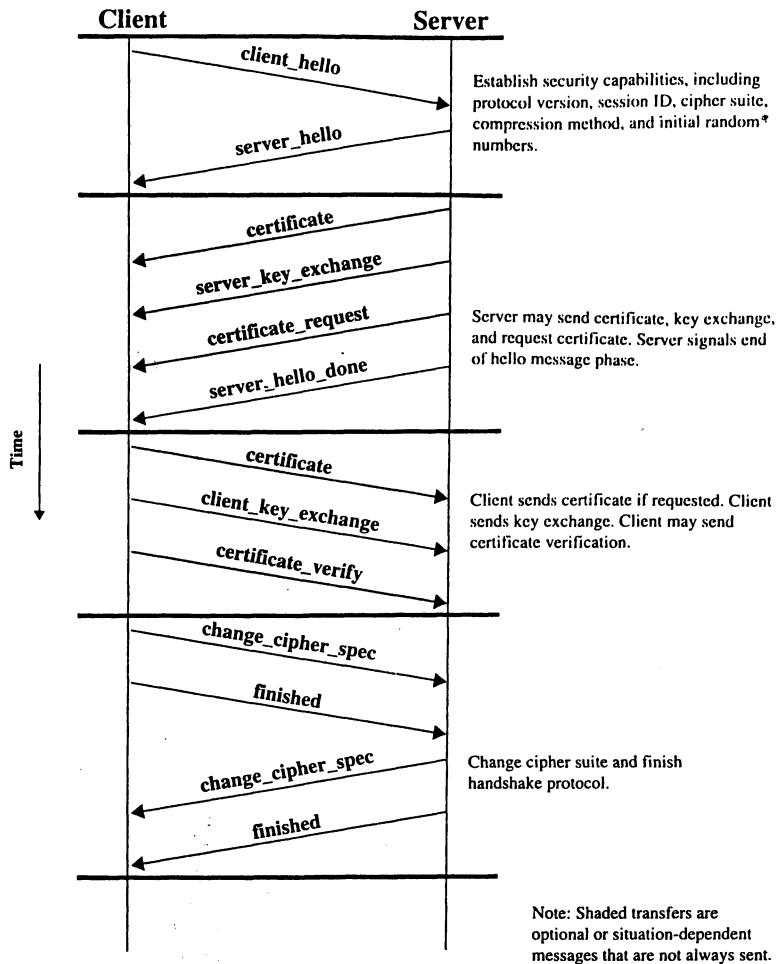
The most complex part of SSL is the Handshake Protocol. This protocol allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in an SSL record. The handshake protocol is used before any application data is transmitted.

The Handshake Protocol consists of a series of messages exchanged by client and server. All of these have the format shown in Figure 14.5c. Each message has three fields:

- **Type (1 byte):** Indicates one of 10 messages. Table 14.2 lists the defined message types.
- **Length (3 bytes):** The length of the message in bytes.

**Table 14.2** SSL Handshake Protocol Message Types

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value



**Figure 14.6** Handshake Protocol Action.

- **Content ( $\geq 1$  byte):** The parameters associated with this message; these are listed in Table 14.2.

Figure 14.6 shows the initial exchange needed to establish a logical connection between client and server. The exchange can be viewed as having four phases.

#### Phase 1. Establish Security Capabilities

This phase is used to initiate a logical connection and to establish the security capabilities that will be associated with it. The exchange is initiated by the client, which sends a **client\_hello** message with the following parameters:

- **Version:** The highest SSL version understood by the client.
- **Random:** A client-generated random structure, consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator. These values serve as nonces and are used during key exchange to prevent replay attacks.

- **Session ID:** A variable-length session identifier. A nonzero value indicates that the client wishes to update the parameters of an existing connection or create a new connection on this session. A zero value indicates that the client wishes to establish a new connection on a new session.
- **CipherSuite:** This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec; these are discussed subsequently.
- **Compression Method:** This is a list of the compression methods the client supports.

After sending the `client_hello` message, the client waits for the **server\_hello message**, which contains the same parameters as the `client_hello` message. For the `server_hello` message, the following conventions apply. The Version field contains the lower of the version suggested by the client and the highest supported by the server. The Random field is generated by the server and is independent of the client's Random field. If the SessionID field of the client was nonzero, the same value is used by the server; otherwise the server's SessionID field contains the value for a new session. The CipherSuite field contains the single cipher suite selected by the server from those proposed by the client. The Compression field contains the compression method selected by the server from those proposed by the client.

The first element of the Cipher Suite parameter is the key exchange method (i.e., the means by which the cryptographic keys for conventional encryption and MAC are exchanged). The following key exchange methods are supported:

- **RSA:** The secret key is encrypted with the receiver's RSA public key. A public-key certificate for the receiver's key must be made available.
- **Fixed Diffie-Hellman:** This is a Diffie-Hellman key exchange in which the server's certificate contains the Diffie-Hellman public parameters signed by the certificate authority (CA). That is, the public-key certificate contains the Diffie-Hellman public-key parameters. The client provides its Diffie-Hellman public key parameters either in a certificate, if client authentication is required, or in a key exchange message.
- **Ephemeral Diffie-Hellman:** This technique is used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, signed using the sender's private RSA or DSS key. The receiver can use the corresponding public key to verify the signature. Certificates are used to authenticate the public keys. This would appear to be the most secure of the three Diffie-Hellman options because it results in a temporary, authenticated key.
- **Anonymous Diffie-Hellman:** The base Diffie-Hellman algorithm is used, with no authentication. That is, each side sends its public Diffie-Hellman parameters to the other, with no authentication. This approach is vulnerable to man-in-the-middle attacks, in which the attacker conducts anonymous Diffie-Hellman with both parties.
- **Fortezza:** The technique defined for the Fortezza scheme.

Following the definition of a key exchange method is the CipherSpec, which includes the following fields:

- **CipherAlgorithm:** Any of the algorithms mentioned earlier: RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza
- **MACAlgorithm:** MD5 or SHA-1
- **CipherType:** Stream or Block
- **IsExportable:** True or False
- **HashSize:** 0, 16 (for MD5), or 20 (for SHA-1) bytes
- **Key Material:** A sequence of bytes that contain data used in generating the write keys
- **IV Size:** The size of the Initialization Value for Cipher Block Chaining (CBC) encryption

### Phase 2. Server Authentication and Key Exchange

The server begins this phase by sending its certificate, if it needs to be authenticated; the message contains one or a chain of X.509 certificates. The **certificate message** is required for any agreed-on key exchange method except anonymous Diffie-Hellman. Note that if fixed Diffie-Hellman is used, this certificate message functions as the server's key exchange message because it contains the server's public Diffie-Hellman parameters.

Next, a **server\_key\_exchange message** may be sent if it is required. It is not required in two instances: (1) The server has sent a certificate with fixed Diffie-Hellman parameters, or (2) RSA key exchange is to be used. The server\_key\_exchange message is needed for the following:

- **Anonymous Diffie-Hellman:** The message content consists of the two global Diffie-Hellman values (a prime number and a primitive root of that number) plus the server's public Diffie-Hellman key (see Figure 6.16),
- **Ephemeral Diffie-Hellman:** The message content includes the three Diffie-Hellman parameters provided for anonymous Diffie-Hellman, plus a signature of those parameters.
- **RSA key exchange, in which the server is using RSA but has a signature-only RSA key:** Accordingly, the client cannot simply send a secret key encrypted with the server's public key. Instead, the server must create a temporary RSA public/private key pair and use the server\_key\_exchange message to send the public key. The message content includes the two parameters of the temporary RSA public key (exponent and modulus; see Figure 6.5) plus a signature of those parameters.
- **Fortezza**

Some further details about the signatures are warranted. As usual, a signature is created by taking the hash of a message and encrypting it with the sender's public key. In this case the hash is defined as

```
hash(ClientHello.random || ServerHello.random || ServerParams)
```

So the hash covers not only the Diffie-Hellman or RSA parameters, but also the two nonces from the initial hello messages. This ensures against replay attacks and misrepresentation. In the case of a DSS signature, the hash is performed using the SHA-1 algorithm. In the case of an RSA signature, both an MD5 and an SHA-1 hash are calculated, and the concatenation of the two hashes (36 bytes) is encrypted with the server's public key.

Next, a nonanonymous server (server not using anonymous Diffie-Hellman) can request a certificate from the client. The **certificate\_request message** includes two parameters: `certificate_type` and `certificate_authorities`. The certificate type indicates the public-key algorithm and its use:

- RSA, signature only
- DSS, signature only
- RSA for fixed Diffie-Hellman; in this case the signature is used only for authentication, by sending a certificate signed with RSA
- DSS for fixed Diffie-Hellman; again, used only for authentication
- RSA for ephemeral Diffie-Hellman
- DSS for ephemeral Diffie-Hellman
- Fortezza

The second parameter in the `certificate_request` message is a list of the distinguished names of acceptable certificate authorities.

The final message in Phase 2, and one that is always required, is the **server\_done message**, which is sent by the server to indicate the end of the server hello and associated messages. After sending this message, the server will wait for a client response. This message has no parameters.

### Phase 3. Client Authentication and Key Exchange

Upon receipt of the `server_done` message, the client should verify that the server provided a valid certificate if required and check that the `server_hello` parameters are acceptable. If all is satisfactory, the client sends one or more messages back to the server.

If the server has requested a certificate, the client begins this phase by sending a **certificate message**. If no suitable certificate is available, the client sends a `no_certificate` alert instead.

Next is the **client\_key\_exchange message**, which must be sent in this phase. The content of the message depends on the type of key exchange, as follows:

- **RSA:** The client generates a 48-byte *pre-master secret* and encrypts with the public key from the server's certificate or temporary RSA key from a `server_key_exchange` message. Its use to compute a *master secret* is explained later.
- **Ephemeral or Anonymous Diffie-Hellman:** The client's public Diffie-Hellman parameters are sent.

- **Fixed Diffie-Hellman:** The client's public Diffie-Hellman parameters were sent in a certificate message, so the content of this message is null.
- **Fortezza:** The client's Fortezza parameters are sent.

Finally, in this phase, the client may send a **certificate\_verify message** to provide explicit verification of a client certificate. This message is only sent following any client certificate that has signing capability (i.e., all certificates except those containing fixed Diffie-Hellman parameters). This message signs a hash code based on the preceding messages, defined as follows:

```
CertificateVerify.signature.md5_hash
MD5(master_secret || pad_2 || MD5(handshake_messages || master_secret || pad_1));
Certificate.signature.sha_hash
SHA(master_secret || pad_2 || SHA(handshake_messages || master_secret || pad_1));
```

where pad\_1 and pad\_2 are the values defined earlier for the MAC, handshake\_messages refers to all Handshake Protocol messages sent or received starting at client\_hello but not including this message, and master\_secret is the calculated secret whose construction is explained later in this section. If the user's private key is DSS, then it is used to encrypt the SHA-1 hash. If the user's private key is RSA, it is used to encrypt the concatenation of the MD5 and SHA-1 hashes. In either case, the purpose is to verify the client's ownership of the private key for the client certificate. Even if someone is misusing the client's certificate, he or she would be unable to send this message.

#### Phase 4. Finish

This phase completes the setting up of a secure connection. The client sends a **change\_cipher\_spec message** and copies the pending CipherSpec into the current CipherSpec. Note that this message is not considered part of the Handshake Protocol but is sent using the Change Cipher Spec Protocol. The client then immediately sends the **finished message** under the new algorithms, keys, and secrets. The finished message verifies that the key exchange and authentication processes were successful. The content of the finished message is the concatenation of two hash values:

```
MD5(master_secret || pad_2 || MD5(handshake_messages || Sender || master_secret || pad_1))
SHA(master_secret || pad_2 || SHA(handshake_messages || Sender || master_secret || pad_1))
```

where Sender is a code that identifies that the sender is the client and handshake\_messages is all of the data from all handshake messages up to but not including this message.

In response to these two messages, the server sends its own change\_cipher\_spec message, transfers the pending to the current CipherSpec, and sends its finished message. At this point the handshake is complete and the client and server may begin to exchange application layer data.

## Cryptographic Computations

Two further items are of interest: the creation of a shared master secret by means of the key exchange, and the generation of cryptographic parameters from the master secret.

### Master Secret Creation

The shared master secret is a one-time 48-byte value (384 bits) generated for this session by means of secure key exchange. The creation is in two stages. First, a `pre_master_secret` is exchanged. Second, the `master_secret` is calculated by both parties. For `pre_master_secret` exchange, there are two possibilities:

- **RSA:** A 48-byte `pre_master_secret` is generated by the client, encrypted with the server's public RSA key, and sent to the server. The server decrypts the ciphertext using its private key to recover the `pre_master_secret`.
- **Diffie-Hellman:** Both client and server generate a Diffie-Hellman public key. After these are exchanged, each side performs the Diffie-Hellman calculation to create the shared `pre_master_secret`.

Both sides now compute the `master_secret` as follows:

```
master_secret = MD5(pre_master_secret || SHA('A' || pre_master_secret ||
    ClientHello.random || ServerHello.random)) ||
    MD5(pre_master_secret || SHA('BB' || pre_master_secret ||
    ClientHello.random || ServerHello.random)) ||
    MD5(pre_master_secret || SHA('CCC' || pre_master_secret ||
    ClientHello.random || ServerHello.random))
```

where `ClientHello.random` and `ServerHello.random` are the two nonce values exchanged in the initial hello messages.

### Generation of Cryptographic Parameters

CipherSpecs require a client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV, which are generated from the master secret in that order. These parameters are generated from the master secret by hashing the master secret into a sequence of secure bytes of sufficient length for all needed parameters.

The generation of the key material from the master secret uses the same format for generation of the master secret from the pre-master secret:

```
key_block = MD5(master_secret || SHA('A' || master_secret ||
    ServerHello.random || ClientHello.random)) ||
    MD5(master_secret || SHA('BB' || master_secret ||
    ServerHello.random || ClientHello.random)) ||
    MD5(master_secret || SHA('CCC' || master_secret ||
    ServerHello.random || ClientHello.random)) || ...
```



until enough output has been generated. The result of this algorithmic structure is a pseudorandom function. We can view the master\_secret as the pseudorandom seed value to the function. The client and server random numbers can be viewed as salt values to complicate cryptanalysis (see Chapter 15 for a discussion of the use of salt values).

### Transport Layer Security

TLS is an IETF standardization initiative whose goal is to produce an Internet standard version of SSL. The current draft version of TLS is very similar to SSLv3. In this section, we highlight the differences.

#### Version Number

The TLS Record Format is the same as that of the SSL Record Format (Figure 14.4), and the fields in the header have the same meanings. The one difference is in version values. For the current draft of TLS, the Major Version is 3 and the Minor Version is 1.

#### Message Authentication Code

There are two differences between the SSLv3 and TLS MAC schemes: the actual algorithm and the scope of the MAC calculation. TLS makes use of the HMAC algorithm defined in RFC 2104. Recall from Chapter 9 that HMAC is defined as follows:

$$\text{HMAC}_K = H[(K^+ \oplus \text{opad}) \parallel H[(K^+ \oplus \text{ipad}) \parallel M]]$$

where

- H = embedded hash function (for TLS, either MD5 or SHA-1)
- M = message input to HMAC
- K<sup>+</sup> = secret key padded with zeros on the left so that the result is equal to the block length of the hash code (for MD5 and SHA-1, block length = 512 bits)
- ipad = 00110110 (36 in hexadecimal) repeated 64 times (512 bits)
- opad = 01011100 (5C in hexadecimal) repeated 64 times (512 bits)

SSLv3 uses the same algorithm, except that the padding bytes are concatenated with the secret key rather than being XORed with the secret key padded to the block length. The level of security should be about the same in both cases.

For TLS, the MAC calculation encompasses the fields indicated in the following expression:

```
HMAC_hash(MAC_write_secret, seq_num || TLSCompressed.type ||
          TLSCompressed.version || TLSCompressed.length || TLSCompressed.fragment))
```

The MAC calculation covers all of the fields covered by the SSLv3 calculation, plus the field TLSCompressed.version, which is the version of the protocol being employed.

### Pseudorandom Function

TLS makes use of a pseudorandom function referred to as PRF to expand secrets into blocks of data for purposes of key generation or validation. The objective is to make use of a relatively small shared secret value but to generate longer blocks of data in a way that is secure from the kinds of attacks made on hash functions and MACs. The PRF is based on the following data expansion function (Figure 14.7):

$$\begin{aligned} \text{P\_hash}(\text{secret}, \text{seed}) = & \text{HMAC\_hash}(\text{secret}, A(1) \parallel \text{seed}) \parallel \\ & \text{HMAC\_hash}(\text{secret}, A(2) \parallel \text{seed}) \parallel \\ & \text{HMAC\_hash}(\text{secret}, A(3) \parallel \text{seed}) \parallel \dots \end{aligned}$$

where  $A()$  is defined as

$$\begin{aligned} A(0) &= \text{seed} \\ A(i) &= \text{HMAC\_hash}(\text{secret}, A(i-1)) \end{aligned}$$

The data expansion function makes use of the HMAC algorithm, with either MD5 or SHA-1 as the underlying hash function. As can be seen, P\_hash can be iterated as many times as necessary to produce the required quantity of data. For example,

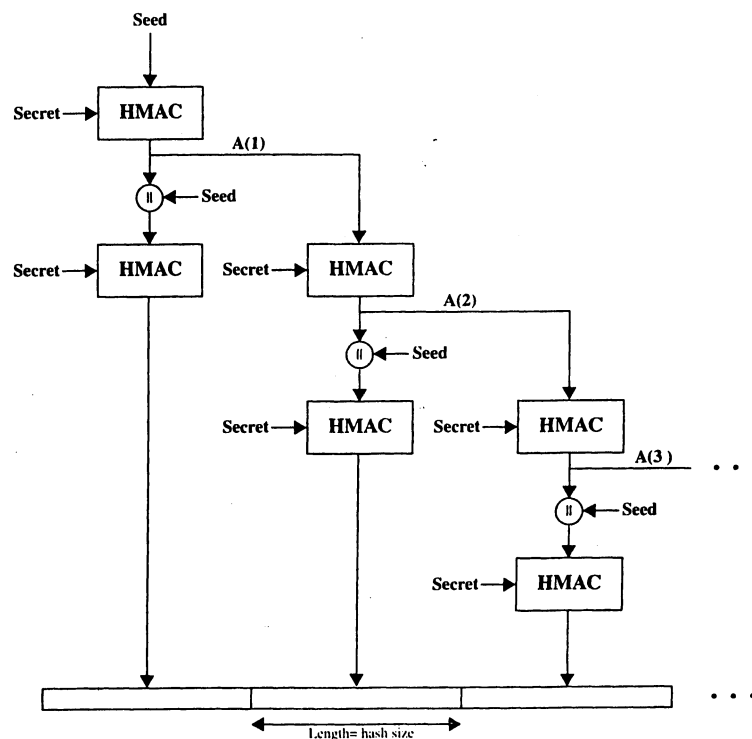


Figure 14.7 TLS Function P\_hash (secret, seed).

if P\_SHA-1 was used to generate 64 bytes of data, it would have to be iterated four times, producing 80 bytes of data, of which the last 16 would be discarded. In this case, P\_MD5 would also have to be iterated four times, producing exactly 64 bytes of data. Note that each iteration involves two executions of HMAC, each of which in turn involves two executions of the underlying hash algorithm.

To make PRF as secure as possible, it uses two hash algorithms in a way that should guarantee its security if either algorithm remains secure. PRF is defined as

$$\text{PRF}(\text{secret}, \text{label}, \text{seed}) = \text{P\_MD5}(\text{S1}, \text{label} \parallel \text{seed}) \oplus \text{P\_SHA-1}(\text{S2}, \text{label} \parallel \text{seed})$$

PRF takes as input a secret value, an identifying label, and a seed value and produces an output of arbitrary length. The output is created by splitting the secret value into two halves (S1 and S2) and performing P\_hash on each half, using MD5 on one half and SHA on the other half. The two results are exclusive-ORed to produce the output; for this purpose, P\_MD5 will generally have to be iterated more times than P\_SHA to produce an equal amount of data for input to the exclusive-OR function.

### Alert Codes

TLS supports all of the alert codes defined in SSLv3 with the exception of `no_certificate`. A number of additional codes are defined in TLS; of these, the following are always fatal:

- **decryption\_failed:** A ciphertext decrypted in an invalid way; either it was not an even multiple of the block length or its padding values, when checked, were incorrect.
- **record\_overflow:** A TLS record was received with a payload (ciphertext) whose length exceeds  $2^{14} + 2048$  bytes, or the ciphertext decrypted to a length of greater than  $2^{14} + 1024$  bytes.
- **unknown\_ca:** A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or could not be matched with a known, trusted CA.
- **access\_denied:** A valid certificate was received, but when access control was applied, the sender decided not to proceed with the negotiation.
- **decode\_error:** A message could not be decoded because a field was out of its specified range or the length of the message was incorrect.
- **export\_restriction:** A negotiation not in compliance with export restrictions on key length was detected.
- **protocol\_version:** The protocol version the client attempted to negotiate is recognized but not supported.
- **insufficient\_security:** Returned instead of `handshake_failure` when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client.
- **internal\_error:** An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue.

The remainder of the new alerts are the following:

- **decrypt\_error:** A handshake cryptographic operation failed, including being unable to verify a signature, decrypt a key exchange, or validate a finished message.
- **user\_canceled:** This handshake is being canceled for some reason unrelated to a protocol failure.
- **no\_renegotiation:** Sent by a client in response to a hello request or by the server in response to a client hello after initial handshaking. Either of these messages would normally result in renegotiation, but this alert indicates that the sender is not able to renegotiate. This message is always a warning.

### Cipher Suites

There are several small differences between the cipher suites available under SSLv3 and under TLS:

- **Key Exchange:** TLS supports all of the key exchange techniques of SSLv3 with the exception of Fortezza.
- **Symmetric Encryption Algorithms:** TLS includes all of the symmetric encryption algorithms found in SSLv3, with the exception of Fortezza.

### Client Certificate Types

TLS defines the following certificate types to be requested in a certificate\_request message: `rsa_sign`, `dss_sign`, `rsa_fixed_dh`, and `dss_fixed_dh`. These are all defined in SSLv3. In addition, SSLv3 includes `rsa_ephemeral_dh`, `dss_ephemeral_dh`, and `fortezza_kea`. Ephemeral Diffie-Hellman involves signing the Diffie-Hellman parameters with either RSA or DSS; for TLS, the `rsa_sign` and `dss_sign` types are used for that function; a separate signing type is not needed to sign Diffie-Hellman parameters. TLS does not include the Fortezza scheme.

### Certificate\_Verify and Finished Messages

In the TLS certificate\_verify message, the MD5 and SHA-1 hashes are calculated only over handshake\_messages. Recall that for SSLv3, the hash calculation also included the master secret and pads. These extra fields were felt to add no additional security.

As with the finished message in SSLv3, the finished message in TLS is a hash based on the shared master\_secret, the previous handshake messages, and a label that identifies client or server. The calculation is somewhat different. For TLS, we have

$$\text{PRF}(\text{master\_secret}, \text{finished\_label}, \text{MD5}(\text{handshake\_messages}) \parallel \text{SHA-1}(\text{handshake\_messages}))$$

where `finished_label` is the string “client finished” for the client and “server finished” for the server.

### Cryptographic Computations

The `pre_master_secret` for TLS is calculated in the same way as in SSLv3. As in SSLv3, the `master_secret` in TLS is calculated as a hash function of the `pre_`

master\_secret and the two hello random numbers. The form of the TLS calculation is different from that of SSLv3 and is defined as follows:

```
master_secret =
    PRF(pre_master_secret, "master secret", ClientHello.random || ServerHello.random)
```

The algorithm is performed until 48 bytes of pseudorandom output are produced. The calculation of the key block material (MAC secret keys, session encryption keys, and IVs) is defined as follows:

```
key_block =
    PRF(master_secret, "key expansion",
        SecurityParameters.server_random || SecurityParameters.client_random)
```

until enough output has been generated. As with SSLv3, the key\_block is a function of the master\_secret and the client and server random numbers, but for TLS the actual algorithm is different.

### Padding

In SSL, the padding added prior to encryption of user data is the minimum amount required so that the total size of the data to be encrypted is a multiple of the cipher's block length. In TLS, the padding can be any amount that results in a total that is a multiple of the cipher's block length, up to a maximum of 255 bytes. For example, if the plaintext (or compressed text if compression is used) plus MAC plus padding.length byte is 79 bytes long, then the padding length, in bytes, can be 1, 9, 17, and so on, up to 249. A variable padding length may be used to frustrate attacks based on an analysis of the lengths of exchanged messages.

## 14.3 SECURE ELECTRONIC TRANSACTION

SET is an open encryption and security specification designed to protect credit card transactions on the Internet. The current version, SETv1, emerged from a call for security standards by MasterCard and Visa in February 1996. A wide range of companies were involved in developing the initial specification, including IBM, Microsoft, Netscape, RSA, Terisa, and VeriSign. Beginning in 1996, there have been numerous tests of the concept, and by 1998 the first wave of SET-compliant products was available.

SET is not itself a payment system. Rather it is a set of security protocols and formats that enables users to employ the existing credit card payment infrastructure on an open network, such as the Internet, in a secure fashion. In essence, SET provides three services:

- Provides a secure communications channel among all parties involved in a transaction
- Provides trust by the use of X.509v3 digital certificates
- Ensures privacy because the information is only available to parties in a transaction when and where necessary

SET is a complex specification defined in three books issued in May of 1997:

- Book 1: Business Description (80 pages)
- Book 2: Programmer's Guide (629 pages)
- Book 3: Formal Protocol Definition (262 pages)

This is a total of 971 pages of specification. In contrast, the SSLv3 specification is 63 pages long and the TLS specification is 71 pages long. Accordingly, only a summary of this many-faceted specification is provided in this section.

### SET Overview

A good way to begin our discussion of SET is to look at the business requirements for SET, its key features, and the participants in SET transactions.

#### Requirements

Book 1 of the SET specification lists the following business requirements for secure payment processing with credit cards over the Internet and other networks:

- **Provide confidentiality of payment and ordering information:** It is necessary to assure cardholders that this information is safe and accessible only to the intended recipient. Confidentiality also reduces the risk of fraud by either party to the transaction or by malicious third parties. SET uses encryption to provide confidentiality.
- **Ensure the integrity of all transmitted data:** That is, ensure that no changes in content occur during transmission of SET messages. Digital signatures are used to provide integrity.
- **Provide authentication that a cardholder is a legitimate user of a credit card account:** A mechanism that links a cardholder to a specific account number reduces the incidence of fraud and the overall cost of payment processing. Digital signatures and certificates are used to verify that a cardholder is a legitimate user of a valid account.
- **Provide authentication that a merchant can accept credit card transactions through its relationship with a financial institution:** This is the complement to the preceding requirement. Cardholders need to be able to identify merchants with whom they can conduct secure transactions. Again, digital signatures and certificates are used.
- **Ensure the use of the best security practices and system design techniques to protect all legitimate parties in an electronic commerce transaction:** SET is a well-tested specification based on highly secure cryptographic algorithms and protocols.
- **Create a protocol that neither depends on transport security mechanisms nor prevents their use:** SET can securely operate over a "raw" TCP/IP stack. However, SET does not interfere with the use of other security mechanisms, such as IPSec and SSL/TLS.

- **Facilitate and encourage interoperability among software and network providers:** The SET protocols and formats are independent of hardware platform, operating system, and Web software.

#### Key Features of SET

To meet the requirements just outlined, SET incorporates the following features:

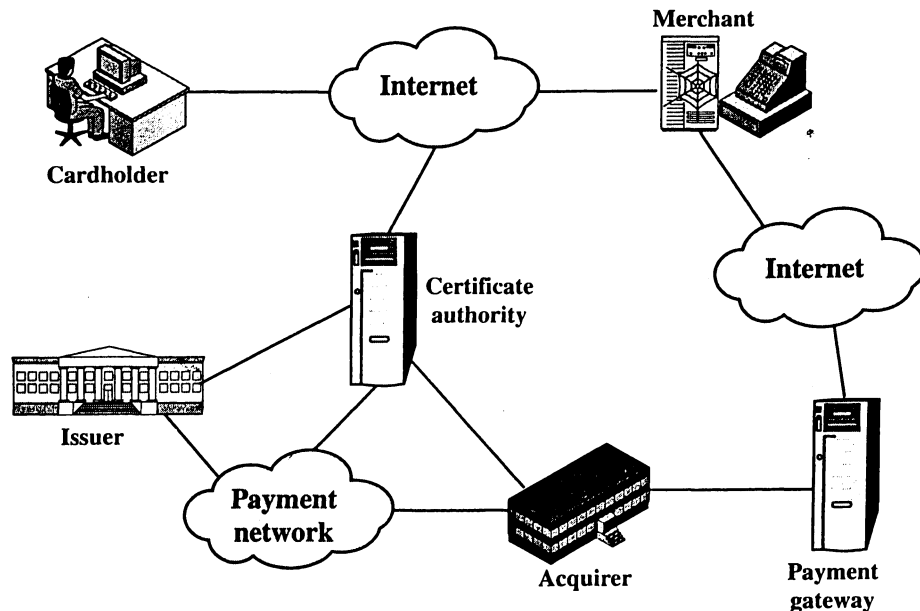
- **Confidentiality of information:** Cardholder account and payment information is secured as it travels across the network. An interesting and important feature of SET is that it prevents the merchant from learning the cardholder's credit card number; this is only provided to the issuing bank. Conventional encryption by DES is used to provide confidentiality.
- **Integrity of data:** Payment information sent from cardholders to merchants includes order information, personal data, and payment instructions. SET guarantees that these message contents are not altered in transit. RSA digital signatures, using SHA-1 hash codes, provide message integrity. Certain messages are also protected by HMAC using SHA-1.
- **Cardholder account authentication:** SET enables merchants to verify that a cardholder is a legitimate user of a valid card account number. SET uses X.509v3 digital certificates with RSA signatures for this purpose.
- **Merchant authentication:** SET enables cardholders to verify that a merchant has a relationship with a financial institution allowing it to accept payment cards. SET uses X.509v3 digital certificates with RSA signatures for this purpose.

Note that unlike IPSec and SSL/TLS, SET provides only one choice for each cryptographic algorithm. This makes sense, because SET is a single application with a single set of requirements, whereas IPSec and SSL/TLS are intended to support a range of applications.

#### SET Participants

Figure 14.8 indicates the participants in the SET system, which include the following:

- **Cardholder:** In the electronic environment, consumers and corporate purchasers interact with merchants from personal computers over the Internet. A cardholder is an authorized holder of a payment card (e.g., MasterCard, Visa) that has been issued by an issuer.
- **Merchant:** A merchant is a person or organization that has goods or services to sell to the cardholder. Typically, these goods and services are offered via a Web site or by electronic mail. A merchant that accepts payment cards must have a relationship with an acquirer.
- **Issuer:** This is a financial institution, such as a bank, that provides the cardholder with the payment card. Typically, accounts are applied for and opened by mail or in person. Ultimately, it is the issuer that is responsible for the payment of the debt of the cardholder.



**Figure 14.8** Secure Electronic Commerce Components.

- **Acquirer:** This is a financial institution that establishes an account with a merchant and processes payment card authorizations and payments. Merchants will usually accept more than one credit card brand but do not want to deal with multiple bankcard associations or with multiple individual issuers. The acquirer provides authorization to the merchant that a given card account is active and that the proposed purchase does not exceed the credit limit. The acquirer also provides electronic transfer of payments to the merchant's account. Subsequently, the acquirer is reimbursed by the issuer over some sort of payment network for electronic funds transfer.
- **Payment Gateway:** This is a function operated by the acquirer or a designated third party that processes merchant payment messages. The payment gateway interfaces between SET and the existing bankcard payment networks for authorization and payment functions. The merchant exchanges SET messages with the payment gateway over the Internet, while the payment gateway has some direct or network connection to the acquirer's financial processing system.
- **Certification Authority (CA):** This is an entity that is trusted to issue X.509v3 public-key certificates for cardholders, merchants, and payment gateways. The success of SET will depend on the existence of a CA infrastructure available for this purpose. As was discussed in previous chapters, a hierarchy of CAs is used, so that participants need not be directly certified by a root authority.

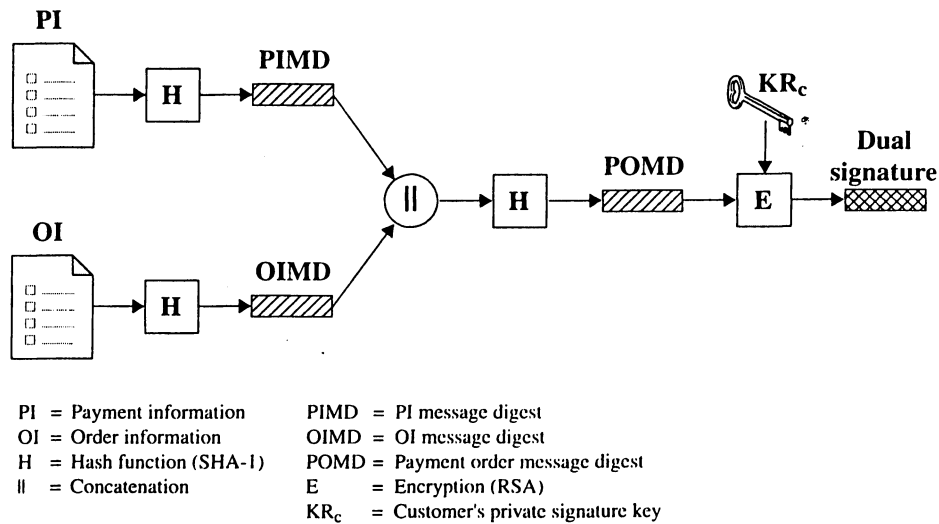
We now briefly describe the sequence of events that are required for a transaction. We will then look at some of the cryptographic details.



1. **The customer opens an account.** The customer obtains a credit card account, such as MasterCard or Visa, with a bank that supports electronic payment and SET.
2. **The customer receives a certificate.** After suitable verification of identity, the customer receives an X.509v3 digital certificate, which is signed by the bank. The certificate verifies the customer's RSA public key and its expiration date. It also establishes a relationship, guaranteed by the bank, between the customer's key pair and his or her credit card.
3. **Merchants have their own certificates.** A merchant who accepts a certain brand of card must be in possession of two certificates for two public keys owned by the merchant: one for signing messages, and one for key exchange. The merchant also needs a copy of the payment gateway's public-key certificate.
4. **The customer places an order.** This is a process that may involve the customer first browsing through the merchant's Web site to select items and determine the price. The customer then sends a list of the items to be purchased to the merchant, who returns an order form containing the list of items, their price, a total price, and an order number.
5. **The merchant is verified.** In addition to the order form, the merchant sends a copy of its certificate, so that the customer can verify that he or she is dealing with a valid store.
6. **The order and payment are sent.** The customer sends both order and payment information to the merchant, along with the customer's certificate. The order confirms the purchase of the items in the order form. The payment contains credit card details. The payment information is encrypted in such a way that it cannot be read by the merchant. The customer's certificate enables the merchant to verify the customer.
7. **The merchant requests payment authorization.** The merchant sends the payment information to the payment gateway, requesting authorization that the customer's available credit is sufficient for this purchase.
8. **The merchant confirms the order.** The merchant sends confirmation of the order to the customer.
9. **The merchant provides the goods or service.** The merchant ships the goods or provides the service to the customer.
10. **The merchant requests payment.** This request is sent to the payment gateway, which handles all of the payment processing.

### Dual Signature

Before looking at the details of the SET protocol, let us discuss an important innovation introduced in SET: the dual signature. The purpose of the dual signature is to link two messages that are intended for two different recipients. In this case, the customer wants to send the order information (OI) to the merchant and the payment information (PI) to the bank. The merchant does not need to know the customer's credit card number, and the bank does not need to know the details of the customer's order. The customer is afforded extra protection in terms of privacy by keeping these two items separate. However, the two items must be linked in a way that can be used to resolve disputes if necessary. The link is needed so that the customer can prove that this payment is intended for this order and not for some other goods or service.



**Figure 14.9** Construction of Dual Signature.

To see the need for the link, suppose that the customer sends the merchant two messages—a signed OI and a signed PI—and the merchant passes the PI on to the bank. If the merchant can capture another OI from this customer, the merchant could claim that this OI goes with the PI rather than the original OI. The linkage prevents this.

Figure 14.9 shows the use of a dual signature to meet the requirement of the preceding paragraph. The customer takes the hash (using SHA-1) of the PI and the hash of the OI. These two hashes are then concatenated and the hash of the result is taken. Finally, the customer encrypts the final hash with his or her private signature key, creating the dual signature. The operation can be summarized as follows:

$$DS = E_{KR_c}[H(H(PI)||H(OI))]$$

where  $KR_c$  is the customer's private signature key. Now suppose that the merchant is in possession of the dual signature (DS), the OI, and the message digest for the PI (PIMD). The merchant also has the public key of the customer, taken from the customer's certificate. Then the merchant can compute the following two quantities:

$$H(PIMD||H(OI)) \text{ and } D_{KU_c}[DS]$$

where  $KU_c$  is the customer's public signature key. If these two quantities are equal, then the merchant has verified the signature. Similarly, if the bank is in possession of DS, PI, the message digest for OI (OIMD), and the customer's public key, then the bank can compute the following:

$$H(H(PI)||OIMD) \text{ and } D_{KU_c}[DS]$$

Again, if these two quantities are equal, then the bank has verified the signature. In summary,

1. The merchant has received OI and verified the signature.
2. The bank has received PI and verified the signature.
3. The customer has linked the OI and PI and can prove the linkage.

For example, suppose the merchant wishes to substitute another OI in this transaction, to its advantage. It would then have to find another OI whose hash matches the existing OIMD. With SHA-1, this is deemed not to be feasible. Thus, the merchant cannot link another OI with this PI.

## Payment Processing

Table 14.3 lists the transaction types supported by SET. In what follows we look in some detail at the following transactions:

- Purchase request
- Payment authorization
- Payment capture

### Purchase Request

Before the Purchase Request exchange begins, the cardholder has completed browsing, selecting, and ordering. The end of this preliminary phase occurs when the merchant sends a completed order form to the customer. All of the preceding occurs without the use of SET.

The purchase request exchange consists of four messages: Initiate Request, Initiate Response, Purchase Request, and Purchase Response.

In order to send SET messages to the merchant, the cardholder must have a copy of the certificates of the merchant and the payment gateway. The customer requests the certificates in the **Initiate Request** message, sent to the merchant. This message includes the brand of the credit card that the customer is using. The message also includes an ID assigned to this request/response pair by the customer and a nonce used to ensure timeliness.

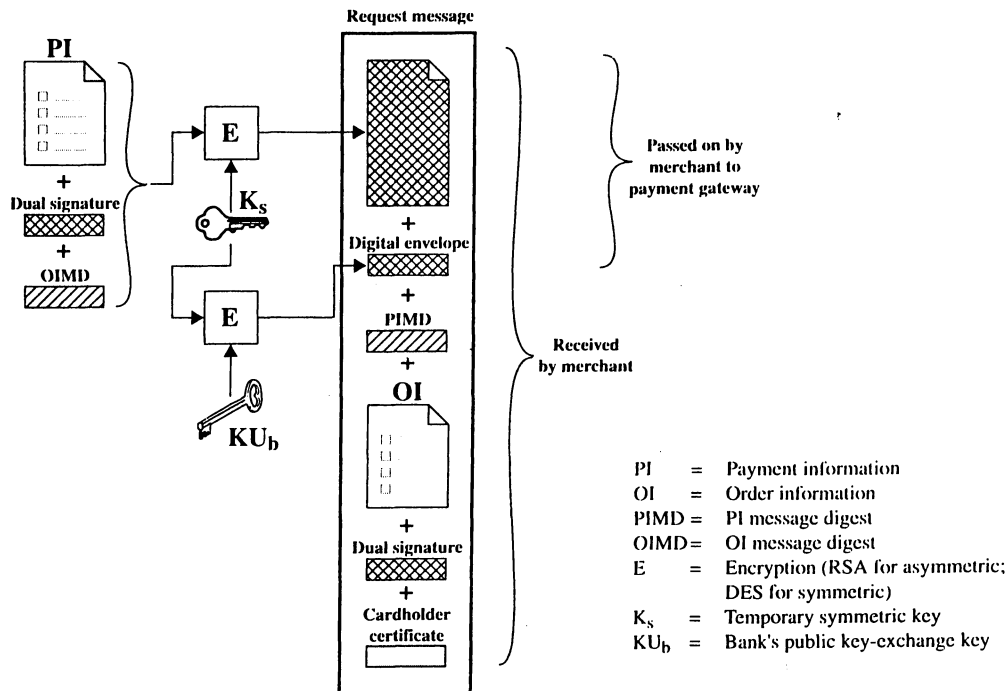
The merchant generates a response and signs it with its private signature key. The response includes the nonce from the customer, another nonce for the customer to return in the next message, and a transaction ID for this purchase transaction. In addition to the signed response, the **Initiate Response** message includes the merchant's signature certificate and the payment gateway's key exchange certificate.

The cardholder verifies the merchant and gateway certificates by means of their respective CA signatures and then creates the OI and PI. The transaction ID assigned by the merchant is placed in both the OI and PI. The OI does not contain explicit order data such as the number and price of items. Rather, it contains an order reference generated in the exchange between merchant and customer during the shopping phase before the first SET message. Next, the cardholder prepares the **Purchase Request** message (Figure 14.10). For this purpose, the cardholder generates a one-time symmetric encryption key,  $K_s$ . The message includes the following:

**Table 14.3** SET Transaction Types

Cardholder registration	Cardholders must register with a CA before they can send SET messages to merchants.
Merchant registration	Merchants must register with a CA before they can exchange SET messages with customers and payment gateways.
Purchase request	Message from customer to merchant containing OI for merchant and PI for bank.
Payment authorization	Exchange between merchant and payment gateway to authorize a given amount for a purchase on a given credit card account.
Payment capture	Allows the merchant to request payment from the payment gateway.
Certificate inquiry and status	If the CA is unable to complete the processing of a certificate request quickly, it will send a reply to the cardholder or merchant indicating that the requester should check back later. The cardholder or merchant sends the <i>Certificate Inquiry</i> message to determine the status of the certificate request and to receive the certificate if the request has been approved.
Purchase inquiry	Allows the cardholder to check the status of the processing of an order after the purchase response has been received. Note that this message does not include information such as the status of back-ordered goods but does indicate the status of authorization, capture, and credit processing.
Authorization reversal	Allows a merchant to correct previous authorization requests. If the order will not be completed, the merchant reverses the entire authorization. If part of the order will not be completed (such as when goods are back ordered), the merchant reverses part of the amount of the authorization.
Capture reversal	Allows a merchant to correct errors in capture requests such as transaction amounts that were entered incorrectly by a clerk.
Credit	Allows a merchant to issue a credit to a cardholder's account such as when goods are returned or were damaged during shipping. Note that the SET <i>Credit</i> message is always initiated by the merchant, not the cardholder. All communications between the cardholder and merchant that result in a credit being processed happen outside of SET.
Credit reversal	Allows a merchant to correct a previously request credit.
Payment gateway certificate request	Allows a merchant to query the payment gateway and receive a copy of the gateway's current key exchange and signature certificates.
Batch administration	Allows a merchant to communicate information to the payment gateway regarding merchant batches.
Error message	Indicates that a responder rejects a message because it fails format or content verification tests.

1. **Purchase-related information.** This information will be forwarded to the payment gateway by the merchant and consists of
  - The PI
  - The dual signature, calculated over the PI and OI, signed with the customer's private signature key
  - The OI message digest (OIMD)



**Figure 14.10** Cardholder Sends Purchase Request.

The OIMD is needed for the payment gateway to verify the dual signature, as explained previously. All of these items are encrypted with K<sub>s</sub>. The final item is

- The digital envelope. This is formed by encrypting K<sub>s</sub> with the payment gateway's public key-exchange key. It is called a digital envelope because this envelope must be opened (decrypted) before the other items listed previously can be read.

The value of K<sub>s</sub> is not made available to the merchant. Therefore, the merchant cannot read any of this payment-related information.

2. **Order-related information.** This information is needed by the merchant and consists of

- The OI
- The dual signature, calculated over the PI and OI, signed with the customer's private signature key
- The PI message digest (PIMD)

The PIMD is needed for the merchant to verify the dual signature. Note that the OI is sent in the clear.

3. **Cardholder certificate.** This contains the cardholder's public signature key. It is needed by the merchant and by the payment gateway.

When the merchant receives the Purchase Request message, it performs the following actions (Figure 14.11):

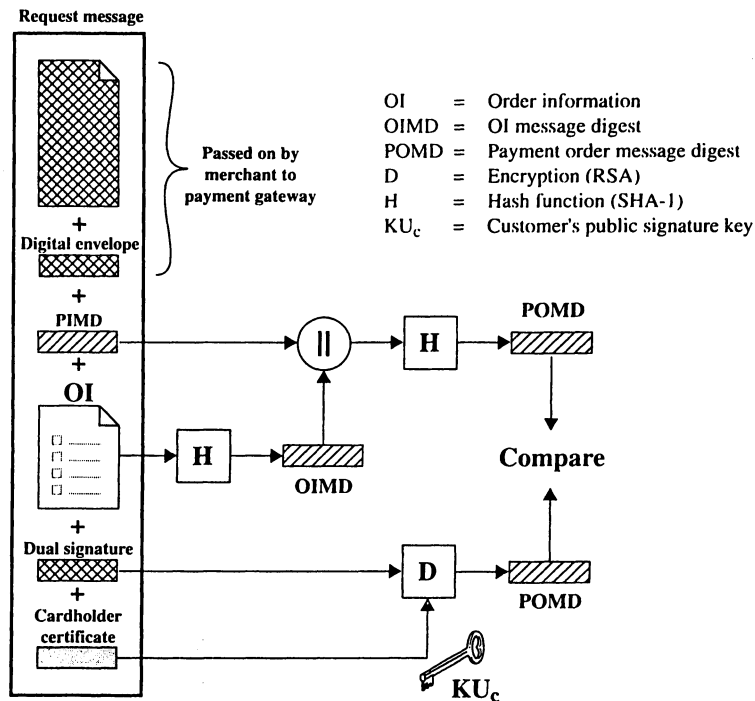


Figure 14.11 Merchant Verifies Customer Purchase Request.

1. Verifies the cardholder certificates by means of its CA signatures.
2. Verifies the dual signature using the customer's public signature key. This ensures that the order has not been tampered with in transit and that it was signed using the cardholder's private signature key.
3. Processes the order and forwards the payment information to the payment gateway for authorization (described later).
4. Sends a purchase response to the cardholder.

The **Purchase Response** message includes a response block that acknowledges the order and references the corresponding transaction number. This block is signed by the merchant using its private signature key. The block and its signature are sent to the customer, along with the merchant's signature certificate.

When the cardholder software receives the purchase response message, it verifies the merchant's certificate and then verifies the signature on the response block. Finally, it takes some action based on the response, such as displaying a message to the user or updating a database with the status of the order.

### Payment Authorization

During the processing of an order from a cardholder, the merchant authorizes the transaction with the payment gateway. The payment authorization ensures that the transaction was approved by the issuer. This authorization guarantees that the

merchant will receive payment; the merchant can therefore provide the services or goods to the customer. The payment authorization exchange consists of two messages: Authorization Request and Authorization response.

The merchant sends an **Authorization Request** message to the payment gateway consisting of

1. **Purchase-related information.** This information was obtained from the customer and consists of:
  - The PI
  - The dual signature, calculated over the PI and OI, signed with the customer's private signature key
  - The OI message digest (OIMD)
  - The digital envelope
2. **Authorization-related information.** This information is generated by the merchant and consists of
  - An authorization block that includes the transaction ID, signed with the merchant's private signature key and encrypted with a one-time symmetric key generated by the merchant
  - A digital envelope. This is formed by encrypting the one-time key with the payment gateway's public key-exchange key.
3. **Certificates.** The merchant includes the cardholder's signature key certificate (used to verify the dual signature), the merchant's signature key certificate (used to verify the merchant's signature), and the merchant's key-exchange certificate (needed in the payment gateway's response).

The payment gateway performs the following tasks:

1. Verifies all certificates
2. Decrypts the digital envelope of the authorization block to obtain the symmetric key and then decrypts the authorization block
3. Verifies the merchant's signature on the authorization block
4. Decrypts the digital envelope of the payment block to obtain the symmetric key and then decrypts the payment block
5. Verifies the dual signature on the payment block
6. Verifies that the transaction ID received from the merchant matches that in the PI received (indirectly) from the customer
7. Requests and receives an authorization from the issuer

Having obtained authorization from the issuer, the payment gateway returns an **Authorization Response** message to the merchant. It includes the following elements:

1. **Authorization-related information.** Includes an authorization block, signed with the gateway's private signature key and encrypted with a one-time symmetric key generated by the gateway. Also includes a digital envelope that contains the one-time key encrypted with the merchant's public key-exchange key.

2. **Capture token information.** This information will be used to effect payment later. This block is of the same form as (1)—namely, a signed, encrypted capture token together with a digital envelope. This token is not processed by the merchant. Rather, it must be returned, as is, with a payment request.
3. **Certificate.** The gateway's signature key certificate.

With the authorization from the gateway, the merchant can provide the goods or service to the customer.

### Payment Capture

To obtain payment, the merchant engages the payment gateway in a payment capture transaction, consisting of a capture request and a capture response message.

For the **Capture Request** message, the merchant generates, signs, and encrypts a capture request block, which includes the payment amount and the transaction ID. The message also includes the encrypted capture token received earlier (in the Authorization Response) for this transaction, as well as the merchant's signature key and key-exchange key certificates.

When the payment gateway receives the capture request message, it decrypts and verifies the capture request block and decrypts and verifies the capture token block. It then checks for consistency between the capture request and capture token. It then creates a clearing request that is sent to the issuer over the private payment network. This request causes funds to be transferred to the merchant's account.

The gateway then notifies the merchant of payment in a **Capture Response** message. The message includes a capture response block that the gateway signs and encrypts. The message also includes the gateway's signature key certificate. The merchant software stores the capture response to be used for reconciliation with payment received from the acquirer.

## 14.4 RECOMMENDED READING

Two good treatments of web security are [RUBI97] and [GARF97]; the latter is somewhat more technical. The best detailed overview of SET is in Book 1 of the specification, available at the MasterCard and VISA SET Web sites. Another excellent overview is [MACG97].

**GARF97** Garfinkel, S., and Spafford, G. *Web Security & Commerce*. Cambridge, MA: O'Reilly and Associates, 1997.

**MACG97** Macgregor, R.; Ezvan, C.; Liguori, L.; and Han, J. *Secure Electronic Transactions: Credit Card Payment on the Web in Theory and Practice*. IBM RedBook SG24-4978-00, 1997. Available at [www.redbooks.ibm.com/SG244978](http://www.redbooks.ibm.com/SG244978).

**RUBI97** Rubin, A.; Geer, D.; and Ranum, M. *Web Security Sourcebook*. New York: Wiley, 1997.

Recommended Web sites:

- **Netscape's SSL Page:** Contains the SSL specification.
- **Transport Layer Security Charter:** Latest RFCs and internet drafts for TLS.
- **MasterCard SET Site:** Latest SET documents, glossary of terms, and application information.
- **Visa-Electronic Commerce Site:** Similar information to that at the MasterCard Site.



**14.5 PROBLEMS**

- 14.1** In SSL and TLS, why is there a separate Change Cipher Spec Protocol, rather than including a `change_cipher_spec` message in the Handshake Protocol?
- 14.2** Consider the following threats to Web security and describe how each is countered by a particular feature of SSL.
- a. Brute-Force Cryptanalytic Attack: An exhaustive search of the key space for a conventional encryption algorithm.
  - b. Known-Plaintext Dictionary Attack: Many messages will contain predictable plaintext, such as the HTTP GET command. An attacker constructs a dictionary containing every possible encryption of the known-plaintext message. When an encrypted message is intercepted, the attacker takes the portion containing the encrypted known plaintext and looks up the ciphertext in the dictionary. The ciphertext should match against an entry that was encrypted with the same secret key. If there are several matches, each of these can be tried against the full ciphertext to determine the right one. This attack is especially effective against small key sizes (e.g., 40-bit keys).
  - c. Replay Attack: Earlier SSL handshake messages are replayed.
  - d. Man-in-the-Middle Attack: An attacker interposes during key exchange, acting as the client to the server and as the server to the client.
  - e. Password Sniffing: Passwords in HTTP or other application traffic are eavesdropped.
  - f. IP Spoofing: Uses forged IP addresses to fool a host into accepting bogus data.
  - g. IP Hijacking: An active, authenticated connection between two hosts is disrupted and the attacker takes the place of one of the hosts.
  - h. SYN Flooding: An attacker sends TCP SYN messages to request a connection but does not respond to the final message to establish the connection fully. The attacked TCP module typically leaves the “half-open connection” around for a few minutes. Repeated SYN messages can clog the TCP module.
- 14.3** Based on what you have learned in this chapter, is it possible in SSL for the receiver to reorder SSL record blocks that arrive out of order? If so, explain how it can be done. If not, why not?



**PART  
FOUR**

**System Security**



# CHAPTER 15

## INTRUDERS AND VIRUSES

*What is the concept of defense: The parrying of a blow. What is its characteristic feature: Awaiting the blow.*

—*On War*, Carl Von Clausewitz

*They agreed that Graham should set the test for Charles Mabledene. It was neither more nor less than that Dragon should get Stern's code. If he had the 'in' at Utting which he claimed to have this should be possible, only loyalty to Moscow Centre would prevent it. If he got the key to the code he would prove his loyalty to London Central beyond a doubt.*

—*Talking to Strange Men*, Ruth Rendell

A significant security problem for networked systems is hostile, or at least unwanted, trespass by users or software. User trespass can take the form of unauthorized logon to a machine or, in the case of an authorized user, acquisition of privileges or performance of actions beyond those that have been authorized. Software trespass can take the form of a virus, worm, or Trojan horse.

All these attacks relate to network security because system entry can be achieved by means of a network. However, these attacks are not confined to network-based attacks. A user with access to a local terminal may attempt trespass without using an intermediate network. A virus or Trojan horse may be introduced into a system by means of a diskette. Only the worm is a uniquely network phenomenon. Thus, system trespass is an area in which the concerns of network security and computer security overlap.

Because the focus of this book is network security, we do not attempt a comprehensive analysis of either the attacks or the security countermea-

asures related to system trespass. Instead, in this chapter we present a broad overview of these concerns.

This chapter begins with the subject of intruders. First, we examine the nature of the attack and then look at strategies intended for prevention and, failing that, detection. Then we examine the well-publicized topic of viruses.

## 15.1 INTRUDERS

One of the two most publicized threats to security is the intruder (the other is viruses), generally referred to as a hacker or cracker. In an important early study of intrusion, Anderson [ANDE80] identified three classes of intruders:

- **Masquerader:** An individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account
- **Misfeasor:** A legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuses his or her privileges
- **Clandestine user:** An individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection

The masquerader is likely to be an outsider; the misfeasor generally is an insider; and the clandestine user can be either an outsider or an insider.

Intruder attacks range from the benign to the serious. At the benign end of the scale, there are many people who simply wish to explore internets and see what is out there. At the serious end are individuals who are attempting to read privileged data, perform unauthorized modifications to data, or disrupt the system.

The intruder threat has been well publicized, particularly because of the famous "Wily Hacker" incident of 1986–1987, documented by Cliff Stoll [STOL88, STOL89]. In 1990 there was a nationwide crackdown on illicit computer hackers, with arrests, criminal charges, one dramatic show trial, several guilty pleas, and confiscation of massive amounts of data and computer equipment [STER92]. Many people believed that the problem had been brought under control.

In fact, the problem has not been brought under control. To cite one example, a group at Bell Labs [BELL92, BELL93] has reported persistent and frequent attacks on its computer complex via the Internet over an extended period and from a variety of sources. At the time of these reports, the Bell group was experiencing the following:

- Attempts to copy the password file (discussed later) at a rate exceeding once every other day
- Suspicious remote procedure call (RPC) requests at a rate exceeding once per week
- Attempts to connect to nonexistent "bait" machines at least every two weeks

Benign intruders might be tolerable, although they do consume resources and may slow performance for legitimate users. However, there is no way in advance to know whether an intruder will be benign or malign. Consequently, even for systems with no particularly sensitive resources, there is a motivation to control this problem.

An example that dramatically illustrates the threat occurred at Texas A&M University [SAFF93]. In August 1992, the computer center there was notified that one of its machines was being used to attack computers at another location via the Internet. By monitoring activity, the computer center personnel learned that there were several outside intruders involved, who were running password-cracking routines on various computers (the site consists of a total of 12,000 interconnected machines). The center disconnected affected machines, plugged known security holes, and resumed normal operation. A few days later, one of the local system managers detected that the intruder attack had resumed. It turned out that the attack was far more sophisticated than had been originally believed. Files were found containing hundreds of captured passwords, including some on major and supposedly secure servers. In addition, one local machine had been set up as a hacker bulletin board, which the hackers used to contact each other and to discuss techniques and progress.

An analysis of this attack revealed that there were actually two levels of hackers. The high level were sophisticated users with a thorough knowledge of the technology; the low level were the “foot soldiers” who merely used the supplied cracking programs with little understanding of how they worked. This teamwork combined the two most serious weapons in the intruder armory: sophisticated knowledge of how to intrude and a willingness to spend countless hours “turning doorknobs” to probe for weaknesses.

One of the results of the growing awareness of the intruder problem has been the establishment of a number of computer emergency response teams (CERTs). These cooperative ventures collect information about system vulnerabilities and disseminate it to systems managers. Unfortunately, hackers can also gain access to CERT reports. In the Texas A&M incident, later analysis showed that the hackers had developed programs to test the attacked machines for virtually every vulnerability that had been announced by CERT. If even one machine had failed to respond promptly to a CERT advisory, it was wide open to such attacks.

In addition to running password-cracking programs, the intruders attempted to modify login software to enable them to capture passwords of users logging on to systems. This made it possible for them to build up an impressive collection of compromised passwords, which was made available on the bulletin board set up on one of the victim’s own machines.

We begin this section by looking at the techniques used for intrusion. Then we examine ways to prevent intrusion. Failing prevention, intrusion detection is a second line of defense and is discussed in the final subsection.

### **Intrusion Techniques**

The objective of the intruder is to gain access to a system or to increase the range of privileges accessible on a system. Generally, this requires the intruder to acquire information that should have been protected. In most cases, this information is in the

form of a user password. With knowledge of some other user's password, an intruder can log in to a system and exercise all the privileges accorded to the legitimate user.

Typically, a system must maintain a file that associates a password with each authorized user. If such a file is stored with no protection, then it is an easy matter to gain access to it and learn passwords. The password file can be protected in one of two ways:

- **One-way encryption:** The system stores only an encrypted form of the user's password. When the user presents a password, the system encrypts that password and compares it with the stored value. In practice, the system usually performs a one-way transformation (not reversible) in which the password is used to generate a key for the encryption function and in which a fixed-length output is produced.
- **Access control:** Access to the password file is limited to one or a very few accounts.

If one or both of these countermeasures are in place, some effort is needed for a potential intruder to learn passwords. On the basis of a survey of the literature and interviews with a number of password crackers, [ALVA90] reports the following techniques for learning passwords:

1. Try default passwords used with standard accounts that are shipped with the system. Many administrators do not bother to change these defaults.
2. Exhaustively try all short passwords (those of one to three characters).
3. Try words in the system's on-line dictionary or a list of likely passwords. Examples of the latter are readily available on hacker bulletin boards.
4. Collect information about users, such as their full names, the names of their spouse and children, pictures in their office, and books in their office that are related to hobbies.
5. Try users' phone numbers, Social Security numbers, and room numbers.
6. Try all legitimate license plate numbers for this state.
7. Use a Trojan horse (described in Section 15.2) to bypass restrictions on access.
8. Tap the line between a remote user and the host system.

The first six methods are various ways of guessing a password. If an intruder has to verify the guess by attempting to log in, it is a tedious and easily countered means of attack. For example, a system can simply reject any login after three password attempts, thus requiring the intruder to reconnect to the host to try again. Under these circumstances, it is not practical to try more than a handful of passwords. However, the intruder is unlikely to try such crude methods. For example, if an intruder can gain access with a low level of privileges to an encrypted password file, then the strategy would be to capture that file and then use the encryption mechanism of that particular system at leisure until a valid password that provided greater privileges was discovered.

Guessing attacks are feasible, and indeed highly effective, when a large number of guesses can be attempted automatically and each guess verified, without the guessing process being detectable. Later in this section, we have much to say about thwarting guessing attacks.



The seventh method of attack listed earlier, the Trojan horse, can be particularly difficult to counter. An example of a program that bypasses access controls was cited in [ALVA90]. A low-privilege user produced a game program and invited the system operator to use it in his or her spare time. The program did indeed play a game, but in the background it also contained code to copy the password file, which was unencrypted but access protected, into the user's file. Because the game was running under the operator's high-privilege mode, it was able to gain access to the password file.

The eighth attack listed, line tapping, is a matter of physical security. It can be countered with link encryption techniques, discussed in Section 5.1.

We turn now to a discussion of the two principal countermeasures: prevention and detection. Prevention is a challenging security goal and an uphill battle at all times. The difficulty stems from the fact that the defender must attempt to thwart all possible attacks, whereas the attacker is free to try to find the weakest link in the defense chain and attack at that point. Detection is concerned with learning of an attack, either before or after its success.

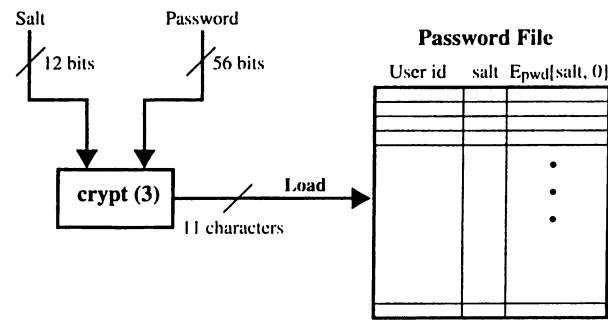
### Password Protection

The front line of defense against intruders is the password system. Virtually all multiuser systems require that a user provide not only a name or identifier (ID) but also a password. The password serves to authenticate the ID of the individual logging on to the system. In turn, the ID provides security in the following ways:

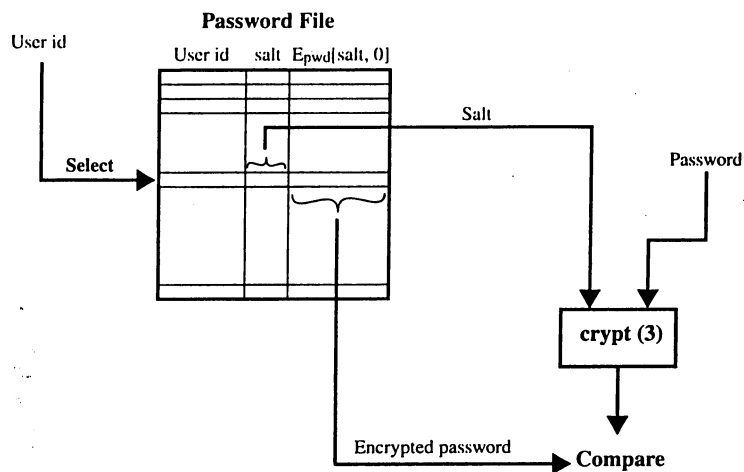
- The ID determines whether the user is authorized to gain access to a system. In some systems, only those who already have an ID filed on the system are allowed to gain access.
- The ID determines the privileges accorded to the user. A few users may have supervisory or "superuser" status that enables them to read files and perform functions that are especially protected by the operating system. Some systems have guest or anonymous accounts, and users of these accounts have more limited privileges than others.
- The ID is used in what is referred to as discretionary access control. For example, by listing the IDs of the other users, a user may grant permission to them to read files owned by that user.

### The Vulnerability of Passwords

To understand the nature of the attack, let us consider a scheme that is widely used on UNIX systems, in which passwords are never stored in the clear. Rather, the following procedure is employed (Figure 15.1a): Each user selects a password of up to eight printable characters in length. This is converted into a 56-bit value (using 7-bit ASCII) that serves as the key input to an encryption routine. The encryption routine, known as crypt(3), is based on DES. The DES algorithm is modified using a 12-bit "salt" value. Typically, this value is related to the time at which the password is assigned to the user. The modified DES algorithm is exercised with a data input consisting of a 64-bit block of zeros. The output of the algorithm then serves as input for a second encryption. This process is repeated for a total of 25 encryptions. The resulting 64-bit output is then translated into an 11-character sequence.



(a) Loading a new password



(b) Verifying a password

**Figure 15.1** UNIX Password Scheme.

The ciphertext password is then stored, together with a plaintext copy of the salt, in the password file for the corresponding user ID.

The salt serves three purposes:

- It prevents duplicate passwords from being visible in the password file. Even if two users choose the same password, those passwords will be assigned at different times. Hence, the “extended” passwords of the two users will differ.
- It effectively increases the length of the password without requiring the user to remember two additional characters. Hence, the number of possible passwords is increased by a factor of 4096, increasing the difficulty of guessing a password.
- It prevents the use of a hardware implementation of DES, which would ease the difficulty of a brute-force guessing attack.

When a user attempts to log on to a UNIX system, the user provides an ID and a password. The operating system uses the ID to index into the password file

and retrieve the plaintext salt and the encrypted password, which are used as input to the encryption routine. If the result matches the stored value, the password is accepted.

The encryption routine is designed to discourage guessing attacks. Software implementations of DES are slow compared to hardware versions, and the use of 25 iterations multiplies the time required by 25. However, since the original design of this algorithm, two changes have occurred. First, newer implementations of the algorithm itself have resulted in speedups. For example, the Internet worm was able to do on-line password guessing of a few hundred passwords in a reasonably short time by using a more efficient encryption algorithm than the standard one stored on the UNIX systems that it attacked. Second, hardware performance continues to increase, so that any software algorithm executes more quickly.

Thus, there are two threats to the UNIX password scheme. First, a user can gain access on a machine using a guest account or by some other means and then run a password guessing program, called a password cracker, on that machine. The attacker should be able to check hundreds and perhaps thousands of possible passwords with little resource consumption. In addition, if an opponent is able to obtain a copy of the password file, then a cracker program can be run on another machine at leisure. This enables the opponent to run through many thousands of possible passwords in a reasonable period.

As an example, a password cracker was reported on the Internet in August 1993 [MADS93]. Using a Thinking Machines Corporation parallel computer, a performance of 1560 encryptions per second per vector unit was achieved. With four vector units per processing node (a standard configuration), this works out to 800,000 encryptions per second on a 128-node machine (which is a modest size) and 6.4 million encryptions per second on a 1024-node machine.

Even these stupendous guessing rates do not yet make it feasible for an attacker to use a dumb brute-force technique of trying all possible combinations of characters to discover a password. Instead, password crackers rely on the fact that some people choose easily guessable passwords.

Some users, when permitted to choose their own password, pick one that is absurdly short. The results of one study at Purdue University are shown in Table 15.1. The study observed password change choices on 54 machines, representing approximately 7000 user accounts. Almost 3% of the passwords were three characters or fewer in length. An attacker could begin the attack by exhaustively testing

**Table 15.1** Observed Password Lengths [SPAF92a]

Length	Number	Fraction of Total
1	55	.004
2	87	.006
3	212	.02
4	449	.03
5	1260	.09
6	3035	.22
7	2917	.21
8	5772	.42
Total	13787	1.0

all possible passwords of length 3 or fewer. A simple remedy is for the system to reject any password choice of fewer than, say, six characters or even to require that all passwords be exactly eight characters in length. Most users would not complain about such a restriction.

Password length is only part of the problem. Many people, when permitted to choose their own password, pick a password that is guessable, such as their own name, their street name, a common dictionary word, and so forth. This makes the job of password cracking straightforward. The cracker simply has to test the password file against lists of likely passwords. Because many people use guessable passwords, such a strategy should succeed on virtually all systems.

One demonstration of the effectiveness of guessing is reported in [KLEI90]. From a variety of sources, the author collected UNIX password files, containing nearly 14,000 encrypted passwords. The result, which the author rightly characterizes as frightening, is shown in Table 15.2. In all, nearly one-fourth of the passwords were guessed. The following strategy was used:

1. Try the user's name, initials, account name, and other relevant personal information. In all, 130 different permutations for each user were tried.

**Table 15.2** Passwords Cracked from a Sample Set of 13,797 Accounts [KLEI90]

Type of Password	Search Size	Number of Matches	Percentage of Passwords Matched	Cost/Benefit Ratio <sup>a</sup>
User/account name	130	368	2.7%	2.830
Character sequences	866	22	0.2%	0.025
Numbers	427	9	0.1%	0.021
Chinese	392	56	0.4%	0.143
Place names	628	82	0.6%	0.131
Common names	2239	548	4.0%	0.245
Female names	4280	161	1.2%	0.038
Male names	2866	140	1.0%	0.049
Uncommon names	4955	130	0.9%	0.026
Myths & legends	1246	66	0.5%	0.053
Shakespearean	473	11	0.1%	0.023
Sports terms	238	32	0.2%	0.134
Science fiction	691	59	0.4%	0.085
Movies and actors	99	12	0.1%	0.121
Cartoons	92	9	0.1%	0.098
Famous people	290	55	0.4%	0.190
Phrases and patterns	933	253	1.8%	0.271
Surnames	33	9	0.1%	0.273
Biology	58	1	0.0%	0.017
System dictionary	19683	1027	7.4%	0.052
Machine names	9018	132	1.0%	0.015
Mnemonics	14	2	0.0%	0.143
King James Bible	7525	83	0.6%	0.011
Miscellaneous words	3212	54	0.4%	0.017
Yiddish words	56	0	0.0%	0.000
Asteroids	2407	19	0.1%	0.007
TOTAL	62727	3340	24.2%	0.053

<sup>a</sup>Computed as the number of matches divided by the search size. The more words that needed to be tested for a match, the lower the cost/benefit ratio.

2. Try words from various dictionaries. The author compiled a dictionary of over 60,000 words, including the on-line dictionary on the system itself, and various other lists as shown.
3. Try various permutations on the words from step 2. This included making the first letter uppercase or a control character, making the entire word uppercase, reversing the word, changing the letter "o" to the digit "zero," and so on. These permutations added another 1 million words to the list.
4. Try various capitalization permutations on the words from step 2 that were not considered in step 3. This added almost 2 million additional words to the list.

Thus, the test involved in the neighborhood of 3 million words. Using the fastest Thinking Machines implementation listed earlier, the time to encrypt all these words for all possible salt values is under an hour. Keep in mind that such a thorough search could produce a success rate of about 25%, whereas even a single hit may be enough to gain a wide range of privileges on a system.

### Access Control

One way to thwart a password attack is to deny the opponent access to the password file. If the encrypted password portion of the file is accessible only by a privileged user, then the opponent cannot read it without already knowing the password of a privileged user. [SPAF92a] points out several flaws in this strategy:

- Many systems, including most UNIX systems, are susceptible to unanticipated break-ins. Once an attacker has gained access by some means, he or she may wish to obtain a collection of passwords in order to use different accounts for different logon sessions to decrease the risk of detection. Or a user with an account may desire another user's account to access privileged data or to sabotage the system.
- An accident of protection might render the password file readable, thus compromising all the accounts.
- Some of the users have accounts on other machines in other protection domains, and they use the same password. Thus, if the passwords could be read by anyone on one machine, a machine in another location might be compromised.

Thus, a more effective strategy would be to force users to select passwords that are difficult to guess.

### Password Selection Strategies

The lesson from the two experiments just described (Tables 15.1 and 15.2) is that, left to their own devices, many users choose a password that is too short or too easy to guess. At the other extreme, if users are assigned passwords consisting of eight randomly selected printable characters, password cracking is effectively impossible. But it would be almost as impossible for most users to remember their passwords. Fortunately, even if we limit the password universe to strings of characters that are reasonably memorable, the size of the universe is still too large to permit practical

cracking. Our goal, then, is to eliminate guessable passwords while allowing the user to select a password that is memorable. Four basic techniques are in use:

- User education
- Computer-generated passwords
- Reactive password checking
- Proactive password checking

Users can be told the importance of using hard-to-guess passwords and can be provided with guidelines for selecting strong passwords. This **user education** strategy is unlikely to succeed at most installations, particularly where there is a large user population or a lot of turnover. Many users will simply ignore the guidelines. Others may not be good judges of what is a strong password. For example, many users (mistakenly) believe that reversing a word or capitalizing the last letter makes a password unguessable.

**Computer-generated passwords** also have problems. If the passwords are quite random in nature, users will not be able to remember them. Even if the password is pronounceable, the user may have difficulty remembering it and so be tempted to write it down. In general, computer-generated password schemes have a history of poor acceptance by users. FIPS PUB 181 defines one of the best-designed automated password generators. The standard includes not only a description of the approach but also a complete listing of the C source code of the algorithm. The algorithm generates words by forming pronounceable syllables and concatenating them to form a word. A random number generator produces a random stream of characters used to construct the syllables and words.

A **reactive password checking** strategy is one in which the system periodically runs its own password cracker to find guessable passwords. The system cancels any passwords that are guessed and notifies the user. This tactic has a number of drawbacks. First, it is resource intensive if the job is done right. Because a determined opponent who is able to steal a password file can devote full CPU time to the task for hours or even days, an effective reactive password checker is at a distinct disadvantage. Furthermore, any existing passwords remain vulnerable until the reactive password checker finds them.

The most promising approach to improved password security is a **proactive password checker**. In this scheme, a user is allowed to select his or her own password. However, at the time of selection, the system checks to see if the password is allowable and, if not, rejects it. Such checkers are based on the philosophy that, with sufficient guidance from the system, users can select memorable passwords from a fairly large password space that are not likely to be guessed in a dictionary attack.

The trick with a proactive password checker is to strike a balance between user acceptability and strength. If the system rejects too many passwords, users will complain that it is too hard to select a password. If the system uses some simple algorithm to define what is acceptable, this provides guidance to password crackers to refine their guessing technique. In the remainder of this subsection, we look at possible approaches to proactive password checking.

The first approach is a simple system for rule enforcement. For example, the following rules could be enforced:

- All passwords must be at least eight characters long.
- In the first eight characters, the passwords must include at least one each of uppercase, lowercase, numeric digits, and punctuation marks.

These rules could be coupled with advice to the user. Although this approach is superior to simply educating users, it may not be sufficient to thwart password crackers. This scheme alerts crackers as to which passwords *not* to try but may still make it possible to do password cracking.

Another possible procedure is simply to compile a large dictionary of possible “bad” passwords. When a user selects a password, the system checks to make sure that it is not on the disapproved list. There are two problems with this approach:

- **Space:** The dictionary must be very large to be effective. For example, the dictionary used in the Purdue study [SPAF92a] occupies more than 30 megabytes of storage.
- **Time:** The time required to search a large dictionary may itself be large. In addition, to check for likely permutations of dictionary words, either those words must be included in the dictionary, making it truly huge, or each search must also involve considerable processing.

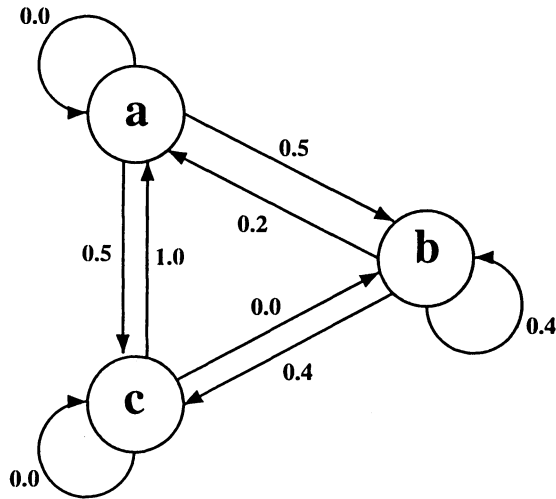
Two techniques for developing an effective and efficient proactive password checker that is based on rejecting words on a list show promise. One of these develops a Markov model for the generation of guessable passwords [DAVI93]. Figure 15.2 shows a simplified version of such a model. This model shows a language consisting of an alphabet of three characters. The state of the system at any time is the identity of the most recent letter. The value on the transition from one state to another represents the probability that one letter follows another. Thus, the probability that the next letter is b given that the current letter is a is 0.5.

In general, a Markov model is a quadruple  $[m, A, T, k]$ , where  $m$  is the number of states in the model,  $A$  is the state space,  $T$  is the matrix of transition probabilities, and  $k$  is the order of the model. For a  $k$ th-order model, the probability of making a transition to a particular letter depends on the previous  $k$  letters that have been generated. Figure 15.2 shows a simple first-order model.

The authors report on the development and use of a second-order model. To begin, a dictionary of guessable passwords is constructed. Then the transition matrix is calculated as follows:

1. Determine the frequency matrix  $f$ , where  $f(i, j, k)$  is the number of occurrences of the trigram consisting of the  $i$ th,  $j$ th, and  $k$ th character. For example, the password *parsnips* yields the trigrams *par*, *ars*, *rsn*, *sni*, *nip*, and *ips*.
2. For each bigram  $ij$ , calculate  $f(i, j, \infty)$  as the total number of trigrams beginning with  $ij$ . For example,  $f(a, b, \infty)$  would be the total number of trigrams of the form *aba*, *abb*, *abc*, and so on.
3. Compute the entries of  $T$  as follows:

$$T(i, j, k) = \frac{f(i, j, k)}{f(i, j, \infty)}$$



$M = \{3, \{a, b, c\}, T, 1\}$  where

$$T = \begin{bmatrix} 0.0 & 0.5 & 0.5 \\ 0.2 & 0.4 & 0.4 \\ 1.0 & 0.0 & 0.0 \end{bmatrix}$$

e.g., string probably from this language: abbcacaba

e.g., string probably not from this language: aaccbbaaa

**Figure 15.2** An Example Markov Model.

The result is a model that reflects the structure of the words in the dictionary. With this model, the question “Is this a bad password?” is transformed into the question “Was this string (password) generated by this Markov model?” For a given password, the transition probabilities of all its trigrams can be looked up. Some standard statistical tests can then be used to determine if the password is likely or unlikely for that model. Passwords that are likely to be generated by the model are rejected. The authors report good results for a second-order model. Their system catches virtually all the passwords in their dictionary and does not exclude so many potentially good passwords as to be user unfriendly.

A quite different approach has been reported by Spafford [SPAF92a, SPAF92b]. It is based on the use of a Bloom filter [BLOO70]. To begin, we explain the operation of the Bloom filter. A Bloom filter of order  $k$  consists of a set of  $k$  independent hash functions  $H_1(x), H_2(x), \dots, H_k(x)$ , where each function maps a password into a hash value in the range  $0$  to  $N - 1$ . That is,

$$H_i(X_j) = y \quad 1 \leq i \leq k; \quad 1 \leq j \leq D; \quad 0 \leq y \leq N - 1$$

where

$X_j$  =  $j$ th word in password dictionary  
 $D$  = number of words in password dictionary



The following procedure is then applied to the dictionary:

1. A hash table of  $N$  bits is defined, with all bits initially set to 0.
2. For each password, its  $k$  hash values are calculated, and the corresponding bits in the hash table are set to 1. Thus, if  $H_i(X_j) = 67$  for some  $(i, j)$ , then the sixty-seventh bit of the hash table is set to 1; if the bit already has the value 1, it remains at 1.

When a new password is presented to the checker, its  $k$  hash values are calculated. If all the corresponding bits of the hash table are equal to 1, then the password is rejected. All passwords in the dictionary will be rejected. But there will also be some "false positives" (that is, passwords that are not in the dictionary but that produce a match in the hash table). To see this, consider a scheme with two hash functions. Suppose that the passwords *undertaker* and *hulkhogan* are in the dictionary, but *xG%#jj98* is not. Further suppose that

$$\begin{array}{lll} H_1(\text{undertaker}) = 25 & H_1(\text{hulkhogan}) = 83 & H_1(\text{xG\%#jj98}) = 665 \\ H_2(\text{undertaker}) = 998 & H_2(\text{hulkhogan}) = 665 & H_2(\text{xG\%#jj98}) = 998 \end{array}$$

If the password *xG%#jj98* is presented to the system, it will be rejected even though it is not in the dictionary. If there are too many such false positives, it will be difficult for users to select passwords. Therefore, we would like to design the hash scheme to minimize false positives. It can be shown that the probability of a false positive can be approximated by

$$P \approx (1 - e^{-kD/N})^k \approx (1 - e^{-k/R})^k$$

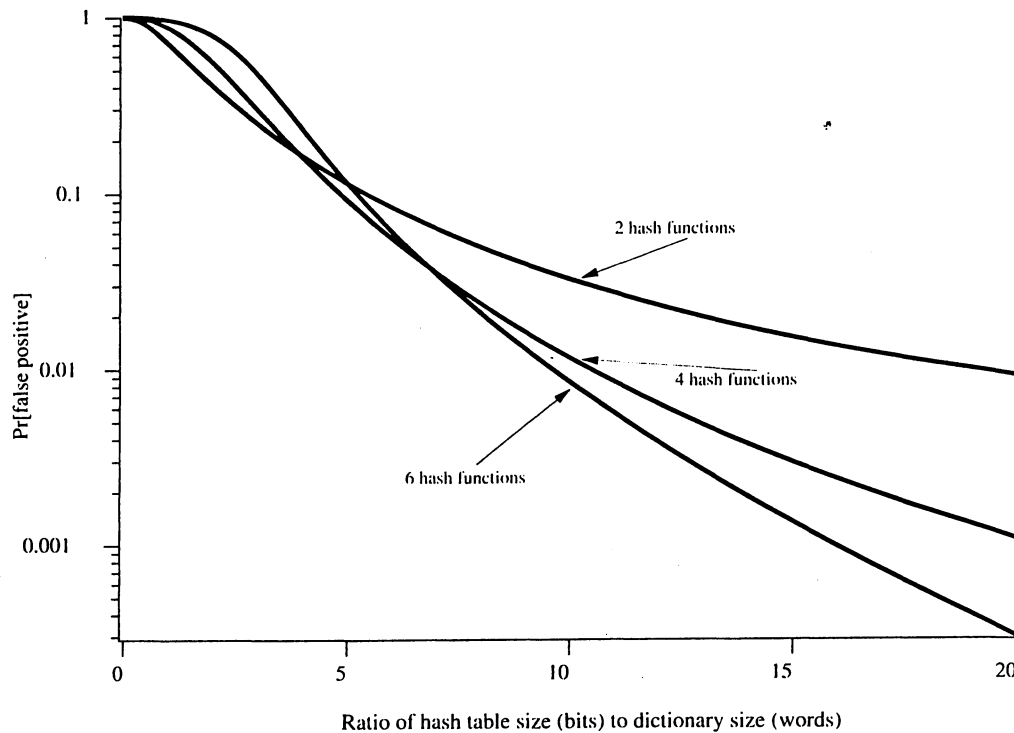
or, equivalently,

$$R \approx \frac{-k}{\ln(1 - P^{1/k})}$$

where

- $k$  = number of hash functions
- $N$  = number of bits in hash table
- $D$  = number of words in dictionary
- $R = N/D$ , ratio of hash table size (bits) to dictionary size (words)

Figure 15.3 plots  $P$  as a function of  $R$  for various values of  $k$ . Suppose we have a dictionary of 1 million words and we wish to have a 0.01 probability of rejecting a password not in the dictionary. If we choose six hash functions, the required ratio is  $R = 9.15$ . Therefore, we need a hash table of  $9.6 \times 10^6$  bits or about 1.2 megabytes of storage. In contrast, storage of the entire dictionary would require on the order of 8 megabytes. Thus, we achieve a compression of almost a factor of 7. Furthermore, password checking involves the straightforward calculation of six hash func-



**Figure 15.3** Performance of Bloom Filter.

tions and is independent of the size of the dictionary, whereas with the use of the full dictionary, there is substantial searching.<sup>1</sup>

### Intrusion Detection

Inevitably, the best intrusion prevention system will fail. A system's second line of defense is intrusion detection, and this has been the focus of much research in recent years. This interest is motivated by a number of considerations, including the following:

1. If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised. Even if the detection is not sufficiently timely to preempt the intruder, the sooner that the intrusion is detected, the less the amount of damage and the more quickly that recovery can be achieved.

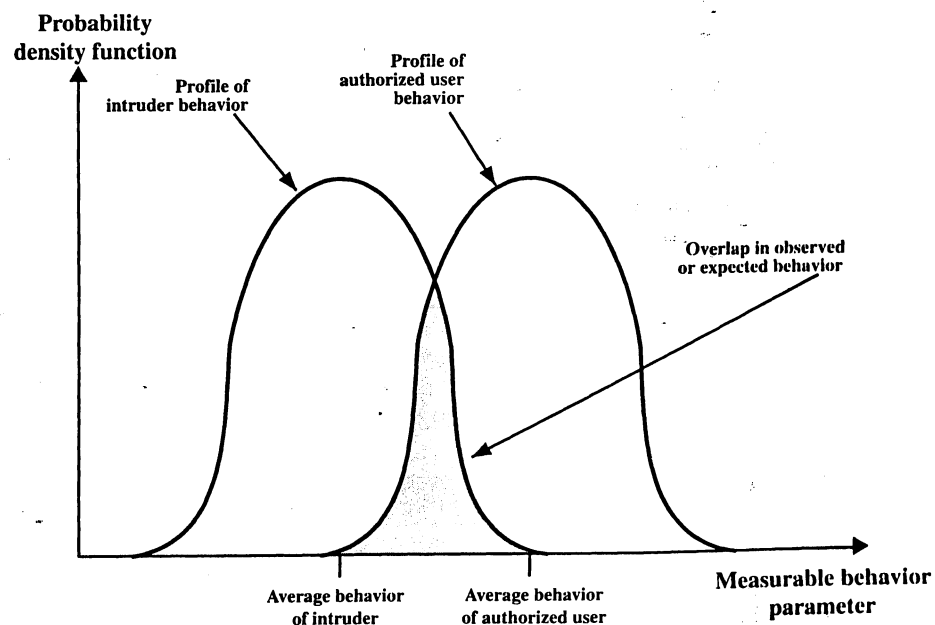
<sup>1</sup>Both the Markov model and the Bloom filter involve the use of probabilistic techniques. In the case of the Markov model, there is a small probability that some passwords in the dictionary will not be caught and a small probability that some passwords not in the dictionary will be rejected. In the case of the Bloom filter, there is a small probability that some passwords not in the dictionary will be rejected. Again we see that taking a probabilistic approach simplifies the solution (e.g., see footnote 1 in Chapter 12).

2. An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions.
3. Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

Intrusion detection is based on the assumption that the behavior of the intruder differs from that of a legitimate user in ways that can be quantified. Of course, we cannot expect that there will be a crisp, exact distinction between an attack by an intruder and the normal use of resources by an authorized user. Rather, we must expect that there will be some overlap.

Figure 15.4 suggests, in very abstract terms, the nature of the task confronting the designer of an intrusion detection system. Although the typical behavior of an intruder differs from the typical behavior of an authorized user, there is an overlap in these behaviors. Thus, a loose interpretation of intruder behavior, which will catch more intruders, will also lead to a number of "false positives," or authorized users identified as intruders. On the other hand, an attempt to limit false positives by a tight interpretation of intruder behavior will lead to an increase in false negatives, or intruders not identified as intruders. Thus, there is an element of compromise and art in the practice of intrusion detection.

In Anderson's study [ANDE80], it was postulated that one could, with reasonable confidence, distinguish between a masquerader and a legitimate user. Patterns of legitimate user behavior can be established by observing past history, and significant deviation from such patterns can be detected. Anderson suggests that the



**Figure 15.4** Profiles of Behavior of Intruders and Authorized Users.

task of detecting a misfeasor (legitimate user performing in an unauthorized fashion) is more difficult, in that the distinction between abnormal and normal behavior may be small. Anderson concluded that such violations would be undetectable solely through the search for anomalous behavior. However, misfeasor behavior might nevertheless be detectable by intelligent definition of the class of conditions that suggest unauthorized use. Finally, the detection of the clandestine user was felt to be beyond the scope of purely automated techniques. These observations, which were made in 1980, remain true today.

[PORR92] identifies the following approaches to intrusion detection:

1. **Statistical anomaly detection:** Involves the collection of data relating to the behavior of legitimate users over a period of time. Then statistical tests are applied to observed behavior to determine with a high level of confidence whether that behavior is not legitimate user behavior.
  - a. **Threshold detection:** This approach involves defining thresholds, independent of user, for the frequency of occurrence of various events.
  - b. **Profile based:** A profile of the activity of each user is developed and used to detect changes in the behavior of individual accounts.
2. **Rule-based detection:** Involves an attempt to define a set of rules that can be used to decide that a given behavior is that of an intruder.
  - a. **Anomaly detection:** Rules are developed to detect deviation from previous usage patterns.
  - b. **Penetration identification:** An expert system approach that searches for suspicious behavior.

In a nutshell, statistical approaches attempt to define normal, or expected, behavior, whereas rule-based approaches attempt to define proper behavior.

In terms of the types of attackers listed earlier, statistical anomaly detection is effective against masqueraders, who are unlikely to mimic the behavior patterns of the accounts they appropriate. On the other hand, such techniques may be unable to deal with misfeasors. For such attacks, rule-based approaches may be able to recognize events and sequences that, in context, reveal penetration. In practice, a system may exhibit a combination of both approaches to be effective against a broad range of attacks.

### **Audit Records**

A fundamental tool for intrusion detection is the audit record. Some record of ongoing activity must be maintained by users as input to an intrusion detection system. Basically, two plans are used:

- **Native audit records:** Virtually all multiuser operating systems include accounting software that collects information on user activity. The advantage of using this information is that no additional collection software is needed. The disadvantage is that the native audit records may not contain the needed information or may not contain it in a convenient form.

- **Detection-specific audit records:** A collection facility can be implemented that generates audit records containing only that information required by the intrusion detection system. One advantage of such an approach is that it could be made vendor independent and ported to a variety of systems. The disadvantage is the extra overhead involved in having, in effect, two accounting packages running on a machine.

A good example of detection-specific audit records is one developed by Dorothy Denning [DENN87]. Each audit record contains the following fields:

- **Subject:** Initiators of actions. A subject is typically a terminal user but might also be a process acting on behalf of users or groups of users. All activity arises through commands issued by subjects. Subjects may be grouped into different access classes, and these classes may overlap.
- **Action:** Operation performed by the subject on or with an object; for example, login, read, perform I/O, execute.
- **Object:** Receptors of actions. Examples include files, programs, messages, records, terminals, printers, and user- or program-created structures. When a subject is the recipient of an action, such as electronic mail, then that subject is considered an object. Objects may be grouped by type. Object granularity may vary by object type and by environment. For example, database actions may be audited for the database as a whole or at the record level.
- **Exception-Condition:** Denotes which, if any, exception condition is raised on return.
- **Resource-Usage:** A list of quantitative elements in which each element gives the amount used of some resource (e.g., number of lines printed or displayed, number of records read or written, processor time, I/O units used, session elapsed time).
- **Time-stamp:** Unique time-and-date stamp identifying when the action took place.

Most user operations are made up of a number of elementary actions. For example, a file copy involves the execution of the user command, which includes doing access validation and setting up the copy, plus the read from one file, plus the write to another file. Consider the command

COPY GAME.EXE TO <Library>GAME.EXE

issued by Smith to copy an executable file GAME from the current directory to the <Library> directory. The following audit records may be generated:

Smith	execute	<Library>COPY.EXE	0	CPU = 00002	11058721678
Smith	read	<Smith>GAME.EXE	0	RECORDS = 0	11058721679
Smith	execute	<Library>COPY.EXE	write-viol	RECORDS = 0	11058721680

In this case, the copy is aborted because Smith does not have write permission to <Library>.

The decomposition of a user operation into elementary actions has three advantages:

1. Because objects are the protectable entities in a system, the use of elementary actions enables an audit of all behavior affecting an object. Thus, the system can detect attempted subversions of access controls (by noting an abnormality in the number of exception conditions returned) and can detect successful subversions by noting an abnormality in the set of objects accessible to the subject.
2. Single-object, single-action audit records simplify the model and the implementation.
3. Because of the simple, uniform structure of the detection-specific audit records, it may be relatively easy to obtain this information or at least part of it by a straightforward mapping from existing native audit records to the detection-specific audit records.

### Statistical Anomaly Detection

As was mentioned, statistical anomaly detection techniques fall into two broad categories: threshold detection and profile-based systems. Threshold detection involves counting the number of occurrences of a specific event type over an interval of time. If the count surpasses what is considered a reasonable number that one might expect to occur, then intrusion is assumed.

Threshold analysis, by itself, is a crude and ineffective detector of even moderately sophisticated attacks. Both the threshold and the time interval must be determined. Because of the variability across users, such thresholds are likely to generate either a lot of false positives or a lot of false negatives. However, simple threshold detectors may be useful in conjunction with more sophisticated techniques.

Profile-based anomaly detection focuses on characterizing the past behavior of individual users or related groups of users and then detecting significant deviations. A profile may consist of a set of parameters, so that deviation on just a single parameter may not be sufficient in itself to signal an alert.

The foundation of this approach is an analysis of audit records. The audit records provide input to the intrusion detection function in two ways. First, the designer must decide on a number of quantitative metrics that can be used to measure user behavior. An analysis of audit records over a period of time can be used to determine the activity profile of the average user. Thus, the audit records serve to define typical behavior. Second, current audit records are the input used to detect intrusion. That is, the intrusion detection model analyzes incoming audit records to determine deviation from average behavior.

Examples of metrics that are useful for profile-based intrusion detection are the following:

- **Counter:** A nonnegative integer that may be incremented but not decremented until it is reset by management action. Typically, a count of certain event types is kept over a particular period of time. Examples include the number of logins by a single user during an hour, the number of times a given

command is executed during a single user session, and the number of password failures during a minute.

- **Gauge:** A nonnegative integer that may be incremented or decremented. Typically, a gauge is used to measure the current value of some entity. Examples include the number of logical connections assigned to a user application and the number of outgoing messages queued for a user process.
- **Interval timer:** The length of time between two related events. An example is the length of time between successive logins to an account.
- **Resource utilization:** Quantity of resources consumed during a specified period. Examples include the number of pages printed during a user session and total time consumed by a program execution.

Given these general metrics, various tests can be performed to determine whether current activity fits within acceptable limits. [DENN87] lists the following approaches that may be taken:

- Mean and standard deviation
- Multivariate
- Markov process
- Time series
- Operational

The simplest statistical test is to measure the **mean and standard deviation** of a parameter over some historical period. This gives a reflection of the average behavior and its variability. The use of mean and standard deviation is applicable to a wide variety of counters, timers, and resource measures. But these measures, by themselves, are typically too crude for intrusion detection purposes.

A **multivariate** model is based on correlations between two or more variables. Intruder behavior may be characterized with greater confidence by considering such correlations (for example, processor time and resource usage, or login frequency and session elapsed time).

A **Markov process** model is used to establish transition probabilities among various states. As an example, this model might be used to look at transitions between certain commands.

A **time series** model focuses on time intervals, looking for sequences of events that happen too rapidly or too slowly. A variety of statistical tests can be applied to characterize abnormal timing.

Finally, an **operational model** is based on a judgment of what is considered abnormal, rather than an automated analysis of past audit records. Typically, fixed limits are defined and intrusion is suspected for an observation that is outside the limits. This type of approach works best where intruder behavior can be deduced from certain types of activities. For example, a large number of login attempts over a short period suggests an attempted intrusion.

As an example of the use of these various metrics and models, Table 15.3 shows various measures considered or tested for the Stanford Research Institute (SRI) intrusion detection system (IDES) [DENN87, JAVI91, LUNT88].

**Table 15.3** Measures That May Be Used for Intrusion Detection

Measure	Model	Type of Intrusion Detected
<b>Login and Session Activity</b>		
Login frequency by day and time	Mean and standard deviation	Intruders may be likely to log in during off-hours.
Frequency of login at different locations	Mean and standard deviation	Intruders may log in from a location that a particular user rarely or never uses.
Time since last login	Operational	Break-in on a "dead" account.
Elapsed time per session	Mean and standard deviation	Significant deviations might indicate masquerader.
Quantity of output to location	Mean and standard deviation	Excessive amounts of data transmitted to remote locations could signify leakage of sensitive data.
Session resource utilization	Mean and standard deviation	Unusual processor or I/O levels could signal an intruder.
Password failures at login	Operational	Attempted break-in by password guessing.
Failures to login from specified terminals	Operational	Attempted break-in.
<b>Command or Program Execution Activity</b>		
Execution frequency	Mean and standard deviation	May detect intruders, who are likely to use different commands, or a successful penetration by a legitimate user, who has gained access to privileged commands.
Program resource utilization	Mean and standard deviation	An abnormal value might suggest injection of a virus or Trojan horse, which performs side-effects that increase I/O or processor utilization.
Execution denials	Operational model	May detect penetration attempt by individual user who seeks higher privileges.
<b>File Access Activity</b>		
Read, write, create, delete frequency	Mean and standard deviation	Abnormalities for read and write access for individual users may signify masquerading or browsing.
Records read, written	Mean and standard deviation	Abnormality could signify an attempt to obtain sensitive data by inference and aggregation.
Failure count for read, write, create, delete	Operational	May detect users who persistently attempt to access unauthorized files.
File resource exhaustion counter	Operational	



The main advantage of the use of statistical profiles is that a prior knowledge of security flaws is not required. The detector program learns what is “normal” behavior and then looks for deviations. The approach is not based on system-dependent characteristics and vulnerabilities. Thus, it should be readily portable among a variety of systems.

### **Rule-Based Intrusion Detection**

Rule-based techniques detect intrusion by observing events in the system and applying a set of rules that lead to a decision regarding whether a given pattern of activity is or is not suspicious. In very general terms, we can characterize all approaches as focusing on either anomaly detection or penetration identification, although there is some overlap in these approaches.

**Rule-based anomaly detection** is similar in terms of its approach and strengths to statistical anomaly detection. With the rule-based approach, historical audit records are analyzed to identify usage patterns and to generate automatically rules that describe those patterns. Rules may represent past behavior patterns of users, programs, privileges, time slots, terminals, and so on. Current behavior is then observed, and each transaction is matched against the set of rules to determine if it conforms to any historically observed pattern of behavior.

As with statistical anomaly detection, rule-based anomaly detection does not require a knowledge of security vulnerabilities within the system. Rather, the scheme is based on observing past behavior and, in effect, assuming that the future will be like the past. In order for this approach to be effective, a rather large database of rules will be needed. For example, a scheme described in [VACC89] contains anywhere from  $10^4$  to  $10^6$  rules.

**Rule-based penetration identification** takes a very different approach to intrusion detection, one based on expert system technology. The key feature of such systems is the use of rules for identifying known penetrations or penetrations that would exploit known weaknesses. Rules can also be defined that identify suspicious behavior, even when the behavior is within the bounds of established patterns of usage. Typically, the rules used in these systems are specific to the machine and operating system. Also, such rules are generated by “experts” rather than by means of an automated analysis of audit records. The normal procedure is to interview system administrators and security analysts to collect a suite of known penetration scenarios and key events that threaten the security of the target system.<sup>2</sup> Thus, the strength of the approach depends on the skill of those involved in setting up the rules.

A simple example of the type of rules that can be used is found in NIDX, an early system that used heuristic rules that can be used to assign degrees of suspicion to activities [BAUE88]. Example heuristics are the following:

1. Users should not read files in other users’ personal directories.
2. Users must not write other users’ files.
3. Users who log in after hours often access the same files they used earlier.

---

<sup>2</sup>Such interviews may even extend to reformed or unreformed crackers who will share their expertise for a fee [FREE93].

4. Users do not generally open disk devices directly but rely on higher-level operating-system utilities.
5. Users should not be logged in more than once to the same system.
6. Users do not make copies of system programs.

The penetration identification scheme used in IDES is representative of the strategy followed. Audit records are examined as they are generated, and they are matched against the rule base. If a match is found, then the user's *suspicion rating* is increased. If enough rules are matched, then the rating will pass a threshold that results in the reporting of an anomaly.

The IDES approach is based on an examination of audit records. A weakness of this plan is its lack of flexibility. For a given penetration scenario, there may be a number of alternative audit record sequences that could be produced, each varying from the others slightly or in subtle ways. It may be difficult to pin down all these variations in explicit rules. Another method is to develop a higher-level model independent of specific audit records. An example of this is a state transition model known as USTAT [ILGU93]. USTAT deals in general actions rather than the detailed specific actions recorded by the UNIX auditing mechanism. USTAT is implemented on a SunOS system that provides audit records on 239 events. Of these, only 28 are used by a preprocessor, which maps these onto 10 general actions (Table 15.4). Using just these actions and the parameters that are invoked with each action, a state transition diagram is developed that characterizes suspicious activity.

**Table 15.4** USTAT Actions versus SunOS Event Types

USTAT Action	SunOS Event Type
Read	open_r, open_rc, open_rtc, open_rwc, open_rwtc, open_rt, open_rw, open_rwt
Write	truncate, ftruncate, creat, open_rtc, open_rwc, open_rwtc, open_rt, open_rw, open_rwt, open_w, open_wt, open_wc, open_wct
Create	mkdir, creat, open_rc, open_rtc, open_rwc, open_rwtc, open_wc, open_wtc, mknod
Delete	rmdir, unlink
Execute	exec, execve
Exit	exit
Modify_Owner	chown, fchown
Modify_Perm	chmod, fchmod
Rename	rename
Hardlink	link

Because a number of different auditable events map into a smaller number of actions, the rule-creation process is simpler. Furthermore, the state transition diagram model is easily modified to accommodate newly learned intrusion behaviors.

### Distributed Intrusion Detection

Until recently, work on intrusion detection systems focused on single-system stand-alone facilities. The typical organization, however, needs to defend a distributed collection of hosts supported by a LAN or internetwork. Although it is possible to mount a defense by using stand-alone intrusion detection systems on each host, a more effective defense can be achieved by coordination and cooperation among intrusion detection systems across the network.

Porras points out the following major issues in the design of a distributed intrusion detection system [PORR92]:

- A distributed intrusion detection system may need to deal with different audit record formats. In a heterogeneous environment, different systems will employ different native audit collection systems and, if using intrusion detection, may employ different formats for security-related audit records.
- One or more nodes in the network will serve as collection and analysis points for the data from the systems on the network. Thus, either raw audit data or summary data must be transmitted across the network. Therefore, there is a requirement to assure the integrity and confidentiality of these data. Integrity is required to prevent an intruder from masking his or her activities by altering the transmitted audit information. Confidentiality is required because the transmitted audit information could be valuable.
- Either a centralized or decentralized architecture can be used. With a centralized architecture, there is a single central point of collection and analysis of all audit data. This eases the task of correlating incoming reports but creates a potential bottleneck and single point of failure. With a decentralized architecture, there are more than one analysis centers, but these must coordinate their activities and exchange information.

A good example of a distributed intrusion detection system is one developed at the University of California at Davis [HEBE92, SNAP91]. Figure 15.5 shows the overall architecture, which consists of three main components:

- **Host agent module:** An audit collection module operating as a background process on a monitored system. Its purpose is to collect data on security-related events on the host and transmit these to the central manager.
- **LAN monitor agent module:** Operates in the same fashion as a host agent module except that it analyzes LAN traffic and reports the results to the central manager.
- **Central manager module:** Receives reports from LAN monitor and host agents and processes and correlates these reports to detect intrusion.

The scheme is designed to be independent of any operating system or system auditing implementation. Figure 15.6 [SNAP91] shows the general approach that is

taken. The agent captures each audit record produced by the native audit collection system. A filter is applied that retains only those records that are of security interest. These records are then reformatted into a standardized format referred to as the host audit record (HAR). Next, a template-driven logic module analyzes the records for suspicious activity. At the lowest level, the agent scans for notable events that are of interest independent of any past events. Examples include failed file accesses, accessing system files, and changing a file's access control. At the next higher level, the agent looks for sequences of events, such as known attack patterns (signatures). Finally, the agent looks for anomalous behavior of an individual user based on a historical profile of that user, such as number of programs executed, number of files accessed, and the like.

When suspicious activity is detected, an alert is sent to the central manager. The central manager includes an expert system that can draw inferences from received data. The manager may also query individual systems for copies of HARs to correlate with those from other agents.

The LAN monitor agent also supplies information to the central manager. The LAN monitor agent audits host-host connections, services used, and volume of traffic. It searches for significant events, such as sudden changes in network load, the use of security-related services, and network activities such as *rlogin*.

The architecture depicted in Figures 15.5 and 15.6 is quite general and flexible. It offers a foundation for a machine-independent approach that can expand from stand-alone intrusion detection to a system that is able to correlate activity from a number of sites and networks to detect suspicious activity that would otherwise remain undetected.

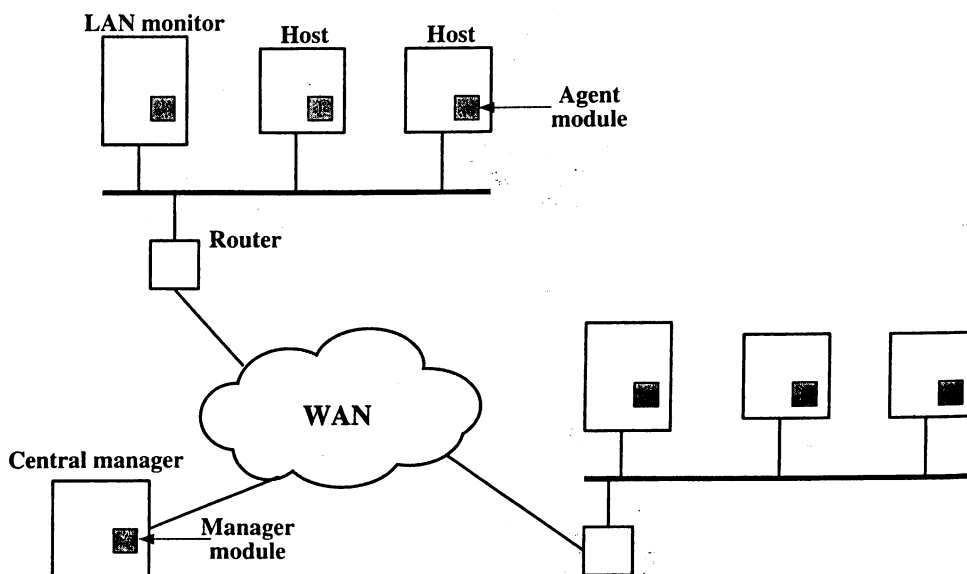


Figure 15.5 Architecture for Distributed Intrusion Detection.

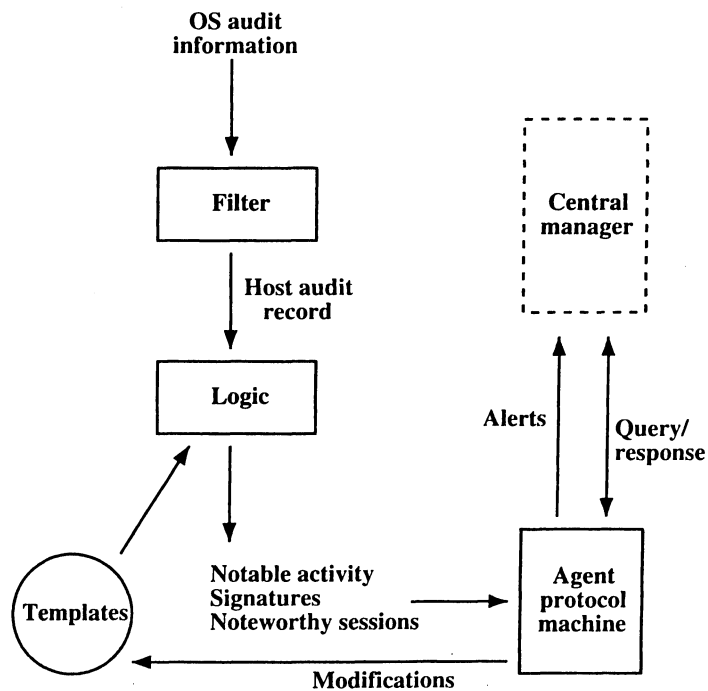


Figure 15.6 Agent Architecture.

## 15.2 VIRUSES AND RELATED THREATS

Perhaps the most sophisticated types of threats to computer systems are presented by programs that exploit vulnerabilities in computing systems. In this context, we are concerned with application programs as well as utility programs, such as editors and compilers.

We begin this section with an overview of the spectrum of such software threats. The remainder of the section is devoted to viruses; we look first at their nature and then at countermeasures.

### Malicious Programs

Figure 15.7 [BOWL92] provides an overall taxonomy of software threats, or malicious programs. These threats can be divided into two categories: those that need a host program, and those that are independent. The former are essentially fragments of programs that cannot exist independently of some actual application program, utility, or system program. The latter are self-contained programs that can be scheduled and run by the operating system.

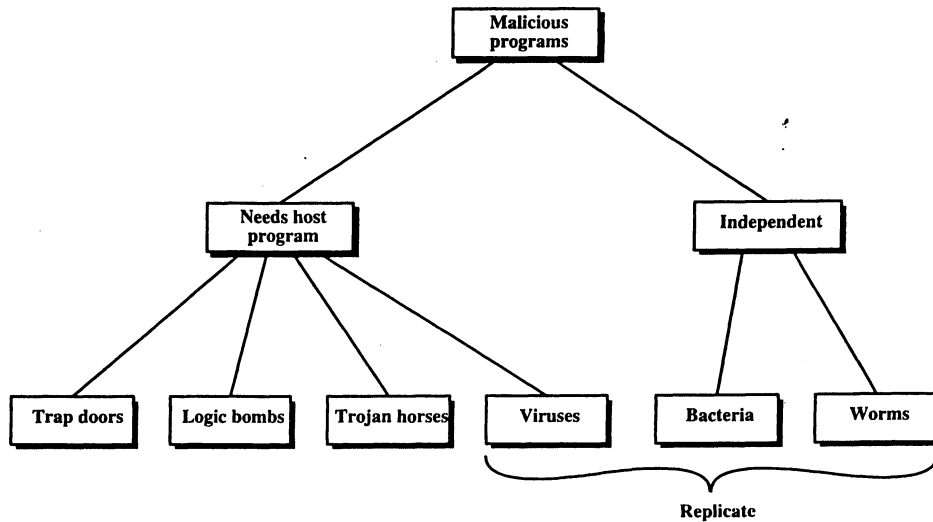


Figure 15.7 Taxonomy of Malicious Programs.

We can also differentiate between those software threats that do not replicate and those that do. The former are fragments of programs that are to be activated when the host program is invoked to perform a specific function. The latter consist of either a program fragment (virus) or an independent program (worm, bacterium) that, when executed, may produce one or more copies of itself to be activated later on the same system or some other system.

Although the taxonomy of Figure 15.7 is useful in organizing the information we are discussing, it is not the whole picture. In particular, logic bombs or Trojan horses may be part of a virus or worm.

### Trap doors

A trap door is a secret entry point into a program that allows someone that is aware of the trap door to gain access without going through the usual security access procedures. Trap doors have been used legitimately for many years by programmers to debug and test programs. This usually is done when the programmer is developing an application that has an authentication procedure, or a long setup, requiring the user to enter many different values to run the application. To debug the program, the developer may wish to gain special privileges or to avoid all the necessary setup and authentication. The programmer may also want to ensure that there is a method of activating the program should something be wrong with the authentication procedure that is being built into the application. The trap door is code that recognizes some special sequence of input or is triggered by being run from a certain user ID or by an unlikely sequence of events.

Trap doors become threats when they are used by unscrupulous programmers to gain unauthorized access. The trap door was the basic idea for the vulnerability portrayed in the movie *War Games* [COOP89]. Another example is that during the development of Multics, penetration tests were conducted by an Air Force “tiger

team" (simulating adversaries). One tactic employed was to send a bogus operating system update to a site running Multics. The update contained a Trojan horse (described later) that could be activated by a trap door and that allowed the tiger team to gain access. The threat was so well implemented that the Multics developers could not find it, even after they were informed of its presence [ENGE80].

It is difficult to implement operating system controls for trap doors. Security measures must focus on the program development and software update activities.

### **Logic Bomb**

One of the oldest types of program threat, predating viruses and worms, is the logic bomb. The logic bomb is code embedded in some legitimate program that is set to "explode" when certain conditions are met. Examples of conditions that can be used as triggers for a logic bomb are the presence or absence of certain files, a particular day of the week or date, or a particular user running the application. In one famous case [SPAF89], a logic bomb checked for a certain employee ID number (that of the bomb's author) and then triggered if the ID failed to appear in two consecutive payroll calculations. Once triggered, a bomb may alter or delete data or entire files, cause a machine halt, or do some other damage. A striking example of how logic bombs can be employed was the case of the Montgomery County, Maryland, library system [TIME90]. The contractor who had developed a computerized circulation system inserted a logic bomb that would disable the system on a certain date unless the contractor had been paid. When the library withheld its final payment because the system had poor response time, the contractor revealed the existence of the bomb and threatened to allow it to go off unless payment was forthcoming.

### **Trojan Horses**

A Trojan horse is a useful, or apparently useful, program or command procedure containing hidden code that, when invoked, performs some unwanted or harmful function.

Trojan horse programs can be used to accomplish functions indirectly that an unauthorized user could not accomplish directly. For example, to gain access to the files of another user on a shared system, a user could create a Trojan horse program that, when executed, changed the invoking user's file permissions so that the files are readable by any user. The author could then induce users to run the program by placing it in a common directory and naming it such that it appears to be a useful utility. An example is a program that ostensibly produces a listing of the user's files in a desirable format. After another user has run the program, the author can then access the information in the user's files. An example of a Trojan horse program that would be difficult to detect is a compiler that has been modified to insert additional code into certain programs as they are compiled, such as a system login program [THOM84]. The code creates a trap door in the login program that permits the author to log on to the system using a special password. This Trojan horse can never be discovered by reading the source code of the login program.

Another common motivation for the Trojan horse is data destruction. The program appears to be performing a useful function (e.g., a calculator program), but it may also be quietly deleting the user's files. For example, a CBS executive was

victimized by a Trojan horse that destroyed all information contained in his computer's memory [TIME90]. The Trojan horse was implanted in a graphics routine offered on an electronic bulletin board system.

### Viruses

A virus is a program that can "infect" other programs by modifying them; the modification includes a copy of the virus program, which can then go on to infect other programs.

Biological viruses are tiny scraps of genetic code—DNA or RNA—that can take over the machinery of a living cell and trick it into making thousands of flawless replicas of the original virus. Like its biological counterpart, a computer virus carries in its instructional code the recipe for making perfect copies of itself. Lodged in a host computer, the typical virus takes temporary control of the computer's disk operating system. Then, whenever the infected computer comes into contact with an uninfected piece of software, a fresh copy of the virus passes into the new program. Thus, the infection can be spread from computer to computer by unsuspecting users who either swap disks or send programs to one another over a network. In a network environment, the ability to access applications and system services on other computers provides a perfect culture for the spread of a virus.

Viruses are examined in greater detail later in this section.

### Worms

Network worm programs use network connections to spread from system to system. Once active within a system, a network worm can behave as a computer virus or bacteria or it could implant Trojan horse programs or perform any number of disruptive or destructive actions.

To replicate itself, a network worm uses some sort of network vehicle. Examples include the following:

- **Electronic mail facility:** A worm mails a copy of itself to other systems.
- **Remote execution capability:** A worm executes a copy of itself on another system.
- **Remote login capability:** A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other.

The new copy of the worm program is then run on the remote system where, in addition to any functions that it performs at that system, it continues to spread in the same fashion.

A network worm exhibits the same characteristics as a computer virus: a dormant phase, a propagation phase, a triggering phase, and an execution phase. The propagation phase generally performs the following functions:

1. Search for other systems to infect by examining host tables or similar repositories of remote system addresses.
2. Establish a connection with a remote system.
3. Copy itself to the remote system and cause the copy to be run.



The network worm may also attempt to determine whether a system has previously been infected before copying itself to the system. In a multiprogramming system, it may also disguise its presence by naming itself as a system process or using some other name that may not be noticed by a system operator.

As with viruses, network worms are difficult to counter. However, both network security and single-system security measures, if properly designed and implemented, minimize the threat of worms.

### **Bacteria**

Bacteria are programs that do not explicitly damage any files. Their sole purpose is to replicate themselves. A typical bacteria program may do nothing more than execute two copies of itself simultaneously on a multiprogramming system, or perhaps create two new files, each of which is a copy of the original source file of the bacteria program. Both of those programs then may copy themselves twice, and so on. Bacteria reproduce exponentially, eventually taking up all the processor capacity, memory, or disk space, denying users access to those resources.

### **The Nature of Viruses**

A virus can do anything that other programs do. The only difference is that it attaches itself to another program and executes secretly when the host program is run. Once a virus is executing, it can perform any function, such as erasing files and programs.

During its lifetime, a typical virus goes through the following four stages:

- **Dormant phase:** The virus is idle. The virus will eventually be activated by some event, such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.
- **Propagation phase:** The virus places an identical copy of itself into other programs or into certain system areas on the disk. Each infected program will now contain a clone of the virus, which will itself enter a propagation phase.
- **Triggering phase:** The virus is activated to perform the function for which it was intended. As with the dormant phase, the triggering phase can be caused by a variety of system events, including a count of the number of times that this copy of the virus has made copies of itself.
- **Execution phase:** The function is performed. The function may be harmless, such as a message on the screen, or damaging, such as the destruction of programs and data files.

Most viruses carry out their work in a manner that is specific to a particular operating system and, in some cases, specific to a particular hardware platform. Thus, they are designed to take advantage of the details and weaknesses of particular systems.

### **Virus Structure**

A virus can be prepended or postpended to an executable program, or it can be embedded in some other fashion. The key to its operation is that the infected program, when invoked, will first execute the virus code and then execute the original code of the program.

```

program V :=

{goto main;
 1234567;

  subroutine infect-executable :=
    {loop:
      file := get-random-executable-file;
      if (first-line-of-file = 1234567)
        then goto loop
        else prepend V to file;}

  subroutine do-damage :=
    {whatever damage is to be done}

  subroutine trigger-pulled :=
    {return true if some condition holds}

main:  main-program :=
      {infect-executable;
      if trigger-pulled then do-damage;
      goto next;}

next:

... }

```

**Figure 15.8** A Simple Virus.

A very general depiction of virus structure is shown in Figure 15.8 (based on [COHE94]). In this case, the virus code, V, is prepended to infected programs, and it is assumed that the entry point to the program, when invoked, is the first line of the program.

An infected program begins with the virus code and works as follows: The first line of code is a jump to the main virus program. The second line is a special marker that is used by the virus to determine whether or not a potential victim program has already been infected with this virus. When the program is invoked, control is immediately transferred to the main virus program. The virus program first seeks out uninfected executable files and infects them. Next, the virus may perform some action, usually detrimental to the system. This action could be performed every time the program is invoked, or it could be a logic bomb that triggers only under certain conditions. Finally, the virus transfers control to the original program. If the infection phase of the program is reasonably rapid, a user is unlikely to notice any difference between the execution of an infected and uninfected program.

A virus such as the one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file so that both the infected and uninfected versions are of identical length. Figure 15.9 [COHE94] shows in general terms the logic required. The key lines in this virus are numbered, and Figure 15.10 [COHE94] illustrates the operation. We assume that

```

program CV :=
[goto main;
 01234567;

  subroutine infect-executable :=
    {loop:
      file := get-random-executable-file;
      if (first-line-of-file = 01234567) then goto loop;
      (1) compress file;
      (2) prepend CV to file;
    }

main: main-program :=
  {if ask-permission then infect-executable;
   (3) uncompress rest-of-file;
   (4) run uncompressed file;}
}

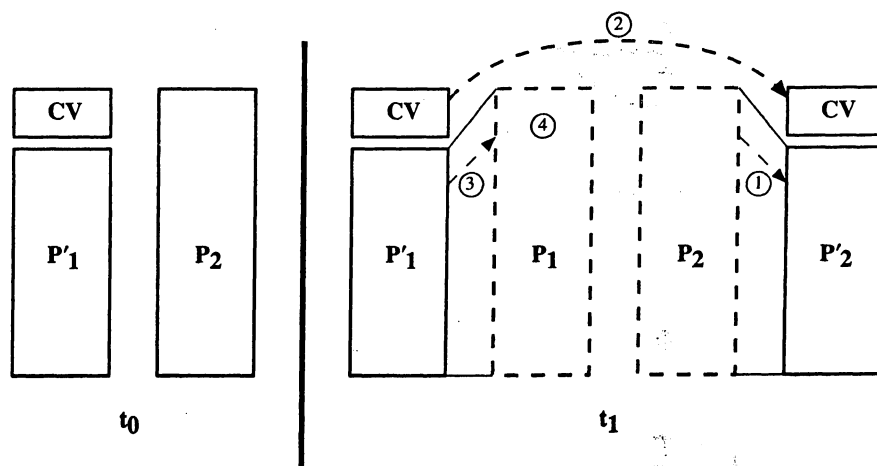
```

**Figure 15.9** Logic for a Compression Virus.

program  $P_1$  is infected with the virus CV. When this program is invoked, control passes to its virus, which performs the following steps:

1. For each uninfected file  $P_2$  that is found, the virus first compresses that file to produce  $P'_2$ , which is shorter than the original program by the size of the virus.
2. A copy of the virus is prepended to the compressed program.
3. The compressed version of the original infected program,  $P'_1$ , is uncompressed.
4. The uncompressed original program is executed.

In this example, the virus does nothing other than propagate. As in the previous example, the virus may include a logic bomb.



**Figure 15.10** A Compression Virus.

### Initial Infection

Once a virus has gained entry to a system by infecting a single program, it is in a position to infect some or all other executable files on that system when the infected program executes. Thus, viral infection can be completely prevented by preventing the virus from gaining entry in the first place. Unfortunately, prevention is extraordinarily difficult because a virus can be part of any program outside a system. Thus, unless one is content to take an absolutely bare piece of iron and write all one's own system and application programs, one is vulnerable.

Most viral infections initiate with a disk from which programs are copied onto a machine. Many of these are disks that have games or simple but handy utilities that employees obtain for their home computers and then bring in and put on an office machine. Some, incredibly, are present on disks that come shrinkwrapped from the manufacturer of an application. Only a small fraction of infections begin across a network connection. Most of these are obtained from an electronic bulletin board system. Again, typically, an employee will download a game or apparently useful utility only to discover later that it contains a virus.

### Types of Viruses

There has been a continuous arms race between virus writers and writers of antivirus software since viruses first appeared. As effective countermeasures have been developed for existing types of viruses, new types have been developed. [STEP93] suggests the following categories as being among the most significant types of viruses:

- **Parasitic virus:** The traditional and still most common form of virus. A parasitic virus attaches itself to executable files and replicates, when the infected program is executed, by finding other executable files to infect.
- **Memory-resident virus:** Lodges in main memory as part of a resident system program. From that point on, the virus infects every program that executes.
- **Boot sector virus:** Infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus.
- **Stealth virus:** A form of virus explicitly designed to hide itself from detection by antivirus software.
- **Polymorphic virus:** A virus that mutates with every infection, making detection by the "signature" of the virus impossible.

One example of a **stealth virus** was discussed earlier: a virus that uses compression so that the infected program is exactly the same length as an uninfected version. Far more sophisticated techniques are possible. For example, a virus can place intercept logic in disk I/O routines, so that when there is an attempt to read suspected portions of the disk using these routines, the virus will present back the original, uninfected program. Thus, *stealth* is not a term that applies to a virus as such but, rather, is a technique used by a virus to evade detection.

A **polymorphic virus** creates copies during replication that are functionally equivalent but have distinctly different bit patterns. As with a stealth virus, the

purpose is to defeat programs that scan for viruses. In this case, the “signature” of the virus will vary with each copy. To achieve this variation, the virus may randomly insert superfluous instructions or interchange the order of independent instructions. A more effective approach is to use encryption. A portion of the virus, generally called a *mutation engine*, creates a random encryption key to encrypt the remainder of the virus. The key is stored with the virus, and the mutation engine itself is altered. When an infected program is invoked, the virus uses the stored random key to decrypt the virus. When the virus replicates, a different random key is selected.

Another weapon in the virus writers’ armory is the virus-creation toolkit. Such a toolkit enables a relative novice to create quickly a number of different viruses. Although viruses created with toolkits tend to be less sophisticated than viruses designed from scratch, the sheer number of new viruses that can be generated creates a problem for antivirus schemes.

Yet another tool of the virus writer is the virus exchange bulletin board. A number of such boards have sprung up [ADAM92] in the United States and other countries. These boards offer copies of viruses that can be downloaded, as well as tips for the creation of viruses.

### Macro Viruses

In recent years, the number of viruses encountered at corporate sites has risen dramatically [BERG97]. Virtually all of this increase is due to the proliferation of one of the newest types of virus: the macro virus. According to the National Computer Security Agency ([www.ncsa.com](http://www.ncsa.com)), macro viruses now make up two-thirds of all computer viruses.

Macro viruses are particularly threatening for a number of reasons:

1. A macro virus is platform independent. Virtually all of the macro viruses infect Microsoft Word documents. Any hardware platform and operating system that supports Word can be infected.
2. Macro viruses infect documents, not executable portions of code. Most of the information introduced onto a computer system is in the form of a document rather than a program.
3. Macro viruses are easily spread. A very common method is by electronic mail.

Macro viruses take advantage of a feature found in Word and other office applications such as Microsoft Excel (namely, the macro). In essence, a macro is an executable program embedded in a word processing document or other type of file. Typically, users employ macros to automate repetitive tasks and thereby save keystrokes. The macro language is usually some form of the Basic programming language. A user might define a sequence of keystrokes in a macro and set it up so that the macro is invoked when a function key or special short combination of keys is input.

What makes it possible to create a macro virus is the autoexecuting macro. This is a macro that is automatically invoked, without explicit user input. Common autoexecute events are opening a file, closing a file, and starting an application. Once a macro is running, it can copy itself to other documents, delete files, and

cause other sorts of damage to the user's system. In Microsoft Word, there are three types of autoexecuting macros:

- **Autoexecute:** If a macro named AutoExec is in the "normal.dot" template or in a global template stored in Word's startup directory, it is executed whenever Word is started.
- **Automacro:** An automacro executes when a defined event occurs, such as opening or closing a document, creating a new document, or quitting Word.
- **Command macro:** If a macro in a global macro file or a macro attached to a document has the name of an existing Word command, it is executed whenever the user invokes that command (e.g., File Save).

A common technique for spreading a macro virus is as follows: An automacro or command macro is attached to a Word document that is introduced into a system by e-mail or disk transfer. At some point after the document is opened, the macro executes. The macro copies itself to the global macro file. When the next session of Word opens, the infected global macro is active. When this macro executes, it can replicate itself and cause damage.

Successive releases of Word provide increased protection against macro viruses. For example, Microsoft offers an optional Macro Virus Protection tool that detects suspicious Word files and alerts the customer to the potential risk of opening a file with macros. Various antivirus product vendors have also developed tools to detect and correct macro viruses. As in other types of viruses, the arms race continues in the field of macro viruses.

### Antivirus Approaches

The ideal solution to the threat of viruses is prevention: Do not allow a virus to get into the system in the first place. This goal is, in general, impossible to achieve, although prevention can reduce the number of successful viral attacks. The next best approach is to be able to do the following:

- **Detection:** Once the infection has occurred, determine that it has occurred and locate the virus.
- **Identification:** Once detection has been achieved, identify the specific virus that has infected a program.
- **Removal:** Once the specific virus has been identified, remove all traces of the virus from the infected program and restore it to its original state. Remove the virus from all infected systems so that the disease cannot spread further.

If detection succeeds but either identification or removal is not possible, then the alternative is to discard the infected program and reload a clean backup version.

Advances in virus and antivirus technology go hand in hand. Early viruses were relatively simple code fragments and could be identified and purged with relatively simple antivirus software packages. As the virus arms race has evolved, both viruses and, necessarily, antivirus software have grown more complex and sophisticated.

[STEP93] identifies four generations of antivirus software:

- First generation: simple scanners
- Second generation: heuristic scanners
- Third generation: activity traps
- Fourth generation: full-featured protection

A **first-generation** scanner requires a virus signature to identify a virus. The virus may contain “wildcards” but has essentially the same structure and bit pattern in all copies. Such signature-specific scanners are limited to the detection of known viruses. Another type of first-generation scanner maintains a record of the length of programs and looks for changes in length.

A **second-generation** scanner does not rely on a specific signature. Rather, the scanner uses heuristic rules to search for probable virus infection. One class of such scanners looks for fragments of code that are often associated with viruses. For example, a scanner may look for the beginning of an encryption loop used in a polymorphic virus and discover the encryption key. Once the key is discovered, the scanner can decrypt the virus to identify it, then remove the infection and return the program to service.

Another second-generation approach is integrity checking. A checksum can be appended to each program. If a virus infects the program without changing the checksum, then an integrity check will catch the change. To counter a virus that is sophisticated enough to change the checksum when it infects a program, an encrypted hash function can be used. The encryption key is stored separately from the program so that the virus cannot generate a new hash code and encrypt that. By using a hash function rather than a simpler checksum, the virus is prevented from adjusting the program to produce the same hash code as before.

**Third-generation** programs are memory-resident programs that identify a virus by its actions rather than its structure in an infected program. Such programs have the advantage that it is not necessary to develop signatures and heuristics for a wide array of viruses. Rather, it is necessary only to identify the small set of actions that indicate an infection is being attempted and then to intervene.

**Fourth-generation** products are packages consisting of a variety of antivirus techniques used in conjunction. These include scanning and activity trap components. In addition, such a package includes access control capability, which limits the ability of viruses to penetrate a system and then limits the ability of a virus to update files in order to pass on the infection.

The arms race continues. With fourth-generation packages, a more comprehensive defense strategy is employed, broadening the scope of defense to more general-purpose computer security measures.

### Advanced Antivirus Techniques

More sophisticated antivirus approaches and products continue to appear. In this subsection, we highlight two of the most important.

#### Generic Decryption

Generic decryption (GD) technology enables the antivirus program to detect easily even the most complex polymorphic viruses, while maintaining fast scanning

speeds [NACH97]. Recall that when a file containing a polymorphic virus is executed, the virus must decrypt itself to activate. In order to detect such a structure, executable files are run through a GD scanner, which contains the following elements:

- **CPU emulator:** A software-based virtual computer. Instructions in an executable file are interpreted by the emulator rather than executed on the underlying processor. The emulator includes software versions of all registers and other processor hardware, so that the underlying processor is unaffected by programs interpreted on the emulator.
- **Virus signature scanner:** A module that scans the target code looking for known virus signatures.
- **Emulation control module:** Controls the execution of the target code.

At the start of each simulation, the emulator begins interpreting instructions in the target code, one at a time. Thus, if the code includes a decryption routine that decrypts and hence exposes the virus, that code is interpreted. In effect, the virus does the work for the antivirus program by exposing the virus. Periodically, the control module interrupts interpretation to scan the target code for virus signatures.

During interpretation, the target code can cause no damage to the actual personal computer environment, because it is being interpreted in a completely controlled environment.

The most difficult design issue with a GD scanner is to determine how long to run each interpretation. Typically, virus elements are activated soon after a program begins executing, but this need not be the case. The longer the scanner emulates a particular program, the more likely it is to catch any hidden viruses. However, the antivirus program can take up only a limited amount of time and resources before users complain.

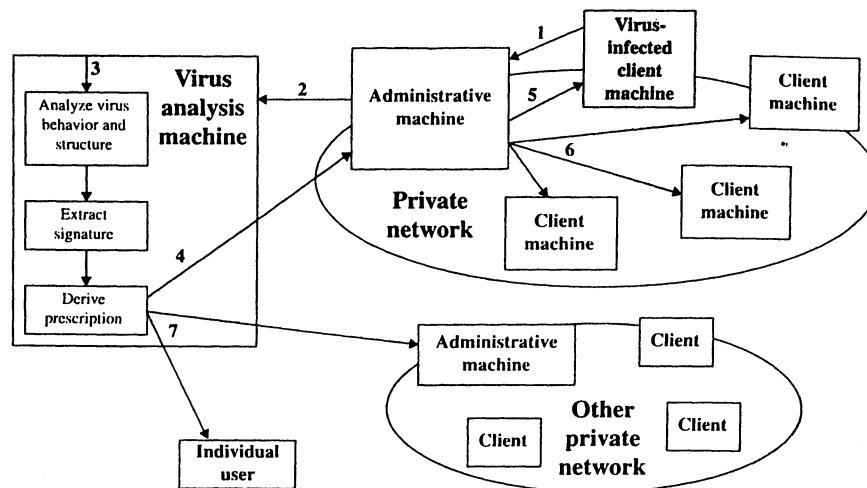
### Digital Immune System

The digital immune system is a comprehensive approach to virus protection developed by IBM [KEPH97a, KEPH97b]. The motivation for this development has been the rising threat of Internet-based virus propagation. We first say a few words about this threat and then summarize IBM's approach.

Currently, the virus threat is characterized by the relatively slow spread of new viruses and new mutations. Antivirus software is typically updated on a monthly basis, and this has been sufficient to control the problem. Also currently, the Internet plays a comparatively small role in the spread of viruses. But as [CHES97] points out, two major trends in Internet technology will have an increasing impact on the rate of virus propagation in coming years:

- **Integrated mail systems:** Systems such as Lotus Notes and Microsoft Outlook make it very simple to send anything to anyone and to work with objects that are received.
- **Mobile-program systems:** Capabilities such as Java and ActiveX allow programs to move on their own from one system to another.





**Figure 15.11** Digital Immune System.

In response to the threat posed by these Internet-based capabilities, IBM has developed a prototype digital immune system. This system expands on the use of program emulation discussed in the preceding subsection and provides a general-purpose emulation and virus-detection system. The objective of this system is to provide rapid response time so that viruses can be stamped out almost as soon as they are introduced. When a new virus enters an organization, the immune system automatically captures it, analyzes it, adds detection and shielding for it, removes it, and passes information about that virus to systems running IBM AntiVirus so that it can be detected before it is allowed to run elsewhere.

Figure 15.11 illustrates the typical steps in digital immune system operation:

1. A monitoring program on each PC uses a variety of heuristics based on system behavior, suspicious changes to programs, or family signature to infer that a virus may be present. The monitoring program forwards a copy of any program thought to be infected to an administrative machine within the organization.
2. The administrative machine encrypts the sample and sends it to a central virus analysis machine.
3. This machine creates an environment in which the infected program can be safely run for analysis. Techniques used for this purpose include emulation, or the creation of a protected environment within which the suspect program can be executed and monitored. The virus analysis machine then produces a prescription for identifying and removing the virus.
4. The resulting prescription is sent back to the administrative machine.
5. The administrative machine forwards the prescription to the infected client.
6. The prescription is also forwarded to other clients in the organization.
7. Subscribers around the world receive regular antivirus updates that protect them from the new virus.

The success of the digital immune system depends on the ability of the virus analysis machine to detect new and innovative virus strains. By constantly analyzing and monitoring the viruses found in the wild, it should be possible continually to update the digital immune software to keep up with the threat.

### 15.3 RECOMMENDED READING

A highly readable account of the most famous intruder incident is [STOL89]. [STER92] provides a useful history of intrusion and discusses the techniques and objectives of the main players: intruders, law enforcement, and civil libertarians. [DENN90] contains reprints of many key papers dealing with intruders.

[HOFF90] and [DENN90] contain reprints of many key papers dealing with viruses and worms. [COHE94] provides an excellent technical account of virus and antivirus technology. Good overview articles on viruses are [FORR97], [KEPH97], and [NACH97].

COHE94 Cohen, F. *A Short Course on Computer Viruses*. New York: Wiley, 1994.

DENN90 Denning, P., editor. *Computers Under Attack: Intruders, Worms, and Viruses*. Reading, MA: Addison-Wesley, 1990.

FORR97 Forrest, S.; Hofmeyr, S.; and Somayaji, A. "Computer Immunology." *Communications of the ACM*, October 1997.

HOFF90 Hoffman, L., editor. *Rogue Programs: Viruses, Worms, and Trojan Horses*. New York: Van Nostrand Reinhold, 1990.

KEPH97 Kephart, J.; Sorkin, G.; Chess, D.; and White, S. "Fighting Computer Viruses." *Scientific American*, November 1997.

NACH97 Nachenberg, C. "Computer Virus-Antivirus Coevolution." *Communications of the ACM*, January 1997.

STER92 Sterling, B. *The Hacker Crackdown: Law and Disorder on the Electronic Frontier*. New York: Bantam, 1992.

STOL89 Stoll, C. *The Cuckoo's Egg*. New York: Doubleday, 1989.

Recommended Web sites:

- **CERT Coordination Center:** The organization that grew from the computer emergency response team formed by the Defense Advanced Research Projects Agency. Site provides good information on Internet security threats, vulnerabilities, and attack statistics.
- **AntiVirus Online:** IBM's site on virus information; one of the best.

### 15.4 PROBLEMS

- 15.1 Assume that passwords are selected from four-character combinations of 26 alphabetic characters. Assume that an adversary is able to attempt passwords at a rate of one per second.
  - a. Assuming no feedback to the adversary until each attempt has been completed, what is the expected time to discover the correct password?
  - b. Assuming feedback to the adversary flagging an error as each incorrect character is entered, what is the expected time to discover the correct password?
- 15.2 Assume that source elements of length  $k$  is mapped in some uniform fashion into a target elements of length  $p$ . If each digit can take on one of  $r$  values, then the number of source elements is  $r^k$  and the number of target elements is the smaller number  $r^p$ . A particular source element  $x_i$  is mapped to a particular target element  $y_j$ .

- a. What is the probability that the correct source element can be selected by an adversary on one try?
  - b. What is the probability that a different source element  $x_k$  ( $x_i \neq x_k$ ) that results in the same target element,  $y_i$ , could be produced by an adversary?
  - c. What is the probability that the correct target element can be produced by an adversary on one try?
- 15.3** A phonetic password generator picks two segments randomly for each six-letter password. The form of each segment is CVC (consonant, vowel, consonant), where  $V = \langle a, e, i, o, u \rangle$  and  $C = \bar{V}$ .
- a. What is the total password population?
  - b. What is the probability of an adversary guessing a password correctly?
- 15.4** Assume that passwords are limited to the use of the 95 printable ASCII characters and that all passwords are 10 characters in length. Assume a password cracker with an encryption rate of 6.4 million encryptions per second. How long will it take to test exhaustively all possible passwords on a UNIX system?
- 15.5** Because of the known risks of the UNIX password system, the SunOS-4.0 documentation recommends that the password file be removed and replaced with a publicly readable file called /etc/publickey. An entry in the file for user A consists of a user's identifier  $ID_A$ , the user's public key,  $KU_A$ , and the corresponding private key  $KR_A$ . This private key is encrypted using DES with a key derived from the user's login password  $P_A$ . When A logs in the system decrypts  $E_{P_A}[KR_A]$  to obtain  $KR_A$ .
- a. The system then verifies that  $P_A$  was correctly supplied. How?
  - b. How can an opponent attack this system?
- 15.6** The encryption scheme used for UNIX passwords is one way; it is not possible to reverse it. Therefore, would it be accurate to say that this is, in fact, a hash code rather than an encryption of the password?
- 15.7** It was stated that the inclusion of the salt in the UNIX password scheme increases the difficulty of guessing by a factor of 4096. But the salt is stored in plaintext in the same entry as the corresponding ciphertext password. Therefore, those two characters are known to the attacker and need not be guessed. Why is it asserted that the salt increases security?
- 15.8** Assuming that you have successfully answered the preceding problem and understand the significance of the salt, here is another question: Wouldn't it be possible to thwart completely all password crackers by dramatically increasing the salt size to, say, 24 or 48 bits?
- 15.9** Consider the Bloom filter discussed in Section 15.1. Define  $k$  = number of hash functions;  $N$  = number of bits in hash table; and  $D$  = number of words in dictionary.
- a. Show that the expected number of bits in the hash table that are equal to zero is expressed as
 
$$\phi = (1 - \frac{k}{N})^D$$
  - b. Show that the probability that an input word, not in the dictionary, will be falsely accepted as being in the dictionary is
 
$$P = (1 - \phi)^k$$
  - c. Show that the preceding expression can be approximated as
 
$$P \approx (1 - e^{-kD/N})^k$$
- 15.10** There is a flaw in the virus program of Figure 15.8. What is it?



# CHAPTER 16

## FIREWALLS

*The function of a strong position is to make the forces holding it practically unassailable.*

**—On War, Carl Von Clausewitz**

*On the day that you take up your command, block the frontier passes, destroy the official tallies, and stop the passage of all emissaries.*

**—The Art of War, Sun Tzu**

**T**he last topic that we consider is the firewall. Firewalls can be an effective means of protecting a local system or network of systems from network-based security threats while at the same time affording access to the outside world via wide area networks and the Internet.

We begin this chapter with an overview of the functionality and design principles of firewalls. Next, we address the issue of the security of the firewall itself and, in particular, the concept of a trusted system, or secure operating system.

### 16.1 FIREWALL DESIGN PRINCIPLES

Information systems in corporations, government agencies, and other organizations have undergone a steady evolution:

- Centralized data processing system, with a central mainframe supporting a number of directly connected terminals
- Local area network (LAN) interconnecting PCs and terminals to each other and the mainframe

- Premises network, consisting of a number of LANs, interconnecting PCs, servers, and perhaps a mainframe or two
- Enterprise-wide network, consisting of multiple, geographically distributed premises networks interconnected by a private wide area network (WAN)
- Internet connectivity, in which the various premises networks all hook into the Internet and may or may not also be connected by a private WAN

Internet connectivity is no longer an option for most organizations. The information and services available are essential to the organization. Moreover, individual users want and need Internet access, and if this is not provided via their LAN, they will use dial-up capability from their PC to an Internet service provider (ISP). However, while Internet access provides benefits to the organization, it enables the outside world to reach and interact with local network assets. This creates a threat to the organization. While it is possible to equip each workstation and server on the premises network with strong security features, such as intrusion protection, this is not a practical approach. Consider a network with hundreds or even thousands of systems, running a mix of various versions of UNIX, plus Windows 95, 98, and NT. When a security flaw is discovered, each potentially affected system must be upgraded to fix that flaw. The alternative, increasingly accepted, is the firewall. The firewall is inserted between the premises network and the Internet to establish a controlled link and to erect an outer security wall or perimeter. The aim of this perimeter is to protect the premises network from Internet-based attacks and to provide a single choke point where security and audit can be imposed. The firewall may be a single computer system or a set of two or more systems that cooperate to perform the firewall function.

In this section, we look first at the general characteristics of firewalls. Then we look at the types of firewalls currently in common use. Finally, we examine some of the most common firewall configurations.

### Firewall Characteristics

[BELL94] lists the following design goals for a firewall:

1. All traffic from inside to outside, and vice versa, must pass through the firewall. This is achieved by physically blocking all access to the local network except via the firewall. Various configurations are possible, as explained later in this section.
2. Only authorized traffic, as defined by the local security policy, will be allowed to pass. Various types of firewalls are used, which implement various types of security policies, as explained later in this section.
3. The firewall itself is immune to penetration. This implies that use of a trusted system with a secure operating system. This topic is discussed in Section 16.2.

[SMIT97] lists four general techniques that firewalls use to control access and enforce the site's security policy. Originally, firewalls focused primarily on service control, but they have since evolved to provide all four:

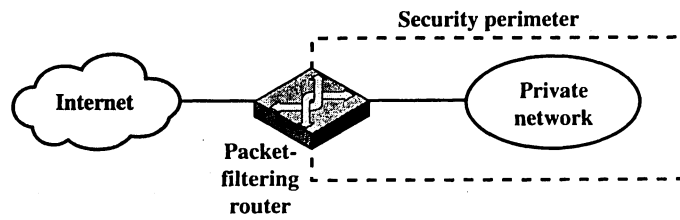
- **Service control:** Determines the types of Internet services that can be accessed, inbound or outbound. The firewall may filter traffic on the basis of IP address and TCP port number; may provide proxy software that receives and interprets each service request before passing it on; or may host the server software itself, such as a Web or mail service.
- **Direction control:** Determines the direction in which particular service requests may be initiated and allowed to flow through the firewall.
- **User control:** Controls access to a service according to which user is attempting to access it. This feature is typically applied to user's inside the firewall perimeter (local users). It may also be applied to incoming traffic from external users; the latter requires some form of secure authentication technology, such as is provided in IPSec (Chapter 13).
- **Behavior control:** Controls how particular services are used. For example, the firewall may filter e-mail to eliminate spam, or it may enable external access to only a portion of the information on a local Web server.

Before proceeding to the details of firewall types and configurations, it is best to summarize what one can expect from a firewall. The following capabilities are within the scope of a firewall:

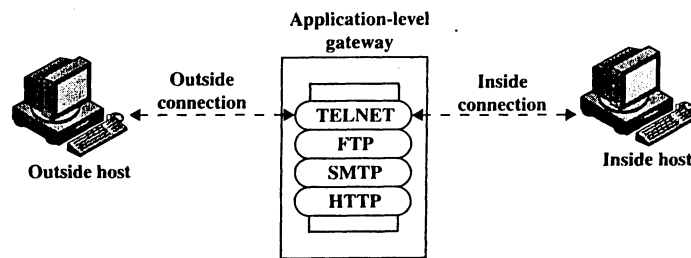
1. A firewall defines a single choke point that keeps unauthorized users out of the protected network, prohibits potentially vulnerable services from entering or leaving the network, and provides protection from various kinds of IP spoofing and routing attacks. The use of a single choke point simplifies security management since security capabilities are consolidated on a single system or set of systems.
2. A firewall provides a location for monitoring security-related events. Audits and alarms can be implemented on the firewall system.
3. A firewall is a convenient platform for several Internet functions that are not security related. These include a network address translator, which maps local addresses to Internet addresses, and a network management function that audits or logs Internet usage.
4. A firewall can serve as the platform for IPSec. Using the tunnel mode capability described in Chapter 13, the firewall can be used to implement virtual private networks.

Firewalls have their limitations, including the following:

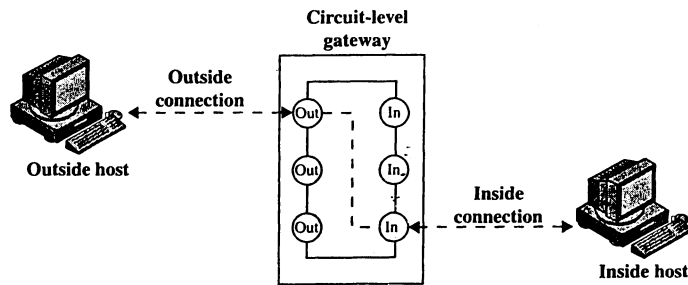
1. The firewall cannot protect against attacks that bypass the firewall. Internal systems may have dial-out capability to connect to an ISP. An internal LAN may support a modem pool that provides dial-in capability for traveling employees and telecommuters.
2. The firewall does not protect against internal threats, such as a disgruntled employee or an employee who unwittingly cooperates with an external attacker.
3. The firewall cannot protect against the transfer of virus-infected programs or files. Because of the variety of operating systems and applications supported



(a) Packet-filtering router



(b) Application-level gateway



(c) Circuit-level gateway

**Figure 16.1** Firewall Types.

inside the perimeter, it would be impractical and perhaps impossible for the firewall to scan all incoming files, e-mail, and messages for viruses.

### Types of Firewalls

Figure 16.1, based on figures in [SEME96], illustrates the three common types of firewalls: packet filters, application-level gateways, and circuit-level gateways. We examine each of these in turn.

#### Packet-Filtering Router

A packet-filtering router applies a set of rules to each incoming IP packet and then forwards or discards the packet. The router is typically configured to filter packets going in both directions (from and to the internal network). Filtering rules



are based on fields in the IP and transport (e.g., TCP or UDP) header, including source and destination IP address, IP protocol field (which defines the transport protocol), and TCP or UDP port number (which defines an application such as SNMP or TELNET).

The packet filter is typically set up as a list of rules based on matches to fields in the IP or TCP header. If there is a match to one of the rules, that rule is invoked to determine whether to forward or discard the packet. If there is no match to any rule, then a default action is taken. Two default policies are possible:

- **Default = discard:** That which is not expressly permitted is prohibited.
- **Default = forward:** That which is not expressly prohibited is permitted.

The default discard policy is the more conservative. Initially, everything is blocked, and services must be added on a case-by-case basis. This policy is more visible to users, who are more likely to see the firewall as a hindrance. The default forward policy increases ease of use for end users but provides reduced security; the security administrator must, in essence, react to each new security threat as it becomes known.

Table 16.1, from [BELL94], gives some examples of packet-filtering rule sets. In each set, the rules are applied top to bottom. The “\*” in a field is a wildcard designator that matches everything. We assume that the default = discard policy is in force.

**Table 16.1** Packet-Filtering Examples

A	action	ourhost	port	theirhost	port	comment	
	block	*	*	SPIGOT	*	we don't trust these people	
	allow	OUR-GW	25	*	*	connection to our SMTP port	

B	action	ourhost	port	theirhost	port	comment	
	block	*	*	*	*	default	

C	action	ourhost	port	theirhost	port	comment	
	allow	*	*	*	25	connection to their SMTP port	

D	action	src	port	dest	port	flags	comment
	allow	{our hosts}	*	*	25		our packets to their SMTP port
	allow	*	25	*	*	ACK	their replies

E	action	src	port	dest	port	flags	comment
	allow	{our hosts}	*	*	*		our outgoing calls
	allow	*	*	*	*	ACK	replies to our calls
	allow	*	*	*	>1024		traffic to nonservers

- A. Inbound mail is allowed (port 25 is for SMTP incoming), but only to a gateway host. However, mail from a particular external host, SPIGOT, is blocked because that host has a history of sending massive files in e-mail messages.
- B. This is an explicit statement of the default policy. All rule sets include this rule implicitly as the last rule.
- C. This rule set is intended to specify that any inside host can send mail to the outside. A TCP packet with a destination port of 25 is routed to the SMTP server on the destination machine. The problem with this rule is that the use of port 25 for SMTP receipt is only a default; an outside machine could be configured to have some other application linked to port 25. As this rule is written, an attacker could gain access to internal machines by sending packets with a TCP source port number of 25.
- D. This rule set achieves the intended result that was not achieved in C. The rules take advantage of a feature of TCP connections. Once a connection is set up, the ACK flag of a TCP segment is set to acknowledge segments sent from the other side. Thus, this rule set states that it allows IP packets where the source IP address is one of a list of designated internal hosts and the destination TCP port number is 25. It also allows incoming packets with a source port number of 25 that include the ACK flag in the TCP segment. Note that we explicitly designate source and destination systems to define these rules explicitly.
- E. This rule set is one approach to handling FTP connections. With FTP, two TCP connections are used: a control connection to set up the file transfer and a data connection for the actual file transfer. The data connection uses a different port number that is dynamically assigned for the transfer. Most servers, and hence most attack targets, live on low-numbered ports; most outgoing calls tend to use a higher-numbered port, typically above 1023. Thus, this rule set allows
  - Packets that originate internally
  - Reply packets to a connection initiated by an internal machine
  - Packets destined for a high-numbered port on an internal machine.

This scheme requires that the systems be configured so that only the appropriate port numbers are in use.

Rule set E points out the difficulty in dealing with applications at the packet-filtering level. Another way to deal with FTP and similar applications is an application-level gateway, described later in this section.

One advantage of a packet-filtering router is its simplicity. Also, packet filters typically are transparent to users and are very fast. The disadvantages include the difficulty of setting up packet filter rules correctly and the lack of authentication.

[SEME96] lists some of the attacks that can be made on packet-filtering routers and the appropriate countermeasures:

- **IP address spoofing:** The intruder transmits packets from the outside with a source IP address field containing an address of an internal host. The attacker hopes that the use of a spoofed address will allow penetration of systems that employ simple source address security, in which packets from specific trusted internal hosts are accepted. The countermeasure is to discard packets with an inside source address if the packet arrives on an external interface.

- **Source routing attacks:** The source station specifies the route that a packet should take as it crosses the Internet, in the hopes that this will bypass security measures that do not analyze the source routing information. The countermeasure is to discard all packets that use this option.
- **Tiny fragment attacks:** The intruder uses the IP fragmentation option to create extremely small fragments and force the TCP header information into a separate packet fragment. This attack is designed to circumvent filtering rules that depend on TCP header information. The attacker hopes that only the first fragment is examined by the filtering router and that the remaining fragments are passed through. A tiny fragment attack can be defeated by discarding all packets where the protocol type is TCP and the IP Fragment Offset is equal to 1.

### Application-Level Gateway

An application-level gateway, also called a proxy server, acts as a relay of application-level traffic (Figure 16.1b). The user contacts the gateway using a TCP/IP application, such as Telnet or FTP, and the gateway asks the user for the name of the remote host to be accessed. When the user responds and provides a valid user ID and authentication information, the gateway contacts the application on the remote host and relays TCP segments containing the application data between the two endpoints. If the gateway does not implement the proxy code for a specific application, the service is not supported and cannot be forwarded across the firewall. Further, the gateway can be configured to support only specific features of an application that the network administrator considers acceptable while denying all other features.

Application-level gateways tend to be more secure than packet filters. Rather than trying to deal with the numerous possible combinations that are to be allowed and forbidden at the TCP and IP level, the application-level gateway need only scrutinize a few allowable applications. In addition, it is easy to log and audit all incoming traffic at the application level.

A prime disadvantage of this type of gateway is the additional processing overhead on each connection. In effect, there are two spliced connections between the end users, with the gateway at the splice point, and the gateway must examine and forward all traffic in both directions.

### Circuit-Level Gateway

A third type of firewall is the circuit-level gateway (Figure 16.1c). This can be a stand-alone system or it can be a specialized function performed by an application-level gateway for certain applications. A circuit-level gateway does not permit an end-to-end TCP connection; rather, the gateway sets up two TCP connections, one between itself and a TCP user on an inner host and one between itself and a TCP user on an outside host. Once the two connections are established, the gateway typically relays TCP segments from one connection to the other without examining the contents. The security function consists of determining which connections will be allowed.

A typical use of circuit-level gateways is a situation in which the system administrator trusts the internal users. The gateway can be configured to support application-level or proxy service on inbound connections and circuit-level functions for outbound connections. In this configuration, the gateway can incur the processing

overhead of examining incoming application data for forbidden functions but does not incur that overhead on outgoing data.

An example of a circuit-level gateway implementation is the SOCKS package [KOBL92]; version 5 of SOCKS is defined in RFC 1928. The RFC defines SOCKS in the following fashion:

The protocol described here is designed to provide a framework for client-server applications in both the TCP and UDP domains to conveniently and securely use the services of a network firewall. The protocol is conceptually a “shim-layer” between the application layer and the transport layer, and as such does not provide network-layer gateway services, such as forwarding of ICMP messages.

SOCKS consists of the following components:

- The SOCKS server, which runs on a UNIX-based firewall.
- The SOCKS client library, which runs on internal hosts protected by the firewall.
- SOCKS-ified versions of several standard client programs such as FTP and TELNET. The implementation of the SOCKS protocol typically involves the recompilation or relinking of TCP-based client applications to use the appropriate encapsulation routines in the SOCKS library.

When a TCP-based client wishes to establish a connection to an object that is reachable only via a firewall (such determination is left up to the implementation), it must open a TCP connection to the appropriate SOCKS port on the SOCKS server system. The SOCKS service is located on TCP port 1080. If the connection request succeeds, the client enters a negotiation for the authentication method to be used, authenticates with the chosen method, and then sends a relay request. The SOCKS server evaluates the request and either establishes the appropriate connection or denies it. UDP exchanges are handled in a similar fashion. In essence, a TCP connection is opened to authenticate a user to send and receive UDP segments, and the UDP segments are forwarded as long as the TCP connection is open.

### **Bastion Host**

A bastion host is a system identified by the firewall administrator as a critical strong point in the network's security. Typically, the bastion host serves as a platform for an application-level or circuit-level gateway. The following are common characteristics of a bastion host:

- The bastion host hardware platform executes a secure version of its operating system, making it a trusted system.
- Only the services that the network administrator considers essential are installed on the bastion host. These include proxy applications such as Telnet, DNS, FTP, SMTP, and user authentication.
- The bastion host may require additional authentication before a user is allowed access to the proxy services. In addition, each proxy service may require its own authentication before granting user access.

- Each proxy is configured to support only a subset of the standard application's command set.
- Each proxy is configured to allow access only to specific host systems. This means that the limited command/feature set may be applied only to a subset of systems on the protected network.
- Each proxy maintains detailed audit information by logging all traffic, each connection, and the duration of each connection. The audit log is an essential tool for discovering and terminating intruder attacks.
- Each proxy module is a very small software package specifically designed for network security. Because of its relative simplicity, it is easier to check such modules for security flaws. For example, a typical UNIX mail application may contain over 20,000 lines of code, while a mail proxy may contain fewer than 1000 [SEME96].
- Each proxy is independent of other proxies on the bastion host. If there is a problem with the operation of any proxy, or if a future vulnerability is discovered, it can be uninstalled without affecting the operation of the other proxy applications. Also, if the user population requires support for a new service, the network administrator can easily install the required proxy on the bastion host.
- A proxy generally performs no disk access other than to read its initial configuration file. This makes it difficult for an intruder to install Trojan horse sniffers or other dangerous files on the bastion host.
- Each proxy runs as a nonprivileged user in a private and secured directory on the bastion host.

### Firewall Configurations

In addition to the use of a simple configuration consisting of a single system, such as a single packet-filtering router or a single gateway (Figure 16.1), more complex configurations are possible and indeed more common. Figure 16.2, based on figures in [SEME96], illustrates three common firewall configurations. We examine each of these in turn.

In the **screened host firewall, single-homed bastion** configuration (Figure 16.2a), the firewall consists of two systems: a packet-filtering router and a bastion host. Typically, the router is configured so that

1. For traffic from the Internet, only IP packets destined for the bastion host are allowed in.
2. For traffic from the internal network, only IP packets from the bastion host are allowed out.

The bastion host performs authentication and proxy functions. This configuration has greater security than simply a packet-filtering router or an application-level gateway alone, for two reasons. First, this configuration implements both packet-level and application-level filtering, allowing for considerable flexibility in defining security policy. Second, an intruder must generally penetrate two separate systems before the security of the internal network is compromised.

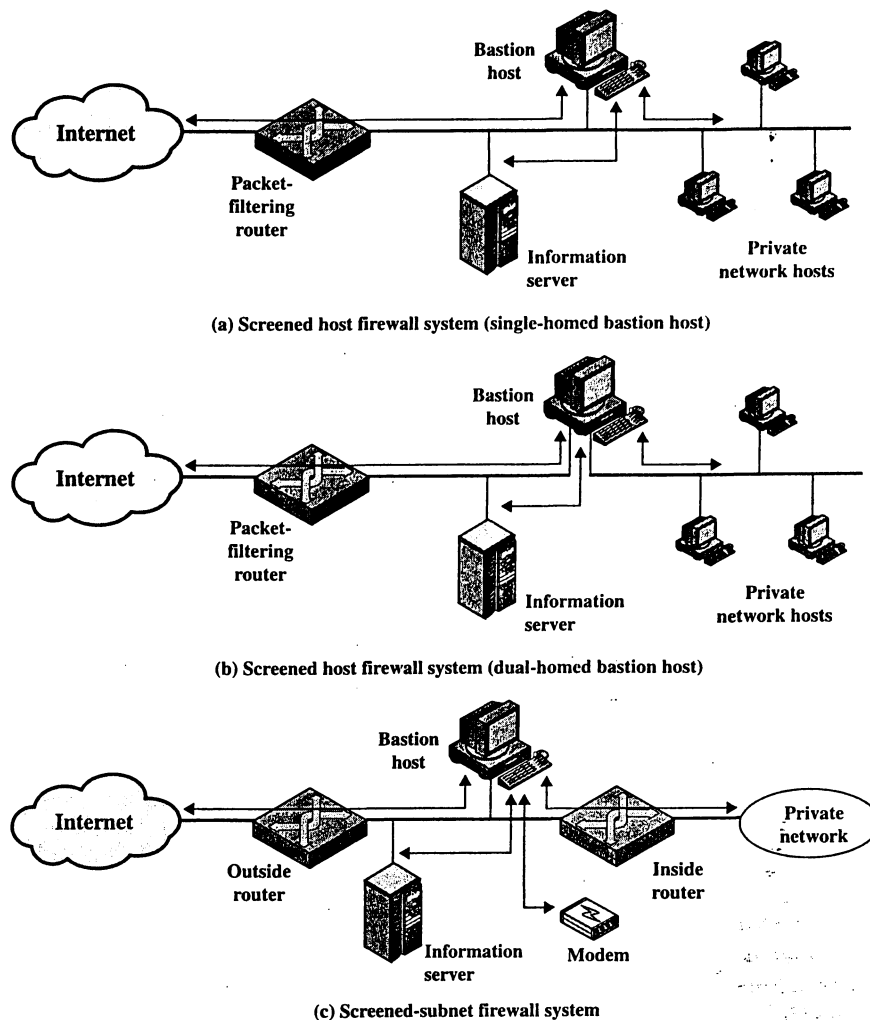


Figure 16.2 Firewall Configurations.

This configuration also affords flexibility in providing direct Internet access. For example, the internal network may include a public information server, such as a Web server, for which a high level of security is not required. In that case, the router can be configured to allow direct traffic between the information server and the Internet.

In the single-homed configuration just described, if the packet-filtering router is completely compromised, traffic could flow directly through the router between the Internet and other hosts on the private network. The **screened host firewall, dual-homed bastion** configuration physically prevents such a security breach (Figure 16.2b). The advantages of dual layers of security that were present in the previous configuration are present here as well. Again, an information server or other hosts can be allowed direct communication with the router if this is in accord with the security policy.

The **screened subnet firewall** configuration of Figure 16.2c is the most secure of those we have considered. In this configuration, two packet-filtering routers are used, one between the bastion host and the Internet and one between the bastion host and the internal network. This configuration creates an isolated subnetwork, which may consist of simply the bastion host, but may also include one or more information servers and modems for dial-in capability. Typically, both the Internet and the internal network have access to hosts on the screened subnet, but traffic across the screened subnet is blocked. This configuration offers several advantages:

- There are now three levels of defense to thwart intruders.
- The outside router advertises only the existence of the screened subnet to the Internet; therefore, the internal network is invisible to the Internet.
- Similarly, the inside router advertises only the existence of the screened subnet to the internal network; therefore, the systems on the inside network cannot construct direct routes to the Internet.

## 16.2 TRUSTED SYSTEMS

One way to enhance the ability of a system to defend against intruders and malicious programs is to implement trusted system technology. This section provides a brief overview of this topic. We begin by looking at some basic concepts of data access control.

### Data Access Control

Following successful logon, the user has been granted access to one or a set of hosts and applications. This is generally not sufficient for a system that includes sensitive data in its database. Through the user access control procedure, a user can be identified to the system. Associated with each user, there can be a profile that specifies permissible operations and file accesses. The operating system can then enforce rules based on the user profile. The database management system, however, must control access to specific records or even portions of records. For example, it may be permissible for anyone in administration to obtain a list of company personnel, but only selected individuals may have access to salary information. The issue is more than just one of level of detail. Whereas the operating system may grant a user permission to access a file or use an application, following which there are no further security checks, the database management system must make a decision on each individual access attempt. That decision will depend not only on the user's identity but also on the specific parts of the data being accessed and even on the information already divulged to the user.

A general model of access control as exercised by a file or database management system is that of an **access matrix** (Figure 16.3a). The basic elements of the model are as follows:

	Program1	...	SegmentA	SegmentB
Process1	Read Execute		Read Write	
Process2				Read
•				
•				
•				

(a) Access Matrix

<b>Access Control List for Program1:</b> Process1 (Read, Execute)
<b>Access Control List for SegmentA:</b> Process1 (Read, Write)
<b>Access Control List for SegmentB:</b> Process2 (Read)

(b) Access Control List

<b>Capability List for Process1:</b> Program1 (Read, Execute) SegmentA (Read, Write)
<b>Capability List for Process2:</b> SegmentB (Read)

(c) Capability List

**Figure 16.3** Access Control Structures.

- **Subject:** An entity capable of accessing objects. Generally, the concept of subject equates with that of process. Any user or application actually gains access to an object by means of a process that represents that user or application.
- **Object:** Anything to which access is controlled. Examples include files, portions of files, programs, and segments of memory.
- **Access right:** The way in which an object is accessed by a subject. Examples are read, write, and execute.

One axis of the matrix consists of identified subjects that may attempt data access. Typically, this list will consist of individual users or user groups, although access could be controlled for terminals, hosts, or applications instead of or in addition to users. The other axis lists the objects that may be accessed. At the greatest level of detail, objects may be individual data fields. More aggregate groupings, such as records, files, or even the entire database, may also be objects in the matrix. Each entry in the matrix indicates the access rights of that subject for that object.



In practice, an access matrix is usually sparse and is implemented by decomposition in one of two ways. The matrix may be decomposed by columns, yielding **access control lists** (Figure 16.3b). Thus, for each object, an access control list lists users and their permitted access rights. The access control list may contain a default, or public, entry. This allows users that are not explicitly listed as having special rights to have a default set of rights. Elements of the list may include individual users as well as groups of users.

Decomposition by rows yields **capability tickets** (Figure 16.3c). A capability ticket specifies authorized objects and operations for a user. Each user has a number of tickets and may be authorized to loan or give them to others. Because tickets may be dispersed around the system, they present a greater security problem than access control lists. In particular, the ticket must be unforgeable. One way to accomplish this is to have the operating system hold all tickets on behalf of users. These tickets would have to be held in a region of memory inaccessible to users.

### The Concept of Trusted Systems

Much of what we have discussed so far has been concerned with protecting a given message or item from passive or active attack by a given user. A somewhat different but widely applicable requirement is to protect data or resources on the basis of levels of security. This is commonly found in the military, where information is categorized as unclassified (U), confidential (C), secret (S), top secret (TS), or beyond. This concept is equally applicable in other areas, where information can be organized into gross categories and users can be granted clearances to access certain categories of data. For example, the highest level of security might be for strategic corporate planning documents and data, accessible by only corporate officers and their staff; next might come sensitive financial and personnel data, accessible only by administration personnel, corporate officers, and so on.

When multiple categories or levels of data are defined, the requirement is referred to as **multilevel security**. The general statement of the requirement for multilevel security is that a subject at a high level may not convey information to a subject at a lower or noncomparable level unless that flow accurately reflects the will of an authorized user. For implementation purposes, this requirement is in two parts and is simply stated. A multilevel secure system must enforce

- **No read up:** A subject can only read an object of less or equal security level. This is referred to in the literature as the **simple security property**.
- **No write down:** A subject can only write into an object of greater or equal security level. This is referred to in the literature as the **\*-Property**<sup>1</sup> (pronounced *star property*).

These two rules, if properly enforced, provide multilevel security. For a data processing system, the approach that has been taken, and has been the object of much

<sup>1</sup>The "\*" does not stand for anything. No one could think of an appropriate name for the property during the writing of the first report on the model. The asterisk was a dummy character entered in the draft so that a text editor could rapidly find and replace all instances of its use once the property was named. No name was ever devised, and so the report was published with the "\*" intact.

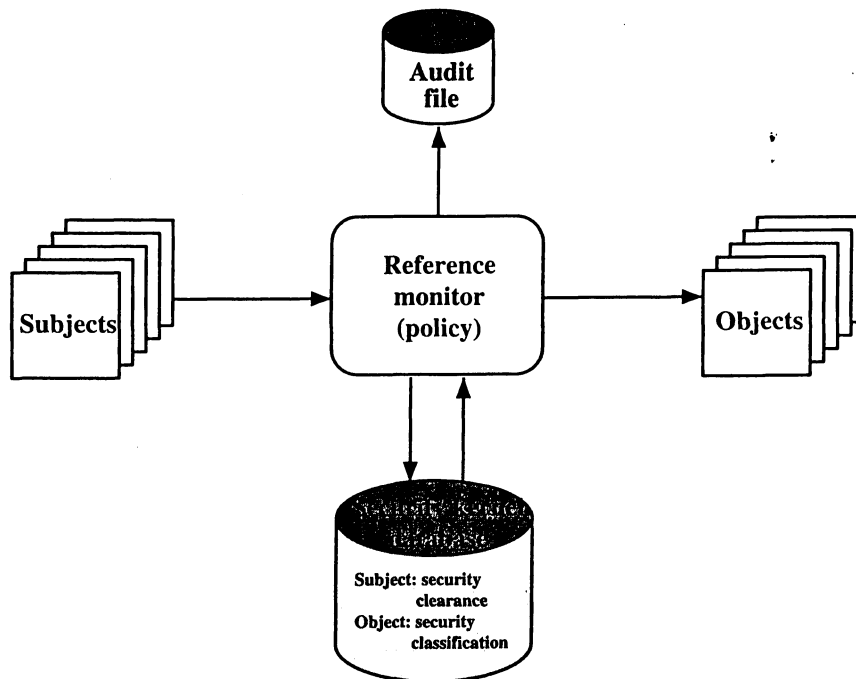


Figure 16.4 Reference Monitor Concept.

research and development, is based on the *reference monitor* concept. This approach is depicted in Figure 16.4. The reference monitor is a controlling element in the hardware and operating system of a computer that regulates the access of subjects to objects on the basis of security parameters of the subject and object. The reference monitor has access to a file, known as the *security kernel database*, that lists the access privileges (security clearance) of each subject and the protection attributes (classification level) of each object. The reference monitor enforces the security rules (no read up, no write down) and has the following properties:

- **Complete mediation:** The security rules are enforced on every access, not just, for example, when a file is opened.
- **Isolation:** The reference monitor and database are protected from unauthorized modification.
- **Verifiability:** The reference monitor's correctness must be provable. That is, it must be possible to demonstrate mathematically that the reference monitor enforces the security rules and provides complete mediation and isolation.

These are stiff requirements. The requirement for complete mediation means that every access to data within main memory and on disk and tape must be mediated. Pure software implementations impose too high a performance penalty to be practical; the solution must be at least partly in hardware. The requirement for

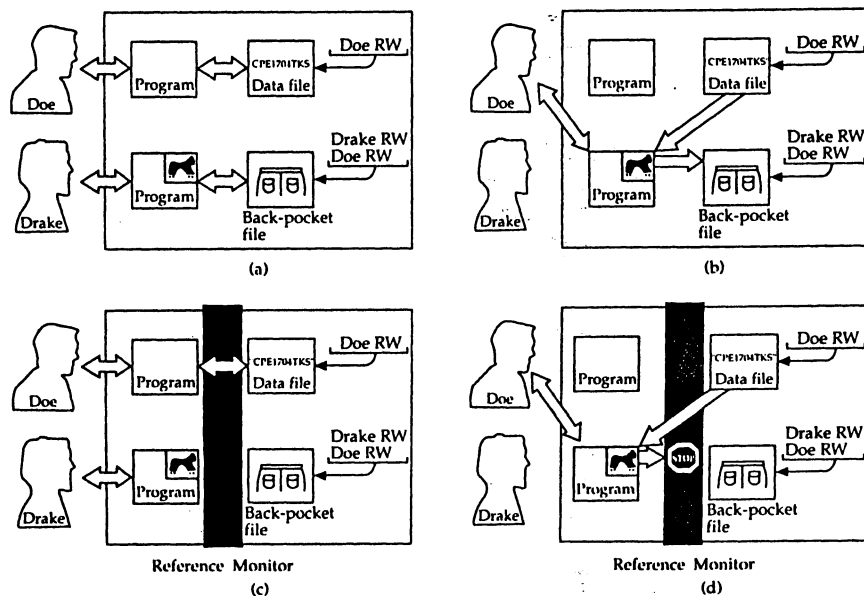
isolation means that it must not be possible for an attacker, no matter how clever, to change the logic of the reference monitor or the contents of the security kernel database. Finally, the requirement for mathematical proof is formidable for something as complex as a general-purpose computer. A system that can provide such verification is referred to as a **trusted system**.

A final element illustrated in Figure 16.4 is an audit file. Important security events, such as detected security violations and authorized changes to the security kernel database, are stored in the audit file.

In an effort to meet its own needs and as a service to the public, the U.S. Department of Defense in 1981 established the Computer Security Center within the National Security Agency (NSA) with the goal of encouraging the widespread availability of trusted computer systems. This goal is realized through the center's Commercial Product Evaluation Program. In essence, the center attempts to evaluate commercially available products as meeting the security requirements just outlined. The center classifies evaluated products according to the range of security features that they provide. These evaluations are needed for Department of Defense procurements but are published and freely available. Hence, they can serve as guidance to commercial customers for the purchase of commercially available, off-the-shelf equipment.

### Trojan Horse Defense

One way to secure against Trojan horse attacks is the use of a secure, trusted operating system. Figure 16.5 illustrates an example. In this case, a Trojan horse is used to get around the standard security mechanism used by most file management and



**Figure 16.5** Trojan Horse and Secure Operating System.

operating systems: the access control list. In this example, a user named Doe interacts through a program with a data file containing the critically sensitive character string "CPE1704TKS." User Doe has created the file with read/write permission provided only to programs executing on his own behalf: that is, only processes that are owned by Doe may access the file.

The Trojan horse attack begins when a hostile user, named Drake, gains legitimate access to the system, and installs both a Trojan horse program and a private file to be used in the attack as a "back pocket." Drake gives read/write permission to himself for this file and gives Doe write-only permission (Figure 16.5a). Drake now induces Doe to invoke the Trojan horse program, perhaps by advertising it as a useful utility. When the program detects that it is being executed by Doe, it copies the sensitive character string from Doe's file and copies it into Drake's back-pocket file (Figure 16.5b). Both the read and write operations satisfy the constraints imposed by access control lists. Drake then has only to access his file at a later time to learn the value of the string.

Now consider the use of a secure operating system in this scenario (Figure 16.5c). Security levels are assigned to subjects at logon on the basis of criteria such as the terminal from which the computer is being accessed and the user involved, as identified by password/ID. In this example, there are two security levels, sensitive and public, ordered so that sensitive is higher than public. Processes owned by Doe and Doe's data file are assigned the security level sensitive. Drake's file and processes are restricted to public. If Doe invokes the Trojan horse program (Figure 16.5d), that program acquires Doe's security level. It is therefore able, under the simple security property, to observe the sensitive character string. When the program attempts to store the string in a public file (the back-pocket file), however, the \*-Property is violated and the attempt is disallowed by the reference monitor. Thus, the attempt to write into the back-pocket file is denied even though the access control list permits it: The security policy takes precedence over the access control list mechanism.

### 16.3 RECOMMENDED READING

The two classic treatments of firewalls, still well worth reading, are [CHES94] and [CHAP95]. [LODI98], [OPPL97] and [BELL94] are good overview articles on the subject.

[GASS88] provides a comprehensive study of trusted computer systems. Another good source of information is [PFLE97]. Reprints of some key papers on this topic are to be found in [ABRA87].

ABRA87 Abrams, M., and Podell, H. *Computer and Network Security*. Los Alamitos, CA: IEEE Computer Society Press, 1987.

BELL94 Bellovin, S., and Cheswick, W. "Network Firewalls." *IEEE Communications Magazine*, September 1994.

CHAP95 Chapman, D., and Zwicky, E. *Building Internet Firewalls*. Sebastopol, CA: O'Reilly, 1995.

CHES94 Cheswick, W., and Bellovin, S. *Firewalls and Internet Security: Repelling the Wily Hacker*. Reading, MA: Addison-Wesley, 1994.

- GASS88 Gasser, M. *Building a Secure Computer System*. New York: Van Nostrand Reinhold, 1988.
- LODI98 Lodin, S., and Schuba, C. "Firewalls Fend Off Invasions from the Net." *IEEE Spectrum*, February 1998.
- OPPL97 Oppliger, R. "Internet Security: Firewalls and Beyond." *Communications of the ACM*, May 1997.
- PFLE97 Pfleeger, C. *Security in Computing*. Upper Saddle River, NJ: Prentice Hall, 1997.

## 16.4 PROBLEMS

- 16.1 The necessity of the "no read up" rule for a multilevel secure system is fairly obvious. What is the importance of the "no write down" rule?
- 16.2 In Figure 16.5 one link of the Trojan horse copy-and-observe-later chain is broken. There are two other possible angles of attack by Drake: Drake logging on and attempting to read the string directly, and Drake assigning a security level of sensitive to the back-pocket file. Does the reference monitor prevent these attacks?



# APPENDIX A

## PROJECTS FOR TEACHING CRYPTOGRAPHY AND NETWORK SECURITY

*Analysis and observation, theory and experience must never disdain or exclude each other; on the contrary, they support each other.*

—On War, Carl Von Clausewitz

Many instructors believe that implementation projects are crucial to the clear understanding of cryptography and network security. Without projects, it may be difficult for students to grasp some of the basic concepts and interactions among components. Projects reinforce the concepts introduced in the book, give the student a greater appreciation of how a cryptographic algorithm or protocol works, and can motivate students and give them confidence that they are capable of not only understanding but implementing the details of a security capability.

In this text, I have tried to present the concepts of cryptography and network security as clearly as possible and have provided approximately 140 homework problems to reinforce those concepts. However, many instructors will wish to supplement this material with projects. This appendix provides some guidance in that regard and describes support material available in the instructor's manual. The support material covers three types of projects:

- Research projects
- Programming projects
- Reading/report projects

## A.1 RESEARCH PROJECTS

An effective way of reinforcing basic concepts from the course and for teaching students research skills is to assign a research project. Such a project could involve a literature search as well as an Internet search of vendor products, research lab activities, and standardization efforts. Projects could be assigned to teams or, for smaller projects, to individuals. In any case, it is best to require some sort of project proposal early in the term, giving the instructor time to evaluate the proposal for appropriate topic and appropriate level of effort. Student handouts for research projects should include

- A format for the proposal
- A format for the final report
- A schedule with intermediate and final deadlines
- A list of possible project topics

The students can select one of the listed topics or devise their own comparable project. The instructor's manual includes a suggested format for the proposal and final report as well as a list of fifteen possible research topics.

## A.2 PROGRAMMING PROJECTS

The programming project is a useful pedagogical tool. There are several attractive features of stand-alone programming projects that are not part of an existing security facility:

1. The instructor can choose from a wide variety of cryptography and network security concepts to assign projects.
2. The projects can be programmed by the students on any available computer and in any appropriate language: they are platform- and language-independent.
3. The instructor need not download, install, and configure any particular infrastructure for stand-alone projects.

There is also flexibility in the size of projects. Larger projects give students more a sense of achievement, but students with less ability or fewer organizational skills can be left behind. Larger projects usually elicit more overall effort from the best students. Smaller projects can have a higher concepts-to-code ratio, and because more of them can be assigned, the opportunity exists to address a variety of different areas.

Again, as with research projects, the students should first submit a proposal. The student handout should include the same elements listed in Section A.1. The instructor's manual includes a set of twelve possible programming projects.



The following individuals have supplied the research and programming projects suggested in the instructor's manual: Henning Schulzrinne of Columbia University; Cetin Kaya Koc of Oregon State University; and David M. Balenson of Trusted Information Systems and George Washington University.

### **A.3 READING/REPORT ASSIGNMENTS**

Another excellent way to reinforce concepts from the course and to give students research experience is to assign papers from the literature to be read and analyzed. The following is a suggested assignment wording:

Read and report on the following paper from the research literature. Your report should be one to two pages long; three-quarters of the report should summarize the paper, and one-quarter of the report should be a critique. Introduce your report with a formal citation of the paper, using the format found in the References section of the textbook.

The instructor's manual includes a suggested list of papers to be assigned. All of the papers are readily available either via the Internet or in any good college technical library. One paper is listed for each chapter in the book.



# GLOSSARY

*In studying the Imperium, Arrakis, and the whole culture which produced Maud'Dib, many unfamiliar terms occur. To increase understanding is a laudable goal, hence the definitions and explanations given below.*

—*Dune*, Frank Herbert

**S**ome of the terms in this glossary are from the *Glossary of Computer Security Terminology* [NIST91]. These are indicated in the glossary by an asterisk.

**Asymmetric Encryption** A form of cryptosystem in which encryption and decryption are performed using two different keys, one of which is referred to as the public key and one of which is referred to as the private key. Also known as public-key encryption.

**Authentication\*** A process used to verify the integrity of transmitted data, especially a message.

**Authenticator** Additional information appended to a message to enable the receiver to verify that the message should be accepted as authentic. The authenticator may be functionally independent of the content of the message itself (e.g., a nonce or a source identifier) or it may be a function of the message contents (e.g., a hash value or a cryptographic checksum).

**Avalanche Effect** A characteristic of an encryption algorithm in which a small change in the plaintext or key gives rise to a large change in the ciphertext. For a hash code, the avalanche effect is a characteristic in which a small change in the message gives rise to a large change in the message digest.

**Bacteria** Program that consumes system resources by replicating itself.

**Block Chaining** A procedure used during symmetric block encryption that makes an output block dependent not only on the current plaintext input block and key, but also on earlier input and/or output. The effect of block chaining is that two instances of the same plaintext input block will produce different ciphertext blocks, making cryptanalysis more difficult.

**Block Cipher** A symmetric encryption algorithm in which a large block of plaintext bits (typically 64) is transformed as a whole into a ciphertext block of the same length.

**Cipher** An algorithm for encryption and decryption. A cipher replaces a piece of information (an element in plaintext) with another object, with the intent to conceal meaning. Typically, the replacement rule is governed by a secret key.

**Ciphertext** The output of an encryption algorithm; the encrypted form of a message or data.

**Code** An unvarying rule for replacing a piece of information (e.g., letter, word, phrase) with another object, not necessarily of the same sort. Generally, there is no intent to conceal meaning. Examples include the ASCII character code (each character is represented by 7 bits) and frequency-shift keying (each binary value is represented by a particular frequency).

**Computationally Secure** Secure because the time and/or cost of defeating the security is too high to be feasible.

**Confusion** A cryptographic technique that seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible. This is achieved by the use of a complex scrambling algorithm that depends on the key and the input.

**Conventional Encryption** Symmetric encryption.

**Covert Channel** A communications channel that enables the transfer of information in a way unintended by the designers of the communications facility.

**Cryptanalysis** The branch of cryptology dealing with the breaking of a cipher to recover information, or forging encrypted information that will be accepted as authentic.

**Cryptographic Checksum** An authenticator that is a cryptographic function of both the data to be authenticated and a secret key. Also referred to as a message authentication code (MAC).

**Cryptography** The branch of cryptology dealing with the design of algorithms for encryption and decryption, intended to ensure the secrecy and/or authenticity of messages.

**Cryptology** The study of secure communications, which encompasses both cryptography and cryptanalysis.

**Decryption** The translation of encrypted text or data (called ciphertext) into original text or data (called plaintext). Also called deciphering.

**Differential Cryptanalysis** A technique in which chosen plaintexts with particular XOR difference patterns are encrypted. The difference patterns of the resulting ciphertext provide information that can be used to determine the encryption key.

**Diffusion** A cryptographic technique that seeks to obscure the statistical structure of the plaintext by spreading out the influence of each individual plaintext digit over many ciphertext digits.

**Digital Signature** An authentication mechanism that enables the creator of a message to attach a code that acts as a signature. The signature guarantees the source and integrity of the message.

**Digram** A two-letter sequence. In English and other languages, the relative frequency of various digrams in plaintext can be used in the cryptanalysis of some ciphers. Also called *digraph*.

**Discretionary Access Control\*** A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing on that permission (perhaps indirectly) to any other subject (unless restrained by mandatory access control).

**Encryption** The conversion of plaintext or data into unintelligible form by means of a reversible translation, based on a translation table or algorithm. Also called enciphering.

**Hash Function** A function that maps a variable-length data block or message into a fixed-length value called a hash code. The function is designed in such a way that, when protected, it provides an authenticator to the data or message. Also referred to as a message digest.

**Initialization Vector** A random block of data that is used to begin the encryption of multiple blocks of plaintext, when a block-chaining encryption technique is used. The IV serves to foil known-plaintext attacks.

**Intruder** An individual who gains, or attempts to gain, unauthorized access to a computer system or to gain unauthorized privileges on that system.

**Kerberos** The name given to Project Athena's code authentication service.

**Key Distribution Center** A system that is authorized to transmit temporary session keys to principals. Each session key is transmitted in encrypted form, using a master key that the key distribution center shares with the target principal.

**Logic Bomb** Logic embedded in a computer program that checks for a certain set of conditions to be present on the system. When these conditions are met, it executes some function resulting in unauthorized actions.

**Mandatory Access Control** A means of restricting access to objects based on fixed security attributes assigned to users and to files and other objects. The controls are mandatory in the sense that they cannot be modified by users or their programs.

**Master Key** A long-lasting key that is used between a key distribution center and a principal for the purpose of encoding the transmission of session keys. Typically, the master keys are distributed by noncryptographic means. Also referred to as a key-encrypting key.

**Message Authentication Code (MAC)** Cryptographic checksum.

**Message Digest** Hash function.

**Multilevel Security** A capability that enforces access control across multiple levels of classification of data.

**Multiple Encryption** Repeated use of an encryption function, with different keys, to produce a more complex mapping from plaintext to ciphertext.

**Nonce** An identifier or number that is used only once.

**One-Way Function** A function that is easily computed, but the calculation of its inverse is infeasible.

**Password\*** A character string used to authenticate an identity. Knowledge of the password and its associated user ID is considered proof of authorization to use the capabilities associated with that user ID.

**Plaintext** The input to an encryption function or the output of a decryption function.

**Private Key** One of the two keys used in an asymmetric encryption system. For secure communication, the private key should only be known to its creator.

**Pseudorandom Number Generator** A function that deterministically produces a sequence of numbers that are apparently statistically random.

**Public Key** One of the two keys used in an asymmetric encryption system. The public key is made public, to be used in conjunction with a corresponding private key.

**Public-Key Encryption** Asymmetric encryption.

**Replay Attacks** An attack in which a service already authorized and completed is forged by another "duplicate request" in an attempt to repeat authorized commands.

**RSA Algorithm** A public-key encryption algorithm based on exponentiation in modular arithmetic. It is the only algorithm generally accepted as practical and secure for public-key encryption.

**Secret Key** The key used in a symmetric encryption system. Both participants must share the same key, and this key must remain secret to protect the communication.

**Session Key** A temporary encryption key used between two principals.

**Stream Cipher** A symmetric encryption algorithm in which ciphertext output is produced bit-by-bit or byte-by-byte from a stream of plaintext input.

**Symmetric Encryption** A form of cryptosystem in which encryption and decryption are performed using the same key. Also known as conventional encryption.

**Trap Door** Secret undocumented entry point into a program, used to grant access without normal methods of access authentication.

**Trap-Door One-Way Function** A function that is easily computed; the calculation of its inverse is infeasible unless certain privileged information is known.

**Trojan Horse\*** A computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit the legitimate authorizations of the invoking process to the detriment of security.

**Trusted System** A computer and operating system that can be verified to implement a given security policy.

**Unconditionally Secure** Secure even against an opponent with unlimited time and unlimited computing resources.

**Virus** Code embedded within a program that causes a copy of itself to be inserted in one or more other programs. In addition to propagation, the virus usually performs some unwanted function.

**Worm** Program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. In addition to propagation, the worm usually performs some unwanted function.



# CRYPTOGRAPHY AND NETWORK SECURITY

## *Principles and Practice*

### Second Edition

### William Stallings

As we enter the age of universal electronic connectivity in which viruses, hackers, electronic eavesdropping, and electronic fraud can threaten the prosperity and productivity of corporations and individuals, security is increasingly important. Fortunately, the disciplines of cryptography and network security have matured, leading to the development of practical, available applications to enforce network security.

Best-selling author and two-time winner of the TEXTY award for the best computer science and engineering text, William Stallings provides a practical survey of both the principles and practice of cryptography and network security.

Extensively reorganized to provide the optimal sequence for classroom instruction and self-study, the second edition includes these key features.

- Looks at system-level security issues, including the threat of and countermeasures for intruders and viruses, and the use of firewalls and trusted systems.
- NEW — Discussion of block cipher design principles, plus coverage of Blowfish, CAST-128, Triple DES, and other algorithms
- NEW — Chapters on IP security and Web security
- Expanded coverage of public-key encryption algorithms and design principles, including RSA and elliptic curve cryptography.
- Covers important network security tools and applications, including Kerberos, X.509v3, PGP, S/MIME, IP security, SSL/TLS, and SET.
- On-line transparency masters, an Internet mailing list, and links to relevant web sites are available to <http://www.shore.net/~ws/Security2e.html>

William Stallings has made a unique contribution to understanding the broad sweep of technical developments in computer networking and computer architecture. He has authored 17 titles, and counting revised editions, a total of 37 books on various aspects of these subjects. He is an independent consultant whose clients have included computer and networking manufacturers and customers, software development firms, and leading-edge government research institutions. Dr. Stallings holds a Ph.D. from M.I.T. in Computer Science and a B.S. from Notre Dame in electrical engineering. All of his Prentice Hall titles can be found at the Prentice Hall web site <http://www.prenhall.com>.

PRENTICE HALL  
Upper Saddle River, NJ 07458  
<http://www.prenhall.com>

