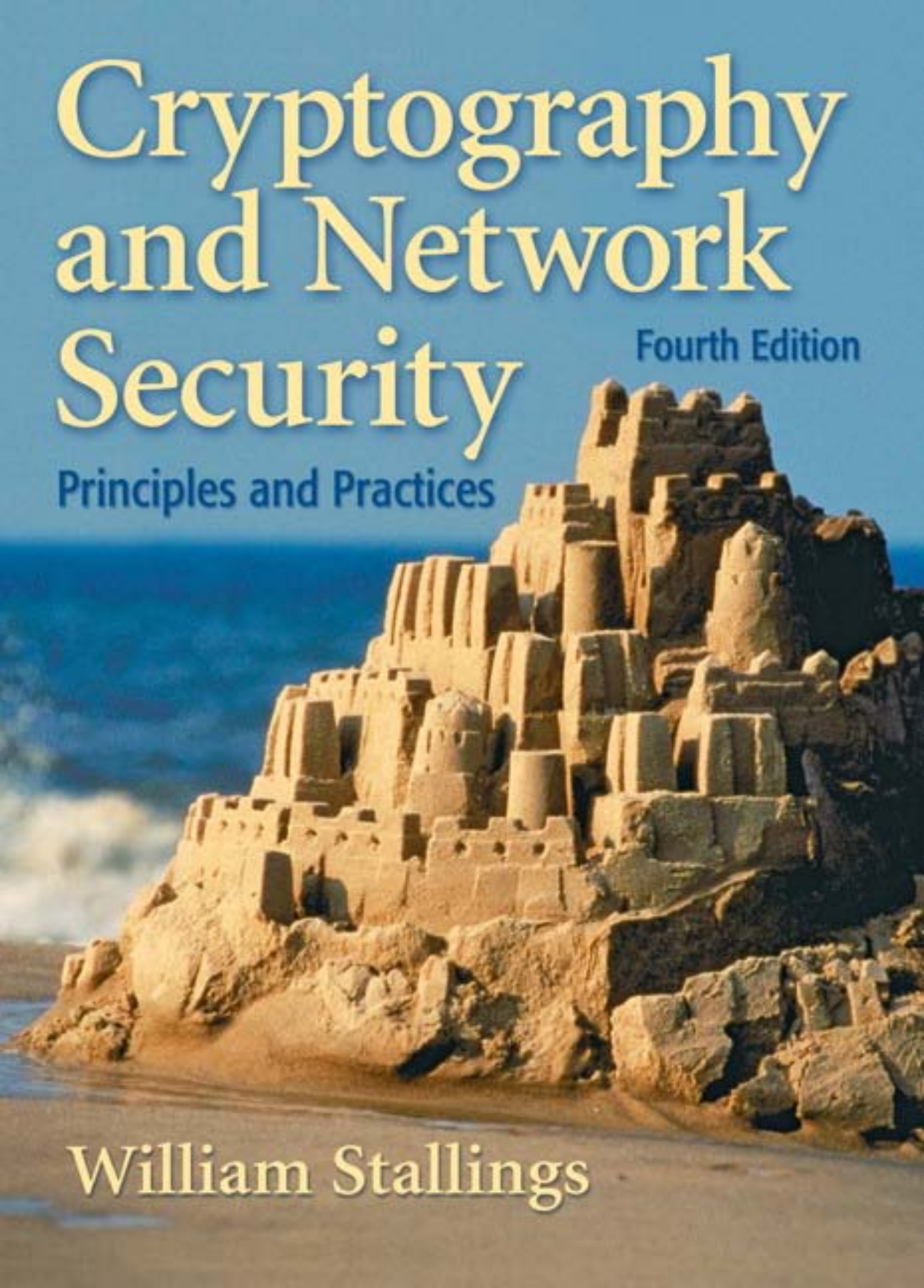


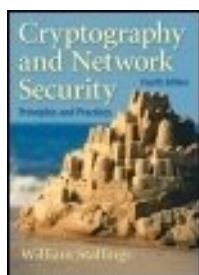
Cryptography and Network Security

Fourth Edition

Principles and Practices

A large, intricate sandcastle built on a beach. The sandcastle is multi-tiered with various towers, battlements, and decorative elements. The ocean waves are crashing against the base of the sandcastle, creating white foam. The sky is a clear, bright blue.

William Stallings



Cryptography and Network Security Principles and Practices, Fourth Edition

By William Stallings

Publisher: Prentice Hall

Pub Date: November 16, 2005

Print ISBN-10: 0-13-187316-4

Print ISBN-13: 978-0-13-187316-2

eText ISBN-10: 0-13-187319-9

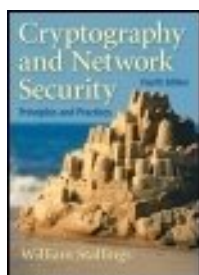
eText ISBN-13: 978-0-13-187319-3

Pages: 592

- [Table of Contents](#)
- [Index](#)

In this age of viruses and hackers, of electronic eavesdropping and electronic fraud, security is paramount.

As the disciplines of cryptography and network security have matured, more practical, readily available applications to enforce network security have developed. This text provides a practical survey of both the principles and practice of cryptography and network security. First, the basic issues to be addressed by a network security capability are explored through a tutorial and survey of cryptography and network security technology. Then, the practice of network security is explored via practical applications that have been implemented and are in use today.



Cryptography and Network Security Principles and Practices, Fourth Edition

By William Stallings

Publisher: Prentice Hall

Pub Date: November 16, 2005

Print ISBN-10: 0-13-187316-4

Print ISBN-13: 978-0-13-187316-2

eText ISBN-10: 0-13-187319-9

eText ISBN-13: 978-0-13-187319-3

Pages: 592

- [Table of Contents](#)
- [Index](#)

Copyright	
Notation	xi
Preface	xiii
Objectives	xiii
Intended Audience	xiii
Plan of the Book	xiv
Internet Services for Instructors and Students	xiv
Projects for Teaching Cryptography and Network Security	xiv
What's New in the Fourth Edition	xv
Acknowledgments	xvi
Chapter 0. Reader's Guide	1
Section 0.1. Outline of this Book	2
Section 0.2. Roadmap	2
Section 0.3. Internet and Web Resources	4
Chapter 1. Introduction	6
Section 1.1. Security Trends	9
Section 1.2. The OSI Security Architecture	12
Section 1.3. Security Attacks	13
Section 1.4. Security Services	16
Section 1.5. Security Mechanisms	19
Section 1.6. A Model for Network Security	22
Section 1.7. Recommended Reading and Web Sites	24
Section 1.8. Key Terms, Review Questions, and Problems	25
Part One: Symmetric Ciphers	26
Chapter 2. Classical Encryption Techniques	28
Section 2.1. Symmetric Cipher Model	30

Section 2.2. Substitution Techniques	35
Section 2.3. Transposition Techniques	49
Section 2.4. Rotor Machines	51
Section 2.5. Steganography	53
Section 2.6. Recommended Reading and Web Sites	55
Section 2.7. Key Terms, Review Questions, and Problems	56
Chapter 3. Block Ciphers and the Data Encryption Standard	62
Section 3.1. Block Cipher Principles	64
Section 3.2. The Data Encryption Standard	72
Section 3.3. The Strength of Des	82
Section 3.4. Differential and Linear Cryptanalysis	83
Section 3.5. Block Cipher Design Principles	86
Section 3.6. Recommended Reading	90
Section 3.7. Key Terms, Review Questions, and Problems	90
Chapter 4. Finite Fields	95
Section 4.1. Groups, Rings, and Fields	97
Section 4.2. Modular Arithmetic	101
Section 4.3. The Euclidean Algorithm	107
Section 4.4. Finite Fields of The Form $GF(p)$	109
Section 4.5. Polynomial Arithmetic	113
Section 4.6. Finite Fields Of the Form $GF(2^n)$	119
Section 4.7. Recommended Reading and Web Sites	129
Section 4.8. Key Terms, Review Questions, and Problems	130
Chapter 5. Advanced Encryption Standard	134
Section 5.1. Evaluation Criteria For AES	135
Section 5.2. The AES Cipher	140
Section 5.3. Recommended Reading and Web Sites	160
Section 5.4. Key Terms, Review Questions, and Problems	161
Appendix 5A Polynomials with Coefficients in $GF(28)$	163
Appendix 5B Simplified AES	165
Chapter 6. More on Symmetric Ciphers	174
Section 6.1. Multiple Encryption and Triple DES	175
Section 6.2. Block Cipher Modes of Operation	181
Section 6.3. Stream Ciphers and RC4	189
Section 6.4. Recommended Reading and Web Site	194
Section 6.5. Key Terms, Review Questions, and Problems	194
Chapter 7. Confidentiality Using Symmetric Encryption	199
Section 7.1. Placement of Encryption Function	201
Section 7.2. Traffic Confidentiality	209

Section 7.3. Key Distribution	210
Section 7.4. Random Number Generation	218
Section 7.5. Recommended Reading and Web Sites	227
Section 7.6. Key Terms, Review Questions, and Problems	228
Part Two: Public-Key Encryption and Hash Functions	232
Chapter 8. Introduction to Number Theory	234
Section 8.1. Prime Numbers	236
Section 8.2. Fermat's and Euler's Theorems	238
Section 8.3. Testing for Primality	242
Section 8.4. The Chinese Remainder Theorem	245
Section 8.5. Discrete Logarithms	247
Section 8.6. Recommended Reading and Web Sites	253
Section 8.7. Key Terms, Review Questions, and Problems	254
Chapter 9. Public-Key Cryptography and RSA	257
Section 9.1. Principles of Public-Key Cryptosystems	259
Section 9.2. The RSA Algorithm	268
Section 9.3. Recommended Reading and Web Sites	280
Section 9.4. Key Terms, Review Questions, and Problems	281
Appendix 9A Proof of the RSA Algorithm	285
Appendix 9B The Complexity of Algorithms	286
Chapter 10. Key Management; Other Public-Key Cryptosystems	289
Section 10.1. Key Management	290
Section 10.2. Diffie-Hellman Key Exchange	298
Section 10.3. Elliptic Curve Arithmetic	301
Section 10.4. Elliptic Curve Cryptography	310
Section 10.5. Recommended Reading and Web Sites	313
Section 10.6. Key Terms, Review Questions, and Problems	314
Chapter 11. Message Authentication and Hash Functions	317
Section 11.1. Authentication Requirements	319
Section 11.2. Authentication Functions	320
Section 11.3. Message Authentication Codes	331
Section 11.4. Hash Functions	334
Section 11.5. Security of Hash Functions and Macs	340
Section 11.6. Recommended Reading	344
Section 11.7. Key Terms, Review Questions, and Problems	344
Appendix 11A Mathematical Basis of the Birthday Attack	346
Chapter 12. Hash and MAC Algorithms	351
Section 12.1. Secure Hash Algorithm	353
Section 12.2. Whirlpool	358

Section 12.3. HMAC	368
Section 12.4. CMAC	372
Section 12.5. Recommended Reading and Web Sites	374
Section 12.6. Key Terms, Review Questions, and Problems	374
Chapter 13. Digital Signatures and Authentication Protocols	377
Section 13.1. Digital Signatures	378
Section 13.2. Authentication Protocols	382
Section 13.3. Digital Signature Standard	390
Section 13.4. Recommended Reading and Web Sites	393
Section 13.5. Key Terms, Review Questions, and Problems	393
Part Three: Network Security Applications	398
Chapter 14. Authentication Applications	400
Section 14.1. Kerberos	401
Section 14.2. X.509 Authentication Service	419
Section 14.3. Public-Key Infrastructure	428
Section 14.4. Recommended Reading and Web Sites	430
Section 14.5. Key Terms, Review Questions, and Problems	431
Appendix 14A Kerberos Encryption Techniques	433
Chapter 15. Electronic Mail Security	436
Section 15.1. Pretty Good Privacy	438
Section 15.2. S/MIME	457
Section 15.3. Key Terms, Review Questions, and Problems	474
Appendix 15A Data Compression Using Zip	475
Appendix 15B Radix-64 Conversion	478
Appendix 15C PGP Random Number Generation	479
Chapter 16. IP Security	483
Section 16.1. IP Security Overview	485
Section 16.2. IP Security Architecture	487
Section 16.3. Authentication Header	493
Section 16.4. Encapsulating Security Payload	498
Section 16.5. Combining Security Associations	503
Section 16.6. Key Management	506
Section 16.7. Recommended Reading and Web Site	516
Section 16.8. Key Terms, Review Questions, and Problems	517
Appendix 16A Internetworking and Internet Protocols	518
Chapter 17. Web Security	527
Section 17.1. Web Security Considerations	528
Section 17.2. Secure Socket Layer and Transport Layer Security	531
Section 17.3. Secure Electronic Transaction	549

Section 17.4. Recommended Reading and Web Sites	560
Section 17.5. Key Terms, Review Questions, and Problems	561
Part Four: System Security	563
Chapter 18. Intruders	565
Section 18.1. Intruders	567
Section 18.2. Intrusion Detection	570
Section 18.3. Password Management	582
Section 18.4. Recommended Reading and Web Sites	591
Section 18.5. Key Terms, Review Questions, and Problems	592
Appendix 18A The Base-Rate Fallacy	594
Chapter 19. Malicious Software	598
Section 19.1. Viruses and Related Threats	599
Section 19.2. Virus Countermeasures	610
Section 19.3. Distributed Denial of Service Attacks	614
Section 19.4. Recommended Reading and Web Sites	619
Section 19.5. Key Terms, Review Questions, and Problems	620
Chapter 20. Firewalls	621
Section 20.1. Firewall Design Principles	622
Section 20.2. Trusted Systems	634
Section 20.3. Common Criteria for Information Technology Security Evaluation	640
Section 20.4. Recommended Reading and Web Sites	644
Section 20.5. Key Terms, Review Questions, and Problems	645
Appendix A. Standards and Standards-Setting Organizations	647
Section A.1. The Importance of Standards	648
Section A.2. Internet Standards and the Internet Society	649
Section A.3. National Institute of Standards and Technology	652
Appendix B. Projects for Teaching Cryptography and Network Security	653
Section B.1. Research Projects	654
Section B.2. Programming Projects	655
Section B.3. Laboratory Exercises	655
Section B.4. Writing Assignments	655
Section B.5. Reading/Report Assignments	656
Glossary	657
References	663
Abbreviations	663
Inside Front Cover	InsideFrontCover
Inside Back Cover	InsideBackCover
Index	

Copyright

[Page ii]

Library of Congress Cataloging-in-Publication Data on File

Vice President and Editorial Director, ECS: *Marcia J. Horton*

Executive Editor: *Tracy Dunkelberger*

Editorial Assistant: *Christianna Lee*

Executive Managing Editor: *Vince O'Brien*

Managing Editor: *Camille Trentacoste*

Production Editor: *Rose Kernan*

Director of Creative Services: *Paul Belfanti*

Cover Designer: *Bruce Kenselaar*

Managing Editor, AV Management and Production: *Patricia Burns*

Art Editor: *Gregory Dulles*

Manufacturing Manager: *Alexis Heydt-Long*

Manufacturing Buyer: *Lisa McDowell*

Marketing Manager: *Robin O'Brien*

Marketing Assistant: *Barrie Reinhold*

© 2006 Pearson Education, Inc.

Pearson Prentice Hall

Pearson Education, Inc.

Upper Saddle River, NJ 07458

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

Pearson Prentice Hall™ is a trademark of Pearson Education, Inc.

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.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Pearson Education Ltd., *London*
Pearson Education Australia Pty. Ltd., *Sydney*
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd., *Hong Kong*
Pearson Education Canada, Inc., *Toronto*
Pearson Educación de México, S.A. de C.V.
Pearson Education Japan, *Tokyo*
Pearson Education Malaysia, Pte. Ltd.
Pearson Education Inc., *Upper Saddle River, New Jersey*

[Page iii]

Dedication

To Antigone never dull never boring always a Sage

◀ PREY

NEXT ▶

Notation

Even the natives have difficulty mastering this peculiar vocabulary.

The Golden Bough, Sir James George Frazer

Symbol	Expression	Meaning
D, K	$D(K, Y)$	Symmetric decryption of ciphertext Y using secret key K .
D, PR_a	$D(PR_a, Y)$	Asymmetric decryption of ciphertext Y using A's private key PR_a
D, PU_a	$D(PU_a, Y)$	Asymmetric decryption of ciphertext Y using A's public key PU_a
E, K	$E(K, X)$	Symmetric encryption of plaintext X using secret key K .
E, PR_a	$E(PR_a, X)$	Asymmetric encryption of plaintext X using A's private key PR_a
E, PU_a	$E(PU_a, X)$	Asymmetric encryption of plaintext X using A's public key PU_a
K		Secret key
PR_a		Private key of user A
PU_a		Public key of user A
C, K	$C(K, X)$	Message authentication code of message X using secret key K .
$GF(p)$		The finite field of order p , where p is prime. The field is defined as the set Z_p together with the arithmetic operations modulo p .
$GF(2^n)$		The finite field of order 2^n .
Z_n		Set of nonnegative integers less than n
gcd	$\text{gcd}(i, j)$	Greatest common divisor; the largest positive integer that divides both i and j with no remainder on division.
mod	$a \text{ mod } m$	Remainder after division of a by m .
mod, \equiv	$a \equiv b \pmod{m}$	$a \text{ mod } m = b \text{ mod } m$
mod, $\not\equiv$	$a \not\equiv b \pmod{m}$	$a \text{ mod } m \neq b \text{ mod } m$
dlog	$\text{dlog}_{a,p}(b)$	Discrete logarithm of the number b for the base $a \pmod{p}$
ϕ	$\phi(n)$	The number of positive integers less than n and relatively prime to n . This is Euler's totient function.

Σ	$\sum_{i=1}^n a_i$	$a_1 + a_2 + \dots + a_n$
Π	$\prod_{i=1}^n a_i$	$a_1 \times a_2 \times \dots \times a_n$
	$i j$	i divides j , which means that there is no remainder when j is divided by i
$ \cdot $	$ a $	Absolute value of a
	$x y$	x concatenated with y
\approx	$x \approx y$	x is approximately equal to y
\oplus	$x \oplus y$	Exclusive-OR of x and y for single-bit variables; Bitwise exclusive-OR of x and y for multiple-bit variables
$\lfloor \rfloor$	$\lfloor x \rfloor$	The largest integer less than or equal to x
\in	$x \in S$	The element x is contained in the set S .
\leftrightarrow	$A \leftrightarrow (a_1, a_2, \dots, a_k)$	The integer A corresponds to the sequence of integers (a_1, a_2, \dots, a_k)

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. It covers the material in IAS2 Security Mechanisms, a core area in the Information Technology body of knowledge; NET4 Security, another core area in the Information Technology body of knowledge; and IT311, Cryptography, an advanced course; these subject areas are part of the Draft ACM/IEEE Computer Society Computing Curricula 2005.

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:

Part One. Conventional Encryption: A detailed examination of conventional encryption algorithms and design principles, including a discussion of the use of conventional encryption for confidentiality.

Part Two. 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.

Part Three. Network Security Practice: Covers important network security tools and applications, including Kerberos, X.509v3 certificates, PGP, S/MIME, IP Security, SSL/TLS, and SET.

Part Four. 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.

In addition, the book includes an extensive glossary, a list of frequently used acronyms, and a bibliography. Each chapter includes homework problems, review questions, a list of key words, suggestions for further reading, and recommended Web sites.

A more detailed, chapter-by-chapter summary of each part appears at the beginning of that part.

Internet Services for Instructors and Students

There is a Web site for this book that provides support for students and instructors. The site includes links to other relevant sites, transparency masters of figures and tables in the book in PDF (Adobe Acrobat) format, and PowerPoint slides. The Web page is at WilliamStallings.com/Crypto/Crypto4e.html. As soon as typos or other errors are discovered, an errata list for this book will be available at WilliamStallings.com. In addition, the Computer Science Student Resource site, at WilliamStallings.com/StudentSupport.html, provides documents, information, and useful links for computer science students and professionals.

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:

[Page xv]

- **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
- **Lab exercises:** A series of projects that involve programming and experimenting with concepts from the book
- **Writing assignments:** A set of suggested writing assignments, by chapter
- **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 B](#) for details.

What's New in the Fourth Edition

In the three years since the third 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 third edition was extensively reviewed by a number of professors who teach the subject. In addition, a number of professionals working in the field reviewed individual chapters. The result is that, in many places, the narrative has been clarified and tightened, and illustrations have been improved. Also, a large number of new "field-tested" problems have been added.

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

- **Simplified AES:** This is an educational, simplified version of AES (Advanced Encryption Standard), which enables students to grasp the essentials of AES more easily.
- **Whirlpool:** This is an important new secure hash algorithm based on the use of a symmetric block cipher.
- **CMAC:** This is a new block cipher mode of operation. CMAC (cipher-based message authentication code) provides message authentication based on the use of a symmetric block cipher.
- **Public-key infrastructure (PKI):** This important topic is treated in this new edition.
- **Distributed denial of service (DDoS) attacks:** DDoS attacks have assumed increasing significance in recent years.
- **Common Criteria for Information Technology Security Evaluation:** The Common Criteria have become the international framework for expressing security requirements and evaluating products and implementations.
- **Online appendices:** Six appendices available at this book's Web site supplement the material in the text.

In addition, much of the other material in the book has been updated and revised.

Acknowledgments

This new edition has benefited from review by a number of people, who gave generously of their time and expertise. The following people reviewed all or a large part of the manuscript: Danny Krizanc (Wesleyan University), Breno de Medeiros (Florida State University), Roger H. Brown (Rensselaer at Hartford), Cristina Nita-Rotarul (Purdue University), and Jimmy McGibney (Waterford Institute of Technology).

Thanks also to the many people who provided detailed technical reviews of a single chapter: Richard Outerbridge, Jorge Nakahara, Jeroen van de Graaf, Philip Moseley, Andre Correa, Brian Bowling, James Muir, Andrew Holt, Décio Luiz Gazzoni Filho, Lucas Ferreira, Dr. Kemal Bicakci, Routo Terada, Anton Stiglic, Valery Pryamikov, and Yongge Wang.

Joan Daemen kindly reviewed the chapter on AES. Vincent Rijmen reviewed the material on Whirlpool. And Edward F. Schaefer reviewed the material on simplified AES.

The following people contributed homework problems for the new edition: Joshua Brandon Holden (Rose-Hulman Institute of Technology), Kris Gaj (George Mason University), and James Muir (University of Waterloo).

Sanjay Rao and Ruben Torres of Purdue developed the laboratory exercises that appear in the instructor's supplement. The following people contributed project assignments that appear in the instructor's supplement: Henning Schulzrinne (Columbia University); Cetin Kaya Koc (Oregon State University); and David Balenson (Trusted Information Systems and George Washington University).

Finally, I would like to thank the many people responsible for the publication of the book, all of whom did their usual excellent job. This includes the staff at Prentice Hall, particularly production manager Rose Kernan; my supplements manager Sarah Parker; and my new editor Tracy Dunkelberger. Also, Patricia M. Daly did the copy editing.

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.

Chapter 0. Reader's Guide

[0.1 Outline of this Book](#)

[0.2 Roadmap](#)

[Subject Matter](#)

[Topic Ordering](#)

[0.3 Internet and Web Resources](#)

[Web Sites for This Book](#)

[Other Web Sites](#)

[USENET Newsgroups](#)

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

This book, with its accompanying Web site, covers a lot of material. Here we give the reader an overview.

0.1. Outline of this Book

Following an introductory chapter, [Chapter 1](#), the book is organized into four parts:

Part One: Symmetric Ciphers: Provides a survey of symmetric encryption, including classical and modern algorithms. The emphasis is on the two most important algorithms, the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES). This part also addresses message authentication and key management.

Part Two: Public-Key Encryption and Hash Functions: Provides a survey of public-key algorithms, including RSA (Rivest-Shamir-Adelman) and elliptic curve. It also covers public-key applications, including digital signatures and key exchange.

Part Three: Network Security Practice: Examines the use of cryptographic algorithms and security protocols to provide security over networks and the Internet. Topics covered include user authentication, e-mail, IP security, and Web security.

Part Four: System Security: Deals with security facilities designed to protect a computer system from security threats, including intruders, viruses, and worms. This part also looks at firewall technology.

Many of the cryptographic algorithms and network security protocols and applications described in this book have been specified as standards. The most important of these are Internet Standards, defined in Internet RFCs (Request for Comments), and Federal Information Processing Standards (FIPS), issued by the National Institute of Standards and Technology (NIST). [Appendix A](#) discusses the standards-making process and lists the standards cited in this book.

0.2. Roadmap

Subject Matter

The material in this book is organized into three broad categories:

Cryptology: This is the study of techniques for ensuring the secrecy and/or authenticity of information. The two main branches of cryptology are **cryptography**, which is the study of the design of such techniques; and **cryptanalysis**, which deals with the defeating such techniques, to recover information, or forging information that will be accepted as authentic.

[Page 3]

Network security: This area covers the use of cryptographic algorithms in network protocols and network applications.

Computer security: In this book, we use this term to refer to the security of computers against intruders (e.g., hackers) and malicious software (e.g., viruses). Typically, the computer to be secured is attached to a network and the bulk of the threats arise from the network.

The first two parts of the book deal with two distinct cryptographic approaches: symmetric cryptographic algorithms and public-key, or asymmetric, cryptographic algorithms. Symmetric algorithms make use of a single shared key shared by two parties. Public-key algorithms make use of two keys: a private key known only to one party, and a public key, available to other parties.

Topic Ordering

This book covers a lot of material. For the instructor or reader who wishes a shorter treatment, there are a number of opportunities.

To thoroughly cover the material in the first two parts, the chapters should be read in sequence. With the exception of the Advanced Encryption Standard (AES), none of the material in [Part One](#) requires any special mathematical background. To understand AES, it is necessary to have some understanding of finite fields. In turn, an understanding of finite fields requires a basic background in prime numbers and modular arithmetic. Accordingly, [Chapter 4](#) covers all of these mathematical preliminaries just prior to their use in [Chapter 5](#) on AES. Thus, if [Chapter 5](#) is skipped, it is safe to skip [Chapter 4](#) as well.

[Chapter 2](#) introduces some concepts that are useful in later chapters of [Part One](#). However, for the reader whose sole interest is contemporary cryptography, this chapter can be quickly skimmed. The two most important symmetric cryptographic algorithms are DES and AES, which are covered in [Chapters 3](#) and [5](#), respectively. [Chapter 6](#) covers two other interesting algorithms, both of which enjoy commercial use. This chapter can be safely skipped if these algorithms are not of interest.

For [Part Two](#), the only additional mathematical background that is needed is in the area of number

theory, which is covered in [Chapter 8](#). The reader who has skipped [Chapters 4](#) and [5](#) should first review the material on [Sections 4.1](#) through [4.3](#).

The two most widely used general-purpose public-key algorithms are RSA and elliptic curve, with RSA enjoying much wider acceptance. The reader may wish to skip the material on elliptic curve cryptography in [Chapter 10](#), at least on a first reading. In [Chapter 12](#), Whirlpool and CMAC are of lesser importance.

[Part Three](#) and [Part Four](#) are relatively independent of each other and can be read in either order. Both parts assume a basic understanding of the material in [Parts One](#) and [Two](#).



0.3. 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 WilliamStallings.com/Crypto/Crypto4e.html. The site includes the following:

- **Useful Web sites:** There are links to other relevant Web sites, organized by chapter, including the sites listed in this section and throughout this book.
- **Errata sheet:** An errata list for this book will be maintained and updated as needed. Please e-mail any errors that you spot to me. Errata sheets for my other books are at WilliamStallings.com.
- **Figures:** All of the figures in this book in PDF (Adobe Acrobat) format.
- **Tables:** All of the tables in this book in PDF format.
- **Slides:** A set of PowerPoint slides, organized by chapter.
- **Cryptography and network security courses:** There are links to home pages for courses based on this book; these pages may be useful to other instructors in providing ideas about how to structure their course.

I also maintain the Computer Science Student Resource Site, at WilliamStallings.com/StudentSupport.html. The purpose of this site is to provide documents, information, and links for computer science students and professionals. Links and documents are organized into four categories:

- **Math:** Includes a basic math refresher, a queuing analysis primer, a number system primer, and links to numerous math sites
- **How-to:** Advice and guidance for solving homework problems, writing technical reports, and preparing technical presentations
- **Research resources:** Links to important collections of papers, technical reports, and bibliographies
- **Miscellaneous:** A variety of other useful documents and links

Other Web Sites

There are numerous Web sites that provide information related to the topics of this book. In subsequent chapters, pointers to specific Web sites can be found in the *Recommended Reading and Web Sites* section. Because the addresses for Web sites tend to change frequently, I have not included URLs 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. Other links not mentioned in this book will be added to the Web site over time.

USENET Newsgroups

A number of USENET newsgroups are devoted to some aspect of cryptography or network security. 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

- **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.
- **sci.crypt:** A general discussion of cryptology and related topics.
- **sci.crypt.random-numbers:** A discussion of cryptographic-strength random number generators.
- **alt.security:** A general discussion of security topics.
- **comp.security.misc:** A general discussion of computer security topics.
- **comp.security.firewalls:** A discussion of firewall products and technology.
- **comp.security.announce:** News, announcements from CERT.
- **comp.risks:** A discussion of risks to the public from computers and users.
- **comp.virus:** A moderated discussion of computer viruses.

◀ PREV

NEXT ▶

Chapter 1. Introduction

[1.1 Security Trends](#)

[1.2 The OSI Security Architecture](#)

[1.3 Security Attacks](#)

[Passive Attacks](#)

[Active Attacks](#)

[1.4 Security Services](#)

[Authentication](#)

[Access Control](#)

[Data Confidentiality](#)

[Data Integrity](#)

[Nonrepudiation](#)

[Availability Service](#)

[1.5 Security Mechanisms](#)

[1.6 A Model for Network Security](#)

[1.7 Recommended Reading and Web Sites](#)

[1.8 Key Terms, Review Questions, and Problems](#)

[Key Terms](#)

[Review Questions](#)

[Problems](#)

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

Key Points

- The OSI (open systems interconnection) security architecture provides a systematic framework for defining security attacks, mechanisms, and services.
- **Security attacks** are classified as either passive attacks, which include unauthorized reading of a message or file and traffic analysis; and active attacks, such as modification of messages or files, and denial of service.
- A **security mechanism** is any process (or a device incorporating such a process) that is designed to detect, prevent, or recover from a security attack. Examples of mechanisms are encryption algorithms, digital signatures, and authentication protocols.
- **Security services** include authentication, access control, data confidentiality, data integrity, nonrepudiation, and availability.

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 network, data network, or the Internet. 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 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,^[1] and the term **internet security** is used.

^[1] We use the term *internet*, with a lowercase "i," to refer to any interconnected collection of networks. 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.

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 or optical disk 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 is 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.

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

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 attacks on those security features. In many cases, successful attacks 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 (Transmission Control Protocol/Internet Protocol) 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 general discussion of network security services and mechanisms and of the types of attacks they are designed for. Then we develop a general overall model within which the security services and mechanisms can be viewed.

[Page 9 (continued)]

1.1. Security Trends

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.

[Page 10]

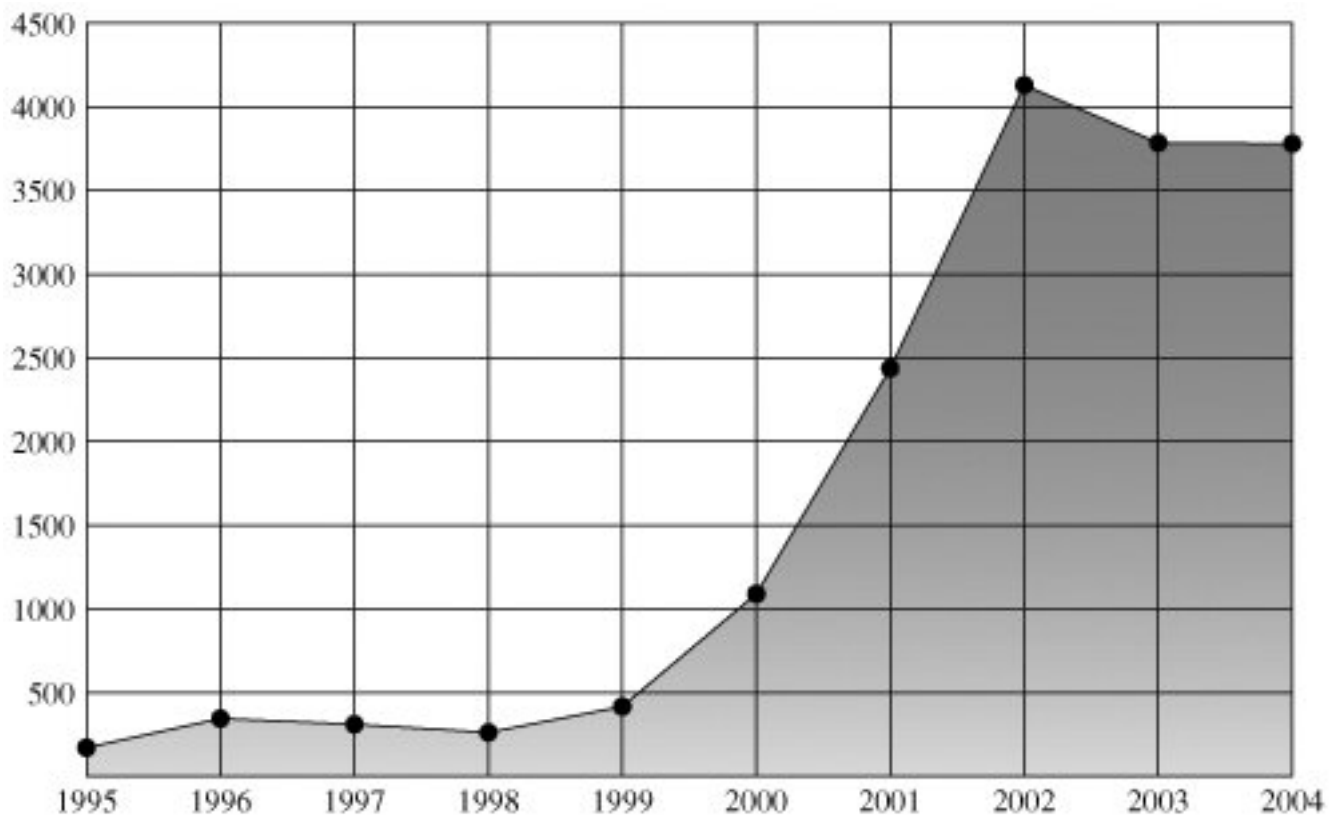
These concerns are fully justified. As confirmation, consider the trends reported by the Computer Emergency Response Team (CERT) Coordination Center (CERT/CC). [Figure 1.1a](#) shows the trend in Internet-related vulnerabilities reported to CERT over a 10-year period. These include security weaknesses in the operating systems of attached computers (e.g., Windows, Linux) as well as vulnerabilities in Internet routers and other network devices. [Figure 1.1b](#) shows the number of security-related incidents reported to CERT. These include denial of service attacks; 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.

[Page 11]

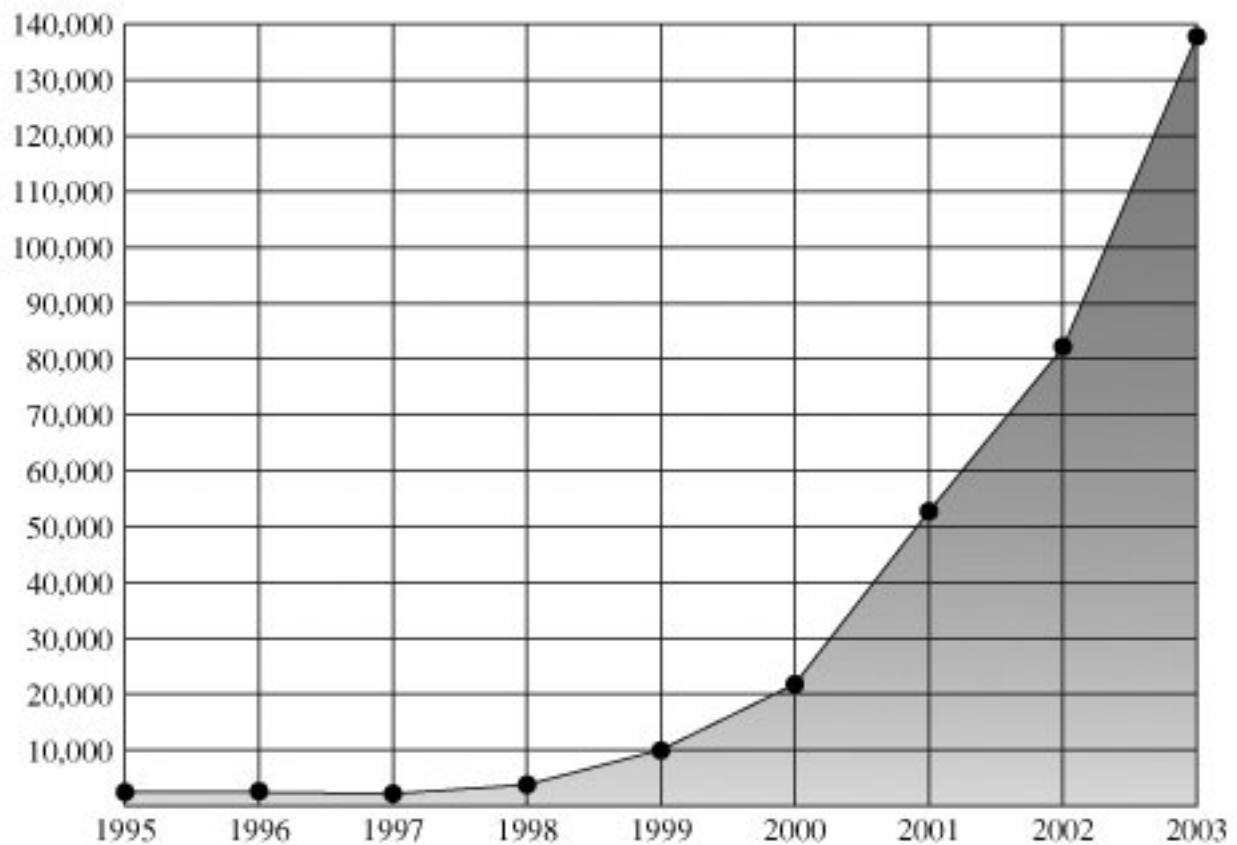
Figure 1.1. CERT Statistics

(This item is displayed on page 10 in the print version)

[\[View full size image\]](#)



(a) Vulnerabilities reported

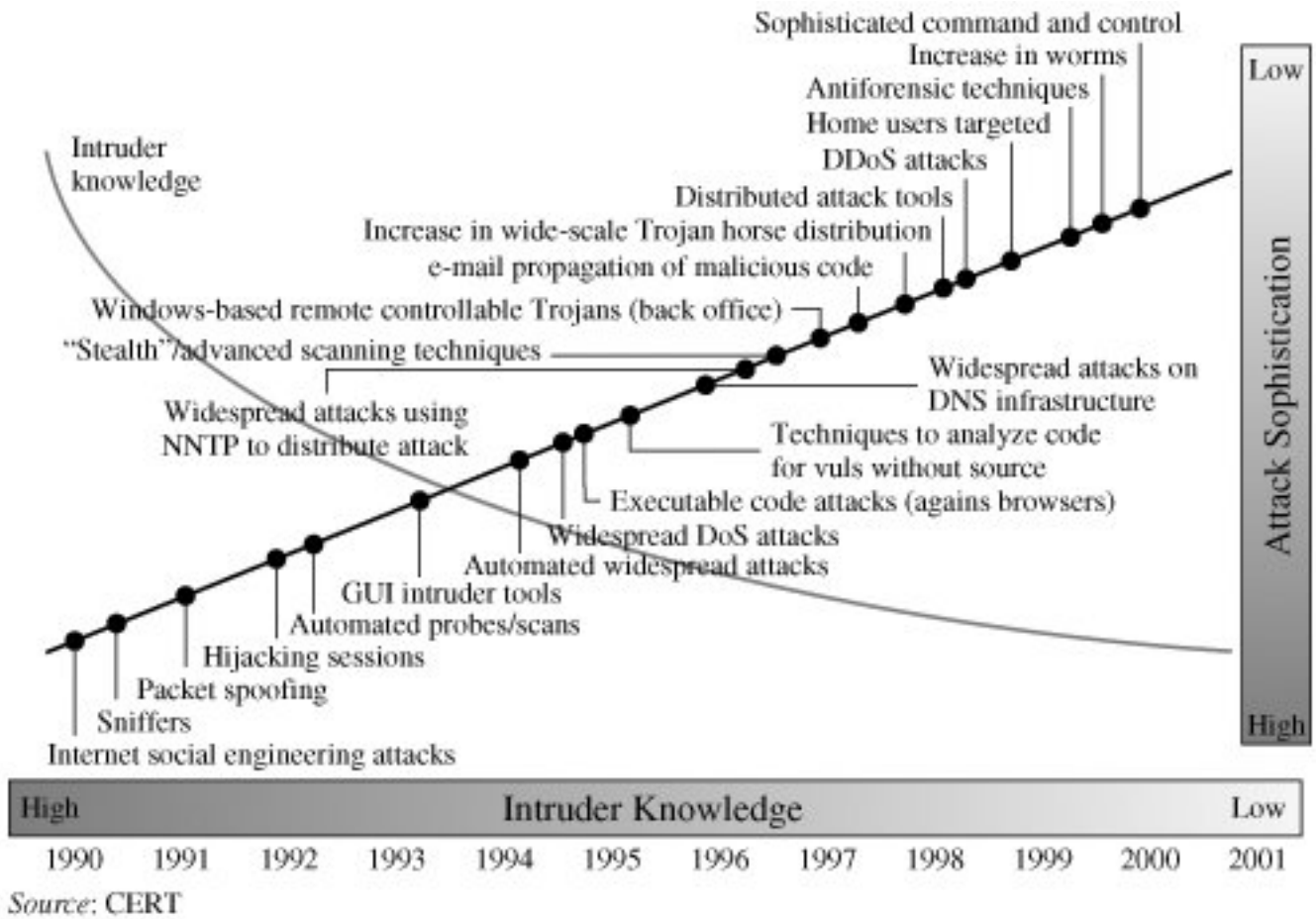


(b) Incidents reported

Over time, the attacks on the Internet and Internet-attached systems have grown more sophisticated while the amount of skill and knowledge required to mount an attack has declined (Figure 1.2). Attacks have become more automated and can cause greater amounts of damage.

Figure 1.2. Trends in Attack Sophistication and Intruder Knowledge

[\[View full size image\]](#)



This increase in attacks coincides with an increased use of the Internet and with increases in the complexity of protocols, applications, and the Internet itself. Critical infrastructures increasingly rely on the Internet for operations. Individual users rely on the security of the Internet, email, the Web, and Web-based applications to a greater extent than ever. Thus, a wide range of technologies and tools are needed to counter the growing threat. At a basic level, cryptographic algorithms for confidentiality and authentication assume greater importance. As well, designers need to focus on Internet-based protocols and the vulnerabilities of attached operating systems and applications. This book surveys all of these technical areas.

1.2. The OSI Security Architecture

To assess effectively the security needs of an organization 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. This is difficult enough in a centralized data processing environment; with the use of local and wide area networks, the problems are compounded.

ITU-T^[2] Recommendation X.800, *Security Architecture for OSI*, defines such a systematic approach.^[3] The OSI security architecture is useful to managers as a way of organizing the task of providing security. Furthermore, because this architecture was developed as an international standard, computer and communications vendors have developed security features for their products and services that relate to this structured definition of services and mechanisms.

^[2] The International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T) is a United Nations-sponsored agency that develops standards, called Recommendations, relating to telecommunications and to open systems interconnection (OSI).

^[3] The OSI security architecture was developed in the context of the OSI protocol architecture, which is described in Appendix H. However, for our purposes in this chapter, an understanding of the OSI protocol architecture is not required.

For our purposes, the OSI security architecture provides a useful, if abstract, overview of many of the concepts that this book deals with. The OSI security architecture focuses on security attacks, mechanisms, and services. These can be defined briefly as follows:

- **Security attack:** Any action that compromises the security of information owned by an organization.
- **Security mechanism:** A process (or a device incorporating such a process) that is designed to detect, prevent, or recover from a security attack.
- **Security service:** A processing or communication 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.

In the literature, the terms *threat* and *attack* are commonly used to mean more or less the same thing. [Table 1.1](#) provides definitions taken from RFC 2828, *Internet Security Glossary*.

Table 1.1. Threats and Attacks (RFC 2828)

Threat
A potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm. That is, a threat is a possible danger that might exploit a vulnerability.
Attack

An assault on system security that derives from an intelligent threat; that is, an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system.



1.3. Security Attacks

A useful means of classifying security attacks, used both in X.800 and RFC 2828, is in terms of *passive attacks* and *active attacks*. A passive attack attempts to learn or make use of information from the system but does not affect system resources. An active attack attempts to alter system resources or affect their operation.

Passive Attacks

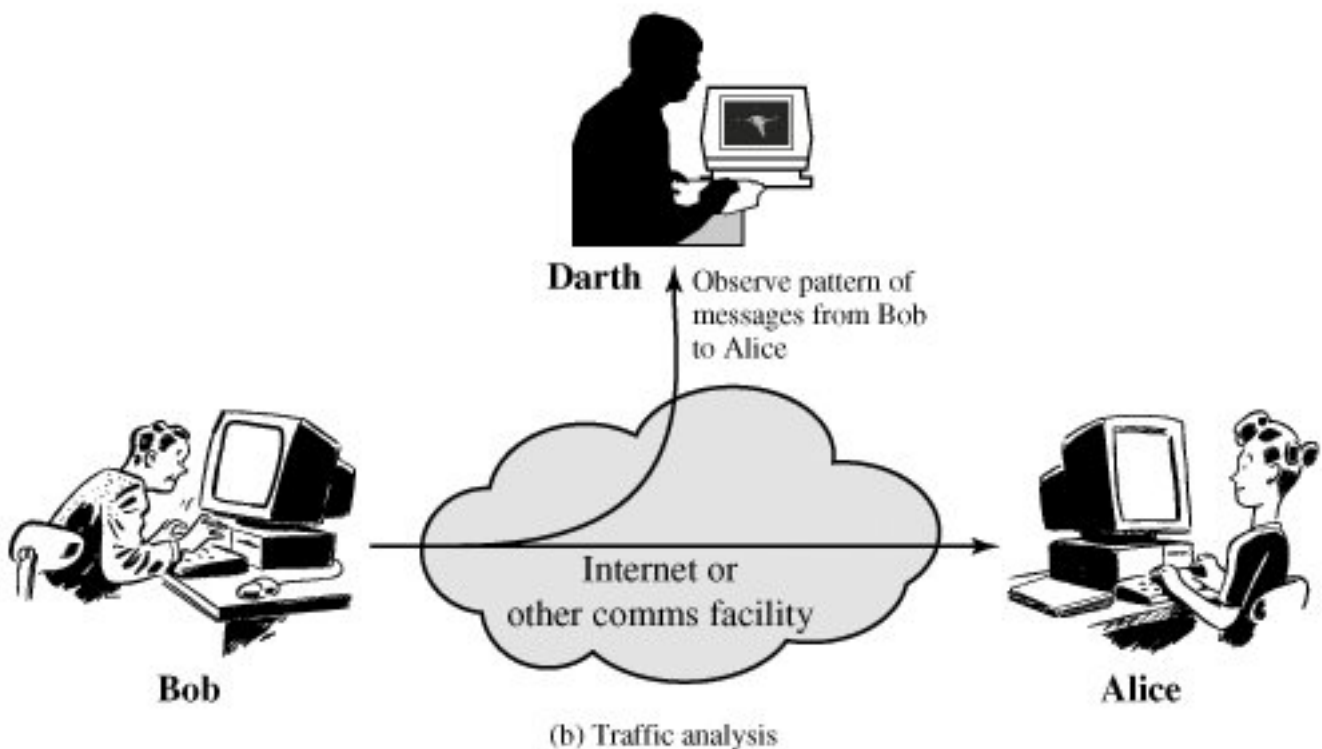
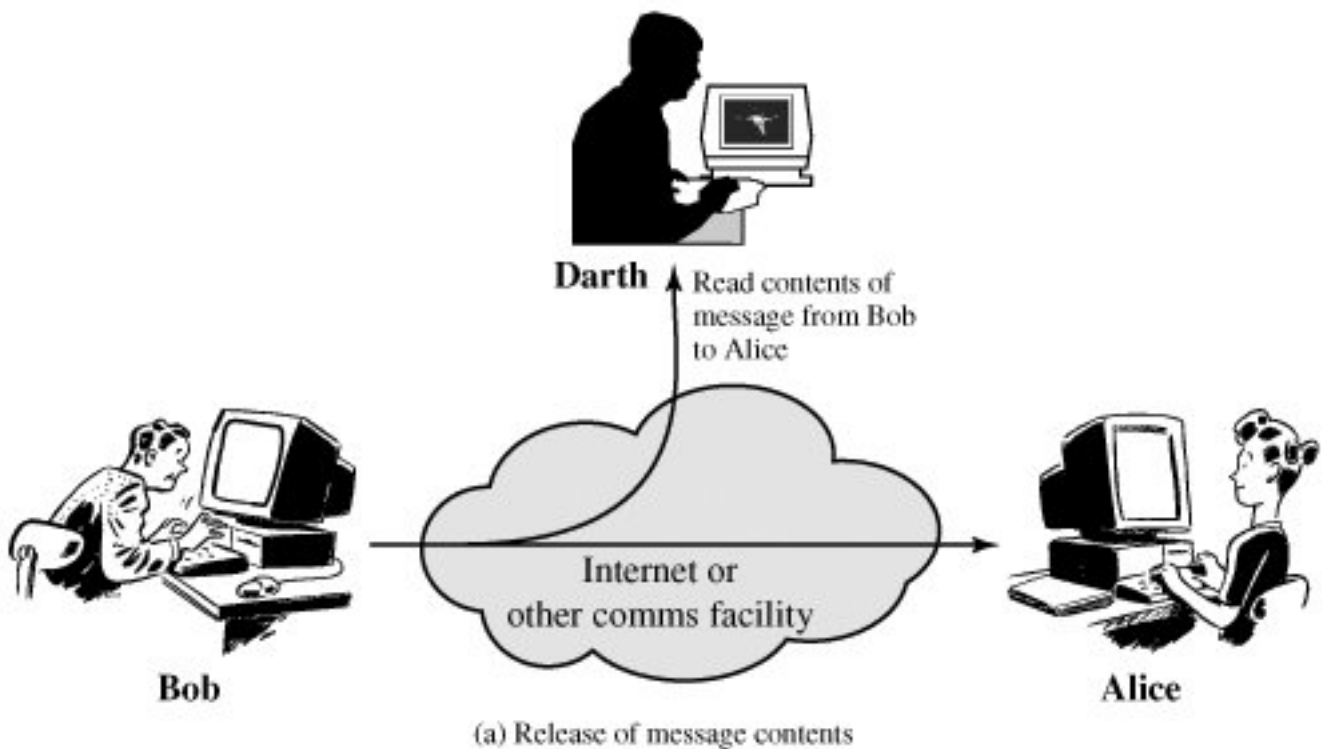
Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the opponent is to obtain information that is being transmitted. Two types of passive attacks are release of message contents and traffic analysis.

The **release of message contents** is easily understood ([Figure 1.3a](#)). A telephone conversation, an electronic mail message, and a transferred file may contain sensitive or confidential information. We would like to prevent an opponent from learning the contents of these transmissions.

Figure 1.3. Passive Attacks

(This item is displayed on page 14 in the print version)

[\[View full size image\]](#)



A second type of passive attack, **traffic analysis**, is subtler ([Figure 1.3b](#)). 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. Typically, the message traffic is sent and received in an apparently normal fashion and neither the sender nor receiver is aware that a third party has read the messages or observed the traffic pattern. However, it is feasible to prevent the success of these attacks, usually by means of encryption. Thus,

the emphasis in dealing with passive attacks is on prevention rather than detection.

Active Attacks

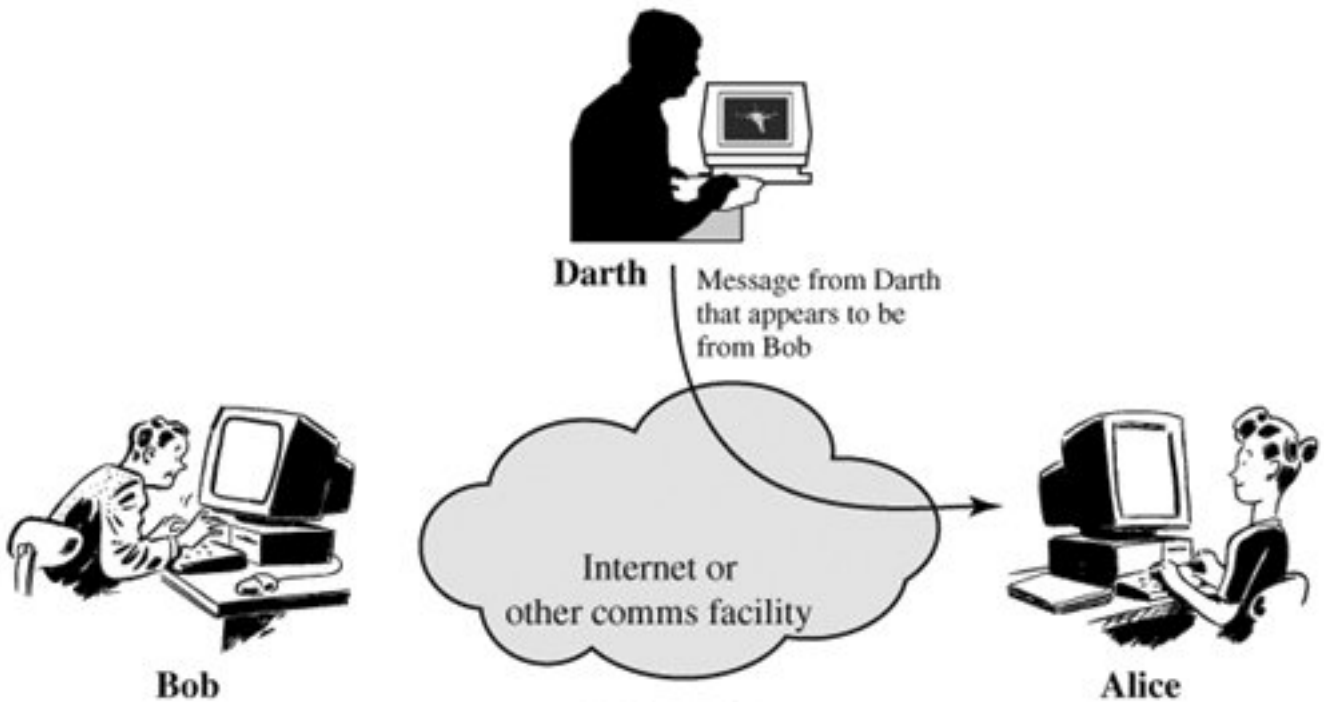
Active attacks involve some modification of the data stream or the creation of a false stream and can be subdivided into four categories: masquerade, replay, modification of messages, and denial of service.

A **masquerade** takes place when one entity pretends to be a different entity (Figure 1.4a). 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.

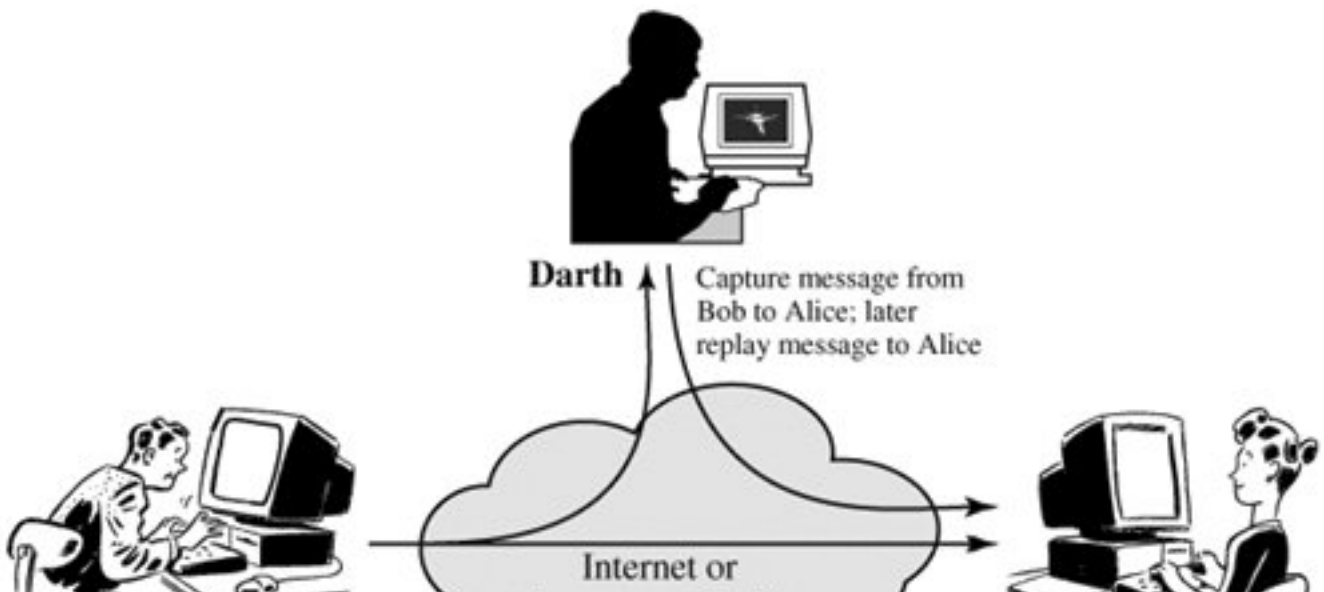
Figure 1.4. Active Attacks

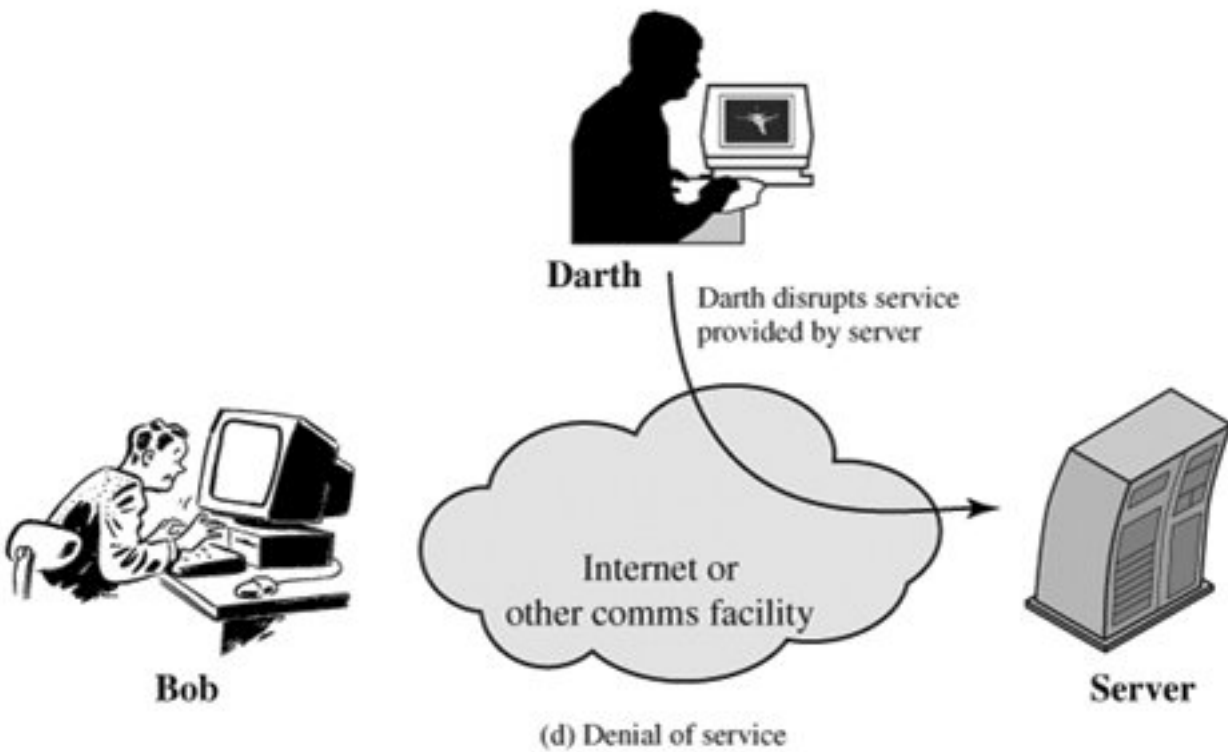
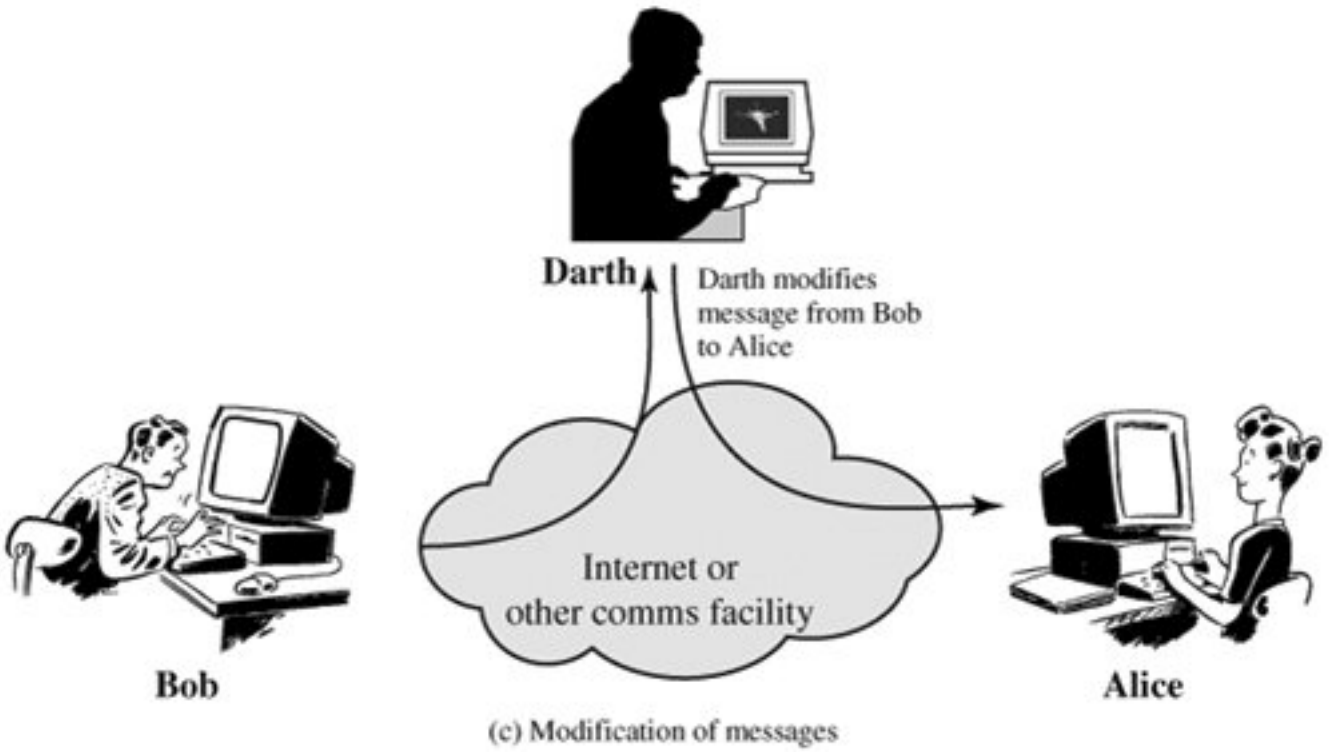
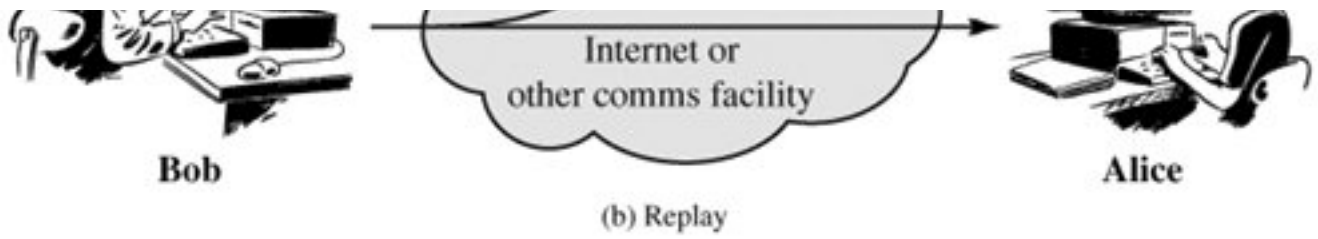
(This item is displayed on pages 15 - 16 in the print version)

[\[View full size image\]](#)



(a) Masquerade





Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect (Figure 1.4b).

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 ([Figure 1.4c](#)). 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 ([Figure 1.4d](#)). 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.

[Page 15]

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 of the wide variety of potential physical, software, and network vulnerabilities. Instead, the goal is to detect active attacks and to recover from any disruption or delays caused by them. If the detection has a deterrent effect, it may also contribute to prevention.



1.4. Security Services

X.800 defines a security service as a service provided by a protocol layer of communicating open systems, which ensures adequate security of the systems or of data transfers. Perhaps a clearer definition is found in RFC 2828, which provides the following definition: a processing or communication service that is provided by a system to give a specific kind of protection to system resources; security services implement security policies and are implemented by security mechanisms.

X.800 divides these services into five categories and fourteen specific services ([Table 1.2](#)). We look at each category in turn.^[4]

^[4] 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 this chapter. Our usage here agrees with both X.800 and RFC 2828.

Table 1.2. Security Services (X.800)

AUTHENTICATION
The assurance that the communicating entity is the one that it claims to be.
Peer Entity Authentication
Used in association with a logical connection to provide confidence in the identity of the entities connected.
Data Origin Authentication
In a connectionless transfer, provides assurance that the source of received data is as claimed.
ACCESS CONTROL
The prevention of unauthorized use of a resource (i.e., this service controls who can have access to a resource, under what conditions access can occur, and what those accessing the resource are allowed to do).
DATA CONFIDENTIALITY
The protection of data from unauthorized disclosure.

Connection Confidentiality

The protection of all user data on a connection.

Connectionless Confidentiality

The protection of all user data in a single data block

Selective-Field Confidentiality

The confidentiality of selected fields within the user data on a connection or in a single data block.

Traffic Flow Confidentiality

The protection of the information that might be derived from observation of traffic flows.

DATA INTEGRITY

The assurance that data received are exactly as sent by an authorized entity (i.e., contain no modification, insertion, deletion, or replay).

Connection Integrity with Recovery

Provides for the integrity of all user data on a connection and detects any modification, insertion, deletion, or replay of any data within an entire data sequence, with recovery attempted.

Connection Integrity without Recovery

As above, but provides only detection without recovery.

Selective-Field Connection Integrity

Provides for the integrity of selected fields within the user data of a data block transferred over a connection and takes the form of determination of whether the selected fields have been modified, inserted, deleted, or replayed.

Connectionless Integrity

Provides for the integrity of a single connectionless data block and may take the form of detection of data modification. Additionally, a limited form of replay detection may be provided.

Selective-Field Connectionless Integrity

Provides for the integrity of selected fields within a single connectionless data block; takes the form of determination of whether the selected fields have been modified.

NONREPUDIATION

Provides protection against denial by one of the entities involved in a communication of having participated in all or part of the communication.

Nonrepudiation, Origin

Proof that the message was sent by the specified party.

Nonrepudiation, Destination

Proof that the message was received by the specified party.

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.

Two specific authentication services are defined in X.800:

- **Peer entity authentication:** Provides for the corroboration of the identity of a peer entity in an association. It is provided for use at the establishment of, or at times during the data transfer phase of, a connection. It attempts to provide confidence that an entity is not performing either a masquerade or an unauthorized replay of a previous connection.
- **Data origin authentication:** Provides for the corroboration of the source of a data unit. It does not provide protection against the duplication or modification of data units. This type of service supports applications like electronic mail where there are no prior interactions between the communicating entities.

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, each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual.

Data Confidentiality

Confidentiality is the protection of transmitted data from passive attacks. With respect to the content of a data transmission, 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, when a TCP connection is set up between two systems, this broad protection prevents the release of any user data transmitted over the TCP connection. 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.

Data 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 without regard to any larger context, generally provides protection against message modification only.

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 alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message.

Availability Service

Both X.800 and RFC 2828 define availability to be the property of a system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system (i.e., a system is available if it provides services according to the system design whenever users request them). 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.

X.800 treats availability as a property to be associated with various security services. However, it makes sense to call out specifically an availability service. An availability service is one that protects a system to ensure its availability. This service addresses the security concerns raised by denial-of-service attacks. It depends on proper management and control of system resources and thus depends on access control service and other security services.

1.5. Security Mechanisms

Table 1.3 lists the security mechanisms defined in X.800. As can be seen the mechanisms are divided into those that are implemented in a specific protocol layer and those that are not specific to any particular protocol layer or security service. These mechanisms will be covered in the appropriate places in the book and so we do not elaborate now, except to comment on the definition of encipherment. X.800 distinguishes between reversible encipherment mechanisms and irreversible encipherment mechanisms. A reversible encipherment mechanism is simply an encryption algorithm that allows data to be encrypted and subsequently decrypted. Irreversible encipherment mechanisms include hash algorithms and message authentication codes, which are used in digital signature and message authentication applications.

[Page 20]

Table 1.3. Security Mechanisms (X.800)

SPECIFIC SECURITY MECHANISMS	
	May be incorporated into the appropriate protocol layer in order to provide some of the OSI security services.
Encipherment	
	The use of mathematical algorithms to transform data into a form that is not readily intelligible. The transformation and subsequent recovery of the data depend on an algorithm and zero or more encryption keys.
Digital Signature	
	Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery (e.g., by the recipient).
Access Control	
	A variety of mechanisms that enforce access rights to resources.
Data Integrity	
	A variety of mechanisms used to assure the integrity of a data unit or stream of data units.
Authentication Exchange	
	A mechanism intended to ensure the identity of an entity by means of information exchange.
Traffic Padding	
	The insertion of bits into gaps in a data stream to frustrate traffic analysis attempts.
Routing Control	
	Enables selection of particular physically secure routes for certain data and allows routing changes, especially when a breach of security is suspected.
Notarization	
	The use of a trusted third party to assure certain properties of a data exchange.
PERVASIVE SECURITY MECHANISMS	
	Mechanisms that are not specific to any particular OSI security service or protocol layer.
Trusted Functionality	
	That which is perceived to be correct with respect to some criteria (e.g., as established by a security policy).

Security Label
The marking bound to a resource (which may be a data unit) that names or designates the security attributes of that resource.
Event Detection
Detection of security-relevant events.
Security Audit Trail
Data collected and potentially used to facilitate a security audit, which is an independent review and examination of system records and activities.
Security Recovery
Deals with requests from mechanisms, such as event handling and management functions, and takes recovery actions.

Table 1.4, based on one in X.800, indicates the relationship between security services and security mechanisms.

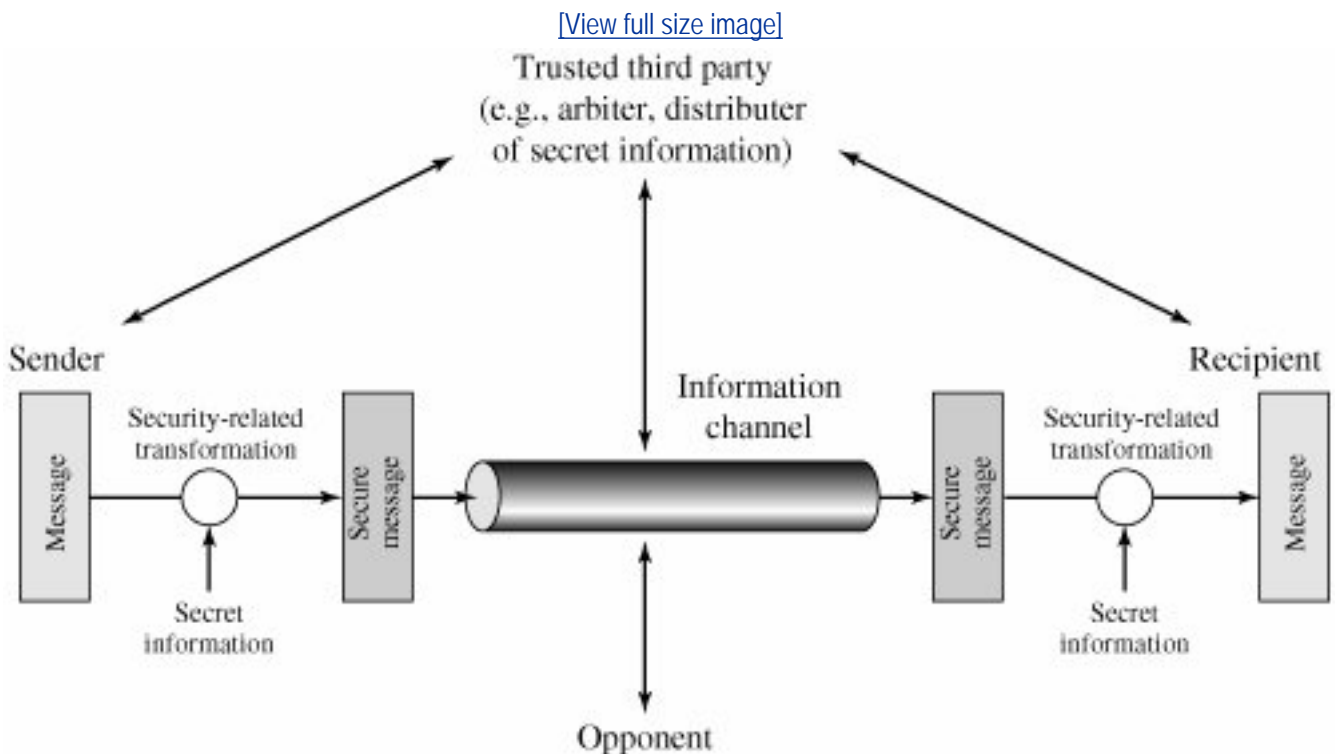
Table 1.4. Relationship between Security Services and Mechanisms

Service	Mechanism							
	Encipherment	Digital Signature	Access Control	Data Integrity	Authentication Exchange	Traffic Padding	Routing Control	Notarization
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Nonrepudiation		Y		Y				Y
Availability				Y	Y			

1.6. A Model for Network Security

A model for much of what we will be discussing is captured, in very general terms, in [Figure 1.5](#). 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.

Figure 1.5. Model for Network Security



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:

- 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. ^[5]

^[5] Part Two discusses a form of encryption, known as public-key encryption, in which only one of the two principals needs to have the secret information.

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.

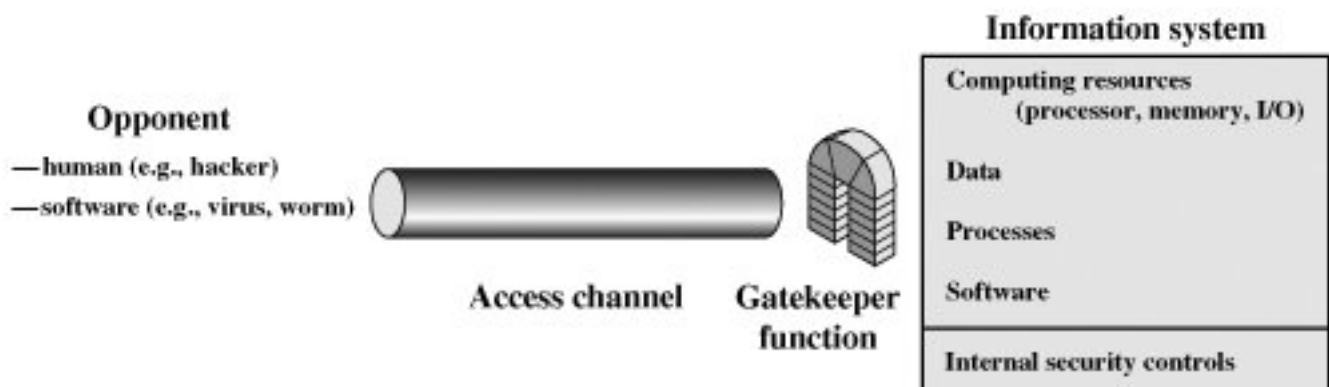
4.

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.

[Parts One](#) through [Three](#) of this book concentrates on the types of security mechanisms and services that fit into the model shown in [Figure 1.5](#). 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 1.6](#), 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).

Figure 1.6. Network Access Security Model

[\[View full size image\]](#)



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. Programs can present two kinds of threats:

- **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.

[Page 24]

Viruses and worms are two examples of software attacks. Such attacks can be introduced into a system by means of a disk 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 1.6](#)). 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 either an unwanted user or unwanted software gains access, 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. These issues are explored in [Part Four](#).



1.7. Recommended Reading and Web Sites

[PFLE02] provides a good introduction to both computer and network security. Two other excellent surveys are [PIEP03] and [BISH05]. [BISH03] covers much the same ground as [BISH05] but with more mathematical detail and rigor. [SCHN00] is valuable reading for any practitioner in the field of computer or network security: it discusses the limitations of technology, and cryptography in particular, in providing security, and the need to consider the hardware, the software implementation, the networks, and the people involved in providing and attacking security.

BISH03 Bishop, M. *Computer Security: Art and Science*. Boston: Addison-Wesley, 2003.

BISH05 Bishop, M. *Introduction to Computer Security*. Boston: Addison-Wesley, 2005.

PFLE02 Pfleeger, C. *Security in Computing*. Upper Saddle River, NJ: Prentice Hall, 2002.

PIEP03 Pieprzyk, J.; Hardjono, T.; and Seberry, J. *Fundamentals of Computer Security*. New York: Springer-Verlag, 2003.

SCHN00 Schneier, B. *Secrets and Lies: Digital Security in a Networked World*. New York: Wiley 2000.

Recommended Web Sites



The following Web sites^[6] are of general interest related to cryptography and network security:

^[6] Because URLs sometimes change, they are not included. For all of the Web sites listed in this and subsequent chapters, the appropriate link is at this book's Web site at williamstallings.com/Crypto/Crypto4e.html.

- **COAST:** Comprehensive set of links related to cryptography and network security.
- **IETF Security Area:** Material related to 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.

- **Helgar Lipma's Cryptology Pointers:** Another 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.
- **Computer Security Resource Center:** Maintained by the National Institute of Standards and Technology (NIST); contains a broad range of information on security threats, technology, and standards.
- **Security Focus:** A wide variety of security information, with an emphasis on vendor products and end-user concerns.
- **SANS Institute:** Similar to Security Focus. Extensive collection of white papers.

◀ PREV

NEXT ▶

1.8. Key Terms, Review Questions, and Problems

Key Terms

[access control](#)

active threat

[authentication](#)

[authenticity](#)

[availability](#)

[data confidentiality](#)

[data integrity](#)

[denial of service](#)

[encryption](#)

[integrity](#)

[intruder](#)

[masquerade](#)

[nonrepudiation](#)

[OSI security architecture](#)

passive threat

[replay](#)

[security attacks](#)

[security mechanisms](#)

[security services](#)

Review Questions

- 1.1 What is the OSI security architecture?
- 1.2 What is the difference between passive and active security threats?
- 1.3 List and briefly define categories of passive and active security attacks.
- 1.4 List and briefly define categories of security services.
- 1.5 List and briefly define categories of security mechanisms.

Problems

- 1.1 Draw a matrix similar to [Table 1.4](#) that shows the relationship between security services and attacks.
- 1.2 Draw a matrix similar to [Table 1.4](#) that shows the relationship between security mechanisms and attacks.

Part One: Symmetric Ciphers

Cryptography is probably the most important aspect of communications security and is becoming increasingly important as a basic building block for computer security.

Computers at Risk: Safe Computing in the Information Age, National Research Council, 1991

The increased use of computer and communications systems by industry has increased the risk of theft of proprietary information. Although these threats may require a variety of countermeasures, encryption is a primary method of protecting valuable electronic information.

Communications Privacy: Federal Policy and Actions, General Accounting Office Report GAO/OSI-94-2, November 1993

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. Part One provides a survey of the basic principles of symmetric encryption, looks at widely used algorithms, and discusses applications of symmetric cryptography.

Road Map for Part One

[Chapter 2: Classical Encryption Techniques](#)

[Chapter 2](#) describes classical symmetric encryption techniques. It provides a gentle and interesting introduction to cryptography and cryptanalysis and highlights important concepts.

[Chapter 3: Block Ciphers and the Data Encryption Standard](#)

[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 symmetric encryption schemes.

[Chapter 4: Finite Fields](#)

Finite fields have become increasingly important in cryptography. A number of

cryptographic algorithms rely heavily on properties of finite fields, notably the Advanced Encryption Standard (AES) and elliptic curve cryptography. This chapter is positioned here so that concepts relevant to AES can be introduced prior to the discussion of AES. [Chapter 4](#) provides the necessary background to the understanding of arithmetic over finite fields of the form $GF(2^n)$.

[Chapter 5: Advanced Encryption Standard](#)

The most important development in cryptography in recent years is the adoption of a new symmetric cipher standard, AES. [Chapter 5](#) provides a thorough discussion of this cipher.

[Chapter 6: More on Symmetric Ciphers](#)

[Chapter 6](#) explores additional topics related to symmetric ciphers. The chapter begins by examining multiple encryption and, in particular, triple DES. Next, we look at the concept of block cipher modes of operation, which deal with ways of handling plaintext longer than a single block. Finally, the chapter discusses stream ciphers and describes RC4.

[Chapter 7: Confidentiality Using Symmetric Encryption](#)

Beyond questions dealing with the actual construction of a symmetric encryption algorithm, a number of design issues relate to the use of symmetric encryption to provide confidentiality. [Chapter 7](#) 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.

Chapter 2. Classical Encryption Techniques

2.1 Symmetric Cipher Model

[Cryptography](#)

[Cryptanalysis](#)

2.2 Substitution Techniques

[Caesar Cipher](#)

[Monoalphabetic Ciphers](#)

[Playfair Cipher](#)

[Hill Cipher](#)

[Polyalphabetic Ciphers](#)

[One-Time Pad](#)

2.3 Transposition Techniques

2.4 Rotor Machines

2.5 Steganography

2.6 Recommended Reading and Web Sites

2.7 Key Terms, Review Questions, and Problems

[Key Terms](#)

[Review Questions](#)

[Problems](#)

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

Key Points

- Symmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the same key. It is also known as conventional encryption.
- Symmetric encryption transforms plaintext into ciphertext using a secret key and an encryption algorithm. Using the same key and a decryption algorithm, the plaintext is recovered from the ciphertext.
- The two types of attack on an encryption algorithm are cryptanalysis, based on properties of the encryption algorithm, and brute-force, which involves trying all possible keys.
- Traditional (precomputer) symmetric ciphers use substitution and/or transposition techniques. Substitution techniques map plaintext elements (characters, bits) into ciphertext elements. Transposition techniques systematically transpose the positions of plaintext elements.
- Rotor machines are sophisticated precomputer hardware devices that use substitution techniques.
- Steganography is a technique for hiding a secret message within a larger one in such a way that others cannot discern the presence or contents of the hidden message.

Symmetric encryption, also referred to as conventional encryption or single-key encryption, was the only type of encryption in use prior to the development of public-key encryption in the 1970s. It remains by far the most widely used of the two types of encryption. Part One examines a number of symmetric ciphers. In this chapter, we begin with a look at a general model for the symmetric 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. Finally, we look briefly at a different approach known as steganography. [Chapter 3](#) examines the most widely used symmetric cipher: DES.

Before beginning, we define some terms. An original message is known as the **plaintext**, while the coded message is called the **ciphertext**. The process of converting from plaintext to ciphertext is known as **enciphering** or **encryption**; restoring the plaintext from the ciphertext is **deciphering** or **decryption**. The many schemes used for encryption constitute the area of study known as **cryptography**. Such a scheme is known as a **cryptographic system** or a **cipher**. Techniques used for deciphering a message without any knowledge of the enciphering details fall into the area of **cryptanalysis**. Cryptanalysis is what the layperson calls "breaking the code." The areas of cryptography and cryptanalysis together are called **cryptology**.

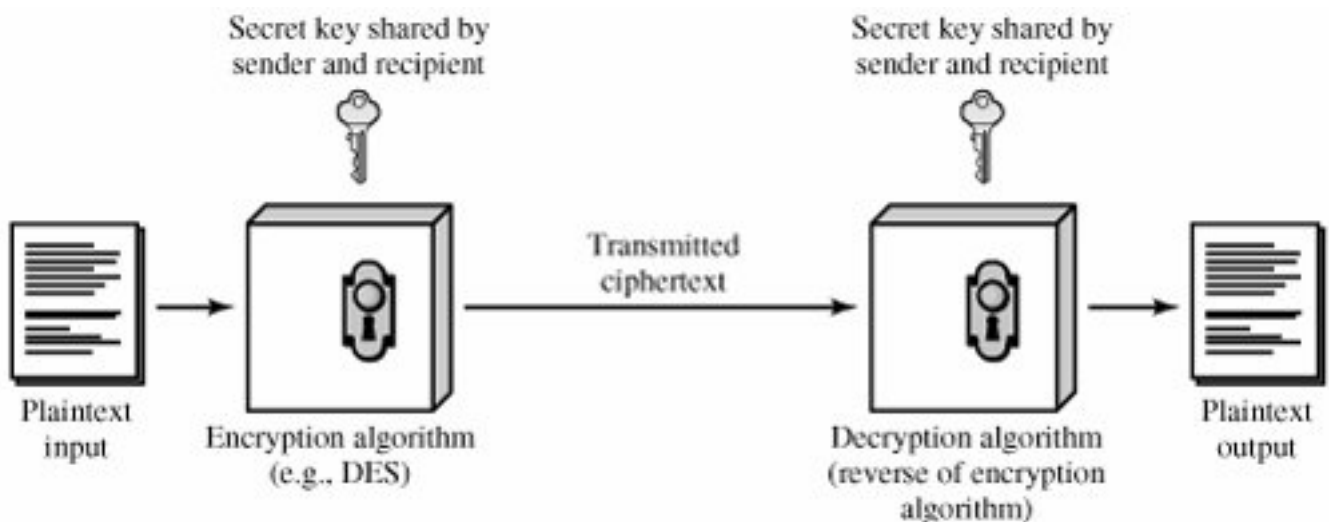
2.1. Symmetric Cipher Model

A symmetric encryption scheme has five ingredients ([Figure 2.1](#)):

- **Plaintext:** This is the original intelligible message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts. The ciphertext is an apparently random stream of data and, as it stands, is unintelligible.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

Figure 2.1. Simplified Model of Conventional Encryption

[\[View full size image\]](#)



There are two requirements for secure use of conventional encryption:

1.

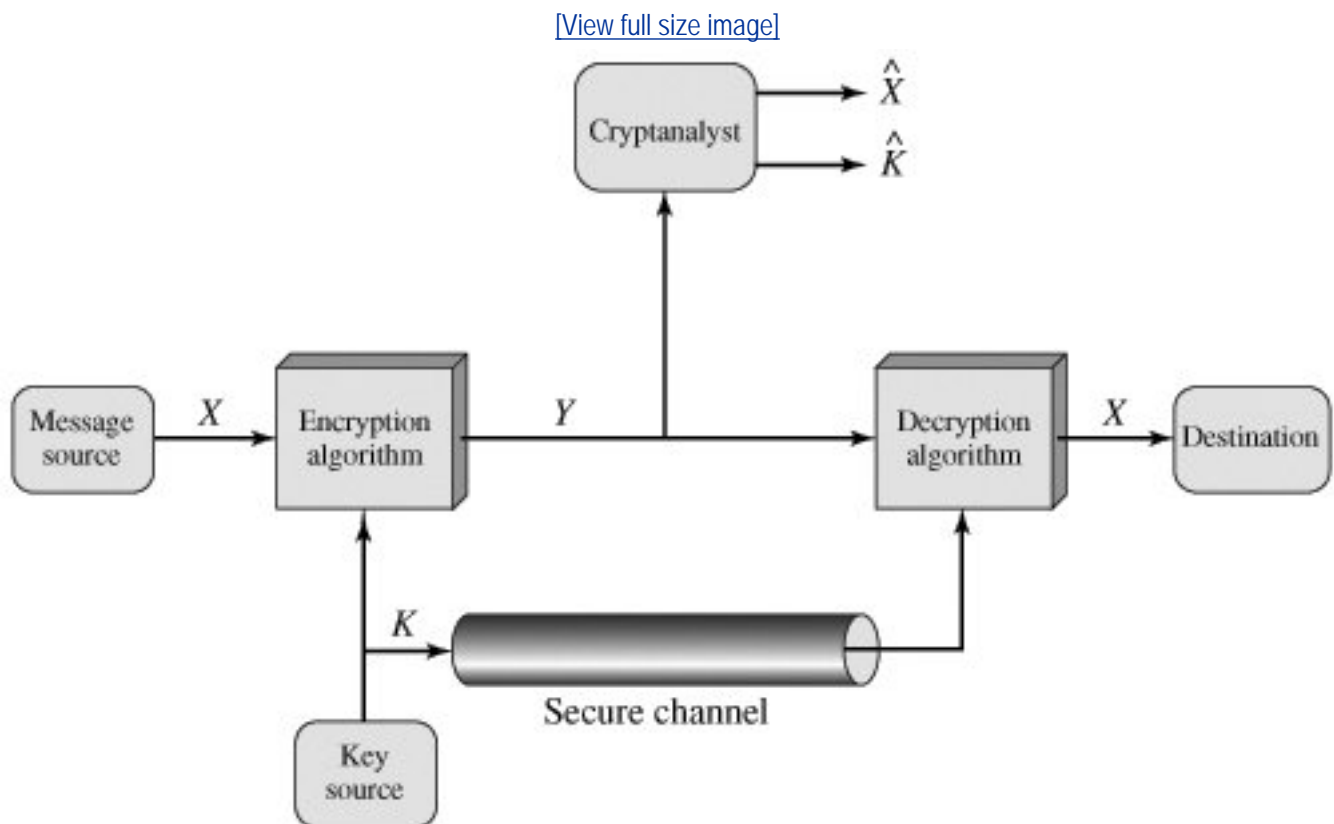
We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.

Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

We assume 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 symmetric 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 symmetric encryption, the principal security problem is maintaining the secrecy of the key.

Let us take a closer look at the essential elements of a symmetric 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 2.2. 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 characterized 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 are 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 use different keys, 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

Typically, the objective of attacking an encryption system is to recover the key in use rather than simply to recover the plaintext of a single ciphertext. There are two general approaches to attacking a conventional encryption scheme:

- **Cryptanalysis:** Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext-ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to

deduce a specific plaintext or to deduce the key being used.

- **Brute-force attack:** The attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

If either type of attack succeeds in deducing the key, the effect is catastrophic: All future and past messages encrypted with that key are compromised.

We first consider cryptanalysis and then discuss brute-force attacks.

[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 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 EXE file, a Java source listing, an accounting file, and so on.

Table 2.1. Types of Attacks on Encrypted Messages

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

Chosen text	<ul style="list-style-type: none"> • Encryption algorithm • Ciphertext • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
-------------	--

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 in [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 either of 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.

All forms of cryptanalysis for symmetric 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 symmetric encryption schemes in this chapter. We will see in [Part Two](#) 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.

A **brute-force attack** 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 four binary key sizes. The 56-bit key size is used with the DES (Data Encryption Standard) algorithm, and the 168-bit key size is used for triple DES. The minimum key size specified for AES (Advanced Encryption Standard) is 128 bits. 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 *ms* 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.

Table 2.2. Average Time Required for Exhaustive Key Search

Key size (bits)	Number of alternative keys		Time required at 1 decryption/ <i>ms</i>		Time required at 10 ⁶ decryption/ <i>ms</i>
32	2 ³²	= 4.3 x 10 ⁹	2 ³¹ <i>ms</i>	= 35.8 minutes	2.15 milliseconds
56	2 ⁵⁶	= 7.2 x 10 ¹⁶	2 ⁵⁵ <i>ms</i>	= 1142 years	10.01 hours
128	2 ¹²⁸	= 3.4 x 10 ³⁸	2 ¹²⁷ <i>ms</i>	= 5.4 x 10 ²⁴ years	5.4 x 10 ¹⁸ years
168	2 ¹⁶⁸	= 3.7 x 10 ⁵⁰	2 ¹⁶⁷ <i>ms</i>	= 5.9 x 10 ³⁶ years	5.9 x 10 ³⁰ years
26 characters (permutation)	26!	= 4 x 10 ²⁶	2 x 10 ²⁶ <i>ms</i>	= 6.4 x 10 ¹² years	6.4 x 10 ⁶ years

2.2. Substitution Techniques

In this section and the next, we examine a sampling of what might be called classical encryption techniques. A study of these techniques enables us to illustrate the basic approaches to symmetric encryption used today and the types of cryptanalytic attacks that must be anticipated.

The two basic building blocks of all encryption techniques are substitution and transposition. We examine these in the next two sections. Finally, we discuss a system that combines both substitution and transposition.

A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols.^[1] If the plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns.

^[1] 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.

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: P H H W P H D I W H U W K H W R J D S D U W B

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

Let us assign a numerical equivalent to each letter:

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Then the algorithm can be expressed as follows. For each plaintext letter p , substitute the ciphertext letter C : ^[2]

^[2] We define $a \bmod n$ to be the remainder when a is divided by n . For example, $11 \bmod 7 = 4$. See [Chapter 4](#) for a further discussion of modular arithmetic.

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

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

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

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

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

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 2.3](#) shows the results of applying this strategy to the example ciphertext. In this case, the plaintext leaps out as occupying the third line.

Figure 2.3. Brute-Force Cryptanalysis of Caesar Cipher

(This item is displayed on page 37 in the print version)

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	hsjlg
12	dvvk	dv	rwkvi	kyv	kfxr	grikp
13	cuuj	cu	qvjuh	jxu	jewq	fghjo
14	btti	bt	puitg	iwt	idvp	epgin
15	assh	as	othsf	hvs	hcuo	dofhm
16	zrrg	zr	nsgre	gur	gbtn	cnegl
17	yqqf	yq	mrfqd	ftq	fasm	bmdfk
18	xppe	xp	lqepc	esp	ezrl	alcej
19	wood	wo	kpdob	dro	dyqk	zkbdi
20	vncn	vn	jocna	cqn	cxpj	yjach
21	ummb	um	inbmz	bpm	bwoi	xizbg
22	tlla	tl	hmaly	aol	avnh	whyaf
23	skkz	sk	glzcx	znk	zumg	vgxze
24	rjyy	rj	fkyjw	ymj	ytlf	ufwyd
25	qiix	qi	ejxiv	xli	xske	tevxc

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 triple DES algorithm, examined in [Chapter 6](#), makes use of a 168-bit key, giving a key space of 2^{168} or greater than 3.7×10^{50} 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 difficult. For example, [Figure 2.4](#) 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.

Figure 2.4. Sample of Compressed Text

```

~+Wµ"- Ω-0)≤4{∞†, ë~Ω%ràu.-í Ø-z-
Ú#2Ò#Åæð æ«q7,Ωn·@3NØÚ Çz'Y-f∞Í[±Û_ èΩ,<NO-±«~xā Åafèù3Å
x)ö§k°Å
_yí ^ΔÉ] ,π J/'iTê&1 'c<uΩ-
ÄD(G WÄC~y_iöÄW PÔ1«ÍÛ†ç},π;~Ï^üÑπ~≈~L^9Ogflo~&æ≤ ~≤ ØÔ§":
^È!SGqèvo^ ú\,S>h<~*6ø†%x'"|ñÓ#≈~my%~zñP<,fi Áj ÅÔ¿~Zù-
Ω"Ö~6Çÿ{%,ΩÊó ,ÿ π+Áî^úO2çSÿ'O-
2Äñßi /@^"ΠK²*PÇπ,úé^'3Σ~ö~ÔZÌ"Y~ÿΩæY> Ω+eô/' <KÉ¿*+~"≤Ù~
B ZøK~Qßÿüf, !òñîzssS/]>ÈQ ü

```

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 4×10^{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. Such an approach is referred to as a **monoalphabetic substitution cipher**, because a single cipher alphabet (mapping from plain alphabet to cipher alphabet) is used per message.

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

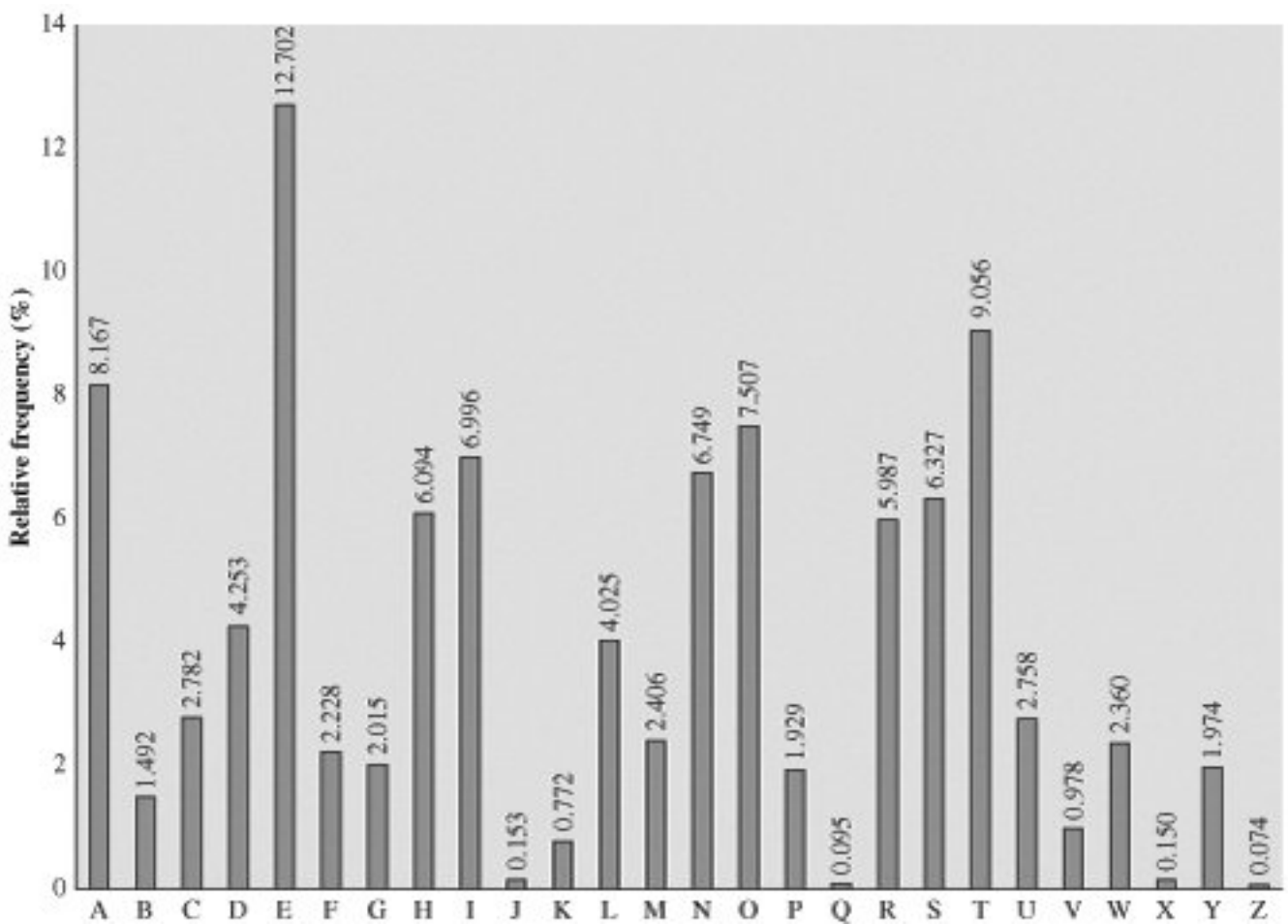
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
 VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
 EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ

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 2.5 (based on [LEWA00]). 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 frequencies 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				

Figure 2.5. Relative Frequency of Letters in English Text

[\[View full size image\]](#)



Comparing this breakdown with [Figure 2.5](#), 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 {a, h, i, n, o, r, s}. The letters with the lowest frequencies (namely, A, B, G, Y, I, J) are likely included in the set {b, j, k, q, v, x, 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 2.5](#) 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 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.

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. If so, 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
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
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.^[3]

^[3] 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.

The Playfair algorithm is based on the use of a 5 x 5 matrix of letters constructed using a keyword. Here is an example, solved by Lord Peter Wimsey in Dorothy Sayers's *Have His Carcase*:^[4]

^[4] The 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 are in the same pair are separated with a filler letter, such as x, so that balloon would be treated as ba lx lo on.
2. Two 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. Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, mu is encrypted as CM.
4. Otherwise, each plaintext letter in a pair 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 2.6](#), 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.^[5] 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 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.

^[5] I am indebted to Gustavus Simmons for providing the plots and explaining their method of construction.

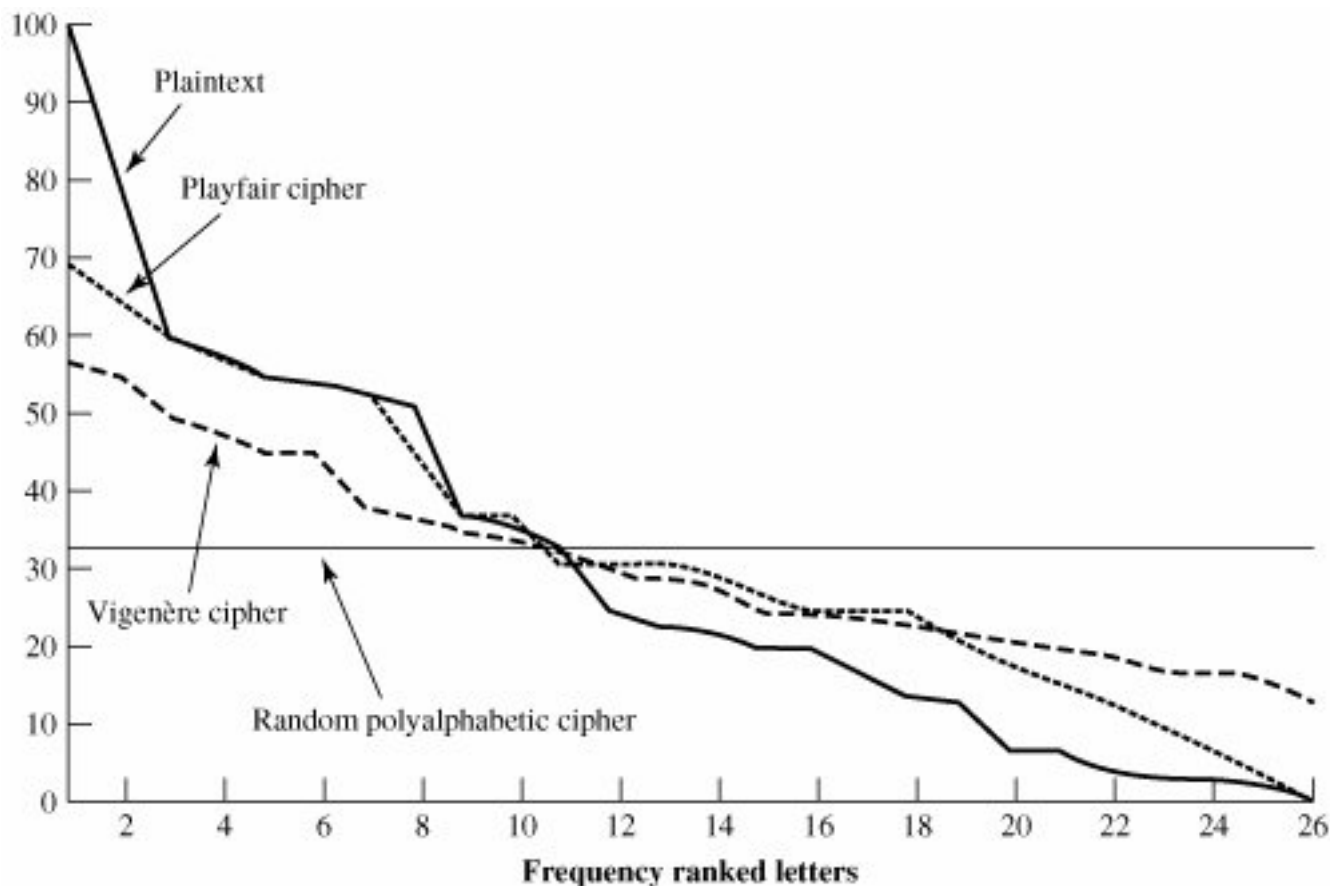


Figure 2.6 also shows the frequency distribution that results when the text is encrypted using the Playfair cipher. To normalize the plot, the number of occurrences of each letter in the ciphertext was again divided by the number of occurrences of e in the plaintext. The resulting plot therefore shows 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^[6]

^[6] This cipher is somewhat more difficult to understand than the others in this chapter, but it illustrates an important point about cryptanalysis that will be useful later on. This subsection can be skipped on a first reading.

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:

$$c_1 = (k_{11}p_1 + k_{12}p_2 + k_{13}p_3) \bmod 26$$

$$c_2 = (k_{21}p_1 + k_{22}p_2 + k_{23}p_3) \text{ mod } 26$$

$$c_3 = (k_{31}p_1 + k_{32}p_2 + k_{33}p_3) \text{ mod } 26$$

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

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \text{ mod } 26$$

or

$$\mathbf{C} = \mathbf{K}\mathbf{P} \text{ mod } 26$$

where \mathbf{C} and \mathbf{P} are column vectors of length 3, representing the plaintext and ciphertext, and \mathbf{K} is a 3 x 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

$$\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix}. \text{ Then } \mathbf{K} \begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix} = \begin{pmatrix} 375 \\ 819 \\ 486 \end{pmatrix} \text{ mod } 26 = \begin{pmatrix} 11 \\ 13 \\ 18 \end{pmatrix} = \text{LNS. Continuing in this fashion,}$$

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:

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

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. [7] 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 2×2 matrix

[7] The basic concepts of linear algebra are summarized in the Math Refresher document at the Computer Science Student Resource site at WilliamStallings.com/StudentSupport.html. The interested reader may consult any text on linear algebra for greater detail.

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

the determinant is $k_{11}k_{22} - k_{12}k_{21}$. For a 3×3 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 $[\mathbf{A}^{-1}]_{ij} = (1)^{i+j}(D_{ij})/\text{det}(\mathbf{A})$, where (D_{ij}) is the subdeterminant formed by deleting the i th row and the j th column of \mathbf{A} and $\text{det}(\mathbf{A})$ 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:

$$\mathbf{C} = \mathbf{E}(\mathbf{K}, \mathbf{P}) = \mathbf{K}\mathbf{P} \text{ mod } 26$$

$$\mathbf{P} = \mathbf{D}(\mathbf{K}, \mathbf{P}) = \mathbf{K}^{-1}\mathbf{C} \text{ mod } 26 = \mathbf{K}^{-1}\mathbf{K}\mathbf{P} = \mathbf{P}$$

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 3×3 Hill cipher hides not only single-letter but also 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

$$\mathbf{P}_j = \begin{pmatrix} p_{1j} \\ p_{2j} \\ \vdots \\ p_{mj} \end{pmatrix} \text{ and } \mathbf{C}_j = \begin{pmatrix} c_{1j} \\ c_{2j} \\ \vdots \\ c_{mj} \end{pmatrix} \text{ such that } \mathbf{C}_j = \mathbf{K}\mathbf{P}_j \text{ for } 1 \leq j \leq m \text{ and for some}$$

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 $\mathbf{Y} = \mathbf{K}\mathbf{X}$. If \mathbf{X} has an inverse, then we can determine $\mathbf{K} = \mathbf{Y}\mathbf{X}^{-1}$. 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 based on one in [STIN02]. Suppose that the plaintext "friday" is encrypted using a 2×2 Hill cipher to yield the ciphertext POCFKU. Thus, we know that

$$\mathbf{K} \begin{pmatrix} 5 \\ 17 \end{pmatrix} \bmod 26 = \begin{pmatrix} 15 \\ 16 \end{pmatrix}; \quad \mathbf{K} \begin{pmatrix} 8 \\ 3 \end{pmatrix} \bmod 26 = \begin{pmatrix} 2 \\ 5 \end{pmatrix}; \quad \text{and} \quad \mathbf{K} \begin{pmatrix} 0 \\ 24 \end{pmatrix} \bmod 26 = \begin{pmatrix} 10 \\ 20 \end{pmatrix}$$

Using the first two plaintext-ciphertext pairs, we have

$$\begin{pmatrix} 15 & 2 \\ 16 & 5 \end{pmatrix} = \mathbf{K} \begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix} \bmod 26$$

[Page 45]

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

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

so

$$\mathbf{K} = \begin{pmatrix} 15 & 2 \\ 16 & 5 \end{pmatrix} \begin{pmatrix} 9 & 2 \\ 1 & 15 \end{pmatrix} = \begin{pmatrix} 137 & 60 \\ 149 & 107 \end{pmatrix} \bmod 26 = \begin{pmatrix} 7 & 8 \\ 19 & 3 \end{pmatrix}$$

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 substitution 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 algorithm 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 2.3). 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.

Table 2.3. The Modern Vigenère Tableau

(This item is displayed on page 46 in the print version)

[\[View full size image\]](#)

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

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:	<i>deceptive</i> <i>deceptive</i> <i>deceptive</i>
plaintext:	<i>wearediscoveredsaveyourself</i>
ciphertext:	<i>ZICVTWQNGRZGVTWAVZHCQYGLMGJ</i>

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 2.6](#) 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.

[Page 47]

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 2.5](#), there should be one cipher letter with a relative frequency of occurrence of about 12.7%, one with about 9.06%, 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. Nevertheless, if the correspondence is close, we can assume a monoalphabetic substitution.

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 2.5](#), can be expected to occur with a frequency of $(0.127)^2 \approx 0.016$, whereas t enciphered by t would occur only about half as often. These regularities can be exploited to achieve successful cryptanalysis. [\[8\]](#)

^[8] 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 remembering when similar claims are made for modern algorithms.

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

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.

One-Time Pad

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 is as long as the message, so that the key need not be repeated. In addition, the key is to be used to encrypt and decrypt a single message, and then is discarded. Each new message requires a new key of the same length as the new message. 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.

An example should illustrate our point. 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. Thus, the tableau of [Table 2.3](#) must be expanded to 27 x 27. Consider the ciphertext

ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS

[Page 49]

We now show two different decryptions using two different keys:

ciphertext: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
key: *p x l m v m s y d o f u y r v z w c t n l e b n e c v g d u p a h f z z l m n y i h*
plaintext: mr mustard with the candlestick in the hall

ciphertext: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
key: *m f u g p m i y d g a x g o u f h k l l l m h s q d q o g t e w b q f g y o v u h w t*
plaintext: miss scarlet with the knife in the library

Suppose that a cryptanalyst had managed to find these two keys. Two plausible plaintexts are produced. How is the cryptanalyst to decide which is the correct decryption (i.e., which is the correct key)? If the actual key were produced in a truly random fashion, then the cryptanalyst cannot say that one of these two keys is more likely than the other. Thus, there is no way to decide which key is correct and therefore which plaintext is correct.

In fact, given any plaintext of equal length to the ciphertext, there is a key that produces that plaintext. Therefore, if you did an exhaustive search of all possible keys, you would end up with many legible plaintexts, with no way of knowing which was the intended plaintext. Therefore, the code is unbreakable.

The security of the one-time pad is entirely due to the randomness of the key. If the stream of characters that constitute the key is truly random, then the stream of characters that constitute the ciphertext will be truly random. Thus, there are no patterns or regularities that a cryptanalyst can use to attack the ciphertext.

In theory, we need look no further for a cipher. The one-time pad offers complete security but, in practice, has two fundamental difficulties:

- 1.

There is the practical problem of making large quantities of random keys. Any heavily used system might require millions of random characters on a regular basis. Supplying truly random characters in this volume is a significant task.

2.

Even more daunting is the problem of key distribution and protection. For every message to be sent, a key of equal length is needed by both sender and receiver. Thus, a mammoth key distribution problem exists.

Because of these difficulties, the one-time pad is of limited utility, and is useful primarily for low-bandwidth channels requiring very high security.



2.3. 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 becomes the key to the algorithm. For example,

```
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
```

[Page 51]

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.



2.4. 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. [9]

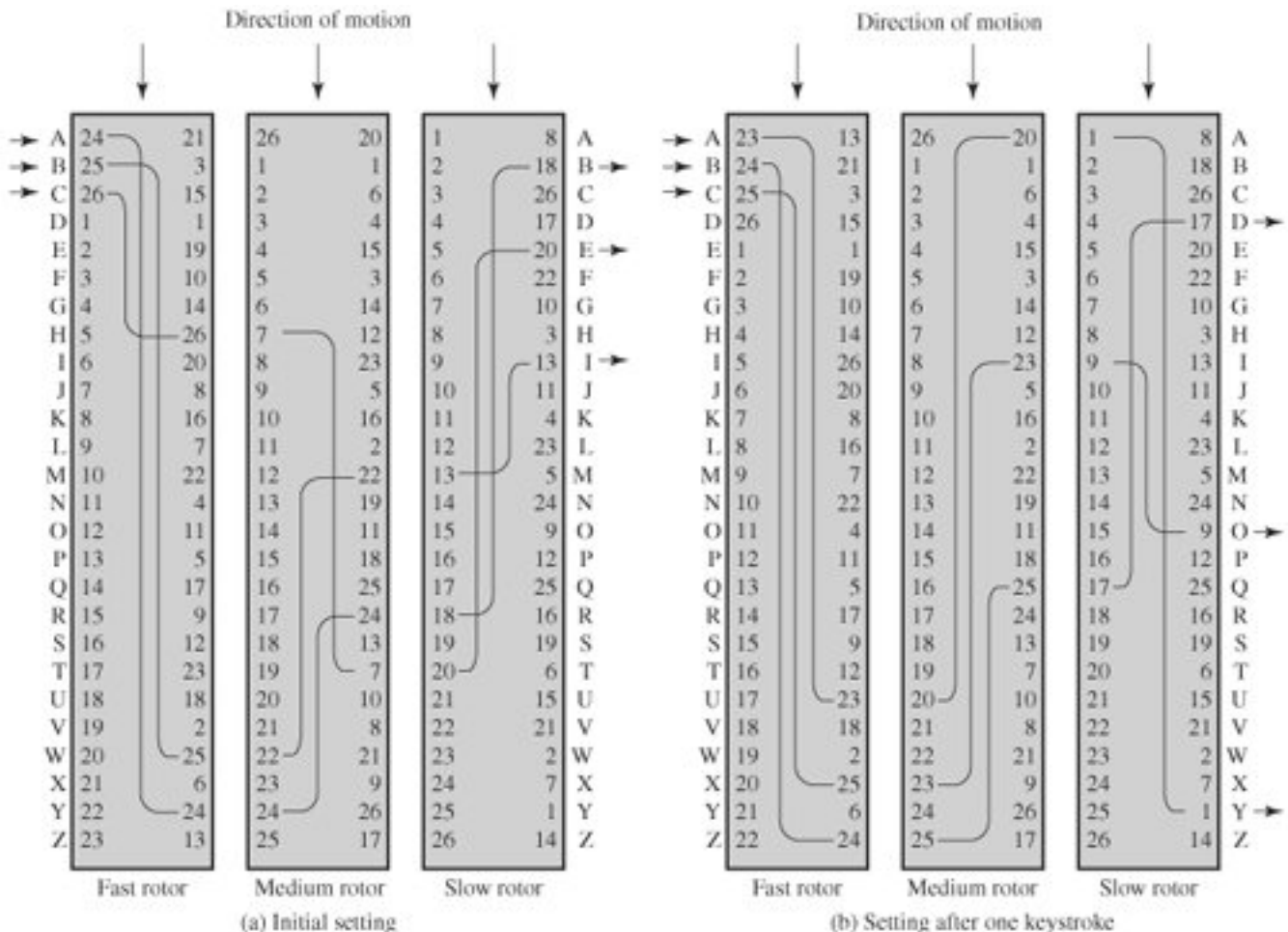
[9] Machines based on the rotor principle were used by both Germany (Enigma) and Japan (Purple) in World War II. The breaking of both codes by the Allies was a significant factor in the war's outcome.

The basic principle of the rotor machine is illustrated in Figure 2.7. 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.

Figure 2.7. Three-Rotor Machine with Wiring Represented by Numbered Contacts

(This item is displayed on page 52 in the print version)

[\[View full size image\]](#)



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 2.7](#), 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 2.7](#) 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 closest to the operator input rotates one pin position with each keystroke. The right half of [Figure 2.7](#) shows the system's configuration after a single keystroke. For every complete rotation of the inner cylinder, the middle cylinder rotates one pin position. Finally, for every complete rotation of the middle cylinder, the outer 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]:

[Page 53]

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](#).

2.5. Steganography

We conclude 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. [\[10\]](#)

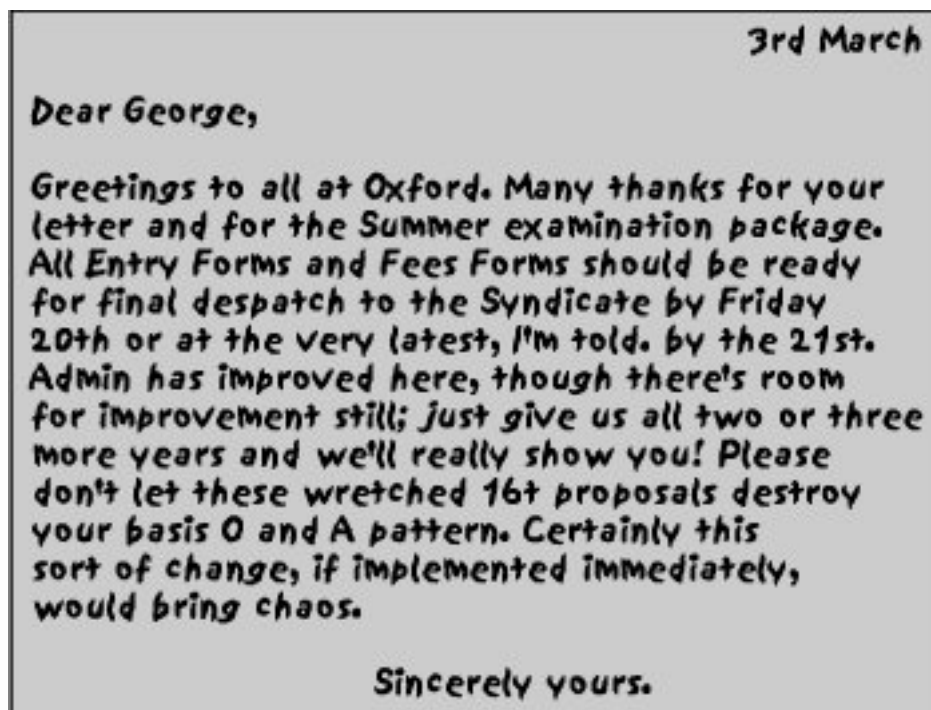
^[10] [Steganography](#) was an obsolete word that was revived by David Kahn and given the meaning it has today [\[KAHN96\]](#).

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 2.8](#) shows an example in which a subset of the words of the overall message is used to convey the hidden message.

Figure 2.8. A Puzzle for Inspector Morse

(This item is displayed on page 54 in the print version)

(From *The Silent World of Nicholas Quinn*, by Colin Dexter)



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

- **Character marking:** Selected letters of printed or typewritten text are overwritten 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.
-

[Page 54]

- **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.

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 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 2.11](#)). 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.

◀ PREV

NEXT ▶

2.6. Recommended Reading and Web Sites

For anyone interested in the history of code making and code breaking, the book to read is [\[KAHN96\]](#). Although it is concerned more with the impact of cryptology than its technical development, it is an excellent introduction and makes for exciting reading. Another excellent historical account is [\[SING99\]](#).

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. Two cryptography books that contain a fair amount of technical material on classical techniques are [\[GARRO1\]](#) and [\[NICH99\]](#). For the truly interested reader, the two-volume [\[NICH96\]](#) covers numerous classical ciphers in detail and provides many ciphertexts to be cryptanalyzed, together with the solutions.

An excellent treatment of rotor machines, including a discussion of their cryptanalysis is found in [\[KUMA97\]](#).

[\[KATZ00\]](#) provides a thorough treatment of steganography. Another good source is [\[WAYN96\]](#).

[GARD72](#) Gardner, M. *Codes, Ciphers, and Secret Writing*. New York: Dover, 1972.

[GARRO1](#) Garrett, P. *Making, Breaking Codes: An Introduction to Cryptology*. Upper Saddle River, NJ: Prentice Hall, 2001.

[KAHN96](#) Kahn, D. *The Codebreakers: The Story of Secret Writing*. New York: Scribner, 1996.

[KATZ00](#) Katzenbeisser, S., ed. *Information Hiding Techniques for Steganography and Digital Watermarking*. Boston: Artech House, 2000.

[KORN96](#) Korner, T. *The Pleasures of Counting*. Cambridge, England: Cambridge University Press, 1996.

[KUMA97](#) Kumar, I. *Cryptology*. Laguna Hills, CA: Aegean Park Press, 1997.

[NICH96](#) Nichols, R. *Classical Cryptography Course*. Laguna Hills, CA: Aegean Park Press, 1996.

[NICH99](#) Nichols, R. ed. *ICSA Guide to Cryptography*. New York: McGraw-Hill, 1999.

[SING99](#) Singh, S. : *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. New York: Anchor Books, 1999.

[SINK66](#) Sinkov, A. *Elementary Cryptanalysis: A Mathematical Approach*. Washington, DC: The Mathematical Association of America, 1966.

Recommended Web Sites



- **American Cryptogram Association:** An association of amateur cryptographers. The Web site includes information and links to sites concerned with classical cryptography.
-

[Page 56]

- **Crypto Corner:** Simon Singh's Web site. Lots of good information, plus interactive tools for learning about cryptography.
- **Steganography:** Good collection of links and documents.

◀ PREV

NEXT ▶

2.7. Key Terms, Review Questions, and Problems

Key Terms

[block cipher](#)

[brute-force attack](#)

[Caesar cipher](#)

[cipher](#)

[ciphertext](#)

[computationally secure](#)

[conventional encryption](#)

[cryptanalysis](#)

[cryptographic system](#)

[cryptography](#)

[cryptology](#)

[deciphering](#)

[decryption](#)

[enciphering](#)

[encryption](#)

[Hill cipher](#)

[monoalphabetic cipher](#)

[one-time pad](#)

[plaintext](#)

[Playfair cipher](#)

[polyalphabetic cipher](#)

rail fence cipher

[single-key encryption](#)

[steganography](#)

[stream cipher](#)

[symmetric encryption](#)

[transposition cipher](#)

[unconditionally secure](#)

[Vigenère cipher](#)

Review Questions

- 2.1 What are the essential ingredients of a symmetric cipher?
- 2.2 What are the two basic functions used in encryption algorithms?
- 2.3 How many keys are required for two people to communicate via a cipher?
- 2.4 What is the difference between a block cipher and a stream cipher?
- 2.5 What are the two general approaches to attacking a cipher?
- 2.6 List and briefly define types of cryptanalytic attacks based on what is known to the attacker.
- 2.7 What is the difference between an unconditionally secure cipher and a computationally secure cipher?
- 2.8 Briefly define the Caesar cipher.
- 2.9 Briefly define the monoalphabetic cipher.
- 2.10 Briefly define the Playfair cipher.
- 2.11 What is the difference between a monoalphabetic cipher and a polyalphabetic cipher?
- 2.12 What are two problems with the one-time pad?

2.13 What is a transposition cipher?

2.14 What is steganography?

Problems

2.1 A generalization of the Caesar cipher, known as the affine Caesar cipher, has the following form: For each plaintext letter p , substitute the ciphertext letter C :

$$C = E([a, b], p) = (ap + b) \bmod 26$$

[Page 57]

A basic requirement of any encryption algorithm is that it be one-to-one. That is, if $p \neq q$, then $E(k, p) \neq E(k, q)$. Otherwise, decryption is impossible, because more than one plaintext character maps into the same ciphertext character. The affine Caesar cipher is not one-to-one for all values of a . For example, for $a = 2$ and $b = 3$, then $E([a, b], 0) = E([a, b], 13) = 3$.

a.

Are there any limitations on the value of b ? Explain why or why not.

b.

Determine which values of a are not allowed.

c.

Provide a general statement of which values of a are and are not allowed. Justify your statement.

2.2 How many one-to-one affine Caesar ciphers are there?

2.3 A ciphertext has been generated with an affine cipher. The most frequent letter of the ciphertext is 'B', and the second most frequent letter of the ciphertext is 'U'. Break this code.

2.4 The following ciphertext was generated using a simple substitution algorithm:

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

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?

[Page 58]

2.7 This problem uses a real-world example, from an old U.S. Special Forces manual (public domain). A copy is available at <ftp://shell.shore.net/members/w/s/ws/Support/Crypto/FM-31-4.pdf>

a.

Using the two keys (memory words) *cryptographic* and *network security*, encrypt the following message:

Be at the third pillar from the left outside the lyceum theatre tonight at seven.
If you are distrustful bring two friends.

Make reasonable assumptions about how to treat redundant letters and excess letters in the memory words and how to treat spaces and punctuation. Indicate what your assumptions are. *Note:* The message is from the Sherlock Holmes novel, *The Sign of Four*.

b.

Decrypt the ciphertext. Show your work.

c.

Comment on when it would be appropriate to use this technique and what its advantages are.

2.8 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 2.2](#) (the one that begins "it was disclosed yesterday"). Determine the keyword.

2.9 When the PT-109 American patrol boat, under the command of Lieutenant John F. Kennedy, was sunk by a Japanese destroyer, a message was received at an Australian wireless station in Playfair code:

```
KXJEY UREBE ZWEHE WRYTU HEYFS
KREHE GOYFI WTTTU OLKSY CAJPO
BOTEI ZONTX BYBNT GONEY CUZWR
GDSON SXBOU YWRHE BAAHY USEDQ
```

The key used was *royal new zealand navy*. Decrypt the message. Translate TT into tt.

2.10

a.

Construct a Playfair matrix with the key *largest*.

b.

Construct a Playfair matrix with the key *occurrence*. Make a reasonable assumption about how to treat redundant letters in the key.

2.11

a.

Using this Playfair matrix

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

[Page 59]

encrypt this message:

Must see you over Cadogan West. Coming at once.

Note: The message is from the Sherlock Holmes story, *The Adventure of the Bruce-Partington Plans*.

b.

Repeat part (a) using the Playfair matrix from Problem 2.10a.

c.

How do you account for the results of this problem? Can you generalize your conclusion?

2.12 a. How many possible keys does the Playfair cipher have? Ignore the fact that some keys might produce identical encryption results. Express your answer as an approximate power of 2.

b.

Now take into account the fact that some Playfair keys produce the same encryption results. How many effectively unique keys does the Playfair cipher have?

2.13 What substitution system results when we use a 25 x 1 Playfair matrix?

2.14 a. Decipher the message YITJP GWJOW FAQTQ XCSMA ETSQU SQAPU SQGKC

PQTYJ using the Hill cipher with the inverse key $\begin{pmatrix} 5 & 1 \\ 2 & 7 \end{pmatrix}$. Show your calculations and the result.

b.

Decipher the message MWALO LIAIW WTGBH JNTAK OZJKA ADAWS SKQKU AYARN CSODN IIAES OQKJY B using the Hill cipher with the inverse key

$\begin{pmatrix} 2 & 23 \\ 21 & 7 \end{pmatrix}$. Show your calculations and the result.

2.15 a. Encrypt the message "meet me at the usual place at ten rather than eight

oclock" using the Hill cipher with the key $\begin{pmatrix} 9 & 4 \\ 5 & 7 \end{pmatrix}$. Show your calculations and the result.

b.

Show the calculations for the corresponding decryption of the ciphertext to recover the original plaintext.

2.16 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.

2.17

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

It can be shown that the Hill cipher with the matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ requires that $(ad - bc)$ is relatively prime to 26; that is the only common positive factor of $(ad - bc)$ and 26 is 1. Thus, if $(ad - bc) = 13$ or is even, the matrix is not allowed. Determine the number of different (good) keys there are for a 2 x 2 Hill cipher without counting them one by one, using the following steps:

- a. Find the number of matrices whose determinant is even because one or both rows are even. (A row is "even" if both entries in the row are even.)
- b. Find the number of matrices whose determinant is even because one or both columns are even. (A column is "even" if both entries in the column are even.)
- c. Find the number of matrices whose determinant is even because all of the entries are odd.
- d. Taking into account overlaps, find the total number of matrices whose determinant is even.
- e. Find the number of matrices whose determinant is a multiple of 13 because the first column is a multiple of 13.
- f. Find the number of matrices whose determinant is a multiple of 13 where the first column is not a multiple of 13 but the second column is a multiple of the first modulo 13.
- g. Find the total number of matrices whose determinant is a multiple of 13.
- h. Find the number of matrices whose determinant is a multiple of 26 because they fit case (a) and (e). (b) and (e). (c) and (e). (a) and (f). And so on ...
- i. Find the total number of matrices whose determinant is neither a multiple of 2 nor a multiple of 13.

2.18 Using the Vigenère cipher, encrypt the word "explanation" using the key *leg*.

2.19 This problem explores the use of a one-time pad version of the Vigenère cipher. In this scheme, the key is a stream of random numbers between 0 and 26. For example, if the key is 3 19 5 ..., then the first letter of plaintext is encrypted with a shift of 3 letters, the second with a shift of 19 letters, the third with a shift of 5 letters, and so on.

a.

Encrypt the plaintext sendmoremoney with the key stream 9 0 1 7 23 15 21 14 11 11 2 8 9.

b.

Using the ciphertext produced in part a, find a key so that the cipher text decrypts to the plaintext cashnotneeded.

2.20 What is the message embedded in [Figure 2.8](#)?

2.21 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?

Figure 2.9. A Puzzle for Lord Peter

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.

Programming Problems

- 2.22 Write a program that can encrypt and decrypt using the general Caesar cipher, also known as an additive cipher.
- 2.23 Write a program that can encrypt and decrypt using the affine cipher described in [Problem 2.1](#).
- 2.24 Write a program that can perform a letter frequency attack on an additive cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify "give me the top 10 possible plaintexts".

[Page 61]

- 2.25 Write a program that can perform a letter frequency attack on any monoalphabetic substitution cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify "give me the top 10 possible plaintexts".
- 2.26 Create software that can encrypt and decrypt using a 2 x 2 Hill cipher.
- 2.27 Create software that can perform a fast known plaintext attack on a Hill cipher, given the dimension m . How fast are your algorithms, as a function of m ?

Chapter 3. Block Ciphers and the Data Encryption Standard

[3.1 Block Cipher Principles](#)

[3.2 The Data Encryption Standard](#)

[3.3 The Strength of Des](#)

[3.4 Differential and Linear Cryptanalysis](#)

[3.5 Block Cipher Design Principles](#)

[3.6 Recommended Reading](#)

[3.7 Key Terms, Review Questions, and Problems](#)

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

Key Points

- A **block cipher** is an encryption/decryption scheme in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length.
- Many block ciphers have a Feistel structure. Such a structure consists of a number of identical rounds of processing. In each round, a substitution is performed on one half of the data being processed, followed by a permutation that interchanges the two halves. The original key is expanded so that a different key is used for each round.
- The Data Encryption Standard (DES) has been the most widely used encryption algorithm until recently. It exhibits the classic Feistel structure. DES uses a 64-bit block and a 56-bit key.
- Two important methods of cryptanalysis are [differential cryptanalysis](#) and linear cryptanalysis. DES has been shown to be highly resistant to these two types of attack.

The objective of this chapter is to illustrate the principles of modern symmetric ciphers. For this purpose, we focus on the most widely used symmetric cipher: the Data Encryption Standard (DES). Although numerous symmetric ciphers have been developed since the introduction of DES, and although it is destined to be replaced by the Advanced Encryption Standard (AES), DES remains the most important such algorithm. Further, a detailed study of DES provides an understanding of the principles used in other symmetric ciphers. We examine other important symmetric ciphers, including AES, in [Chapters 5](#) and [6](#).

This chapter begins with a discussion of the general principles of symmetric block ciphers, which are the type of symmetric ciphers studied in this book (with the exception of the stream cipher RC4 in [Chapter 6](#)). Next, we cover full DES. Following this look at a specific algorithm, we return to a more general discussion of block cipher design.

Compared to public-key ciphers such as RSA, the structure of DES, and most symmetric ciphers, is very complex and cannot be explained as easily as RSA and similar algorithms. Accordingly, the reader may wish to begin with a simplified version of DES, which is described in Appendix C. This version allows the reader to perform encryption and decryption by hand and gain a good understanding of the working of the algorithm details. Classroom experience indicates that a study of this simplified version enhances understanding of DES. [\[1\]](#)

^[1] However, you may safely skip Appendix C, 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.

3.1. Block Cipher Principles

Most 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 or 128 bits is used. Using some of the modes of operation explained in [Chapter 6](#), 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 symmetric cryptographic applications make use of block ciphers. Accordingly, the concern in this chapter, and in our discussions throughout the book of symmetric encryption, will focus 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

Plaintext Ciphertext

00	11
01	10
10	00
11	01

Irreversible Mapping

Plaintext Ciphertext

00	11
01	10
10	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 $2^n!$.

[Page 65]

[Figure 3.1](#) 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. Feistel refers to this as the *ideal block cipher*, because it allows for the maximum number of possible encryption mappings from the plaintext block [[FEIS75](#)].

[Page 66]

Figure 3.1. General n -bit- n -bit Block Substitution (shown with $n = 4$)

(This item is displayed on page 65 in the print version)

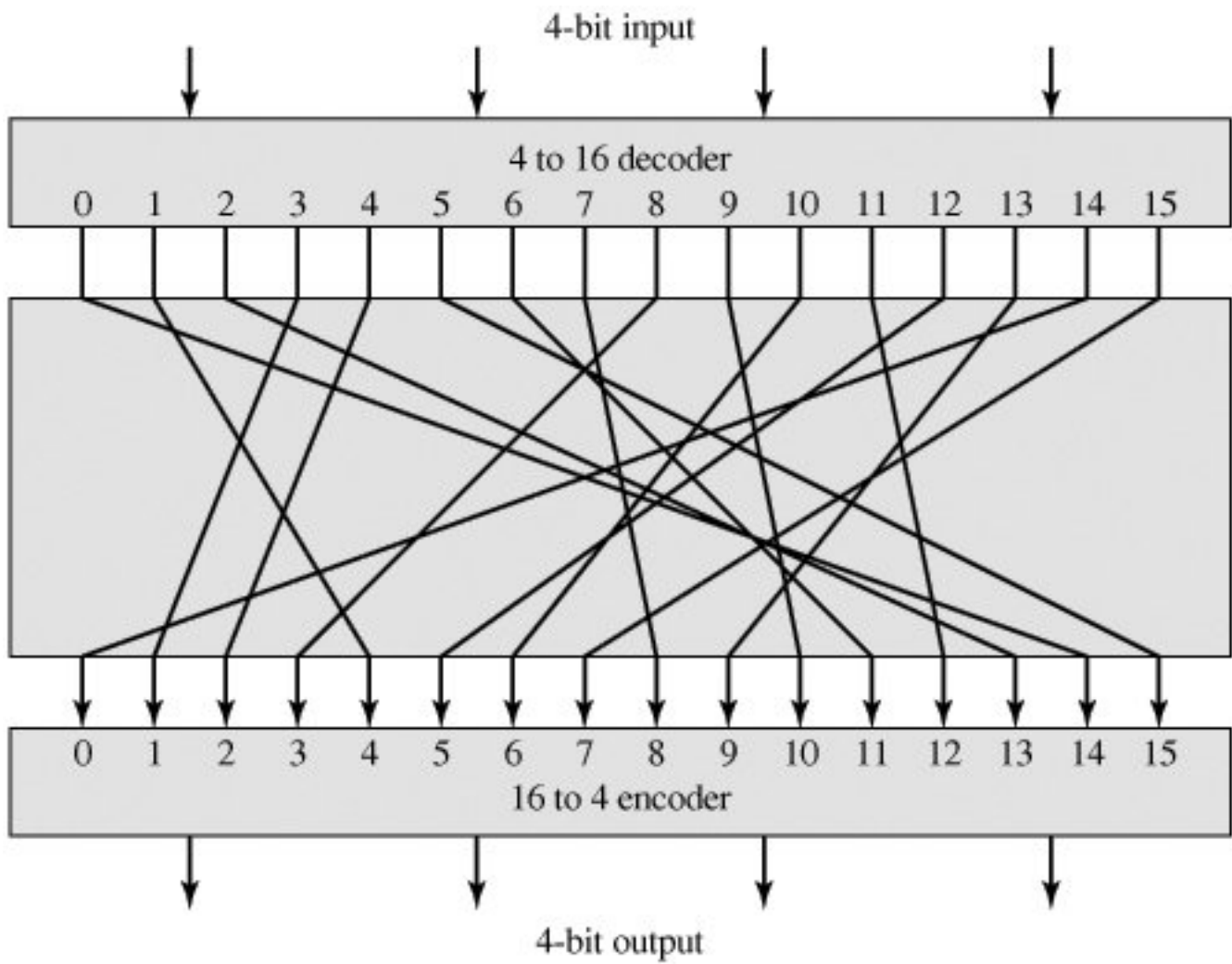


Table 3.1.
Encryption and
Decryption Tables
for Substitution
Cipher of [Figure 3.4](#)

(This item is displayed on
page 65 in the print
version)

Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010

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

But there is a practical problem with the ideal block cipher. 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 sufficiently 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 (the ideal block 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 constitutes 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, using this straightforward method of defining the key, the required key length is $(4 \text{ bits}) \times (16 \text{ rows}) = 64 \text{ bits}$. In general, for an n -bit ideal block cipher, the length of the key defined in this fashion is $n \times 2^n$ bits. For a 64-bit block, which is a desirable length to thwart statistical attacks, the required key length 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 the 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 use the general block substitution cipher but, to make its implementation tractable, confine ourselves to a subset of the 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

$$y_1 = k_{11}x_1 + k_{12}x_2 + k_{13}x_3 + k_{14}x_4$$

$$y_2 = k_{21}x_1 + k_{22}x_2 + k_{23}x_3 + k_{24}x_4$$

$$y_3 = k_{31}x_1 + k_{32}x_2 + k_{33}x_3 + k_{34}x_4$$

$$y_4 = k_{41}x_1 + k_{42}x_2 + k_{43}x_3 + k_{44}x_4$$

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 quite vulnerable.

The Feistel Cipher

Feistel proposed [[FEIS73](#)] that we can approximate the ideal block cipher by utilizing the concept of a product cipher, which is the execution of two or more simple ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers. The essence of the approach is to develop a block cipher with a key length of k bits and a block length of n bits, allowing a total of 2^k possible transformations, rather than the $2^n!$ transformations available with the ideal block cipher.

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 [SHAN49]. 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 many 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].^[2] 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 3.1) is such a cipher, but as we have seen, is impractical.

^[2] 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.

Other than recourse to ideal systems, Shannon suggests 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; generally this is equivalent to having each ciphertext digit be 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:

$$y_n = \left(\sum_{i=1}^k m_{n+i} \right) \bmod 26$$

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.^[3]

^[3] 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.

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 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.

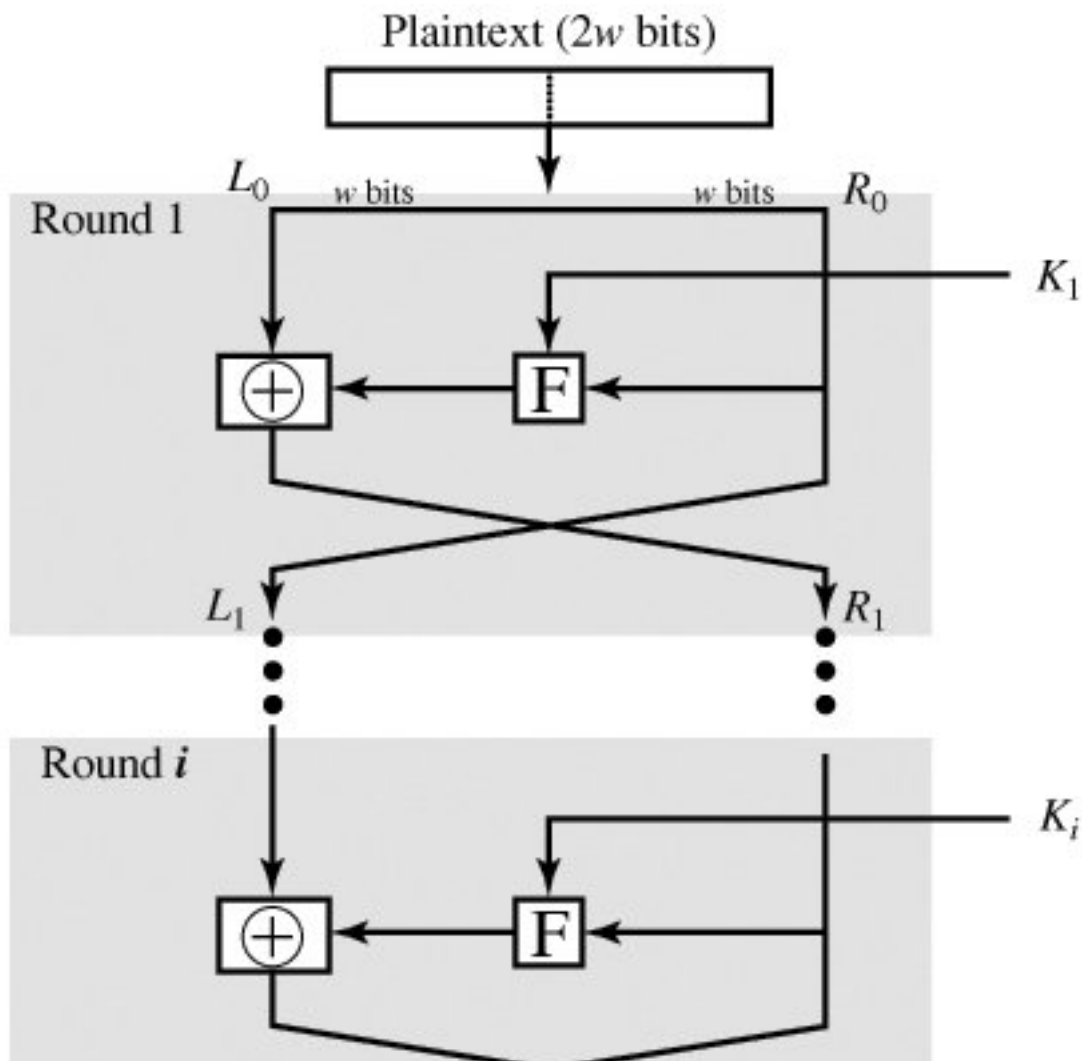
As [ROBS95b] points out, so successful are diffusion and confusion in capturing the essence of the desired attributes of a block cipher that they have become the cornerstone of modern block cipher design.

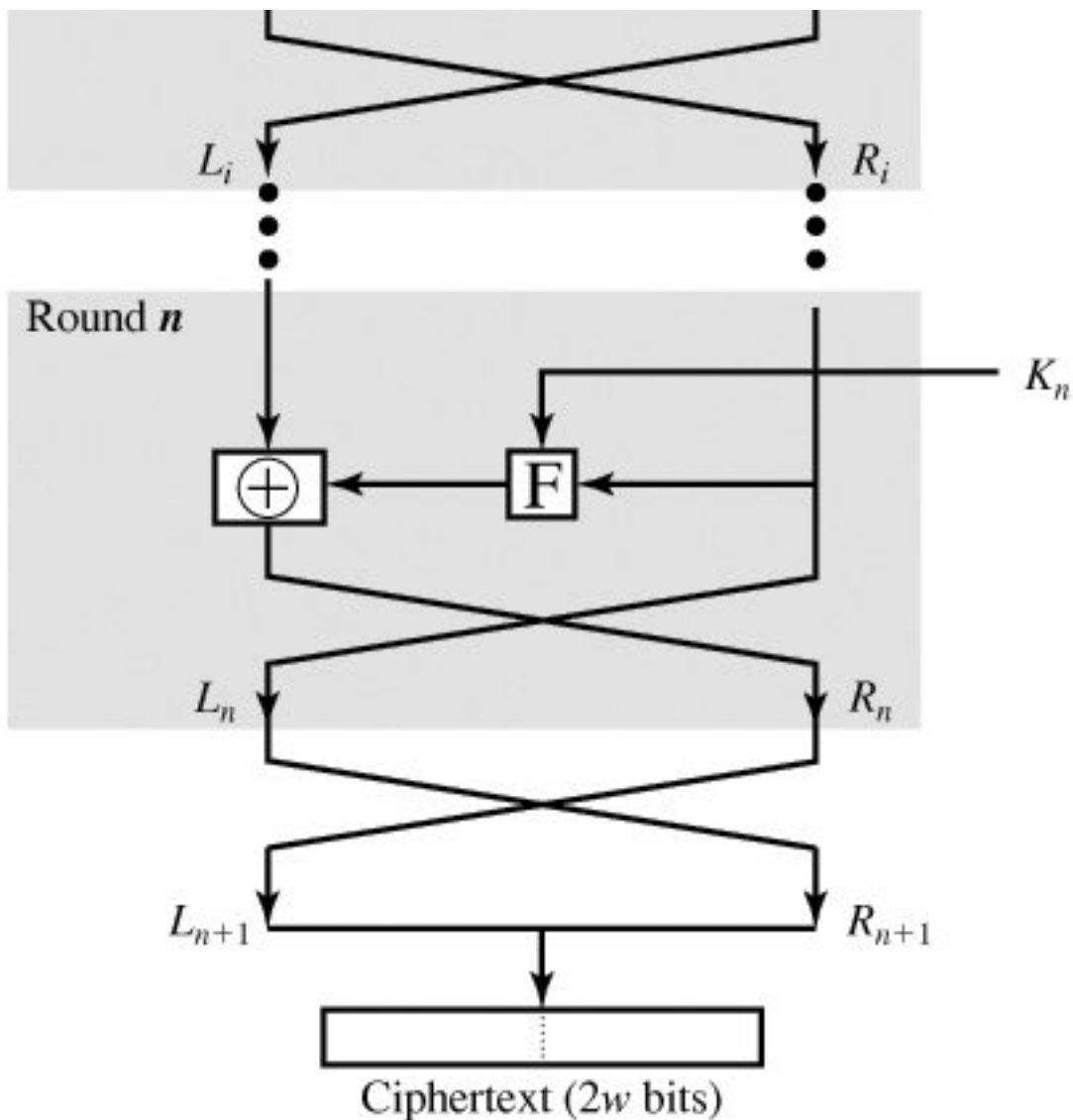
Feistel Cipher Structure

Figure 3.2 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.

Figure 3.2. Classical Feistel Network

(This item is displayed on page 69 in the print version)





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. ^[4] This structure is a particular form of the substitution-permutation network (SPN) proposed by Shannon.

^[4] 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.

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 for a given algorithm. The greater security is achieved by greater diffusion. Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.

- **Key size:** Larger key size means greater security but may decrease encryption/decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. 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.
- **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.

[Page 70]

- **Round function:** Again, greater complexity generally means greater resistance to cryptanalysis.

There are two other considerations in the design of a Feistel cipher:

- **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.

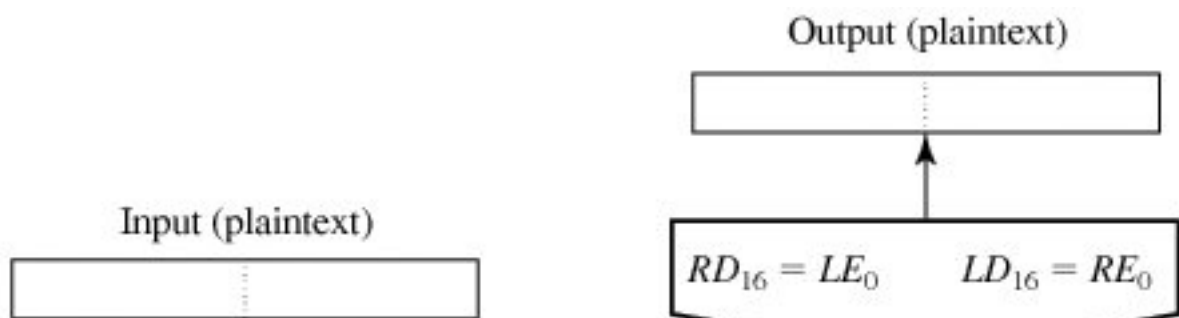
Feistel Decryption Algorithm

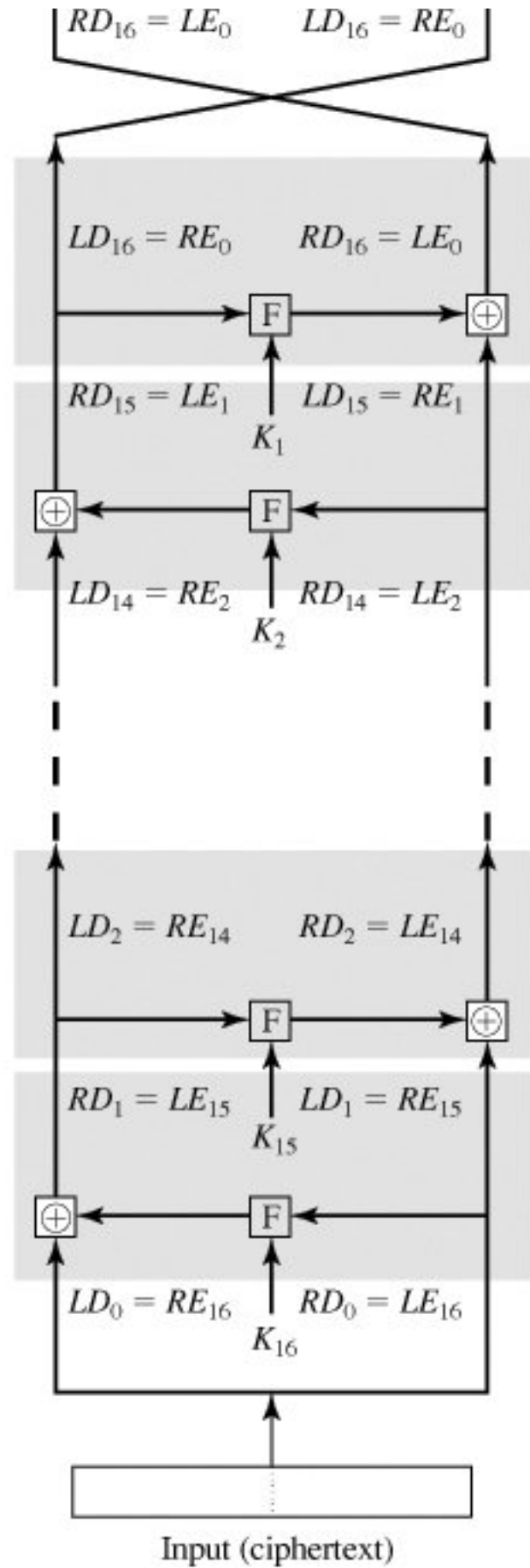
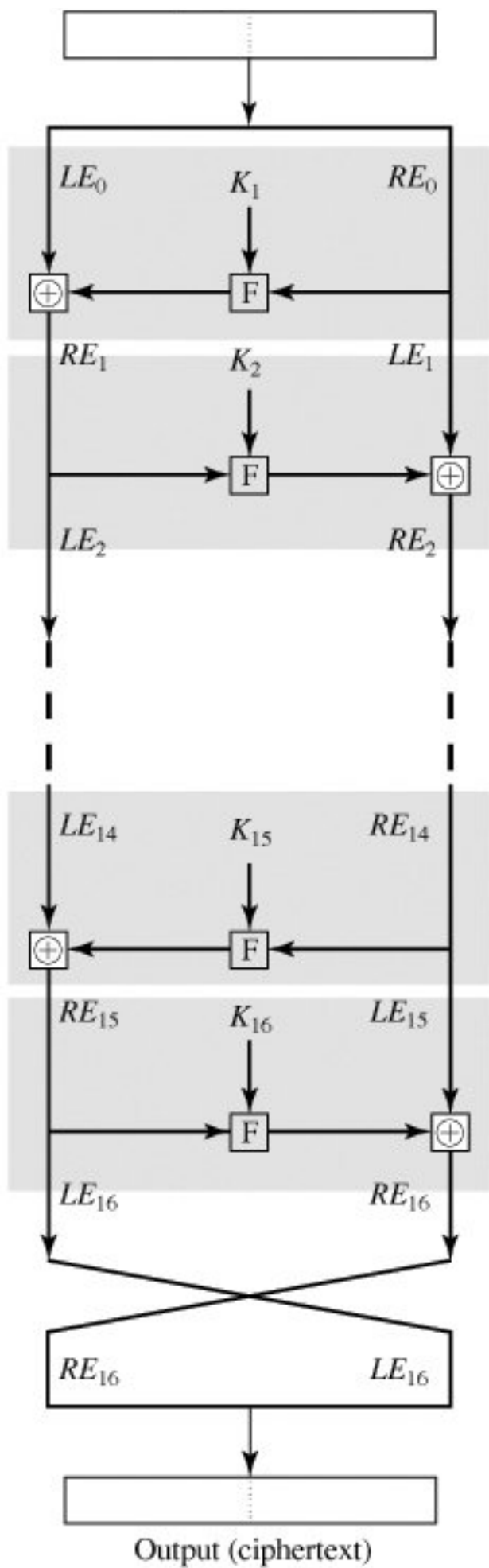
The process of decryption with a Feistel cipher is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the algorithm, but use the subkeys K_i in reverse order. That is, use K_n in the first round, K_{n-1} in the second round, and so on until K_1 is used 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 3.3](#), 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 $RE_i || LE_i$, or, equivalently, $RD_{16-i} || LD_{16-i}$.

Figure 3.3. Feistel Encryption and Decryption

(This item is displayed on page 71 in the print version)





Let us walk through [Figure 3.3](#) to demonstrate the validity of the preceding assertions.^[5] 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.

^[5] To simplify the diagram, it is untwisted, 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 .

[Page 71]

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

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

$$RE_{16} = LE_{15} \times F(RE_{15}, K_{16})$$

[Page 72]

On the decryption side,

$$LD_1 = RD_0 = LE_{16} = RE_{15}$$

$$RD_1 = LD_0 \times F(RD_0, K_{16})$$

$$= RE_{16} \times F(RE_{15}, K_{16})$$

$$= [LE_{15} \times F(RE_{15}, K_{16})] \times F(RE_{15}, K_{16})$$

The XOR has the following properties:

$$[A \times B] \times C = A \times [B \times C]$$

$$D \times D = 0$$

$$E \times 0 = E$$

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,

$$LE_i = RE_{i-1}$$

$$RE_i = LE_{i-1} \times F(RE_{i-1}, K_i)$$

Rearranging terms,

$$RE_{i-1} = LE_i$$

$$LE_{i-1} = RE_i \times F(RE_{i-1}, K_i) = RE_i \times F(LE_i, K_i)$$

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 3.3](#).

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.

◀ PREY

NEXT ▶

3.2. 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). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).^[6] 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.

^[6] The terminology is a bit confusing. Until recently, the terms *DES* and *DEA* could be used interchangeably. However, the most recent edition of the DES document includes a specification of the DEA described here plus the triple DEA (TDEA) described in [Chapter 6](#). Both DEA and TDEA are part of the Data Encryption Standard. Further, until the recent adoption of the official term *TDEA*, the triple DEA algorithm was typically referred to as *triple DES* and written as 3DES. For the sake of convenience, we use the term 3DES.

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.

In the late 1960s, IBM set up a research project in computer cryptography led by Horst Feistel. The project concluded in 1971 with the development of an algorithm with the designation LUCIFER [\[FEIS73\]](#), which was sold to Lloyd's of London for use in a cash-dispensing system, also developed by IBM. LUCIFER is a Feistel block cipher that operates on blocks of 64 bits, using a key size of 128 bits. Because of the promising results produced by the LUCIFER project, IBM embarked on an effort to develop a marketable commercial encryption product that ideally could be implemented on a single chip. The effort was headed by Walter Tuchman and Carl Meyer, and it involved not only IBM researchers but also outside consultants and technical advice from NSA. The outcome of this effort was a refined version of LUCIFER that was more resistant to cryptanalysis but that had a reduced key size of 56 bits, to fit on a single chip.

In 1973, the National Bureau of Standards (NBS) issued a request for proposals for a national cipher standard. IBM submitted the results of its Tuchman-Meyer project. This was by far the best algorithm proposed and was adopted in 1977 as the Data Encryption Standard.

Before its adoption as a standard, the proposed DES was subjected to intense criticism, which has not subsided to this day. Two areas drew the critics' fire. First, the key length in IBM's original LUCIFER algorithm was 128 bits, but that of the proposed system was only 56 bits, an enormous reduction in key size of 72 bits. Critics feared that this key length was too short to withstand brute-force attacks. The second area of concern was that the design criteria for the internal structure of DES, the S-boxes, were classified. Thus, users could not be sure that the internal structure of DES was free of any hidden weak points that would enable NSA to decipher messages without benefit of the key. Subsequent events, particularly the recent work on differential cryptanalysis, seem to indicate that DES has a very strong internal structure. Furthermore, according to IBM participants, the only changes that were made to the proposal were changes to the S-boxes, suggested by NSA, that removed vulnerabilities identified in the course of the evaluation process.

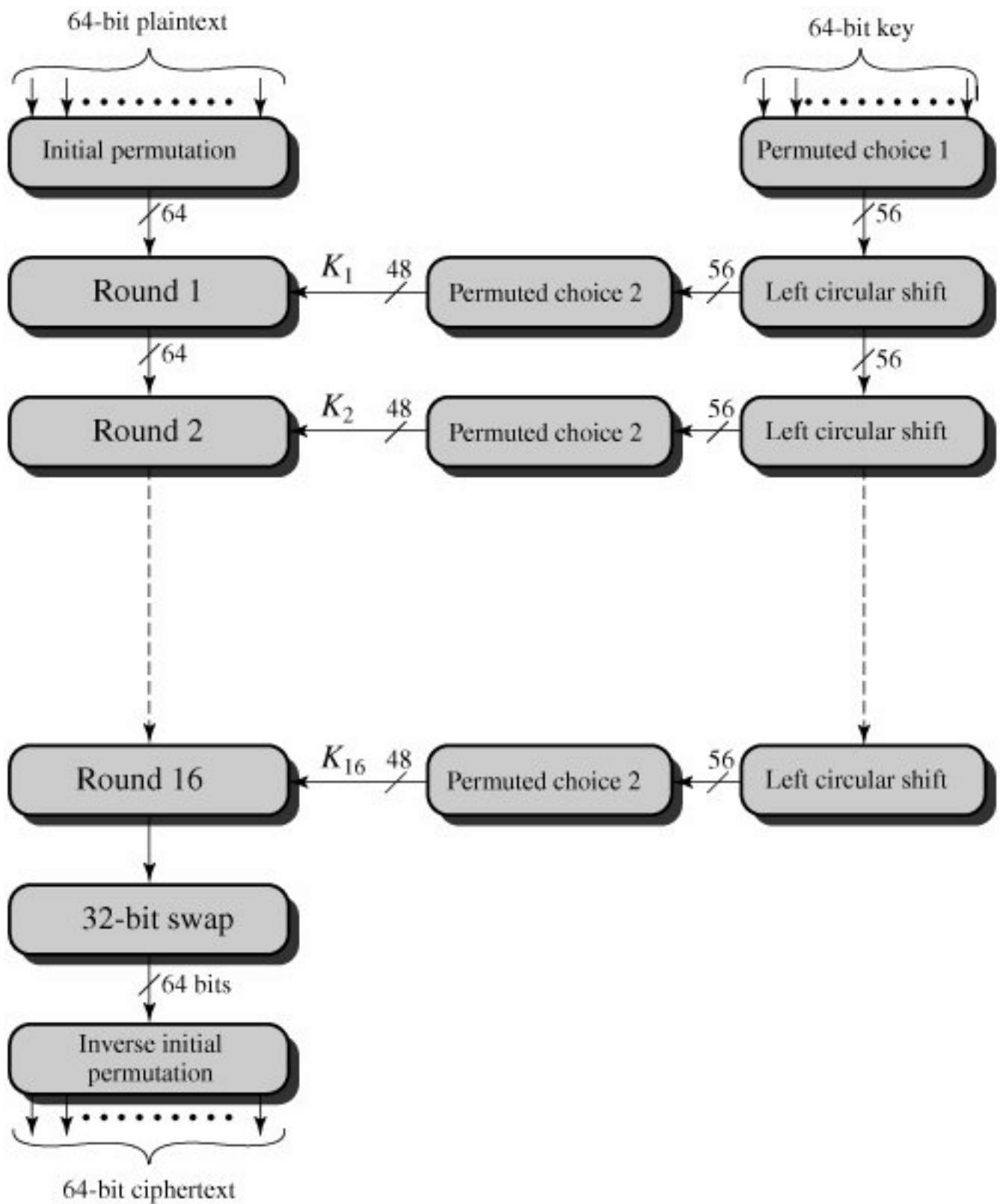
Whatever the merits of the case, DES has flourished and is widely used, especially in financial applications. In 1994, NIST reaffirmed DES for federal use for another five years; NIST recommended the use of DES for applications other than the protection of classified information. In 1999, NIST issued a new version of its standard (FIPS PUB 46-3) that indicated that DES should only be used for legacy systems and that triple DES (which in essence involves repeating the DES algorithm three times on the plaintext using two or three different keys to produce the ciphertext) be used. We study triple DES in [Chapter 6](#). Because the underlying encryption and decryption algorithms are the same for DES and triple DES, it remains important to understand the DES cipher.

DES Encryption

The overall scheme for DES encryption is illustrated in [Figure 3.4](#). 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 must be 64 bits in length and the key is 56 bits in length. [\[7\]](#)

^[7] Actually, the function expects a 64-bit key as input. However, only 56 of these bits are ever used; the other 8 bits can be used as parity bits or simply set arbitrarily.

Figure 3.4. General Depiction of DES Encryption Algorithm



Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an 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 3.2](#).

The right-hand portion of [Figure 3.4](#) 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 shifts of the key bits.

Initial Permutation

The initial permutation and its inverse are defined by tables, as shown in [Tables 3.2a](#) and [3.2b](#), respectively. The tables are to be interpreted as follows. The input to a table consists of 64 bits numbered from 1 to 64. The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits.

Table 3.2. Permutation Tables for DES

(This item is displayed on page 76 in the print version)

(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

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 3.5](#) 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:

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

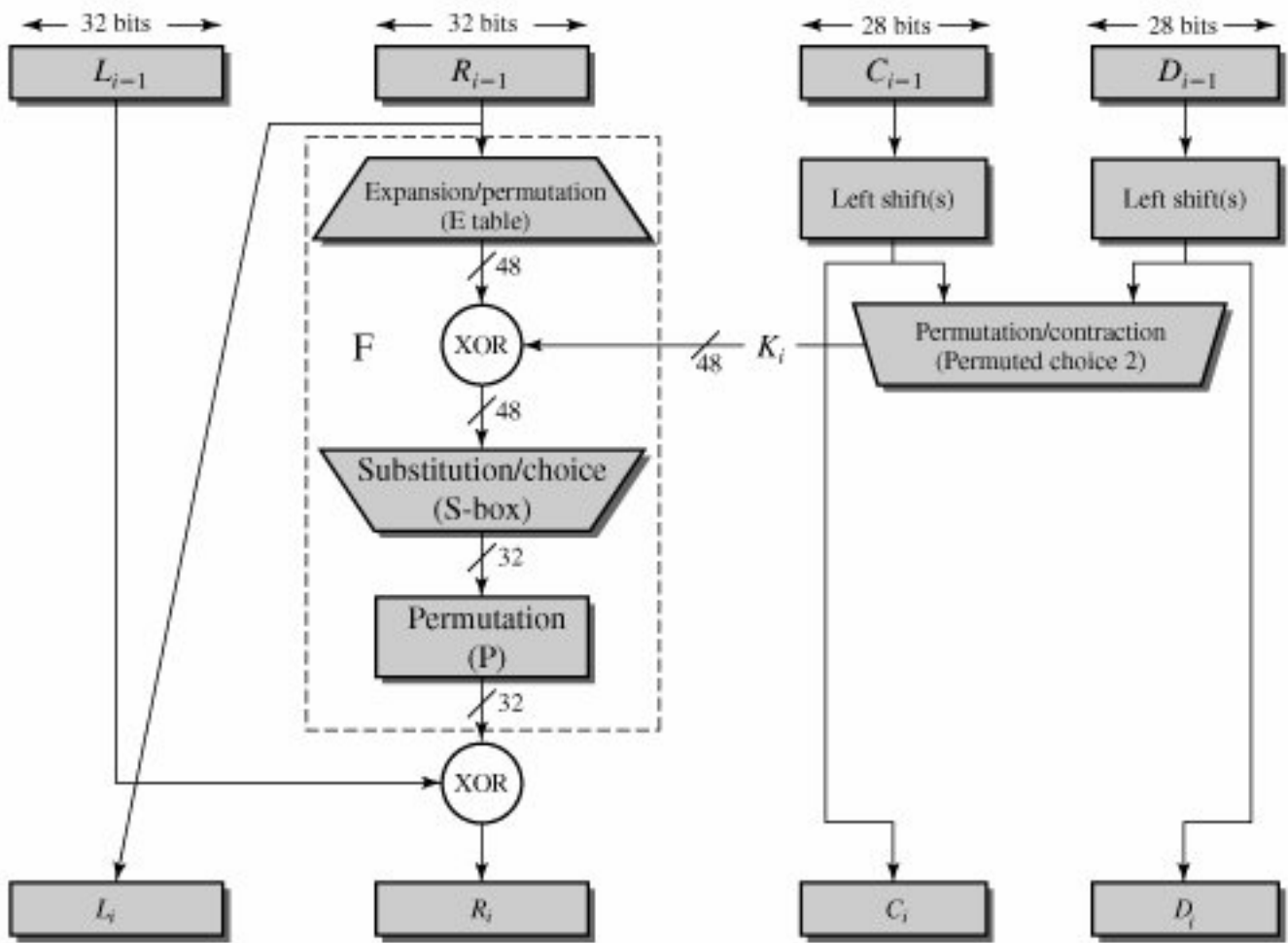
$$R_i = L_{i-1} \times F(R_{i-1}, K_i)$$

[Page 76]

[Page 77]

Figure 3.5. Single Round of DES Algorithm

[View full size image](#)



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 3.6. 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_j form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for S_j . The middle four bits select one of the sixteen columns. 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.

Figure 3.6. Calculation of $F(R, K)$

(This item is displayed on page 78 in the print version)

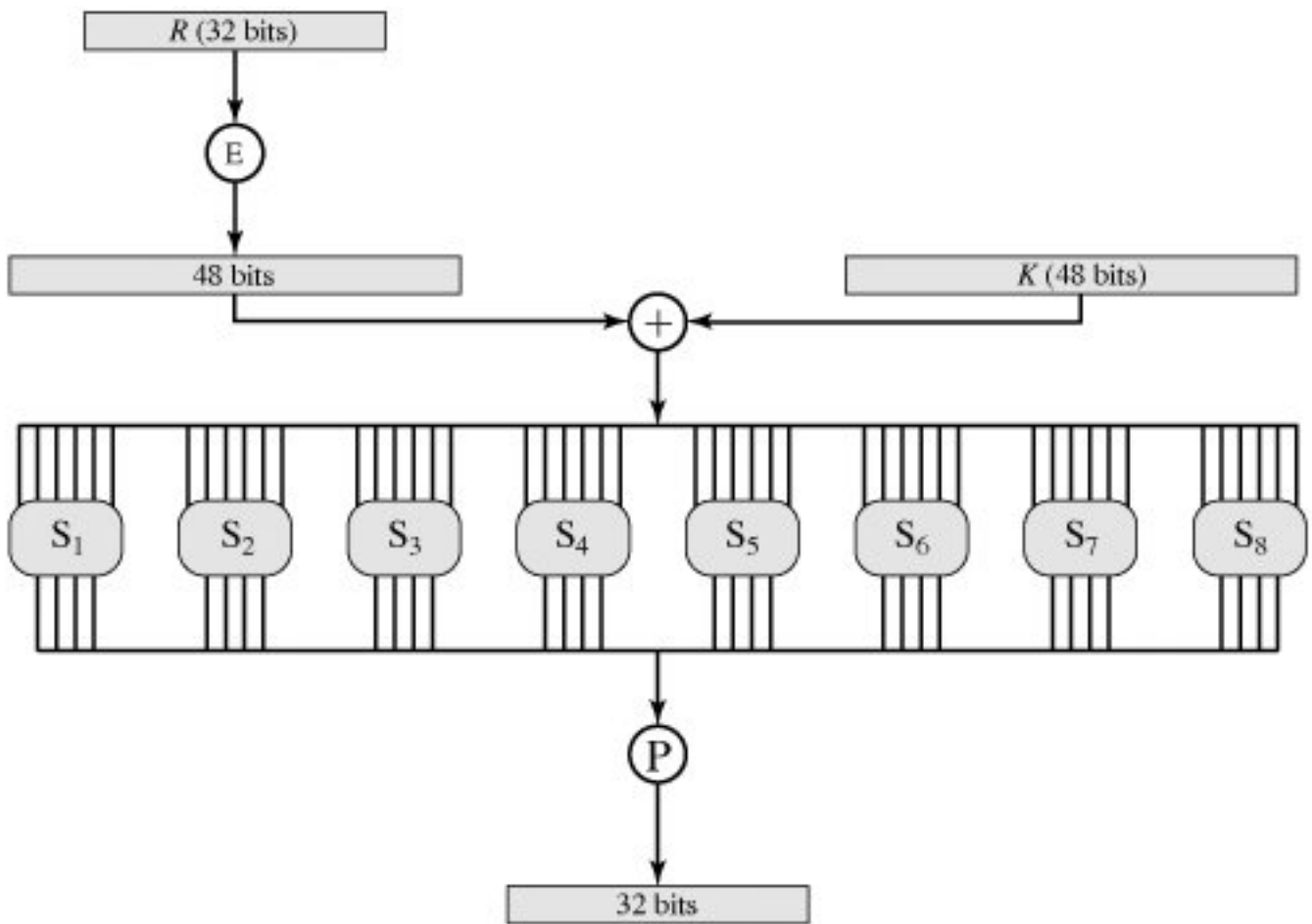


Table 3.3. Definition of DES S-Boxes

(This item is displayed on page 79 in the print version)

[\[View full size image\]](#)

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

Each row of an S-box defines a general reversible substitution. [Figure 3.1](#) may be useful in understanding the mapping. The figure shows the substitution for row 0 of box S_1 .

The operation 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 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 lmnopq ...

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 3.4](#) and [3.5](#), we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated by the lack of shading in [Table 3.4a](#). The key is first subjected to a permutation governed by a table labeled Permuted Choice One ([Table 3.4b](#)). 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 3.4d](#). These shifted values serve as input to the next round. They also serve as input to Permuted Choice Two ([Table 3.4c](#)), which produces a 48-bit output that serves as input to the function $F(R_{i-1}, K_i)$.

Table 3.4. DES Key Schedule Calculation

(a) Input Key															
	1		2		3		4		5		6		7		8
	9		10		11		12		13		14		15		16
	17		18		19		20		21		22		23		24

	25		26		27		28		29		30		31		32
	33		34		35		36		37		38		39		40
	41		42		43		44		45		46		47		48
	49		50		51		52		53		54		55		56
	57		58		59		60		61		62		63		64

(b) 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	

(c) 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

(d) 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 3.5a](#), 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

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

The [Table 3.5a](#) 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 00101111 01111010 00010011 01110110 11101011 10100100

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.

3.3. 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 7.2×10^{16} . Thus, on the face of it, 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.

DES finally and definitively proved insecure in July 1998, when the Electronic Frontier Foundation (EFF) announced that it had broken a DES encryption using a special-purpose "DES cracker" machine that was built for less than \$250,000. The attack took less than three days. The EFF has published a detailed description of the machine, enabling others to build their own cracker [[EFF98](#)]. And, of course, hardware prices will continue to drop as speeds increase, making DES virtually worthless.

It is important to note that 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 EFF approach addresses this issue as well and introduces some automated techniques that would be effective in many contexts.

Fortunately, there are a number of alternatives to DES, the most important of which are AES and triple DES, discussed in [Chapters 5](#) and [6](#), respectively.

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.

[8]

^[8] At least, no one has publicly acknowledged such a discovery.

Timing Attacks

We discuss timing attacks in more detail in Part Two, as they relate to public-key algorithms. However, the issue may also be relevant for symmetric ciphers. In essence, a timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs. [\[HEVI99\]](#) reports on an approach that yields the Hamming weight (number of bits equal to one) of the secret key. This is a long way from knowing the actual key, but it is an intriguing first step. The authors conclude that DES appears to be fairly resistant to a successful timing attack but suggest some avenues to explore. Although this is an interesting line of attack, it so far appears unlikely that this technique will ever be successful against DES or more powerful symmetric ciphers such as triple DES and AES.

3.4. 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} encryptions, requiring 2^{47} chosen plaintexts. Although 2^{47} is certainly significantly less than 2^{55} the need for the adversary 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 Attack

The differential cryptanalysis attack is complex; [[BIHA93](#)] provides a complete description. The rationale behind differential cryptanalysis is to observe the behavior of pairs of text blocks evolving along each round of the cipher, instead of observing the evolution of a single text block. 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:

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

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: $m_i = m_i \oplus m'_i$. Then we have:

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

Now, 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 Δm_{i-1} and Δm_i with high probability, then we know Δm_{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 .

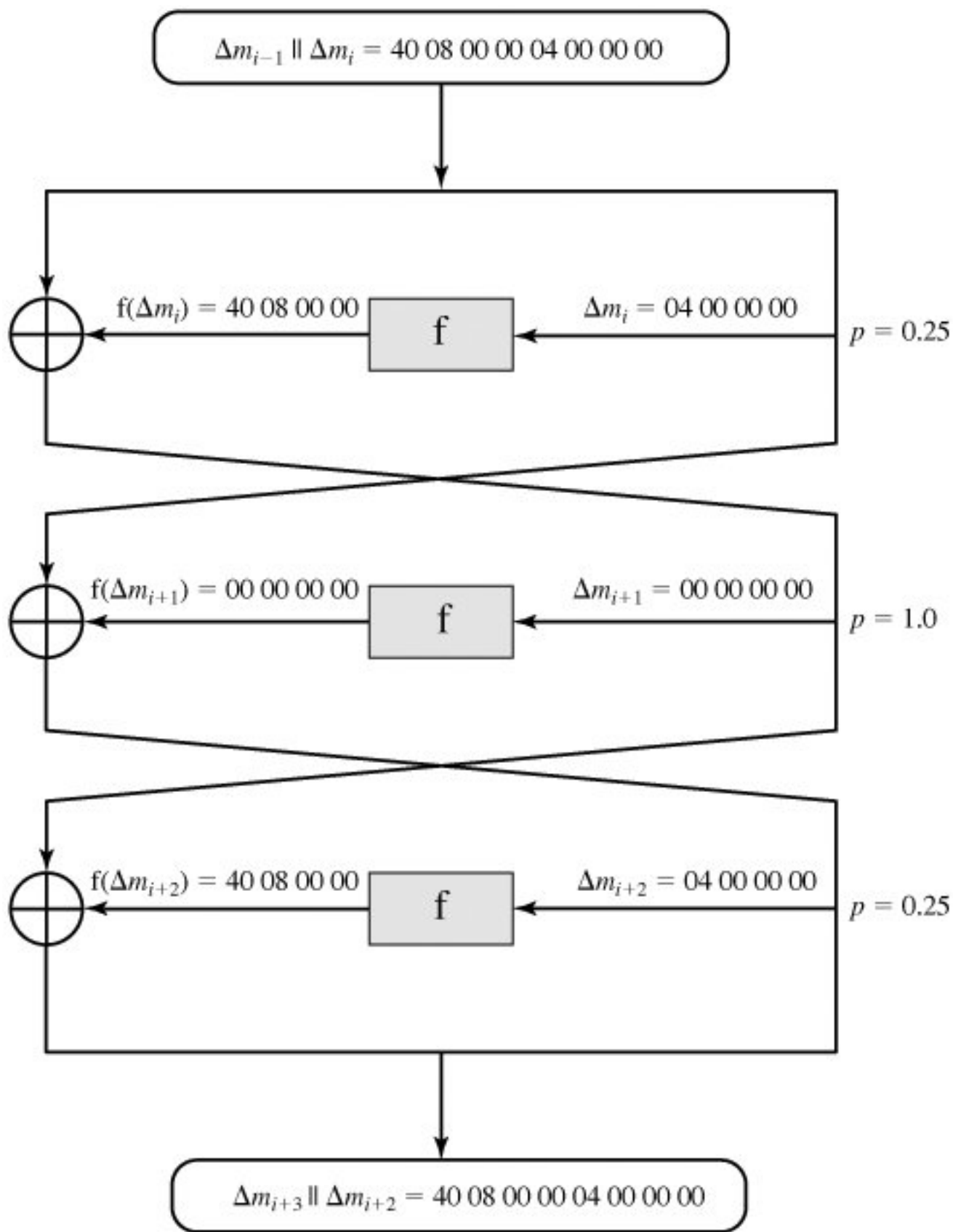
The overall strategy of differential cryptanalysis is based on these considerations for a single round. The procedure is to begin with two plaintext messages m and m' with a given difference and trace through a probable pattern of differences after each round to yield a probable difference for the ciphertext.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

$$E(K, m) \oplus E(K, m') = (\Delta m_{17} || m_{16})$$

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 3.7](#), 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 3.7. Differential Propagation through Three Round 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^{43} 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.

[Page 86]

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 α , β and γ 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.

◀ PREV

NEXT ▶

3.5. 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 3.6). 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 $1/2$.

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.

[Page 87]

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, [9] whereas brute force requires 2^{55} . If DES had 15 or fewer rounds, differential cryptanalysis would require less effort than brute-force key search.

[9] 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 $2^{55.1}$.

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.

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 $1/2$ 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. ^[10] 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.

^[10] A good summary of S-box design studies through early 1996 can be found in [[SCHN96](#)].

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 6×4 S-boxes. Blowfish, described in [Chapter 6](#), has 8×32 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 in a different manner than 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, in an 8×32 S-box, if the input is 00001001, the output consists of the 32 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. They 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](#)]. 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 π if, for a 1-bit input change, at least π 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 8×32 , 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 [[ROBS95b](#)]):

- **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., 6×4) but should be acceptable for large S-boxes (e.g., 8×32).
- **Random with testing:** Choose S-box entries randomly, then test the results against various criteria, and throw away those that do not pass.
- **Human-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.

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 6](#), 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.

3.6. Recommended Reading

There is a wealth of information on symmetric 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 up to the time of the writing of the book. 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 [MENE97]. A rigorous mathematical treatment is [STIN02].

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

Perhaps the most detailed description of DES is [SIMO95]; the book also contains an extensive discussion of differential and linear cryptanalysis of DES. [BARK91] provides a readable and interesting analysis of the structure of DES and of potential cryptanalytic approaches to DES. [EFF98] details the most effective brute-force attack on DES. [COPP94] looks at the inherent strength of DES and its ability to stand up to cryptanalysis.

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.

EFF98 Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design*. Sebastopol, CA: O'Reilly, 1998

MENE97 Menezes, A.; van Oorschot, P.; and Vanstone, S. *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1997.

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

SIMO95 Simovits, M. *The DES: An Extensive Documentation and Evaluation*. Laguna Hills, CA: Aegean Park Press, 1995.

STIN02 Stinson, D. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 2002.

3.7. Key Terms, Review Questions, and Problems

Key Terms

[avalanche effect](#)

[block cipher](#)

[confusion](#)

[Data Encryption Standard \(DES\)](#)

[differential cryptanalysis](#)

[diffusion](#)

[Feistel cipher](#)

[irreversible mapping](#)

[key](#)

[linear cryptanalysis](#)

[permutation](#)

[product cipher](#)

[reversible mapping](#)

[round](#)

[round function](#)

[subkey](#)

[substitution](#)

- 3.1 Why is it important to study the Feistel cipher?
- 3.2 What is the difference between a block cipher and a stream cipher?
- 3.3 Why is it not practical to use an arbitrary reversible substitution cipher of the kind shown in [Table 3.1](#)?
- 3.4 What is a product cipher?
- 3.5 What is the difference between diffusion and confusion?
- 3.6 Which parameters and design choices determine the actual algorithm of a Feistel cipher?
- 3.7 What is the purpose of the S-boxes in DES?
- 3.8 Explain the avalanche effect.
- 3.9 What is the difference between differential and linear cryptanalysis?

Problems

- 3.1
 - a. In [Section 3.1](#), under the subsection on the motivation for the Feistel cipher structure, it was stated that, for a block of n bits, the number of different reversible mappings for the ideal block cipher is $2^n!$. Justify.

b.

In that same discussion, it was stated that for the ideal block cipher, which allows all possible reversible mappings, the size of the key is $n \times 2^n$ bits. But, if there are $2^n!$ possible mappings, it should take $\log_2 2^n!$ bits to discriminate among the different mappings, and so the key length should be $\log_2 2^n!$. However, $\log_2 2^n! < n \times 2^n$. Explain the discrepancy.

- 3.2** Consider a Feistel cipher composed of 16 rounds with block length 128 bits and key length 128 bits. Suppose that, for a given k , the key scheduling algorithm determines values for the first 8 round keys, k_1, k_2, \dots, k_8 , and then sets

$$k_9 = k_8, k_{10} = k_7, k_{11} = k_6, \dots, k_{16} = k_1$$

Suppose you have a ciphertext c . Explain how, with access to an encryption oracle, you can decrypt c and determine m using just a single oracle query. This shows that such a cipher is vulnerable to a chosen plaintext attack. (An encryption oracle can be thought of as a device that, when given a plaintext, returns the corresponding ciphertext. The internal details of the device are not known to you and you cannot break open the device. You can only gain information from the oracle by making queries to it and observing its responses.)

- 3.3** 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 $C = E(K', P_i)$ for $1 \neq i \neq t$. 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 $E(K, \cdot)$ and $E(K', \cdot)$ exactly agree on the t plaintext-ciphertext pairs P_i, C_i and agree on no other pairs.

a.

What is the probability that $E(K, \cdot)$ and $E(K', \cdot)$ are in fact distinct mappings?

b.

What is the probability that $E(K, \cdot)$ and $E(K', \cdot)$ agree on another t' plaintext-ciphertext pairs where $0 \neq t' \neq N - t$?

- 3.4** Let π be a permutation of the integers $0, 1, 2, \dots, (2^n - 1)$ such that $\pi(m)$ gives the permuted value of m , $0 \neq m \leq 2^n$. Put another way, π 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 π has a fixed point at m if $\pi(m) = m$. That is, if π is an encryption mapping, then a fixed point corresponds to a message that encrypts to itself. We are interested in the probability that π has no fixed points. Show the somewhat unexpected result that over 60% of mappings will have at least one fixed point.

- 3.5** Consider the substitution defined by row 1 of S-box S_1 in [Table 3.3](#). Show a block diagram similar to [Figure 3.1](#) that corresponds to this substitution.

3.6 Compute the bits number 1, 16, 33, and 48 at the output of the first round of the DES decryption, assuming that the ciphertext block is composed of all ones and the external key is composed of all ones.

3.7 Suppose the DES F function mapped every 32-bit input R, regardless of the value of the input K, to

a.

32-bit string of ones,

b.

bitwise complement of R.

Hint: Use the following properties of the XOR operation:

1.

What function would DES then compute?

2.

What would the decryption look like?

$$(A \otimes B) \otimes C = A \otimes (B \otimes C)$$

$$A \otimes A = 0$$

$$A \otimes 0 = A$$

$$A \otimes 1 = \text{bitwise complement of } A$$

where

A, B, C are n -bit strings of bits

0 is an n -bit string of zeros

1 is an n -bit string of one

3.8 This problem provides a numerical example of encryption using a one-round version of DES. We start with the same bit pattern for the key K and the plaintext, namely:

in hexadecimal notation: 0 1 2 3 4 5 6 7 8 9 A B C D E F

in binary notation: 0000 0001 0010 0011 0100 0101 0110 0111
 1000 1001 1010 1011 0100 1101 1110 1111

a.

Derive K_1 , the first-round subkey.

b.

Derive L_0, R_0 .

c.

Expand R_0 to get $E[R_0]$, where $E[\cdot]$ is the expansion function of Figure 3.8.

d.

Calculate $A = E[R_0] \otimes K_1$.

e.

Group the 48-bit result of (d) into sets of 6 bits and evaluate the corresponding S-box substitutions.

f.

Concatenate the results of (e) to get a 32-bit result, B .

g.

Apply the permutation to get $P(B)$.

h.

Calculate $R_1 = P(B) \otimes L_0$.

i.

Write down the ciphertext.

- 3.9** Show that DES decryption is, in fact, the inverse of DES encryption.
- 3.10** 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 3.7. 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.

[Page 93]

a.

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

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

b.

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:

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

Does it?

- 3.11** Compare the initial permutation table ([Table 3.2a](#)) with the permuted choice one table ([Table 3.4b](#)). Are the structures similar? If so, describe the similarities. What conclusions can you draw from this analysis?
- 3.12** 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 3.5](#) is no longer valid. Design a key-generation scheme with the appropriate shift schedule (analogous to [Table 3.4d](#)) for the decryption process.

3.13

a.

Let X' be the bitwise complement of X . Prove that if the complement of the plaintext block is taken and the complement of an encryption key is taken, then the result of DES encryption with these values is the complement of the original ciphertext. That is,

$$\text{If } Y = E(K, X)$$

$$\text{Then } Y' = E(K', X')$$

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

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?

3.14 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.

3.15 For any block cipher, the fact that it is a nonlinear function is crucial to its security. To see this, suppose that we have a linear block cipher EL that encrypts 128-bit blocks of plaintext into 128-bit blocks of ciphertext. Let $EL(k, m)$ denote the encryption of a 128-bit message m under a key k (the actual bit length of k is irrelevant). Thus

$$EL(k, [m_1 \oplus m_2]) = EL(k, m_1) \oplus EL(k, m_2) \text{ for all 128-bit patterns } m_1, m_2$$

Describe how, with 128 chosen ciphertexts, an adversary can decrypt any ciphertext without knowledge of the secret key k . (A "chosen ciphertext" means that an adversary has the ability to choose a ciphertext and then obtain its decryption. Here, you have 128 plaintext/ciphertext pairs to work with and you have the ability to choose the value of the ciphertexts.)

Note: The following problems refer to simplified DES, described in Appendix C.

3.16 Refer to Figure C.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?

- 3.17** 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 .

[Page 94]

- 3.18** Using S-DES, decrypt the string (10100010) using the key (0111111101) 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 midway check, after the application of SW, the string should be (00010011).

Programming Problems

- 3.19** Create software that can encrypt and decrypt using a general substitution block cipher.
- 3.20** Create software that can encrypt and decrypt using S-DES. Test data: Use plaintext, ciphertext, and key of [Problem 3.15](#).

◀ PREV

NEXT ▶

Chapter 4. Finite Fields

[4.1 Groups, Rings, and Fields](#)

[4.2 Modular Arithmetic](#)

[4.3 The Euclidean Algorithm](#)

[4.4 Finite Fields of the Form \$GF\(p\)\$](#)

[4.5 Polynomial Arithmetic](#)

[4.6 Finite Fields of the Form \$GF\(2^n\)\$](#)

[4.7 Recommended Reading and Web Sites](#)

[4.8 Key Terms, Review Questions, and Problems](#)

The next morning at daybreak, Star flew indoors, seemingly keen for a lesson. I said, "Tap eight." She did a brilliant exhibition, first tapping it in 4, 4, then giving me a hasty glance and doing it in 2, 2, 2, 2, before coming for her nut.

It is astonishing that Star learned to count up to 8 with no difficulty, and of her own accord discovered that each number could be given with various different divisions, this leaving no doubt that she was consciously thinking each number. In fact, she did mental arithmetic, although unable, like humans, to name the numbers. But she learned to recognize their spoken names almost immediately and was able to remember the sounds of the names. Star is unique as a wild bird, who of her own free will pursued the science of numbers with keen interest and astonishing intelligence.

Living with Birds, Len Howard

Key Points

- A field is a set of elements on which two arithmetic operations (addition and multiplication) have been defined and which has the properties of ordinary arithmetic, such as closure, associativity, commutativity, distributivity, and having both additive and multiplicative inverses.
- Modular arithmetic is a kind of integer arithmetic that reduces all numbers to one of a fixed set $[0 \dots n - 1]$ for some number n . Any integer outside this range is reduced to one in this range by taking the remainder after division by n .
- The greatest common divisor of two integers is the largest positive integer that exactly divides both integers.
- Finite fields are important in several areas of cryptography. A finite field is simply a field with a finite number of elements. It can be shown that the order of a finite field (number of elements in the field) must be a power of a prime p^n , where n is a positive integer.
- Finite fields of order p can be defined using arithmetic mod p .
- Finite fields of order p^n , for $n > 1$ can be defined using arithmetic over polynomials.

Finite fields have become increasingly important in cryptography. A number of cryptographic algorithms rely heavily on properties of finite fields, notably the Advanced Encryption Standard (AES) and elliptic curve cryptography.

The chapter begins with a brief overview of the concepts of group, ring, and field. This section is somewhat abstract; the reader may prefer to quickly skim this section on a first reading. Next, we need some elementary background in modular arithmetic and the Euclidean algorithm. We are then ready to discuss finite fields of the form $\text{GF}(p)$, where p is a prime number. Next, we need some additional background, this time in polynomial arithmetic. The chapter concludes with a discussion of finite fields of the form $\text{GF}(2^n)$ where n is a positive integer.

The concepts and techniques of number theory are quite abstract, and it is often difficult to grasp them intuitively without examples [\[RUB197\]](#). Accordingly, this chapter and [Chapter 8](#) include a number of examples, each of which is highlighted in a shaded box.

4.1. Groups, Rings, and Fields

Groups, rings, and fields are the fundamental elements of a branch of mathematics known as abstract algebra, or modern algebra. In abstract algebra, we are concerned with sets on whose elements we can operate algebraically; that is, we can combine two elements of the set, perhaps in several ways, to obtain a third element of the set. These operations are subject to specific rules, which define the nature of the set. By convention, the notation for the two principal classes of operations on set elements is usually the same as the notation for addition and multiplication on ordinary numbers. However, it is important to note that, in abstract algebra, we are not limited to ordinary arithmetical operations. All this should become clear as we proceed.

Groups

A **group** G , sometimes denoted by $\{G, \cdot\}$ is a set of elements with a binary operation, denoted by \cdot , that associates to each ordered pair (a, b) of elements in G an element $(a \cdot b)$ in G , such that the following axioms are obeyed: ^[1]

^[1] The operator \cdot is generic and can refer to addition, multiplication, or some other mathematical operation.

- (A1)** Closure: If a and b belong to G , then $a \cdot b$ is also in G .
- (A2)** Associative: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all a, b, c in G .
- (A3)** Identity element: There is an element e in G such that $a \cdot e = e \cdot a = a$ for all a in G .
- (A4)** Inverse element: For each a in G there is an element a' in G such that $a \cdot a' = a' \cdot a = e$.

Let N_n denote a set of n distinct symbols that, for convenience, we represent as $\{1, 2, \dots, n\}$. A permutation of n distinct symbols is a one-to-one mapping from N_n to N_n . Define S_n to be the set of all permutations of n distinct symbols. Each element of S_n is represented by a permutation of the integers in $\{1, 2, \dots, n\}$. It is easy to demonstrate that S_n is a group:

[Page 98]

A1: If $\pi, \rho \in S_n$, then the composite mapping $\pi \cdot \rho$ is formed by permuting the elements of ρ according to the permutation π . For example, $\{3, 2, 1\} \cdot \{1, 3, 2\} = \{2, 3, 1\}$. Clearly, $\pi \cdot \rho \in S_n$.

A2: The composition of mappings is also easily seen to be associative.

A3: The identity mapping is the permutation that does not alter the order of the n elements. For S_n , the identity element is $\{1, 2, \dots, n\}$.

A4: For any $\pi \in S_n$, the mapping that undoes the permutation defined by π is the inverse element for π . There will always be such an inverse. For example $\{2, 3, 1\} \cdot \{3, 1, 2\} = \{1, 2, 3\}$

If a group has a finite number of elements, it is referred to as a **finite group**, and the **order** of the group is equal to the number of elements in the group. Otherwise, the group is an **infinite group**.

A group is said to be **abelian** if it satisfies the following additional condition:

(A5) Commutative: $a \cdot b = b \cdot a$ for all a, b in G .

The set of integers (positive, negative, and 0) under addition is an abelian group. The set of nonzero real numbers under multiplication is an abelian group. The set S_n from the preceding example is a group but not an abelian group for $n > 2$.

When the group operation is addition, the identity element is 0; the inverse element of a is $-a$; and subtraction is defined with the following rule: $a - b = a + (-b)$.

Cyclic Group

We define exponentiation within a group as repeated application of the group operator, so that $a^3 = a \cdot a \cdot a$. Further, we define $a^0 = e$, the identity element; and $a^{-n} = (a^{-1})^n$. A group G is **cyclic** if every element of G is a power a^k (k is an integer) of a fixed element $a \in G$. The element a is said to **generate** the group G , or to be a **generator** of G . A cyclic group is always abelian, and may be finite or infinite.

The additive group of integers is an infinite cyclic group generated by the element 1. In this case, powers are interpreted additively, so that n is the n th power of 1.

Rings

A **ring** R , sometimes denoted by $\{R, +, \cdot\}$, is a set of elements with two binary operations, called *addition* and *multiplication*,^[2] such that for all a, b, c in R the following axioms are obeyed:

^[2] Generally, we do not use the multiplication symbol, \times , but denote multiplication by the concatenation of two elements.

(A1-A5) R is an abelian group with respect to addition; that is, R satisfies axioms A1 through A5. For the case of an additive group, we denote the identity element as 0 and the inverse of a as $-a$.

(M1) Closure under multiplication: If a and b belong to R , then ab is also in R .

(M2) Associativity of multiplication: $a(bc) = (ab)c$ for all a, b, c in R .

(M3) Distributive laws: $a(b + c) = ab + ac$ for all a, b, c in R .
 $(a + b)c = ac + bc$ for all a, b, c in R .

[Page 99]

In essence, a ring is a set in which we can do addition, subtraction [$a - b = a + (-b)$], and multiplication without leaving the set.

With respect to addition and multiplication, the set of all n -square matrices over the real numbers is a ring.

A ring is said to be **commutative** if it satisfies the following additional condition:

(M4) Commutativity of multiplication: $ab = ba$ for all a, b in R .

Let S be the set of even integers (positive, negative, and 0) under the usual operations of addition and multiplication. S is a commutative ring. The set of all n -square matrices defined in the preceding example is not a commutative ring.

Next, we define an **integral domain**, which is a commutative ring that obeys the following axioms:

(M5) Multiplicative identity: There is an element 1 in R such that $a1 = 1a = a$ for all a in R .

(M6) No zero divisors: If a, b in R and $ab = 0$, then either $a = 0$ or $b = 0$.

Let S be the set of integers, positive, negative, and 0, under the usual operations of addition and multiplication. S is an integral domain.

Fields

A **field** F , sometimes denoted by $\{F, +, \times\}$, is a set of elements with two binary operations, called *addition* and *multiplication*, such that for all a, b, c in F the following axioms are obeyed:

(A1M6) F is an integral domain; that is, F satisfies axioms A1 through A5 and M1 through M6.

(M7) Multiplicative inverse: For each a in F , except 0, there is an element a^{-1} in F such that $aa^{-1} = (a^{-1})a = 1$.

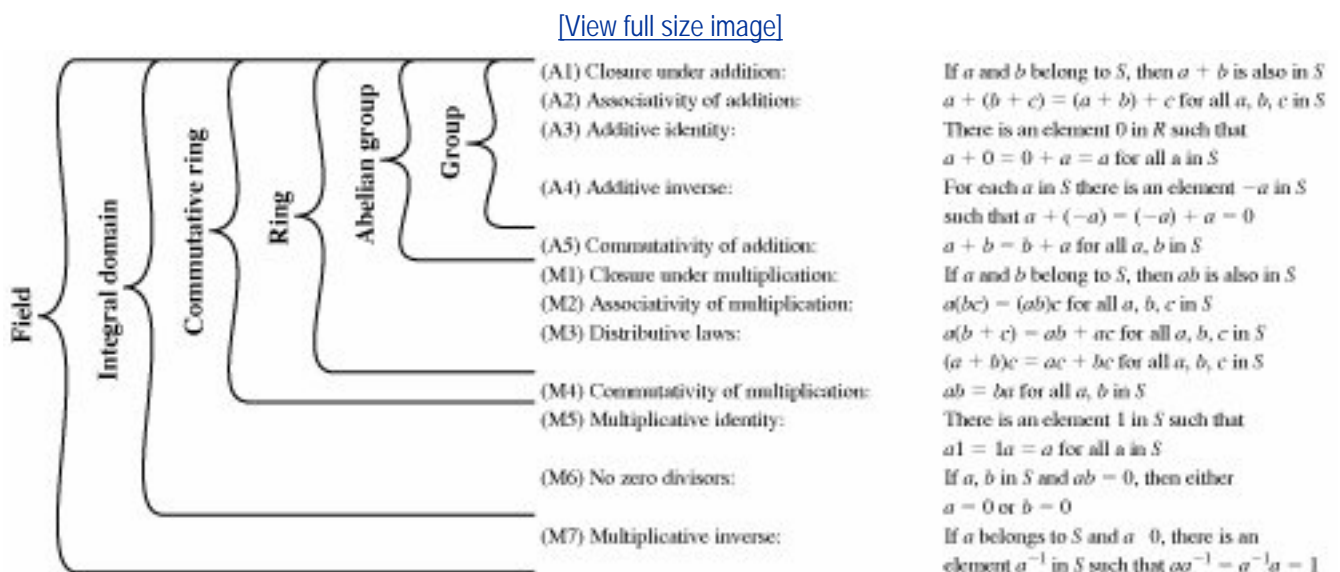
In essence, a field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set. Division is defined with the following rule: $a/b = a(b^{-1})$.

Familiar examples of fields are the rational numbers, the real numbers, and the complex numbers. Note that the set of all integers is not a field, because not every element of the set has a multiplicative inverse; in fact, only the elements 1 and -1 have multiplicative inverses in the integers.

[Figure 4.1](#) summarizes the axioms that define groups, rings, and fields.

[Page 100]

Figure 4.1. Group, Ring, and Field



4.2. Modular Arithmetic

Given any positive integer n and any nonnegative integer a , if we divide a by n , we get an integer quotient q and an integer remainder r that obey the following relationship:

Equation 4-1

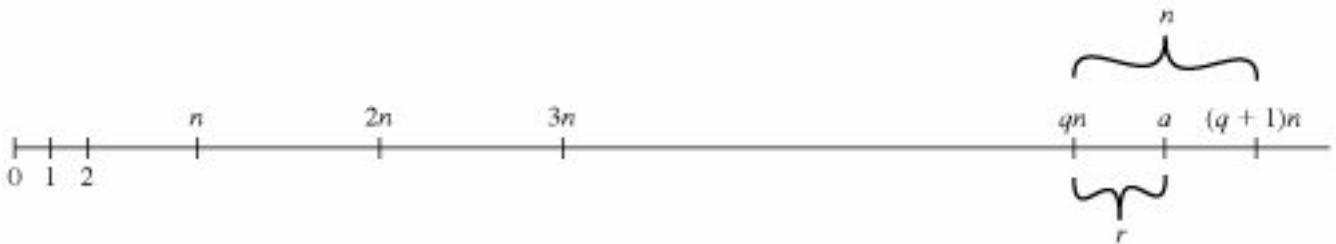
$$a = qn + r \quad 0 \leq r < n; q = \lfloor a/n \rfloor$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x .

Figure 4.2 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**.

Figure 4.2. The Relationship $a = qn + r, 0 \leq r < n$

[\[View full size image\]](#)



$a = 11;$	$n = 7;$	$11 = 1 \times 7 + 4;$	$r = 4$	$q = 1$
$a = -11;$	$n = 7;$	$-11 = (-2) \times 7 + 3;$	$r = 3$	$q = -2$

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 . The integer n is called the **modulus**. Thus, for any integer a , we can always write:

$$a = \lfloor a/n \rfloor \times n + (a \bmod n)$$

$11 \bmod 7 = 4;$	$-11 \bmod 7 = 3$
-------------------	-------------------

Two integers a and b are said to be **congruent modulo n** , if $(a \bmod n) = (b \bmod n)$. This is written as $a \equiv b \pmod{n}$.^[3]

^[3] We have just used the operator \bmod in two different ways: first as a binary operator that produces a remainder, as in the expression $a \bmod b$; second as a congruence relation that shows the equivalence of two integers, as in the expression To distinguish the two uses, the \bmod term is enclosed in parentheses for a congruence relation; this is common but not universal in the literature. See Appendix D for a further discussion.

$73 \equiv 4 \pmod{23};$	$21 \equiv -9 \pmod{10}$
--------------------------	--------------------------

[Page 102]

Divisors

We say that a nonzero b 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 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 integers g_1 .

If $b|h$, then h is of the form $h = b \times h_1$ for some integers h_1 .

So

$$mg + nh = mbg_1 + nbh_1 = b \times (mg_1 + nh_1)$$

and therefore b divides $mg + nh$.

$$b = 7; g = 14; h = 63; m = 3; n = 2.$$

$$7|14 \text{ and } 7|63. \text{ To show: } 7|(3 \times 14 + 2 \times 63)$$

$$\text{We have } (3 \times 14 + 2 \times 63) = 7(3 \times 2 + 2 \times 9)$$

$$\text{And it is obvious that } 7|(7(3 \times 2 + 2 \times 9))$$

Note that if $a \equiv 0 \pmod{n}$, then $n|a$.

Properties of Congruences

Congruences have the following properties:

1.

$$a \equiv b \pmod{n} \text{ if } n|(a - b).$$

2.

$$a \equiv b \pmod{n} \text{ implies } b \equiv a \pmod{n}.$$

3.

$$a \equiv b \pmod{n} \text{ and } b \equiv c \pmod{n} \text{ imply } a \equiv c \pmod{n}.$$

To demonstrate the first point, if $n|(a - b)$, then $(a - b) = kn$ for some k . So we can write $a = b + kn$. Therefore, $(a \pmod{n}) = (\text{remainder when } b + kn \text{ is divided by } n) = (\text{remainder when } b \text{ is divided by } n) = (b \pmod{n})$

$23 \equiv 8 \pmod{5}$	because	$23 - 8 = 15 = 5 \times 3$
$11 \equiv 5 \pmod{8}$	because	$11 - 5 = 6 = 8 \times (-1) + 6$
$81 \equiv 0 \pmod{27}$	because	$81 - 0 = 81 = 27 \times 3$

The remaining points are as easily proved.

Modular Arithmetic Operations

Note that, by definition ([Figure 4.2](#)), the (\pmod{n}) operator maps all integers into the set of integers $\{0, 1, \dots, n-1\}$.

$1, \dots, (n-1)\}$. This suggests the question: Can we perform arithmetic operations within the confines of this set? It turns out that we can; this technique is known as [modular arithmetic](#).

Modular arithmetic exhibits the following properties:

1.

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$$

2.

$$[(a \bmod n) (b \bmod n)] \bmod n = (a b) \bmod n$$

3.

$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

We demonstrate the first property. Define $(a \bmod n) = r_a$ and $(b \bmod n) = r_b$. Then we can write $a = r_a + jn$ for some integer j and $b = r_b + kn$ for some integer k . Then

$$(a + b) \bmod n = (r_a + jn + r_b + kn) \bmod n$$

$$= (r_a + r_b + (k + j)n) \bmod n$$

$$= (r_a + r_b) \bmod n$$

$$= [(a \bmod n) + (b \bmod n)] \bmod n$$

The remaining properties are as easily proved. Here are examples of the three properties:

$$11 \bmod 8 = 3; 15 \bmod 8 = 7$$

$$[(11 \bmod 8) + (15 \bmod 8)] \bmod 8 = 10 \bmod 8 = 2$$

$$(11 + 15) \bmod 8 = 26 \bmod 8 = 2$$

$$[(11 \bmod 8) (15 \bmod 8)] \bmod 8 = 4 \bmod 8 = 4$$

$$(11 \cdot 15) \bmod 8 = 165 \bmod 8 = 5$$

$$[(11 \bmod 8) \times (15 \bmod 8)] \bmod 8 = 21 \bmod 8 = 5$$

$$(11 \times 15) \bmod 8 = 165 \bmod 8 = 5$$

Exponentiation is performed by repeated multiplication, as in ordinary arithmetic. (We have more to say about exponentiation in [Chapter 8](#).)

To find $11^7 \pmod{13}$, we can proceed as follows:

$$11^2 = 121 \equiv 4 \pmod{13}$$

$$11^4 = (11^2)^2 \equiv 4^2 \equiv 3 \pmod{13}$$

$$11^7 \equiv 11 \times 4 \times 3 \equiv 132 \equiv 2 \pmod{13}$$

Thus, the rules for ordinary arithmetic involving addition, subtraction, and multiplication carry over into modular arithmetic.

[Table 4.1](#) 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. Both matrices are symmetric about the main diagonal, in conformance to the commutative property of addition and multiplication. As in ordinary addition, there is an additive inverse, or negative, to each integer in modular arithmetic. In this case, the negative of an integer x is the integer y such that $(x + y) \pmod{8} = 0$. To find the additive inverse of an integer in the left-hand column, scan across the corresponding row of the matrix to find the value 0; the integer at the top of that column is the additive inverse; thus $(2 + 6) \pmod{8} = 0$. Similarly, the entries in the multiplication table are straightforward. In ordinary arithmetic, there is a multiplicative inverse, or reciprocal, to each integer. In modular arithmetic mod 8, the multiplicative inverse of x is the integer y such that $(x \times y) \pmod{8} = 1 \pmod{8}$. Now, to find the multiplicative inverse of an integer from the multiplication table, scan across the matrix in the row for that integer to find the value 1; the integer at the top of that column is the multiplicative inverse; thus $(3 \times 3) \pmod{8} = 1$. Note that not all integers mod 8 have a multiplicative inverse; more about that later.

Table 4.1. Arithmetic Modulo 8

[\[View full size image\]](#)

+	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

	w	$-w$	w^{-1}
0	0	0	—
1	1	7	1
2	2	6	—
3	3	5	3
4	4	4	—
5	5	3	5
6	6	2	—
7	7	1	7

(c) Additive and multiplicative inverses 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**, or **residue classes** modulo n . To be more precise, each integer in Z_n represents a residue class. We can label the residue classes modulo n as $[0], [1], [2], \dots, [n-1]$, where

$$[r] = \{a : a \text{ is an integer, } a \equiv r \pmod{n}\}$$

The residue classes modulo 4 are	
[0]	= { ..., 16, 12, 8, 4, 0, 4, 8, 12, 16, ... }
[1]	= { ..., 15, 11, 7, 3, 1, 5, 9, 13, 17, ... }
[2]	= { ..., 14, 10, 6, 2, 2, 6, 10, 14, 18, ... }
[3]	= { ..., 13, 9, 5, 1, 3, 7, 11, 15, 19, ... }

Of all the integers in a residue class, the smallest nonnegative integer is the one usually used to represent the residue class. Finding the smallest nonnegative integer to which k is congruent modulo n is called **reducing k modulo n** .

If we perform modular arithmetic within Z_n , the properties shown in [Table 4.2](#) hold for integers in Z_n . Thus, Z_n is a commutative ring with a multiplicative identity element ([Figure 4.1](#)).

Table 4.2. Properties of Modular Arithmetic 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 laws	$[w + (x \times y)] \bmod n = [(w \times x) + (w \times y)] \bmod n$ $[w \times (x + y)] \bmod n = [(w + x) \times (w + y)] \bmod n$
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 $w + z \equiv 0 \pmod n$

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:

Equation 4-2

if $(a + b) \equiv (a + c) \pmod n$ then $b \equiv c \pmod n$

$$(5 + 23) \equiv (5 + 7)(\text{mod } 8); 23 \equiv 7 (\text{mod } 8)$$

[Equation \(4.2\)](#) is consistent with the existence of an additive inverse. Adding the additive inverse of a to both sides of [Equation \(4.2\)](#), we have:

$$((a) + a + b) \equiv ((a) + a + c)(\text{mod } n)$$

$$b \equiv c (\text{mod } n)$$

However, the following statement is true only with the attached condition:

Equation 4-3

if $(a \times b) \equiv (a \times c)(\text{mod } n)$ **then** $b \equiv c (\text{mod } n)$ **if** a is relatively prime to n

where the term *relatively prime* is defined as follows: two integers are **relatively prime** if their only common positive integer factor is 1. Similar to the case of [Equation \(4.2\)](#), we can say that [Equation \(4.3\)](#) is consistent with the existence of a multiplicative inverse. Applying the multiplicative inverse of a to both sides of [Equation \(4.2\)](#), we have:

$$((a^{-1})ab) \equiv ((a^{-1})ac)(\text{mod } n)$$

$$b \equiv c (\text{mod } n)$$

To see this, consider an example in which the condition of [Equation \(4.3\)](#) does not hold. The integers 6 and 8 are not relatively prime, since they have the common factor 2. We have the following:

$$6 \times 3 = 18 \equiv 2 (\text{mod } 8)$$

$$6 \times 7 = 42 \equiv 2 (\text{mod } 8)$$

$$\text{Yet } 3 \not\equiv 7 (\text{mod } 8).$$

The reason for this strange result is that for any general modulus n , a multiplier a that is applied in turn to the integers 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 integer in Z_8 maps into the same residue. Specifically, $6 \times 0 \pmod 8 = 6 \times 4 \pmod 8$; $6 \times 1 \pmod 8 = 6 \times 5 \pmod 8$; 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 $n = 8$, whose only common factor is 1,

Z_8	0	1	2	3	4	5	6	7
Multiply by 6	0	5	10	15	20	25	30	35
Residues	0	5	2	7	4	1	6	3

The line of residues contains all the integers in Z_8 , in a different order.

In general, an integer has a multiplicative inverse in Z_n if that integer is relatively prime to n . [Table 4.1c](#) shows that the integers 1, 3, 5, and 7 have a multiplicative inverse in Z_8 , but 2, 4, and 6 do not.

4.3. The Euclidean Algorithm

One of the basic techniques of number theory is the Euclidean algorithm, which is a simple procedure for determining the greatest common divisor of two positive integers.

Greatest Common Divisor

Recall that nonzero b is defined to be a divisor of a if $a = mb$ for some m , where a , b , and m are integers. 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 $\gcd(a, 0) = |a|$.

We stated that two integers a and b are relatively prime if their only common positive integer 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 positive divisors of 8 are 1, 2, 4, and 8, and the positive divisors of 15 are 1, 3, 5, and 15, so 1 is the only integer on both lists.

Finding the Greatest Common Divisor

The Euclidean algorithm is based on the following theorem: For any nonnegative integer a and any positive integer b ,

Equation 4-4

$$\gcd(a, b) = \gcd(b, a \bmod b)$$

$$\gcd(55, 22) = \gcd(22, 55 \bmod 22) = \gcd(22, 11) = 11$$

[Page 108]

To see that [Equation \(4.4\)](#) works, let $d = \gcd(a, b)$. Then, by the definition of \gcd , $d|a$ and $d|b$. For any positive integer b , a can be expressed in the form

$$a = kb + r \equiv r \pmod{b}$$

$$a \bmod b = r$$

with k, r integers. 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 pair is the same as the \gcd of the other pair, proving the theorem.

[Equation \(4.4\)](#) can be used repetitively to determine the greatest common divisor.

$$\gcd(18, 12) = \gcd(12, 6) = \gcd(6, 0) = 6$$

$$\gcd(11, 10) = \gcd(10, 1) = \gcd(1, 0) = 1$$

The Euclidean algorithm makes repeated use of [Equation \(4.4\)](#) to determine the greatest common divisor, as follows. The algorithm assumes $a > b > 0$. It is acceptable to restrict the algorithm to positive integers because $\gcd(a, b) = \gcd(|a|, |b|)$.

EUCLID(a, b)

1. $A \leftarrow a; B \leftarrow b$
2. **if** $B = 0$ **return** $A = \gcd(a, b)$
3. $R = A \bmod B$
4. $A \leftarrow B$
5. $B \leftarrow R$
6. **goto** 2

The algorithm has the following progression:

$$A_1 = B_1 \times Q_1 + R_1$$

$$A_2 = B_2 \times Q_2 + R_2$$

$$A_3 = B_3 \times Q_3 + R_3$$

$$A_4 = B_4 \times Q_4 + R_4$$

[Page 109]

To find gcd(1970, 1066)		
1970	= 1 x 1066 + 904	gcd(1066, 904)
1066	= 1 x 904 + 162	gcd(904, 162)
904	= 5 x 162 + 94	gcd(162, 94)
162	= 1 x 94 + 68	gcd(94, 68)
94	= 1 x 68 + 26	gcd(68, 26)
68	= 2 x 26 + 16	gcd(26, 16)
26	= 1 x 16 + 10	gcd(16, 10)
16	= 1 x 10 + 6	gcd(10, 6)
10	= 1 x 6 + 4	gcd(6, 4)
6	= 1 x 4 + 2	gcd(4, 2)
4	= 2 x 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 B divides A? If not, we would get an endless sequence of positive integers, each one strictly smaller than the one before, and this is clearly impossible.

4.4. Finite Fields of The Form GF(p)

In Section 4.1, we defined a field as a set that obeys all of the axioms of Figure 4.1 and gave some examples of infinite fields. Infinite fields are not of particular interest in the context of cryptography. However, finite fields play a crucial role in many cryptographic algorithms. It can be shown that the order of a finite field (number of elements in the field) must be a power of a prime p^n , where n is a positive integer. We discuss prime numbers in detail in Chapter 8. Here, we need only say that a prime number is an integer whose only positive integer factors are itself and 1. That is, the only positive integers that are divisors of p are p and 1.

The finite field of order p^n is generally written $GF(p^n)$; stands for Galois field, in honor of the mathematician who first studied finite fields. Two special cases are of interest for our purposes. For $n = 1$, we have the finite field $GF(p)$; this finite field has a different structure than that for finite fields with $n > 1$ and is studied in this section. In Section 4.6, we look at finite fields of the form $GF(2^n)$.

Finite Fields of Order p

For a given prime, p , the finite field of order p , $GF(p)$ is defined as the set Z_p of integers $\{0, 1, \dots, p - 1\}$, together with the arithmetic operations modulo p .

Recall that we showed in Section 4.2 that the set Z_n of integers $\{0, 1, \dots, n - 1\}$, together with the arithmetic operations modulo n , is a commutative ring (Table 4.2). We further observed that any integer in Z_n has a multiplicative inverse if and only if that integer is relatively prime to n [see discussion of Equation (4.3)].^[4] If n is prime, then all of the nonzero integers in Z_n are relatively prime to n , and therefore there exists a multiplicative inverse for all of the nonzero integers in Z_n . Thus, we can add the following properties to those listed in Table 4.2 for Z_p :

^[4] As stated in the discussion of Equation (4.3), two integers are **relatively prime** if their only common positive integer factor is 1.

Multiplicative inverse (w^{-1})	For each $w \in Z_p, w \neq 0$, there exists a $z \in Z_p$ such that $w \times z \equiv 1 \pmod{p}$
-------------------------------------	--

Because w is relatively prime to p , if we multiply all the elements of Z_p by w , the resulting residues are all of the elements of Z_p permuted. Thus, exactly one of the residues has the value 1. Therefore, there is some integer Z_p in that, when multiplied by w , yields the residue 1. That integer is the multiplicative inverse of w , designated w^{-1} . Therefore, Z_p is in fact a finite field. Further, [Equation \(4.3\)](#) is consistent with the existence of a multiplicative inverse and can be rewritten without the condition:

Equation 4-5

if $(a \times b) \equiv (a \times c) \pmod{p}$ then $b \equiv c \pmod{p}$

Multiplying both sides of [Equation \(4.5\)](#) by the multiplicative inverse of a , we have:

$$((a^{-1}) \times a \times b) \equiv ((a^{-1}) \times a \times c) \pmod{p}$$

$$b \equiv c \pmod{p}$$

The simplest finite field is GF(2). Its arithmetic operations are easily summarized:

$+$	0	1	\times	0	1	w	$-w$	w^{-1}
0	0	1	0	0	0	0	0	—
1	1	0	1	0	1	1	1	1

Addition Multiplication Inverses

In this case, addition is equivalent to the exclusive-OR (XOR) operation, and multiplication is equivalent to the logical AND operation.

[Table 4.3](#) shows $GF(7)$. This is a field of order 7 using modular arithmetic modulo 7. As can be seen, it satisfies all of the properties required of a field ([Figure 4.1](#)). Compare this table with [Table 4.1](#). In the latter case, we see that the set Z_8 using modular arithmetic modulo 8, is not a field. Later in this chapter, we show how to define addition and multiplication operations on Z_8 in such a way as to form a finite field.

Table 4.3. Arithmetic in $GF(7)$

(This item is displayed on page 111 in the print version)

[View full size image](#)

+	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^{-1}
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

Finding the Multiplicative Inverse in $GF(p)$

It is easy to find the multiplicative inverse of an element in $GF(p)$ for small values of p . You simply construct a multiplication table, such as shown in [Table 4.3b](#), and the desired result can be read directly. However, for large values of p , this approach is not practical.

If $\gcd(m, b) = 1$, then b has a multiplicative inverse modulo m . That is, for positive integer $b < m$, there exists a $b^{-1} < m$ such that $bb^{-1} = 1 \pmod{m}$. The Euclidean algorithm can be extended so that, in addition to finding $\gcd(m, b)$, if the gcd is 1, the algorithm returns the multiplicative inverse of b .

EXTENDED EUCLID(m, b)

1. $(A1, A2, A3) \leftarrow (1, 0, m); (B1, B2, B3) \leftarrow (0, 1, b)$
2. **if** $B3 = 0$ **return** $A3 = \text{gcd}(m, b)$; no inverse
3. **if** $B3 = 1$ **return** $B3 = \text{gcd}(m, b)$; $B2 = b^{-1} \text{ mod } m$

4. $Q = \left\lfloor \frac{A3}{B3} \right\rfloor$

5. $(T1, T2, T3) \leftarrow (A1 - QB1, A2 - QB2, A3 - QB3)$
6. $(A1, A2, A3) \leftarrow (B1, B2, B3)$
7. $(B1, B2, B3) \leftarrow (T1, T2, T3)$
8. **goto** 2

Throughout the computation, the following relationships hold:

$$mT1 + bT2 = T3 \quad mA1 + bA2 = A3 \quad mB1 + bB2 = B3$$

To see that this algorithm correctly returns $\text{gcd}(m, b)$, note that if we equate A and B in the Euclidean algorithm with $A3$ and $B3$ in the extended Euclidean algorithm, then the treatment of the two variables is identical. At each iteration of the Euclidean algorithm, A is set equal to the previous value of B and B is set equal to the previous value of $A \text{ mod } B$. Similarly, at each step of the extended Euclidean algorithm, $A3$ is set equal to the previous value of $B3$, and $B3$ is set equal to the previous value of $A3$ minus the integer quotient of $A3$ multiplied by $B3$. This latter value is simply the remainder of $A3$ divided by $B3$, which is $A3 \text{ mod } B3$.

Note also that if $\text{gcd}(m, b) = 1$, then on the final step we would have $B3 = 0$ and $A3 = 1$. Therefore, on the preceding step, $B3 = 1$. But if $B3 = 1$, then we can say the following:

$$mB1 + bB2 = B3$$

$$mB1 + bB2 = 1$$

$$bB2 = 1 - mB1$$

$$bB2 \equiv 1 \pmod{m}$$

And $B2$ is the multiplicative inverse of b , modulo m .

[Table 4.4](#) is an example of the execution of the algorithm. It shows that $\gcd(1759, 550) = 1$ and that the multiplicative inverse of 550 is 355; that is, $550 \times 355 \equiv 1 \pmod{1759}$.

Table 4.4. Finding the Multiplicative Inverse of 550 in $GF(1759)$

Q	A1	A2	A3	B1	B2	B3
	1	0	1759	0	1	550
3	0	1	550	1	3	109
5	1	3	109	5	16	5
21	5	16	5	106	339	4
1	106	339	4	111	355	1

For a more detailed proof of this algorithm, see [\[KNUT97\]](#).

Summary

In this section, we have shown how to construct a finite field of order p , where p is prime. Specifically, we defined $GF(p)$ with the following properties:

1.

$GF(p)$ consists of p elements.

2.

The binary operations $+$ and \times are defined over the set. The operations of addition, subtraction, multiplication, and division can be performed without leaving the set. Each element of the set other than 0 has a multiplicative inverse.

We have shown that the elements of $GF(p)$ are the integers $\{0, 1, \dots, p\}$ and that the arithmetic operations are addition and multiplication mod p .

4.5. Polynomial Arithmetic

Before pursuing our discussion of finite fields, we need to introduce the interesting subject of polynomial arithmetic. We are concerned with polynomials in a single variable x , and we can distinguish three classes of polynomial arithmetic:

- Ordinary polynomial arithmetic, using the basic rules of algebra
- Polynomial arithmetic in which the arithmetic on the coefficients is performed modulo p ; that is, the coefficients are in $\text{GF}(p)$
- Polynomial arithmetic in which the coefficients are in $\text{GF}(p)$, and the polynomials are defined modulo a polynomial $m(x)$ whose highest power is some integer n

This section examines the first two classes, and the next section covers the last class.

Ordinary Polynomial Arithmetic

A **polynomial** of degree n (integer $n \geq 0$) is an expression of the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^n a_i x^i$$

where the a_i are elements of some designated set of numbers S , called the **coefficient set**, and $a_n \neq 0$. We say that such polynomials are defined over the coefficient set S .

A zeroth-degree polynomial is called a **constant polynomial** and is simply an element of the set of coefficients. An n th-degree polynomial is said to be a **monic polynomial** if $a_n = 1$.

In the context of abstract algebra, we are usually not interested in evaluating a polynomial for a particular value of x [e.g., $f(7)$]. To emphasize this point, the variable x is sometimes referred to as the **indeterminate**.

Polynomial arithmetic includes the operations of addition, subtraction, and multiplication. These operations are defined in a natural way as though the variable x was an element of S . Division is similarly defined, but requires that S be a field. Examples of fields include the real numbers, rational numbers, and \mathbb{Z}_p for p prime. Note that the set of all integers is not a field and does not support polynomial division.

Addition and subtraction are performed by adding or subtracting corresponding coefficients. Thus, if

$$f(x) = \sum_{i=0}^n a_i x^i; \quad g(x) = \sum_{i=0}^m b_i x^i; \quad n \geq m$$

then addition is defined as

$$f(x) + g(x) = \sum_{i=0}^m (a_i + b_i)x^i + \sum_{i=m+1}^n a_i x^i$$

[Page 114]

and multiplication is defined as

$$f(x) \times g(x) = \sum_{i=0}^{n+m} c_i x^i$$

where

$$c_k = a_0 b_{k1} + a_1 b_{k1} + \dots + a_{k1} b_1 + a_k b_0$$

In the last formula, we treat a_i as zero for $i > n$ and b_i as zero for $i > m$. Note that the degree of the product is equal to the sum of the degrees of the two polynomials.

As an example, let $f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 x + 1$, where S is the set of integers. Then

$$f(x) + g(x) = x^3 + 2x^2 x + 3$$

$$f(x) g(x) = x^3 + x + 1$$

$$f(x) \times g(x) = x^5 + 3x^2 2x + 2$$

[Figures 4.3a](#) through [4.3c](#) show the manual calculations. We comment on division subsequently.

Figure 4.3. Examples of Polynomial Arithmetic

$$\begin{array}{r} x^3 + x^2 + 2 \\ + (x^2 - x + 1) \\ \hline x^3 + 2x^2 - x + 3 \end{array}$$

(a) Addition

$$\begin{array}{r} x^3 + x^2 + 2 \\ - (x^2 - x + 1) \\ \hline x^3 + x + 1 \end{array}$$

(b) Subtraction

$$\begin{array}{r} x^3 + x^2 + 2 \\ \times (x^2 - x + 1) \\ \hline x^3 + x^2 + 2 \\ - x^4 - x^3 - 2x \\ \hline x^5 + x^4 + 2x^2 \end{array}$$

(c) Multiplication

$$\begin{array}{r} x^2 - x + 1 \overline{) x^3 + x^2 + 2} \\ \underline{x^3 + x^2 + x} \\ 2x^2 - x + 2 \\ \underline{2x^2 - 2x + 2} \\ x \end{array}$$

(d) Division

Polynomial Arithmetic with Coefficients in \mathbb{Z}_p

Let us now consider polynomials in which the coefficients are elements of some field F . We refer to this as a polynomial over the field F . In that case, it is easy to show that the set of such polynomials is a ring, referred to as a **polynomial ring**. That is, if we consider each distinct polynomial to be an element of the set, then that set is a ring. ^[5]

^[5] In fact, the set of polynomials whose coefficients are elements of a commutative ring forms a polynomial ring, but that is of no interest in the present context.

When polynomial arithmetic is performed on polynomials over a field, then division is possible. Note that this does not mean that *exact division* is possible. Let us clarify this distinction. Within a field, given two elements a and b , the quotient a/b is also an element of the field. However, given a ring R that is not a field, in general division will result in both a quotient and a remainder; this is not exact division.

Consider the division $5/3$ within a set S . If S is the set of rational numbers, which is a field, then the result is simply expressed as $5/3$ and is an element of S . Now suppose that S is the field Z_7 . In this case, we calculate (using [Table 4.3c](#)):

$$5/3 = (5 \times 3^1) \bmod 7 = (5 \times 5) \bmod 7 = 4$$

which is an exact solution. Finally, suppose that S is the set of integers, which is a ring but not a field. Then $5/3$ produces a quotient of 1 and a remainder of 2:

$$5/3 = 1 + 2/3$$

$$5 = 1 \times 3 + 2$$

Thus, division is not exact over the set of integers.

Now, if we attempt to perform polynomial division over a coefficient set that is not a field, we find that division is not always defined.

If the coefficient set is the integers, then $(5x^2)/(3x)$ does not have a solution, because it would require a coefficient with a value of $5/3$, which is not in the coefficient set. Suppose that we perform the same polynomial division over Z_7 . Then we have $(5x^2)/(3x) = 4x$ which is a valid polynomial over Z_7 .

However, as we demonstrate presently, even if the coefficient set is a field, polynomial division is not necessarily exact. In general, division will produce a quotient and a remainder:

Equation 4-6

$$\frac{f(x)}{g(x)} = q(x) + \frac{r(x)}{g(x)}$$

$$f(x) = q(x)g(x) + r(x)$$

If the degree of $f(x)$ is n and the degree of $g(x)$ is m , ($m \geq n$), then the degree of the quotient $q(x)$ is $n - m$ and the degree of the remainder is at most $m - 1$. With the understanding that remainders are allowed, we can say that polynomial division is possible if the coefficient set is a field.

In an analogy to integer arithmetic, we can write $f(x) \bmod g(x)$ for the remainder $r(x)$ in [Equation \(4.6\)](#). That is, $r(x) = f(x) \bmod g(x)$. If there is no remainder [i.e., $r(x) = 0$], then we can say $g(x)$ **divides** $f(x)$, written as $g(x) | f(x)$; equivalently, we can say that $g(x)$ is a **factor** of $f(x)$ or $g(x)$ is a **divisor** of $f(x)$.

For the preceding example and $[f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 x + 1]$, $f(x)/g(x)$ produces a quotient of $q(x) = x + 2$ and a remainder $r(x) = x$ as shown in [Figure 4.3d](#). This is easily verified by noting that

$$\begin{aligned} q(x)g(x) + r(x) &= (x + 2)(x^2 x + 1) + x = (x^3 + x^2 x + 2) + x \\ &= x^3 + x^2 + 2 = f(x) \end{aligned}$$

For our purposes, polynomials over $\text{GF}(2)$ are of most interest. Recall from [Section 4.4](#) that in $\text{GF}(2)$, addition is equivalent to the XOR operation, and multiplication is equivalent to the logical AND operation. Further, addition and subtraction are equivalent mod 2: $1 + 1 = 1 \ 1 = 0$; $1 + 0 = 1 \ 0 = 1$; $0 + 1 = 0 \ 1 = 1$.

[Figure 4.4](#) shows an example of polynomial arithmetic over $\text{GF}(2)$. For $f(x) = (x^7 + x^5 + x^4 + x^3 + x + 1)$ and $g(x) = (x^3 + x + 1)$, the figure shows $f(x) + g(x)$; $f(x) g(x)$; $f(x) \times g(x)$; and $f(x)/g(x)$. Note that $g(x) | f(x)$

Figure 4.4. Examples of Polynomial Arithmetic over $\text{GF}(2)$

$$\begin{array}{r}
 x^7 + x^5 + x^4 + x^3 + x + 1 \\
 + (x^3 + x + 1) \\
 \hline
 x^7 + x^5 + x^4
 \end{array}$$

(a) Addition

$$\begin{array}{r}
 x^7 + x^5 + x^4 + x^3 + x + 1 \\
 - (x^3 + x + 1) \\
 \hline
 x^7 + x^5 + x^4
 \end{array}$$

(b) Subtraction

$$\begin{array}{r}
 x^7 + x^5 + x^4 + x^3 + x + 1 \\
 \times (x^3 + x + 1) \\
 \hline
 x^7 + x^5 + x^4 + x^3 + x + 1 \\
 x^8 + x^6 + x^5 + x^4 + x^2 + x \\
 \hline
 x^{10} + x^8 + x^7 + x^6 + x^4 + x^3 \\
 \hline
 x^{10} + x^4 + x^2 + 1
 \end{array}$$

(c) Multiplication

$$\begin{array}{r}
 x^4 + 1 \\
 \hline
 x^3 + x + 1 \overline{) x^7 + x^5 + x^4 + x^3 + x + 1} \\
 \underline{x^7 + x^5 + x^4} \\
 x^3 + x + 1 \\
 \underline{x^3 + x + 1} \\
 0
 \end{array}$$

(d) Division

A polynomial $f(x)$ over a field F is called **irreducible** if and only if $f(x)$ cannot be expressed as a product of two polynomials, both over F , and both of degree lower than that of $f(x)$. By analogy to integers, an irreducible polynomial is also called a **prime polynomial**.

The polynomial ^[6] $f(x) = x^4 + 1$ over $\text{GF}(2)$ is reducible, because $x^4 + 1 = (x + 1)(x^3 + x^2 + x + 1)$

^[6] In the remainder of this chapter, unless otherwise noted, all examples are of polynomials over $\text{GF}(2)$.

Consider the polynomial $f(x) = x^3 + x + 1$. It is clear by inspection that x is not a factor of $f(x)$. We easily show that $x + 1$ is not a factor of $f(x)$:

$$\begin{array}{r}
 x^2 + x \\
 \hline
 x + 1 \overline{) x^3 + x + 1} \\
 \underline{x^3 + x^2} \\
 x^2 + x \\
 \underline{x^2 + x} \\
 1
 \end{array}$$

Thus $f(x)$ has no factors of degree 1. But it is clear by inspection that if $f(x)$ is reducible, it must have one factor of degree 2 and one factor of degree 1. Therefore, $f(x)$ is irreducible.

Finding the Greatest Common Divisor

We can extend the analogy between polynomial arithmetic over a field and integer arithmetic by defining the greatest common divisor as follows. The polynomial $c(x)$ is said to be the greatest common divisor of $a(x)$ and $b(x)$ if

1.

$c(x)$ divides both $a(x)$ and $b(x)$;

2.

any divisor of $a(x)$ and $b(x)$ is a divisor of $c(x)$.

An equivalent definition is the following: $\gcd[a(x), b(x)]$ is the polynomial of maximum degree that divides both $a(x)$ and $b(x)$.

We can adapt the Euclidean algorithm to compute the greatest common divisor of two polynomials. The equality in [Equation \(4.4\)](#) can be rewritten as the following theorem:

Equation 4-7

$$\gcd[a(x), b(x)] = \gcd[b(x), a(x) \bmod b(x)]$$

The Euclidean algorithm for polynomials can be stated as follows. The algorithm assumes that the degree of $a(x)$ is greater than the degree of $b(x)$. Then, to find $\gcd[a(x), b(x)]$,

EUCLID[$a(x), b(x)$]

1. $A(x) \leftarrow a(x); B(x) \leftarrow b(x)$
2. **if** $B(x) = 0$ **return** $A(x) = \gcd[a(x), b(x)]$
3. $R(x) = A(x) \bmod B(x)$
4. $A(x) \leftarrow B(x)$
5. $B(x) \leftarrow R(x)$
6. **goto** 2

Find $\gcd[a(x), b(x)]$ for $a(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$ and $b(x) = x^4 + x^2 + x + 1$.

$A(x) = a(x); B(x) = b(x)$

$$\begin{array}{r}
 x^2 + x \\
 \hline
 x^4 + x^2 + x + 1 \overline{) x^6 + x^5 + x^4 + x^3 + x^2 + x + 1} \\
 \underline{x^6 + x^4 + x^3 + x^2} \\
 x^5 + x + 1 \\
 \underline{ x^5 + x^3 + x^2 + x} \\
 x^3 + x^2 + 1
 \end{array}$$

$R(x) = A(x) \bmod B(x) = x^3 + x^2 + 1$

$A(x) = x^4 + x^2 + x + 1; B(x) = x^3 + x^2 + 1$

$$\begin{array}{r}
 x^3 + x^2 + 1 \overline{) x^4 + x^3 + x^2 + x + 1} \\
 \underline{x^4 + x^3 + x} \\
 x^3 + x^2 + 1 \\
 \underline{x^3 + x^2} \\
 1
 \end{array}$$

$$R(x) = A(x) \bmod B(x) = 0$$

$$\gcd[a(x), b(x)] = A(x) = x^3 + x^2 + 1$$

Summary

We began this section with a discussion of arithmetic with ordinary polynomials. In ordinary polynomial arithmetic, the variable is not evaluated; that is, we do not plug a value in for the variable of the polynomials. Instead, arithmetic operations are performed on polynomials (addition, subtraction, multiplication, division) using the ordinary rules of algebra. Polynomial division is not allowed unless the coefficients are elements of a field.

[Page 119]

Next, we discussed polynomial arithmetic in which the coefficients are elements of $\text{GF}(p)$. In this case, polynomial addition, subtraction, multiplication, and division are allowed. However, division is not exact; that is, in general division results in a quotient and a remainder.

Finally, we showed that the Euclidean algorithm can be extended to find the greatest common divisor of two polynomials whose coefficients are elements of a field.

All of the material in this section provides a foundation for the following section, in which polynomials are used to define finite fields of order p^n .

◀ PREY

NEXT ▶

4.6. Finite Fields Of the Form $\text{GF}(2^n)$

Earlier in this chapter, we mentioned that the order of a finite field must be of the form p^n where p is a prime and n is a positive integer. In [Section 4.4](#), we looked at the special case of finite fields with order p . We found that, using modular arithmetic in \mathbb{Z}_p , all of the axioms for a field ([Figure 4.1](#)) are satisfied.

For polynomials over p^n , with $n > 1$, operations modulo p^n do not produce a field. In this section, we show what structure satisfies the axioms for a field in a set with p^n elements, and concentrate on $\text{GF}(2^n)$.

Motivation

Virtually all encryption algorithms, both symmetric and public key, involve arithmetic operations on integers. If one of the operations that is used in the algorithm is division, then we need to work in arithmetic defined over a field. For convenience and for implementation efficiency, we would also like to work with integers that fit exactly into a given number of bits, with no wasted bit patterns. That is, we wish to work with integers in the range 0 through $2^n - 1$, which fit into an n -bit word.

Suppose we wish to define a conventional encryption algorithm that operates on data 8 bits at a time and we wish to perform division. With 8 bits, we can represent integers in the range 0 through 255. However, 256 is not a prime number, so that if arithmetic is performed in \mathbb{Z}_{256} (arithmetic modulo 256), this set of integers will not be a field. The closest prime number less than 256 is 251. Thus, the set \mathbb{Z}_{251} , using arithmetic modulo 251, is a field. However, in this case the 8-bit patterns representing the integers 251 through 255 would not be used, resulting in inefficient use of storage.

As the preceding example points out, if all arithmetic operations are to be used, and we wish to represent a full range of integers in n bits, then arithmetic modulo will not work; equivalently, the set of integers modulo 2^n , for $n > 1$, is not a field. Furthermore, even if the encryption algorithm uses only addition and multiplication, but not division, the use of the set \mathbb{Z}_{2^n} is questionable, as the following example illustrates.

Suppose we wish to use 3-bit blocks in our encryption algorithm, and use only the operations of addition and multiplication. Then arithmetic modulo 8 is well defined, as shown in [Table 4.1](#). However, note that in the multiplication table, the nonzero integers do not appear an equal number of times. For example, there are only four occurrences of 3, but twelve occurrences of 4. On the other hand, as was mentioned, there are finite fields of the form $GF(2^n)$ so there is in particular a finite field of order $2^3 = 8$. Arithmetic for this field is shown in [Table 4.5](#). In this case, the number of occurrences of the nonzero integers is uniform for multiplication. To summarize,

Integer	1	2	3	4	5	6	7
Occurrences in Z_8	4	8	4	12	4	8	4
Occurrences in $GF(2^3)$	7	7	7	7	7	7	7

Table 4.5. Arithmetic in $GF(2^3)$

(This item is displayed on page 121 in the print version)

[View full size image](#)

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

(a) Addition

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

(b) Multiplication

	w	-w	w ⁻¹
0	0	—	—
1	1	1	1
2	2	2	5
3	3	3	6
4	4	4	7
5	5	5	2
6	6	6	3
7	7	7	4

(c) Additive and multiplicative inverses

For the moment, let us set aside the question of how the matrices of [Table 4.5](#) were constructed and instead make some observations.

1.

The addition and multiplication tables are symmetric about the main diagonal, in conformance to the commutative property of addition and multiplication. This property is also exhibited in [Table 4.1](#), which uses mod 8 arithmetic.

2.

All the nonzero elements defined by [Table 4.5](#) have a multiplicative inverse, unlike the case with [Table 4.1](#).

3.

The scheme defined by [Table 4.5](#) satisfies all the requirements for a finite field. Thus, we can refer to this scheme as $GF(2^3)$.

4.

For convenience, we show the 3-bit assignment used for each of the elements of $GF(2^3)$.

Intuitively, it would seem that an algorithm that maps the integers unevenly onto themselves might be cryptographically weaker than one that provides a uniform mapping. Thus, the finite fields of the form $GF(2^n)$ are attractive for cryptographic algorithms.

To summarize, we are looking for a set consisting of 2^n elements, together with a definition of addition and multiplication over the set that define a field. We can assign a unique integer in the range 0 through $2^n - 1$ to each element of the set. Keep in mind that we will not use modular arithmetic, as we have seen that this does not result in a field. Instead, we will show how polynomial arithmetic provides a means for constructing the desired field.

Modular Polynomial Arithmetic

Consider the set S of all polynomials of degree $n - 1$ or less over the field Z_p . Thus, each polynomial has the form

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

where each a_i takes on a value in the set $\{0, 1, \dots, p-1\}$. There are a total of p^n different polynomials in S .

[Page 121]

For $p = 3$ and $n = 2$, the $3^2 = 9$ polynomials in the set are		
0	x	$2x$
1	$x + 1$	$2x + 1$
2	$x + 2$	$2x + 2$
For $p = 2$ and $n = 3$, the $2^3 = 8$ the polynomials in the set are		
0	$x + 1$	$x^2 + x$
1	x^2	$x^2 + x + 1$
x	$x^2 + 1$	

With the appropriate definition of arithmetic operations, each such set S is a finite field. The definition consists of the following elements:

1.

Arithmetic follows the ordinary rules of polynomial arithmetic using the basic rules of algebra, with the following two refinements.

2.

Arithmetic on the coefficients is performed modulo p . That is, we use the rules of arithmetic for the finite field Z_p .

[Page 122]

3.

If multiplication results in a polynomial of degree greater than $n-1$, then the polynomial is reduced modulo some irreducible polynomial $m(x)$ of degree n . That is, we divide by $m(x)$ and keep the remainder. For a polynomial $f(x)$, the remainder is expressed as $r(x) = f(x) \bmod m(x)$.

The Advanced Encryption Standard (AES) uses arithmetic in the finite field $GF(2^8)$, with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. Consider the two polynomials $f(x) = x^6 + x^4 + x^2 + x + 1$ and $g(x) = x^7 + x + 1$. Then

$$f(x) + g(x) = x^6 + x^4 + x^2 + x + 1 + x^7 + x + 1$$

$$f(x) \times g(x) = x^{13} + x^{11} + x^9 + x^8 + x^7 +$$

$$x^7 + x^5 + x^3 + x^2 + x +$$

$$x^6 + x^4 + x^2 + x + 1$$

$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

$$\begin{array}{r}
 x^8 + x^4 + x^3 + x + 1 \overline{) x^{13} + x^{11} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1} \\
 \underline{x^{13} \phantom{+ x^{11} + x^9 + x^8} + x^9 + x^8 } \\
 x^{11} + x^4 + x^3 \\
 \underline{ x^{11} + x^7 + x^6 } \\
 \phantom{x^{11} +} x^7 + x^6
 \end{array}$$

Therefore, $f(x) \times g(x) \text{ mod } m(x) = x^7 + x^6 + 1$.

As with ordinary modular arithmetic, we have the notion of a set of residues in modular polynomial arithmetic. The set of residues modulo $m(x)$, an n th-degree polynomial, consists of p^n elements. Each of these elements is represented by one of the p^n polynomials of degree $m < n$.

The residue class $[x + 1]$, modulo $m(x)$, consists of all polynomials $a(x)$ such that $a(x) \in (x + 1) \pmod{m(x)}$. Equivalently, the residue class $[x + 1]$ consists of all polynomials $a(x)$ that satisfy the equality $a(x) \text{ mod } m(x) = x + 1$.

It can be shown that the set of all polynomials modulo an irreducible n th-degree polynomial $m(x)$ satisfies the axioms in [Figure 4.1](#), and thus forms a finite field. Furthermore, all finite fields of a given order are isomorphic; that is, any two finite-field structures of a given order have the same structure, but the representation, or labels, of the elements may be different.

To construct the finite field $GF(2^3)$, we need to choose an irreducible polynomial of degree 3. There are only two such polynomials: $(x^3 + x^2 + 1)$ and $(x^3 + x + 1)$. Using the latter, [Table 4.6](#) shows the addition and multiplication tables for $GF(2^3)$. Note that this set of tables has the identical structure to those of [Table 4.5](#). Thus, we have succeeded in finding a way to define a field of order 2^3 .

Table 4.6. Polynomial Arithmetic Modulo $(x^3 + x + 1)$

(This item is displayed on page 124 in the print version)

[\[View full size image\]](#)

		000	001	010	011	100	101	110	111
	+	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
000	0	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
001	1	1	0	$x+1$	x	x^2+1	x^2	x^2+x+1	x^2+x
010	x	x	$x+1$	0	1	x^2+x	x^2+x+1	x^2	x^2+1
011	$x+1$	$x+1$	x	1	0	x^2+x+1	x^2+x	x^2+1	x^2
100	x^2	x^2	x^2+1	x^2+x	x^2+x+1	0	1	x	$x+1$
101	x^2+1	x^2+1	x^2	x^2+x+1	x^2+x	1	0	$x+1$	x
110	x^2+x	x^2+x	x^2+x+1	x^2	x^2+1	x	$x+1$	0	1
111	x^2+x+1	x^2+x+1	x^2+x	x^2+1	x^2	$x+1$	x	1	0

(a) Addition

		000	001	010	011	100	101	110	111
	×	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
000	0	0	0	0	0	0	0	0	0
001	1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
010	x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
011	$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
100	x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
101	x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
110	x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
111	x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

(b) Multiplication

Finding the Multiplicative Inverse

Just as the Euclidean algorithm can be adapted to find the greatest common divisor of two polynomials, the extended Euclidean algorithm can be adapted to find the multiplicative inverse of a polynomial. Specifically, the algorithm will find the multiplicative inverse of $b(x)$ modulo $m(x)$ if the degree of $b(x)$ is less than the degree of $m(x)$ and $\gcd[m(x), b(x)] = 1$. If $m(x)$ is an irreducible polynomial, then it has no factor other than itself or 1, so that $\gcd[m(x), b(x)] = 1$. The algorithm is as follows:

EXTENDED EUCLID[$m(x), b(x)$]

1. $[A1(x), A2(x), A3(x)] \leftarrow [1, 0, m(x)]; [B1(x), B2(x), B3(x)] \leftarrow [0, 1, b(x)]$
2. **if** $B3(x) = 0$ **return** $A3(x) = \gcd[m(x), b(x)];$ no inverse
3. **if** $B3(x) = 1$ **return** $B3(x) = \gcd[m(x), b(x)];$
 $B2(x) = b(x)^{-1} \pmod{m(x)}$

4. $Q(x) = \text{quotient of } A3(x)/B3(x)$
5. $[T1(x), T2(x), T3(x)] \leftarrow [A1(x) - Q(x)B1(x), A2(x) - Q(x)B2(x), A3(x) - QB3(x)]$
6. $[A1(x), A2(x), A3(x)] \leftarrow [B1(x), B2(x), B3(x)]$
7. $[B1(x), B2(x), B3(x)] \leftarrow [T1(x), T2(x), T3(x)]$
8. goto 2

Table 4.7 shows the calculation of the multiplicative inverse of $(x^7 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$. The result is that $(x^7 + x + 1)^{-1} = (x^7)$. That is, $(x^7 + x + 1)(x^7) \equiv 1 \pmod{(x^8 + x^4 + x^3 + x + 1)}$.

Table 4.7. Extended Euclid $[(x^8 + x^4 + x^3 + x + 1), (x^7 + x + 1)]$

(This item is displayed on page 125 in the print version)

Initialization	$A1(x) = 1; A2(x) = 0; A3(x) = x^8 + x^4 + x^3 + x + 1$ $B1(x) = 0; B2(x) = 1; B3(x) = x^7 + x + 1$
Iteration 1	$Q(x) = x$ $A1(x) = 0; A2(x) = 1; A3(x) = x^7 + x + 1$ $B1(x) = 1; B2(x) = x; B3(x) = x^4 + x^3 + x^2 + 1$
Iteration 2	$Q(x) = x^3 + x^2 + 1$ $A1(x) = 1; A2(x) = x; A3(x) = x^4 + x^3 + x^2 + 1$ $B1(x) = x^3 + x^2 + 1; B2(x) = x^4 + x^3 + x + 1; B3(x) = x$
Iteration 3	$Q(x) = x^3 + x^2 + x$ $A1(x) = x^3 + x^2 + 1; A2(x) = x^4 + x^3 + x + 1; A3(x) = x$ $B1(x) = x^6 + x^2 + x + 1; B2(x) = x^7; B3(x) = 1$
Iteration 4	$B3(x) = \text{gcd}[(x^7 + x + 1), (x^8 + x^4 + x^3 + x + 1)] = 1$ $B2(x) = (x^7 + x + 1)^{-1} \bmod (x^8 + x^4 + x^3 + x + 1) = x^7$

Computational Considerations

A polynomial $f(x)$ in $\text{GF}(2^n)$

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

can be uniquely represented by its n binary coefficients $(a_{n-1}a_{n-2}\dots a_0)$. Thus, every polynomial in $GF(2^n)$ can be represented by an n -bit number.

[Tables 4.5](#) and [4.6](#) show the addition and multiplication tables for $GF(2^3)$ modulo $m(x) = (x^3 + x + 1)$. [Table 4.5](#) uses the binary representation, and [Table 4.6](#) uses the polynomial representation.

Addition

We have seen that addition of polynomials is performed by adding corresponding coefficients, and, in the case of polynomials over Z_2 addition is just the XOR operation. So, addition of two polynomials in $GF(2^n)$ corresponds to a bitwise XOR operation.

Consider the two polynomials in $GF(2^8)$ from our earlier example: $f(x) = x^6 + x^4 + x^2 + x + 1$ and $g(x) = x^7 + x + 1$.

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \quad (\text{polynomial notation})$$

$$(01010111) \oplus (10000011) = (11010100) \quad (\text{binary notation})$$

$$\{57\} \oplus \{83\} = \{D4\} \quad (\text{hexadecimal notation})$$

^[7] A basic refresher on number systems (decimal, binary, hexadecimal) can be found at the Computer Science Student Resource Site at WilliamStallings.com/StudentSupport.html. Here each of two groups of 4 bits in a byte is denoted by a single hexadecimal character, the two characters enclosed in brackets.

Multiplication

There is no simple XOR operation that will accomplish multiplication in $GF(2^n)$. However, a reasonably straightforward, easily implemented technique is available. We will discuss the technique with reference to $GF(2^8)$ using $m(x) = x^8 + x^4 + x^3 + x + 1$, which is the finite field used in AES. The technique readily generalizes to $GF(2^n)$.

The technique is based on the observation that

Equation 4-8

$$x^8 \bmod m(x) = [m(x) - x^8] = (x^4 + x^3 + x + 1)$$

[Page 126]

A moment's thought should convince you that [Equation \(4.8\)](#) is true; if not, divide it out. In general, in $GF(2^n)$ with an n th-degree polynomial $p(x)$, we have $x^n \bmod p(x) = [p(x) - x^n]$.

Now, consider a polynomial in $GF(2^8)$, which has the form $f(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$. If we multiply by x , we have

Equation 4-9

$$x \times f(x) = (b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x) \bmod m(x)$$

If $b_7 = 0$, then the result is a polynomial of degree less than 8, which is already in reduced form, and no further computation is necessary. If $b_7 = 1$, then reduction modulo $m(x)$ is achieved using [Equation \(4.8\)](#):

$$x \times f(x) = (b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 +$$

$$b_1x^2 + b_0x) + (x^4 + x^3 + x + 1)$$

It follows that multiplication by x (i.e., 00000010) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (00011011), which represents $(x^4 + x^3 + x + 1)$. To summarize,

Equation 4-10

$$x \times f(x) = \begin{cases} (b_6b_5b_4b_3b_2b_1b_00) & \text{if } b_7 = 0 \\ (b_6b_5b_4b_3b_2b_1b_00) \oplus (00011011) & \text{if } b_7 = 1 \end{cases}$$

Multiplication by a higher power of x can be achieved by repeated application of [Equation \(4.10\)](#). By adding intermediate results, multiplication by any constant in $GF(2^8)$ can be achieved.

In an earlier example, we showed that for $f(x) = x^6 + x^4 + x^2 + x + 1$, $g(x) = x^7 + x + 1$, and $m(x) = x^8 + x^4 + x^3 + x + 1$, $f(x) \times g(x) \bmod m(x) = x^7 + x^6 + 1$. Redoing this in binary arithmetic, we need to compute $(01010111) \times (10000011)$. First, we determine the results of multiplication by powers of x :

$$(01010111) \times (00000001) = (10101110)$$

$$(01010111) \times (00000100) = (01011100) \oplus (00011011) = (01000111)$$

$$(01010111) \times (00001000) = (10001110)$$

$$(01010111) \times (00010000) = (00011100) \oplus (00011011) = (00000111)$$

$$(01010111) \times (00100000) = (00001110)$$

$$(01010111) \times (01000000) = (00011100)$$

$$(01010111) \times (10000000) = (00111000)$$

So,

$$(01010111) \times (10000011) = (01010111) \times [(00000001) \times (00000010) \times (10000000)]$$

$$= (01010111) \oplus (10101110) \oplus (00111000) = (11000001)$$

which is equivalent to $x^7 + x^6 + 1$.

Using a Generator

An equivalent technique for defining a finite field of the form $\text{GF}(2^n)$ using the same irreducible polynomial, is sometimes more convenient. To begin, we need two definitions: A **generator** g of a finite field F of order q (contains q elements) is an element whose first $q - 1$ powers generate all the nonzero elements of F . That is, the elements of F consist of $0, g^0, g^1, \dots, g^{q-2}$. Consider a field F defined by a polynomial $f(x)$. An element b contained in F is called a **root** of the polynomial if $f(b) = 0$. Finally, it can be shown that a root g of an irreducible polynomial is a generator of the finite field defined on that polynomial.

Let us consider the finite field $GF(2^3)$, defined over the irreducible polynomial $x^3 + x + 1$, discussed previously. Thus, the generator g must satisfy $f(x) = g^3 + g + 1 = 0$. Keep in mind, as discussed previously, that we need not find a numerical solution to this equality. Rather, we deal with polynomial arithmetic in which arithmetic on the coefficients is performed modulo 2. Therefore, the solution to the preceding equality is $g^3 = g + 1$. We now show that g in fact generates all of the polynomials of degree less than 3. We have the following:

$$g^4 = g(g^3) = g(g + 1) = g^2 + g$$

$$g^5 = g(g^4) = g(g^2 + g) = g^3 + g^2 = g^2 + g + 1$$

$$g^6 = g(g^5) = g(g^2 + g + 1) = g^3 + g^2 + g = g^2 + g + g + 1 = g^2 + 1$$

$$g^7 = g(g^6) = g(g^2 + 1) = g^3 + g = g + g + 1 = 1 = g^0$$

We see that the powers of g generate all the nonzero polynomials in $GF(2^3)$. Also, it should be clear that $g^k = g^{k \bmod 7}$ for any integer k . [Table 4.8](#) shows the power representation, as well as the polynomial and binary representations.

Table 4.8. Generator for $GF(2^3)$ using $x^3 + x + 1$

Power Representation	Polynomial Representation	Binary Representation	Decimal (Hex) Representation
0	0	000	0
$g^0 (= g^7)$	1	001	1
g^1	g	010	2
g^2	g^2	100	4
g^3	$g + 1$	011	3
g^4	$g^2 + g$	110	6
g^5	$g^2 + g + 1$	111	7
g^6	$g^2 + 1$	101	5

This power representation makes multiplication easy. To multiply in the power notation, add exponents modulo 7. For example, $g^4 \times g^6 = g^{(10 \bmod 7)} = g^3 = g + 1$. The same result is achieved using polynomial arithmetic, as follows: we have $g^4 = g^2 + g$ and $g^6 = g^2 + 1$. Then, $(g^2 + g) \times (g^2 + 1) = g^4 + g^3 + g^2 + 1$. Next, we need to determine $(g^4 + g^3 + g^2 + 1) \bmod (g^3 + g + 1)$ by division:

$$\begin{array}{r}
 g^3 + g^2 + 1 \overline{) g^4 + g^3 + g^2 + g} \\
 \underline{g^4 + + g^2 + g} \\
 g^3 + + \\
 \underline{g^3 + + } \\
 g + 1 \\
 \underline{g + 1} \\
 0
 \end{array}$$

We get a result of $g + 1$, which agrees with the result obtained using the power representation.

Table 4.9 shows the addition and multiplication tables for $GF(2^3)$ using the power representation. Note that this yields the identical results to the polynomial representation (Table 4.6) with some of the rows and columns interchanged.

Table 4.9. $GF(2^3)$ Arithmetic Using Generator for the Polynomial $(x^3 + x + 1)$

(This item is displayed on page 128 in the print version)

[\[View full size image\]](#)

		000	001	010	100	011	110	111	101
		0	1	g	g^2	g^3	g^4	g^5	g^6
000	0	0	1	g	g^2	$g+1$	g^2+g	g^2+g+1	g^2+1
001	1	1	0	$g+1$	g^2+1	g	g^2+g+1	g^2+g	g^2
010	g	g	$g+1$	0	g^2+g	1	g^2	g^2+1	g^2+g+1
100	g^2	g^2	g^2+1	g^2+g	0	g^2+g+1	g	$g+1$	1
011	g^3	$g+1$	g	1	g^2+g+1	0	g^2+1	g^2	g^2+g
110	g^4	g^2+g	g^2+g+1	g^2	g	g^2+1	0	1	$g+1$
111	g^5	g^2+g+1	g^2+g	g^2+1	$g+1$	g^2	1	0	g
101	g^6	g^2+1	g^2	g^2+g+1	1	g^2+g	$g+1$	g	0

(a) Addition

		000	001	010	100	011	110	111	101
		0	1	g	g^2	g^3	g^4	g^5	g^6
000	0	0	0	0	0	0	0	0	0
001	1	0	1	g	g^2	$g+1$	g^2+g	g^2+g+1	g^2+1
010	g	0	g	g^2	$g+1$	g^2+g	g^2+g+1	g^2+1	1
100	g^2	0	g^2	$g+1$	g^2+g	g^2+g+1	g^2+1	1	g
011	g^3	0	$g+1$	g^2+g	g^2+g+1	g^2+1	1	g	g^2
110	g^4	0	g^2+g	g^2+g+1	g^2+1	1	g	g^2	$g+1$
111	g^5	0	g^2+g+1	g^2+1	1	g	g^2	$g+1$	g^2+g
101	g^6	0	g^2+1	1	g	g^2	$g+1$	g^2+g	g^2+g+1

(b) Multiplication

In general, for $GF(2^n)$ with irreducible polynomial $f(x)$, determine $g^n = f(x) g^n$. Then calculate all of the

powers of g from g^{n+1} through g^{2^n-1} . The elements of the field correspond to the powers of g from through g^{2^n-1} , plus the value 0. For multiplication of two elements in the field, use the equality $g^k = g^k \pmod{2^n-1}$ for any integer k .

Summary

In this section, we have shown how to construct a finite field of order 2^n . Specifically, we defined $\text{GF}(2^n)$ with the following properties:

1.

$\text{GF}(2^n)$ consists of 2^n elements.

2.

The binary operations $+$ and \times are defined over the set. The operations of addition, subtraction, multiplication, and division can be performed without leaving the set. Each element of the set other than 0 has a multiplicative inverse.

We have shown that the elements of $\text{GF}(2^n)$ can be defined as the set of all polynomials of degree $n-1$ or less with binary coefficients. Each such polynomial can be represented by a unique n -bit value. Arithmetic is defined as polynomial arithmetic modulo some irreducible polynomial of degree n . We have also seen that an equivalent definition of a finite field $\text{GF}(2^n)$ makes use of a generator and that arithmetic is defined using powers of the generator.

[← PREV](#)

[NEXT →](#)

4.7. Recommended Reading and Web Sites

[[HERS75](#)], still in print, is the classic treatment of abstract algebra; it is readable and rigorous; [[DESK92](#)] is another good resource. [[KNUT98](#)] provides good coverage of polynomial arithmetic.

[Page 130]

One of the best treatments of the topics of this chapter is [[BERL84](#)], still in print. [[GARR01](#)] also has extensive coverage. A thorough and rigorous treatment of finite fields is [[LIDL94](#)]. [[HORO71](#)] is a good overview of the topics of this chapter.

[BERL84](#) Berlekamp, E. *Algebraic Coding Theory*. Laguna Hills, CA: Aegean Park Press, 1984.

[DESK92](#) Deskins, W. *Abstract Algebra*. New York: Dover, 1992.

[GARR01](#) Garrett, P. *Making, Breaking Codes: An Introduction to Cryptology*. Upper Saddle River, NJ: Prentice Hall, 2001.

[HERS75](#) Herstein, I. *Topics in Algebra*. New York: Wiley, 1975.

[HORO71](#) Horowitz, E. "Modular Arithmetic and Finite Field Theory: A Tutorial." *Proceedings of the Second ACM Symposium and Symbolic and Algebraic Manipulation*, March 1971.

[KNUT98](#) Knuth, D. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1998.

[LIDL94](#) Lidl, R., and Niederreiter, H. *Introduction to Finite Fields and Their Applications*. Cambridge: Cambridge University Press, 1994.

Recommended Web Sites



- **PascGalois Project:** Contains a clever set of examples and projects to aid in giving students a visual understanding of key concepts in abstract algebra

4.8. Key Terms, Review Questions, and Problems

Key Terms

[abelian group](#)

[associative](#)

[coefficient set](#)

[commutative](#)

[commutative ring](#)

[cyclic group](#)

[divisor](#)

[Euclidean algorithm](#)

[field](#)

[finite group](#)

finite ring

[finite field](#)

[generator](#)

[greatest common divisor](#)

[group](#)

[identity element](#)

[infinite group](#)

infinite ring

[infinite field](#)

[integral domain](#)

[inverse element](#)

[irreducible polynomial](#)

[modular arithmetic](#)

[modular polynomial arithmetic](#)

modulo operator

[modulus](#)

[monic polynomial](#)

[order](#)

[polynomial](#)

[polynomial arithmetic](#)

[polynomial ring](#)

[prime number](#)

[prime polynomial](#)

[relatively prime](#)

[residue](#)

[ring](#)

Review Questions

4.1. Briefly define a group.

4.2. Briefly define a ring.

- 4.3. Briefly define a field.
- 4.4. What does it mean to say that b is a divisor of a ?
- 4.5. What is the difference between modular arithmetic and ordinary arithmetic?
- 4.6. List three classes of polynomial arithmetic.

Problems

4.1 For the group S_n of all permutations of n distinct symbols,

a.

What is the number of elements in S_n ?

b.

Show that S_n is not abelian for $n > 2$.

4.2 Does the set of residue classes modulo 3 form a group

a.

with respect to addition?

b.

with respect to multiplication?

4.3 Consider the set $S = \{a, b\}$ with addition and multiplication defined by the tables:

+	a	b
a	a	b
b	b	a

×	a	b
a	a	a
b	a	b

Is S a ring? Justify your answer.

4.4 Reformulate [Equation \(4.1\)](#), removing the restriction that a is a nonnegative integer. That is, let a be any integer.

4.5 Draw a figure similar to [Figure 4.2](#) for $a < 0$.

4.6 Find integers x such that

a.

$$5x \equiv 4 \pmod{3}$$

b.

$$7x \equiv 6 \pmod{5}$$

c.

$$9x \equiv 8 \pmod{7}$$

4.7 In this text we assume that the modulus is a positive integer. But the definition of the expression $a \bmod n$ also makes perfect sense if n is negative. Determine the following:

a.

$$5 \bmod 3$$

b.

$$5 \bmod 3$$

c.

$$5 \bmod 3$$

d.

$$5 \pmod{3}$$

4.8 A modulus of 0 does not fit the definition, but is defined by convention as follows: $a \pmod{0} = a$. With this definition in mind, what does the following expression mean: $a \equiv b \pmod{0}$?

4.9 In [Section 4.2](#), we define the congruence relationship as follows: Two integers a and b are said to be congruent modulo n , if $(a \pmod{n}) = (b \pmod{n})$. We then proved that $a \equiv b \pmod{n}$ if $n|(a - b)$. 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 $n|(a - b)$. Using this latter definition as the starting point, prove that if $(a \pmod{n}) = (b \pmod{n})$, then n divides $(a - b)$.

4.10 What is the smallest positive integer that has exactly k divisors, for $1 \leq k \leq 6$?

4.11 Prove the following:

a.

$$a \equiv b \pmod{n} \text{ implies } b \equiv a \pmod{n}$$

b.

$$a \equiv b \pmod{n} \text{ and } b \equiv c \pmod{n} \text{ imply } a \equiv c \pmod{n}$$

4.12 Prove the following:

a.

$$[(a \pmod{n}) (b \pmod{n})] \pmod{n} = (a b) \pmod{n}$$

b.

$$[(a \pmod{n}) \times (b \pmod{n})] \pmod{n} = (a \times b) \pmod{n}$$

4.13 Find the multiplicative inverse of each nonzero element in \mathbb{Z}_5 .

4.14 Show that an integer N is congruent modulo 9 to the sum of its decimal digits. For example, $475 \equiv 4 + 7 + 5 \equiv 16 \equiv 1 + 6 \equiv 7 \pmod{9}$. This is the basis for the familiar procedure of "casting out 9's" when checking computations in arithmetic.

4.15 a. Determine $\gcd(24140, 16762)$.

b. Determine $\gcd(4655, 12075)$.

4.16 The purpose of this problem is to set an upper bound on the number of iterations of the Euclidean algorithm.

a. Suppose that $m = qn + r$ with $q > 0$ and $0 \leq r < n$. Show that $m/2 > r$.

b. Let A_i be the value of A in the Euclidean algorithm after the i th iteration. Show that

$$A_{i+2} < \frac{A_i}{2}$$

c. Show that if m , n , and N are integers with $1 \leq m, n \leq 2^N$, then the Euclidean algorithm takes at most $2N$ steps to find $\gcd(m, n)$.

4.17 The Euclidean 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 $A_1 = A, B_1 = B, C_1 = 1$

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 the Euclidean and Stein's algorithm.

b.

What is the apparent advantage of Stein's algorithm over the Euclidean algorithm?

4.18

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

$$A_{n+2}B_{n+2} \leq \frac{A_nB_n}{2}$$

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 the Euclidean algorithm.

d.

Demonstrate that Stein's algorithm does indeed return $\gcd(A, B)$.

4.19 Using the extended Euclidean algorithm, find the multiplicative inverse of

a.

1234 mod 4321

b.

24140 mod 40902

c.

550 mod 1769

4.20 Develop a set of tables similar to [Table 4.3](#) for $\text{GF}(5)$.

4.21 Demonstrate that the set of polynomials whose coefficients form a field is a ring.

4.22 Demonstrate whether each of these statements is true or false for polynomials over a field:

a.

The product of monic polynomials is monic.

b.

The product of polynomials of degrees m and n has degree $m + n$

c.

The sum of polynomials of degrees m and n has degree $\max[m, n]$.

[Page 133]

4.23 For polynomial arithmetic with coefficients in Z_{10} , perform the following calculations:

a.

$$(7x + 2)(x^2 + 5)$$

b.

$$(6x^2 + x + 3) \times (5x^2 + 2)$$

4.24 Determine which of the following are reducible over $GF(2)$:

a.

$$x^3 + 1$$

b.

$$x^3 + x^2 + 1$$

c.

$$x^4 + 1 \text{ (be careful)}$$

4.25 Determine the gcd of the following pairs of polynomials:

a.

$$x^3 + x + 1 \text{ and } x^2 + x + 1 \text{ over GF}(2)$$

b.

$$x^3 x + 1 \text{ and } x^2 + 1 \text{ over GF}(3)$$

c.

$$x^5 + x^4 + x^3 x^2 x + 1 \text{ and } x^3 + x^2 + x + 1 \text{ over GF}(3)$$

d.

$$x^5 + 88x^4 + 73x^3 + 83x^2 + 51x + 67 \text{ and } x^3 + 97x^2 + 40x + 38 \text{ over GF}(101)$$

4.26 Develop a set of tables similar to [Table 4.6](#) for GF(4) with $m(x) = x^4 + x + 1$.

4.27 Determine the multiplicative inverse of $x^3 + x + 1$ in GF(2⁴), with $m(x) = x^4 + x + 1$.

4.28 Develop a table similar to [Table 4.8](#) for GF(2⁴) with $m(x) = x^4 + x + 1$.

Programming Problems

4.29 Write a simple four-function calculator in GF(2⁴). You may use table lookups for the multiplicative inverses.

4.30 Write a simple four-function calculator in GF(2⁸). You should compute the multiplicative inverses on the fly.

Chapter 5. Advanced Encryption Standard

5.1 Evaluation Criteria For AES

[The Origins of AES](#)

[AES Evaluation](#)

5.2 The AES Cipher

[Substitute Bytes Transformation](#)

[ShiftRows Transformation](#)

[MixColumns Transformation](#)

[AddRoundKey Transformation](#)

[AES Key Expansion](#)

[Equivalent Inverse Cipher](#)

[Implementation Aspects](#)

5.3 Recommended Reading and Web Sites

5.4 Key Terms, Review Questions, and Problems

[Key Terms](#)

[Review Questions](#)

[Problems](#)

Appendix 5A Polynomials With Coefficients In GF (2⁸)

[MixColumns Transformation](#)

[Multiplication by x](#)

Appendix 5B Simplified AES

[Overview](#)

[S-AES Encryption and Decryption](#)

[Key Expansion](#)

[The S-Box](#)

[S-AES Structure](#)

[Page 135]

"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

Key Points

- AES is a block cipher intended to replace DES for commercial applications. It uses a 128-bit block size and a key size of 128, 192, or 256 bits.
- AES does not use a Feistel structure. Instead, each full round consists of four separate functions: byte substitution, permutation, arithmetic operations over a finite field, and XOR with a key.

The Advanced Encryption Standard (AES) was published by NIST (National Institute of Standards and Technology) in 2001. AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. In this chapter, we first look at the evaluation criteria used by NIST to select a candidate for AES and then examine the cipher itself.

Compared to public-key ciphers such as RSA, the structure of AES, and most symmetric ciphers, is very complex and cannot be explained as easily as RSA and similar algorithms. Accordingly, the reader may wish to begin with a simplified version of AES, which is described in [Appendix 5B](#). This version allows the reader to perform encryption and decryption by hand and gain a good understanding of the working of the algorithm details. Classroom experience indicates that a study of this simplified version enhances understanding of AES. ^[1]

^[1] However, you may safely skip [Appendix 5B](#), at least on a first reading. If you get lost or bogged down in the details of AES, then you can go back and start with simplified AES.



5.1. Evaluation Criteria For AES

The Origins of AES

We mentioned in [Chapter 3](#) that in 1999, NIST issued a new version of its DES standard (FIPS PUB 46-3) that indicated that DES should only be used for legacy systems and that triple DES (3DES) be used. We describe 3DES in [Chapter 6](#). 3DES has two attractions that assure its widespread use over the next few years. First, with its 168-bit key length, it overcomes the vulnerability to brute-force attack of DES. Second, the underlying encryption algorithm in 3DES is the same as in DES. This algorithm has been subjected to more scrutiny than any other encryption algorithm over a longer period of time, and no effective cryptanalytic attack based on the algorithm rather than brute force has been found. Accordingly, there is a high level of confidence that 3DES is very resistant to cryptanalysis. If security were the only consideration, then 3DES would be an appropriate choice for a standardized encryption algorithm for decades to come.

The principal drawback of 3DES is that the algorithm is relatively sluggish in software. The original DES was designed for mid-1970s hardware implementation and does not produce efficient software code. 3DES, which has three times as many rounds as DES, is correspondingly slower. A secondary drawback is that both DES and 3DES use a 64-bit block size. For reasons of both efficiency and security, a larger block size is desirable.

Because of these drawbacks, 3DES is not a reasonable candidate for long-term use. As a replacement, NIST in 1997 issued a call for proposals for a new Advanced Encryption Standard (AES), which should have a security strength equal to or better than 3DES and significantly improved efficiency. In addition to these general requirements, NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits.

In a first round of evaluation, 15 proposed algorithms were accepted. A second round narrowed the field to 5 algorithms. NIST completed its evaluation process and published a final standard (FIPS PUB 197) in November of 2001. NIST selected Rijndael as the proposed AES algorithm. The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium: Dr. Joan Daemen and Dr. Vincent Rijmen.

Ultimately, AES is intended to replace 3DES, but this process will take a number of years. NIST anticipates that 3DES will remain an approved algorithm (for U.S. government use) for the foreseeable future.

AES Evaluation

It is worth examining the criteria used by NIST to evaluate potential candidates. These criteria span the range of concerns for the practical application of modern symmetric block ciphers. In fact, two set of criteria evolved. When NIST issued its original request for candidate algorithm nominations in 1997 [[NIST97](#)], the request stated that candidate algorithms would be compared based on the factors shown in [Table 5.1](#) (ranked in descending order of relative importance). The three categories of criteria were as follows:

- **Security:** This refers to the effort required to cryptanalyze an algorithm. The emphasis in the evaluation was on the practicality of the attack. Because the minimum key size for AES is 128 bits, brute-force attacks with current and projected technology were considered impractical. Therefore, the emphasis, with respect to this point, is cryptanalysis other than a brute-force attack.
- **Cost:** NIST intends AES to be practical in a wide range of applications. Accordingly, AES must have high computational efficiency, so as to be usable in high-speed applications, such as broadband links.

[Page 137]

- **Algorithm and implementation characteristics:** This category includes a variety of considerations, including flexibility; suitability for a variety of hardware and software implementations; and simplicity, which will make an analysis of security more straightforward.

Table 5.1. NIST Evaluation Criteria for AES (September 12, 1997)

SECURITY
<ul style="list-style-type: none"> • Actual security: compared to other submitted algorithms (at the same key and block size). • Randomness: the extent to which the algorithm output is indistinguishable from a random permutation on the input block. • Soundness: of the mathematical basis for the algorithm's security. • Other security factors: raised by the public during the evaluation process, including any attacks which demonstrate that the actual security of the algorithm is less than the strength claimed by the submitter.
COST
<ul style="list-style-type: none"> • Licensing requirements: NIST intends that when the AES is issued, the algorithm(s) specified in the AES shall be available on a worldwide, non-exclusive, royalty-free basis. • Computational efficiency: The evaluation of computational efficiency will be applicable to both hardware and software implementations. Round 1 analysis by NIST will focus primarily on software implementations and specifically on one key-block size combination (128-128); more attention will be paid to hardware implementations and other supported key-block size combinations during Round 2 analysis. Computational efficiency essentially refers to the speed of the algorithm. Public comments on each algorithm's efficiency (particularly for various platforms and applications) will also be taken into consideration by NIST. • Memory requirements: The memory required to implement a candidate algorithm for both hardware and software implementations of the algorithm will also be considered during the evaluation process. Round 1 analysis by NIST will focus primarily on software implementations; more attention will be paid to hardware implementations during Round 2. Memory requirements will include such factors as gate counts for hardware implementations, and code size and RAM requirements for software implementations.
ALGORITHM AND IMPLEMENTATION CHARACTERISTICS

- **Flexibility:** Candidate algorithms with greater flexibility will meet the needs of more users than less flexible ones, and therefore, inter alia, are preferable. However, some extremes of functionality are of little practical application (e.g., extremely short key lengths); for those cases, preference will not be given. Some examples of flexibility may include (but are not limited to) the following:

a.

The algorithm can accommodate additional key- and block-sizes (e.g., 64-bit block sizes, key sizes other than those specified in the Minimum Acceptability Requirements section, [e.g., keys between 128 and 256 that are multiples of 32 bits, etc.]

b.

The algorithm can be implemented securely and efficiently in a wide variety of platforms and applications (e.g., 8-bit processors, ATM networks, voice & satellite communications, HDTV, B-ISDN, etc.).

c.

The algorithm can be implemented as a stream cipher, message authentication code (MAC) generator, pseudorandom number generator, hashing algorithm, etc.

- **Hardware and software suitability:** A candidate algorithm shall not be restrictive in the sense that it can only be implemented in hardware. If one can also implement the algorithm efficiently in firmware, then this will be an advantage in the area of flexibility.
- **Simplicity:** A candidate algorithm shall be judged according to relative simplicity of design.

Using these criteria, the initial field of 21 candidate algorithms was reduced first to 15 candidates and then to 5 candidates. By the time that a final evaluation had been done the evaluation criteria, as described in [[NECHOO](#)], had evolved. The following criteria were used in the final evaluation:

- **General security:** To assess general security, NIST relied on the public security analysis conducted by the cryptographic community. During the course of the three-year evaluation process, a number of cryptographers published their analyses of the strengths and weaknesses of the various candidates. There was particular emphasis on analyzing the candidates with respect to known attacks, such as differential and linear cryptanalysis. However, compared to the analysis of DES, the amount of time and the number of cryptographers devoted to analyzing Rijndael are quite limited. Now that a single AES cipher has been chosen, we can expect to see a more extensive security analysis by the cryptographic community.
- **Software implementations:** The principal concerns in this category are execution speed, performance across a variety of platforms, and variation of speed with key size.
- **Restricted-space environments:** In some applications, such as smart cards, relatively small amounts of random-access memory (RAM) and/or read-only memory (ROM) are available for such purposes as code storage (generally in ROM); representation of data objects such as S-boxes (which could be stored in ROM or RAM, depending on whether pre-computation or Boolean representation is used); and subkey storage (in RAM).
- **Hardware implementations:** Like software, hardware implementations can be optimized for speed or for size. However, in the case of hardware, size translates much more directly into cost than is usually the case for software implementations. Doubling the size of an encryption

program may make little difference on a general-purpose computer with a large memory, but doubling the area used in a hardware device typically more than doubles the cost of the device.

- **Attacks on implementations:** The criterion of general security, discussed in the first bullet, is concerned with cryptanalytic attacks that exploit mathematical properties of the algorithms. There is another class of attacks that use physical measurements conducted during algorithm execution to gather information about quantities such as keys. Such attacks exploit a combination of intrinsic algorithm characteristics and implementation-dependent features. Examples of such attacks are timing attacks and power analysis. Timing attacks are described in [Chapter 3](#). The basic idea behind power analysis [[KOCH98](#), [BIHA00](#)] is the observation that the power consumed by a smart card at any particular time during the cryptographic operation is related to the instruction being executed and to the data being processed. For example, multiplication consumes more power than addition, and writing 1s consumes more power than writing 0s.
- **Encryption versus decryption:** This criterion deals with several issues related to considerations of both encryption and decryption. If the encryption and decryption algorithms differ, then extra space is needed for the decryption. Also, whether the two algorithms are the same or not, there may be timing differences between encryption and decryption.

[Page 139]

- **Key agility:** Key agility refers to the ability to change keys quickly and with a minimum of resources. This includes both subkey computation and the ability to switch between different ongoing security associations when subkeys may already be available.
- **Other versatility and flexibility:** [[NECH00](#)] indicates two areas that fall into this category. Parameter flexibility includes ease of support for other key and block sizes and ease of increasing the number of rounds in order to cope with newly discovered attacks. Implementation flexibility refers to the possibility of optimizing cipher elements for particular environments.
- **Potential for instruction-level parallelism:** This criterion refers to the ability to exploit ILP features in current and future processors.

[Table 5.2](#) shows the assessment that NIST provided for Rijndael based on these criteria.

Table 5.2. Final NIST Evaluation of Rijndael (October 2, 2000)

[Page 140]

General Security
Rijndael has no known security attacks. Rijndael uses S-boxes as nonlinear components. Rijndael appears to have an adequate security margin, but has received some criticism suggesting that its mathematical structure may lead to attacks. On the other hand, the simple structure may have facilitated its security analysis during the timeframe of the AES development process.
Software Implementations
Rijndael performs encryption and decryption very well across a variety of platforms, including 8-bit and 64-bit platforms, and DSPs. However, there is a decrease in performance with the higher key sizes because of the increased number of rounds that are performed. Rijndael's high inherent parallelism facilitates the efficient use of processor resources, resulting in very good software performance even when implemented in a mode not capable of interleaving. Rijndael's key setup time is fast.
Restricted-Space Environments

In general, Rijndael is very well suited for restricted-space environments where either encryption or decryption is implemented (but not both). It has very low RAM and ROM requirements. A drawback is that ROM requirements will increase if both encryption and decryption are implemented simultaneously, although it appears to remain suitable for these environments. The key schedule for decryption is separate from encryption.

Hardware Implementations

Rijndael has the highest throughput of any of the finalists for feedback modes and second highest for non-feedback modes. For the 192 and 256-bit key sizes, throughput falls in standard and unrolled implementations because of the additional number of rounds. For fully pipelined implementations, the area requirement increases, but the throughput is unaffected.

Attacks on Implementations

The operations used by Rijndael are among the easiest to defend against power and timing attacks. The use of masking techniques to provide Rijndael with some defense against these attacks does not cause significant performance degradation relative to the other finalists, and its RAM requirement remains reasonable. Rijndael appears to gain a major speed advantage over its competitors when such protections are considered.

Encryption vs. Decryption

The encryption and decryption functions in Rijndael differ. One FPGA study reports that the implementation of both encryption and decryption takes about 60% more space than the implementation of encryption alone. Rijndael's speed does not vary significantly between encryption and decryption, although the key setup performance is slower for decryption than for encryption.

Key Agility

Rijndael supports on-the-fly subkey computation for encryption. Rijndael requires a one-time execution of the key schedule to generate all subkeys prior to the first decryption with a specific key. This places a slight resource burden on the key agility of Rijndael.

Other Versatility and Flexibility

Rijndael fully supports block sizes and key sizes of 128 bits, 192 bits and 256 bits, in any combination. In principle, the Rijndael structure can accommodate any block sizes and key sizes that are multiples of 32, as well as changes in the number of rounds that are specified.

Potential for Instruction-Level Parallelism

Rijndael has an excellent potential for parallelism for a single block encryption.

5.2. The AES Cipher^[2]

^[2] Much of the material in this section originally appeared in [\[STAL02\]](#).

The Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192, or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. A number of AES parameters depend on the key length ([Table 5.3](#)). In the description of this section, we assume a key length of 128 bits, which is likely to be the one most commonly implemented.

Table 5.3. AES Parameters

Key size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

Rijndael was designed to have the following characteristics:

- Resistance against all known attacks
- Speed and code compactness on a wide range of platforms
- Design simplicity

[Figure 5.1](#) shows the overall structure of AES. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a square matrix of bytes. This block is copied into the **State** array, which is modified at each stage of encryption or decryption. After the final stage, **State** is copied to an output matrix. These operations are depicted in [Figure 5.2a](#). Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words; each word is four bytes and the total key schedule is 44 words for the 128-bit key ([Figure 5.2b](#)). Note that the ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the **in** matrix, the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the **w** matrix.

Figure 5.1. AES Encryption and Decryption

[\[View full size image\]](#)

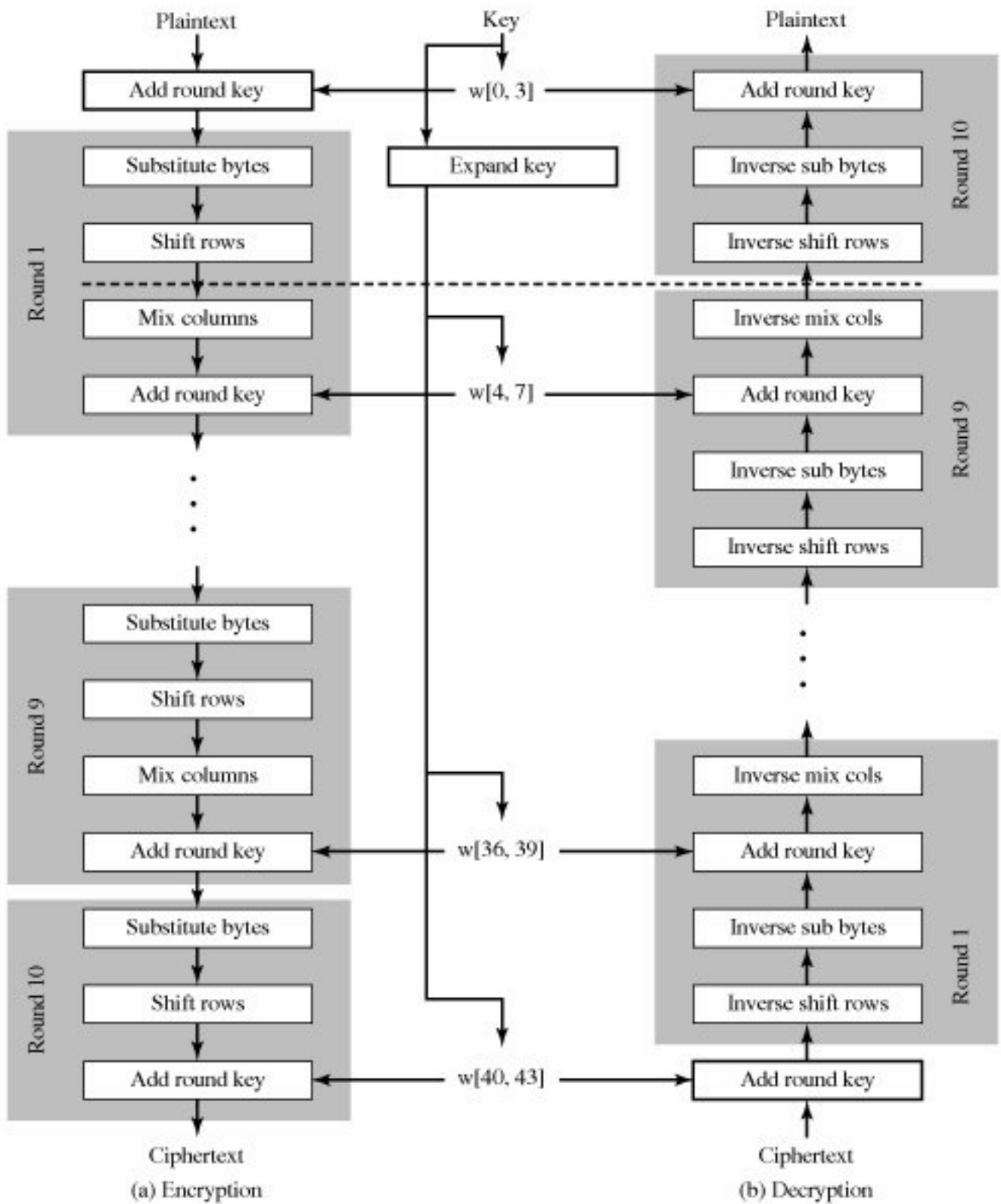
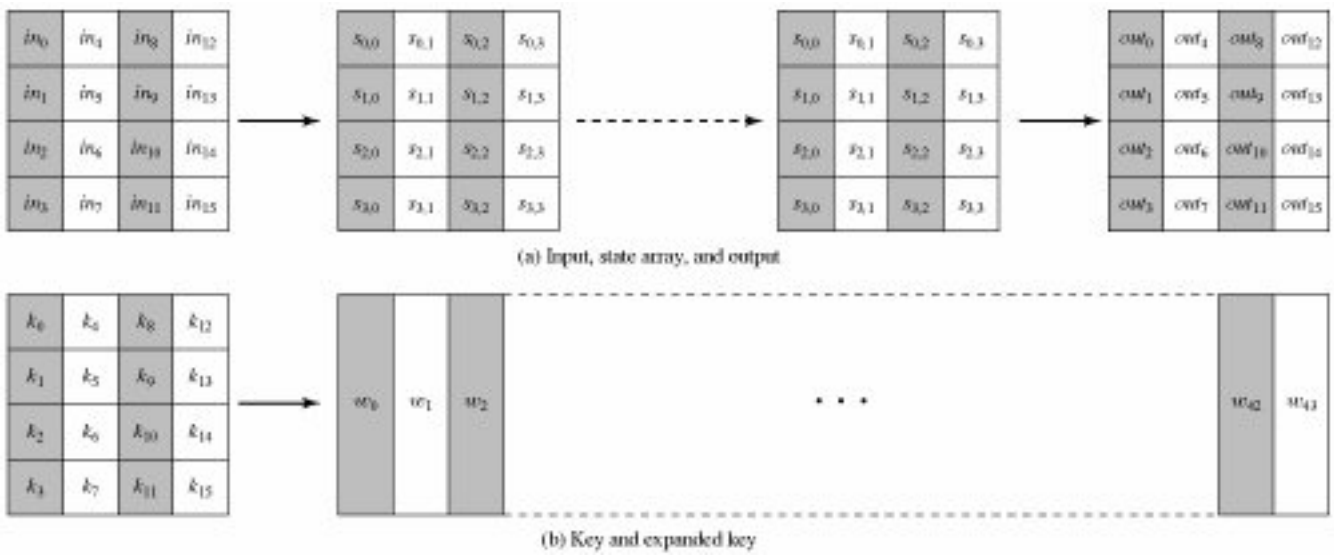


Figure 5.2. AES Data Structures

[View full size image](#)



Before delving into details, we can make several comments about the overall AES structure:

1.

One noteworthy feature of this structure is that it is not a Feistel structure. Recall that in the classic Feistel structure, half of the data block is used to modify the other half of the data block, and then the halves are swapped. Two of the AES finalists, including Rijndael, do not use a Feistel structure but process the entire data block in parallel during each round using substitutions and permutation.

2.

The key that is provided as input is expanded into an array of forty-four 32-bit words, $w[j]$. Four distinct words (128 bits) serve as a round key for each round; these are indicated in [Figure 5.1](#).

3.

Four different stages are used, one of permutation and three of substitution:

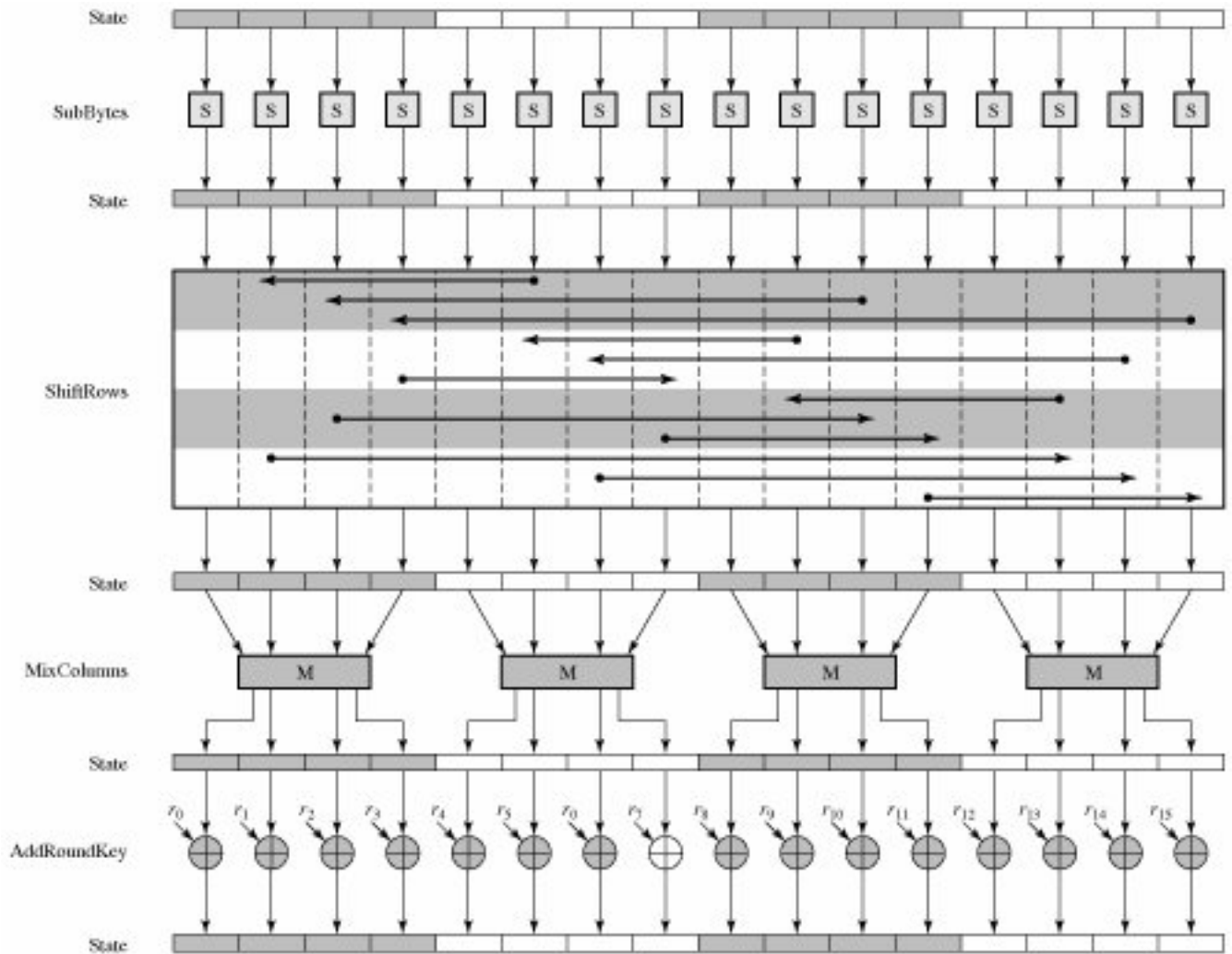
- o **Substitute bytes:** Uses an S-box to perform a byte-by-byte substitution of the block
- o **ShiftRows:** A simple permutation
- o **MixColumns:** A substitution that makes use of arithmetic over $GF(2^8)$
- o **AddRoundKey:** A simple bitwise XOR of the current block with a portion of the expanded key

4.

The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. [Figure 5.3](#) depicts the structure of a full encryption round.

Figure 5.3. AES Encryption Round

[\[View full size image\]](#)



5.

Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.

6.

The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.

7.

Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that $A \oplus A \oplus B = B$.

8.

As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.

9.

Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. [Figure 5.1](#) lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), **State** is the same for both encryption and decryption.

[Page 145]

10.

The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

We now turn to a discussion of each of the four stages used in AES. For each stage, we describe the forward (encryption) algorithm, the inverse (decryption) algorithm, and the rationale for the stage. This is followed by a discussion of key expansion.

As was mentioned in [Chapter 4](#), AES uses arithmetic in the finite field $GF(2^8)$, with the irreducible polynomial^[3] $m(x) = x^8 + x^4 + x^3 + x + 1$. The developers of Rijndael give as their motivation for selecting this one of the 30 possible irreducible polynomials of degree 8 that it is the first one on the list given in [\[LIDL94\]](#).

^[3] In the remainder of this discussion, references to $GF(2^8)$ refer to the finite field defined with this polynomial.

Substitute Bytes Transformation

Forward and Inverse Transformations

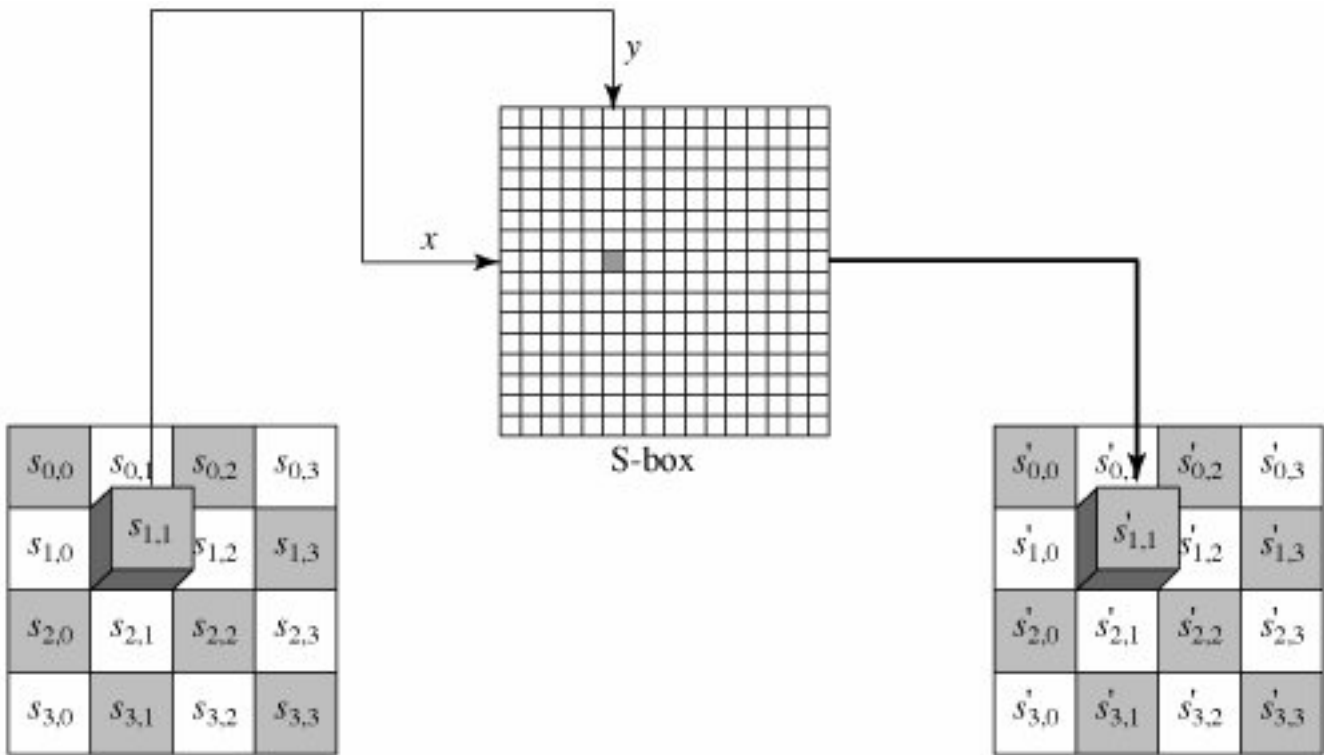
The **forward substitute byte transformation**, called SubBytes, is a simple table lookup ([Figure 5.4a](#)). AES defines a 16 x 16 matrix of byte values, called an S-box ([Table 5.4a](#)), that contains a permutation of all possible 256 8-bit values. Each individual byte of **State** is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value^[4] {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

^[4] In FIPS PUB 197, a hexadecimal number is indicated by enclosing it in curly brackets. We use that convention in this chapter.

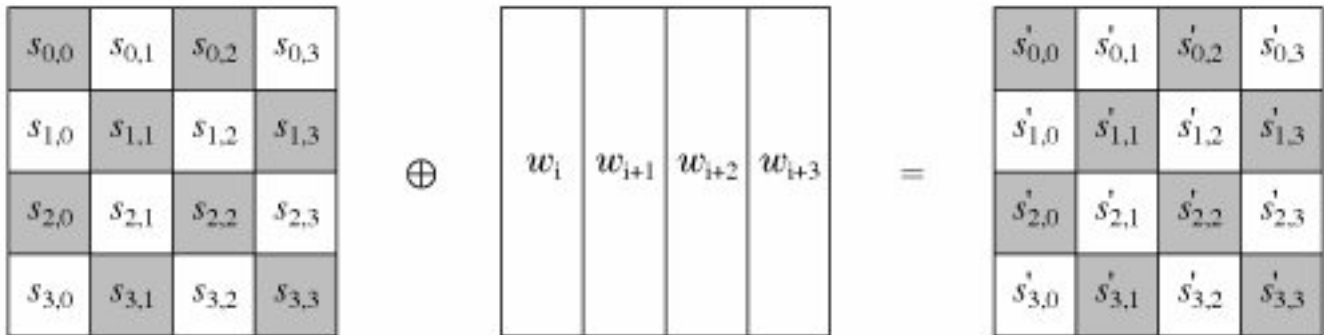
[Page 147]

Figure 5.4. AES Byte-Level Operations

[\[View full size image\]](#)



(a) Substitute byte transformation



(b) Add Round Key Transformation

Table 5.4. AES S-Boxes

[\[View full size image\]](#)

(a) S-box

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(b) Inverse S-box

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Here is an example of the SubBytes transformation:

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

The S-box is constructed in the following fashion:

1.

Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02}, ..., {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row x , column y is $\{xy\}$.

2.

Map each byte in the S-box to its multiplicative inverse in the finite field $GF(2^8)$; the value {00} is mapped to itself.

3.

Consider that each byte in the S-box consists of 8 bits labeled $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$. Apply the following transformation to each bit of each byte in the S-box:

Equation 5-1

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

where c_i is the i th bit of byte c with the value {63}; that is, $(c_7c_6c_5c_4c_3c_2c_1c_0) = (01100011)$.

The prime (') indicates that the variable is to be updated by the value on the right. The AES standard depicts this transformation in matrix form as follows:

Equation 5-2

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

[Equation \(5.2\)](#) has to be interpreted carefully. In ordinary matrix multiplication,^[5] each element in the product matrix is the sum of products of the elements of one row and one column. In this case, each element in the product matrix is the bitwise XOR of products of elements of one row and one column. Further, the final addition shown in [Equation \(5.2\)](#) is a bitwise XOR.

^[5] For a brief review of the rules of matrix and vector multiplication, see the Math Refresher document and the Computer Science Student Resource site at williamstallings.com/StudentSupport.html.

As an example, consider the input value {95}. The multiplicative inverse in GF(2⁸) is {95}¹ = {8A}, which is 10001010 in binary. Using [Equation \(5.2\)](#),

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

The result is {2A}, which should appear in row {09} column {05} of the S-box. This is verified by checking [Table 5.4a](#).

The **inverse substitute byte transformation**, called InvSubBytes, makes use of the inverse S-box shown in [Table 5.4b](#). Note, for example, that the input {2A} produces the output {95} and the input {95} to the S-box produces {2A}. The inverse S-box is constructed by applying the inverse of the

transformation in [Equation \(5.1\)](#) followed by taking the multiplicative inverse in GF(2⁸). The inverse transformation is:

$$b'_i = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i$$

where byte $d = \{05\}$, or 00000101. We can depict this transformation as follows:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

To see that InvSubBytes is the inverse of SubBytes, label the matrices in SubBytes and InvSubBytes as \mathbf{X} and \mathbf{Y} , respectively, and the vector versions of constants c and d as \mathbf{C} and \mathbf{D} , respectively. For some 8-bit vector \mathbf{B} , [Equation \(5.2\)](#) becomes $\mathbf{B}' = \mathbf{XB} \oplus \mathbf{C}$. We need to show that $\mathbf{Y}(\mathbf{XB} \oplus \mathbf{C}) \oplus \mathbf{D} = \mathbf{B}$.

Multiply out, we must show $\mathbf{YXB} \oplus \mathbf{YC} \oplus \mathbf{D} = \mathbf{B}$. This becomes

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

We have demonstrated that \mathbf{YX} equals the identity matrix, and the $\mathbf{YC} = \mathbf{D}$, so that $\mathbf{YC} \oplus \mathbf{D}$ equals the null vector.

Rationale

The S-box is designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design that has a low correlation between input bits and output bits, and the property that the output cannot be described as a simple mathematical function of the input [DAEM01]. In addition, the constant in Equation (5.1) was chosen so that the S-box has no fixed points [S-box(a) = a] and no "opposite fixed points" [S-box(a) = ā], where ā is the bitwise complement of a.

Of course, the S-box must be invertible, that is, IS-box[S-box(a)] = a. However, the S-box is not self-inverse in the sense that it is not true that S-box(a) = IS-box(a). For example, [S-box({95}) = {2A}], but IS-box({95}) = {AD}.

ShiftRows Transformation

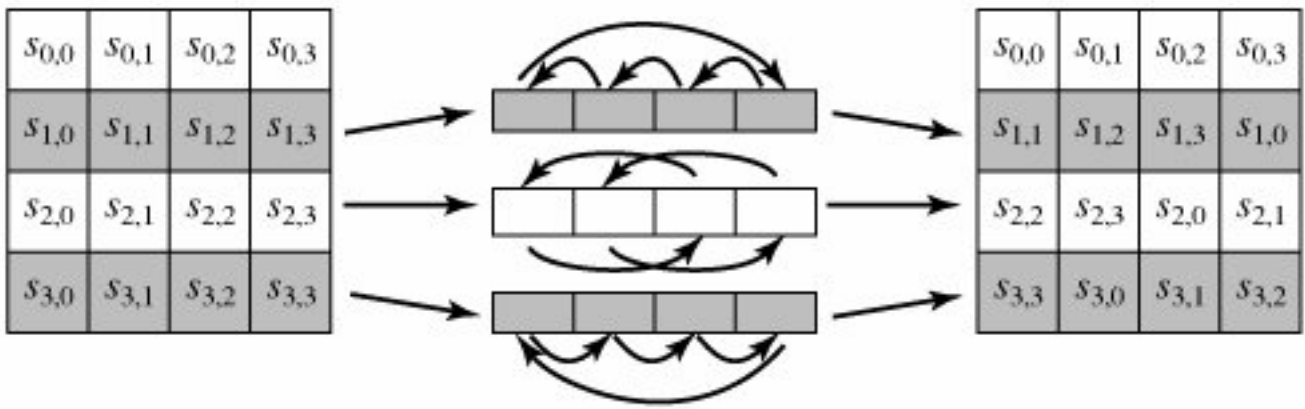
Forward and Inverse Transformations

The **forward shift row transformation**, called ShiftRows, is depicted in [Figure 5.5a](#). The first row of **State** is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following is an example of ShiftRows:

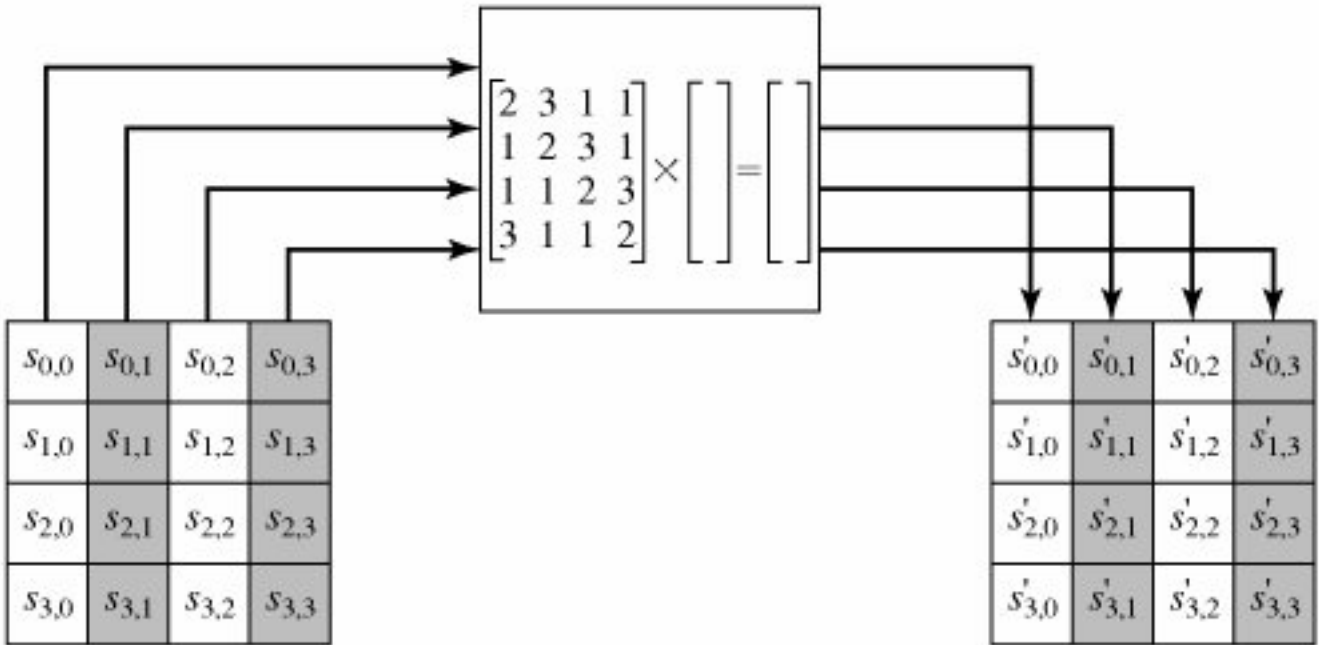
87	F2	4D	97	→	87	F2	4D	97
EC	6E	4C	90		6E	4C	90	EC
4A	C3	46	E7		46	E7	4A	C3
8C	D8	95	A6		A6	8C	D8	95

Figure 5.5. AES Row and Column Operations

[\[View full size image\]](#)



(a) Shift row transformation



(b) Mix column transformation

The **inverse shift row transformation**, called `InvShiftRows`, performs the circular shifts in the opposite direction for each of the last three rows, with a one-byte circular right shift for the second row, and so on.

Rationale

The shift row transformation is more substantial than it may first appear. This is because the **State**, as well as the cipher input and output, is treated as an array of four 4-byte columns. Thus, on encryption, the first 4 bytes of the plaintext are copied to the first column of **State**, and so on. Further, as will be seen, the round key is applied to **State** column by column. Thus, a row shift moves an individual byte from one column to another, which is a linear distance of a multiple of 4 bytes. Also note that the transformation ensures that the 4 bytes of one column are spread out to four different columns. [Figure 5.3](#) illustrates the effect.

MixColumns Transformation

Forward and Inverse Transformations

The **forward mix column transformation**, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the following matrix multiplication on **State** (Figure 5.5b):

Equation 5-3

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications^[6] are performed in GF(2⁸). The MixColumns transformation on a single column j ($0 \leq j \leq 3$) of **State** can be expressed as

^[6] We follow the convention of FIPS PUB 197 and use the symbol \cdot to indicate multiplication over the finite field GF(2⁸) and \oplus to indicate bitwise XOR, which corresponds to addition in GF(2⁸).

Equation 5-4

$$\begin{aligned} s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \\ s'_{3,j} &= (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j}) \end{aligned}$$

The following is an example of MixColumns:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

Let us verify the first column of this example. Recall from [Section 4.6](#) that, in GF(2⁸), addition is the bitwise XOR operation and that multiplication can be performed according to the rule established in [Equation \(4.10\)](#). In particular, multiplication of a value by x (i.e., by {02}) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the original

value (prior to the shift) is 1. Thus, to verify the MixColumns transformation on the first column, we need to show that

$$\begin{array}{rclcl}
 (\{02\} \cdot \{87\}) \oplus (\{03\} \cdot \{6E\}) \oplus \{46\} \oplus \{A6\} & = & \{47\} \\
 \{87\} \oplus (\{02\} \cdot \{6E\}) \oplus (\{03\} \cdot \{46\}) \oplus \{A6\} & = & \{37\} \\
 \{87\} \oplus \{6E\} \oplus (\{02\} \cdot \{46\}) \oplus (\{03\} \cdot \{A6\}) & = & \{94\} \\
 (\{03\} \cdot \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \cdot \{A6\}) & = & \{ED\}
 \end{array}$$

[Page 152]

For the first equation, we have $\{02\} \cdot \{87\} = (0000\ 1110) \oplus (0001\ 1011) = (0001\ 0101)$; and $\{03\} \cdot \{6E\} = \{6E\} \oplus (\{02\} \cdot \{6E\}) = (0110\ 1110) \oplus (1101\ 1100) = (1011\ 0010)$. Then

$$\begin{array}{rcl}
 \{02\} \cdot \{87\} & = & 0001\ 0101 \\
 \{03\} \cdot \{6E\} & = & 1011\ 0010 \\
 \{46\} & = & 0100\ 0110 \\
 \{A6\} & = & 1010\ 0110 \\
 & & 0100\ 0111 = \{47\}
 \end{array}$$

The other equations can be similarly verified.

The **inverse mix column transformation**, called `InvMixColumns`, is defined by the following matrix multiplication:

Equation 5-5

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

It is not immediately clear that [Equation \(5.5\)](#) is the **inverse** of [Equation \(5.3\)](#). We need to show that:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

which is equivalent to showing that:

Equation 5-6

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

That is, the inverse transformation matrix times the forward transformation matrix equals the identity matrix. To verify the first column of [Equation \(5.6\)](#), we need to show that:

$$(\{0E\} \cdot \{02\}) \oplus \{0B\} \oplus \{0D\} \oplus (\{09\} \cdot \{03\}) = \{01\}$$

$$(\{09\} \cdot \{02\}) \oplus \{0E\} \oplus \{0B\} \oplus (\{0D\} \cdot \{03\}) = \{00\}$$

$$(\{0D\} \cdot \{02\}) \oplus \{09\} \oplus \{0E\} \oplus (\{0B\} \cdot \{03\}) = \{00\}$$

$$(\{0B\} \cdot \{02\}) \oplus \{0D\} \oplus \{09\} \oplus (\{0E\} \cdot \{03\}) = \{00\}$$

[Page 153]

For the first equation, we have $\{0E\} \cdot \{02\} \oplus 00011100$; and $\{09\} \cdot \{03\} = \{09\} \oplus (\{09\} \cdot \{02\}) = 00001001 \oplus 00010010 = 00011011$. Then

$$\{0E\} \cdot \{02\} = 00011100$$

$$\{0B\} = 00001011$$

$$\{0D\} = 00001101$$

$$\{09\} \cdot \{03\} = 00011011$$

$$00000001$$

The other equations can be similarly verified.

The AES document describes another way of characterizing the MixColumns transformation, which is in terms of polynomial arithmetic. In the standard, MixColumns is defined by considering each column of **State** to be a four-term polynomial with coefficients in $GF(2^8)$. Each column is multiplied modulo $(x^4 + 1)$ by the fixed polynomial $a(x)$, given by

Equation 5-7

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

[Appendix 5A](#) demonstrates that multiplication of each column of **State** by $a(x)$ can be written as the matrix multiplication of [Equation \(5.3\)](#). Similarly, it can be seen that the transformation in [Equation \(5.5\)](#) corresponds to treating each column as a four-term polynomial and multiplying each column by $b(x)$, given by

Equation 5-8

$$b(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$$

It can readily be shown that $b(x) = a^{-1}(x) \pmod{(x^4 + 1)}$.

Rationale

The coefficients of the matrix in [Equation \(5.3\)](#) are based on a linear code with maximal distance between code words, which ensures a good mixing among the bytes of each column. The mix column transformation combined with the shift row transformation ensures that after a few rounds, all output bits depend on all input bits. See [\[DAEM99\]](#) for a discussion.

In addition, the choice of coefficients in MixColumns, which are all $\{01\}$, $\{02\}$, or $\{03\}$, was influenced by implementation considerations. As was discussed, multiplication by these coefficients involves at most a shift and an XOR. The coefficients in InvMixColumns are more formidable to implement. However, encryption was deemed more important than decryption for two reasons:

1.

For the CFB and OFB cipher modes ([Figures 6.5](#) and [6.6](#); described in [Chapter 6](#)), only encryption is used.

2.

As with any block cipher, AES can be used to construct a message authentication code (Part Two), and for this only encryption is used.

AddRoundKey Transformation

Forward and Inverse Transformations

In the **forward add round key transformation**, called `AddRoundKey`, the 128 bits of **State** are bitwise XORed with the 128 bits of the round key. As shown in [Figure 5.4b](#), the operation is viewed as a columnwise operation between the 4 bytes of a **State** column and one word of the round key; it can also be viewed as a byte-level operation. The following is an example of `AddRoundKey`:

[Page 154]

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

 \oplus

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

 =

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D2

The first matrix is **State**, and the second matrix is the round key.

The **inverse add round key transformation** is identical to the forward add round key transformation, because the XOR operation is its own inverse.

Rationale

The add round key transformation is as simple as possible and affects every bit of **State**. The complexity of the round key expansion, plus the complexity of the other stages of AES, ensure security.

AES Key Expansion

Key Expansion Algorithm

The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a 4-word round key for the initial `AddRoundKey` stage and each of the 10 rounds of the cipher. The following pseudocode describes the expansion:

[\[View full width\]](#)

```
KeyExpansion (byte key[16], word w[44])
{
    word temp
    for (i = 0; i < 4; i++) w[i] = (key[4*i],
    key[4*i+1],
                                key[4*i+2],
                                key[4*i+3]);

    for (i = 4; i < 44; i++)
    {
        temp = w[i-1];
        if (i mod 4 = 0) temp = SubWord (RotWord (temp))
                                 $\oplus$  Rcon[i/4];

        w[i] = w[i-4]  $\approx$  temp
    }
}
```



```
}
```

The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time. Each added word $w[i]$ depends on the immediately preceding word, $w[i-1]$, and the word four positions back, $w[i-4]$. In three out of four cases, a simple XOR is used. For a word whose position in the w array is a multiple of 4, a more complex function is used. [Figure 5.6](#) illustrates the generation of the first eight words of the expanded key, using the symbol g to represent that complex function. The function g consists of the following subfunctions:

[Page 155]

1.

RotWord performs a one-byte circular left shift on a word. This means that an input word $[b_0, b_1, b_2, b_3]$ is transformed into $[b_1, b_2, b_3, b_0]$.

2.

SubWord performs a byte substitution on each byte of its input word, using the S-box ([Table 5.4a](#)).

3.

The result of steps 1 and 2 is XORed with a round constant, $Rcon[j]$.

Figure 5.6. AES Key Expansion

36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3
----	----------	----------	----------	----------	----------	----------	----------

Rationale

The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks. The inclusion of a round-dependent round constant eliminates the symmetry, or similarity, between the ways in which round keys are generated in different rounds. The specific criteria that were used are as follows [[DAEM99](#)]:

[Page 156]

- Knowledge of a part of the cipher key or round key does not enable calculation of many other round key bits
- An invertible transformation [i.e., knowledge of any Nk consecutive words of the Expanded Key enables regeneration the entire expanded key ($Nk = \text{key size in words}$)]
- Speed on a wide range of processors
- Usage of round constants to eliminate symmetries
- Diffusion of cipher key differences into the round keys; that is, each key bit affects many round key bits
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only
- Simplicity of description

The authors do not quantify the first point on the preceding list, but the idea is that if you know less than Nk consecutive words of either the cipher key or one of the round keys, then it is difficult to reconstruct the remaining unknown bits. The fewer bits one knows, the more difficult it is to do the reconstruction or to determine other bits in the key expansion.

Equivalent Inverse Cipher

As was mentioned, the AES decryption cipher is not identical to the encryption cipher ([Figure 5.1](#)). That is, the sequence of transformations for decryption differs from that for encryption, although the form of the key schedules for encryption and decryption is the same. This has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption. There is, however, an equivalent version of the decryption algorithm that has the same structure as the encryption algorithm. The equivalent version has the same sequence of transformations as the encryption algorithm (with transformations replaced by their inverses). To achieve this equivalence, a change in key schedule is needed.

Two separate changes are needed to bring the decryption structure in line with the encryption structure. An encryption round has the structure SubBytes, ShiftRows, MixColumns, AddRoundKey. The standard decryption round has the structure InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns. Thus, the first two stages of the decryption round need to be interchanged, and the second two stages of the decryption round need to be interchanged.

Interchanging InvShiftRows and InvSubBytes

InvShiftRows affects the sequence of bytes in **State** but does not alter byte contents and does not depend on byte contents to perform its transformation. InvSubBytes affects the contents of bytes in **State** but does not alter byte sequence and does not depend on byte sequence to perform its

transformation. Thus, these two operations commute and can be interchanged. For a given **State** S_i ,

$$\text{InvShiftRows} [\text{InvSubBytes} (S_i)] = \text{InvSubBytes} [\text{InvShiftRows} (S_i)]$$

[Page 157]

Interchanging AddRoundKey and InvMixColumns

The transformations AddRoundKey and InvMixColumns do not alter the sequence of bytes in **State**. If we view the key as a sequence of words, then both AddRoundKey and InvMixColumns operate on **State** one column at a time. These two operations are linear with respect to the column input. That is, for a given **State** S_i and a given round key w_j :

$$\text{InvMixColumns} (S_i \oplus w_j) = [\text{InvMixColumns} (S_i)] \oplus [\text{InvMixColumns} (w_j)]$$

To see this, suppose that the first column of **State** S_i is the sequence (y_0, y_1, y_2, y_3) and the first column of the round key w_j is (k_0, k_1, k_2, k_3) . Then we need to show that

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \oplus k_0 \\ y_1 \oplus k_1 \\ y_2 \oplus k_2 \\ y_3 \oplus k_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \oplus \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

Let us demonstrate that for the first column entry. We need to show that:

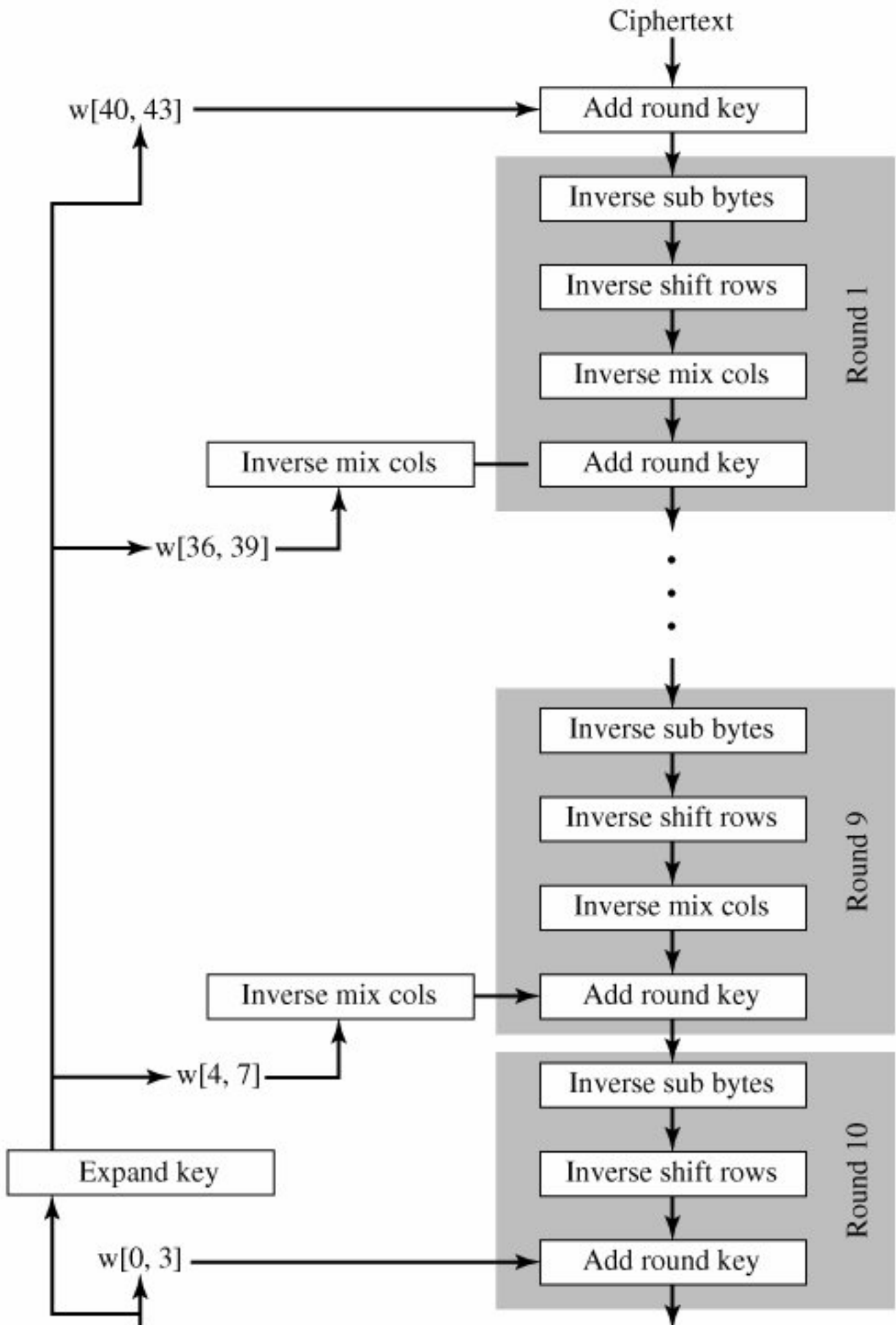
$$\begin{aligned} & [\{0E\} \cdot (y_0 \oplus k_0)] \oplus [\{0B\} \cdot (y_1 \oplus k_1)] \oplus [\{0D\} \cdot (y_2 \oplus k_2)] \oplus [\{09\} \cdot (y_3 \oplus k_3)] \\ &= [\{0E\} \cdot y_0] \oplus [\{0B\} \cdot y_1] \oplus [\{0D\} \cdot y_2] \oplus [\{09\} \cdot y_3] \\ & \oplus [\{0E\} \cdot k_0] \oplus [\{0B\} \cdot k_1] \oplus [\{0D\} \cdot k_2] \oplus [\{09\} \cdot k_3] \end{aligned}$$

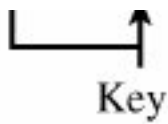
This equation is valid by inspection. Thus, we can interchange AddRoundKey and InvMixColumns, provided that we first apply InvMixColumns to the round key. Note that we do not need to apply InvMixColumns to the round key for the input to the first AddRoundKey transformation (preceding the first round) nor to the last AddRoundKey transformation (in round 10). This is because these two AddRoundKey transformations are not interchanged with InvMixColumns to produce the equivalent decryption algorithm.

[Figure 5.7](#) illustrates the equivalent decryption algorithm.

Figure 5.7. Equivalent Inverse Cipher

(This item is displayed on page 158 in the print version)





Key



Plaintext

Implementation Aspects

The Rijndael proposal [[DAEM99](#)] provides some suggestions for efficient implementation on 8-bit processors, typical for current smart cards, and on 32-bit processors, typical for PCs.

8-Bit Processor

AES can be implemented very efficiently on an 8-bit processor. AddRoundKey is a bitwise XOR operation. ShiftRows is a simple byte shifting operation. SubBytes operates at the byte level and only requires a table of 256 bytes.

The transformation MixColumns requires matrix multiplication in the field $GF(2^8)$, which means that all operations are carried out on bytes. MixColumns only requires multiplication by $\{02\}$ and $\{03\}$, which, as we have seen, involved simple shifts, conditional XORs, and XORs. This can be implemented in a more efficient way that eliminates the shifts and conditional XORs. [Equation Set \(5.4\)](#) shows the equations for the MixColumns transformation on a single column. Using the identity $\{03\} \cdot x = (\{02\} \cdot x) \oplus x$, we can rewrite [Equation Set \(5.4\)](#) as follows:

[Page 158]

Equation 5-9

$$\begin{aligned}
 Tmp &= s_{0,j} \oplus s_{1,j} \oplus s_{2,j} \oplus s_{3,j} \\
 s'_{0,j} &= s_{0,j} \oplus Tmp \oplus [2 \cdot (s_{0,j} \oplus s_{1,j})] \\
 s'_{1,j} &= s_{1,j} \oplus Tmp \oplus [2 \cdot (s_{1,j} \oplus s_{2,j})] \\
 s'_{2,j} &= s_{2,j} \oplus Tmp \oplus [2 \cdot (s_{2,j} \oplus s_{3,j})] \\
 s'_{3,j} &= s_{3,j} \oplus Tmp \oplus [2 \cdot (s_{3,j} \oplus s_{0,j})]
 \end{aligned}$$

[Equation Set \(5.9\)](#) is verified by expanding and eliminating terms.

[Page 159]

The multiplication by $\{02\}$ involves a shift and a conditional XOR. Such an implementation may be vulnerable to a timing attack of the sort described in [Section 3.4](#). To counter this attack and to increase processing efficiency at the cost of some storage, the multiplication can be replaced by a table lookup. Define the 256-byte table X2, such that $X2[i] = \{02\} \cdot i$. Then [Equation Set \(5.9\)](#) can be rewritten as

$$Tmp = s_{0,j} \oplus s_{1,j} \oplus s_{2,j} \oplus s_{3,j}$$

$$s'_{0,j} = s_{0,j} \oplus Tmp \oplus X2[s_{0,j} \oplus s_{1,j}]$$

$$s'_{1,c} = s_{1,j} \oplus Tmp \oplus X2[s_{1,j} \oplus s_{2,j}]$$

$$s'_{2,c} = s_{2,j} \oplus Tmp \oplus X2[s_{2,j} \oplus s_{3,j}]$$

$$s'_{3,j} = s_{3,j} \oplus Tmp \oplus X2[s_{3,j} \oplus s_{0,j}]$$

32-Bit Processor

The implementation described in the preceding subsection uses only 8-bit operations. For a 32-bit processor, a more efficient implementation can be achieved if operations are defined on 32-bit words. To show this, we first define the four transformations of a round in algebraic form. Suppose we begin with a **State** matrix consisting of elements $a_{i,j}$ and a round key matrix consisting of elements $k_{i,j}$. Then the transformations can be expressed as follows:

SubBytes	$b_{i,j} = S[a_{i,j}]$	
ShiftRows	$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-1} \\ b_{2,j-2} \\ b_{3,j-3} \end{bmatrix}$	
MixColumns	$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$	
AddRoundKey	$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$	

In the ShiftRows equation, the column indices are taken mod 4. We can combine all of these expressions

into a single equation:

$$\begin{aligned}
 \begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} &= \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j-1}] \\ S[a_{2,j-2}] \\ S[a_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \\
 &= \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[a_{0,j}] \right) \oplus \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[a_{1,j-1}] \right) \oplus \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[a_{2,j-2}] \right) \\
 &\quad \oplus \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[a_{3,j-3}] \right) \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}
 \end{aligned}$$

[Page 160]

In the second equation, we are expressing the matrix multiplication as a linear combination of vectors. We define four 256-word (1024-byte) tables as follows:

$T_0[x] = \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[x] \right)$	$T_1[x] = \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[x] \right)$	$T_2[x] = \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[x] \right)$	$T_3[x] = \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[x] \right)$
---	---	---	---

Thus, each table takes as input a byte value and produces a column vector (a 32-bit word) that is a function of the S-box entry for that byte value. These tables can be calculated in advance.

We can define a round function operating on a column in the following fashion:

$$\begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = T_0[s_{0,j}] \oplus T_1[s_{1,j-1}] \oplus T_2[s_{2,j-2}] \oplus T_3[s_{3,j-3}] \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

As a result, an implementation based on the preceding equation requires only four table lookups and four XORs per column per round, plus 4 Kbytes to store the table. The developers of Rijndael believe that this compact, efficient implementation was probably one of the most important factors in the selection of Rijndael for AES.



5.3. Recommended Reading and Web Sites

The most thorough description of AES so far available is the book by the developers of AES, [DAEMO2]. The authors also provide a brief description and design rationale in [DAEMO1]. [LANDO4] is a rigorous mathematical treatment of AES and its cryptanalysis.

DAEMO1 Daemen, J., and Rijmen, V. "Rijndael: The Advanced Encryption Standard." *Dr. Dobbs's Journal*, March 2001.

DAEMO2 Daemen, J., and Rijmen, V. *The Design of Rijndael: The Wide Trail Strategy Explained*. New York, Springer-Verlag, 2002.

LANDO4 Landau, S. "Polynomials in the Nation's Service: Using Algebra to Design the Advanced Encryption Standard." *American Mathematical Monthly*, February 2004.

Recommended Web Sites



- **AES home page:** NIST's page on AES. Contains the standard plus a number of other relevant documents.
- **The AES Lounge:** Contains a comprehensive bibliography of documents and papers on AES, with access to electronic copies.

5.4. Key Terms, Review Questions, and Problems

Key Terms

[Advanced Encryption Standard \(AES\)](#)

[National Institute of Standards and Technology \(NIST\)](#)

[power analysis](#)

[Rijndael](#)

[S-box](#)

Review Questions

- 5.1 What was the original set of criteria used by NIST to evaluate candidate AES ciphers?
- 5.2 What was the final set of criteria used by NIST to evaluate candidate AES ciphers?
- 5.3 What is power analysis?
- 5.4 What is the difference between Rijndael and AES?
- 5.5 What is the purpose of the **State** array?
- 5.6 How is the S-box constructed?
- 5.7 Briefly describe SubBytes.
- 5.8 Briefly describe ShiftRows.
- 5.9 How many bytes in **State** are affected by ShiftRows?

- 5.10 Briefly describe MixColumns.
- 5.11 Briefly describe AddRoundKey.
- 5.12 Briefly describe the key expansion algorithm.
- 5.13 What is the difference between SubBytes and SubWord?
- 5.14 What is the difference between ShiftRows and RotWord?
- 5.15 What is the difference between the AES decryption algorithm and the equivalent inverse cipher?

Problems

- 5.1 In the discussion of MixColumns and Inverse MixColumns, it was stated that

$$b(x) = a^{-1}(x) \text{ mod } (x^4 + 1)$$

where $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ and $b(x) = \{03\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$. Show that this is true.

[Page 162]

- 5.2
1. What is $\{01\}^{-1}$ in $GF(2^8)$?

- 2.

Verify the entry for $\{01\}$ in the S-box.

- 5.3 Show the first eight words of the key expansion for a 128-bit key of all zeros.

5.4 Given the plaintext {000102030405060708090A0B0C0D0E0F} and the key {01010101010101010101010101010101},

a.

Show the original contents of **State**, displayed as a 4 x 4 matrix.

b.

Show the value of **State** after initial AddRoundKey.

c.

Show the value of **State** after SubBytes.

d.

Show the value of **State** after ShiftRows.

e.

Show the value of **State** after MixColumns.

5.5 Verify [Equation \(5.11\)](#). That is show that $x^i (x^4 + 1) = x^{i \bmod 4}$.

5.6 Compare AES to DES. For each of the following elements of DES, indicate the comparable element in AES or explain why it is not needed in AES.

a.

XOR of subkey material with the input to the f function

b.

XOR of the f function output with the left half of the block

c.

The f function

d.

Permutation P

e.

Swapping of halves of the block

- 5.7 In the subsection on implementation aspects, it is mentioned that the use of tables helps thwart timing attacks. Suggest an alternative technique.
- 5.8 In the subsection on implementation aspects, a single algebraic equation is developed that describes the four stages of a typical round of the encryption algorithm. Provide the equivalent equation for the tenth round.
- 5.9 Compute the output of the MixColumns transformation for the following sequence of input bytes "67 89 AB CD". Apply the InvMixColumns transformation to the obtained result to verify your calculations. Change the first byte of the input from '67' to '77', perform the MixColumns transformation again for the new input, and determine how many bits have changed in the output. *Note:* You can perform all calculations by hand or write a program supporting these computations. If you choose to write a program, it should be written entirely by you; no use of libraries or public domain source code is allowed in this assignment.
- 5.10 Use the key 1010 0111 0011 1011 to encrypt the plaintext "ok" as expressed in ASCII, that is 0110 1111 0110 1011. The designers of S-AES got the ciphertext 0000 0111 0011 1000. Do you?
- 5.11 Show that the matrix given below, with entries in $GF(2^4)$, is the inverse of the matrix used in the MixColumns step of S-AES.

$$\begin{pmatrix} x^3 + 1 & x \\ x & x^3 + 1 \end{pmatrix}$$

- 5.12 Carefully write up a complete decryption of the ciphertext 0000 0111 0011 1000, using the key 1010 0111 0011 1011 and the S-AES algorithm. You should get the plaintext we started with in Problem 5.10. Note that the inverse of the S-boxes can be done with a reverse table lookup. The inverse of the MixColumns step is given by the matrix in the previous problem.

Programming Problems

- 5.13 Create software that can encrypt and decrypt using S-AES. Test data: a binary plaintext of 0110 1111 0110 1011 encrypted with a binary key of 1010 0111 0011 1011 should give a binary ciphertext of 0000 0111 0011 1000 less ecb \$\$\$). Decryption should work correspondingly

5.14 Implement a differential cryptanalysis attack on 1-round S-AES.



Appendix 5A Polynomials with Coefficients in $GF(2^8)$

In [Section 4.5](#), we discussed polynomial arithmetic in which the coefficients are in Z_p and the polynomials are defined modulo a polynomial $M(x)$ whose highest power is some integer n . In this case, addition and multiplication of coefficients occurred within the field Z_p ; that is, addition and multiplication were performed modulo p .

The AES document defines polynomial arithmetic for polynomials of degree 3 or less with coefficients in $GF(2^8)$. The following rules apply:

1.

Addition is performed by adding corresponding coefficients in $GF(2^8)$. As was pointed out [Section 4.5](#), if we treat the elements of $GF(2^8)$ as 8-bit strings, then addition is equivalent to the XOR operation. So, if we have

Equation 5-8

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

Equation 5-9

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$

then

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$$

2.

Multiplication is performed as in ordinary polynomial multiplication, with two refinements:

a.

Coefficients are multiplied in $GF(2^8)$.

b.

The resulting polynomial is reduced mod $(x^4 + 1)$.

We need to keep straight which polynomial we are talking about. Recall from [Section 4.6](#) that each element of $GF(2^8)$ is a polynomial of degree 7 or less with binary coefficients, and multiplication is carried out modulo a polynomial of degree 8. Equivalently, each element of $GF(2^8)$ can be viewed as an 8-bit byte whose bit values correspond to the binary coefficients of the corresponding polynomial. For the sets defined in this section, we are defining a polynomial ring in which each element of this ring is a polynomial of degree 3 or less with coefficients in $GF(2^8)$, and multiplication is carried out modulo a polynomial of degree 4. Equivalently, each element of this ring can be viewed as a 4-byte word whose byte values are elements of $GF(2^8)$ that correspond to the 8-bit coefficients of the corresponding polynomial.

We denote the modular product of $a(x)$ and $b(x)$ by $a(x) \otimes b(x)$. To compute $d(x) = a(x) \otimes b(x)$, the first step is to perform a multiplication without the modulo operation and to collect coefficients of like powers. Let us express this as $c(x) = a(x) \times b(x)$ Then

Equation 5-10

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

where

$$c_0 = a_0 \cdot b_0$$

$$c_1 = (a_1 \cdot b_0) \oplus (a_0 \cdot b_1)$$

$$c_2 = (a_2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2)$$

$$c_3 = (a_3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3)$$

$$c_4 = (a_3 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus (a_1 \cdot b_3)$$

$$c_5 = (a_3 \cdot b_2) \oplus (a_2 \cdot b_3)$$

$$c_6 = (a_3 \cdot b_3)$$

The final step is to perform the modulo operation:

$$d(x) = c(x) \text{ mod } (x^4 + 1)$$

That is, $d(x)$ must satisfy the equation

$$c(x) = [(x^4 + 1) \times q(x)] \oplus d(x)$$

such that the degree of $d(x)$ is 3 or less.

A practical technique for performing multiplication over this polynomial ring is based on the observation that

Equation 5-11

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4}$$

If we now combine [Equations \(5.10\)](#) and [\(5.11\)](#), we end up with

$$\begin{aligned} d(x) &= c(x) \bmod (x^4 + 1) = [c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0] \bmod (x^4 + 1) \\ &= c_3x^3 + (c_2 \oplus c_6)x^2 + (c_1 \oplus c_5)x + (c_0 \oplus c_4) \end{aligned}$$

Expanding the c_i coefficients, we have the following equations for the coefficients of $d(x)$:

$$d_0 = (a_0 \cdot b_0) \oplus (a_3 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_2 \cdot b_3)$$

$$d_1 = (a_1 \cdot b_0) \oplus (a_0 \cdot b_1) \oplus (a_3 \cdot b_2) \oplus (a_2 \cdot b_3)$$

$$d_2 = (a_2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2) \oplus (a_3 \cdot b_3)$$

$$d_3 = (a_3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3)$$

This can be written in matrix form:

Equation 5-12

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

MixColumns Transformation

In the discussion of MixColumns, it was stated that there were two equivalent ways of defining the transformation. The first is the matrix multiplication shown in [Equation \(5.3\)](#), repeated here:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

The second method is to treat each column of **State** as a four-term polynomial with coefficients in GF(2⁸). Each column is multiplied modulo (x⁴ + 1) by the fixed polynomial a(x), given by

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

From [Equation \(5.8\)](#), we have a₃ = {03}; a₂ = {01}; a₀ = {02}. For the jth column of **State**, we have the polynomial col_j(x) = s_{3,j}x³ + s_{2,j}x² + s_{1,j}x + s_{0,j}. Substituting into [Equation \(5.12\)](#), we can express d(x) = a(x) x col_j(x) as

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix}$$

which is equivalent to [Equation \(5.3\)](#).

Multiplication by x

Consider the multiplication of a polynomial in the ring by x: c(x) = x ⊗ b(x). We have

$$c(x) = x \otimes b(x) = [x \times (b_3x^3) + b_2x^2 + b_1x + b_0] \text{ mod } (x^4 + 1)$$

$$= (b_3x^4 + b_2x^3 + b_1x^2 + b_0x) \text{ mod } (x^4 + 1)$$

$$= b_2x^3 + b_1x^2 + b_0x + b_3$$

Thus, multiplication by x corresponds to a 1-byte circular left shift of the 4 bytes in the word representing the polynomial. If we represent the polynomial as a 4-byte column vector, then we have

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

◀ PREV

NEXT ▶

Appendix 5B Simplified AES

Simplified AES (S-AES) was developed by Professor Edward Schaefer of Santa Clara University and several of his students [MUSA03]. It is an educational rather than a secure encryption algorithm. It has similar properties and structure to AES with much smaller parameters. The reader might find it useful to work through an example by hand while following the discussion in this appendix. A good grasp of S-AES will make it easier for the student to appreciate the structure and workings of AES.

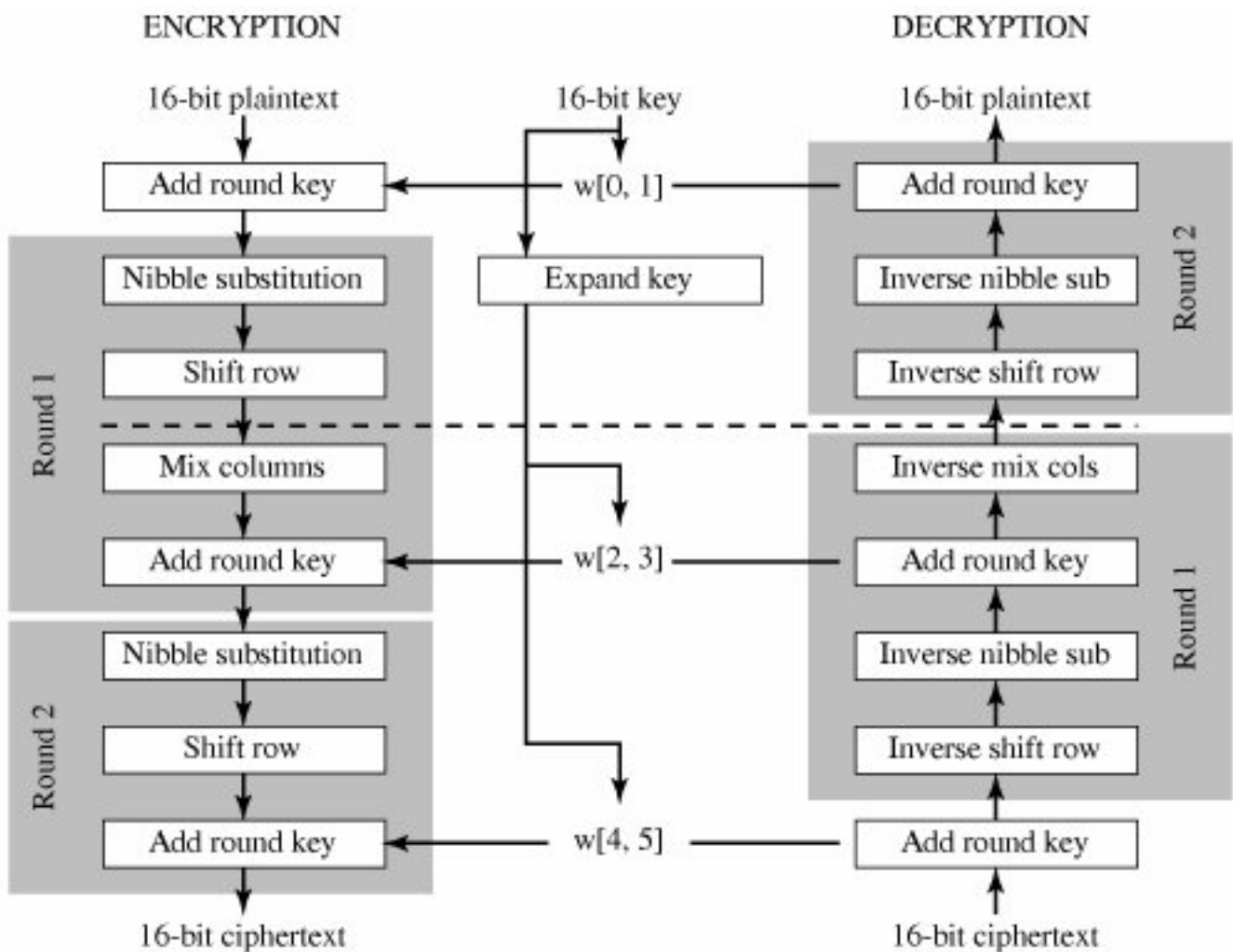
Overview

[Figure 5.8](#) illustrates the overall structure of S-AES. The encryption algorithm takes a 16-bit block of plaintext as input and a 16-bit key and produces a 16-bit block of ciphertext as output. The S-AES decryption algorithm takes an 16-bit block of ciphertext and the same 16-bit key used to produce that ciphertext as input and produces the original 16-bit block of plaintext as output.

Figure 5.8. S-AES Encryption and Decryption

(This item is displayed on page 165 in the print version)

[View full size image](#)



The encryption algorithm involves the use of four different functions, or transformations: add key (A_K), nibble substitution (NS), shift row (SR), and mix column (MC), whose operation is explained subsequently.

We can concisely express the encryption algorithm as a composition^[7] of functions:

^[7] **Definition:** If f and g are two functions, then the function F with the equation $y = F(x) = g[f(x)]$ is called the **composition** of f and g and is denoted as $F = g \circ f$.

$$A_{K_2} \circ SR \circ NS \circ A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0}$$

so that A_{K_0} is applied first.

The encryption algorithm is organized into three rounds. Round 0 is simply an add key round; round 1 is a full round of four functions; and round 2 contains only 3 functions. Each round includes the add key function, which makes use of 16 bits of key. The initial 16-bit key is expanded to 48 bits, so that each round uses a distinct 16-bit round key.

Each function operates on a 16-bit state, treated as a 2×2 matrix of nibbles, where one nibble equals 4 bits. The initial value of the state matrix is the 16-bit plaintext; the state matrix is modified by each subsequent function in the encryption process, producing after the last function the 16-bit ciphertext. As

Figure 5.9a shows, the ordering of nibbles within the matrix is by column. So, for example, the first eight bits of a 16-bit plaintext input to the encryption cipher occupy the first column of the matrix, and the second eight bits occupy the second column. The 16-bit key is similarly organized, but it is somewhat more convenient to view the key as two bytes rather than four nibbles (Figure 5.9b). The expanded key of 48 bits is treated as three round keys, whose bits are labeled as follows: $K_0 = k_0 \dots k_{15}$; $K_1 = k_{16} \dots k_{31}$; $K_2 = k_{32} \dots k_{47}$.

Figure 5.9. S-AES Data Structures

[\[View full size image\]](#)

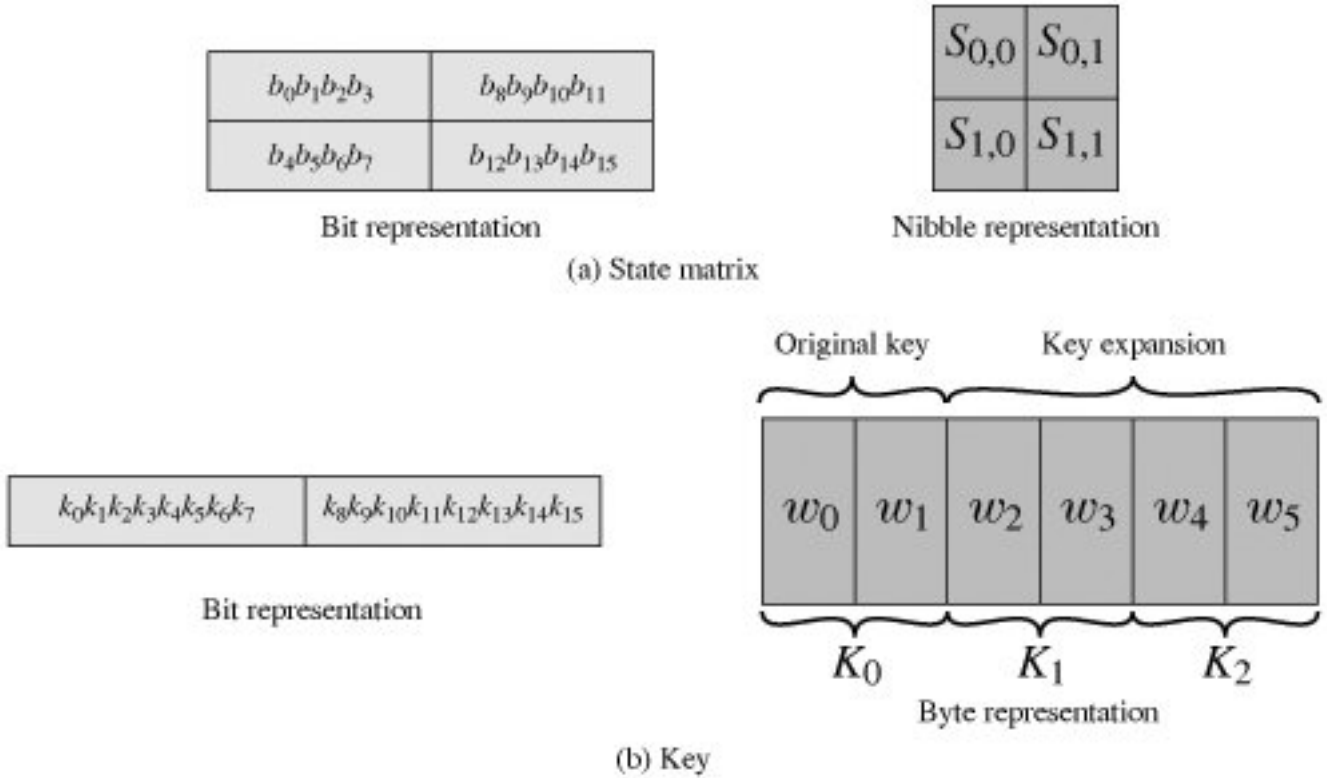
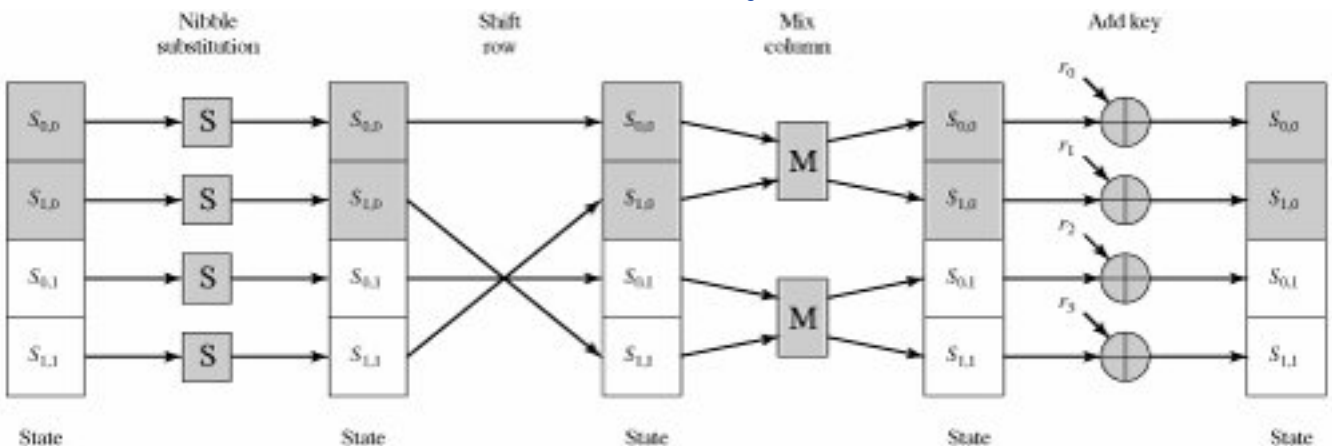


Figure 5.10 shows the essential elements of a full round of S-AES.

Figure 5.10. S-AES Encryption Round

(This item is displayed on page 167 in the print version)

[\[View full size image\]](#)



Decryption is also shown in [Figure 5.8](#) and is essentially the reverse of encryption:

$$A_{K_0} \circ \text{INS} \circ \text{ISR} \circ \text{IMC} \circ A_{K_1} \circ \text{INS} \circ \text{ISR} \circ A_{K_2}$$

[Page 168]

in which three of the functions have a corresponding inverse function: inverse nibble substitution (INS), inverse shift row (ISR), and inverse mix column (IMC).

S-AES Encryption and Decryption

We now look at the individual functions that are part of the encryption algorithm.

Add Key

The add key function consists of the bitwise XOR of the 16-bit state matrix and the 16-bit round key. [Figure 5.11](#) depicts this as a columnwise operation, but it can also be viewed as a nibble-wise or bitwise operation. The following is an example:

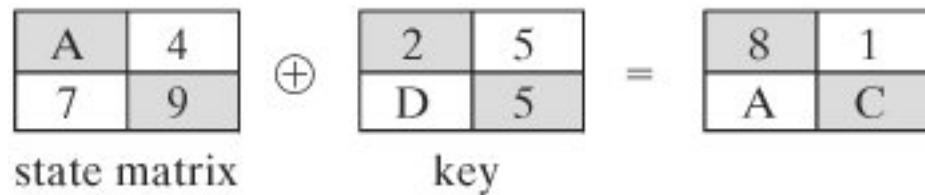
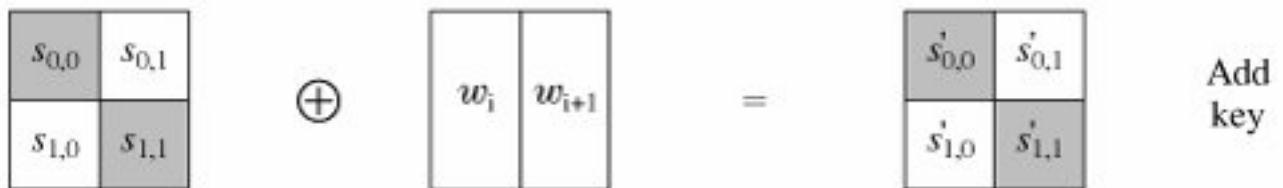
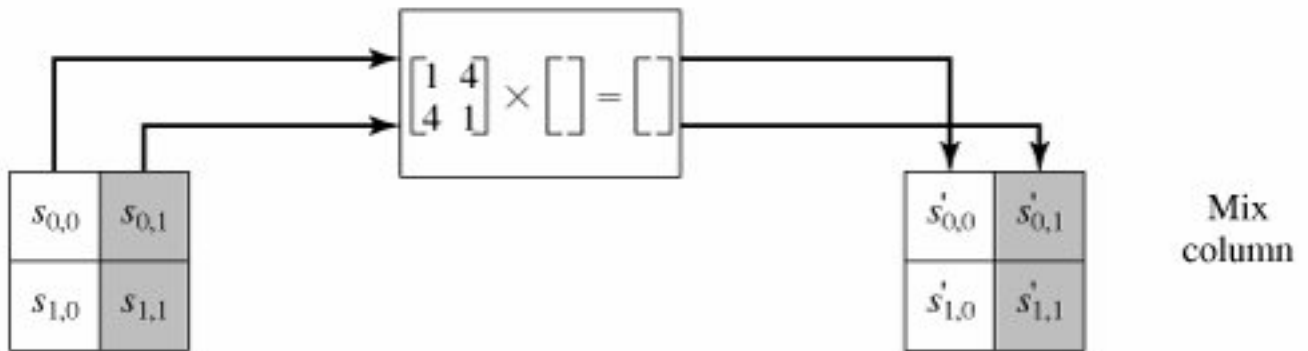
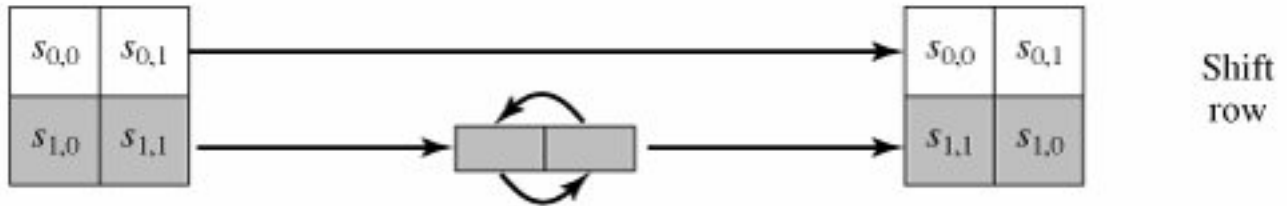
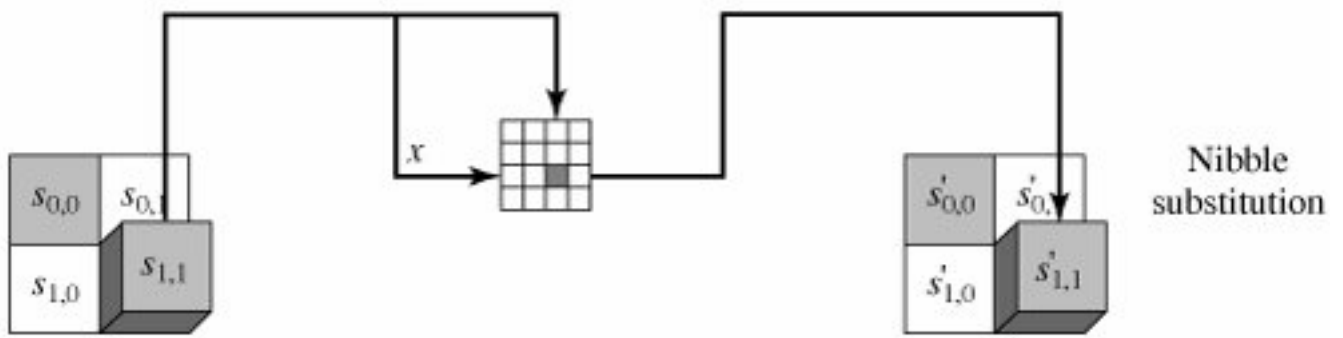


Figure 5.11. S-AES Transformations

[\[View full size image\]](#)



The inverse of the add key function is identical to the add key function, because the XOR operation is its own inverse.

The nibble substitution function is a simple table lookup ([Figure 5.11](#)). AES defines a 4 x 4 matrix of nibble values, called an S-box ([Table 5.5a](#)), that contains a permutation of all possible 4-bit values. Each individual nibble of the state matrix is mapped into a new nibble in the following way: The leftmost 2 bits of the nibble are used as a row value and the rightmost 2 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 4-bit output value. For example, the hexadecimal value A references row 2, column 2 of the S-box, which contains the value 0. Accordingly, the value A is mapped into the value 0.

Table 5.5. S-AES S-Boxes

Note: Hexadecimal numbers in shaded boxes; binary numbers in unshaded boxes.

[\[View full size image\]](#)

		<i>j</i>			
		00	01	10	11
<i>i</i>	00	9	4	A	B
	01	D	1	8	5
	10	6	2	0	3
	11	C	E	F	7

(a) S-Box

		<i>j</i>			
		00	01	10	11
<i>i</i>	00	A	5	9	B
	01	1	7	8	F
	10	6	0	2	3
	11	C	4	D	E

(b) Inverse S-Box

Here is an example of the nibble substitution transformation:

8	1	→	6	4
A	C		0	C

The inverse nibble substitution function makes use of the inverse S-box shown in [Table 5.5b](#). Note, for example, that the input 0 produces the output A, and the input A to the S-box produces 0.

Shift Row

The shift row function performs a one-nibble circular shift of the second row of the state matrix; the first row is not altered ([Figure 5.11](#)). The following is an example:

6	4	→	6	4
0	C		C	0

The inverse shift row function is identical to the shift row function, because it shifts the second row back to its original position.

Mix Column

The mix column function operates on each column individually. Each nibble of a column is mapped into a new value that is a function of both nibbles in that column. The transformation can be defined by the

following matrix multiplication on the state matrix ([Figure 5.11](#)):

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}$$

Performing the matrix multiplication, we get:

$$S'_{0,0} = S_{0,0} \oplus (4 \cdot S_{1,0})$$

$$S'_{1,0} = (4 \cdot S_{0,0}) \oplus S_{1,0}$$

$$S'_{0,1} = S_{0,1} \oplus (4 \cdot S_{1,1})$$

$$S'_{1,1} = (4 \cdot S_{0,1}) \oplus S_{1,1}$$

[Page 170]

Where arithmetic is performed in $GF(2^4)$, and the symbol \cdot refers to multiplication in $GF(2^4)$. Appendix E provides the addition and multiplication tables. The following is an example:

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 6 & 4 \\ C & 0 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 7 & 3 \end{bmatrix}$$

The inverse mix column function is defined as follows:

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}$$

We demonstrate that we have indeed defined the inverse in the following fashion:

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix}$$

The preceding matrix multiplication makes use of the following results in $GF(2^4)$: $9 + (2 \cdot 4) = 9 + 8 = 1$; $(9 \cdot 4) + 2 = 2 + 2 = 0$. These operations can be verified using the arithmetic tables in Appendix E or by polynomial arithmetic.

The mix column function is the most difficult to visualize. Accordingly, we provide an additional perspective on it in Appendix E.

Key Expansion

For key expansion, the 16 bits of the initial key are grouped into a row of two 8-bit words. [Figure 5.12](#) shows the expansion into 6 words, by the calculation of 4 new words from the initial 2 words. The algorithm is as follows:

$$w_2 = w_0 \oplus g(w_1) = w_0 \oplus \text{RCON}(1) \oplus \text{SubNib}(\text{RotNib}(w_1))$$

$$w_3 = w_2 \oplus w_1$$

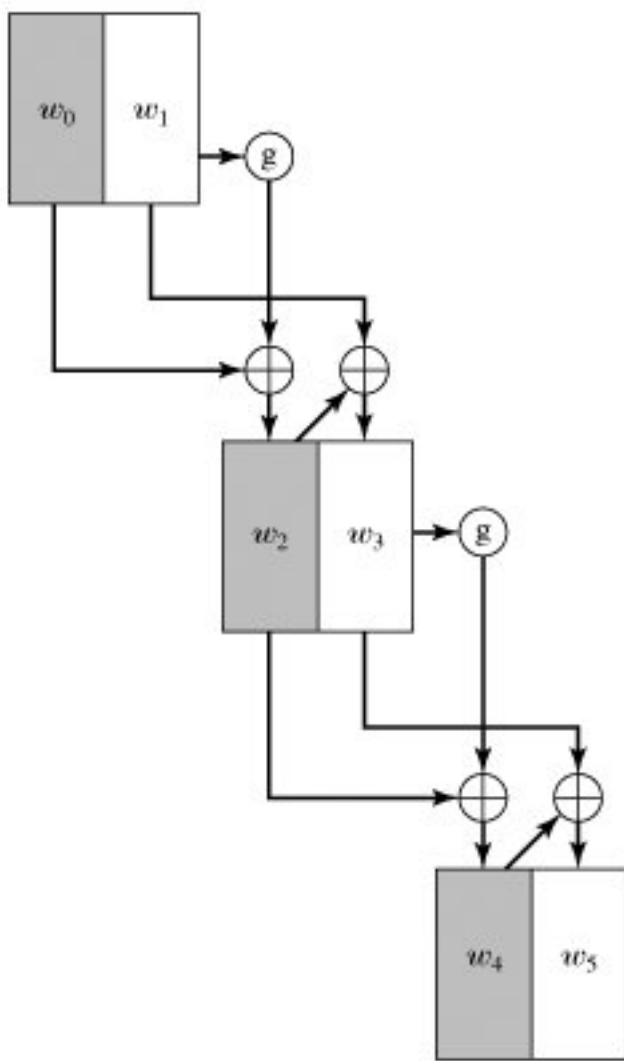
$$w_4 = w_2 \oplus g(w_3) = w_2 \oplus \text{RCON}(2) \oplus \text{SubNib}(\text{RotNib}(w_3))$$

$$w_5 = w_4 \oplus w_3$$

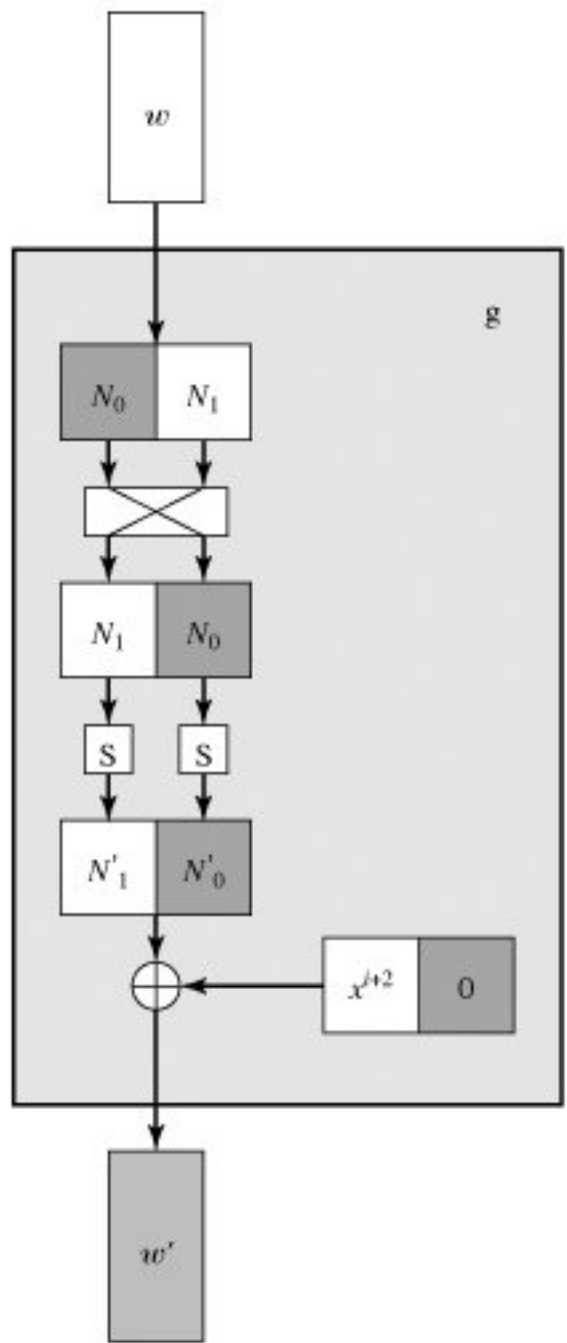
Figure 5.12. S-AES Key Expansion

(This item is displayed on page 171 in the print version)

[\[View full size image\]](#)



(a) Overall algorithm



(b) Function g

RCON is a round constant, defined as follows: $RC[i] = x^i + 2$, so that $RC[1] = x^3 = 1000$ and $RC[2] = x^4 \bmod (x^4 + x + 1) = x + 1 = 0011$. $RC[i]$ forms the leftmost nibble of a byte, with the rightmost nibble being all zeros. Thus, $RCON(1) = 10000000$ and $RCON(2) = 00110000$.

For example, suppose the key is $2D55 = 0010\ 1101\ 0101\ 0101 = w_0w_1$. Then

$$w_2 = 00101101 \oplus 10000000 \oplus \text{SubNib}(01010101)$$

$$= 00101101 \oplus 10000000 \oplus 00010001 = 10111100$$

$$w_3 = 10111100 \oplus 01010101 = 11101001$$

$$w_4 = 10111110 \oplus 00110000 \oplus \text{SubNib}(10011110)$$

$$= 10111100 \oplus 00110000 \oplus 00101111 = 10100011$$

$$w_5 = 10100011 \oplus 11101001 = 01001010$$

The S-Box

The S-box is constructed as follows:

1.

Initialize the S-box with the nibble values in ascending sequence row by row. The first row contains the hexadecimal values 0, 1, 2, 3; the second row contains 4, 5, 6, 7; and so on. Thus, the value of the nibble at row i , column j is $4i + j$.

[Page 171]

2.

Treat each nibble as an element of the finite field $\text{GF}(2^4)$ modulo $x^4 + x + 1$. Each nibble $a_0a_1a_2a_3$ represents a polynomial of degree 3.

3.

Map each byte in the S-box to its multiplicative inverse in the finite field $\text{GF}(2^4)$ modulo $x^4 + x + 1$; the value 0 is mapped to itself.

4.

Consider that each byte in the S-box consists of 4 bits labeled (b_0, b_1, b_2, b_3) . Apply the following transformation to each bit of each byte in the S-box: The AES standard depicts this transformation in matrix form as follows:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The prime (') indicates that the variable is to be updated by the value on the right. Remember that addition and multiplication are being calculated modulo 2.

[Table 5.5a](#) shows the resulting S-box. This is a nonlinear, invertible matrix. The inverse S-box is shown in [Table 5.5b](#).

S-AES Structure

We can now examine several aspects of interest concerning the structure of AES. First, note that the encryption and decryption algorithms begin and end with the add key function. Any other function, at the beginning or end, is easily reversible without knowledge of the key and so would add no security but just a processing overhead. Thus, there is a round 0 consisting of only the add key function.

The second point to note is that round 2 does not include the mix column function. The explanation for this in fact relates to a third observation, which is that although the decryption algorithm is the reverse of the encryption algorithm, as clearly seen in [Figure 5.8](#), it does not follow the same sequence of functions. Thus

$$\text{Encryption: } A_{K_2} \circ \text{SR} \circ \text{NS} \circ A_{K_1} \circ \text{MC} \circ \text{SR} \circ \text{NS} \circ A_{K_0}$$

$$\text{Decryption: } A_{K_0} \circ \text{INS} \circ \text{ISR} \circ \text{IMC} \circ A_{K_1} \circ \text{INS} \circ \text{ISR} \circ A_{K_2}$$

From an implementation point of view, it would be desirable to have the decryption function follow the same function sequence as encryption. This allows the decryption algorithm to be implemented in the same way as the encryption algorithm, creating opportunities for efficiency.

Note that if we were able to interchange the second and third functions, the fourth and fifth functions, and the sixth and seventh functions in the decryption sequence, we would have the same structure as the encryption algorithm. Let's see if this is possible. First, consider the interchange of INS and ISR. Given a state N consisting of the nibbles (N_0, N_1, N_2, N_3) the transformation $\text{INS}(\text{ISR}(N))$ proceeds as follows:

$$\begin{pmatrix} N_0 & N_2 \\ N_1 & N_3 \end{pmatrix} \rightarrow \begin{pmatrix} N_0 & N_2 \\ N_3 & N_1 \end{pmatrix} \rightarrow \begin{pmatrix} \text{IS}[N_0] & \text{IS}[N_2] \\ \text{IS}[N_3] & \text{IS}[N_1] \end{pmatrix}$$

Where IS refers to the inverse S-Box. Reversing the operations, the transformation $\text{ISR}(\text{INS}(M))$ proceeds as follows:

$$\begin{pmatrix} N_0 & N_2 \\ N_1 & N_3 \end{pmatrix} \rightarrow \begin{pmatrix} \text{IS}[N_0] & \text{IS}[N_2] \\ \text{IS}[N_1] & \text{IS}[N_3] \end{pmatrix} \rightarrow \begin{pmatrix} \text{IS}[N_0] & \text{IS}[N_2] \\ \text{IS}[N_3] & \text{IS}[N_1] \end{pmatrix}$$

which is the same result. Thus, $\text{INS}(\text{ISR}(M)) = \text{ISR}(\text{INS}(M))$.

Now consider the operation of inverse mix column followed by add key: $\text{IMC}(A_{K_1}(N))$ where the round key K_1 consists of the nibbles $(k_{0,0}, k_{1,0}, k_{0,1}, k_{1,1})$ Then:

$$\begin{aligned}
\begin{pmatrix} 9 & 2 \\ 2 & 9 \end{pmatrix} \left(\begin{pmatrix} k_{0,0} & k_{0,1} \\ k_{1,0} & k_{1,1} \end{pmatrix} \oplus \begin{pmatrix} N_0 & N_2 \\ N_1 & N_3 \end{pmatrix} \right) &= \begin{pmatrix} 9 & 2 \\ 2 & 9 \end{pmatrix} \begin{pmatrix} k_{0,0} \oplus N_0 & k_{0,1} \oplus N_2 \\ k_{1,0} \oplus N_1 & k_{1,1} \oplus N_3 \end{pmatrix} \\
&= \begin{pmatrix} 9(k_{0,0} \oplus N_0) \oplus 2(k_{1,0} \oplus N_1) & 9(k_{0,1} \oplus N_2) \oplus 2(k_{1,1} \oplus N_3) \\ 2(k_{0,0} \oplus N_0) \oplus 9(k_{1,0} \oplus N_1) & 2(k_{0,1} \oplus N_2) \oplus 9(k_{1,1} \oplus N_3) \end{pmatrix} \\
&= \begin{pmatrix} (9k_{0,0} \oplus 2k_{1,0}) \oplus (9N_0 \oplus 2N_1) & (9k_{0,1} \oplus 2k_{1,1}) \oplus (9N_2 \oplus 2N_3) \\ (2k_{0,0} \oplus 9k_{1,0}) \oplus (2N_0 \oplus 9N_1) & (2k_{0,1} \oplus 9k_{1,1}) \oplus (2N_2 \oplus 9N_3) \end{pmatrix} \\
&= \begin{pmatrix} (9k_{0,0} \oplus 2k_{1,0}) & (9k_{0,1} \oplus 2k_{1,1}) \\ (2k_{0,0} \oplus 9k_{1,0}) & (2k_{0,1} \oplus 9k_{1,1}) \end{pmatrix} \oplus \begin{pmatrix} (9N_0 \oplus 2N_1) & (9N_2 \oplus 2N_3) \\ (2N_0 \oplus 9N_1) & (2N_2 \oplus 9N_3) \end{pmatrix} \\
&= \begin{pmatrix} 9 & 2 \\ 2 & 9 \end{pmatrix} \begin{pmatrix} k_{0,0} & k_{0,1} \\ k_{1,0} & k_{1,1} \end{pmatrix} \oplus \begin{pmatrix} 9 & 2 \\ 2 & 9 \end{pmatrix} \begin{pmatrix} N_0 & N_2 \\ N_1 & N_3 \end{pmatrix}
\end{aligned}$$

[Page 173]

All of the above steps make use of the properties of finite field arithmetic. The result is that $\text{IMC}(A_{K_1}(N)) = \text{IMC}(K_1) \oplus \text{IMC}(N)$. Now let us define the inverse round key for round 1 to be $\text{IMC}(K_1)$ and the inverse add key operation IA_{K_1} to be the bitwise XOR of the inverse round key with the state vector. Then we have $\text{IMC}(A_{K_1}(N)) = \text{IA}_{K_1}(\text{IMC}(N))$. As a result, we can write the following:

Encryption: $A_{K_2} \circ \text{SR} \circ \text{NS} \circ A_{K_1} \circ \text{MC} \circ \text{SR} \circ \text{NS} \circ A_{K_0}$

Decryption: $A_{K_0} \circ \text{INS} \circ \text{ISR} \circ \text{IMC} \circ A_{K_1} \circ \text{INS} \circ \text{ISR} \circ A_{K_2}$

Decryption: $A_{K_0} \circ \text{ISR} \circ \text{INS} \circ A_{\text{IMC}(K_1)} \circ \text{IMC} \circ \text{ISR} \circ \text{INS} \circ A_{K_2}$

Both encryption and decryption now follow the same sequence. Note that this derivation would not work as effectively if round 2 of the encryption algorithm included the MC function. In that case, we would have

Encryption: $A_{K_2} \circ \text{MC} \circ \text{SR} \circ \text{NS} \circ A_{K_1} \circ \text{MC} \circ \text{SR} \circ \text{NS} \circ A_{K_0}$

Decryption: $A_{K_0} \circ \text{INS} \circ \text{ISR} \circ \text{IMC} \circ A_{K_1} \circ \text{INS} \circ \text{ISR} \circ \text{IMC} \circ A_{K_2}$

There is now no way to interchange pairs of operations in the decryption algorithm so as to achieve the same structure as the encryption algorithm.

Chapter 6. More on Symmetric Ciphers

6.1 Multiple Encryption and Triple DES

[Double DES](#)

[Triple DES with Two Keys](#)

[Triple DES with Three Keys](#)

6.2 Block Cipher Modes of Operation

[Electronic Codebook Mode](#)

[Cipher Block Chaining Mode](#)

[Cipher Feedback Mode](#)

[Output Feedback Mode](#)

[Counter Mode](#)

6.3 Stream Ciphers and RC4

[Stream Cipher Structure](#)

[The RC4 Algorithm](#)

6.4 Recommended Reading and Web Site

6.5 Key Terms, Review Questions, and Problems

[Key Terms](#)

[Review Questions](#)

[Problems](#)

"I am fairly familiar with all the forms of secret writings, and am myself the author of a trifling monograph upon the subject, in which I analyze one hundred and sixty separate ciphers," said Holmes.

The Adventure of the Dancing Men, Sir Arthur Conan Doyle

Key Points

- Multiple encryption is a technique in which an encryption algorithm is used multiple times. In the first instance, plaintext is converted to ciphertext using the encryption algorithm. This ciphertext is then used as input and the algorithm is applied again. This process may be repeated through any number of stages.
- Triple DES makes use of three stages of the DES algorithm, using a total of two or three distinct keys.
- A mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream.
- Five modes of operation have been standardized for use with symmetric block ciphers such as DES and AES: electronic codebook mode, cipher block chaining mode, cipher feedback mode, output feedback mode, and counter mode.
- A stream cipher is a symmetric encryption algorithm in which ciphertext output is produced bit-by-bit or byte-by-byte from a stream of plaintext input. The most widely used such cipher is RC4.

This chapter continues our discussion of symmetric ciphers. We begin with the topic of multiple encryption, looking in particular at the most widely used multiple-encryption scheme: triple DES.

The chapter next turns to the subject of block cipher modes of operation. We find that there are a number of different ways to apply a block cipher to plaintext, each with its own advantages and particular applications.

Finally, this chapter addresses the subject of symmetric stream ciphers, which differ in significant ways from symmetric block ciphers. We also look at the most important such cipher, RC4.

6.1. Multiple Encryption and Triple DES

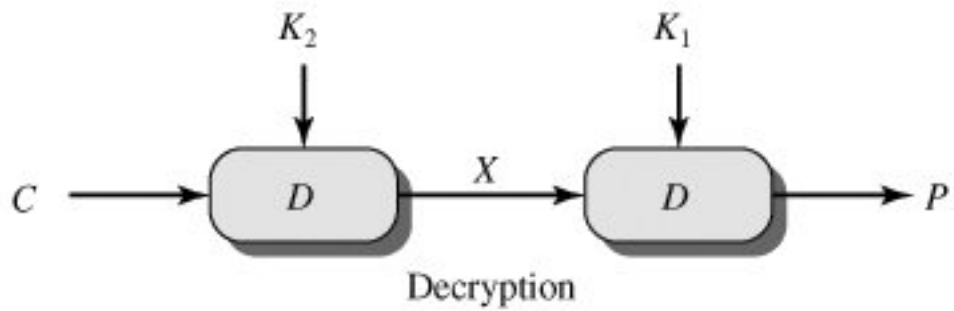
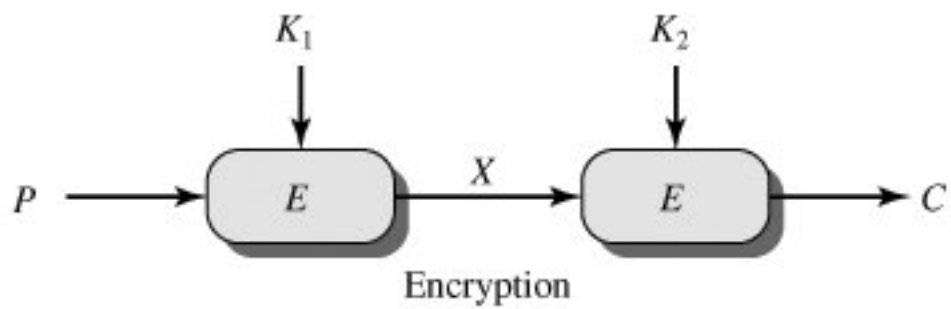
Given the potential vulnerability of DES to a brute-force attack, there has been considerable interest in finding an alternative. One approach is to design a completely new algorithm, of which AES is a prime example. Another alternative, which would preserve the existing investment in software and equipment, 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 (3DES) approach.

Double DES

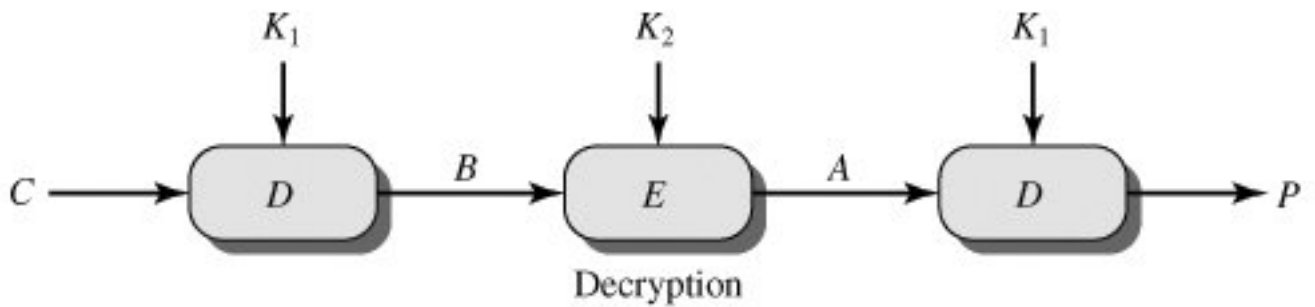
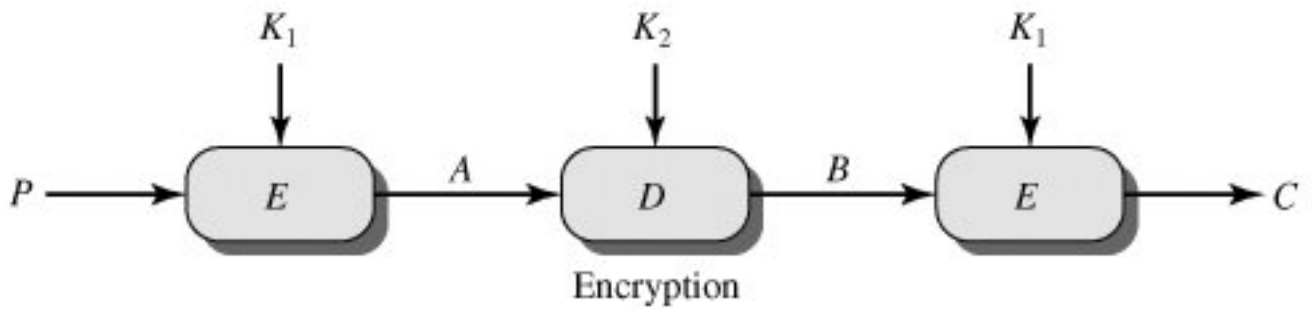
The simplest form of multiple encryption has two encryption stages and two keys ([Figure 6.1a](#)). Given a plaintext P and two encryption keys K_1 and K_2 , ciphertext C is generated as

$$C = E(K_2, E(K_1, P))$$

Figure 6.1. Multiple Encryption



(a) Double encryption



(b) Triple encryption

Decryption requires that the keys be applied in reverse order:

$$P = D(K_1, D(K_2, C))$$

For DES, this scheme apparently involves a key length of $56 \times 2 = 112$ bits, of 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

Equation 6-1

$$E(K_2, E(K_1, P)) = E(K_3, P)$$

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 \(6.1\)](#) 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

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

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 are 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 a 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 6.1a](#))

$$X = E(K_1, P) = D(K_2, P)$$

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.

[Page 178]

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 more 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 6.1b](#)):

$$C = E(K_1, D(K_2, E(K_1, P)))$$

There is no cryptographic significance to the use of decryption for the second stage. Its only advantage is that it allows users of 3DES to decrypt data encrypted by users of the older single DES:

$$C = E(K_1, D(K_1, E(K_1, P))) = E(K_1, P)$$

3DES 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. [\[1\]](#)

^[1] (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.

Currently, there are no practical cryptanalytic attacks on 3DES. Coppersmith [[COPP94](#)] notes that the cost of a brute-force key search on 3DES 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} .

It is worth looking at several proposed attacks on 3DES 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.

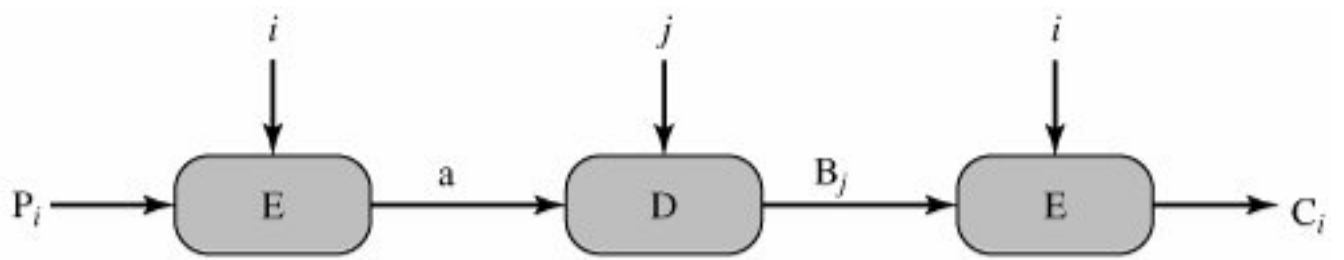
The first serious proposal came from Merkle and Hellman [[MERK81](#)]. Their plan involves finding plaintext values that produce a first intermediate value of $A = 0$ ([Figure 6.1b](#)) and then using the meet-in-the-middle attack to determine the two keys. The level of effort is 2^{56} , but the technique 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 [[VANO90](#)]. 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 6.1b](#)), 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:

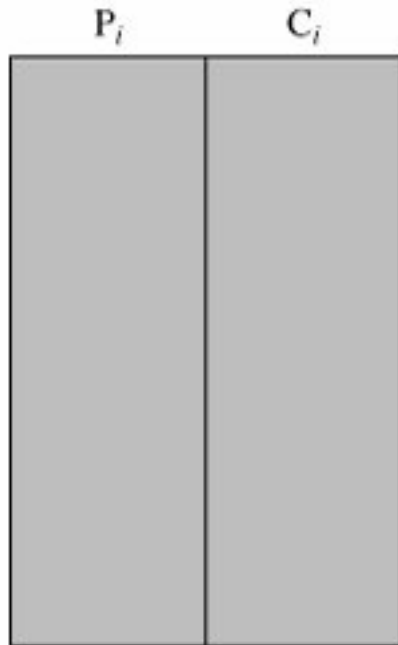
- 1.

Obtain n (P, C) pairs. This is the known plaintext. Place these in a table ([Table 1](#)) sorted on the values of P ([Figure 6.2b](#)).

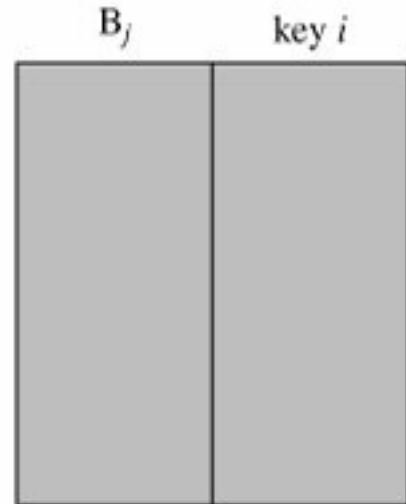
Figure 6.2. Known-Plaintext Attack on Triple DES



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



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



(c) Table of intermediate values and candidate keys

2.

Pick an arbitrary value a for A , and create a second table ([Figure 6.2c](#)) 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 :

$$P_i = D(i, a)$$

[Page 180]

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 6.2a](#)).

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 3DES may feel some concern. Thus, many researchers now feel that three-key 3DES is the preferred alternative (e.g., [[KALI96a](#)]). Three-key 3DES 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 3DES, including PGP and S/MIME, both discussed in [Chapter 15](#).

6.2. Block Cipher Modes of Operation

A block cipher algorithm is a basic building block for providing data security. To apply a block cipher in a variety of applications, four "modes of operation" have been defined by NIST (FIPS 81). In essence, a mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream. The four modes are intended to cover virtually all the possible applications of encryption for which a block cipher could be used. As new applications and requirements have appeared, NIST has expanded the list of recommended modes to five in Special Publication 800-38A. These modes are intended for use with any symmetric block cipher, including triple DES and AES. The modes are summarized in [Table 6.1](#) and described briefly in the remainder of this section.

Table 6.1. Block Cipher 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"> Secure transmission of single values (e.g., an encryption key)
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"> General-purpose block-oriented transmission Authentication
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"> General-purpose stream-oriented transmission Authentication
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"> Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	<ul style="list-style-type: none"> General-purpose block-oriented transmission Useful for high-speed requirements

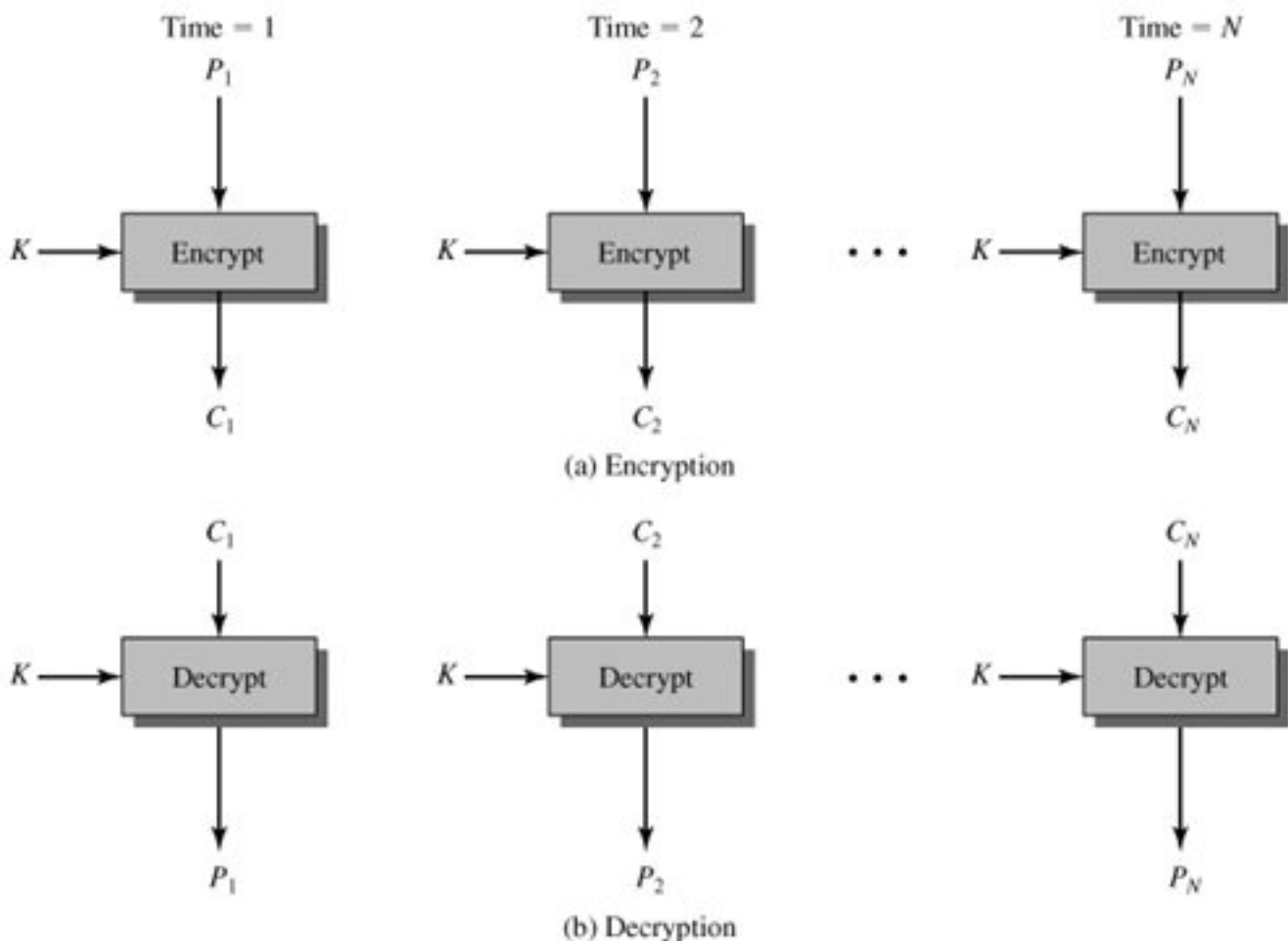
Electronic Codebook Mode

The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key (Figure 6.3). The term *codebook* is used because, for a given key, there is a unique ciphertext for every b -bit block of plaintext. Therefore, we can imagine a gigantic codebook in which there is an entry for every possible b -bit plaintext pattern showing its corresponding ciphertext.

[Page 182]

Figure 6.3. Electronic Codebook (ECB) Mode

[\[View full size image\]](#)



For a message longer than b bits, the procedure is simply to break the message into b -bit blocks, padding the last block if necessary. Decryption is performed one block at a time, always using the same key. In Figure 6.3, the plaintext (padded as necessary) consists of a sequence of b -bit blocks, P_1, P_2, \dots, P_N ; the corresponding sequence of ciphertext blocks is C_1, C_2, \dots, C_N .

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 b -bit block of plaintext, if it appears more than once in the message, always produces the same ciphertext.

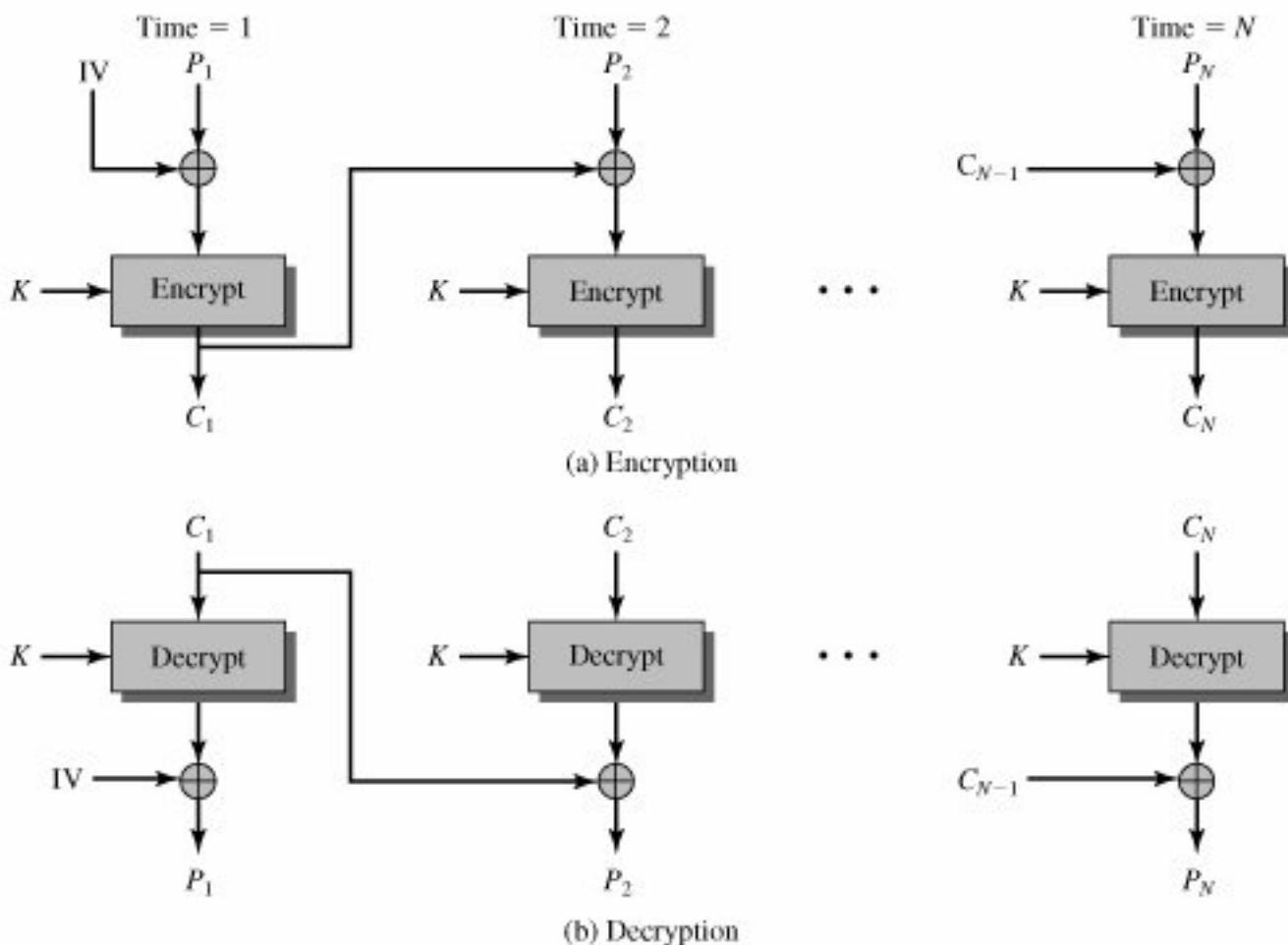
For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities. For example, if it is known that the message always starts out with certain predefined fields, then the cryptanalyst may have a number of known plaintext-ciphertext pairs to work with. If the message has repetitive elements, with a period of repetition a multiple of b 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 6.4). 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 plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating patterns of b bits are not exposed.

Figure 6.4. Cipher Block Chaining (CBC) Mode

[\[View full size image\]](#)



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

$$C_j = E(K, [C_{j-1} \oplus P_j])$$

Then

$$D(K, C_j) = D(K, E(K, [C_{j-1} \oplus P_j]))$$

$$D(K, C_j) = C_{j-1} \oplus P_j$$

$$C_{j-1} \oplus D(K, C_j) = C_{j-1} \oplus C_{j-1} \oplus P_j = P_j$$

[Page 184]

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 is a data block that is that same size as the cipher block.

The IV must be known to both the sender and receiver but be unpredictable by a third party. For maximum security, the IV should be protected against unauthorized changes. 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 b -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 b bits.

In addition to its use to achieve confidentiality, the CBC mode can be used for authentication. This use is described in [Part Two](#).

Cipher Feedback Mode

The DES scheme is essentially a block cipher technique that uses b -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.

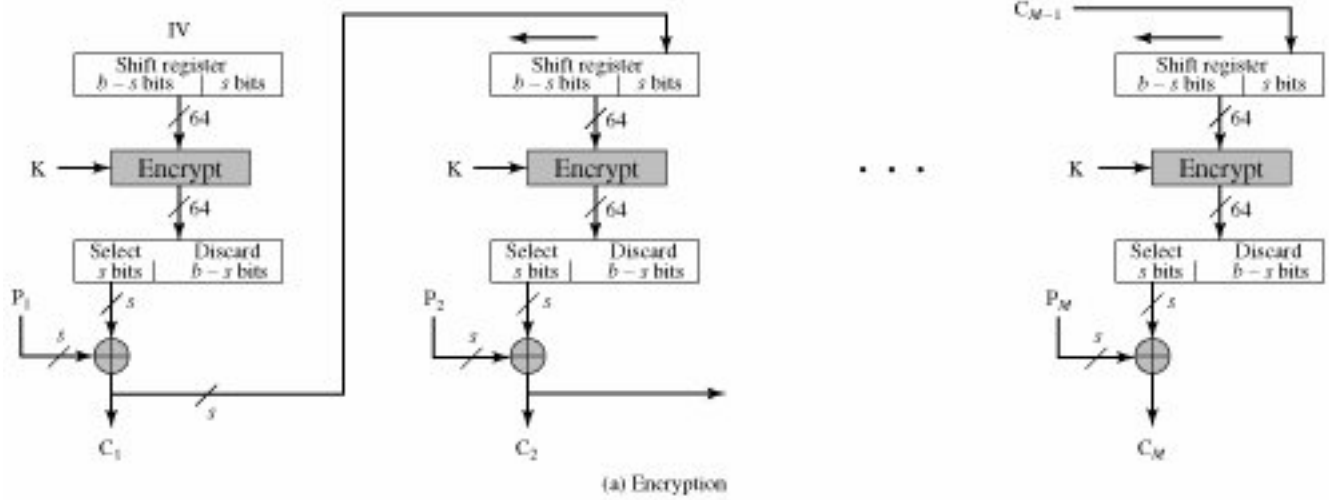
One desirable property of a stream cipher is that the ciphertext be of the same length as the plaintext. Thus, if 8-bit characters are being transmitted, each character should be encrypted to produce a ciphertext output of 8 bits. If more than 8 bits are produced, transmission capacity is wasted.

[Figure 6.5](#) depicts the CFB scheme. In the figure, it is assumed that the unit of transmission is s bits; a common value is $s = 8$. 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. In this case, rather than units of b bits, the plaintext is divided into *segments* of s bits.

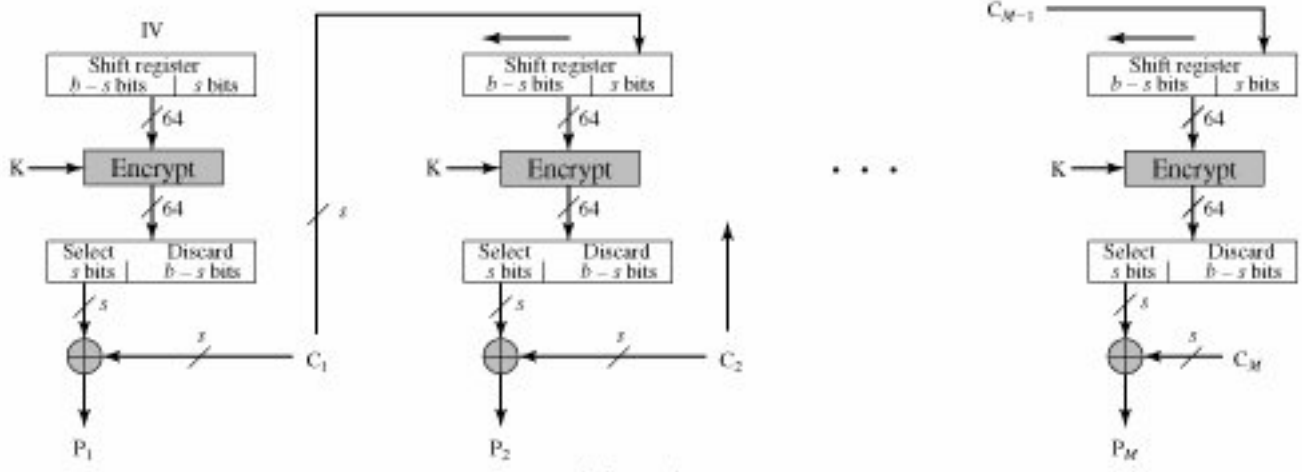
[Page 185]

Figure 6.5. s -bit Cipher Feedback (CFB) Mode

[View full size image](#)



(a) Encryption



(b) Decryption

First, consider encryption. The input to the encryption function is a b -bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant) s bits of the output of the encryption function are XORed with the first segment 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 s bits and C_1 is placed in the rightmost (least significant) s bits of the shift register. This process continues until all plaintext units have been encrypted.

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 $S_s(X)$ be defined as the most significant s bits of X . Then

$$C_1 = P_1 \oplus S_s[E(K, IV)]$$

Therefore,

$$P_1 = C_1 \oplus S_s[E(K, IV)]$$

The same reasoning holds for subsequent steps in the process.

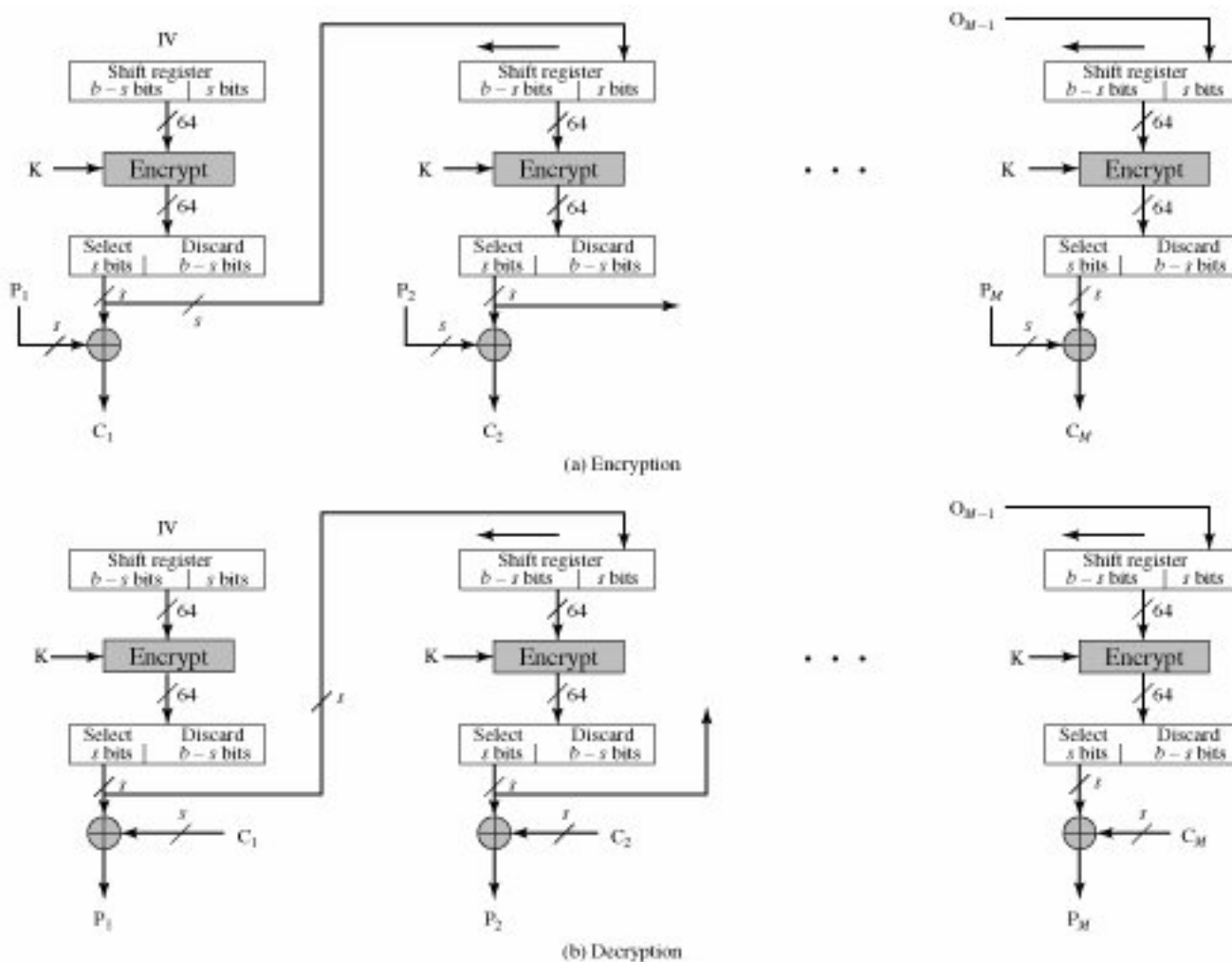
Output Feedback Mode

The output feedback (OFB) mode is similar in structure to that of CFB, as illustrated in [Figure 6.6](#). 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.

Figure 6.6. s-bit Output Feedback (OFB) Mode

(This item is displayed on page 187 in the print version)

[\[View full size image\]](#)



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 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].

Counter Mode

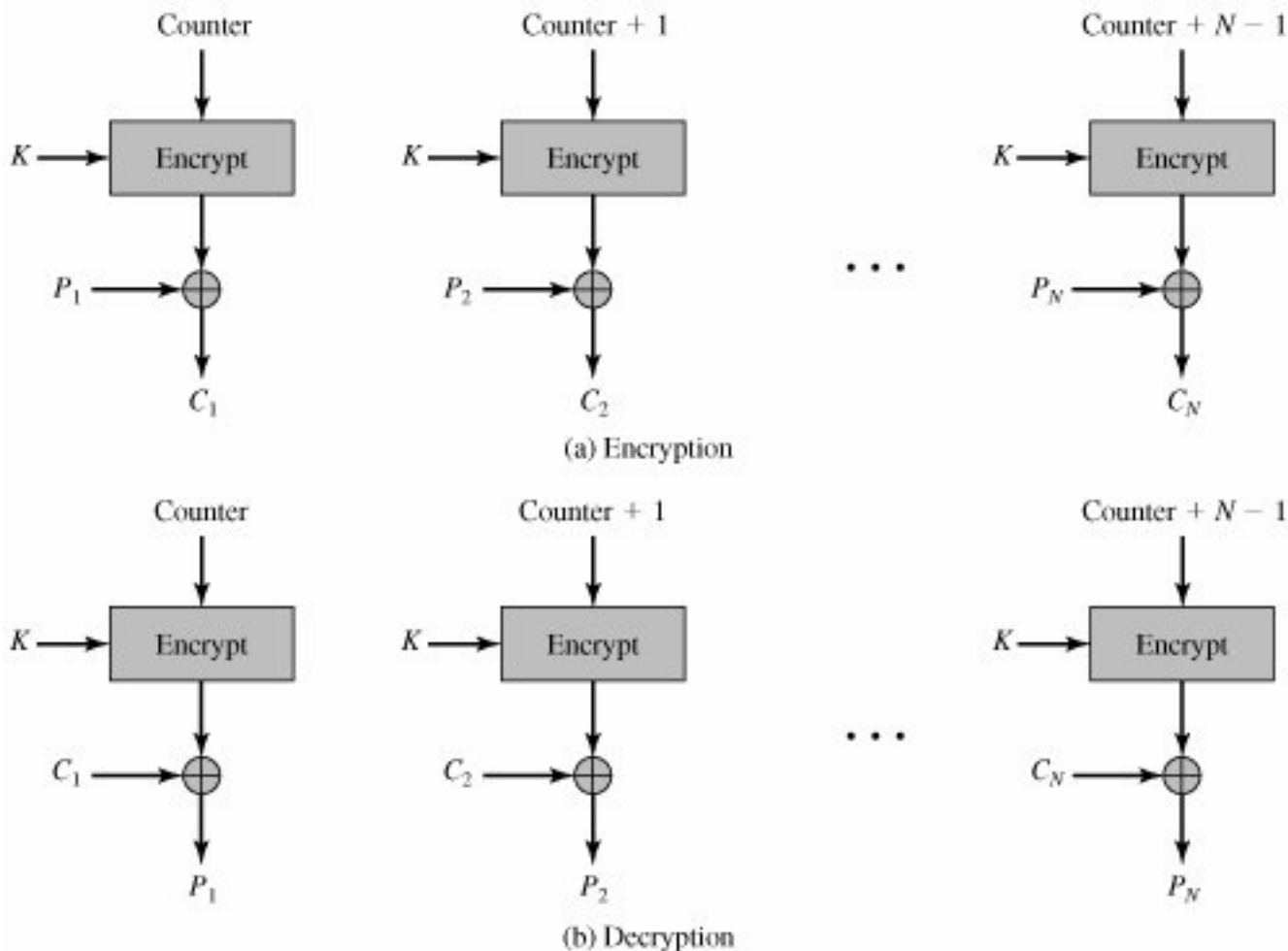
Although interest in the counter mode (CTR) has increased recently, with applications to ATM (asynchronous transfer mode) network security and IPsec (IP security), this mode was proposed early on (e.g., [DIFF79]).

Figure 6.7 depicts the CTR mode. A counter, equal to the plaintext block size is used. The only requirement stated in SP 800-38A is that the counter value must be different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^b where b is the block size). For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.

[Page 188]

Figure 6.7. Counter (CTR) Mode

[View full size image](#)



[LIPMOO] lists the following advantages of CTR mode:

- **Hardware efficiency:** Unlike the three chaining modes, encryption (or decryption) in CTR mode can be done in parallel on multiple blocks of plaintext or ciphertext. For the chaining modes, the algorithm must complete the computation on one block before beginning on the next block. This limits the maximum throughput of the algorithm to the reciprocal of the time for one execution of block encryption or decryption. In CTR mode, the throughput is only limited by the amount of parallelism that is achieved.
 - **Software efficiency:** Similarly, because of the opportunities for parallel execution in CTR mode, processors that support parallel features, such as aggressive pipelining, multiple instruction dispatch per clock cycle, a large number of registers, and SIMD instructions, can be effectively utilized.
 - **Preprocessing:** The execution of the underlying encryption algorithm does not depend on input of the plaintext or ciphertext. Therefore, if sufficient memory is available and security is maintained, preprocessing can be used to prepare the output of the encryption boxes that feed into the XOR functions in [Figure 6.7](#). When the plaintext or ciphertext input is presented, then the only computation is a series of XORs. Such a strategy greatly enhances throughput.
-

[Page 189]

- **Random access:** The i th block of plaintext or ciphertext can be processed in random-access fashion. With the chaining modes, block C_i cannot be computed until the $i - 1$ prior block are computed. There may be applications in which a ciphertext is stored and it is desired to decrypt just one block; for such applications, the random access feature is attractive.
- **Provable security:** It can be shown that CTR is at least as secure as the other modes discussed in this section.
- **Simplicity:** Unlike ECB and CBC modes, CTR mode requires only the implementation of the encryption algorithm and not the decryption algorithm. This matters most when the decryption algorithm differs substantially from the encryption algorithm, as it does for AES. In addition, the decryption key scheduling need not be implemented.

6.3. Stream Ciphers and RC4

In this section we look at perhaps the most popular symmetric stream cipher, RC4. We begin with an overview of stream cipher structure, and then examine RC4.

Stream Cipher Structure

A typical stream cipher encrypts plaintext one byte at a time, although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time. [Figure 6.8](#) is a representative diagram of stream cipher structure. In this structure a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. We discuss pseudorandom number generators in [Chapter 7](#). For now, we simply say that a pseudorandom stream is one that is unpredictable without knowledge of the input key. The output of the generator, called a **keystream**, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation. For example, if the next byte generated by the generator is 01101100 and the next plaintext byte is 11001100, then the resulting ciphertext byte is

[Page 190]

```

11001100  plaintext
⊕ 01101100  key stream
  -----
10100000  ciphertext

```

Decryption requires the use of the same pseudorandom sequence:

```

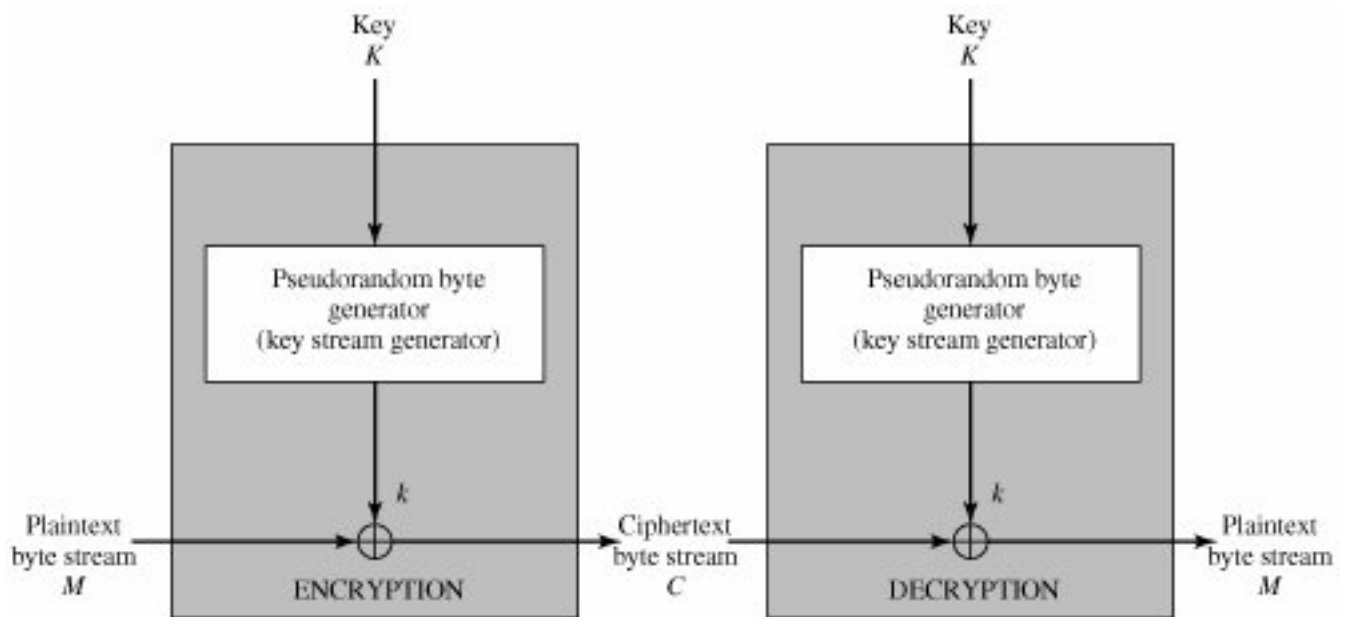
10100000  ciphertext
⊕ 01101100  key stream
  -----
11001100  plaintext

```

Figure 6.8. Stream Cipher Diagram

(This item is displayed on page 189 in the print version)

[\[View full size image\]](#)



The stream cipher is similar to the one-time pad discussed in [Chapter 2](#). The difference is that a one-time pad uses a genuine random number stream, whereas a stream cipher uses a pseudorandom number stream.

[[KUMA97](#)] lists the following important design considerations for a stream cipher:

1.

The encryption sequence should have a large period. A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats. The longer the period of repeat the more difficult it will be to do cryptanalysis. This is essentially the same consideration that was discussed with reference to the Vigenère cipher, namely that the longer the keyword the more difficult the cryptanalysis.

2.

The keystream should approximate the properties of a true random number stream as close as possible. For example, there should be an approximately equal number of 1s and 0s. If the keystream is treated as a stream of bytes, then all of the 256 possible byte values should appear approximately equally often. The more random-appearing the keystream is, the more randomized the ciphertext is, making cryptanalysis more difficult.

3.

Note from [Figure 6.8](#) that the output of the pseudorandom number generator is conditioned on the value of the input key. To guard against brute-force attacks, the key needs to be sufficiently long. The same considerations as apply for block ciphers are valid here. Thus, with current technology, a key length of at least 128 bits is desirable.

With a properly designed pseudorandom number generator, a stream cipher can be as secure as block cipher of comparable key length. The primary advantage of a stream cipher is that stream ciphers are almost always faster and use far less code than do block ciphers. The example in this section, RC4, can be implemented in just a few lines of code. [Table 6.2](#), using data from [[RESCO1](#)], compares execution times of RC4 with three well-known symmetric block ciphers. The advantage of a block cipher is that you can reuse keys. However, if two plaintexts are encrypted with the same key using a stream cipher,

then cryptanalysis is often quite simple [DAWS96]. If the two ciphertext streams are XORed together, the result is the XOR of the original plaintexts. If the plaintexts are text strings, credit card numbers, or other byte streams with known properties, then cryptanalysis may be successful.

Table 6.2. Speed Comparisons of Symmetric Ciphers on a Pentium II

Cipher	Key Length	Speed (Mbps)
DES	56	9
3DES	168	3
RC2	variable	0.9
RC4	variable	45

For applications that require encryption/decryption of a stream of data, such as over a data communications channel or a browser/Web link, a stream cipher might be the better alternative. For applications that deal with blocks of data, such as file transfer, e-mail, and database, block ciphers may be more appropriate. However, either type of cipher can be used in virtually any application.

The RC4 Algorithm

RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. Analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10^{100} [ROBS95a]. Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software. RC4 is used in the SSL/TLS (Secure Sockets Layer/Transport Layer Security) standards that have been defined for communication between Web browsers and servers. It is also used in the WEP (Wired Equivalent Privacy) protocol and the newer WiFi Protected Access (WPA) protocol that are part of the IEEE 802.11 wireless LAN standard. RC4 was kept as a trade secret by RSA Security. In September 1994, the RC4 algorithm was anonymously posted on the Internet on the Cypherpunks anonymous remailers list.

The RC4 algorithm is remarkably simply and quite easy to explain. A variable-length key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state vector S , with elements $S[0], S[1], \dots, S[255]$. At all times, S contains a permutation of all 8-bit numbers from 0 through 255. For encryption and decryption, a byte k (see Figure 6.8) is generated from S by selecting one of the 255 entries in a systematic fashion. As each value of k is generated, the entries in S are once again permuted.

Initialization of S

To begin, the entries of S are set equal to the values from 0 through 255 in ascending order; that is; $S[0] = 0, S[1] = 1, \dots, S[255] = 255$. A temporary vector, T , is also created. If the length of the key K is 256 bytes, then K is transferred to T . Otherwise, for a key of length $keylen$ bytes, the first $keylen$

elements of T are copied from K and then K is repeated as many times as necessary to fill out T. These preliminary operations can be summarized as follows:

[Page 192]

```
/* Initialization */
for i = 0 to 255 do
S[i] = i;
T[i] = K[i mod keylen];
```

Next we use T to produce the initial permutation of S. This involves starting with S[0] and going through to S[255], and, for each S[i], swapping S[i] with another byte in S according to a scheme dictated by T[i]:

```
/* Initial Permutation of S */
j = 0;
for i = 0 to 255 do
  j = (j + S[i] + T[i]) mod 256;
  Swap (S[i], S[j]);
```

Because the only operation on S is a swap, the only effect is a permutation. S still contains all the numbers from 0 through 255.

Stream Generation

Once the S vector is initialized, the input key is no longer used. Stream generation involves cycling through all the elements of S[i], and, for each S[i], swapping S[i] with another byte in S according to a scheme dictated by the current configuration of S. After S[255] is reached, the process continues, starting over again at S[0]:

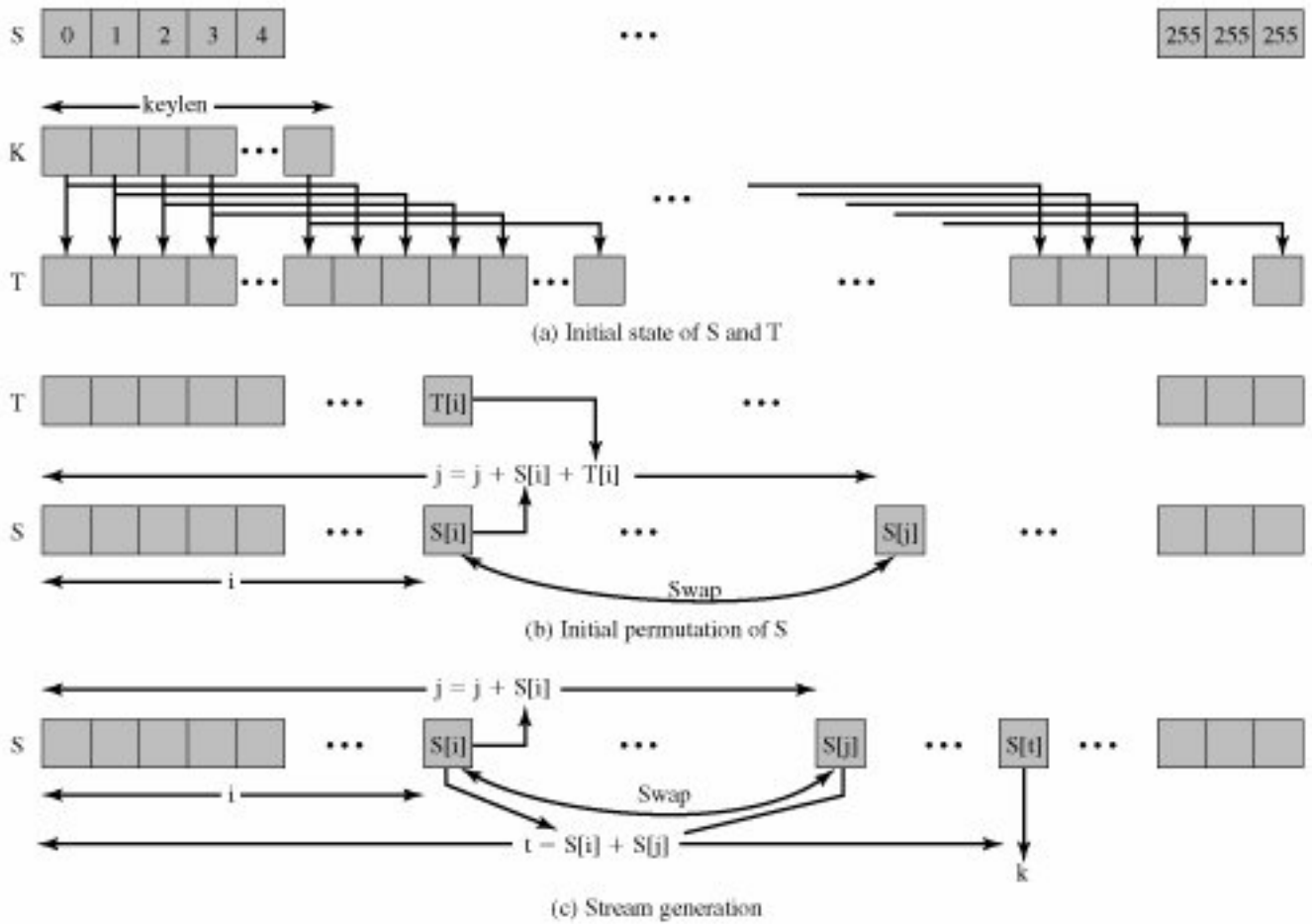
```
/* Stream Generation */
i, j = 0;
while (true)
  i = (i + 1) mod 256;
  j = (j + S[i]) mod 256;
  Swap (S[i], S[j]);
  t = (S[i] + S[j]) mod 256;
  k = S[t];
```

To encrypt, XOR the value *k* with the next byte of plaintext. To decrypt, XOR the value *k* with the next byte of ciphertext.

[Figure 6.9](#) illustrates the RC4 logic.

Figure 6.9. RC4

(This item is displayed on page 193 in the print version)



Strength of RC4

A number of papers have been published analyzing methods of attacking RC4 [e.g., [KNUD98], [MIST98], [FLUH00], [MANTO1]]. None of these approaches is practical against RC4 with a reasonable key length, such as 128 bits. A more serious problem is reported in [FLUH01]. The authors demonstrate that the WEP protocol, intended to provide confidentiality on 802.11 wireless LAN networks, is vulnerable to a particular attack approach. In essence, the problem is not with RC4 itself but the way in which keys are generated for use as input to RC4. This particular problem does not appear to be relevant to other applications using RC4 and can be remedied in WEP by changing the way in which keys are generated. This problem points out the difficulty in designing a secure system that involves both cryptographic functions and protocols that make use of them.

6.4. Recommended Reading and Web Site

[[SCHN96](#)] provides details on numerous symmetric block ciphers as well as some stream ciphers. [[ROBS95b](#)] is an interesting and worthwhile examination of many design issues related to symmetric block ciphers.

[[KUMA97](#)] contains an excellent and lengthy discussion of stream cipher design principles. Another good treatment, quite mathematical, is [[RUEP92](#)]. [[ROBS95a](#)] is an interesting and worthwhile examination of many design issues related to stream ciphers.

[KUMA97](#) Kumar, I. *Cryptology*. Laguna Hills, CA: Aegean Park Press, 1997.

[ROBS95a](#) Robshaw, M. *Stream Ciphers*. RSA Laboratories Technical Report TR-701, July 1995. <http://www.rsasecurity.com/rsalabs>

[ROBS95b](#) Robshaw, M. *Block Ciphers*. RSA Laboratories Technical Report TR-601, August 1995. <http://www.rsasecurity.com/rsalabs>

[RUEP92](#) Rueppel, T. "Stream Ciphers." In [[SIMM92](#)].

[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.

Recommended Web Site



- **Block cipher modes of operation:** NIST page with full information on NIST-approved modes of operation

6.5. Key Terms, Review Questions, and Problems

Key Terms

[Block cipher modes of operation](#)

[cipher block chaining mode \(CBC\)](#)

[cipher feedback mode \(CFB\)](#)

[meet-in-the-middle attack](#)

[counter mode \(CTR\)](#)

[electronic codebook mode \(ECB\)](#)

[output feedback mode \(OFB\)](#)

[RC4](#)

[stream cipher](#)

[Triple DES \(3DES\)](#)

Review Questions

- 6.1 What is triple encryption?
- 6.2 What is a meet-in-the-middle attack?
- 6.3 How many keys are used in triple encryption?
- 6.4 Why is the middle portion of 3DES a decryption rather than an encryption?
- 6.5 List important design considerations for a stream cipher.
- 6.6 Why is it not desirable to reuse a stream cipher key?

6.7 What primitive operations are used in RC4?

6.8 Why do some block cipher modes of operation only use encryption while others use both encryption and decryption?

Problems

6.1 You want to build a hardware device to do block encryption in the cipher block chaining (CBC) mode using an algorithm stronger than DES. 3DES is a good candidate. Figure 6.10 shows two possibilities, both of which follow from the definition of CBC. Which of the two would you choose:

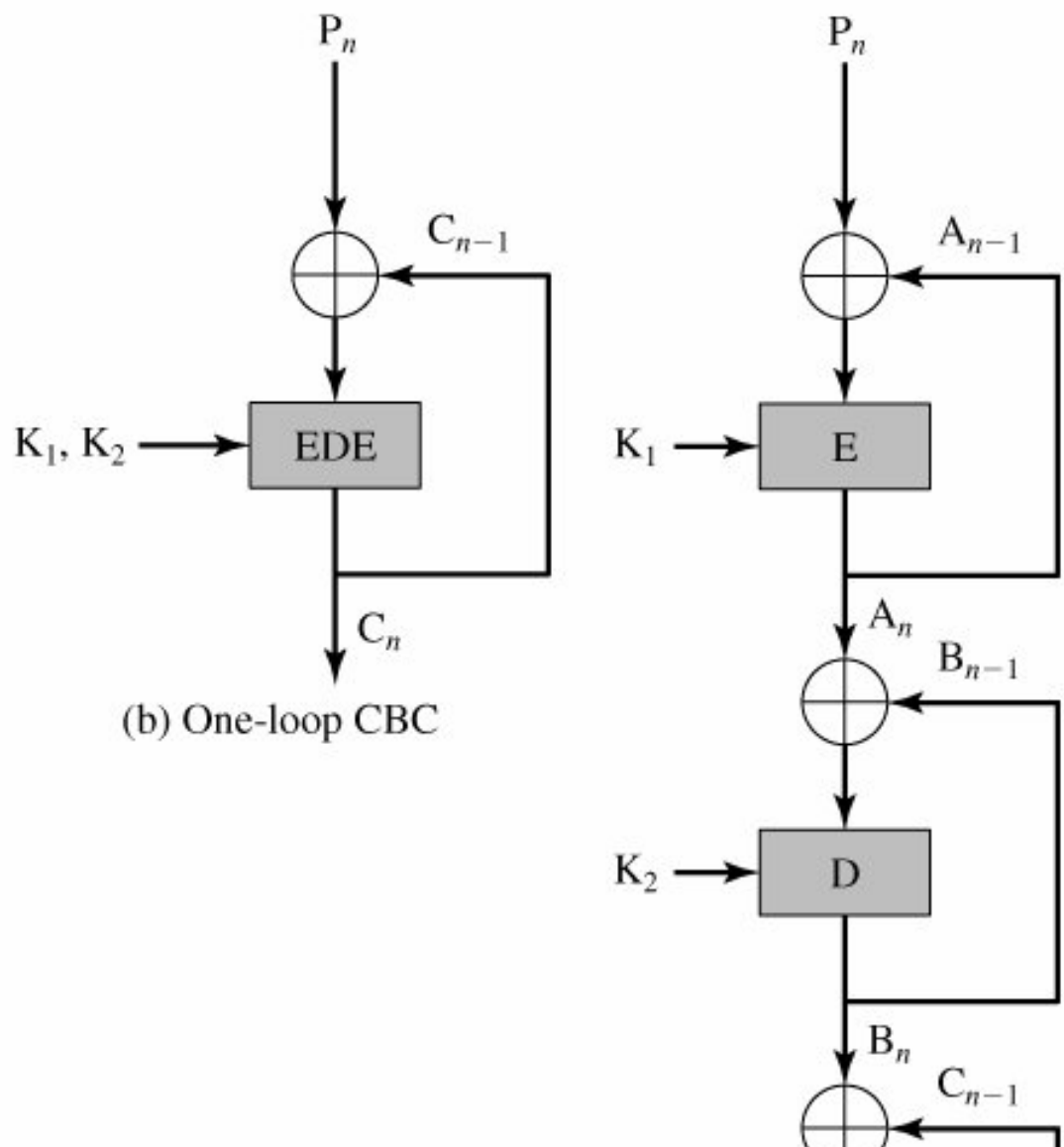
a.

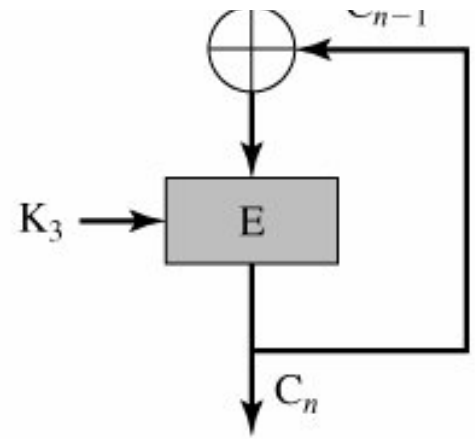
For security?

b.

For performance?

Figure 6.10. Use of Triple DES in CBC Mode





(b) Three-loop CBC

- 6.2 Can you suggest a security improvement to either option in [Figure 6.10](#), using only three DES chips and some number of XOR functions? Assume you are still limited to two keys.
- 6.3 The Merkle-Hellman attack on 3DES begins by assuming a value $A = 0$ of ([Figure 6.1b](#)). 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.
- 6.4 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 6.4](#)) obviously corrupts P_1 and P_2 .
- a.
- Are any blocks beyond P_2 affected?
- b.
- 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?
- 6.5 If a bit error occurs in the transmission of a ciphertext character in 8-bit CFB mode, how far does the error propagate?

6.6 Fill in the remainder of this table:

Mode	Encrypt	Decrypt
ECB	$C_j = E(K, P_j) \quad j = 1, \dots, N$	$P_j = D(K, C_j) \quad j = 1, \dots, N$
CBC	$C_1 = E(K, [P_1 \oplus IV])$ $C_j = E(K, [P_j \oplus C_{j-1}]) \quad j = 2, \dots, N$	$P_1 = D(K, C_1) \oplus IV$ $P_j = D(K, C_j) \oplus C_{j-1} \quad j = 2, \dots, N$
CFB		
OFB		
CTR		

6.7 CBC-Pad is a block cipher mode of operation used in the RC5 block cipher, but it could be used in any block cipher. CBC-Pad handles plaintext of any length. The ciphertext is longer than the plaintext by at most the size of a single block. Padding is used to assure that the plaintext input is a multiple of the block length. It is assumed that the original plaintext is an integer number of bytes. This plaintext is padded at the end by from 1 to bb bytes, where bb equals the block size in bytes. The pad bytes are all the same and 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. Why not allow zero bytes of padding? That is, if the original plaintext is an integer multiple of the block size, why not refrain from padding?

6.8 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. A mode for that purpose is the ciphertext stealing (CTS) mode. [Figure 6.11a](#) shows an implementation of this mode.

a.

Explain how it works.

b.

Describe how to decrypt C_{n-1} and C_n

6.9 [Figure 6.11b](#) shows an alternative to CTS for producing ciphertext of equal length to the plaintext when the plaintext is not an integer multiple of the block size.

a.

Explain the algorithm.

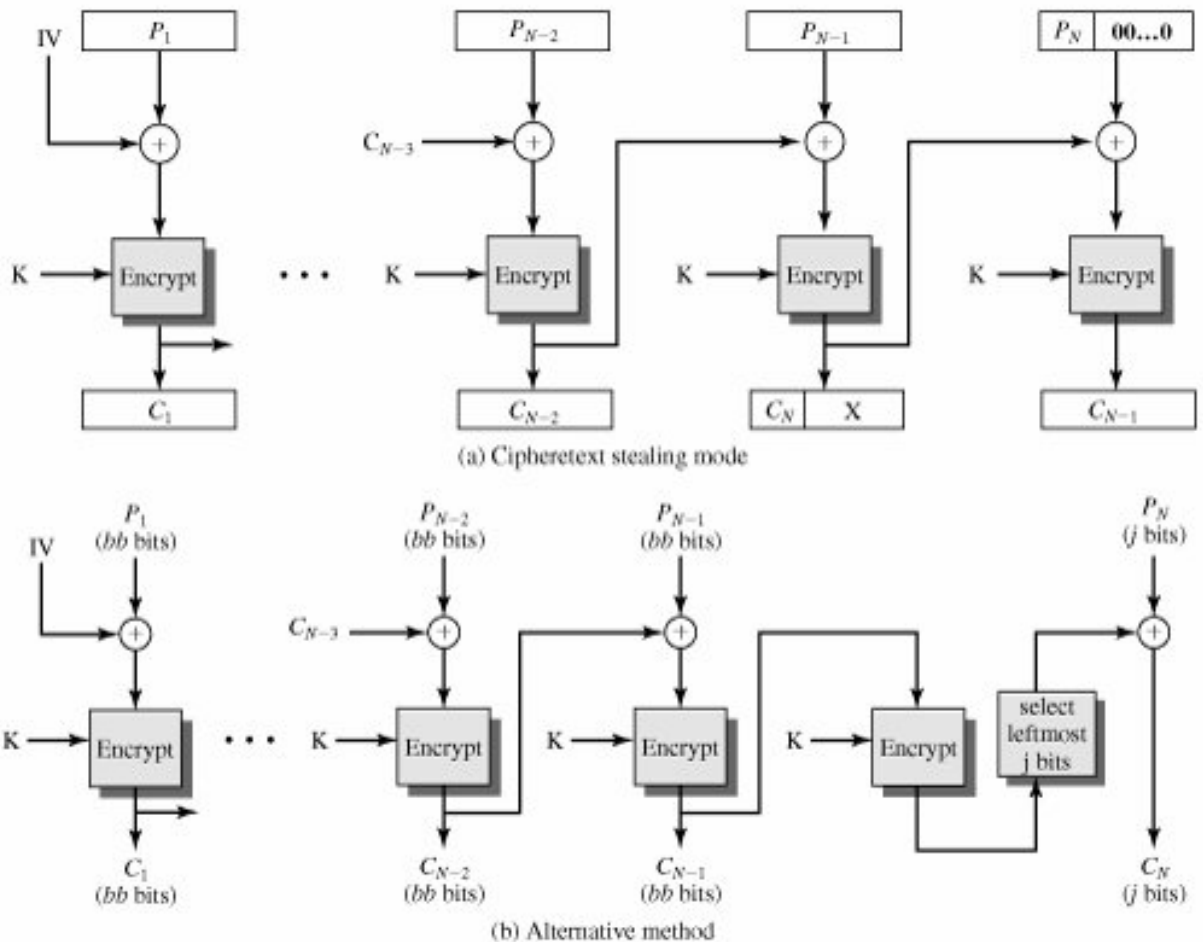
b.

Explain why CTS is preferable to this approach illustrated in [Figure 6.11b](#).

Figure 6.11. Block Cipher Modes for Plaintext not a Multiple of Block Size

(This item is displayed on page 197 in the print version)

[\[View full size image\]](#)



6.10 What RC4 key value will leave S unchanged during initialization? That is, after the initial permutation of S, the entries of S will be equal to the values from 0 through 255 in ascending order.

6.11 RC4 has a secret internal state which is a permutation of all the possible values of the vector \mathbf{S} and the two indices i and j .

a.

Using a straightforward scheme to store the internal state, how many bits are used?

b.

Suppose we think of it from the point of view of how much information is represented by the state. In that case, we need to determine how many different states there are, then take the log to the base 2 to find out how many bits of information this represents. Using this approach, how many bits would be needed to represent the state?

6.12 Alice and Bob agree to communicate privately via email using a scheme based on RC4, but want to avoid using a new secret key for each transmission. Alice and Bob privately agree on a 128-bit key k . To encrypt a message m , consisting of a string of bits, the following procedure is used:

1.

Choose a random 80-bit value v

2.

Generate the ciphertext $c = \text{RC4}(v \parallel k) \oplus m$

3.

Send the bit string $(v \parallel C)$

a.

Suppose Alice uses this procedure to send a message m to Bob. Describe how Bob can recover the message m from $(v \parallel C)$ using k .

b.

If an adversary observes several values $(v_1 \parallel C_1)$, $(v_2 \parallel C_2)$, ... transmitted between Alice and Bob, how can he/she determine when the same key stream has been used to encrypt two messages?

[Page 198]

c.

Approximately how many messages can Alice expect to send before the same key stream will be used twice? Use the result from the birthday paradox described in [Appendix 11A \[Equation \(11.7\)\]](#).

d.

What does this imply about the lifetime of the key k (i.e., the number of messages that can be encrypted using k)?

Programming Problems

6.13 Create software that can encrypt and decrypt in Cipher Block Chaining mode using one of the following ciphers: affine modulo 256, Hill modulo 256, S-DES, DES. Test data for S-DES: using a binary initialization vector of 1010 1010, a binary plaintext of 0000 0001 0010 0011 encrypted with a binary key of 01111 11101 should give a binary plaintext of 1111 0100 0000 1011. Decryption should work correspondingly.

6.14 Create software that can encrypt and decrypt in 4-bit Cipher Feedback mode using one of the following ciphers: additive modulo 256, affine modulo 256, S-DES;

or

8-bit Cipher Feedback mode using one of the following ciphers: 2 x 2 Hill modulo 256. Test data for S-DES: using a binary initialization vector of 1010 1011, a binary plaintext of 0001 0010 0011 0100 encrypted with a binary key of 01111 11101 should give a binary plaintext of 1110 1100 1111 1010. Decryption should work correspondingly.

6.15 Create software that can encrypt and decrypt in 4-bit Output Feedback mode using one of the following ciphers: additive modulo 256, affine modulo 256, S-DES;

or

8-bit Output Feedback mode using one of the following ciphers: 2 x 2 Hill modulo 256,

6.16 Create software that can encrypt and decrypt in Counter mode using one of the following ciphers: affine modulo 256, Hill modulo 256, S-DES.

Test data for S-DES: using a counter starting at 0000 0000, a binary plaintext of 0000 0001 0000 0010 0000 0100 encrypted with a binary key of 01111 11101 should give a binary plaintext of 0011 1000 0100 1111 0011 0010. Decryption should work correspondingly.

6.17 Implement a differential cryptanalysis attack on 3-round S-DES.

Chapter 7. Confidentiality Using Symmetric Encryption

7.1 Placement of Encryption Function

[Potential Locations for Confidentiality Attacks](#)

[Link versus End-to-End Encryption](#)

7.2 Traffic Confidentiality

[Link Encryption Approach](#)

[End-to-End Encryption Approach](#)

7.3 Key Distribution

[A Key Distribution Scenario](#)

[Hierarchical Key Control](#)

[Session Key Lifetime](#)

[A Transparent Key Control Scheme](#)

[Decentralized Key Control](#)

[Controlling Key Usage](#)

7.4 Random Number Generation

[The Use of Random Numbers](#)

[Pseudorandom Number Generators \(PRNGs\)](#)

[Linear Congruential Generators](#)

[Cryptographically Generated Random Numbers](#)

[Blum Blum Shub Generator](#)

[True Random Number Generators](#)

[Skew](#)

[7.5 Recommended Reading and Web Sites](#)

[7.6 Key Terms, Review Questions, and Problems](#)

[Key Terms](#)

[Review Questions](#)

[Problems](#)

[Page 200]

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

Key Points

- In a distributed environment, encryption devices can be placed to support either link encryption or end-to-end encryption. With link encryption, each vulnerable communications link is equipped on both ends with an encryption device. With end-to-end encryption, the encryption process is carried out at the two end systems.
- Even if all traffic between users is encrypted, a traffic analysis may yield information of value to an opponent. An effective countermeasure is traffic padding, which involves sending random bits during periods when no encrypted data are available for transmission.
- Key distribution is the function that delivers a key to two parties who wish to exchange secure encrypted data. Some sort of mechanism or protocol is needed to provide for the secure distribution of keys.
- Key distribution often involves the use of master keys, which are infrequently used and are long lasting, and session keys, which are generated and distributed for temporary use between two parties.
- A capability with application to a number of cryptographic functions is random or pseudorandom number generation. The principle requirement for this capability is that the generated number stream be unpredictable.

[Page 201]

Historically, the focus of cryptology has been on the use of symmetric 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 symmetric 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 encryption 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.

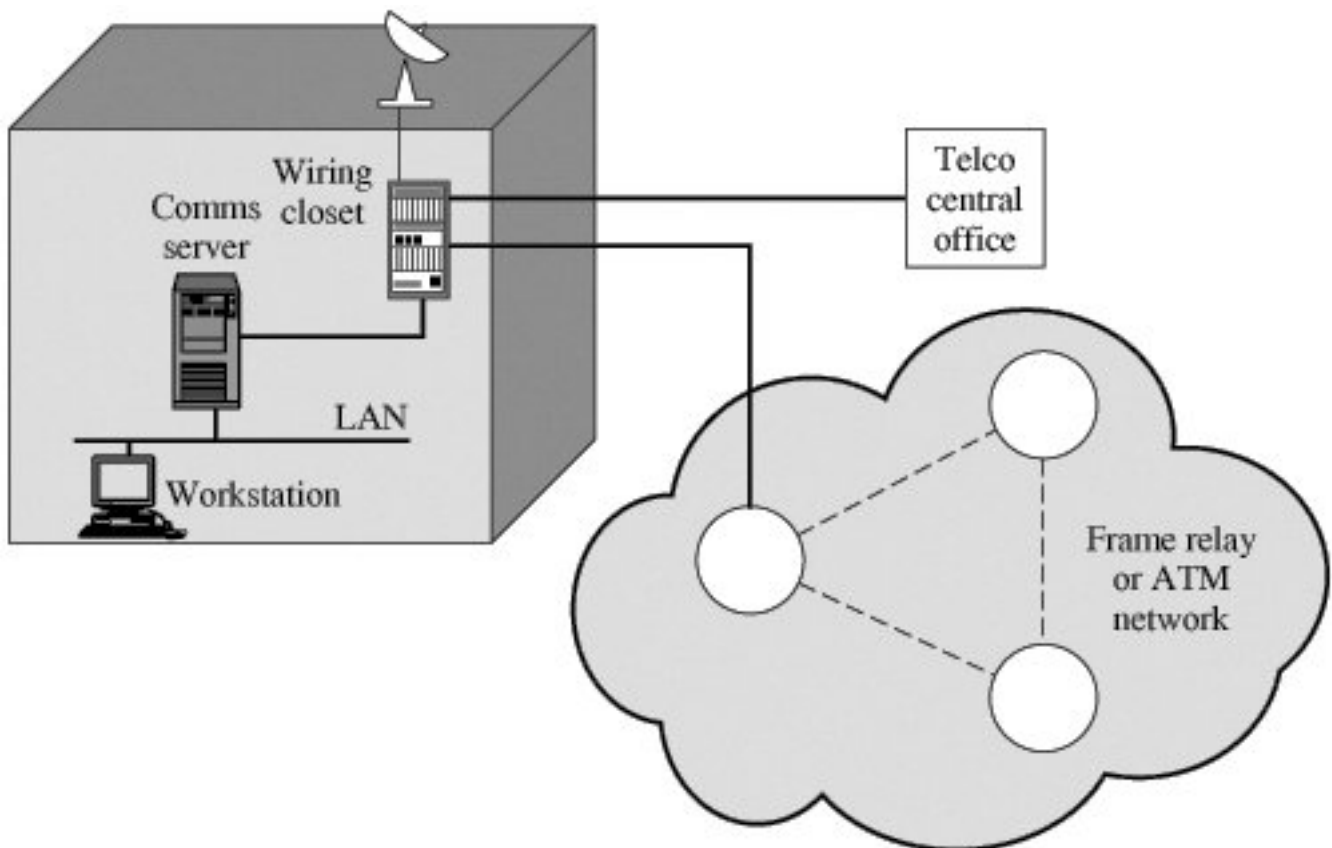
7.1. Placement of Encryption Function

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 7.1](#) suggests the types of communications facilities that might be employed by such a workstation and therefore gives an indication of the points of vulnerability.

Figure 7.1. 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. If part or all of the LAN is wireless, then the potential for eavesdropping is greater.

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 that connects to the Internet, 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. 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.

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. 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 on 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 emanations. Invasive taps allow both active and passive attacks, whereas inductive taps are useful for passive attacks. Neither type of tap is as effective 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.

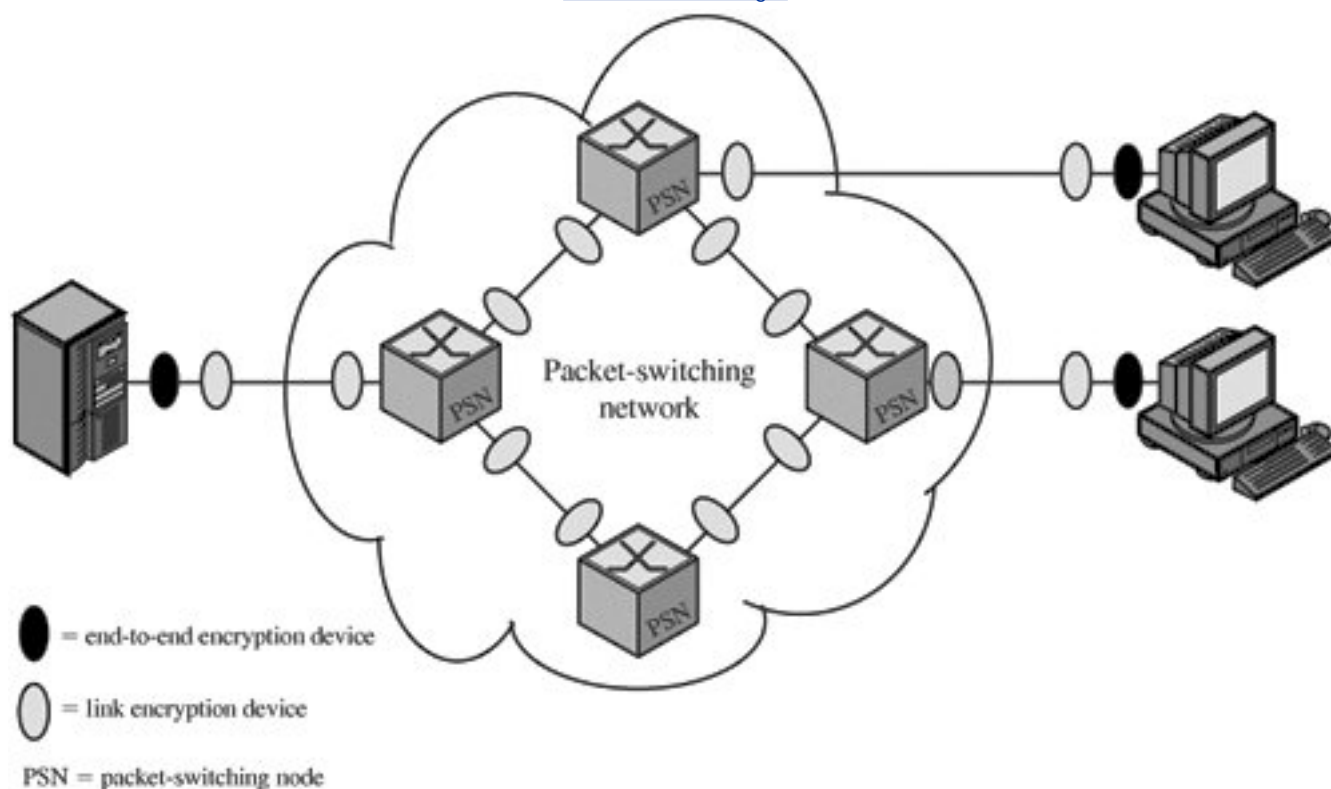
Link versus End-to-End Encryption

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 7.2](#) indicates, there are two fundamental alternatives: link encryption and end-to-end encryption.

Figure 7.2. Encryption Across a Packet-Switching Network

(This item is displayed on page 204 in the print version)

[\[View full size image\]](#)



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 switch (such as a frame relay switch) because the switch must read the address (logical connection number) in the packet header in order to route the frame. Thus, the message is vulnerable at each switch. If working with a public 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.

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 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 a frame relay or ATM network, sets up a logical connection 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 frame relay or ATM switch 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 7.2](#). 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 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 7.1](#) summarizes the key characteristics of the two encryption strategies.

Table 7.1. Characteristics of Link and End-to-End Encryption [[PFLE02](#)]

Link Encryption	End-to-End Encryption
Security within End Systems and Intermediate Systems	
Message exposed in sending host	Message encrypted in sending host
Message exposed in intermediate nodes	Message encrypted in intermediate nodes
Role of User	
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	Users selects encryption scheme
Can be done in hardware	Software implementation
All or no messages encrypted	User chooses to encrypt, or not, for each message
Implementation Concerns	
Requires one key per (host-intermediate node) pair and (intermediate node-intermediate node) pair	Requires one key per user pair
Provides host authentication	Provides user authentication

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.

[Page 206]

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 the frame relay or ATM protocol, so that the user data portion of all frames or ATM cells is encrypted.

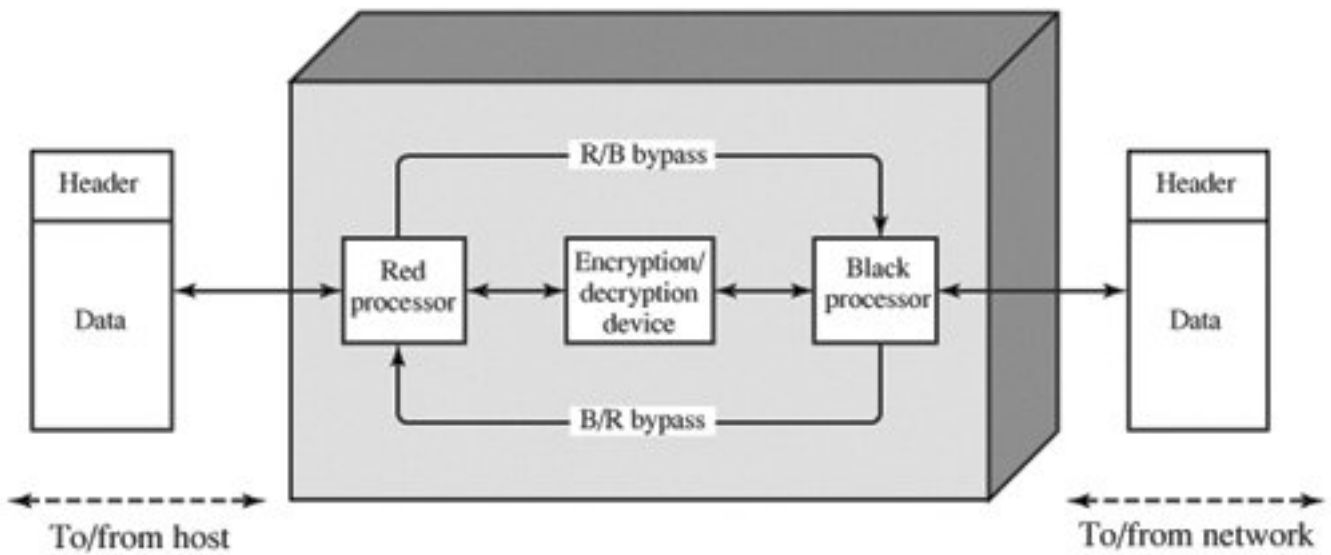
With network-layer encryption, the number of identifiable and separately protected entities corresponds to the number of end systems in the network. Each end system can engage in an encrypted exchange with another end system if the two share a secret key. All the user processes and applications within 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 7.3](#) 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.^[1] 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 delivered to the host. If the transport layer functionality (e.g., 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.

^[1] The terms *red* and *black* are frequently used. Red data are sensitive or classified data in the clear. Black data are encrypted data.

Figure 7.3. Front-End Processor Function

[\[View full size image\]](#)



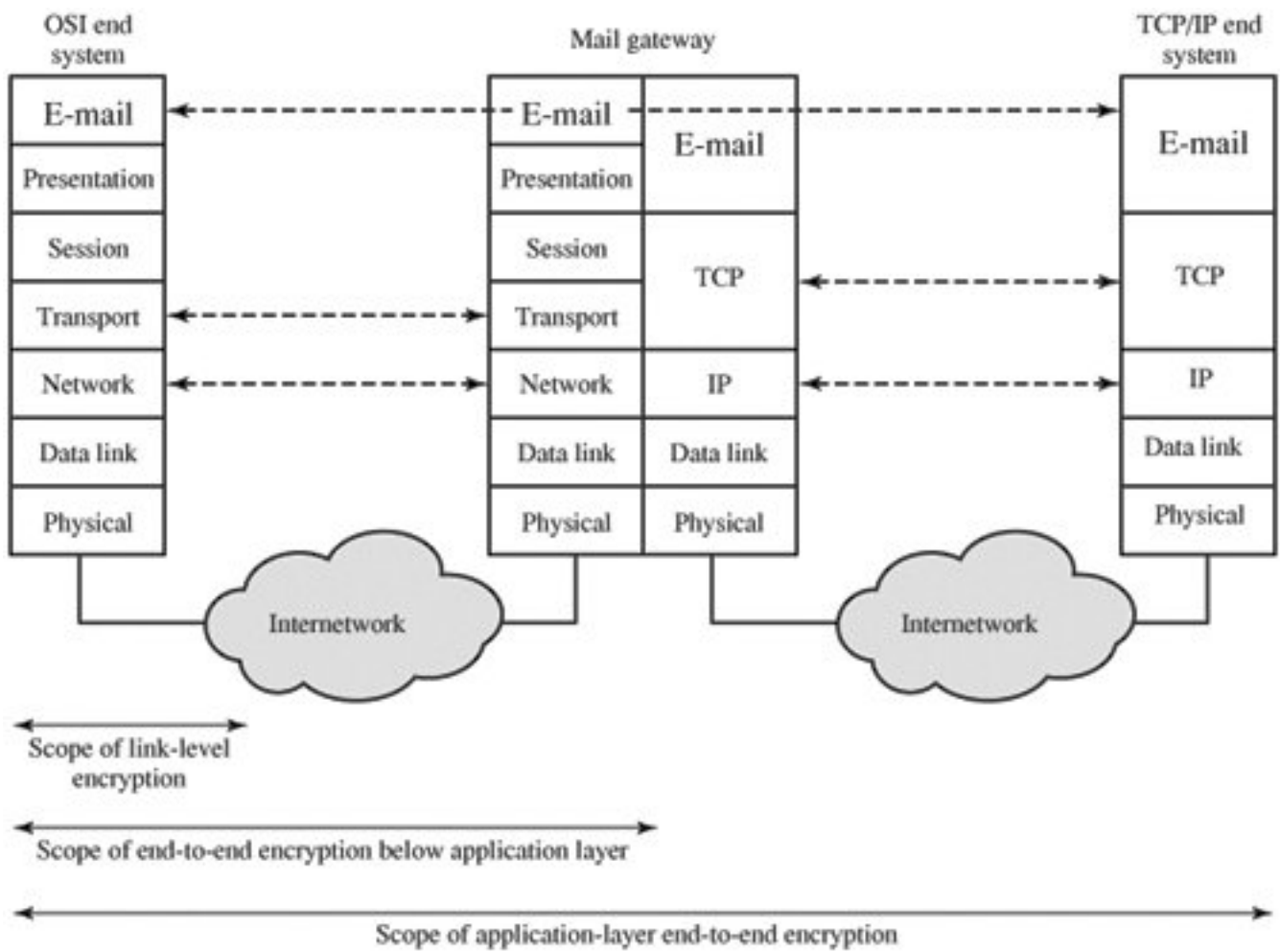
Deployment of encryption services on end-to-end protocols, such as a network-layer frame relay or TCP, provides end-to-end security for traffic within a fully integrated internetwork. However, such a scheme cannot deliver the necessary service for traffic that crosses internetwork boundaries, such as electronic mail, electronic data interchange (EDI), and file transfers.

[Figure 7.4](#) 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. ^[2] 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.

^[2] Appendix H provides a brief overview of the OSI and TCP/IP protocol architectures.

Figure 7.4. Encryption Coverage Implications of Store-and-Forward Communications

[\[View full size image\]](#)



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 7.5](#) 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. ^[3]

^[3] Unfortunately, most TCP/IP documents use the term *gateway* to refer to what is more commonly referred to as a *router*.

Figure 7.5. Relationship between Encryption and Protocol Levels

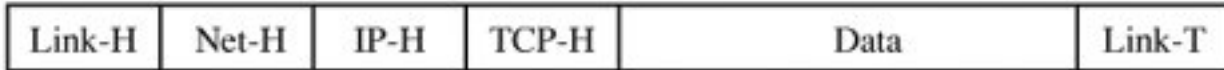
[\[View full size image\]](#)



(a) Application-level encryption (on links and at routers and gateways)



On links and at routers

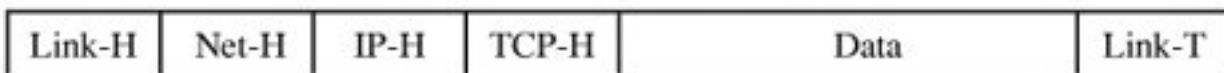


In gateways

(b) TCP-level encryption



On links



In routers and gateways

(c) Link-level encryption

Shading indicates 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

With application-level encryption ([Figure 7.5a](#)), 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 7.5b](#)), 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 7.5c](#)), 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. ^[4]

^[4] 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.

7.2. 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:

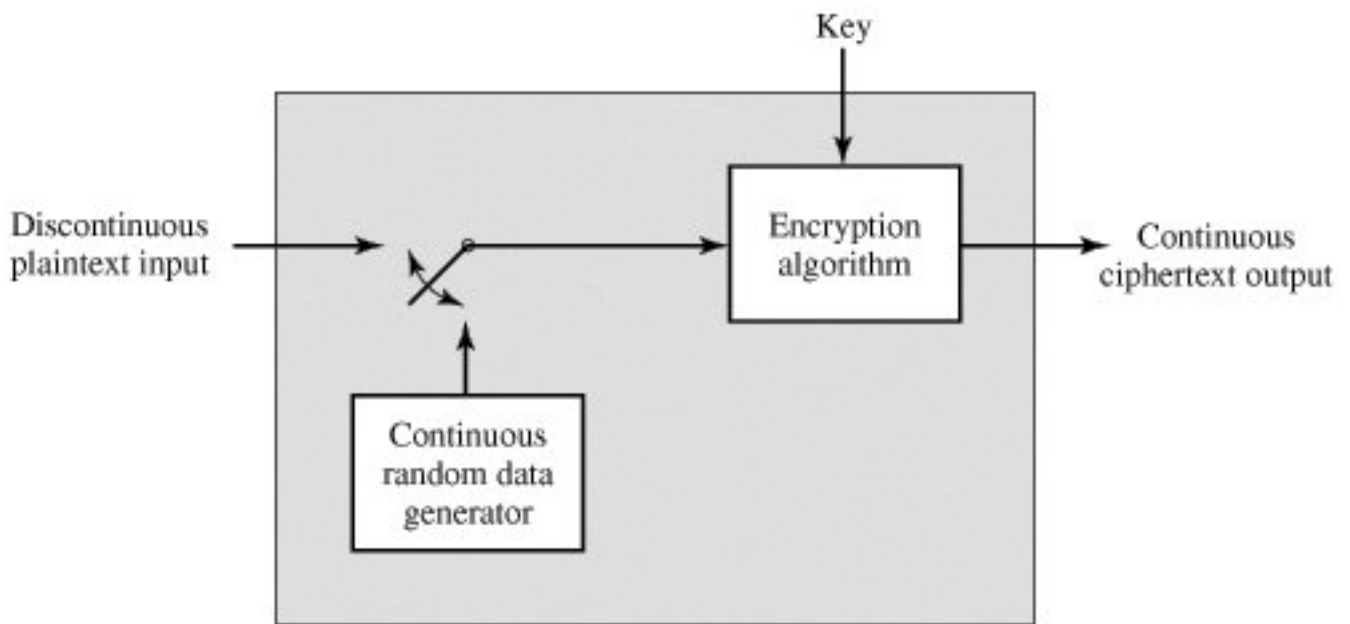
- Identities of partners
- How frequently the partners are communicating
- Message pattern, message length, or quantity of messages that suggest important information is being exchanged
- The events that correlate with special conversations between particular partners

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, network-layer headers (e.g., frame or cell header) 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 7.6](#).

Figure 7.6. Traffic-Padding Encryption Device



[Page 210]

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.

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.

7.3. Key Distribution

For symmetric 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 can select a key and physically deliver it 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.

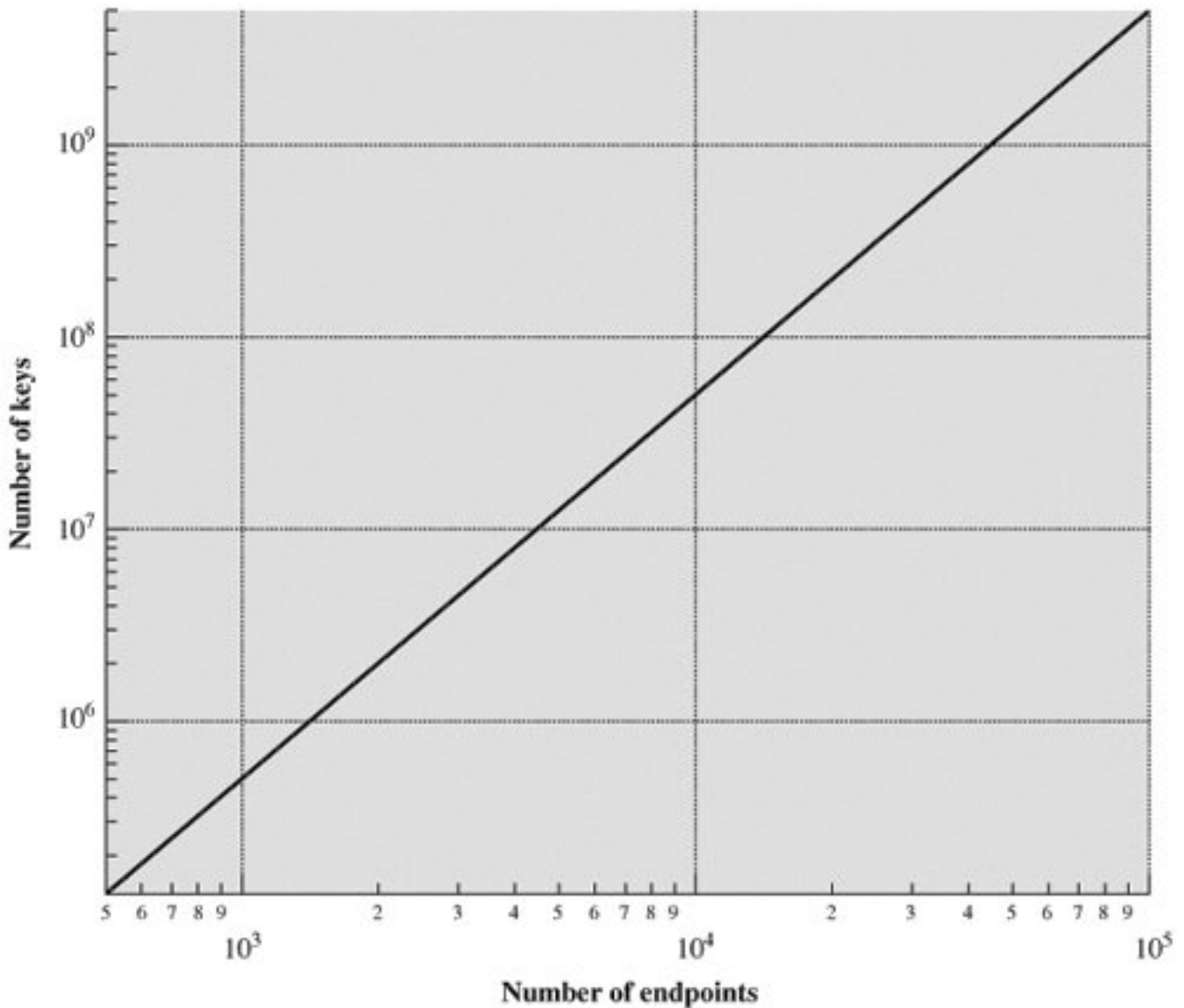
Options 1 and 2 call for manual delivery of a key. For link encryption, this is a reasonable requirement, because each link encryption device is going to be exchanging data only with its partner on the other end of the link. However, for end-to-end encryption, manual delivery is awkward. In a distributed 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 7.7](#) illustrates the magnitude of the key distribution task for end-to-end encryption. ^[5] 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.

^[5] Note that this figure uses a log-log scale, so that a linear graph indicates exponential growth. A basic review of log scales is in the math refresher document at the Computer Science Student Resource Site at WilliamStallings.com/StudentSupport.html.

Figure 7.7. Number of Keys Required to Support Arbitrary Connections between Endpoints

[\[View full size image\]](#)



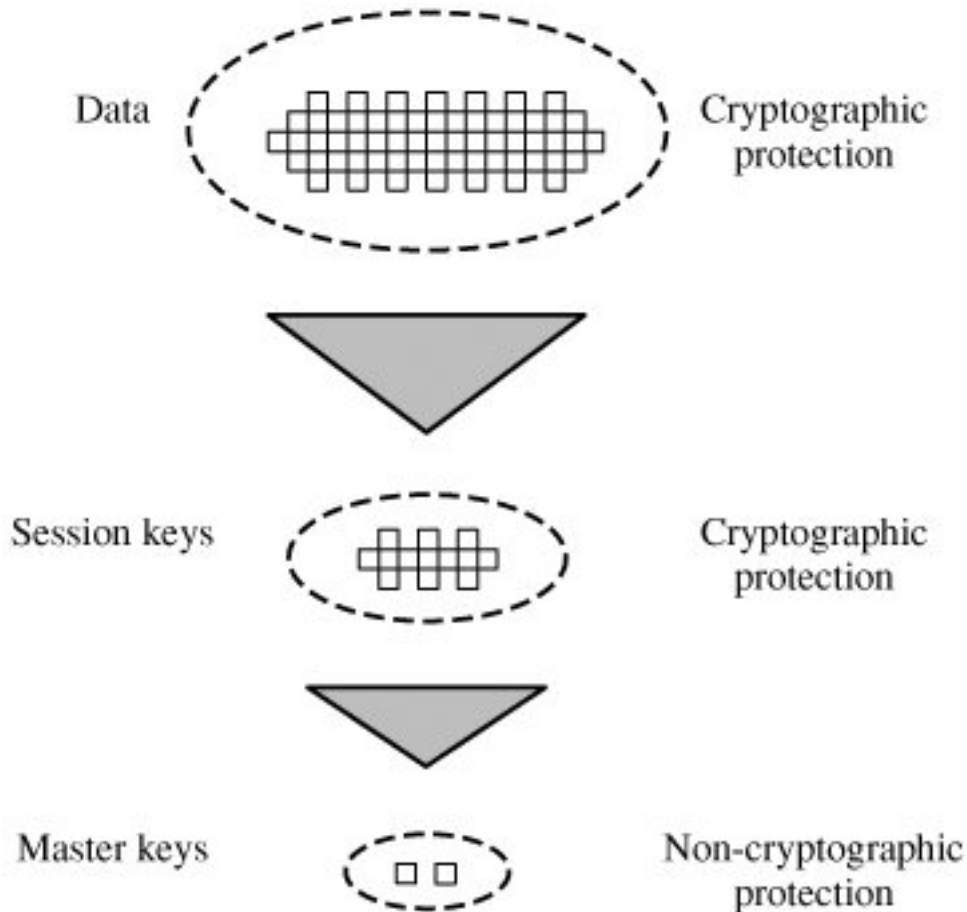
Returning to our list, option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys will be revealed. Furthermore, the initial distribution of potentially millions of keys must still be made.

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 7.8](#)). 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 frame relay connection 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.

Figure 7.8. The Use of a Key Hierarchy



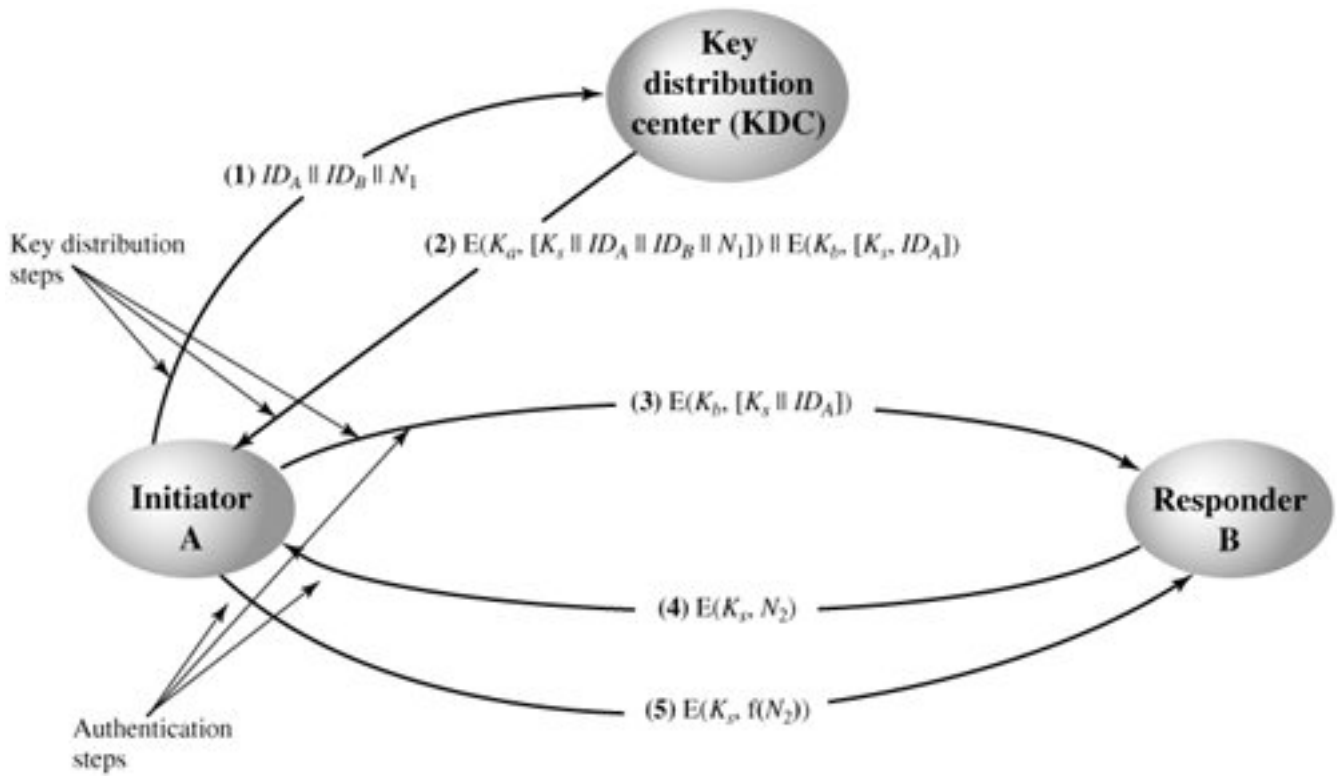
For each end system or user, there is a unique master key that it shares with the key distribution center. Of course, these master keys must be distributed in some fashion. However, the scale of the problem is vastly reduced. If there are N entities that wish to communicate in pairs, then, as was mentioned, 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.

A Key Distribution Scenario

The key distribution concept can be deployed in a number of ways. A typical scenario is illustrated in [Figure 7.9](#), 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).

Figure 7.9. Key Distribution Scenario

(This item is displayed on page 213 in the print version)



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 master 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**.^[6] The nonce may be a timestamp, a counter, or a random number; the minimum requirement is that it differs 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.

[6] 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.

2. The KDC responds with a message encrypted using K_a . Thus, A is the only one who can successfully read the message, and A knows that it originated at the KDC. The message includes two items intended for A:
 - 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.

[Page 214]

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, $E(K_b, [K_s || ID_A])$. 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 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.

[Page 215]

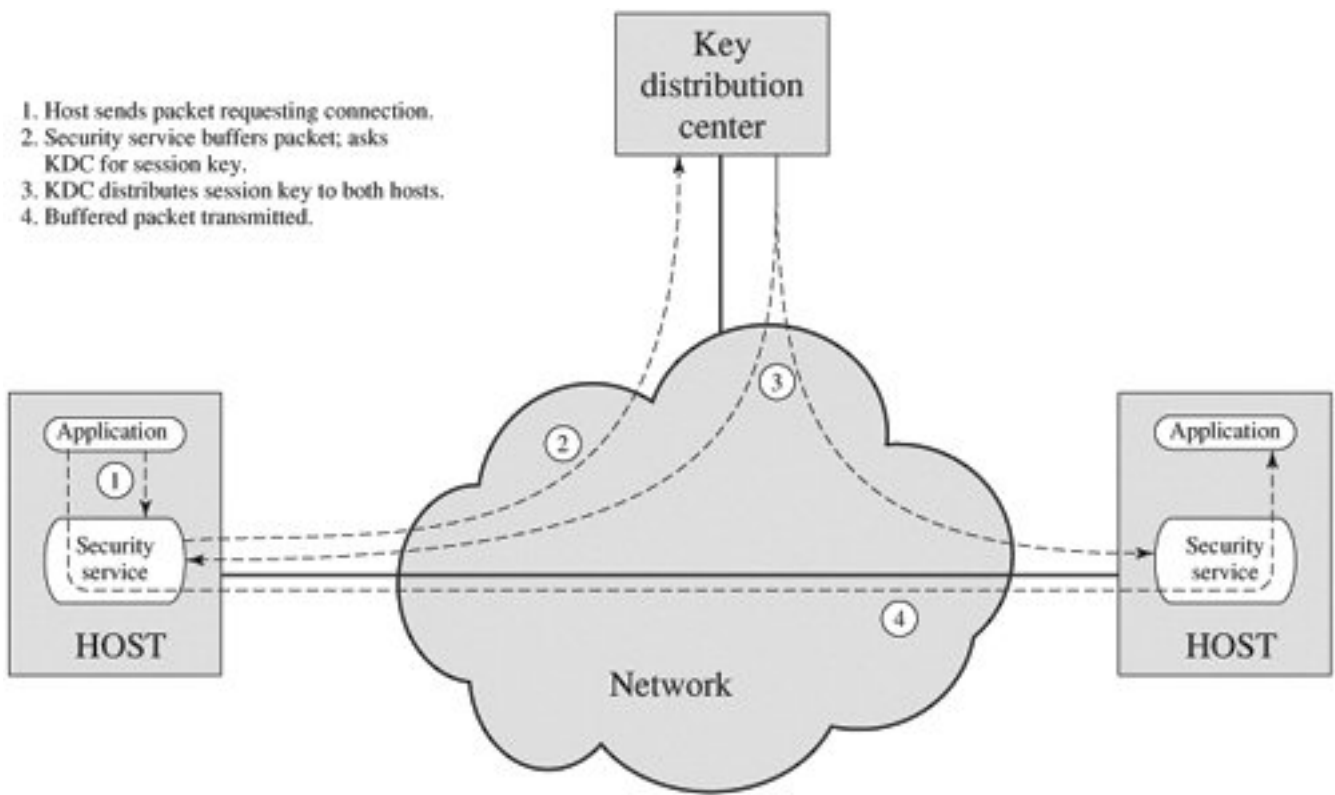
A Transparent Key Control Scheme

The approach suggested in [Figure 7.9](#) has many variations, one of which is described in this subsection. The scheme ([Figure 7.10](#)) 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 TCP. The noteworthy element of this approach is a session security module (SSM), which may consist of functionality at one protocol layer, that performs end-to-end encryption and obtains session keys on behalf of its host or terminal.

Figure 7.10. Automatic Key Distribution for Connection-Oriented Protocol

(This item is displayed on page 216 in the print version)

[\[View full size image\]](#)



The steps involved in establishing a connection are shown in the figure. When one host wishes to set up a connection to another host, it transmits a connection-request packet (step 1). The SSM saves that packet and applies to the KDC for permission to establish the connection (step 2). The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC. If the KDC approves the connection request, it generates the session key and delivers it to the two appropriate SSMs, using a unique permanent key for each SSM (step 3). The requesting SSM 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 SSMs using the one-time session key.

The automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of terminal users to access a number of hosts and for the hosts to exchange data with each other.

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 symmetric encryption only, it may be useful within a local context.

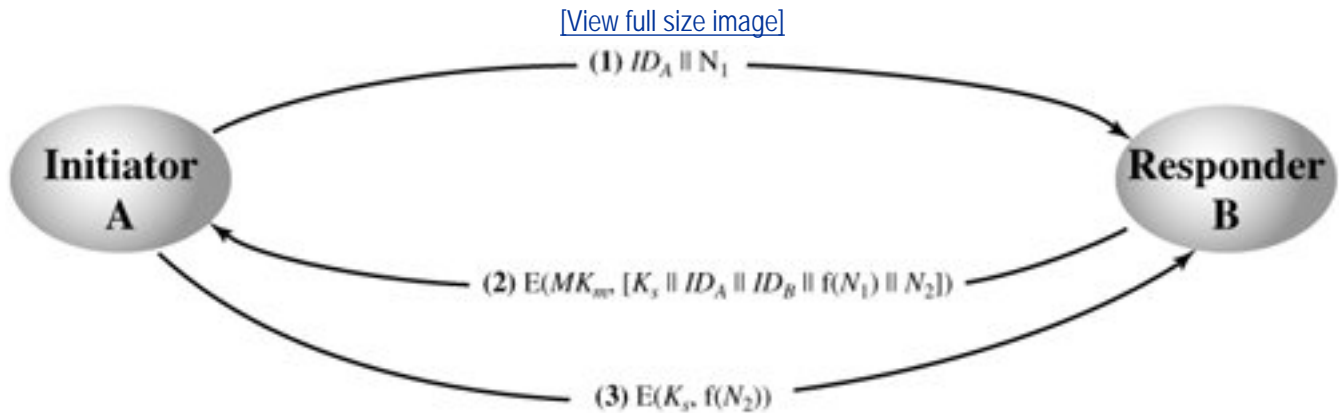
A decentralized approach requires that each end system be able to communicate in a secure manner with all potential partner end systems for purposes of session key distribution. Thus, there may need to be as many as $[n(n-1)]/2$ master keys for a configuration with n end systems.

A session key may be established with the following sequence of steps ([Figure 7.11](#)):

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.

Figure 7.11. Decentralized Key Distribution



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 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.

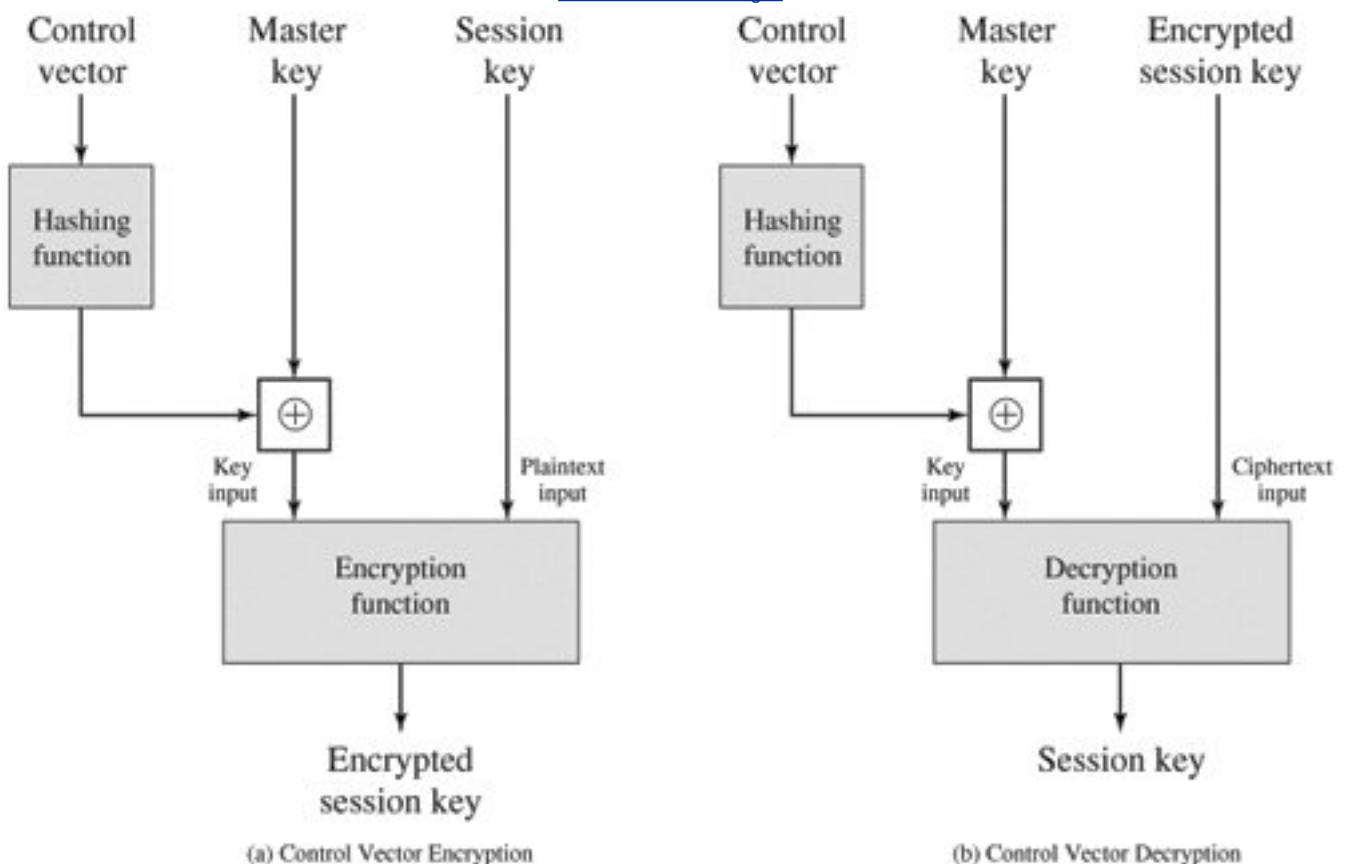
A more flexible scheme, referred to as the control vector, is described in [MATY91a and b]. In this scheme, each session key has an associated control vector consisting of a number of fields that specify the uses and restrictions for that session key. The length of the control vector may vary.

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 7.12. 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 11. 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.

Figure 7.12. Control Vector Encryption and Decryption

(This item is displayed on page 219 in the print version)

[\[View full size image\]](#)



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:

$$D([K_m \oplus H], E([K_m \oplus H], K_s))$$

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.

7.4. 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 7.9](#) and [7.11](#). 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 9](#)).

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 if 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 9](#) 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 odd 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 random 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 seldom 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.

[Page 221]

Pseudorandom Number Generators (PRNGs)

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 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.

Linear Congruential 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	$0 < a < m$
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 $\{5, 25, 29, 17, 21, 9, 13, 1, 5, \text{etc.}\}$, which increases the period to 8.

[Page 222]

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 tests to be used in evaluating a random number generator:

T_1 : 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_2 : The generated sequence should appear random. Because it is generated deterministically, the sequence is not random. There is a variety of statistical tests that can be used to assess the degree to which a sequence exhibits randomness.

T_3 : 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_1 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 [[LEWI69](#)]. This generator is widely used and has been subjected to a more thorough testing than any other PRNG. 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 .

[Page 223]

Thus, although it is nice to be able to use a good PRNG, 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.

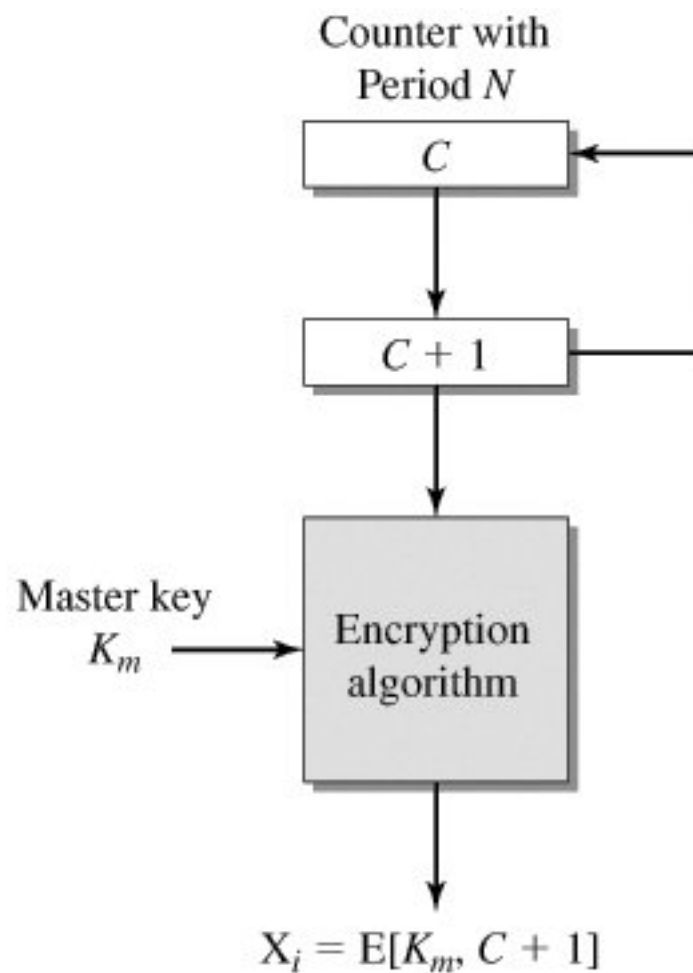
Cyclic Encryption

Figure 7.13 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 session keys (random numbers) through knowledge of one or more earlier session keys.

Figure 7.13. Pseudorandom Number Generation from a Counter



To strengthen the algorithm further, the input could be the output of a full-period PRNG rather than a simple counter.

DES Output Feedback Mode

The output feedback (OFB) mode of DES, illustrated in [Figure 6.6](#), 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 s 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 PRNG

One of the strongest (cryptographically speaking) PRNGs is specified in ANSI X9.17. A number of applications employ this technique, including financial security applications and PGP (the latter described in [Chapter 15](#)).

[Figure 7.14](#) 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 generation. 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:

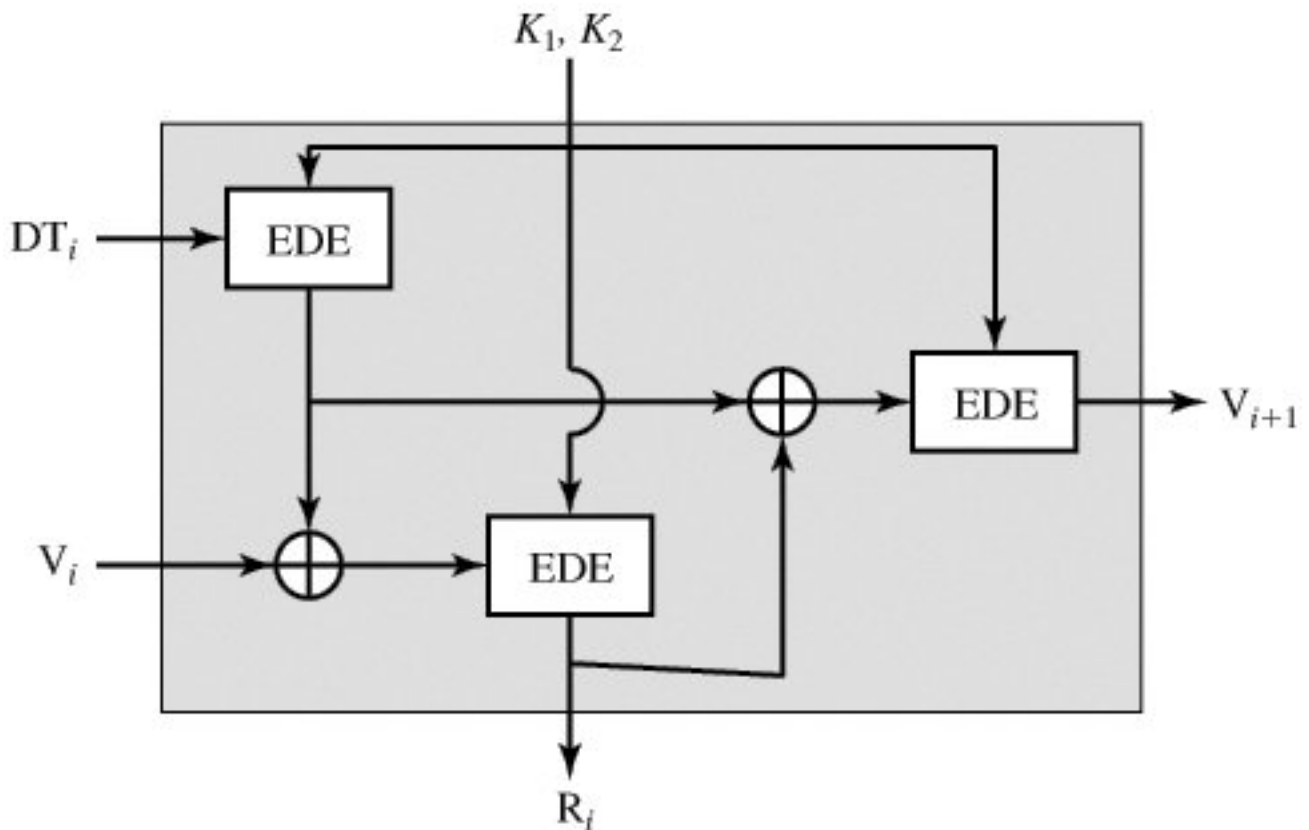
DT_i Date/time value at the beginning of i th generation stage

V_i Seed value at the beginning of i th generation stage

R_i Pseudorandom number produced by the i th generation stage

K_1, K_2 DES keys used for each stage

Figure 7.14. ANSI X9.17 Pseudorandom Number Generator



Then

$$R_i = \text{EDE}([K_1, K_2], [V_i \oplus \text{EDE}([K_1, K_2], DT_i)])$$

$$V_{j+1} = \text{EDE}([K_1, K_2], [R_i \oplus \text{EDE}([K_1, K_2], DT_i)])$$

where $\text{EDE}([K_1, K_2], X)$ refers to the sequence encrypt-decrypt-encrypt using two-key triple DES to encrypt X .

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_{j+1} from the R_i because an additional EDE operation is used to produce the V_{j+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 4](#), 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:

```

X0 = s2 mod n
for i = 1 to ∞
    Xi = (Xi-1)2 mod n
    Bi = Xi mod 2

```

Thus, the least significant bit is taken at each iteration. [Table 7.2](#), shows an example of BBS operation. Here, $n = 192649 = 383 \times 503$ and the seed $s = 101355$.

Table 7.2. Example Operation of BBS Generator

(This item is
displayed on page
226 in the print
version)

i	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
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

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 [7] that, on input of the first k bits of an output sequence, can predict the $(k + 1)^{\text{st}}$ 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 .

[7] A polynomial-time algorithm of order k is one whose running time is bounded by a polynomial of order k .

True Random Number Generators

A true random number generator (TRNG) uses a nondeterministic source to produce randomness. Most operate by measuring unpredictable natural processes, such as pulse detectors of ionizing radiation events, gas discharge tubes, and leaky capacitors. Intel has developed a commercially available chip that samples thermal noise by amplifying the voltage measured across undriven resistors [[JUN99](#)]. A group at Bell Labs has developed a technique that uses the variations in the response time of raw read requests for one disk sector of a hard disk [[JAKO98](#)]. LavaRnd is an open source project for creating truly random numbers using inexpensive cameras, open source code, and inexpensive hardware. The system uses a saturated CCD in a light-tight can as a chaotic source to produce the seed. Software processes the result into truly random numbers in a variety of formats.

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 published collection of good-quality random numbers (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.

Skew

A true random number generator may produce an output that is biased in some way, such as having more ones than zeros or vice versa. Various methods of modifying a bit stream to reduce or eliminate the bias have been developed. These are referred to as *deskewing algorithms*. One approach to deskew is to pass the bit stream through a hash function such as MD5 or SHA-1 (described in [Part Two](#)). The hash function produces an n -bit output from an input of arbitrary length. For deskewing, blocks of m input bits, with $m \geq n$ can be passed through the hash function.

7.5. Recommended Reading and Web Sites

[[FUMY93](#)] is a good survey of key management principles.

Perhaps the best treatment of PRNGs 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 PRNG algorithms for use in generating variable-length keys for Vernam types of ciphers.

An excellent survey of PRNGs, with an extensive bibliography, is [[RITT91](#)]. [[MENE97](#)] also provides a good discussions of secure PRNGs. Another good treatment, with an emphasis on practical implementation issues, is RFC 1750. This RFC also describes a number of deskewing techniques. [[KELS98](#)] is a good survey of secure PRNG techniques and cryptanalytic attacks on them.

[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.

[FUMY93](#) Fumy, S., and Landrock, P. "Principles of Key Management." *IEEE Journal on Selected Areas in Communications*, June 1993.

[KELS98](#) Kelsey, J.; Schneier, B.; and Hall, C. "Cryptanalytic Attacks on Pseudorandom Number Generators." *Proceedings, Fast Software Encryption*, 1998. <http://www.schneier.com/paper-prngs.html>

[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.

[RITT91](#) Ritter, T. "The Efficient Generation of Cryptographic Confusion Sequences." *Cryptologia*, vol. 15 no. 2, 1991. <http://www.ciphersbyritter.com/ARTS/CRNG2ART.HTM>

[ZENG91](#) Zeng, K.; Yang, C.; Wei, D.; and Rao, T. "Pseudorandom Bit Generators in Stream-Cipher Cryptography." *Computer*, February 1991.



Recommended Web Sites

- **NIST Random Number Generation Technical Working Group:** Contains documents and tests developed by NIST that related to PRNGs for cryptographic applications. Also has useful set of links.
-

[Page 228]

- **LavaRnd:** LavaRnd is an open source project that uses a chaotic source to generate truly random numbers. The site also has background information on random numbers in general.
- **A Million Random Digits:** Compiled by the RAND Corporation.

◀ PREV

NEXT ▶

7.6. Key Terms, Review Questions, and Problems

Key Terms

[Blum, Blum, Shub generator](#)

[covert channel](#)

[deskewing](#)

[end-to-end encryption](#)

[key distribution](#)

[key distribution center \(KDC\)](#)

[linear congruential](#)

[link encryption](#)

[master key](#)

[nonce](#)

[pseudorandom number generator \(PRNG\)](#)

[session key](#)

[skew](#)

[traffic padding](#)

[true random number generator](#)

[wiring closet](#)

Review Questions

- 7.1 For a user workstation in a typical business environment, list potential locations for confidentiality attacks.
- 7.2 What is the difference between link and end-to-end encryption?

- 7.3 What types of information might be derived from a traffic analysis attack?
- 7.4 What is traffic padding and what is its purpose?
- 7.5 List ways in which secret keys can be distributed to two communicating parties.
- 7.6 What is the difference between a session key and a master key?
- 7.7 What is a nonce?
- 7.8 What is a key distribution center?
- 7.9 What is the difference between statistical randomness and unpredictability?

Problems

- 7.1 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*.
 - a.

Leaving aside considerations of security, discuss the relative advantages and disadvantages of the two methods.
 - b.

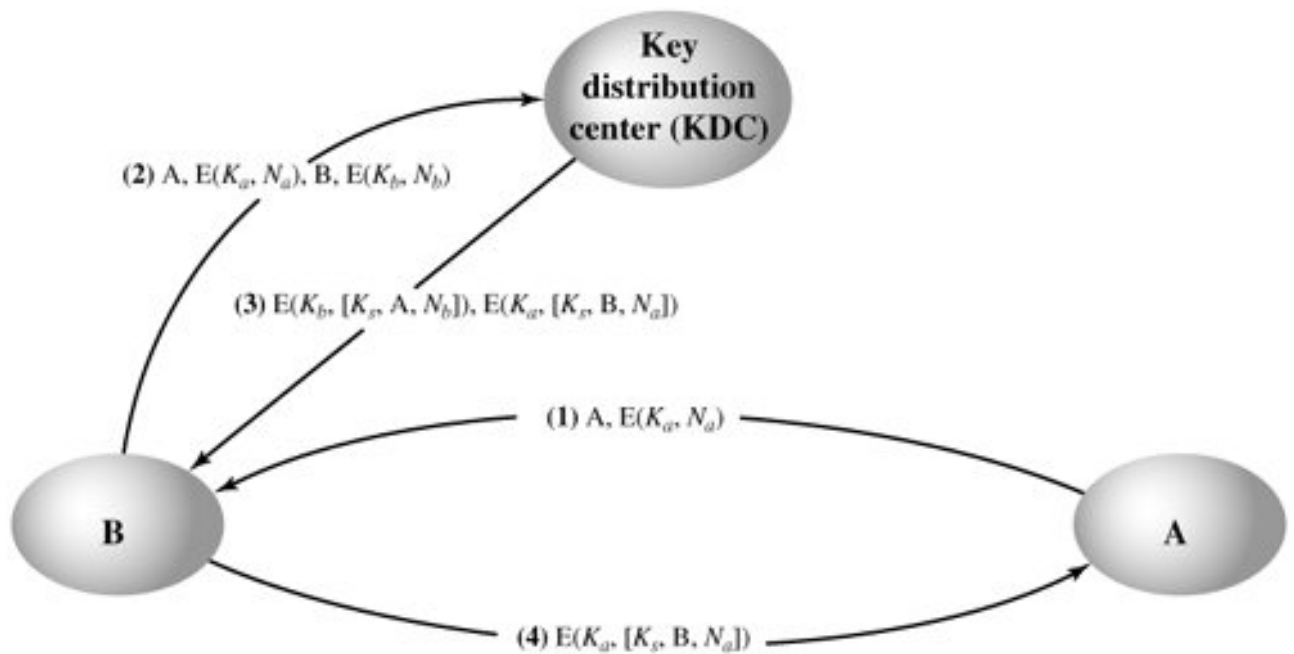
Discuss the security requirements and implications of the two methods.
- 7.2 [Section 7.2](#) describes the use of message length as a means of constructing a covert channel. Describe three additional schemes for using traffic patterns to construct a covert channel.
- 7.3 One local area network vendor provides a key distribution facility, as illustrated in [Figure 7.15](#).
 - a.

Describe the scheme.
 - b.

Compare this scheme to that of [Figure 7.9](#). What are the pros and cons?

Figure 7.15. Figure for Problem 7.3

[\[View full size image\]](#)



7.4 "We are under great pressure, Holmes." Detective Lestrade looked nervous. "We have learned that copies of sensitive government documents are stored in computers of one foreign embassy here in London. Normally these documents exist in electronic form only on a selected few government computers that satisfy the most stringent security requirements. However, sometimes they must be sent through the network connecting all government computers. But all messages in this network are encrypted using a top secret encryption algorithm certified by our best crypto experts. Even the 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 government 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 N 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 $E(K_a, R)$.

2.

Server responds by sending to $E(K_b, R)$ 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."

[Page 230]

"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 $E(K_x, R)$ implicitly authenticates user X as the sender, as only X (and the server) knows K_x . But you know that $E(K_x, R)$ 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 $E(K_a, R)$ 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 ...

Finish the sentence for Holmes.

7.5 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 $m - 1$ 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 .

7.6**a.**

What is the maximum period obtainable from the following generator?

$$X_{n+1} = (aX_n) \bmod 2^4$$

b.

What should be the value of a ?

c.

What restrictions are required on the seed?

7.7

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?

7.8

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?

7.9

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 the following: Given two randomly chosen integers, x and y , the probability that $\gcd(x, y) = 1$ is $6/\pi^2$. Use this theorem in a program to determine statistically the value of π . 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 π .

If the result is close to 3.14, congratulations! If not, then the result is probably low, usually a value of around 2.7. Why would such an inferior result be obtained?

7.10 Suppose you have a true random bit generator where each bit in the generated stream has the same probability of being a 0 or 1 as any other bit in the stream and that the bits are not correlated; that is the bits are generated from identical independent distribution.

However, the bit stream is biased. The probability of a 1 is $0.5 + \delta$ and the probability of a 0 is $0.5 - \delta$ where $0 < \delta < 0.5$. A simple deskewing algorithm is as follows: Examine the bit stream as a sequence of non-overlapping pairs. Discard all 00 and 11 pairs. Replace each 01 pair with 0 and each 10 pair with 1.

[Page 231]

a.

What is the probability of occurrence of each pair in the original sequence?

b.

What is the probability of occurrence of 0 and 1 in the modified sequence?

c.

What is the expected number of input bits to produce x output bits?

d.

Suppose that the algorithm uses overlapping successive bit pairs instead of nonoverlapping successive bit pairs. That is, the first output bit is based on input bits 1 and 2, the second output bit is based on input bits 2 and 3, and so on. What can you say about the output bit stream?

7.11 Another approach to deskewing is to consider the bit stream as a sequence of non-overlapping groups of n bits each and the output the parity of each group. That is, if a group contains an odd number of ones, the output is 1; otherwise the output is 0.

a.

Express this operation in terms of a basic Boolean function.

b.

Assume, as in the preceding problem, that the probability of a 1 is $0.5 + \delta$. If each group consists of 2 bits, what is the probability of an output of 1?

c.

If each group consists of 4 bits, what is the probability of an output of 1?

d.

Generalize the result to find the probability of an output of 1 for input groups of n

bits.

- 7.12** 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?

 PREY

NEXT 

Part Two: Public-Key Encryption and Hash Functions

For practical reasons, it is desirable to use different encryption and decryption keys in a crypto-system. Such asymmetric systems allow the encryption key to be made available to anyone while preserving confidence that only people who hold the decryption key can decipher the information.

Computers at Risk: Safe Computing in the Information Age, National Research Council, 1991

After symmetric encryption, the other major form of encryption is public-key encryption, which has revolutionized communications security. A related cryptographic area is that of cryptographic hash functions. Hash functions are used in conjunction with symmetric ciphers for digital signatures. In addition, hash functions are used for message authentication. Symmetric ciphers are also used for key management. All of these areas are discussed in [Part Two](#).

Road Map for Part Two

[Chapter 8](#): 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 8](#) provides an overview and numerous examples to clarify the concepts.

[Chapter 9](#): Public-Key Cryptography and RSA

[Chapter 9](#) introduces public-key cryptography and concentrates on its use to provide confidentiality. This chapter also examines the most widely used public-key cipher, the Rivest-Shamir-Adleman (RSA) algorithm.

[Chapter 10](#): Key Management; Other Public-Key Cryptosystems

[Chapter 10](#) revisits the issue of key management in light of the capabilities of symmetric ciphers. 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.

[Chapter 11](#): Message 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 11](#) 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 12](#): Hash and MAC Algorithms

[Chapter 12](#) extends the discussion of the preceding chapter to discuss two of the most important cryptographic hash functions (SHA and Whirlpool) and two of the most important MACs (HMAC) and CMAC.

[Chapter 13](#): Digital Signatures and Authentication Protocols

An important type of authentication is the digital signature. [Chapter 13](#) 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 14](#).

Chapter 8. Introduction to Number Theory

[8.1 Prime Numbers](#)

[8.2 Fermat's and Euler's Theorems](#)

[Fermat's Theorem](#)

[Euler's Totient Function](#)

[Euler's Theorem](#)

[8.3 Testing for Primality](#)

[Miller-Rabin Algorithm](#)

[A Deterministic Primality Algorithm](#)

[Distribution of Primes](#)

[8.4 The Chinese Remainder Theorem](#)

[8.5 Discrete Logarithms](#)

[The Powers of an Integer, Modulo \$n\$](#)

[Logarithms for Modular Arithmetic](#)

[Calculation of Discrete Logarithms](#)

[8.6 Recommended Reading and Web Site](#)

[8.7 Key Terms, Review Questions, and Problems](#)

[Key Terms](#)

[Review Questions](#)

[Problems](#)

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

Key Points

- A prime number is an integer that can only be divided without remainder by positive and negative values of itself and 1. Prime numbers play a critical role both in number theory and in cryptography.
- Two theorems that play important roles in public-key cryptography are Fermat's theorem and Euler's theorem.
- An important requirement in a number of cryptographic algorithms is the ability to choose a large prime number. An area of ongoing research is the development of efficient algorithms for determining if a randomly chosen large integer is a prime number.
- Discrete logarithms are fundamental to a number of public-key algorithms. Discrete logarithms are analogous to ordinary logarithms, but operate over modular arithmetic.

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.

As with [Chapter 4](#), this chapter includes a number of examples, each of which is highlighted in a shaded box.

8.1. Prime Numbers^[1]

^[1] In this section, unless otherwise noted, we deal only with the nonnegative integers. The use of negative integers would introduce no essential differences.

A central concern of number theory is the study of prime numbers. Indeed, whole books have been written on the subject (e.g., [CRAN01], [RIBE96]). In this section we provide an overview relevant to the concerns of this book.

An integer $p > 1$ is a prime number if and only if its only divisors^[2] are ± 1 and $\pm p$. Prime numbers play a critical role in number theory and in the techniques discussed in this chapter. Table 8.1 shows the primes less than 2000. Note the way the primes are distributed. In particular, note the number of primes in each range of 100 numbers.

^[2] Recall from Chapter 4 that integer a is said to be a divisor of integer b if there is no remainder on division. Equivalently, we say that a divides b .

Table 8.1. Primes under 2000

(This item is displayed on page 237 in the print version)

[\[View full size image\]](#)

2	101	211	307	401	503	601	701	809	0	1009	1103	1201	1301	1409	1511	1601	1709	1801	1901
3	103	223	311	409	509	607	709	811	911	1013	1109	1213	1303	1423	1523	1607	1721	1811	1907
5	107	227	313	419	521	613	719	821	919	1019	1117	1217	1307	1427	1531	1609	1723	1823	1913
7	109	229	317	421	523	617	727	823	929	1021	1123	1223	1319	1429	1543	1613	1733	1831	1931
11	113	233	331	431	541	619	733	827	937	1031	1129	1229	1321	1433	1549	1619	1741	1847	1933
13	127	239	337	433	547	631	739	829	941	1033	1151	1231	1327	1439	1553	1621	1747	1861	1949
17	131	241	347	439	557	641	743	839	947	1039	1153	1237	1361	1447	1559	1627	1753	1867	1951
19	137	251	349	443	563	643	751	853	953	1049	1163	1249	1367	1451	1567	1637	1759	1871	1973
23	139	257	353	449	569	647	757	857	967	1051	1171	1259	1373	1453	1571	1657	1777	1873	1979
29	149	263	359	457	571	653	761	859	971	1061	1181	1277	1381	1459	1579	1663	1783	1877	1987
31	151	269	367	461	577	659	769	863	977	1063	1187	1279	1399	1471	1583	1667	1787	1879	1999
37	157	271	373	463	587	661	773	877	983	1069	1193	1283		1481	1597	1669	1789	1889	1997
41	163	277	379	467	593	673	787	881	991	1087		1289		1483		1693			1999
43	167	281	383	479	599	677	797	883	997	1091		1291		1487		1697			
47	173	283	389	487		683		887		1093		1297		1489		1699			
53	179	293	397	491		691				1097				1493					
59	181			499										1499					
61	191																		
67	193																		
71	197																		
73	199																		
79																			
83																			
89																			
97																			

Any integer $a > 1$ can be factored in a unique way as

Equation 8-1

$$a = p_1^{a_1} p_2^{a_2} \dots p_t^{a_t}$$

where $p_1 < p_2 < \dots < p_t$ are prime numbers and where each is a positive integer. This is known as the fundamental theorem of arithmetic; a proof can be found in any text on number theory.

91	= 7 x 13
3600	= 2 ⁴ x 3 ² x 5 ²
11011	= 7 x 11 ² x 13

It is useful for what follows to express this another way. If P is the set of all prime numbers, then any positive integer a can be written uniquely in the following form:

$$a = \prod_{p \in P} p^{a_p} \quad \text{where each } a_p \geq 0$$

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.

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 $\{a_2 = 1, a_3 = 2\}$.
The integer 91 is represented by $\{a_7 = 2, a_{13} = 1\}$.

Multiplication of two numbers is equivalent to adding the corresponding exponents. Given a

$$a = \prod_{p \in P} p^{a_p} \quad b = \prod_{p \in P} p^{b_p}$$

. Define $k = ab$ We know that the integer k can be expressed as the

$$k = \prod_{p \in P} p^{k_p}$$

product of powers of primes:

. It follows that $k_p = a_p + b_p$ for all $p \in P$.

$k = 12 \times 18 = (2^2 \times 3) \times (2 \times 3^2) = 216$

$$k_2 = 2 + 1 = 3; k_3 = 1 + 2 = 3$$

$$216 = 2^3 \times 3^3 = 8 \times 27$$

What does it mean, in terms of the prime factors of a and b , to say that a divides b ? Any integer of the form can be divided only by an integer that is of a lesser or equal power of the same prime number, p^j with $j \leq n$. Thus, we can say the following:

Given $a = \prod_{p \in P} p^{a_p}$, $b = \prod_{p \in P} p^{b_p}$ If $a|b$, then $a_p \leq b_p$ then 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	$= 1 \leq 2 = b_3$
Thus, the inequality $a_p \leq b_p$ is satisfied for all prime numbers.	

It is easy to determine the greatest common divisor^[3] of two positive integers if we express each integer as the product of primes.

^[3] Recall from [Chapter 4](#) that the greatest common divisor of integers a and b , expressed $\gcd(a, b)$, is an integer c that divides both a and b without remainder and that any divisor of a and b is a divisor of c .

300	$= 2^2 \times 3^1 \times 5^2$
18	$= 2^1 \times 3^2$
$\gcd(18,300)$	$= 2^1 \times 3^1 \times 5^0 = 6$

The following relationship always holds:

If $k = \gcd(a,b)$ then $k_p = \min(a_p, b_p)$ 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 practical method of calculating the greatest common divisor.

8.2. 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^[4]

^[4] This is sometimes referred to as Fermat's little theorem.

Fermat's theorem states the following: If p is prime and a is a positive integer not divisible by p , then

Equation 8-2

$$a^{p-1} \equiv 1 \pmod{p}$$

Proof: Consider the set of positive integers less than p : $\{1, 2, \dots, p-1\}$ and multiply each element by a , modulo p , to get the set $X = \{a \pmod{p}, 2a \pmod{p}, \dots, (p-1)a \pmod{p}\}$. None of the elements of X is equal to zero because p does not divide a . Furthermore no two of the integers in X are equal. To see this, assume that $ja \equiv ka \pmod{p}$ where $1 \leq j < k \leq p-1$. Because a is relatively prime^[5] to p , we can eliminate a from both sides of the equation [see [Equation \(4.3\)](#)] resulting in: $j \equiv k \pmod{p}$. This last equality is impossible because j and k are both positive integers less than p . Therefore, we know that the $(p-1)$ elements of X are all positive integers, with no two elements equal. We can conclude the X consists of the set of integers $\{1, 2, \dots, p-1\}$ in some order. Multiplying the numbers in both sets and taking the result mod p yields

^[5] Recall from [Chapter 4](#) that two numbers are relatively prime if they have no prime factors in common; that is, their only common divisor is 1. This is equivalent to saying that two numbers are relatively prime if their greatest common divisor is 1.

$$a \times 2a \times \dots \times (p-1)a \equiv [(1 \times 2 \times \dots \times (p-1)) \pmod{p}]$$

$$a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}$$

We can cancel the $(p-1)!$ term because it is relatively prime to p [see [Equation \(4.3\)](#)]. This yields [Equation \(8.2\)](#).

$a = 7, p = 19$
$7^2 = 49 \equiv 11 \pmod{19}$

$$7^4 \equiv 121 \equiv 7 \pmod{19}$$

$$7^8 \equiv 49 \equiv 7 \pmod{19}$$

$$7^{16} \equiv 121 \equiv 7 \pmod{19}$$

$$a^{p-1} = 7^{18} = 7^{16} \times 7^2 \equiv 7 \times 11 \equiv 1 \pmod{19}$$

An alternative form of Fermat's theorem is also useful: If p is prime and a is a positive integer, then

Equation 8-3

$$a^p \equiv a \pmod{p}$$

Note that the first form of the theorem [[Equation \(8.2\)](#)] requires that a be relatively prime to p , but this form does not.

$p = 5, a = 3$	$a^p = 3^5 = 243 \equiv 3 \pmod{5} = a \pmod{p}$
$p = 5, a = 10$	$a^p = 10^5 = 100000 \equiv 10 \pmod{5} = 0 \pmod{5} = a \pmod{p}$

[Page 240]

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 $\phi(n)$, defined as the number of positive integers less than n and relatively prime to n . By convention, $\phi(1) = 1$.

Determine $\phi(37)$ and $\phi(35)$.

Because 37 is prime, all of the positive integers from 1 through 36 are relatively prime to 37. Thus $\phi(37) = 36$.

To determine $\phi(35)$, we list all of the positive integers less than 35 that are relatively prime to it:

1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18,

19, 22, 23, 24, 26, 27, 29, 31, 32, 33, 34.

There are 24 numbers on the list, so $\phi(35) = 24$.

[Table 8.2](#) lists the first 30 values of $\phi(n)$. The value $\phi(1)$ is without meaning but is defined to have the value 1.

**Table
8.2.
Some
Values
of
Euler's
Totient
Function
 $f(n)$**

n	f(n)
1	1
2	1
3	2
4	2
5	4
6	2
7	6
8	4
9	6
10	4
11	10
12	4
13	12
14	6
15	8
16	8
17	16
18	6
19	18
20	8
21	12

22	10
23	22
24	8
25	20
26	12
27	18
28	12
29	28
30	8

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 , with $p \neq q$. Then we can show that for $n = pq$,

$$\phi(n) = \phi(pq) = \phi(p) \times \phi(q) = (p - 1) \times (q - 1)$$

To see that $\phi(n) = \phi(p) \times \phi(q)$, consider that the set of positive integers less than n is the set $\{1, \dots, (pq - 1)\}$. The integers in this set that are not relatively prime to n are the set $\{p, 2p, \dots, (q - 1)p\}$ and the set $\{q, 2q, \dots, (p - 1)q\}$. Accordingly,

$$\phi(n) = (pq - 1) - [(q - 1)p + (p - 1)q]$$

$$= pq - (p + q) + 1$$

$$= (p - 1) \times (q - 1)$$

$$= \phi(p) \times \phi(q)$$

$$\phi(21) = \phi(3) \times \phi(7) = (3 - 1) \times (7 - 1) = 2 \times 6 = 12$$

where the 12 integers are $\{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$

Euler's Theorem

Euler's theorem states that for every a and n that are relatively prime:

Equation 8-4

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

$a = 3; n = 10; \phi(10) = 4$	$a^{\phi(n)} = 3^4 = 81 \equiv 1 \pmod{10} = 1 \pmod{n}$
$a = 2; n = 11; \phi(11) = 10$	$a^{\phi(n)} = 2^{10} = 1024 \equiv 1 \pmod{11} = 1 \pmod{n}$

Proof: [Equation \(8.4\)](#) 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:

$$R = \{x_1, x_2, \dots, x_{\phi(n)}\}$$

That is, each element x_i of R is a unique positive integer less than n with $\gcd(x_i, n) = 1$. Now multiply each element by a , modulo n :

$$S = \{(ax_1 \pmod{n}), (ax_2 \pmod{n}), \dots, (ax_{\phi(n)} \pmod{n})\}$$

The set S 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 that are less than n and that are relatively prime to n .

2.

There are no duplicates in S . Refer to [Equation \(4.3\)](#). If $ax_i \pmod{n} = ax_j \pmod{n}$ then $x_i = x_j$.

Therefore,

$$\prod_{i=1}^{\phi(n)} (ax_i \pmod{n}) = \prod_{i=1}^{\phi(n)} x_i$$

$$\prod_{i=1}^{\phi(n)} ax_i \equiv \prod_{i=1}^{\phi(n)} x_i \pmod{n}$$

$$a^{\phi(n)} \times \left[\prod_{i=1}^{\phi(n)} x_i \right] \equiv \prod_{i=1}^{\phi(n)} x_i \pmod{n}$$
$$a^{\phi(n)} \equiv 1 \pmod{n}$$

This is the same line of reasoning applied to the proof of Fermat's theorem. As is the case for Fermat's theorem, an alternative form of the theorem is also useful:

Equation 8-5

$$a^{\phi(n)+1} \equiv a \pmod{n}$$

Again, similar to the case with Fermat's theorem, the first form of Euler's theorem [[Equation \(8.4\)](#)] requires that a be relatively prime to n , but this form does not.

◀ PREV

NEXT ▶

8.3. Testing for Primality

For many cryptographic algorithms, it is necessary to select one or more very large prime numbers at random. Thus we are faced with the task of determining whether a given large number is prime. There is no simple yet efficient means of accomplishing this task.

In this section, we present one attractive and popular algorithm. You may be surprised to learn that this algorithm yields a number that is not necessarily a prime. However, the algorithm can yield a number that is almost certainly a prime. This will be explained presently. We also make reference to a deterministic algorithm for finding primes. The section closes with a discussion concerning the distribution of primes.

Miller-Rabin Algorithm^[6]

^[6] Also referred to in the literature as the Rabin-Miller algorithm, or the Rabin-Miller test, or the Miller-Rabin test.

The algorithm due to Miller and Rabin [[MILL75](#), [RABI80](#)] is typically used to test a large number for primality. Before explaining the algorithm, we need some background. First, any positive odd integer $n \geq 3$ can be expressed as follows:

$$n - 1 = 2^k q \text{ with } k > 0, q \text{ odd}$$

To see this, note that $(n - 1)$ is an even integer. Then, divide $(n - 1)$ by 2 until the result is an odd number q , for a total of k divisions. If n is expressed as a binary number, then the result is achieved by shifting the number to the right until the rightmost digit is a 1, for a total of k shifts. We now develop two properties of prime numbers that we will need.

Two Properties of Prime Numbers

The **first property** is stated as follows: If p is prime and a is a positive integer less than p , then $a^2 \bmod p = 1$ if and only if either $a \bmod p = 1$ or $a \bmod p = p - 1$. By the rules of modular arithmetic $(a \bmod p)(a \bmod p) = a^2 \bmod p$. Thus if either $a \bmod p = 1$ or $a \bmod p = p - 1$, then $a^2 \bmod p = 1$. Conversely, if $a^2 \bmod p = 1$, then $(a \bmod p)^2 = 1$, which is true only for $a \bmod p = 1$ or $a \bmod p = p - 1$.

The **second property** is stated as follows: Let p be a prime number greater than 2. We can then write $p - 1 = 2^k q$, with $k > 0$ q odd. Let a be any integer in the range $1 < a < p - 1$. Then one of the two following conditions is true:

1.

a^q is congruent to 1 modulo p . That is, $a^q \bmod p = 1$, or equivalently, $a^q \equiv 1 \pmod{p}$.

2.

One of the numbers $a^q, a^{2^q}, a^{4^q}, \dots, a^{2^{k-1}q}$ is congruent to 1 modulo p . That is, there is some number j in the range $(1 \leq j \leq k)$ such that $a^{2^{j-1}q} \bmod p = 1 \bmod p = p \cdot 1$, or equivalently, $a^{2^{j-1}q} \equiv 1 \pmod{p}$.

Proof: Fermat's theorem [Equation (8.2)] states that $a^{n-1} \equiv 1 \pmod{n}$ if n is prime. We have $p-1 = 2^k q$. Thus, we know that $a^{p-1} \bmod p = a^{2^k q} \bmod p = 1$. Thus, if we look at the sequence of numbers

Equation 8-6

$$a^q \bmod p, a^{2^q} \bmod p, a^{4^q} \bmod p, \dots, a^{2^{k-1}q} \bmod p, a^{2^k q} \bmod p$$

we know that the last number in the list has value 1. Further, each number in the list is the square of the previous number. Therefore, one of the following possibilities must be true:

1.

The first number on the list, and therefore all subsequent numbers on the list, equals 1.

2.

Some number on the list does not equal 1, but its square mod p does equal 1. By virtue of the first property of prime numbers defined above, we know that the only number that satisfies this condition $p-1$ is $p-1$. So, in this case, the list contains an element equal to $p-1$.

This completes the proof.

Details of the Algorithm

These considerations lead to the conclusion that if n is prime, then either the first element in the list of residues, or remainders, $(a^q, a^{2^q}, \dots, a^{2^{k-1}q}, a^{2^k q})$ modulo n equals 1, or some element in the list equals $(n-1)$; otherwise n is composite (i.e., not a prime). On the other hand, if the condition is met, that does not necessarily mean that n is prime. For example, if $n = 2047 = 23 \times 89$, then $n-1 = 2 \times 1023$. Computing, $2^{1023} \bmod 2047 = 1$, so that 2047 meets the condition but is not prime.

We can use the preceding property to devise a test for primality. The procedure TEST takes a candidate integer n as input and returns the result **composite** if n is definitely not a prime, and the result **inconclusive** if n may or may not be a prime.

TEST (n)

1. Find integers k, q , with $k > 0, q$ odd, so that $(n-1) = 2^k q$;
2. Select a random integer $a, 1 < a < n-1$;
3. if $a^q \bmod n = 1$ then return("inconclusive");
4. for $j = 0$ to $k-1$ do


```

5.     if  $a^{2^j q} \bmod n \equiv n - 1$  then return("inconclusive");
6.     return("composite");

```

[Page 244]

Let us apply the test to the prime number $n = 29$. We have $(n - 1) = 28 = 2^2(7) = 2^k q$. First, let us try $a = 10$. We compute $10^7 \bmod 29 = 17$, which is neither 1 nor 28, so we continue the test. The next calculation finds that $(10^7)^2 \bmod 29 = 28$, and the test returns **inconclusive** (i.e., 29 may be prime). Let's try again with $a = 2$. We have the following calculations: $2^7 \bmod 29 = 12$; $2^{14} \bmod 29 = 28$; and the test again returns **inconclusive**. If we perform the test for all integers a in the range 1 through 28, we get the same inconclusive result, which is compatible with n being a prime number.

Now let us apply the test to the composite number $n = 13 \times 17 = 221$. Then $(n - 1) = 220 = 2^2(55) = 2^k q$. Let us try $a = 5$. Then we have $5^{55} \bmod 221 = 112$, which is neither 1 nor 220; $(5^{55})^2 \bmod 221 = 168$. Because we have used all values of j (i.e., $j = 0$ and $j = 1$) in line 4 of the TEST algorithm, the test returns **composite**, indicating that 221 is definitely a composite number. But suppose we had selected $a = 21$. Then we have $21^{55} \bmod 221 = 200$; $(21^{55})^2 \bmod 221 = 220$; and the test returns **inconclusive**, indicating that 221 may be prime. In fact, of the 218 integers from 2 through 219, four of these will return an inconclusive result, namely 21, 47, 174, and 200.

Repeated Use of the Miller-Rabin Algorithm

How can we use the Miller-Rabin algorithm to determine with a high degree of confidence whether or not an integer is prime? It can be shown [KNUT98] that given an odd number n that is not prime and a randomly chosen integer, a with $1 < a < n - 1$, the probability that TEST will return **inconclusive** (i.e., fail to detect that n is not prime) is less than $1/4$. Thus, if t different values of a are chosen, the probability that all of them will pass TEST (return inconclusive) for n is less than $(1/4)^t$. For example, for $t = 10$, the probability that a nonprime number will pass all ten tests is less than 10^{-6} . Thus, for a sufficiently large value of t , we can be confident that n is prime if Miller's test always returns **inconclusive**.

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 TEST (n) using randomly chosen values for a . If, at any point, TEST returns **composite**, then n is determined to be nonprime. If TEST continues to return **inconclusive** for t tests, for a sufficiently large value of t , assume that n is prime.

A Deterministic Primality Algorithm

Prior to 2002, there was no known method of efficiently proving the primality of very large numbers. All of the algorithms in use, including the most popular (Miller-Rabin), produced a probabilistic result. In 2002, Agrawal, Kayal, and Saxena [AGRA02] developed a relatively simple deterministic algorithm that efficiently determines whether a given large number is a prime. The algorithm, known as the AKS algorithm, does not appear to be as efficient as the Miller-Rabin algorithm. Thus far, it has not supplanted this older, probabilistic technique [BORN03].

Distribution of Primes

It is worth noting how many numbers are likely to be rejected before a prime number is found using the Miller-Rabin test, or any other test for primality. 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. Because all even integers can be immediately rejected, the correct figure is $0.5 \ln(n)$. For example, if a prime on the order of magnitude of 2^{200} were sought, then about $0.5 \ln(2^{200}) = 69$ trials would be needed to find a prime. However, this figure is just an average. In some places along the number line, primes are closely packed, and in other places there are large gaps.

The two consecutive odd integers 1,000,000,000,061 and 1,000,000,000,063 are both prime. On the other hand, $1001! + 2, 1001! + 3, \dots, 1001! + 1000, 1001! + 1001$ is a sequence of 1000 consecutive composite integers.

[◀ PREV](#)[NEXT ▶](#)

8.4. The Chinese Remainder Theorem

One of the most useful results of number theory is the Chinese remainder theorem (CRT).^[7] 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.

^[7] The CRT is so called because it is believed to have been discovered by the Chinese mathematician Sun-Tsu in around 100 A.D.

The 10 integers in Z_{10} , that is the integers 0 through 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 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 8.17. 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 A in Z^M by a k -tuple whose elements are in Z_{m_i} using the following correspondence:

Equation 8-7

$$A \leftrightarrow (a_1, a_2, \dots, a_k)$$

where $A \in Z_M$, $a_i \in Z_{m_i}$ and $a_i = A \bmod m_i$ for $1 \leq i \leq k$. The CRT makes two assertions.

The mapping of [Equation \(8.7\)](#) is a one-to-one correspondence (called a **bijection**) between Z_M and the Cartesian product $Z_{m_1} \times Z_{m_2} \times \dots \times Z_{m_k}$. 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 integer A in Z_M .

2.

Operations performed on the elements of Z_M can be equivalently performed on the corresponding k -tuples by performing the operation independently in each coordinate position in the appropriate system.

Let us demonstrate the **first assertion**. The transformation from A to (a_1, a_2, \dots, a_k) is obviously unique; that is, each a_i is uniquely calculated as $a_i = A \bmod m_i$. 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

Equation 8-8

$$c_i = M_i \times (M_i^{-1} \bmod m_i) \quad \text{for } 1 \leq i \leq k$$

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 \(8.8\)](#) is well defined and produces a unique value c_i . We can now compute:

Equation 8-9

$$A \equiv \left(\sum_{i=1}^k a_i c_i \right) \pmod{M}$$

To show that the value of A produced by [Equation \(8.9\)](#) 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. That is, the second assertion can be stated as follows: If

$$\begin{aligned} A &\longleftrightarrow (a_1, a_2, \dots, a_k) \\ B &\longleftrightarrow (b_1, b_2, \dots, b_k) \end{aligned}$$

then

$$(A + B) \bmod M \iff ((a_1 + b_1) \bmod m_1, \dots, (a_k + b_k) \bmod m_k)$$

$$(A - B) \bmod M \iff ((a_1 - b_1) \bmod m_1, \dots, (a_k - b_k) \bmod m_k)$$

$$(A \times B) \bmod M \iff ((a_1 \times b_1) \bmod m_1, \dots, (a_k \times b_k) \bmod m_k)$$

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. However, note that it is necessary to know beforehand the factorization of M .

[Page 247]

To represent $973 \bmod 1813$ as a pair of numbers mod 37 and 49, define [\[8\]](#)

$$m_1 = 37$$

$$m_2 = 49$$

$$M = 1813$$

$$A = 973$$

We also have $M_1 = 49$ and $M_2 = 37$. Using the extended Euclidean algorithm, we compute

$$M_1^{-1} = 34 \bmod m_1 \text{ and } M_2^{-1} = 4 \bmod m_2. \text{ (Note that we only need to compute each } M_i$$

and each M_i^{-1} once.) 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 $(678) \iff (678 \bmod 37, 678 \bmod 49) = (12, 41)$. Then we add the 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

$$(23, 34) \iff a_1 M_1 M_1^{-1} + a_2 M_2 M_2^{-1} \bmod M$$

$$= [(23)(49)(34) + (34)(37)(4)] \bmod 1813$$

$$= 43350 \bmod 1813$$

$$= 1651$$

and check that it is equal to $(973 + 678) \bmod 1813 = 1651$. Remember that in the above derivation, M_1^{-1} is the multiplicative inverse of M_1 modulo m_1 , and M_2^{-1} is the multiplicative inverse of M_2 modulo m_2 .

Suppose we want to multiply $1651 \pmod{1813}$ 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

$$(32, 14) \longleftrightarrow [(14)(49)(34) + (32)(37)(4)] \bmod 1813$$

$$= 865$$

$$= 1651 \times 73 \bmod 1813$$

^[8] This example was provided by Professor Ken Calvert of Georgia Tech.

8.5. 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 [ORE67] and [LEVE90].

The Powers of an Integer, Modulo n

Recall from Euler's theorem [Equation (8.4)] that, for every a and n that are relatively prime:

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

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:

Equation 8-10

$$a^m \equiv 1 \pmod{n}$$

If a and n are relatively prime, then there is at least one integer m that satisfies Equation (8.10), namely, $m = \phi(n)$. The least positive exponent m for which Equation (8.10) holds is referred to in several ways:

- the order of $a \pmod{n}$
- the exponent to which a belongs \pmod{n}
- the length of the period generated by a

To see this last point, consider the powers of 7, modulo 19:

$$\begin{aligned}7^1 &\equiv 7 \pmod{19} \\7^2 &= 49 = 2 \times 19 + 11 \equiv 11 \pmod{19} \\7^3 &= 343 = 18 \times 19 + 1 \equiv 1 \pmod{19} \\7^4 &= 2401 = 126 \times 19 + 7 \equiv 7 \pmod{19} \\7^5 &= 16807 = 884 \times 19 + 11 \equiv 11 \pmod{19}\end{aligned}$$

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 8.3](#) 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 8.3. Powers of Integers, Modulo 19

[\[View full size image\]](#)

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

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 $2, 4, p^\alpha$, and $2p^\alpha$, where p is any odd prime and α is a positive integer. The proof is not simple but can be found in many number theory books, including [ORE76].

Logarithms for Modular Arithmetic

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:

$$\log_x(1) = 0$$

$$\log_x(x) = 1$$

Equation 8-11

$$\log_x(yz) = \log_x(y) + \log_x(z)$$

Equation 8-12

$$\log_x(y^r) = r \times \log_x(y)$$

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 satisfies

$$b \equiv r \pmod{p} \text{ for some } r, \text{ where } 0 \leq r \leq (p - 1)$$

by the definition of modular arithmetic. It follows that for any integer b and a primitive root a of prime number p , we can find a unique exponent i such that

$$b \equiv a^i \pmod{p} \text{ where } 0 \leq i \leq (p - 1)$$

[Page 251]

This exponent i is referred to as the **discrete logarithm** of the number b for the base $a \pmod{p}$. We denote this value as $\text{dlog}_{a,p}(b)$.^[9]

^[9] Many texts refer to the discrete logarithm as the *index*. There is no generally agreed notation for this concept, much less an agreed name.

Note the following:

Equation 8-13

$$\text{dlog}_{a,p}(1) = 0, \text{ because } a^0 \pmod{p} = 1 \pmod{p} = 1$$

Equation 8-14

$$\text{dlog}_{a,p}(a) = 1, \text{ because } a^1 \bmod p = a$$

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

$$2^0 = 1 \quad 2^4 \equiv 7 \pmod{9}$$

$$2^1 = 2 \quad 2^5 \equiv 5 \pmod{9}$$

$$2^2 = 4 \quad 2^6 \equiv 1 \pmod{9}$$

$$2^3 = 8$$

This gives us the following table of the numbers with given discrete logarithms (mod 9) for the root $a = 2$:

Logarithm	0	1	2	3	4	5
Number	1	2	4	8	7	5

To make it easy to obtain the discrete logarithms of a given number, we rearrange the table:

Number	1	2	4	5	7	8
Logarithm	0	1	2	5	4	3

Now consider

$$x = a^{\text{dlog}_{a,p}(x)} \bmod p \quad y = a^{\text{dlog}_{a,p}(y)} \bmod p$$

$$xy = a^{\text{dlog}_{a,p}(xy)} \bmod p$$

Using the rules of modular multiplication,

$$xy \bmod p = [(x \bmod p)(y \bmod p)] \bmod p$$

$$\begin{aligned} a^{\text{dlog}_{a,p}(xy)} \bmod p &= \left[\left(a^{\text{dlog}_{a,p}(x)} \bmod p \right) \left(a^{\text{dlog}_{a,p}(y)} \bmod p \right) \right] \bmod p \\ &= \left(a^{\text{dlog}_{a,p}(x) + \text{dlog}_{a,p}(y)} \right) \bmod p \end{aligned}$$

But now consider Euler's theorem, which states that, for every a and n that are relatively prime:

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

[Page 253]

Any positive integer z can be expressed in the form $z = q + k\phi(n)$, with $0 \leq q < \phi(n)$. Therefore, by Euler's theorem,

$$a^z \equiv a^q \pmod{n} \text{ if } z \equiv q \pmod{\phi(n)}$$

Applying this to the foregoing equality, we have

$$\text{dlog}_{a,p}(xy) \equiv [\text{dlog}_{a,p}(x) + \text{dlog}_{a,p}(y)] \pmod{\phi(p)}$$

and generalizing,

$$\text{dlog}_{a,p}(y^r) \equiv [r \times \text{dlog}_{a,p}(y)] \pmod{\phi(n)}$$

This demonstrates the analogy between true logarithms and 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 8.4](#), which is directly derived from [Table 8.3](#), shows the sets of discrete logarithms that can be defined for modulus 19.

Table 8.4. Tables of Discrete Logarithms, Modulo 19

(This item is displayed on page 252 in the print version)

[\[View full size image\]](#)

(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
$\log_{2,19}(a)$	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
$\log_{3,19}(a)$	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
$\log_{10,19}(a)$	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
$\log_{13,19}(a)$	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
$\log_{14,19}(a)$	18	13	7	8	10	2	6	3	14	5	12	15	11	1	17	16	4	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
$\log_{15,19}(a)$	18	5	11	10	8	16	12	15	4	13	6	3	7	17	1	2	14	9

Calculation of Discrete Logarithms

Consider the equation

$$y = g^x \text{ mod } p$$

Given g , x , and p , it is a straightforward matter to calculate y . At the worst, we must perform x repeated multiplications, and algorithms exist for achieving greater efficiency (see [Chapter 9](#)).

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 [\[BETH91\]](#):

$$e^{((\ln p)^{1/3}(\ln(\ln p))^{2/3})}$$

which is not feasible for large primes.

8.6. Recommended Reading and Web Sites

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 [ROSE05]. [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,

ROSE05 Rosen, K. *Elementary Number Theory and its Applications*. Reading, MA: Addison-Wesley, 2000.

Recommended Web Site



- **The Prime Pages:** Prime number research, records, and resources

8.7. Key Terms, Review Questions, and Problems

Key Terms

[bijection](#)

[composite number](#)

[Chinese remainder theorem](#)

[discrete logarithm](#)

[Euler's theorem](#)

[Euler's totient function](#)

[Fermat's theorem](#)

[index](#)

[order](#)

[prime number](#)

[primitive root](#)

Review Questions

- 8.1 What is a prime number?
- 8.2 What is the meaning of the expression a divides b ?
- 8.3 What is Euler's totient function?
- 8.4 The Miller-Rabin test can determine if a number is not prime but cannot determine if a number is prime. How can such an algorithm be used to test for primality?
- 8.5 What is a primitive root of a number?
- 8.6 What is the difference between an index and a discrete logarithm?

Problems

- 8.1** The purpose of this problem is to determine how many prime numbers there are. Suppose there are a total of n prime numbers, and we list these in order:

$$p_1 = 2 < p_2 = 3 < p_3 = 5 < \dots < p_n.$$

a.

Define $X = 1 + p_1 p_2 \dots p_n$. That is, X is equal to one plus the product of all the primes. Can we find a prime number p_m that divides X ?

b.

What can you say about m ?

c.

Deduce that the total number of primes cannot be finite.

d.

Show that $p_{n+1} \leq 1 + p_1 p_2 \dots p_n$.

[Page 255]

- 8.2** The purpose of this problem is to demonstrate 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 quantity $\gcd\left(\frac{a}{d}, \frac{b}{d}\right)$

b.

The sum of the result of part (a) over all possible values of d is 1. That is:

$$\sum_{d \geq 1} \Pr[\gcd(a, b) = d] = 1$$

. Use this equality to determine the value of P .
Hint: Use

the identity $\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$.

- 8.3** Why is $\gcd(n, n + 1) = 1$ for two consecutive integers n and $n + 1$?
- 8.4** Using Fermat's theorem, find $3^{201} \pmod{11}$.
- 8.5** Use Fermat's Theorem to find a number a between 0 and 72 with a congruent to 9794 modulo 73.
- 8.6** Use Fermat's Theorem to find a number x between 0 and 28 with x^{85} congruent to 6 modulo 29. (You should not need to use any brute force searching.)
- 8.7** Use Euler's Theorem to find a number a between 0 and 9 such that a is congruent to 7^{1000} modulo 10. (Note that this is the same as the last digit of the decimal expansion of 7^{1000} .)
- 8.8** Use Euler's Theorem to find a number x between 0 and 28 with x^{85} congruent to 6 modulo 35. (You should not need to use any brute force searching.)
- 8.9** Notice in [Table 8.2](#) that $\phi(n)$ is even for $n > 2$. This is true for all $n > 2$. Give a concise argument why this is so.
- 8.10** Prove the following: If p is prime, then $\phi(p^i) = p^i - p^{i-1}$. *Hint:* What numbers have a factor in common with p^i ?
- 8.11** It can be shown (see any book on number theory) that if $\gcd(m, n) = 1$ then $\phi(mn) = \phi(m)\phi(n)$. Using this property and the property developed in the preceding problem and the property that $\phi(p) = p - 1$ for p prime, it is straightforward to determine the value of $\phi(n)$ for any n . Determine the following:
- a.**
- $\phi(41)$
- b.**
- $\phi(27)$
- c.**
- $\phi(231)$
- d.**
- $\phi(440)$

8.12 It can also be shown that for arbitrary positive integer a , $\phi(a)$ is given by:

$$\phi(a) = \prod_{i=1}^t [p_i^{a_i-1}(p_i - 1)]$$

where a is given by [Equation \(8.1\)](#), namely: $a = p_1^{a_1} p_2^{a_2} \cdots p_t^{a_t}$. Demonstrate this result.

8.13 Consider the function: $f(n)$ = number of elements in the set $\{a: 0 \leq a < n \text{ and } \gcd(a, n) = 1\}$. What is this function?

8.14 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 $(2^n - 2)$.

a.

Give an example that satisfies the condition using an odd prime.

b.

The condition is obviously true for $n = 2$. Prove that the condition is true if n is an odd prime (proving the **if** condition)

c.

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.

d.

Unfortunately, the ancient Chinese never tried $n = 341$, which is nonprime ($341 = 11 \times 31$) and yet 341 divides $2^{341} - 2$ with out remainder. Demonstrate that $2^{341} \equiv 2 \pmod{341}$ (disproving the **only if** condition). *Hint:* It is not necessary to calculate 2^{341} ; play around with the congruences instead.

8.15 Show that if n is an odd composite integer, then the Miller-Rabin test will return **inconclusive** for $a = 1$ and $a = (n - 1)$.

- 8.16** If n is composite and passes the Miller-Rabin test for the base a , then n is called a *strong pseudoprime to the base a* . Show that 2047 is a strong pseudoprime to the base 2.
- 8.17** A common formulation of the Chinese remainder theorem (CRT) is as follows: 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:

$$x \equiv a_1 \pmod{m_1}$$

$$x \equiv a_2 \pmod{m_2}$$

⋮

$$x \equiv a_k \pmod{m_k}$$

has a unique solution modulo M . Show that the theorem stated in this form is true.

- 8.18** The example used by Sun-Tsu to illustrate the CRT was

$$x \equiv 2 \pmod{3}; \quad x \equiv 3 \pmod{5}; \quad x \equiv 2 \pmod{7}$$

Solve for x .

- 8.19** 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.
- 8.20** Find all primitive roots of 25.

8.21 Given 2 as a primitive root of 29, construct a table of discrete logarithms, and use it to solve the following congruences:

a.

$$17x^2 \equiv 10 \pmod{29}$$

b.

$$x^2 - 4x + 16 \equiv 0 \pmod{29}$$

c.

$$x^7 \equiv 17 \pmod{29}$$

Programming Problems

8.22 Write a computer program that implements fast exponentiation (successive squaring) modulo n .

8.23 Write a computer program that implements the Miller-Rabin algorithm for a user-specified n . The program should allow the user two choices: (1) specify a possible witness a to test using the Witness procedure, or (2) specify a number s of random witnesses for the Miller-Rabin test to check.

Chapter 9. Public-Key Cryptography and RSA

9.1 Principles of Public-Key Cryptosystems

[Public-Key Cryptosystems](#)

[Applications for Public-Key Cryptosystems](#)

[Requirements for Public-Key Cryptography](#)

[Public-Key Cryptanalysis](#)

9.2 The RSA Algorithm

[Description of the Algorithm](#)

[Computational Aspects](#)

[The Security of RSA](#)

9.3 Recommended Reading and Web Site

9.4 Key Terms, Review Questions, and Problems

[Key Terms](#)

[Review Questions](#)

[Problems](#)

Appendix 9A Proof of the RSA Algorithm

[Basic Results](#)

[Proof](#)

Appendix 9B The Complexity of Algorithms

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

Key Points

- Asymmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the different keys one a public key and one a private key. It is also known as public-key encryption.
- Asymmetric encryption transforms plaintext into ciphertext using a one of two keys and an encryption algorithm. Using the paired key and a decryption algorithm, the plaintext is recovered from the ciphertext.
- Asymmetric encryption can be used for confidentiality, authentication, or both.
- The most widely used public-key cryptosystem is RSA. The difficulty of attacking RSA is based on the difficulty of finding the prime factors of a composite number.

The 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 symmetric 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 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 symmetric encryption. 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 symmetric 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 symmetric encryption obsolete. On the contrary, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that symmetric 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 symmetric encryption. In fact, some form of protocol is needed, generally involving a central agent, and the procedures involved are not simpler nor any more efficient than those required for symmetric encryption (e.g., see analysis in [\[NEED78\]](#)).

This chapter and the next provide an overview of public-key cryptography. 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. Further topics are explored in [Chapter 10](#) and Appendix F.

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 8](#) provides the necessary background in number theory.

[← PREV](#)

[NEXT →](#)

9.1. 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 symmetric encryption. The first problem is that of key distribution, which was examined in some detail in [Chapter 7](#).

As we have seen, key distribution under symmetric encryption requires either (1) that two communicants already share a key, which somehow has been distributed to them; or (2) 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 13](#).

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. ^[1]

^[1] 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 National Security Agency (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 nonsecret encryption and describes the discovery in [[ELLI99](#)].

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

Asymmetric 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.

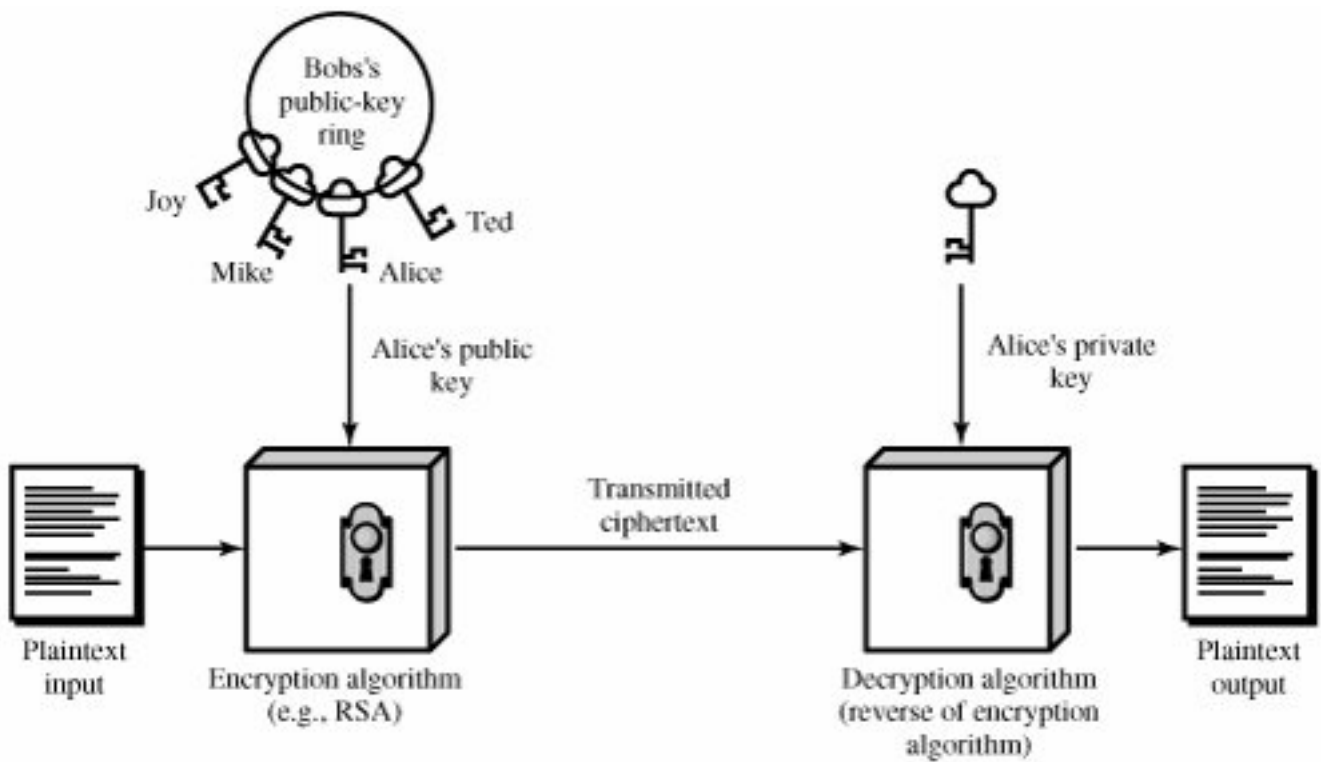
A public-key encryption scheme has six ingredients ([Figure 9.1a](#); compare with [Figure 2.1](#)):

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
- **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

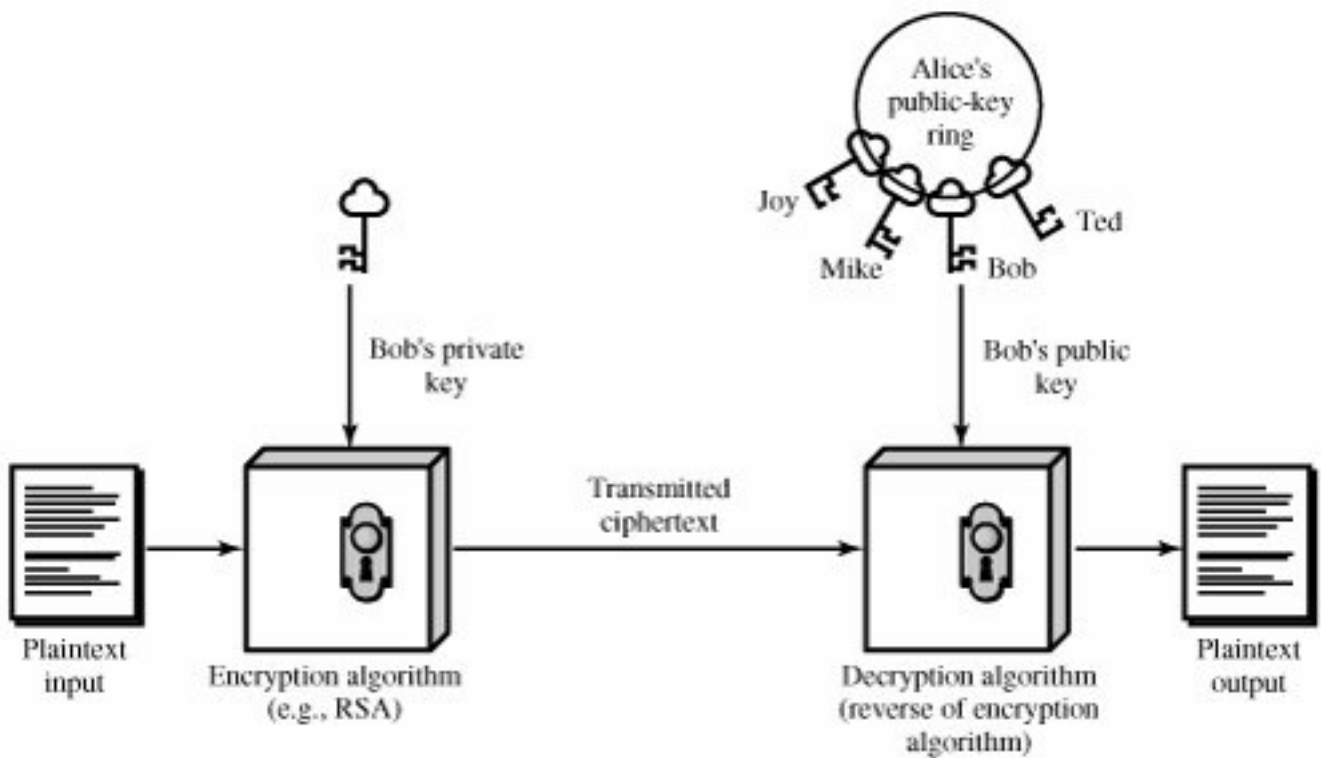
[Page 261]

Figure 9.1. Public-Key Cryptography

[\[View full size image\]](#)



(a) Encryption



(b) Authentication

The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As [Figure 9.1a](#) suggests, each user maintains a collection of public keys obtained from others.

3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

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 user's private key remains protected and secret, 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 9.1](#) summarizes some of the important aspects of symmetric and public-key encryption. To discriminate between the two, we refer to the key used in symmetric encryption as a **secret key**. The two keys used for asymmetric encryption are referred to as the **public key** and the **private key**.^[2] Invariably, the private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with symmetric encryption.

^[2] The following notation is used consistently throughout. A secret key is represented by K_m , where m is some modifier; for example, K_a is a secret key owned by user A. A public key is represented by PU_a , for user A, and the corresponding private key is PR_a . Encryption of plaintext X can be performed with a secret key, a public key, or a private key, denoted by $E(K_a, X)$, $E(PU_a, X)$, and $E(PR_a, X)$, respectively. Similarly, decryption of ciphertext C can be performed with a secret key, a public key, or a private key, denoted by $D(K_a, X)$, $D(PU_a, X)$, and $D(PR_a, X)$, respectively.

Table 9.1. Conventional and Public-Key Encryption

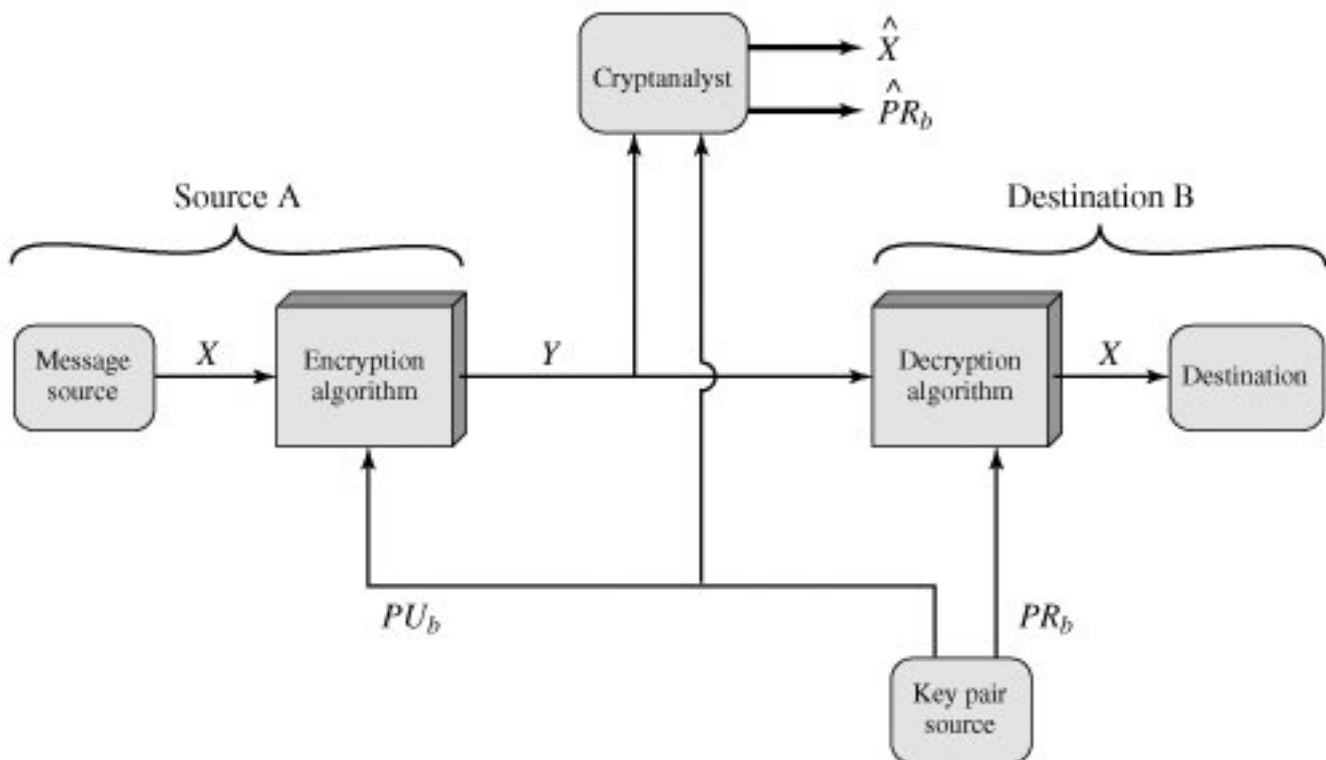
Conventional Encryption	Public-Key Encryption
Needed to Work:	Needed to Work:
<ol style="list-style-type: none"> 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. 	<ol style="list-style-type: none"> 1. One algorithm 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:	Needed for Security:

<ol style="list-style-type: none"> 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. 	<ol style="list-style-type: none"> 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.
--	---

Let us take a closer look at the essential elements of a public-key encryption scheme, using [Figure 9.2](#) (compare with [Figure 2.2](#)). There is some source A that 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 destination B. B generates a related pair of keys: a public key, PU_b , and a private key, PR_b . PU_b is known only to B, whereas PR_b is publicly available and therefore accessible by A.

Figure 9.2. Public-Key Cryptosystem: Secrecy

[\[View full size image\]](#)



With the message X and the encryption key PU_b as input, A forms the ciphertext $Y = [Y_1, Y_2, \dots, Y_M]$:

$$Y = E(PU_b, X)$$

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

$$X = D(PR_b, Y)$$

An adversary, observing Y and having access to PU_b but not having access to PR_b or X , must attempt to recover X and/or PR_b . It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms. If the adversary 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 adversary is interested in being able to read future messages as well, in which case an attempt is made to recover PR_b by generating an estimate \hat{PR}_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 9.2](#) provides confidentiality, [Figures 9.1b](#) and [9.3](#) show the use of public-key encryption to provide authentication:

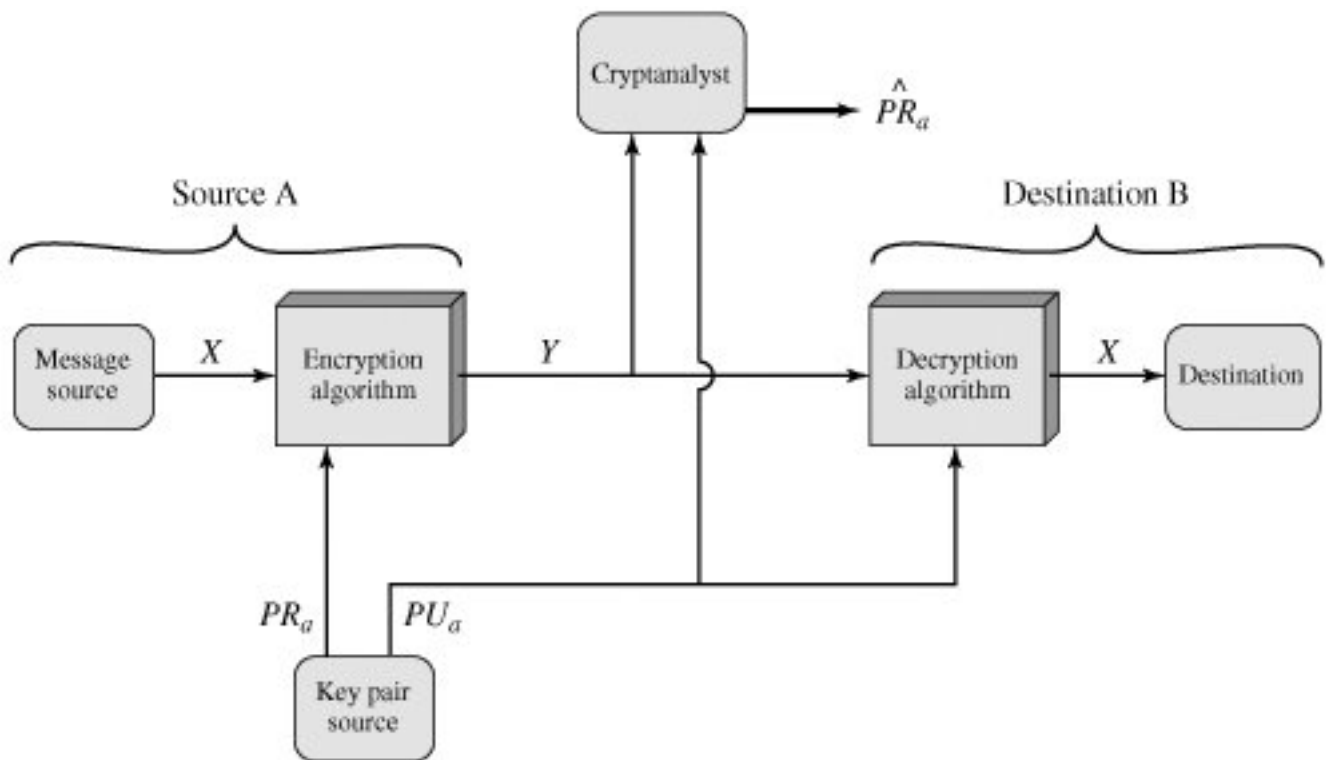
[Page 264]

$$Y = E(PR_a, X)$$

$$Y = E(PU_a, Y)$$

Figure 9.3. Public-Key Cryptosystem: Authentication

[\[View full size image\]](#)



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 13](#) examines this technique in detail.

It is important to emphasize that the encryption process depicted in [Figures 9.1b](#) and [9.3](#) 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 9.3](#), there is no protection of confidentiality because any observer can decrypt the message by using the sender's public key.

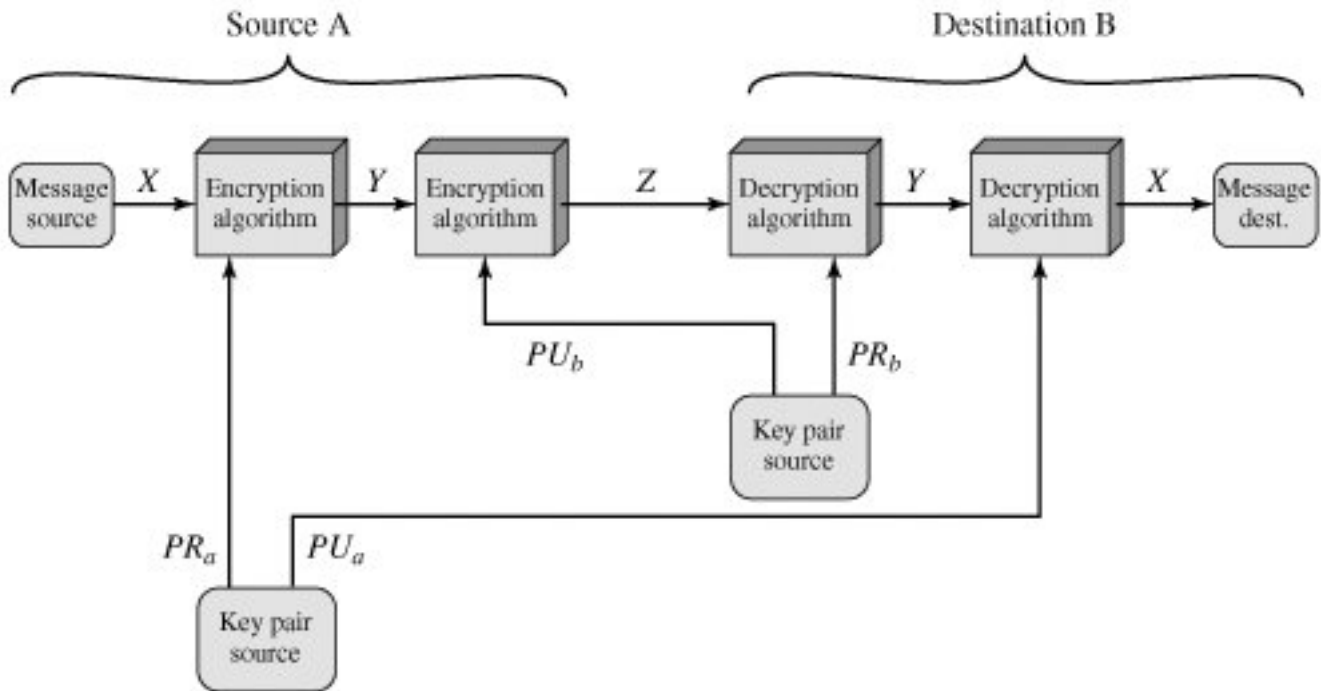
It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme ([Figure 9.4](#)):

$$Z = E(PU_b, E(PR_a, X))$$

$$X = D(PU_a, E(PR_b, Z))$$

Figure 9.4. Public-Key Cryptosystem: Authentication and Secrecy

[\[View full size image\]](#)



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 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 cryptographic 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.

[Page 266]

- **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 9.2](#) indicates the applications supported by the algorithms discussed in this book.

Table 9.2. Applications for Public-Key Cryptosystems

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Requirements for Public-Key Cryptography

The cryptosystem illustrated in [Figures 9.2](#) through [9.4](#) depends on a cryptographic algorithm based on two related keys. Diffie and Hellman postulated this system 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 PU_b , private key PR_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:

$$C = E(PU_b, M)$$

3.

It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

$$M = D(PR_b, C) = D[PR_b, E(PU_b, M)]$$

4.

It is computationally infeasible for an adversary, knowing the public key, PU_b , to determine the private key, PR_b .

5.

It is computationally infeasible for an adversary, knowing the public key, PU_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 two keys can be applied in either order:

$$M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$$

These are formidable requirements, as evidenced by the fact that only a few algorithms (RSA, elliptic curve cryptography, Diffie-Hellman, DSS) have received widespread acceptance in the several decades since the concept of public-key cryptography was proposed.

[Page 267]

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](#)^[3] 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:

^[3] 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 11](#)).

$Y = f(X)$ easy

$X = f^{-1}(Y)$ infeasible

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 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 inadequate 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 9B](#).

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

$Y = f_k(X)$ easy, if k and X are known

$X = f_k^{-1}(Y)$ easy, if k and Y are known

$X = f_k^{-1}(Y)$ infeasible, if Y is known but k is not known

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 symmetric 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.

[Page 268]

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 adversary could encrypt all possible 56-bit DES keys using the public key and could discover the encrypted key 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.

9.2. 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](#)].^[4] The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption.

^[4] 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.

The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and $n-1$ for some n . A typical size for n is 1024 bits, or 309 decimal digits. That is, n is less than 2^{1024} . We examine RSA in this section in some detail, beginning with an explanation of the algorithm. 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 i bits, where $2^i < n \leq 2^{i+1}$. 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 $PU = \{e, n\}$ and a private key of $PU = \{d, n\}$. 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 $M^{ed} \bmod n = M$ for all $M < n$.

2.

It is relatively easy to calculate $M^e \bmod n$ and C^d for all values of $M < n$.

3.

It is infeasible to determine d given e and n .

For now, we focus on the first requirement and consider the other questions later. We need to find a relationship of the form

$$M^{ed} \bmod n = M$$

The preceding relationship holds if e and d are multiplicative inverses modulo $\phi(n)$, where $\phi(n)$ is the Euler totient function. It is shown in [Chapter 8](#) that for p, q prime, $\phi(pq) = (p-1)(q-1)$. The relationship between e and d can be expressed as

Equation 9-1

$$ed \bmod \phi(n) = 1$$

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, $\gcd(\phi(n), d) = 1$. See [Appendix 9A](#) for a proof that [Equation \(9.1\)](#) satisfies the requirement for RSA.

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 $C = M^e \bmod n$ and transmits C . On receipt of this ciphertext, user A decrypts by calculating $M = C^d \bmod n$.

[Figure 9.5](#) summarizes the RSA algorithm. An example, from [\[SING99\]](#), is shown in [Figure 9.6](#). For this example, the keys were generated as follows:

1.

Select two prime numbers, $p = 17$ and $q = 11$.

2.

Calculate $n = pq = 17 \times 11 = 187$.

[Page 270]

3.

Calculate $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$.

4.

Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$ we choose $e = 7$.

5.

Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because $23 \times 7 = 161 = 10 \times 160 + 1$; d can be calculated using the extended Euclid's algorithm ([Chapter 4](#)).

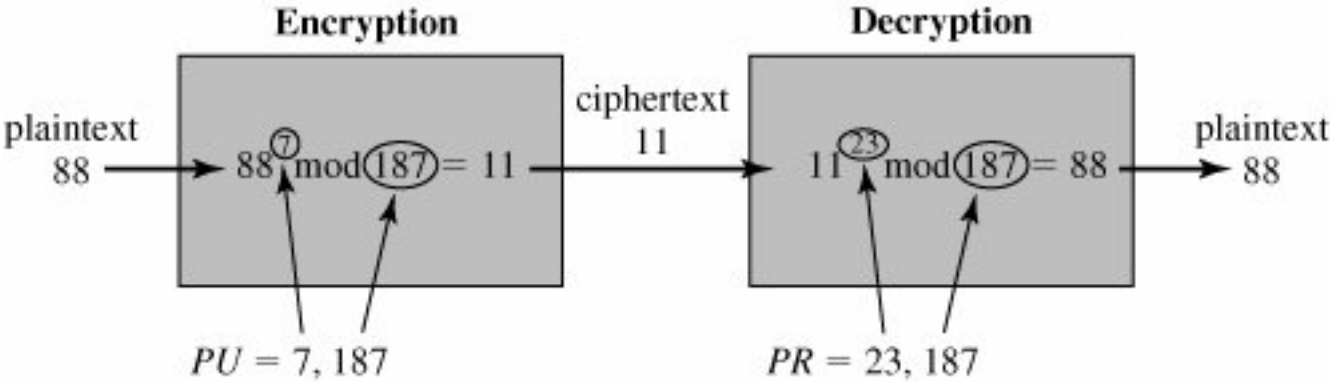
Figure 9.5. The RSA Algorithm

Key Generation	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\text{gcd}(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod n$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod n$

Figure 9.6. Example of RSA Algorithm



The resulting keys are public key $PU = \{7, 187\}$ and private key $PR = \{23, 187\}$. The example shows the

use of these keys for a plaintext input of $M = 88$. For encryption, we need to calculate $C = 88^7 \bmod 187$. Exploiting the properties of modular arithmetic, we can do this as follows:

[Page 271]

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$

$$88^1 \bmod 187 = 88$$

$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$

$$88^4 \bmod 187 = 59,969,536 \bmod 187 = 132$$

$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894,432 \bmod 187 = 11$$

For decryption, we calculate $M = 11^{23} \bmod 187$:

$$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14,641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214,358,881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79,720,245 \bmod 187 = 88$$

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: encryption/decryption and key generation. Let us look first at the process of encryption and decryption and then consider key generation.

Exponentiation in Modular Arithmetic

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, as the preceding example shows, 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} . As another example, suppose we wish to calculate $x^{11} \bmod n$ for some integers x and n . Observe that $x^{11} = x^{1+2+8} = (x)(x^2)(x^8)$. In this case we compute $x \bmod n$, $x^2 \bmod n$, $x^4 \bmod n$, and $x^8 \bmod n$ and then calculate $[(x \bmod n) \times (x^2 \bmod n) \times (x^8 \bmod n)] \bmod n$.

More generally, suppose we wish to find the value a^b with a and b positive integers. If we express b as a binary number $b_k b_{k-1} \dots b_0$ then we have

$$b = \sum_{b_i \neq 0} 2^i$$

Therefore,

$$a^b = a^{\left(\sum_{b_i \neq 0} 2^i\right)} = \prod_{b_i \neq 0} a^{(2^i)}$$

$$a^b \bmod n = \left[\prod_{b_i \neq 0} a^{(2^i)} \right] \bmod n = \left(\prod_{b_i \neq 0} [a^{(2^i)} \bmod n] \right) \bmod n$$

We can therefore develop the algorithm^[5] for computing $a^b \bmod n$, shown in [Figure 9.7](#). [Table 9.3](#) 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.

^[5] The algorithm has a long history; this particular pseudocode expression is from [\[CORM01\]](#).

Figure 9.7. Algorithm for Computing $a^b \bmod n$

Note: The integer b is expressed as a binary number $b_k b_{k-1} \dots b_0$


```

c ← 0; f ← 1
for i ← k downto 0
  do c ← 2 × c
     f ← (f × f) mod n
  if bi = 1
    then c ← c + 1
        f ← (f × a) mod n
return f

```

Table 9.3. Result of the Fast Modular Exponentiation Algorithm for $a^b \bmod n$, where $a = 7$, $b = 560 = 1000110000$, $n = 561$

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
f	7	49	157	526	160	241	298	166	67	1

Efficient Operation Using the Public Key

To speed up the operation of the RSA algorithm using the public key, a specific choice of e is usually made. The most common choice is 65537 ($2^{16} + 1$); two other popular choices are 3 and 17 . Each of these choices has only two 1 bits and so the number of multiplications required to perform exponentiation is minimized.

However, with a very small public key, such as $e = 3$, RSA becomes vulnerable to a simple attack. Suppose we have three different RSA users who all use the value $e = 3$ but have unique values of n , namely n_1, n_2, n_3 . If user A sends the same encrypted message M to all three users, then the three ciphertexts are $C_1 = M^3 \bmod n_1$; $C_2 = M^3 \bmod n_2$; $C_3 = M^3 \bmod n_3$. It is likely that n_1, n_2 , and n_3 are

pairwise relatively prime. Therefore, one can use the Chinese remainder theorem (CRT) to compute $M^3 \pmod{(n_1 n_2 n_3)}$. By the rules of the RSA algorithm, M is less than each of the n_i therefore $M^3 < n_1 n_2 n_3$.

Accordingly, the attacker need only compute the cube root of M^3 . This attack can be countered by adding a unique pseudorandom bit string as padding to each instance of M to be encrypted. This approach is discussed subsequently.

The reader may have noted that the definition of the RSA algorithm ([Figure 9.5](#)) requires that during key generation the user selects a value of e that is relatively prime to $\phi(n)$. Thus, for example, if a user has preselected $e = 65537$ and then generated primes p and q , it may turn out that $\gcd(\phi(n), e) \neq 1$. Thus, the user must reject any value of p or q that is not congruent to 1 (mod 65537).

Efficient Operation Using the Private Key

We cannot similarly choose a small constant value of d for efficient operation. A small value of d is vulnerable to a brute-force attack and to other forms of cryptanalysis [[WIEN90](#)]. However, there is a way to speed up computation using the CRT. We wish to compute the value $M = C^d \pmod n$. Let us define the following intermediate results:

$$V_p = C^d \pmod p \quad V_q = C^d \pmod q$$

Following the CRT, [Equation \(8.8\)](#), define the quantities:

$$X_p = q \times (q^{-1} \pmod p) \quad X_q = p \times (p^{-1} \pmod q)$$

The CRT then shows, using [Equation \(8.9\)](#), that

$$M = (V_p X_p + V_q X_q) \pmod n$$

Further, we can simplify the calculation of V_p and V_q using Fermat's theorem, which states that $a^{p-1} \equiv 1 \pmod p$ if p and a are relatively prime. Some thought should convince you that the following are valid:

$$V_p = C^d \pmod p = C^{d \pmod{(p-1)}} \pmod p \quad V_q = C^d \pmod q = C^{d \pmod{(q-1)}} \pmod q$$

The quantities $d \pmod{(p-1)}$ and $d \pmod{(q-1)}$ can be precalculated. The end result is that the calculation is approximately four times as fast as evaluating $M = C^d \pmod n$ directly [[BONE02](#)].

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 adversary, 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 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 8](#). 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, with a as a parameter. 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 (PU , PR) 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 $d \equiv e^{-1} \pmod{\phi(n)}$. 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 8](#). 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 8.2](#)).

The Security of RSA

Four 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 effort to factoring the product of two primes.
- **Timing attacks:** These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA 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 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 $d \equiv e^{-1} \pmod{\phi(n)}$.
- Determine $\phi(n)$ directly, without first determining p and q . Again, this enables determination of $d \equiv e^{-1} \pmod{\phi(n)}$.
- 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 [[GARD77](#)]. 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 [[LEUT94](#)]. 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 the RSA cipher with key sizes of 100, 110, 120, and so on, digits. The latest challenge to be met is the RSA-200 challenge with a key length of 200 decimal digits, or about 663 bits. [Table 9.4](#) 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 3×10^{13} instructions executed. A 1 GHz Pentium is about a 250-MIPS machine.

Table 9.4. Progress in Factorization

Number of Decimal Digits	Approximate Number of Bits	Date 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	1000	Generalized number field sieve
140	465	February 1999	2000	Generalized number field sieve
155	512	August 1999	8000	Generalized number field sieve
160	530	April 2003		Lattice sieve
174	576	December 2003		Lattice sieve
200	663	May 2005		Lattice sieve

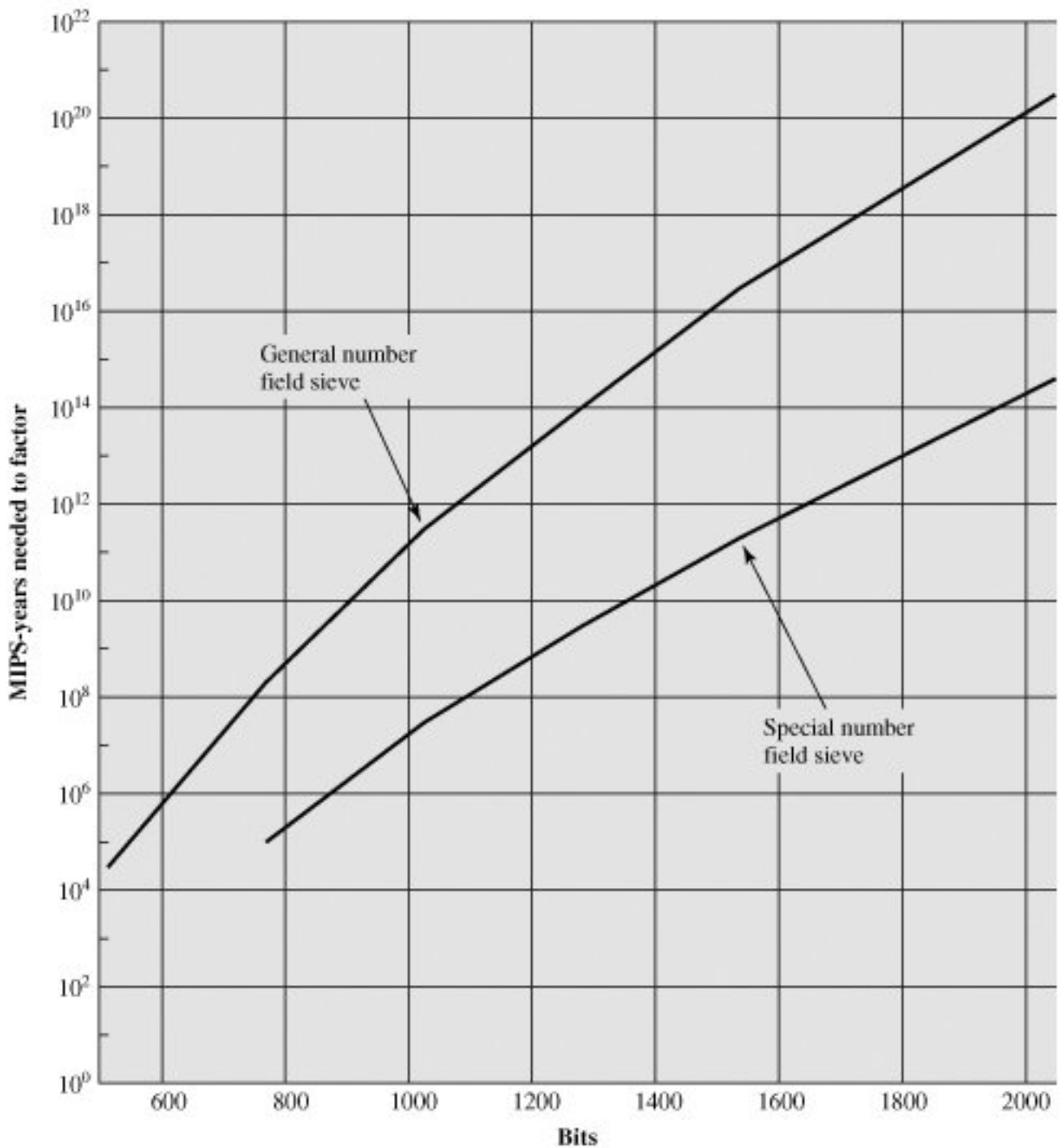
A striking fact about [Table 9.4](#) concerns the method used. Until the mid-1990s, factoring attacks were made using an approach known as the quadratic sieve. The attack on RSA-130 used a newer algorithm, the generalized number field sieve (GNFS), and was able to factor a larger number than RSA-129 at only 20% 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 9.8](#) 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.

Figure 9.8. MIPS-years Needed to Factor

(This item is displayed on page 277 in the print version)

[\[View full size image\]](#)



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, for a 1024-bit key (309 decimal digits), both p and q should be on the order of magnitude 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](#)].

[Page 277]

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](#), [KALI96b](#)]. 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 of [Figure 9.7](#), 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.

[Page 278]

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 don't 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 $C' = C(r^e) \bmod n$, 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 8](#) 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.

Chosen Ciphertext Attack and Optimal Asymmetric Encryption Padding

The basic RSA algorithm is vulnerable to a chosen ciphertext attack (CCA). CCA is defined as an attack in which adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key. Thus, the adversary could select a plaintext, encrypt it with the target's public key and then be able to get the plaintext back by having it decrypted with the private key. Clearly, this provides the adversary with no new information. Instead, the adversary exploits properties of RSA and selects blocks of data that, when processed using the target's private key, yield information needed for cryptanalysis.

A simple example of a CCA against RSA takes advantage of the following property of RSA:

Equation 9-2

$$E(PU, M_1) \times E(PU, M_2) = E(PU, [M_1 \times M_2])$$

We can decrypt $C = M^e$ using a CCA as follows.

1.

Compute $X = (C \times 2^e) \bmod n$.

2.

Submit X as a chosen ciphertext and receive back $Y = X^d \bmod n$.

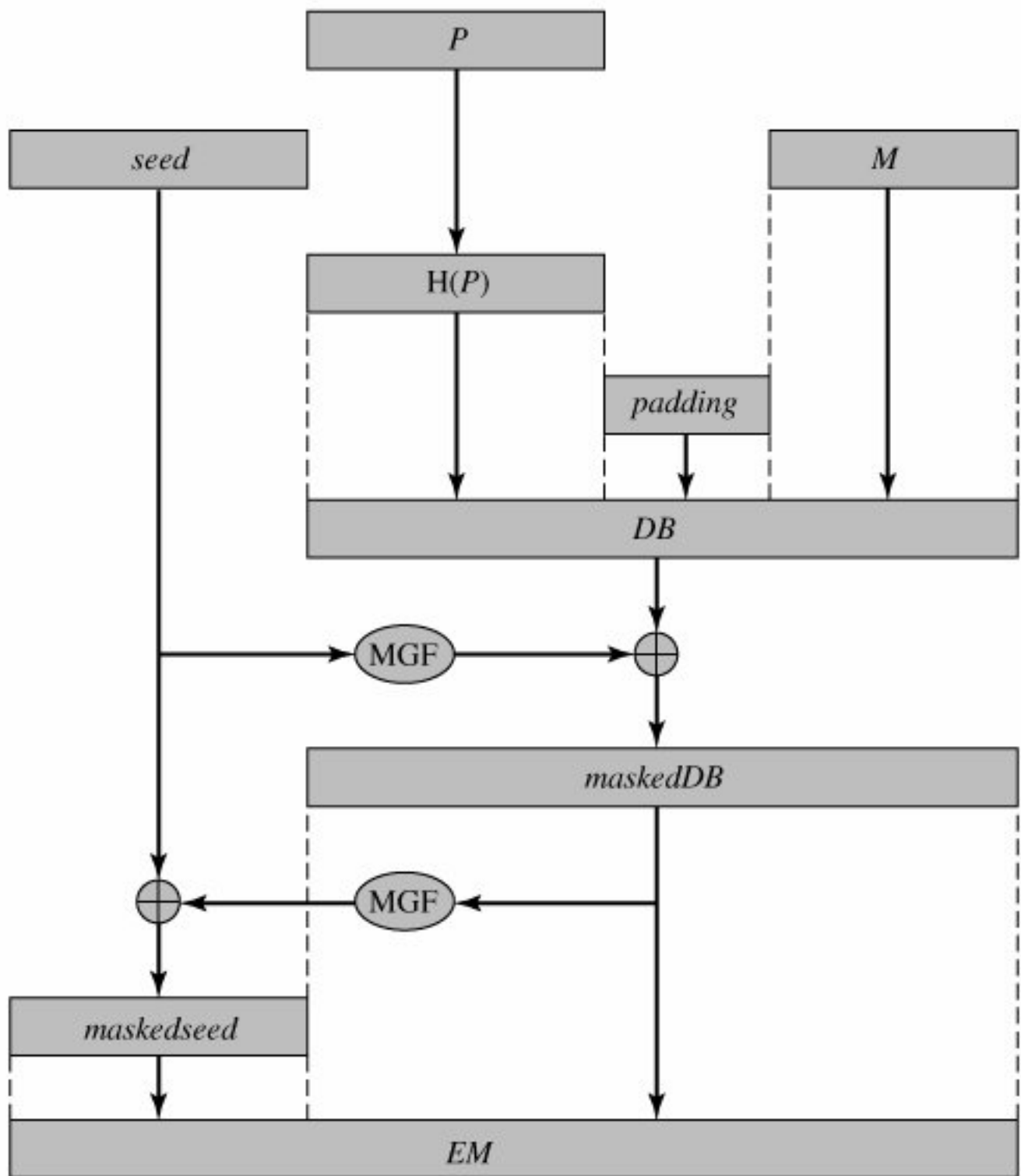
But now note the following:

$$\begin{aligned} X &= (C \bmod n) \times (2^e \bmod n) \\ &= (M^e \bmod n) \times (2^e \bmod n) \\ &= (2M)^e \bmod n \end{aligned}$$

Therefore, $Y = (2M) \bmod n$. From this, we can deduce M . To overcome this simple attack, practical RSA-based cryptosystems randomly pad the plaintext prior to encryption. This randomizes the ciphertext so that [Equation \(9.2\)](#) no longer holds. However, more sophisticated CCAs are possible and a simple padding with a random value has been shown to be insufficient to provide the desired security. To counter such attacks RSA Security Inc., a leading RSA vendor and former holder of the RSA patent, recommends modifying the plaintext using a procedure known as optimal asymmetric encryption padding (OAEP). A full discussion of the threats and OAEP are beyond our scope; see [\[POIN02\]](#) for an introduction and [\[BELL94a\]](#) for a thorough analysis. Here, we simply summarize the OAEP procedure.

[Figure 9.9](#) depicts OAEP encryption. As a first step the message M to be encrypted is padded. A set of optional parameters P is passed through a hash function H .^[6] The output is then padded with zeros to get the desired length in the overall data block (DB). Next, a random seed is generated and passed through another hash function, called the mask generating function (MGF). The resulting hash value is bit-by-bit XORed with DB to produce a maskedDB. The maskedDB is in turn passed through the MGF to form a hash that is XORed with the seed to produce the masked seed. The concatenation of the maskedseed and the maskedDB forms the encoded message EM. Note that the EM includes the padded message, masked by the seed, and the seed, masked by the maskedDB. The EM is then encrypted using RSA.

^[6] A hash function maps a variable-length data block or message into a fixed-length value called a hash code. Hash functions are discussed in depth in [Chapters 11](#) and [12](#).



P = encoding parameters
 M = message to be encoded
 H = hash function

DB = data block
 MGF = mask generating function
 EM = encoded message

9.3. Recommended Reading and Web Sites

The recommended treatments of encryption listed in [Chapter 3](#) cover public-key as well as symmetric 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. [\[CORM01\]](#) provides a concise but complete and readable summary of all of the algorithms relevant to the verification, computation, and cryptanalysis of RSA. [\[BONE99\]](#) discusses various cryptanalytic attacks on RSA. A more recent discussion is [\[SHAM03\]](#).

[BONE99](#) Boneh, D. "Twenty Years of Attacks on the RSA Cryptosystem." *Notices of the American Mathematical Society*, February 1999.

[CORM01](#) Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2001.

[DIFF88](#) Diffie, W. "The First Ten Years of Public-Key Cryptography." *Proceedings of the IEEE*, May 1988. Reprinted in [\[SIMM92\]](#).

[SHAM03](#) Shamir, A., and Tromer, E. "On the Cost of Factoring RSA-1024." *CryptoBytes*, Summer 2003. <http://www.rsasecurity.com/rsalabs>



Recommended Web Site

- **RSA Laboratories:** Extensive collection of technical material on RSA and other topics in cryptography

9.4. Key Terms, Review Questions, and Problems

Key Terms

[chosen ciphertext attack \(CCA\)](#)

[digital signature](#)

[key exchange](#)

[one-way function](#)

[optimal asymmetric encryption padding \(OAEP\)](#)

[private key](#)

[public key](#)

[public key cryptography](#)

[public key cryptosystems](#)

[public key encryption](#)

[RSA](#)

[time complexity](#)

[timing attack](#)

[trapdoor one-way function](#)

Review Questions

- 9.1 What are the principal elements of a public-key cryptosystem?
- 9.2 What are the roles of the public and private key?
- 9.3 What are three broad categories of applications of public-key cryptosystems?
- 9.4 What requirements must a public key cryptosystems fulfill to be a secure algorithm?
- 9.5 What is a one-way function?
- 9.6 What is a trapdoor one-way function?

9.7 Describe in general terms an efficient procedure for picking a prime number.

Problems

9.1 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 $1 < x_i, y_i, z_i \leq N$. 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 look-ups 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:

[Page 282]

$$f_3(f_2(f_1(k), p), k) = p \quad \text{for all } k, p \text{ with } 1 \leq k, p \leq N$$

In words,

1.

$M1$ takes an input k and produces an output x .

2.

$M2$ takes inputs x and p giving 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:

M1 =	5
	4
	2
	3
	1

M2 =	5	2	3	4	1
	4	2	5	1	3
	1	3	2	4	5
	3	1	4	2	5
	2	5	3	4	1

M3 =					

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.

9.2 Perform encryption and decryption using the RSA algorithm, as in [Figure 9.6](#), for the following:

1.

$$p = 3; q = 11, e = 7; M = 5$$

2.

$$p = 5; q = 11, e = 3; M = 9$$

3.

$$p = 7; q = 11, e = 17; M = 8$$

4.

$$p = 11; q = 13, e = 11; M = 7$$

5.

$$p = 17; q = 31, e = 7; M = 2. \text{ Hint: Decryption is not as hard as you think; use some finesse.}$$

9.3 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 ?

9.4 In an RSA system, the public key of a given user is $e = 31, n = 3599$. What is the private key of this user? *Hint*: First use trial and error to determine p and q ; then use the extended Euclidean algorithm to find the multiplicative inverse of 31 modulo $\phi(n)$.

- 9.5 In using the RSA algorithm, if a small number of repeated encodings give back the plaintext, what is the likely cause?
- 9.6 Suppose we have a set of blocks encoded with the RSA algorithm and we don't have the private key. Assume $n = pq$, e is the public key. Suppose also someone tells us they know one of the plaintext blocks has a common factor with n . Does this help us in any way?
- 9.7 In the RSA public-key encryption scheme, each user has a 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?
- 9.8 Suppose Bob uses the RSA cryptosystem with a very large modulus n for which the factorization cannot be found in a reasonable amount of time. Suppose Alice sends a message to Bob by representing each alphabetic character as an integer between 0 and 25 ($A \rightarrow 0, \dots, Z \rightarrow 25$), and then encrypting each number separately using RSA with large e and large n . Is this method secure? If not, describe the most efficient attack against this encryption method.

[Page 283]

- 9.9 Using a spreadsheet (such as Excel), or a calculator, perform the described below operations. Document results of all intermediate modular multiplications. Determine a number of modular multiplications per each major transformation (such as encryption, decryption, primality testing, etc.).
- a.
- Test all odd numbers in the range from 233 to 241 for primality using the Miller-Rabin test with base 2.
- b.
- Encrypt the message block $M = 2$ using RSA with the following parameters: $e = 23$ and $n = 233 \times 241$.
- c.
- Compute a private key (d, p, q) corresponding to the given above public key (e, n) .
- d.
- Perform the decryption of the obtained ciphertext using two different methods:
1.
without using the Chinese Remainder Theorem,
 2.
using the Chinese Remainder Theorem.

9.10 Assume that you generate an authenticated and encrypted message by first applying the RSA transformation determined by your private key, and then enciphering the message using recipient's public key (note that you do NOT use hash function before the first transformation). Will this scheme work correctly [i.e., give the possibility to reconstruct the original message at the recipient's side, for all possible relations between the sender's modulus n_S and the recipient's modulus n_R ($n_S > n_R$, $n_S < n_R$, $n_S = n_R$)]? Explain your answer. In case your answer is "no," how would you correct this scheme?

9.11 "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' face lost its sleepy look.

"Yes, Holmes. The idea is quite 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, 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, on 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."

Show how this can be done.

9.12 Show how RSA can be represented by matrices $M1$, $M2$, and $M3$ of [Problem 9.1](#).

9.13 Consider the following scheme:

1.

Pick an odd number, E .

2.

Pick two prime numbers, P and Q , where $(P-1)(Q-1)$ 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.

9.14 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 $M \equiv C^{Q'} \pmod{P}$.

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 M_1 , M_2 , and M_3 of [Problem 9.1](#).

9.15 "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 the following block: $(A, E(PU_b, [M, A]), B)$.

2.

B acknowledges receipt by

sending to A the following block: $(B, E(PU_a, [M, B]), A)$.

"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 the block: $(A, E(PU_b, M), B)$.

2.

B acknowledges receipt by sending to A the block: $(B, E(PU_a, M), A)$.

"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:"

Complete the description.

- 9.16** Use the fast exponentiation algorithm of [Figure 9.7](#) to determine $5^{596} \bmod 1234$. Show the steps involved in the computation.
- 9.17** Here is another realization of the fast exponentiation algorithm. Demonstrate that it is equivalent to the one in [Figure 9.7](#).

[Page 285]

1.
 $f \leftarrow 1; T \leftarrow a; E \leftarrow b$
2.
if odd(e) **then** $f \leftarrow d \times T$
3.
 $E \leftarrow \lfloor E/2 \rfloor$
4.
 $T \leftarrow T \times T$
5.
if $E > 0$ **then goto** 2
6.
output f

- 9.18** The problem illustrates a simple application of the chosen ciphertext attack. Bob intercepts a ciphertext C intended for Alice and encrypted with Alice's public key e . Bob wants to obtain the original message $M = C^d \bmod n$. Bob chooses a random value r less than n and computes

$$Z = r^e \bmod n$$

$$X = ZC \bmod n$$

$$t = r^{-1} \bmod n$$

Next, Bob gets Alice to authenticate (sign) X with her private key (as in [Figure 9.3](#)), thereby decrypting X . Alice returns $Y = X^d \bmod n$. Show how Bob can use the information now available to him to determine M .

- 9.19** Show the OAEP decoding operation, used for decryption, that corresponds to the encoding operation of [Figure 9.9](#).

9.20 Improve on algorithm P1 in [Appendix 9B](#).

a.

Develop an algorithm that requires $2n$ multiplications and $n + 1$ additions. *Hint:* $x^{j+1} = x^j \times x$.

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 $(n - 1)$.

Note: The remaining problems concern the knapsack public-key algorithm described in Appendix F.

9.21 What items are in the knapsack in Figure F.1?

9.22 Perform encryption and decryption using the knapsack algorithm for the following:

a.

$$\mathbf{a}' = (1, 3, 5, 10); w = 7; m = 20; \mathbf{x} = 1101$$

b.

$$\mathbf{a}' = (1, 3, 5, 11, 23, 46, 136, 263); w = 203; m = 491; \mathbf{x} = 11101000$$

c.

$$\mathbf{a}' = (2, 3, 6, 12, 25); w = 46; m = 53; \mathbf{x} = 11101$$

d.

$$\mathbf{a}' = (15, 92, 108, 279, 563, 1172, 2243, 4468); w = 2393; m = 9291; \mathbf{x} = 10110001$$

9.23

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

Why is it a requirement that

Appendix 9A Proof of the RSA Algorithm

The basic elements of the RSA algorithm can be summarized as follows. Given two prime numbers p and q , with $n = pq$ and a message block $M < n$, two integers e and d are chosen such that

$$M^{ed} \bmod n = M$$

We state in [Section 9.2](#), that the preceding relationship holds if e and d are multiplicative inverses modulo $\phi(n)$, where $\phi(n)$ is the Euler totient function. It is shown in [Chapter 8](#) that for p, q prime, $\phi(n) = (p-1)(q-1)$. The relationship between e and d can be expressed as

$$ed \bmod \phi(n) = 1$$

Another way to state this is that there is an integer k such that $ed = k\phi(n) + 1$. Thus, we must show that

Equation 9-3

$$M^{k\phi(n)+1} \bmod n = M^{k(p-1)(q-1)+1} \bmod n = M$$

Basic Results

Before proving [Equation \(9.3\)](#), we summarize some basic results. In [Chapter 4](#), we showed that a property of modular arithmetic is the following:

$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

From this, it should be easy to see that if we have $x \bmod n = 1$ then $x^2 \bmod n = 1$ and for any integer y , we have $x^y \bmod n = 1$. Similarly, if we have $x \bmod n = 0$, for any integer y , we have $x^y \bmod n = 0$.

Another property of modular arithmetic is

$$[(a \bmod n) (b \bmod n)] \bmod n = (a b) \bmod n$$

The other result we need is Euler's theorem, which was developed in [Chapter 8](#). If integers a and n are relatively prime, then $a^{\phi(n)} \bmod n = 1$.

Proof

First we show that $M^{k(p-1)(q-1)+1} \bmod p = M \bmod p$. There are two cases to consider.

Case 1: M and p are not relatively prime; that is, p divides M . In this case $M \bmod p = 0$ and therefore $M^{k(p_1)(q_1)+1} \bmod p = 0$. Thus, $M^{k(p_1)(q_1)+1} \bmod p = M \bmod p$.

Case 2: If M and p are relatively prime, by Euler's theorem, $M^{\phi(p)} \bmod p = 1$. We proceed as follows:

$$\begin{aligned}
 M^{k(p_1)(q_1)+1} \bmod p &= [(M)M^{k(p_1)(q_1)}] \bmod p \\
 &= [(M)(M^{\phi(p)})^{k(q_1)}] \bmod p \\
 &= [(M)(M^{\phi(p)})^{k(q_1)}] \bmod p \\
 &= (M \bmod p) \times [(M^{\phi(p)} \bmod p)^{k(q_1)}] \\
 &= (M \bmod p) \times (1)^{k(q_1)} && \text{(by Euler's theorem)} \\
 &= M \bmod p
 \end{aligned}$$

We now observe that

$$[M^{k(p_1)(q_1)+1} M] \bmod p [M^{k(p_1)(q_1)+1} \bmod p] [M \bmod p] = 0$$

Thus, p divides $[M^{k(p_1)(q_1)+1} M]$. By the same reasoning, we can show that q divides $[M^{k(p_1)(q_1)+1} M]$. Because p and q are distinct primes, there must exist an integer r that satisfies

$$[M^{k(p_1)(q_1)+1} M] = (pq)r = nr$$

Therefore, p divides $[M^{k(p_1)(q_1)+1} M]$, and so $M^{k(p_1)(q_1)+1} \bmod n = M^{k(p_1)(q_1)+1} \bmod n = M$.



Appendix 9B 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: if and only if there exist two numbers a and M such that

Equation 9-4

$$|f(n)| \leq a \times |g(n)|, \quad n \geq M$$

An example helps clarify the use of this notation. Suppose we wish to evaluate a general polynomial of the form

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

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 x S[i];
      S[i] := a[i] x 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 \(9.4\)](#), 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 [i.e., $(k + 2)(k + 1)/2 < k^2$]. Then, with $n = k + 1$.

[Page 288]

$$\begin{aligned}
 \frac{(n + 2)(n + 1)}{2} &= \frac{(k + 3)(k + 2)}{2} \\
 &= \frac{(k + 2)(k + 1)}{2} + k + 2 \\
 &\leq k^2 + k + 2 \\
 &\leq k^2 + 2k + 1 = (k + 1)^2 = n^2
 \end{aligned}$$

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, two of the best accounts are in [GRAH94] and [KNUT97].

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 operations per unit time. Table 9.5 shows 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 9.5. 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}
2^2	39	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 the Advanced Encryption Standard (AES), for example, the number of possible keys is 2^{128} and the length of the key is 128 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 10. Key Management; Other Public-Key Cryptosystems

10.1 Key Management

[Distribution of Public Keys](#)

[Distribution of Secret Keys Using Public-Key Cryptography](#)

10.2 Diffie-Hellman Key Exchange

[The Algorithm](#)

[Key Exchange Protocols](#)

[Man-in-the-Middle Attack](#)

10.3 Elliptic Curve Arithmetic

[Abelian Groups](#)

[Elliptic Curves over Real Numbers](#)

[Elliptic Curves over \$Z_p\$](#)

[Elliptic Curves over \$GF\(2^m\)\$](#)

10.4 Elliptic Curve Cryptography

[Analog of Diffie-Hellman Key Exchange](#)

[Elliptic Curve Encryption/Decryption](#)

[Security of Elliptic Curve Cryptography](#)

10.5 Recommended Reading and Web Sites

10.6 Key Terms, Review Questions, and Problems

No Singhalese, whether man or woman, would venture out of the house without a bunch of keys in his hand, for without such a talisman he would fear that some devil might take advantage of his weak state to slip into his body.

The Golden Bough, Sir James George Frazer

Key Points

- Public-key encryption schemes are secure only if the authenticity of the public key is assured. A public-key certificate scheme provides the necessary security.
- A simple public-key algorithm is Diffie-Hellman key exchange. This protocol enables two users to establish a secret key using a public-key scheme based on discrete logarithms. The protocol is secure only if the authenticity of the two participants can be established.
- Elliptic curve arithmetic can be used to develop a variety of elliptic curve cryptography (ECC) schemes, including key exchange, encryption, and digital signature.
- For purposes of ECC, elliptic curve arithmetic involves the use of an elliptic curve equation defined over a finite field. The coefficients and variables in the equation are elements of a finite field. Schemes using Z_p and $GF(2^m)$ have been developed.

This chapter continues our overview of public-key encryption. 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.

10.1. Key Management

In [Chapter 7](#), 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 cryptography 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 10.1](#)). For example, because of the growing popularity of PGP (pretty good privacy, discussed in [Chapter 15](#)), 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.

Figure 10.1. Uncontrolled Public-Key Distribution

[\[View full size image\]](#)



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 9.3](#)).

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 10.2](#)). 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.

[Page 292]

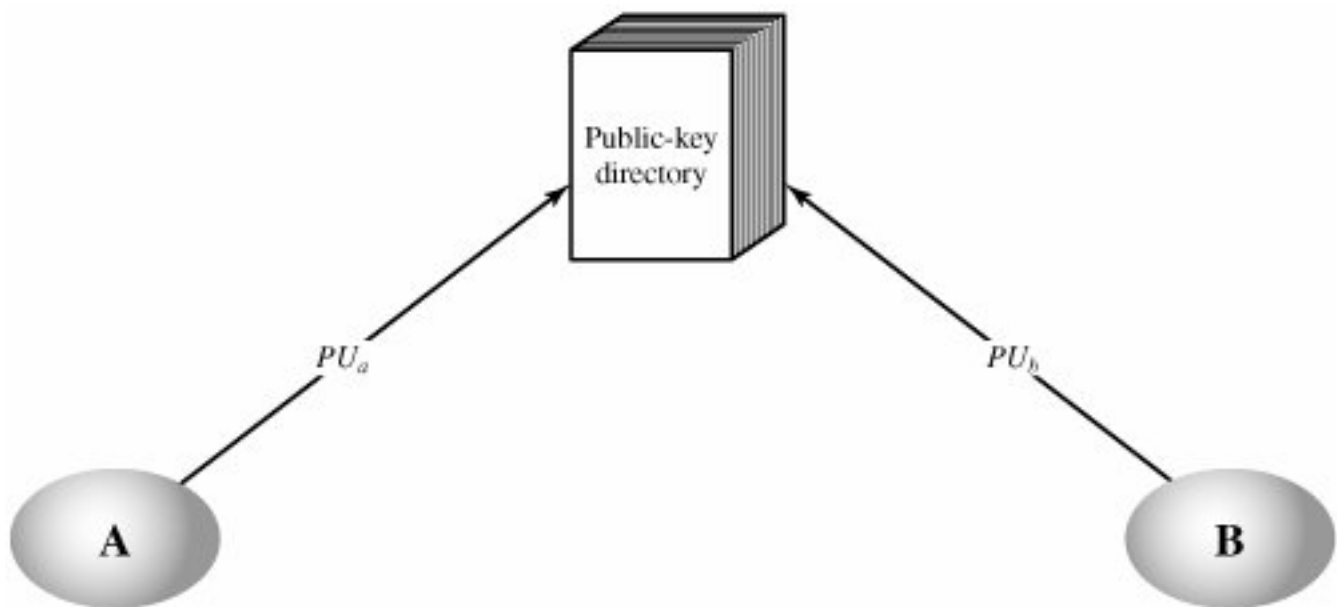
3.

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.

Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

Figure 10.2. Public-Key Publication



This scheme is clearly more secure than individual public announcements but still has vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary 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 adversary 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 10.3](#), 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 10.3](#)) 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, PR_{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:

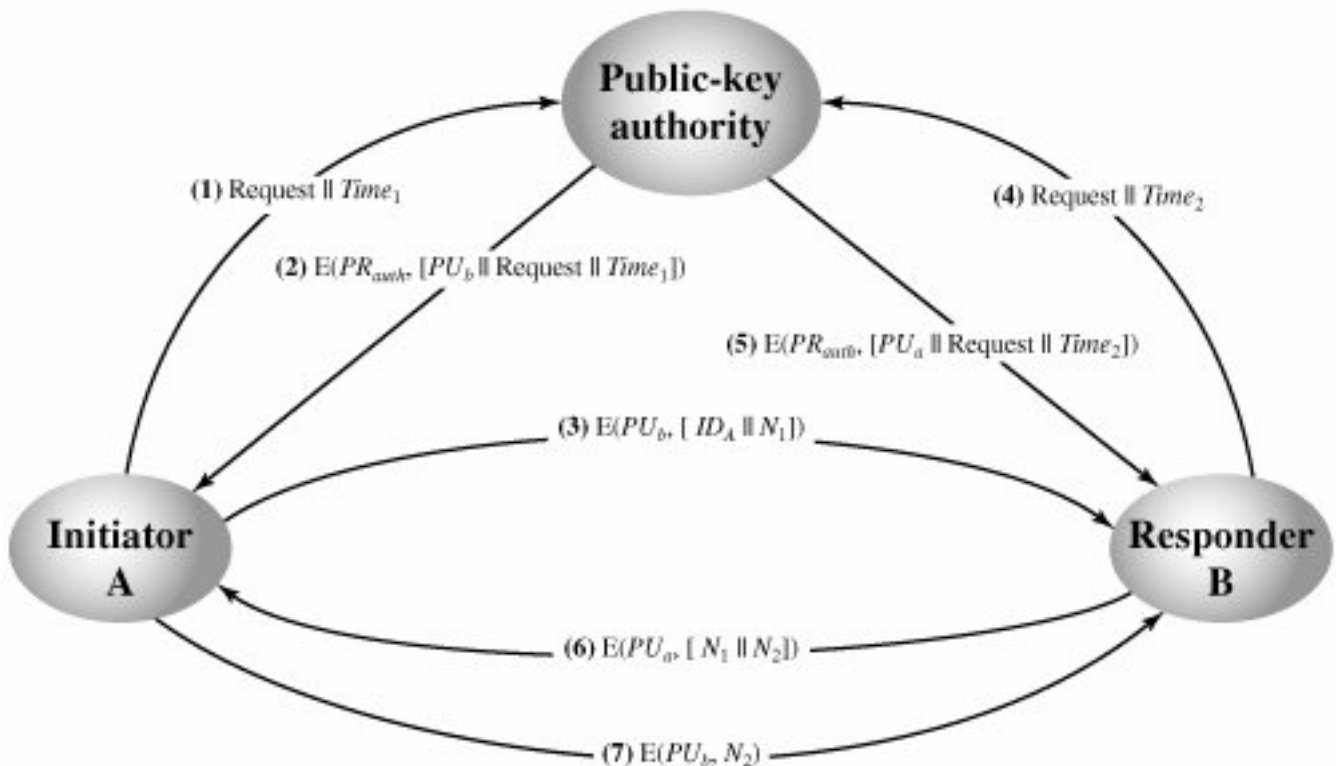
- B's public key, PU_b which A can use to encrypt messages destined for B
- The original request, to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority

- 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 also 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. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.
5. 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 PU_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 10.3. Public-Key Distribution Scenario

[\[View full size image\]](#)



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 10.3](#) 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 [[KOHNN78](#)], 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. In essence, a certificate consists of a public key plus an identifier of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a certificate authority, such as a government agency or a financial institution, that is trusted by the user community. A user can present his or her public key to the authority in a secure manner, and obtain a certificate. The user can then publish the certificate. Anyone needed this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. A participant can also convey 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 [[KOHNN78](#)]. Denning [[DENNN83](#)] added the following additional requirement:

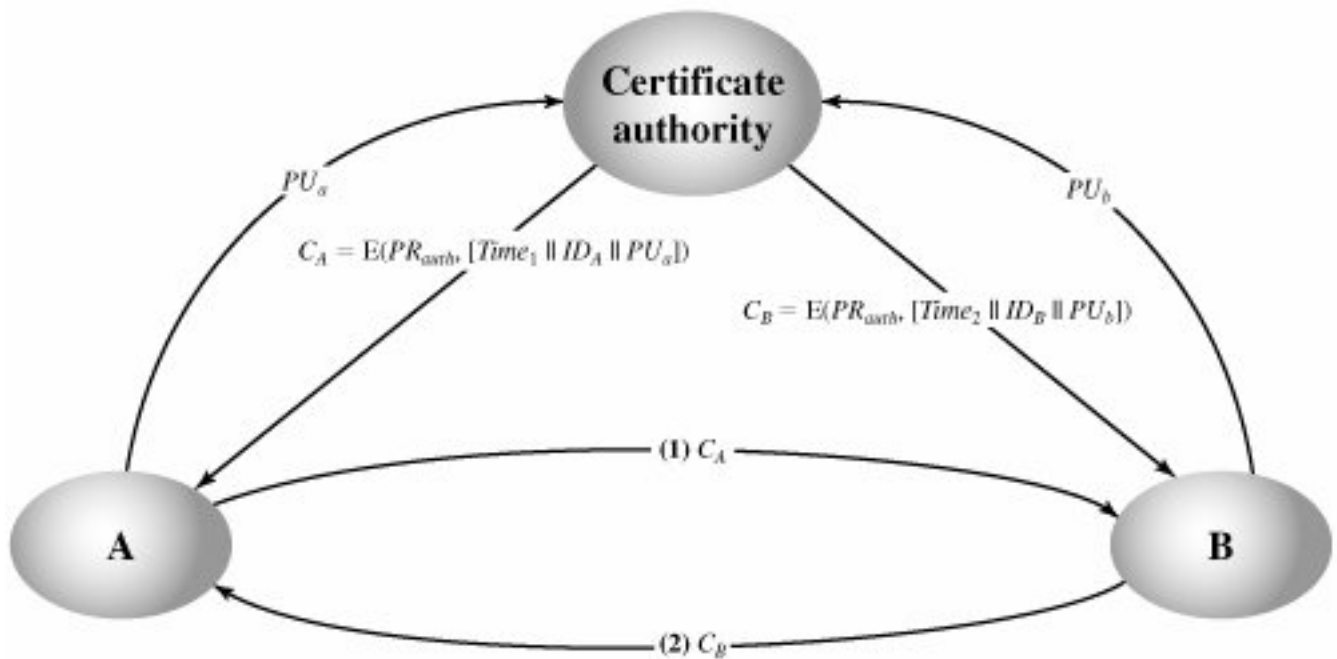
4.

Any participant can verify the currency of the certificate.

A certificate scheme is illustrated in [Figure 10.4](#). Each participant applies to the certificate authority, supplying a public key and requesting a certificate.

Figure 10.4. Exchange of Public-Key Certificates

[\[View full size image\]](#)



Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form

$$C_A = E(PR_{auth}, [T || ID_A || PU_a])$$

where PR_{auth} is the private key used by the authority and T is a timestamp. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

$$D(PU_{auth}, C_A) = D(PU_{auth}, E(PR_{auth}, [T || ID_A || PU_a])) = (T || ID_A || PU_a)$$

The recipient uses the authority's public key, PU_{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 PU_a provide the recipient with the name and public key of the certificate's holder. The timestamp T validates the currency of the certificate. The timestamp counters the following scenario. A's private key is learned by an adversary. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary 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 timestamp serves as something like an expiration date. If a certificate is sufficiently old, it is assumed to be expired.

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME, all of which are discussed in Part Two. X.509 is examined in detail in [Chapter 14](#).

Distribution of Secret Keys Using Public-Key Cryptography

Once public keys have been distributed or have become accessible, secure communication that thwarts eavesdropping ([Figure 9.2](#)), tampering ([Figure 9.3](#)), or both ([Figure 9.4](#)) 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 provides for the distribution of secret keys to be used for conventional encryption.

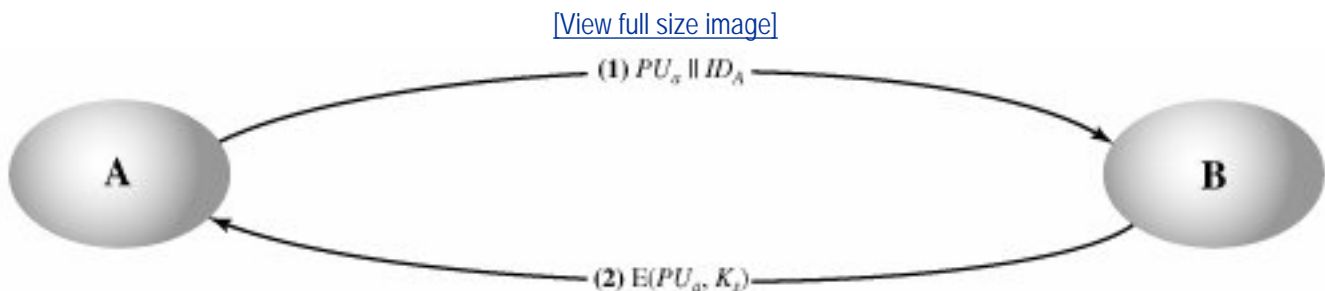
Simple Secret Key Distribution

An extremely simple scheme was put forward by Merkle [[MERK79](#)], as illustrated in [Figure 10.5](#). If A wishes to communicate with B, the following procedure is employed:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message to B consisting of PU_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 $D(PR_a, E(PU_a, K_s))$ 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 PU_a and PR_a and B discards PU_a .

[Page 296]

Figure 10.5. Simple Use of Public-Key Encryption to Establish a Session Key



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.

The protocol depicted in [Figure 10.5](#) is insecure against an adversary who can intercept messages and

then either relay the intercepted message or substitute another message (see [Figure 1.4c](#)). Such an attack is known as a **man-in-the-middle attack** [[RIVE84](#)]. In this case, if an adversary, 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 $\{PU_a, PR_a\}$ and transmits a message intended for B consisting of PU_a and an identifier of A, ID_A .

2.

E intercepts the message, creates its own public/private key pair $\{PU_e, PR_e\}$ and transmits $PU_e || ID_A$ to B.

3.

B generates a secret key, K_s , and transmits $E(PU_e, K_s)$.

4.

E intercepts the message, and learns K_s by computing $D(PR_e, E(PU_e, K_s))$.

5.

E transmits $E(PU_a, K_s)$ 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 10.6](#), 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:

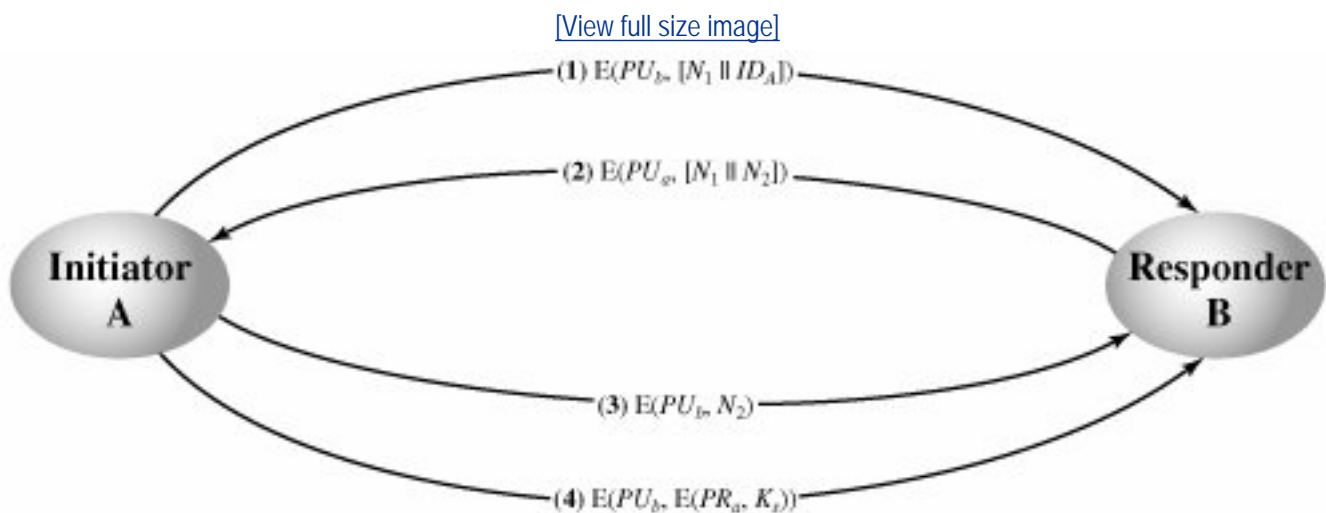
1. A uses B's public key to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.

- B sends a message to A encrypted with PU_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 (1), the presence of N_1 in message (2) assures A that the correspondent is B.

[Page 297]

- A returns N_2 encrypted using B's public key, to assure B that its correspondent is A.
- A selects a secret key K_s and sends $M = E(PU_b, E(PR_a, K_s))$ 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.
- B computes $D(PU_a, D(PR_b, M))$ to recover the secret key.

Figure 10.6. Public-Key Distribution of Secret Keys



Notice that the first three steps of this scheme are the same as the last three steps of [Figure 10.3](#). 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.



10.2. 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.

[1] A number of commercial products employ this key exchange technique.

[1] 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 [ELLI99] for a discussion.

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.

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 modulo p 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 , we can find a unique exponent i such that

$$b \equiv a^i \pmod{p} \text{ where } 0 \leq i \leq (p-1)$$

The exponent i is referred to as the discrete logarithm of b for the base a , mod p . We express this value as $\text{dlog}_{a,p}(b)$. See [Chapter 8](#) for an extended discussion of discrete logarithms.

The Algorithm

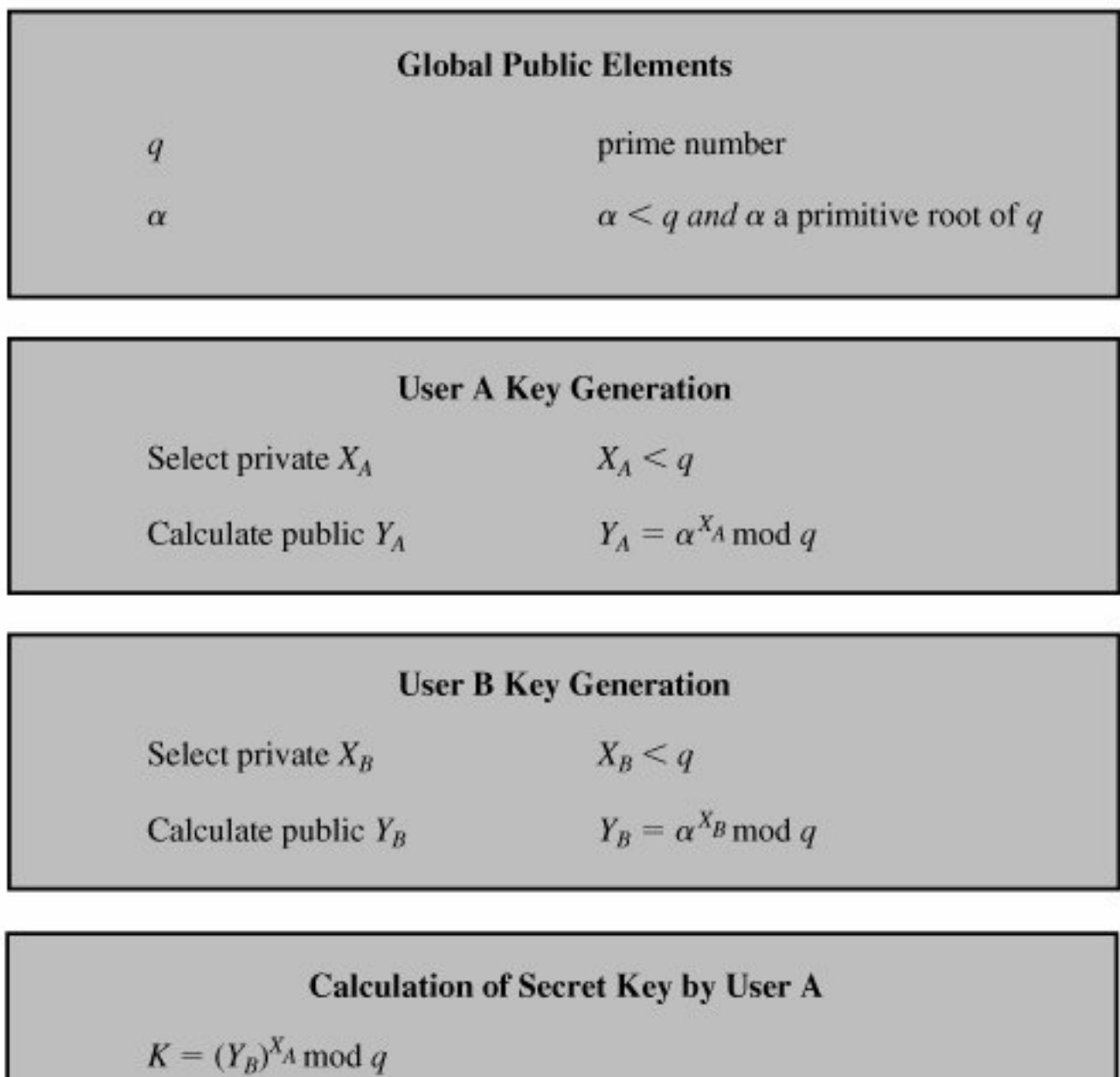
[Figure 10.7](#) summarizes the Diffie-Hellman key exchange algorithm. For this scheme, there are two publicly known numbers: a prime number q and an integer that is a primitive root of q . Suppose the users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{X_A} \bmod q$. 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:

$$K = (Y_B)^{X_A} \bmod q$$

$$\begin{aligned}
&= (\alpha^{X^B} \bmod q)^{X^A} \bmod q \\
&= (\alpha^{X^B})^{X^A} \bmod q && \text{by the rules of modular arithmetic} \\
&= (\alpha^{X^B X^A} \bmod q) \\
&= (\alpha^{X^A})^{X^B} \bmod q \\
&= (\alpha^{X^A} \bmod q) \\
&= (\alpha^{X^A} \bmod q)^{X^B} \bmod q \\
&= (Y_A)^{X^B} \bmod q
\end{aligned}$$

[Page 299]

Figure 10.7. The Diffie-Hellman Key Exchange Algorithm



$$K = (Y_B)^{X_A} \bmod q$$

Calculation of Secret Key by User B

$$K = (Y_A)^{X_B} \bmod q$$

The result is that the two sides have exchanged a secret value. Furthermore, because X_A and X_B are private, an adversary only has the following ingredients to work with: q , α , Y_A , and Y_B . Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute

$$X_B = \text{dlog}_{\alpha, q}(Y_B)$$

The adversary 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. Key exchange is based on the use of the prime number $q = 353$ and a primitive root of 353, in this case $\alpha = 3$. A and B select secret keys $X_A = 97$ and $X_B = 233$, respectively. Each computes its public key:

$$\text{A computes } Y_A = 3^{97} \bmod 353 = 40.$$

$$\text{B computes } Y_B = 3^{233} \bmod 353 = 248.$$

[Page 300]

After they exchange public keys, each can compute the common secret key:

$$\text{A computes } K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160.$$

$$\text{B computes } K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160.$$

We assume an attacker would have available the following information:

$$q = 353; \alpha = 3; Y_A = 40; Y_B = 248$$

In this simple example, it would be possible by brute force to determine the secret key 160. In particular, an attacker E can determine the common key by discovering a solution to the equation $3^a \bmod 353 = 40$ or the equation $3^b \bmod 353 = 248$. The brute-force approach is to calculate powers of 3 modulo 353, stopping when the result equals either 40 or 248. The desired answer is reached with the exponent value of 97, which provides $3^{97} \bmod 353 = 40$.

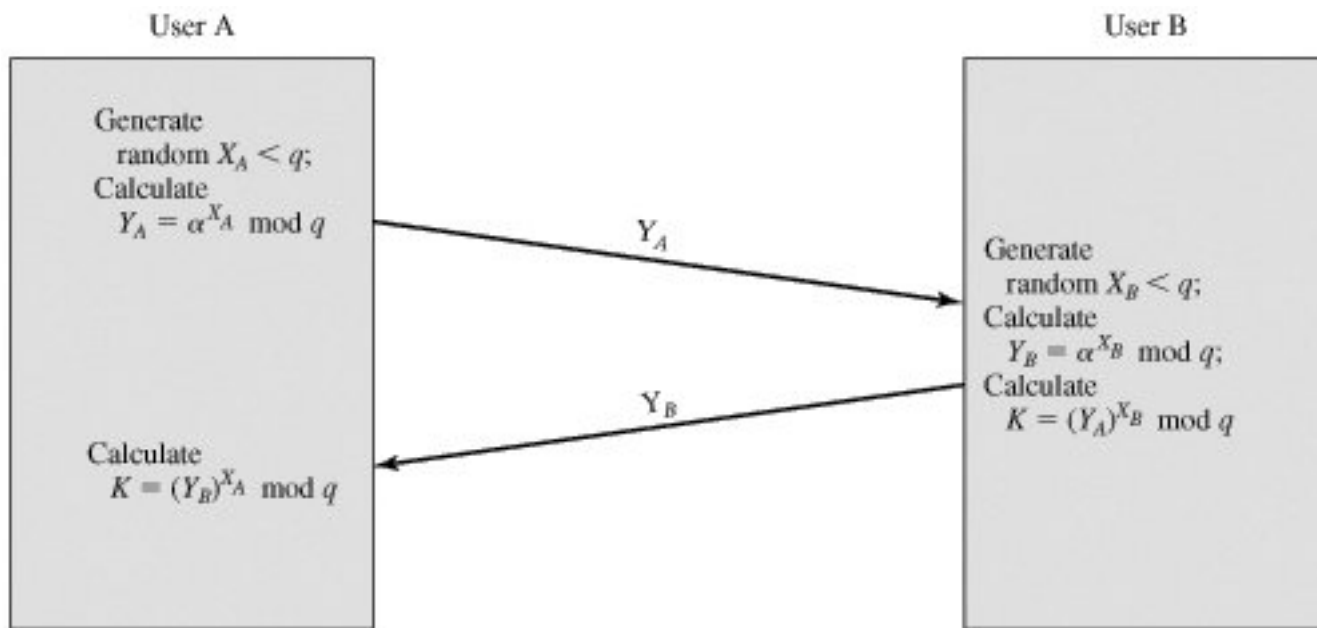
With larger numbers, the problem becomes impractical.

Key Exchange Protocols

Figure 10.8 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 α would need to be known ahead of time. Alternatively, user A could pick values for q and α and include those in the first message.

Figure 10.8. Diffie-Hellman Key Exchange

[\[View full size image\]](#)



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_i (for user i) and calculate a public value Y_i .

These public values, together with global public values for q and α , are stored in some central directory. At any time, user j can access user i '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 communication provides both confidentiality and a degree of authentication. Because only i and j can determine the key, no other user can read the message (confidentiality). Recipient i knows that only user j could have created a message using this key (authentication). However, the technique does not protect against replay attacks.

Man-in-the-Middle Attack

The protocol depicted in [Figure 10.8](#) is insecure against a man-in-the-middle attack. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows:

1.

Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2} .

2.

Alice transmits Y_A to Bob.

3.

Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.

4.

Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_B} \bmod q$.

5.

Bob transmits X_B to Alice.

6.

Darth intercepts X_B and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)^{X_{D1}} \bmod q$.

7.

Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key $K1$ and Alice and Darth share secret key $K2$. All future communication between Bob and Alice is compromised in the following way:

1.

Alice sends an encrypted message M : $E(K_2, M)$.

2.

Darth intercepts the encrypted message and decrypts it, to recover M .

3.

Darth sends Bob $E(K_1, M)$ or $E(K_1, M')$, where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates; these topics are explored in [Chapters 13](#) and [14](#).

◀ PREV

NEXT ▶

10.3. Elliptic Curve Arithmetic

Most of the products and standards that use public-key cryptography for encryption and digital signatures use RSA. As we have seen, the key 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 key 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. Accordingly, 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 and the next give some background on elliptic curves and ECC. We begin with a brief review of the concept of abelian group. Next, we examine the concept of elliptic curves defined over the real numbers. This is followed by a look at elliptic curves defined over finite fields. Finally, we are able to examine elliptic curve ciphers.

The reader may wish to review the material on finite fields in [Chapter 4](#) before proceeding.

Abelian Groups

Recall from [Chapter 4](#), that an abelian **group** G , sometimes denoted by $\{G, \bullet\}$, is a set of elements with a binary operation, denoted by \bullet , that associates to each ordered pair (a, b) of elements in G an element $(a \bullet b)$ in G , such that the following axioms are obeyed: ^[2]

^[2] The operator \bullet is generic and can refer to addition, multiplication, or some other mathematical operation.

- (A1) Closure: If a and b belong to G , then $a \bullet b$ is also in G .
- (A2) Associative: $a \bullet (b \bullet c) = (a \bullet b) \bullet c$ for all a, b, c in G .
- (A3) Identity element: There is an element e in G such that $a \bullet e = e \bullet a = a$ for all a in G .
- (A4) Inverse element: For each a in G there is an element a' in G such that $a \bullet a' = a' \bullet a = e$.
- (A5) Commutative: $a \bullet b = b \bullet a$ for all a, b in G .

A number of public-key ciphers are based on the use of an abelian group. For example, Diffie-Hellman

key exchange involves multiplying pairs of nonzero integers modulo a prime number q . Keys are generated by exponentiation over the group, with exponentiation defined as repeated multiplication. For

$$q = \underbrace{(a \times a \times \dots \times a)}_{k \text{ times}}$$

example, $a^k \text{ mod } q =$ $k \text{ times}$ $\text{ mod } q$. To attack Diffie-Hellman, the attacker must determine k given a and a^k ; this is the discrete log problem.

For elliptic curve cryptography, an operation over elliptic curves, called addition, is used. Multiplication is

$$a \times k = \underbrace{(a + a + \dots + a)}_{k \text{ times}}$$

defined by repeated addition. For example, performed over an elliptic

curve. Cryptanalysis involves determining k given a and $(a \times k)$.

An elliptic curve is defined by an equation in two variables, with coefficients. For cryptography, the variables and coefficients are restricted to elements in a finite field, which results in the definition of a finite abelian group. Before looking at this, we first look at elliptic curves in which the variables and coefficients are real numbers. This case is perhaps easier to visualize.

Elliptic Curves over Real Numbers

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 and x and y take on values in the real numbers. ^[3] For our purpose, it is sufficient to limit ourselves to equations of the form

^[3] Note that x and y are true variables, which take on values. This is in contrast to our discussion of polynomial rings and fields in [Chapter 4](#), where x was treated as an indeterminate.

Equation 10-1

$$y^2 = x^3 + ax + b$$

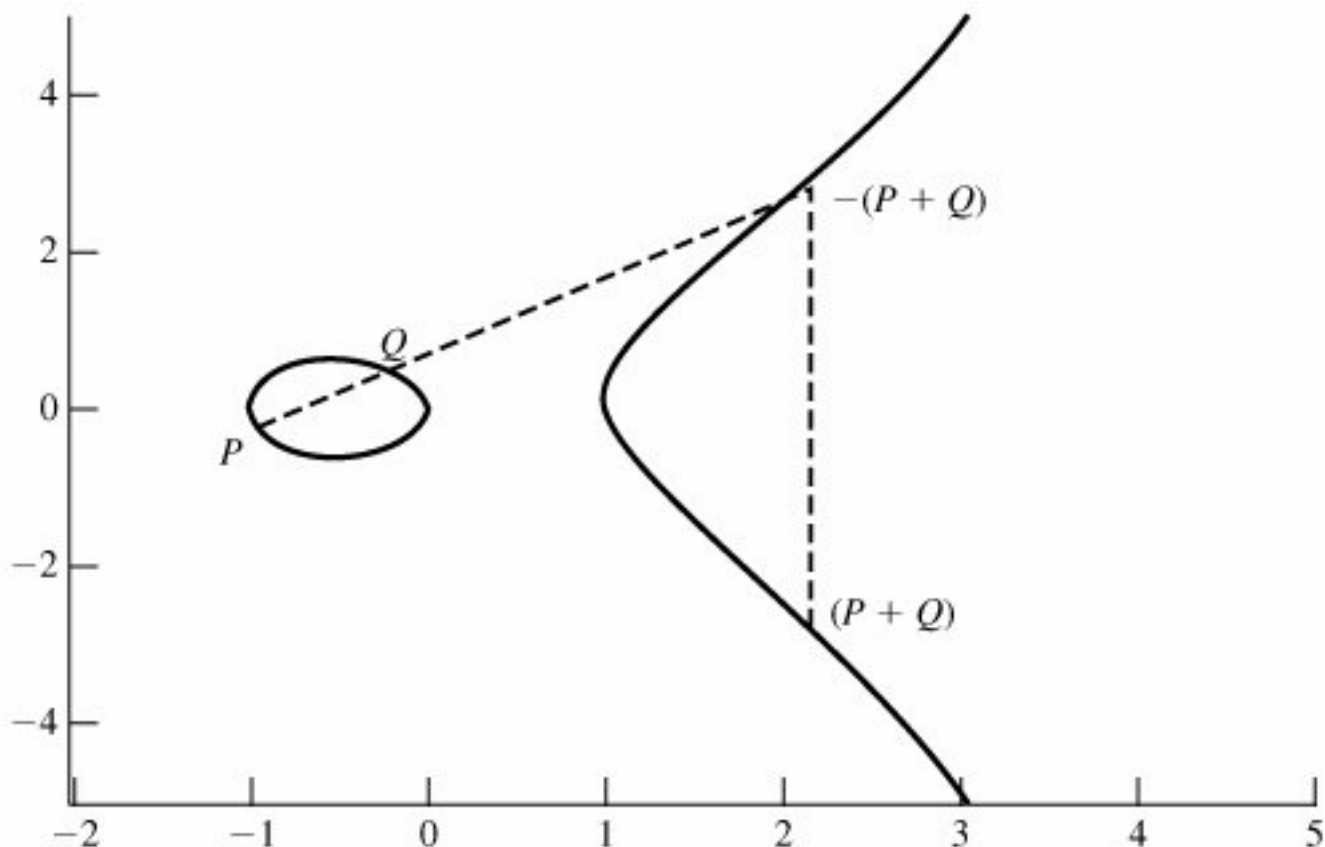
Such equations are said to be cubic, or of degree 3, because the highest exponent they contain is a 3. Also included in the definition of an elliptic curve is a single element denoted O and called the *point at infinity* or the *zero point*, which we discuss subsequently. To plot such a curve, we need to compute

$$y = \sqrt{x^3 + ax + b}$$

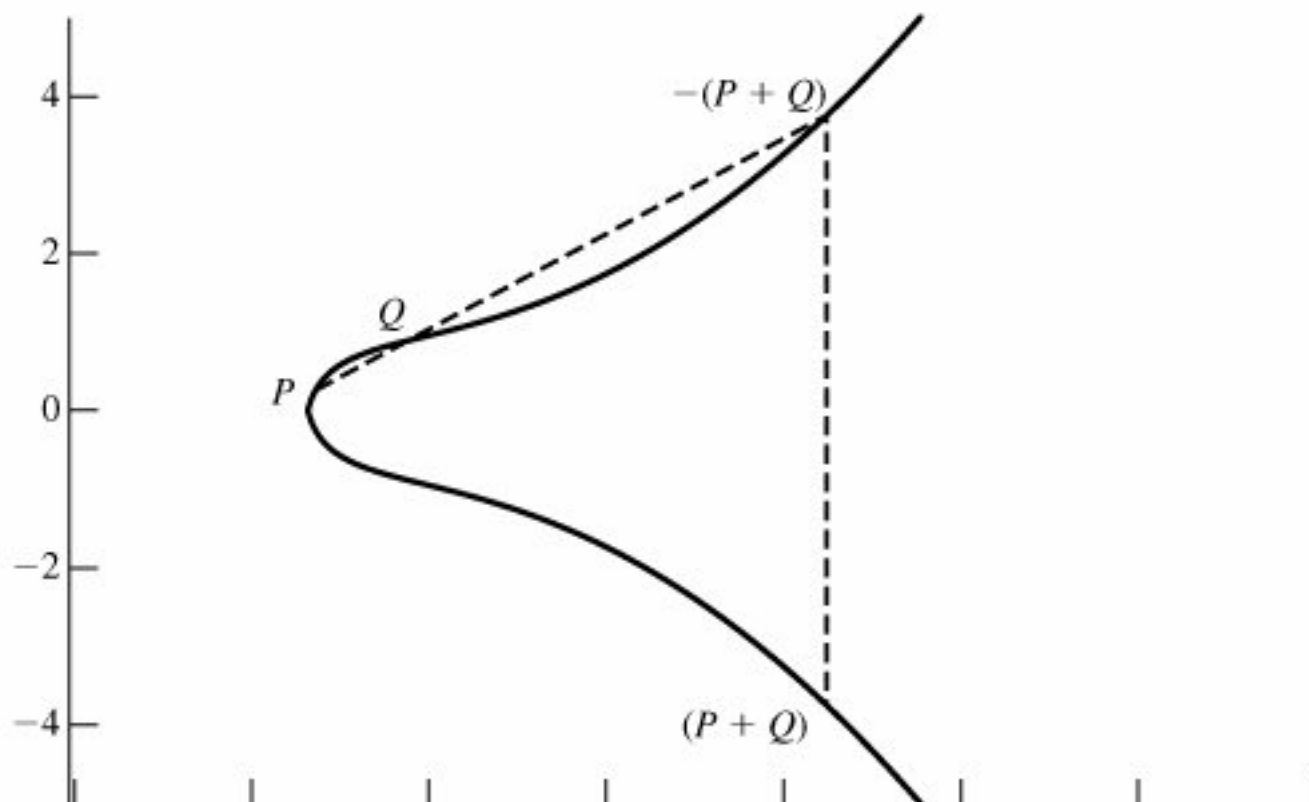
For given values of a and b , the plot consists of positive and negative values of y for each value of x . Thus each curve is symmetric about $y = 0$. [Figure 10.9](#) shows two examples of elliptic curves. As you can see, the formula sometimes produces weird-looking curves.

Figure 10.9. Example of Elliptic Curves

(This item is displayed on page 304 in the print version)



(a) $y^2 = x^3 - x$





$$(b) y^2 = x^3 + x + 1$$

Now, consider the set of points $E(a, b)$ consisting of all of the points (x, y) that satisfy [Equation \(10.1\)](#) together with the element O . Using a different value of the pair (a, b) results in a different set $E(a, b)$. Using this terminology, the two curves in [Figure 10.9](#) depict the sets $E(1,0)$ and $E(1, 1)$, respectively.

Geometric Description of Addition

It can be shown that a group can be defined based on the set $E(a, b)$ for specific values of a and b in [Equation \(10.1\)](#), provided the following condition is met:

Equation 10-2

$$4a^3 + 27b^2 \neq 0$$

To define the group, we must define an operation, called addition and denoted by $+$, for the set $E(a, b)$, where a and b satisfy [Equation \(10.2\)](#). In geometric terms, the rules for addition can be 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, $P + O = P$. In what follows, we assume $P \neq O$ and $Q \neq O$.

[Page 304]

2.

The negative of a point P is the point with the same x coordinate but the negative of the y coordinate; that is, if $P = (x, y)$, then $P = (x, -y)$. Note that these two points can be joined by a vertical line. Note that $P + (P) = P + (-P) = O$.

3.

To add two points P and Q with different x coordinates, draw a straight line between them and find the third point of intersection R . It is easily seen that there is a unique point R that is the point of intersection (unless the line is tangent to the curve at either P or Q , in which case we take $R = P$ or $R = Q$, respectively). To form a group structure, we need to define addition on these three points as follows: $P + Q = R$. That is, we define $P + Q$ to be the mirror image (with respect to the x axis) of the third point of intersection. [Figure 10.9](#) illustrates this construction.

[Page 305]

4.

The geometric interpretation of the preceding item also applies to two points, P and P , with the same x coordinate. The points are joined by a vertical line, which can be viewed as also intersecting the curve at the infinity point. We therefore have $P + (-P) = O$, consistent with item (2).

5.

To double a point Q , draw the tangent line and find the other point of intersection S . Then $Q + Q = 2Q = S$.

With the preceding list of rules, it can be shown that the set $E(a, b)$ is an abelian group.

Algebraic Description of Addition

In this subsection we present some results that enable calculation of additions over elliptic curves. ^[4] For two distinct points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ that are not negatives of each other, the slope of the line l that joins them is $\Delta = (y_Q - y_P) / (x_Q - x_P)$. There is exactly one other point where l intersects the elliptic curve, and that is the negative of the sum of P and Q . After some algebraic manipulation, we can express the sum $R = P + Q$ as follows:

^[4] For derivations of these results, see [\[KOB94\]](#) or other mathematical treatments of elliptic curves.

Equation 10-3

$$x_R = \Delta^2 - x_P - x_Q$$

$$y_R = -y_P + \Delta(x_P - x_R)$$

We also need to be able to add a point to itself: $P + P = 2P = R$. When $y_P \neq 0$, the expressions are

Equation 10-4

$$x_R = \left(\frac{3x_P^2 + a}{2y_P} \right)^2 - 2x_P$$

$$y_R = \left(\frac{3x_P^2 + a}{2y_P} \right)(x_P - x_R) - y_P$$

Elliptic Curves over \mathbb{Z}_p

Elliptic curve cryptography makes use of elliptic curves in which the variables and coefficients are all

restricted to elements of a finite field. Two families of elliptic curves are used in cryptographic applications: prime curves over Z_p and binary curves over $GF(2^m)$. For a **prime curve** over Z_p , we use a cubic equation in which the variables and coefficients all take on values in the set of integers from 0 through $p - 1$ and in which calculations are performed modulo p . For a **binary curve** defined over $GF(2^m)$, the variables and coefficients all take on values in $GF(2^n)$ and in calculations are performed over $GF(2^n)$. [FERN99] points out that prime curves are best for software applications, because the extended bit-fiddling operations needed by binary curves are not required; and that binary curves are best for hardware applications, where it takes remarkably few logic gates to create a powerful, fast cryptosystem. We examine these two families in this section and the next.

There is no obvious geometric interpretation of elliptic curve arithmetic over finite fields. The algebraic interpretation used for elliptic curve arithmetic over real numbers does readily carry over, and this is the approach we take.

For elliptic curves over Z_p , as with real numbers, we limit ourselves to equations of the form of [Equation \(10.1\)](#), but in this case with coefficients and variables limited to Z_p :

Equation 10-5

$$y^2 \bmod p = (x^3 + ax + b) \bmod p$$

For example, [Equation \(10.5\)](#) is satisfied for $a = 1$, $b = 1$, $x = 9$, $y = 9$, $y = 7$, $p = 23$:

$$\begin{aligned} 7^2 \bmod 23 &= (9^3 + 9 + 1) \bmod 23 \\ 49 \bmod 23 &= 739 \bmod 23 \\ 3 &= 3 \end{aligned}$$

Now consider the set $E_p(a, b)$ consisting of all pairs of integers (x, y) that satisfy [Equation \(10.5\)](#), together with a point at infinity O . The coefficients a and b and the variables x and y are all elements of Z_p .

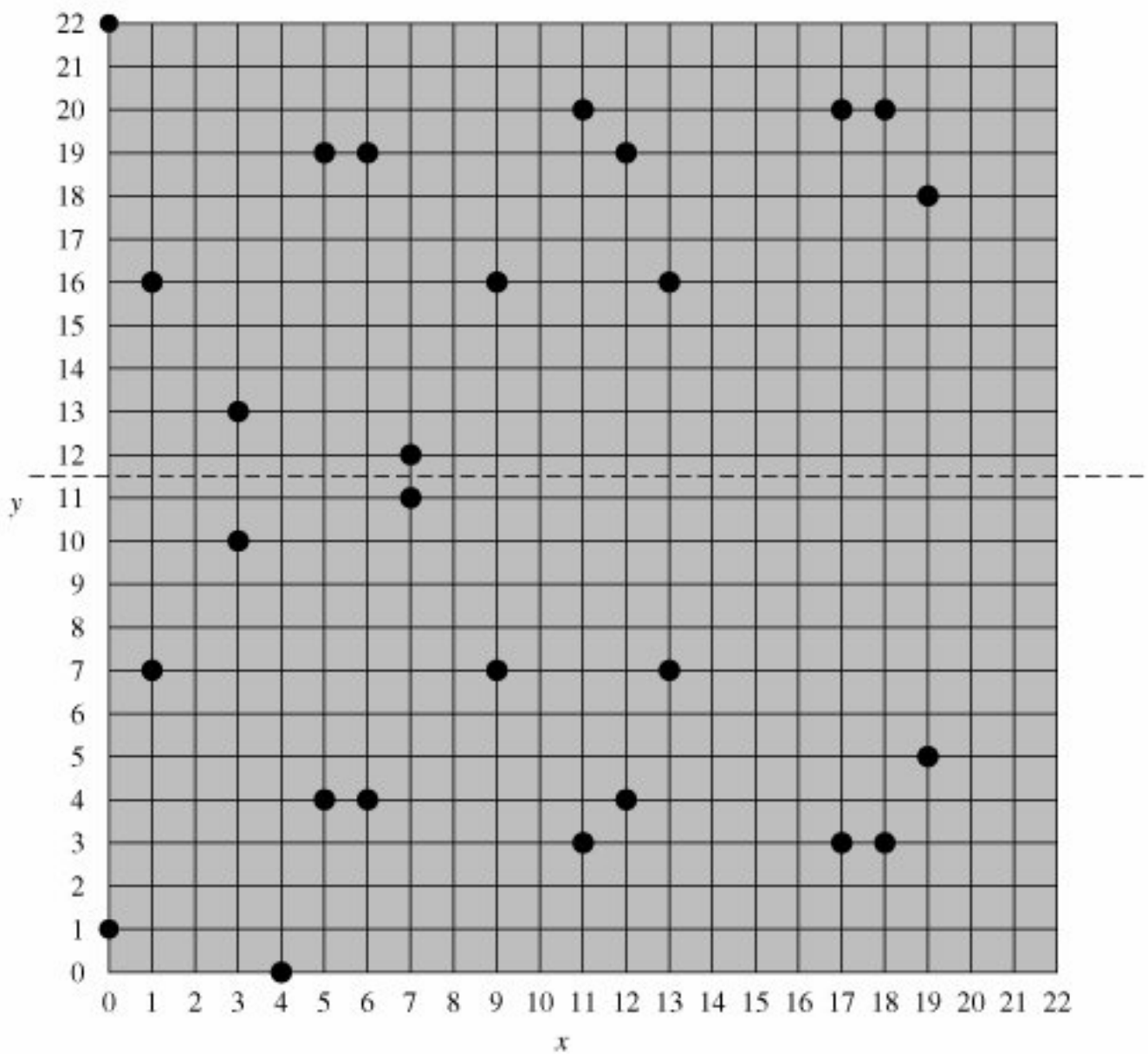
For example, let $p = 23$ and consider the elliptic curve $y^2 = x^3 + x + 1$. In this case, $a = b = 1$. Note that this equation is the same as that of [Figure 10.9b](#). The figure shows a continuous curve with all of the real points that satisfy the equation. For the set $E_{23}(1, 1)$, we are only interested in the nonnegative integers in the quadrant from $(0, 0)$ through $(p - 1, p - 1)$ that satisfy the equation mod p . [Table 10.1](#) lists the points (other than O) that are part of $E_{23}(1, 1)$. [Figure 10.10](#) plots the points of $E_{23}(1, 1)$; note that the points, with one exception, are symmetric about $y = 11.5$.

**Table 10.1. 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)

Figure 10.10. The Elliptic Curve $E_{23}(1,1)$

(This item is displayed on page 307 in the print version)



It can be shown that a finite abelian group can be defined based on the set $E_p(a, b)$ provided that $(x^3 + ax + b) \pmod p$ has no repeated factors. This is equivalent to the condition

Equation 10-6

$$(4a^3 + 27b^2) \pmod p \neq 0 \pmod p$$

Note that [Equation \(10.6\)](#) has the same form as [Equation \(10.2\)](#).

The rules for addition over $E_p(a, b)$ correspond to the algebraic technique described for elliptic curves defined over real number. For all points $P, Q \in E_p(a, b)$;

1.

$$P + O = P.$$

2.

If $P = (x_P, y_P)$ then $P + (x_P, y_P) = O$. The point (x_P, y_P) is the negative of P , denoted as $-P$. 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 $E_{23}(1,1)$.

[Page 307]

3.

If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ with $P \neq Q$, then $R = P + Q = (x_R, y_R)$ is determined by the following rules:

$$x_R = (\lambda^2 x_P x_Q) \bmod p$$

$$y_R = (\lambda(x_P x_R) y_P) \bmod p$$

where

$$\lambda = \begin{cases} \left(\frac{y_Q - y_P}{x_Q - x_P} \right) \bmod p & \text{if } P \neq Q \\ \left(\frac{3x_P^2 + a}{2y_P} \right) \bmod p & \text{if } P = Q \end{cases}$$

4.

Multiplication is defined as repeated addition; for example, $4P = P + P + P + P$.

For example, let $P = (3,10)$ and $Q = (9,7)$ in $E_{23}(1,1)$. Then

$$\lambda = \left(\frac{7 - 10}{9 - 3} \right) \bmod 23 = \left(\frac{-3}{6} \right) \bmod 23 = \left(\frac{-1}{2} \right) \bmod 23 = 11$$

$$x_R = (11^2 \cdot 3 \cdot 9) \bmod 23 = 17$$

$$y_R = (11(3 \cdot 17) \cdot 10) \bmod 23 = 164 \bmod 23 = 20$$

So $P + Q = (17, 20)$. To find $2P$,

$$\lambda = \left(\frac{3(3^2) + 1}{2 \times 10} \right) \bmod 23 = \left(\frac{5}{20} \right) \bmod 23 = \left(\frac{1}{4} \right) \bmod 23 = 6$$

The last step in the preceding equation involves taking the multiplicative inverse of 4 in Z_{23} . This can be done using the extended Euclidean algorithm defined in [Section 4.4](#). To confirm, note that $(6 \times 4) \bmod 23 = 24 \bmod 23 = 1$.

$$x_R = (6^2 - 3 \cdot 3) \bmod 23 = 30 \bmod 23 = 7$$

$$y_R = (6(3 - 7) - 10) \bmod 23 = (-34) \bmod 23 = 12$$

and $2P = (7, 12)$.

For determining the security of various elliptic curve ciphers, it is of some interest to know the number of points in a finite abelian group defined over an elliptic curve. In the case of the finite group $E_p(a, b)$, the number of points N is bounded by

$$p + 1 - 2\sqrt{p} \leq N \leq p + 1 + 2\sqrt{p}$$

Note that the number of points in $E_p(a, b)$ is approximately equal to the number of elements in Z_p , namely p elements.

Elliptic Curves over $GF(2^m)$

Recall from [Chapter 4](#) that a finite field $GF(2^m)$ consists of 2^m elements, together with addition and multiplication operations that can be defined over polynomials. For elliptic curves over $GF(2^m)$, we use a cubic equation in which the variables and coefficients all take on values in $GF(2^m)$, for some number m , and in which calculations are performed using the rules of arithmetic in $GF(2^m)$.

It turns out that the form of cubic equation appropriate for cryptographic applications for elliptic curves is somewhat different for $GF(2^m)$ than for Z_p . The form is

Equation 10-7

$$y^2 + xy = x^3 + ax^2 + b$$

where it is understood that the variables x and y and the coefficients a and b are elements of $\text{GF}(2^m)$ of and that calculations are performed in $\text{GF}(2^m)$.

Now consider the set $E_2^m(a, b)$ consisting of all pairs of integers (x, y) that satisfy [Equation \(10.7\)](#), together with a point at infinity O .

For example, let us use the finite field $\text{GF}(2^4)$ with the irreducible polynomial $f(x) = x^4 + x + 1$. This yields a generator that satisfies $f(g) = 0$, with a value of $g^4 = g + 1$, or in binary 0010. We can develop the powers of g as follows:

$g^0 = 0001$	$g^4 = 0011$	$g^8 = 0101$	$g^{12} = 1111$
$g^1 = 0010$	$g^5 = 0110$	$g^9 = 1010$	$g^{13} = 1101$
$g^2 = 0100$	$g^6 = 1100$	$g^{10} = 0111$	$g^{14} = 1001$
$g^3 = 1000$	$g^7 = 1011$	$g^{11} = 1110$	$g^{15} = 0001$

For example, $g^5 = (g^4)(g) = g^2 + g = 0110$.

[Page 309]

Now consider the elliptic curve $y^2 + xy = x^3 + g^4x^2 + 1$. In this case $a = g^4$ and $b = g^0 = 1$. One point that satisfies this equation is (g^5, g^3) :

$$(g^3)^2 + (g^5)(g^3) = (g^5)^3 + (g^4)(g^5)^2 + 1$$

$$g^6 + g^8 = g^{15} + g^{14} + 1$$

$$1100 + 0101 = 0001 + 1001 + 0001$$

$$1001 = 1001$$

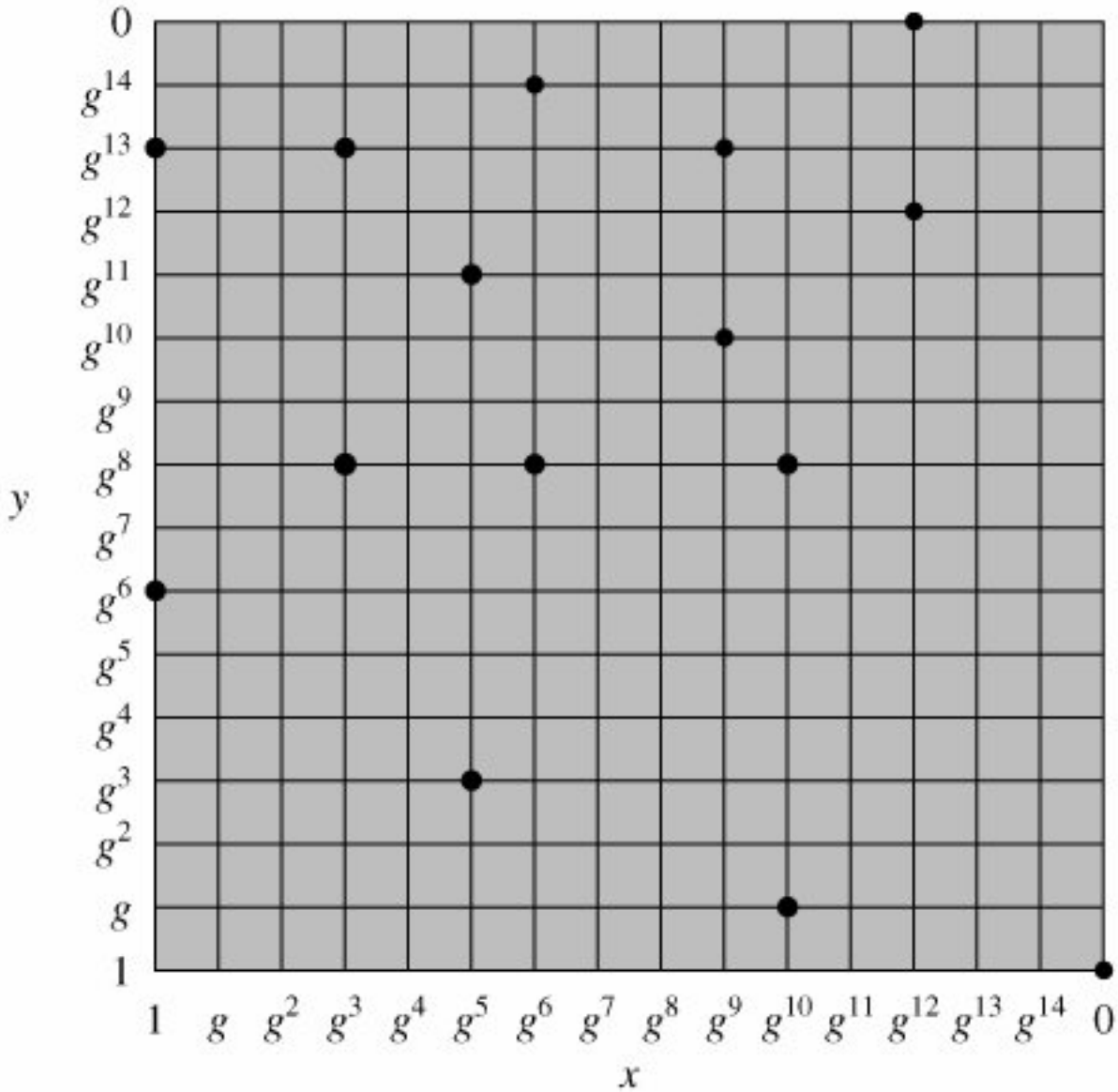
[Table 10.2](#) lists the points (other than O) that are part of $E_{2^4}(g^4, 1)$. [Figure 10.11](#) plots the points of $E_{2^4}(g^4, 1)$.

Table 10.2. Points on the Elliptic Curve $E_{2^4}(g^4, 1)$

$(0, 1)$	(g^5, g^3)	(g^9, g^{13})
$(1, g^6)$	(g^5, g^{11})	(g^{10}, g)
$(1, g^{13})$	(g^6, g^8)	(g^{10}, g^8)

(g^3, g^8)	(g^6, g^{14})	$(g^{12}, 0)$
(g^3, g^{13})	(g^9, g^{10})	(g^{12}, g^{12})

Figure 10.11. The Elliptic Curve $E_{2^4}(g^4, 1)$



It can be shown that a finite abelian group can be defined based on the set $E_{2^m}(a, b)$, provided that b

$\neq 0$. The rules for addition can be stated as follows. For all points $P, Q \in E_{2^m}(a, b)$:

1.

$$P + O = P.$$

2.

If $P = (x_P, y_P)$, then $P + (x_P, x_P + y_P) = O$. The point $(x_P, x_P + y_P)$ is the negative of P , denoted as P .

[Page 310]

3.

If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ with $P \neq Q$ and $P \neq -Q$, then $R = P + Q = (x_R, y_R)$ is determined by the following rules:

$$x_R = \lambda^2 + \lambda + x_P + x_Q + a$$

$$y_R = \lambda(x_P + x_Q) + x_R + y_P$$

where

$$\lambda = \frac{y_Q + y_P}{x_Q + x_P}$$

4.

If $P = (x_P, y_P)$ then $R = 2P = (x_R, y_R)$ is determined by the following rules:

$$x_R = \lambda^2 + \lambda + a$$

$$y_R = x_P^2 + (\lambda + 1)x_R$$

where

$$\lambda = x_P + \frac{y_P}{x_P}$$

10.4. Elliptic Curve Cryptography

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 . This is called the discrete logarithm problem for elliptic curves.

We give an example taken from the Certicom Web site (www.certicom.com). Consider the group $E_{23}(9, 17)$. This is the group defined by the equation $y^2 \bmod 23 = (x^3 + 9x + 17) \bmod 23$. What is the discrete logarithm k of $Q = (4, 5)$ to the base $P = (16, 5)$? The brute-force method is to compute multiples of P until Q is found.

Thus

$P = (16, 5)$; $2P = (20, 20)$; $3P = (14, 14)$; $4P = (19, 20)$; $5P = (13, 10)$; $6P = (7, 3)$; $7P = (8, 7)$; $8P = (12, 17)$; $9P = (4, 5)$.

Because $9P = (4, 5) = Q$, the discrete logarithm $Q = (4, 5)$ to the base $P = (16, 5)$ is $k = 9$. In a real application, k would be so large as to make the brute-force approach infeasible.

In the remainder of this section, 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 large integer q , which is either a prime number p or an integer of the form 2^m and elliptic curve parameters a and b for [Equation \(10.5\)](#) or [Equation \(10.7\)](#). This defines the elliptic group of points $E_q(a, b)$. Next, pick a *base point* $G = (x_1, y_1)$ in $E_p(a, b)$ whose order is a very large value n . The **order** n of a point G on an elliptic curve is the smallest positive integer n such that $nG = O$. $E_q(a, b)$ and G are parameters of the cryptosystem known to all participants.

A key exchange between users A and B can be accomplished as follows ([Figure 10.12](#)):

1.

A selects an integer n_A less than n . This is A's private key. A then generates a public key $P_A = n_A$

$x G$; the public key is a point in $Eq(a, b)$.

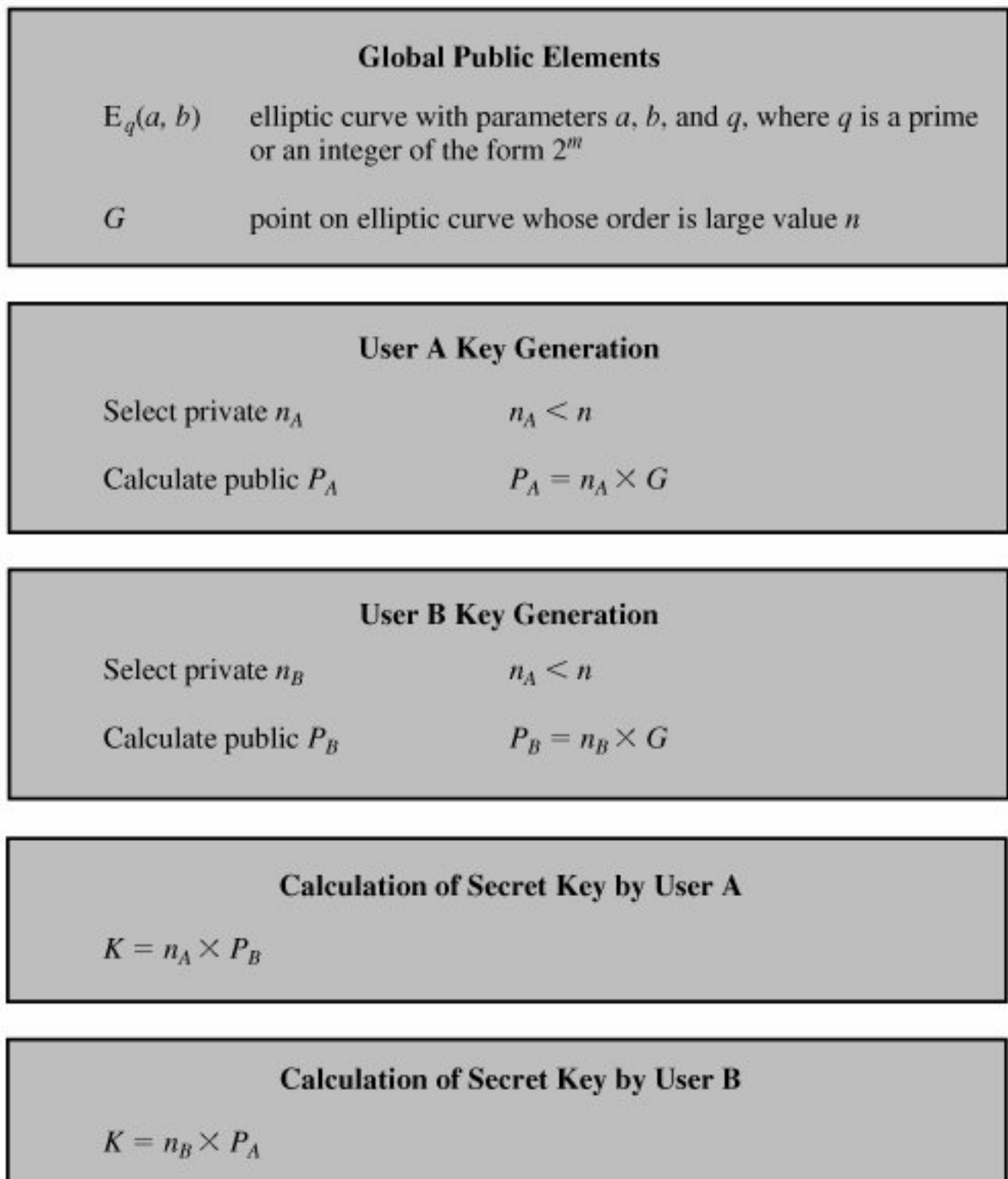
2.

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 $K = n_B \times P_A$.

Figure 10.12. ECC Diffie-Hellman Key Exchange



$$K = n_B \times P_A$$

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.

[Page 312]

As an example, ^[5] take $p = 211$; $Ep(0, 4)$, which is equivalent to the curve $y^2 = x^3 + 4$; and $G = (2, 2)$. One can calculate that $240G = O$. 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 $121(130, 203) = 203(115, 48) = (161, 69)$.

^[5] Provided by Ed Schaefer of Santa Clara University.

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.

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 $Eq(a, b)$; for example, see [Table 10.1](#). 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 $Eq(a, b)$ 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 [KOB94]), take $p = 751$; $E_p(1, 188)$, which is equivalent to the curve $y^2 = x^3 + 188x$; 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 10.3 compares various algorithms by showing comparable key sizes in terms of computational effort for cryptanalysis. 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 [JUR97]. Thus, there is a computational advantage to using ECC with a shorter key length than a comparably secure RSA.

Table 10.3. Comparable Key Sizes in Terms of Computational Effort for Cryptanalysis

Symmetric Scheme (key size in bits)	ECC-Based Scheme (size of n in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
92	384	7680
256	512	15360



10.5. Recommended Reading and Web Sites

A quite readable treatment of elliptic curve cryptography is [ROSI99]; the emphasis is on software implementation. Another readable, but rigorous, book is [HANK04]. Two other good treatments, both of which contain some rather stiff mathematics, are [BLAK99] and [ENGE99]. There are also good but more concise descriptions in [KUMA98], [STIN02], and [KOB94]. Two interesting survey treatments are [FERN99] and [JURI97].

BLAK99 Blake, I.; Seroussi, G.; and Smart, N. *Elliptic Curves in Cryptography*. Cambridge: Cambridge University Press, 1999.

ENGE99 Enge, A. *Elliptic Curves and Their Applications to Cryptography*. Norwell, MA: Kluwer Academic Publishers, 1999.

FERN99 Fernandes, A. "Elliptic Curve Cryptography." *Dr. Dobb's Journal*, December 1999.

HANK04 Hankerson, D.; Menezes, A.; and Vanstone, S. *Guide to Elliptic Curve Cryptography*. New York: Springer, 2004.

JURI97 Jurisic, A., and Menezes, A. "Elliptic Curves and Cryptography." *Dr. Dobb's Journal*, April 1997.

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.

ROSI99 Rosing, M. *Implementing Elliptic Curve Cryptography*. Greenwich, CT: Manning Publications, 1999.

STIN02 Stinson, D. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 2002.

Recommended Web Site



- **Certicom:** Extensive collection of technical material on elliptic curve cryptography and other topics in cryptography

◀ PREV

NEXT ▶

10.6. Key Terms, Review Questions, and Problems

Key Terms

[abelian group](#)

[binary curve](#)

[cubic equation](#)

[Diffie-Hellman key exchange](#)

[discrete logarithm](#)

[elliptic curve](#)

[elliptic curve arithmetic](#)

[elliptic curve cryptography](#)

[finite field](#)

[key distribution](#)

[key management](#)

[man-the-middle attack](#)

[prime curve](#)

[primitive root](#)

[public-key certificate](#)

[public-key directory](#)

[zero point](#)

Review Questions

10.1 What are two different uses of public-key cryptography related to key distribution?

- 10.2 List four general categories of schemes for the distribution of public keys.
- 10.3 What are the essential ingredients of a public-key directory?
- 10.4 What is a public-key certificate?
- 10.5 What are the requirements for the use of a public-key certificate scheme?
- 10.6 Briefly explain Diffie-Hellman key exchange.
- 10.7 What is an elliptic curve?
- 10.8 What is the zero point of an elliptic curve?
- 10.9 What is the sum of three points on an elliptic curve that lie on a straight line?

Problems

- 10.1 Users A and B use the Diffie-Hellman key exchange technique with a common prime $q = 71$ and a primitive root $\alpha = 7$.
 - a.
If user A has private key $X_A = 5$, what is A's public key Y_A ?
 - b.
If user B has private key $X_B = 12$, what is B's public key Y_B ?
 - c.
What is the shared secret key?

10.2 Consider a Diffie-Hellman scheme with a common prime $q = 11$ and a primitive root $\alpha = 2$.

a.

Show that 2 is a primitive root of 11.

b.

If user A has public key $Y_A = 9$, what is A's private key X_A ?

c.

If user B has public key $Y_B = 3$, what is the shared secret key K , shared with A?

10.3 In the Diffie-Hellman protocol, each participant selects a secret number x and sends the other participant $\alpha^x \bmod q$ for some public number α . What would happen if the participants sent each other x^α for some public number α instead? Give at least one method Alice and Bob could use to agree on a key. Can Eve break your system without finding the secret numbers? Can Eve find the secret numbers?

[Page 315]

10.4 This problem illustrates the point that the Diffie-Hellman protocol is not secure without the step where you take the modulus; i.e. the "Indiscrete Log Problem" is not a hard problem! You are Eve, and have captured Alice and Bob and imprisoned them. You overhear the following dialog.

Bob: Oh, let's not bother with the prime in the Diffie-Hellman protocol, it will make things easier.

Alice: Okay, but we still need a base to raise things to. How about $g = 3$?

Bob: All right, then my result is 27.

Alice: And mine is 243.

What is Bob's secret X_B and Alice's secret X_A ? What is their secret combined key?

(Don't forget to show your work.)

10.5 [Section 10.2](#) describes a man-in-the-middle attack on the Diffie-Hellman key exchange protocol in which the adversary generates two public-private key pairs for the attack. Could the same attack be accomplished with one pair? Explain.

10.6 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 α , 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 $K = (Y_B)^k \bmod q$.

3.

Encrypt M as the pair of integers (C_1, C_2) where

$$C_1 = \alpha^k \bmod q \quad C_2 = KM \bmod q$$

User B recovers the plaintext as follows:

1.

Compute $K = (C_1)^{X_B} \bmod q$.

2.

Compute $M = (C_2 K^{-1}) \bmod q$.

Show that the system works; that is, show that the decryption process does recover the plaintext.

- 10.7** 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 ?
- 10.8** Rule (5) for doing arithmetic in elliptic curves over real numbers states that to double a point Q , draw the tangent line and find the other point of intersection S . Then $Q + Q = 2Q = S$. If the tangent line is not vertical, there will be exactly one point of intersection. However, suppose the tangent line is vertical? In that case, what is the value $2Q$? What is the value $3Q$?
- 10.9** Demonstrate that the two elliptic curves of [Figure 10.9](#) each satisfy the conditions for a group over the real numbers.
- 10.10** Is $(4, 7)$ a point on the elliptic curve $y^2 = x^3 - 5x + 5$ over real numbers?
- 10.11** On the elliptic curve over the real numbers $y^2 = x^3 - 36x$, let $P = (3.5, 9.5)$ and $Q = (2.5, 8.5)$. Find $P + Q$ and $2P$.
- 10.12** Does the elliptic curve equation $y^2 = x^3 + 10x + 5$ define a group over Z_{17} ?
- 10.13** 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 .
- 10.14** What are the negatives of the following elliptic curve points over Z_{17} ? $P = (5, 8)$; $Q = (3, 0)$; $R = (0, 6)$.
- 10.15** For $E_{11}(1, 6)$, consider the point $G = (2, 7)$. Compute the multiples of G from $2G$ through $13G$.

10.16 This problem performs elliptic curve encryption/decryption using the scheme outlined in [Section 10.4](#). 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 .

10.17 The following is a first attempt at an Elliptic Curve signature scheme. We have a global elliptic curve, prime p , and "generator" G . Alice picks a private signing key X_A and forms the public verifying key $Y_A = X_A G$. To sign a message M :

- Alice picks a value k .
- Alice sends Bob M , k and the signature $S = M + kX_A G$.
- Bob verifies that $M = S - kY_A$

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.

10.18 Here is an improved version of the scheme given in the previous problem. As before, we have a global elliptic curve, prime p , and "generator" G . Alice picks a private signing key X_A and forms the public verifying key $Y_A = X_A G$. To sign a message M ,

- Bob picks a value k .
- Bob sends Alice $C_1 = kG$.
- Alice sends Bob M and the signature $S = M X_A C_1$.
- Bob verifies that $M = S + kY_A$

a.

Show that this scheme works. That is, show that the verification process produces an equality if the signature is valid.

b.

Show that forging a message in this scheme is as hard as breaking (ElGamal) Elliptic Curve Cryptography. (Or find an easier way to forge a message?)

c.

This scheme has an extra "pass" compared to other cryptosystems and signature schemes we have looked at. What are some drawbacks to this?

Chapter 11. Message Authentication and Hash Functions

[11.1 Authentication Requirements](#)

[11.2 Authentication Functions](#)

[Message Encryption](#)

[Message Authentication Code](#)

[Hash Function](#)

[11.3 Message Authentication Codes](#)

[Requirements for MACs](#)

[Message Authentication Code Based on DES](#)

[11.4 Hash Functions](#)

[Requirements for a Hash Function](#)

[Simple Hash Functions](#)

[Birthday Attacks](#)

[Block Chaining Techniques](#)

[11.5 Security of Hash Functions and MACs](#)

[Brute-Force Attacks](#)

[Cryptanalysis](#)

[11.6 Recommended Reading](#)

[11.7 Key Terms, Review Questions, and Problems](#)

[Key Terms](#)

[Review Questions](#)

[Problems](#)

[Appendix 11A Mathematical Basis of the Birthday Attack](#)

[Related Problem](#)

[The Birthday Paradox](#)

[Useful Inequality](#)

[The General Case of Duplications](#)

[Overlap between Two Sets](#)

[Page 318]

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

Key Points

- Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent by (i.e., contain no modification, insertion, deletion, or replay) and that the purported identity of the sender is valid.
- Symmetric encryption provides authentication among those who share the secret key. Encryption of a message by a sender's private key also provides a form of authentication.
- The two most common cryptographic techniques for message authentication are a message authentication code (MAC) and a secure hash function.
- A MAC is an algorithm that requires the use of a secret key. A MAC takes a variable-length message and a secret key as input and produces an authentication code. A recipient in possession of the secret key can generate an authentication code to verify the integrity of the message.

- A hash function maps a variable-length message into a fixed length hash value, or message digest. For message authentication, a secure hash function must be combined in some fashion with a secret key.

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.

[Page 319]

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 12](#).

◀ PREV

NEXT ▶

11.1. 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.

Source repudiation: Denial of transmission of message by source.

8.

Destination repudiation: Denial of receipt of message by destination.

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 of the attacks listed under items 3 through 6. Dealing with item 8 may require a combination of the use of digital signatures and a protocol designed to counter this attack.

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 the source.

[← PREV](#)

[NEXT →](#)

11.2. Authentication Functions

Any message authentication or digital signature mechanism has two levels of functionality. 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 a 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 function of the message and a secret key that produces a fixed-length value that serves as the authenticator
- **Hash function:** A 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 then examined in greater detail in [Sections 11.3](#) and [11.4](#).

Message Encryption

Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryption schemes.

Symmetric Encryption

Consider the straightforward use of symmetric encryption ([Figure 11.1a](#)). A message M 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.

Figure 11.1. Basic Uses of Message Encryption

(This item is displayed on page 321 in the print version)

[\[View full size image\]](#)

legitimate message. This conclusion is incontrovertible: If M can be any bit pattern, then regardless of the value of X , the value $Y = D(K, X)$ is *some* bit pattern and therefore must be accepted as authentic plaintext.

[Page 321]

Thus, in general, we require that only a small subset of all possible bit patterns be 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:

mareseatoatsanddoeseatoatsandlittlelambseativy

[Page 322]

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:

ytuqrndufpwkyvhfzlcumlgolbbwhttqdnreabdasprwp

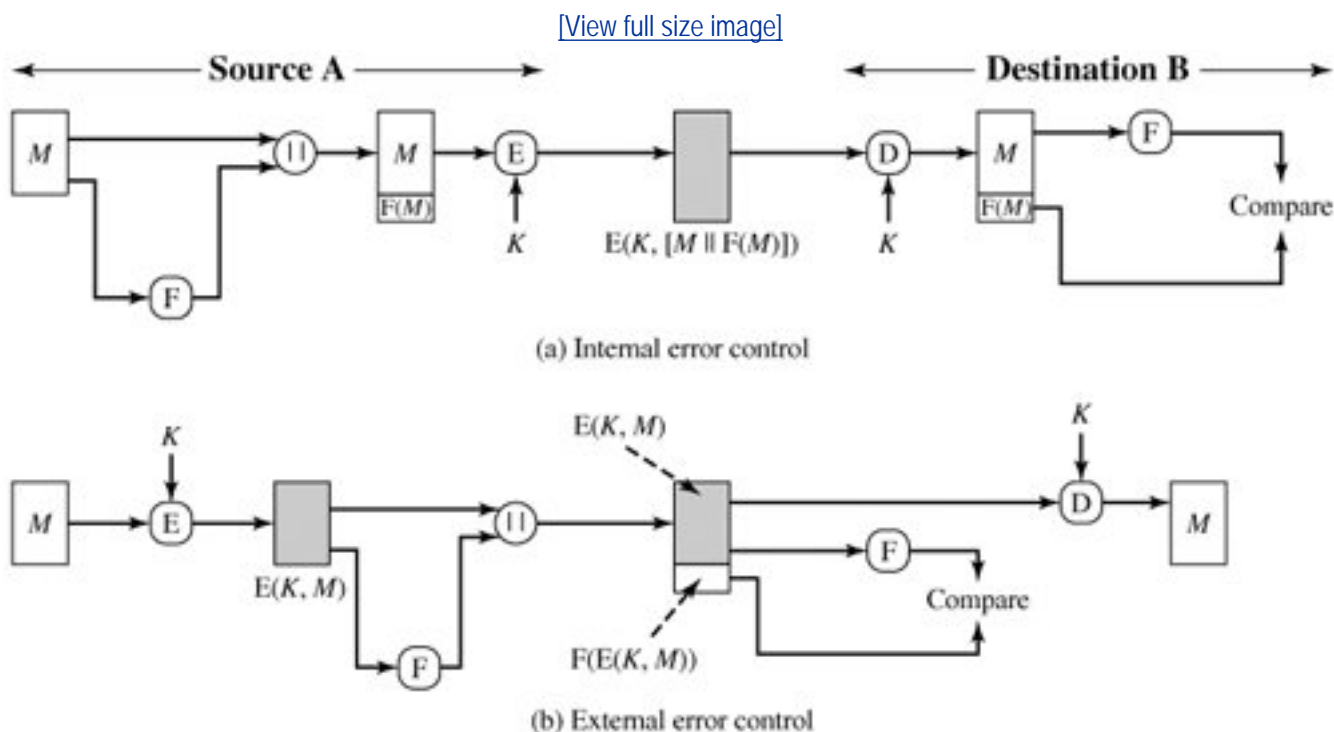
which does not fit the profile of ordinary English.

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 11.2a](#). 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 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.

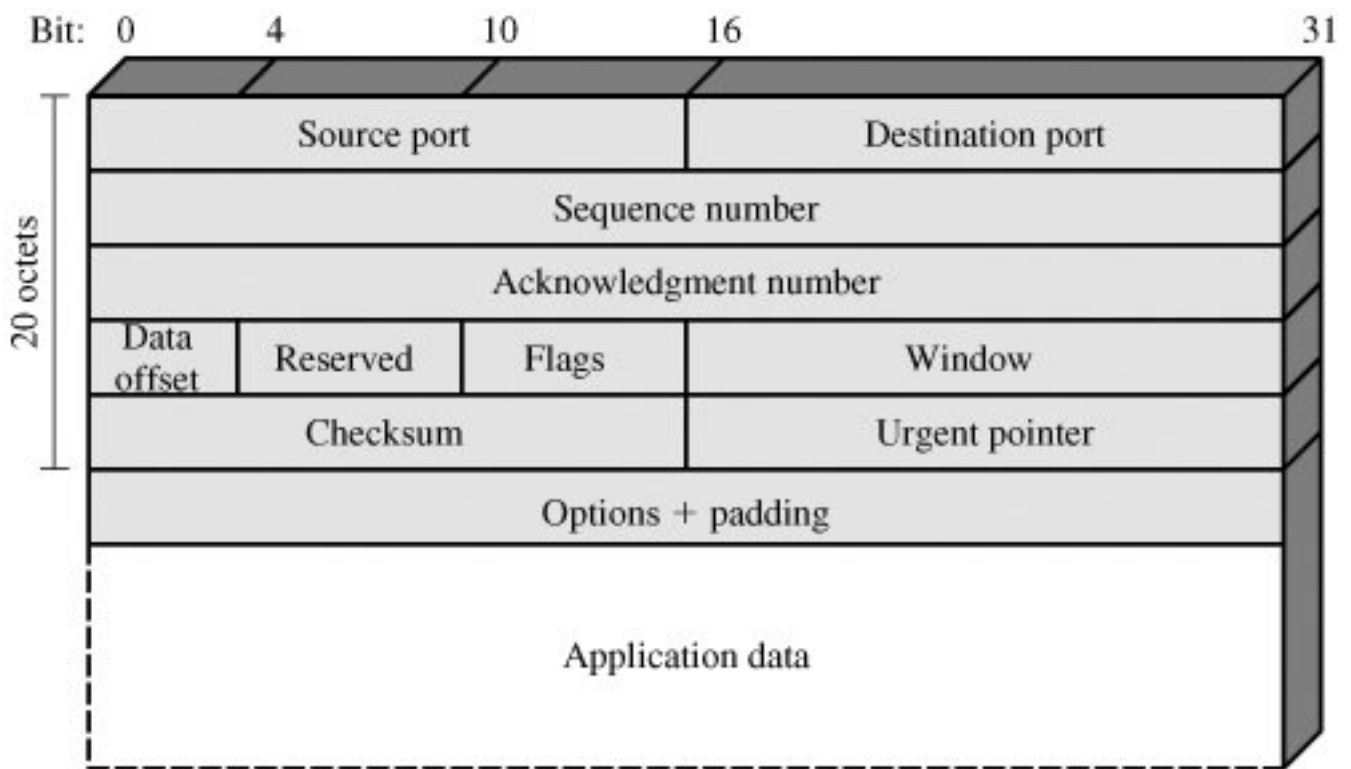
Figure 11.2. Internal and External Error Control



Note that the order in which the FCS and encryption functions are performed is critical. The sequence illustrated in [Figure 11.2a](#) is referred to in [\[DIFF79\]](#) as internal error control, which the authors contrast with external error control ([Figure 11.2b](#)). 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 11.3](#) 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 we could simply encrypt all of the datagram except the IP header (see [Figure 7.5](#)). 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 also 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 11.3. TCP Segment



Public-Key Encryption

The straightforward use of public-key encryption ([Figure 11.1b](#)) provides confidentiality but not authentication. The source (A) uses the public key PU_b of the destination (B) to encrypt M . Because only B has the corresponding private key PR_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.

[Page 324]

To provide authentication, A uses its private key to encrypt the message, and B uses A's public key to decrypt ([Figure 11.1c](#)). This provides authentication using the same type of reasoning as in the symmetric encryption case: The message must have come from A because A is the only party that possesses PR_a and therefore the only party with the information necessary to construct ciphertext that can be decrypted with PU_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 11.1c](#) does provide authentication. It also provides what is known as digital signature.^[1] Only A could have constructed the ciphertext because only A possesses PR_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.

[1] This is not the way in which digital signatures are constructed, as we shall see, but the principle is the same.

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 11.1d). 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 11.1 summarizes the confidentiality and authentication implications of these various approaches to message encryption.

Table 11.1. Confidentiality and Authentication Implications of Message Encryption (see Figure 11.1)

(This item is displayed on page 325 in the print version)

A \longrightarrow B: E(K, M)
• Provides confidentiality
Only A and B share K
• Provides a degree of authentication
Could come only from A
Has not been altered in transit
Requires some formatting/redundancy
• Does not provide signature
Receiver could forge message
Sender could deny message
(a) Symmetric encryption
A \longrightarrow B: E(PU_b, M)
• Provides confidentiality
Only B has PR_b to decrypt
• Provides no authentication

Any party could use PU_b to encrypt message and claim to be A
(b) Public-key (asymmetric) encryption: confidentiality
$A \longrightarrow B: E(PR_a, M)$
<ul style="list-style-type: none"> • Provides authentication and signature
<p>Only A has PR_b to encrypt</p> <p>Has not been altered in transit</p> <p>Requires some formatting/redundancy</p> <p>Any party can use PU_a to verify signature</p>
(c) Public-key encryption: authentication and signature
$A \longrightarrow B: E(PU_b, E(PR_a, M))$
<ul style="list-style-type: none"> • Provides confidentiality because of PU_b
<ul style="list-style-type: none"> • Provides authentication and signature because of PR_a
(d) Public-key encryption: confidentiality, authentication, and signature

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: $MAC = C(K, M)$, where

M = input message

C = MAC function

K = shared secret key

MAC = message authentication code

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 11.4a](#)). 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

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.

[Page 325]

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.

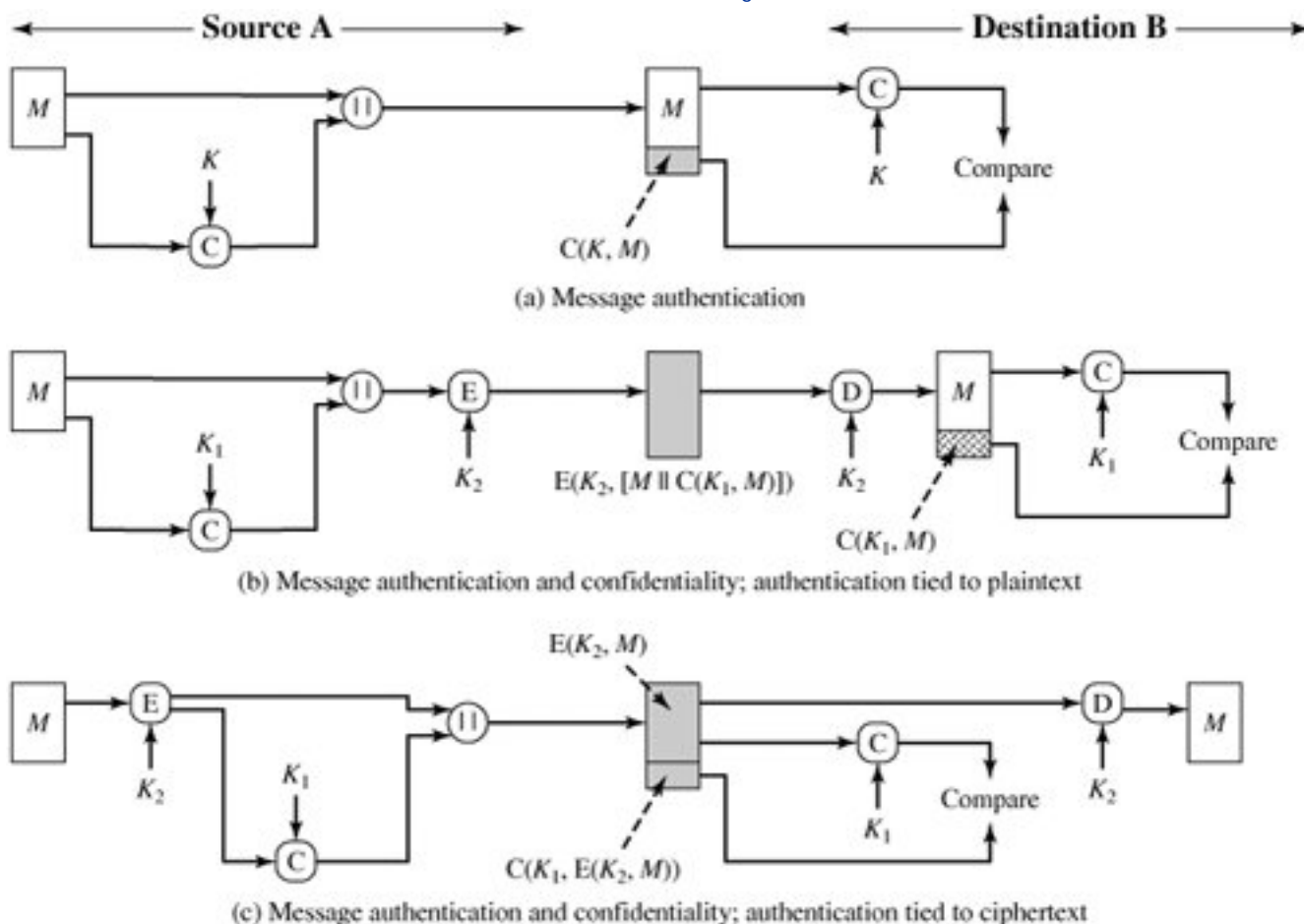
3.

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.

[Page 326]

Figure 11.4. Basic Uses of Message Authentication Code (MAC)

[\[View full size image\]](#)



A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must for decryption. 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.

For example, suppose that we are using 100-bit messages and a 10-bit MAC. Then, there are a total of 2^{100} different messages but only 2^{10} different MACs. So, on average, each MAC value is generated by a total of $2^{100}/2^{10} = 2^{90}$ different messages. If a 5-bit key is used, then there are $2^5 = 32$ different mappings from the set of messages to the set of MAC values.

It turns out that because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption.

The process depicted in [Figure 11.4a](#) 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 11.4b](#)) or before ([Figure 11.4c](#)) 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 11.4b](#) is used.

[Page 327]

Because symmetric 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 associated 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 11.2](#) summarizes the confidentiality and authentication implications of the approaches illustrated in [Figure 11.4](#).

Table 11.2. Basic Uses of Message Authentication Code C
(see [Figure 11.4](#))

A → B: $M C(K, M)$
•Provides authentication
Only A and B share K
(a) Message authentication
A → B: $E(K_2, [M C(K, M)])$

• Provides authentication
Only A and B share K_1
• Provides confidentiality
Only A and B share K_2
(b) Message authentication and confidentiality: authentication tied to plaintext
A \rightarrow B: $E(K_2, M) C(K_1, E(K_2, M))$
• Provides authentication
Using K_1
• Provides confidentiality
Using K_2
(c) Message authentication and confidentiality: authentication tied to ciphertext

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 output, referred to as a **hash code** $H(M)$. Unlike a MAC, a hash code does not use a key but is a function only of the input message. The hash code is also referred to as a **message digest** or **hash value**. 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 11.5](#) illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows:

a.

The message plus concatenated hash code is encrypted using symmetric encryption. This is identical in structure to the internal error control strategy shown in [Figure 11.2a](#). 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 symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

[Page 330]

Note that the combination of hashing and encryption results in an overall function that is, in fact, a MAC ([Figure 11.4a](#)). 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 private-key-encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.

e.

It is possible to use 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.

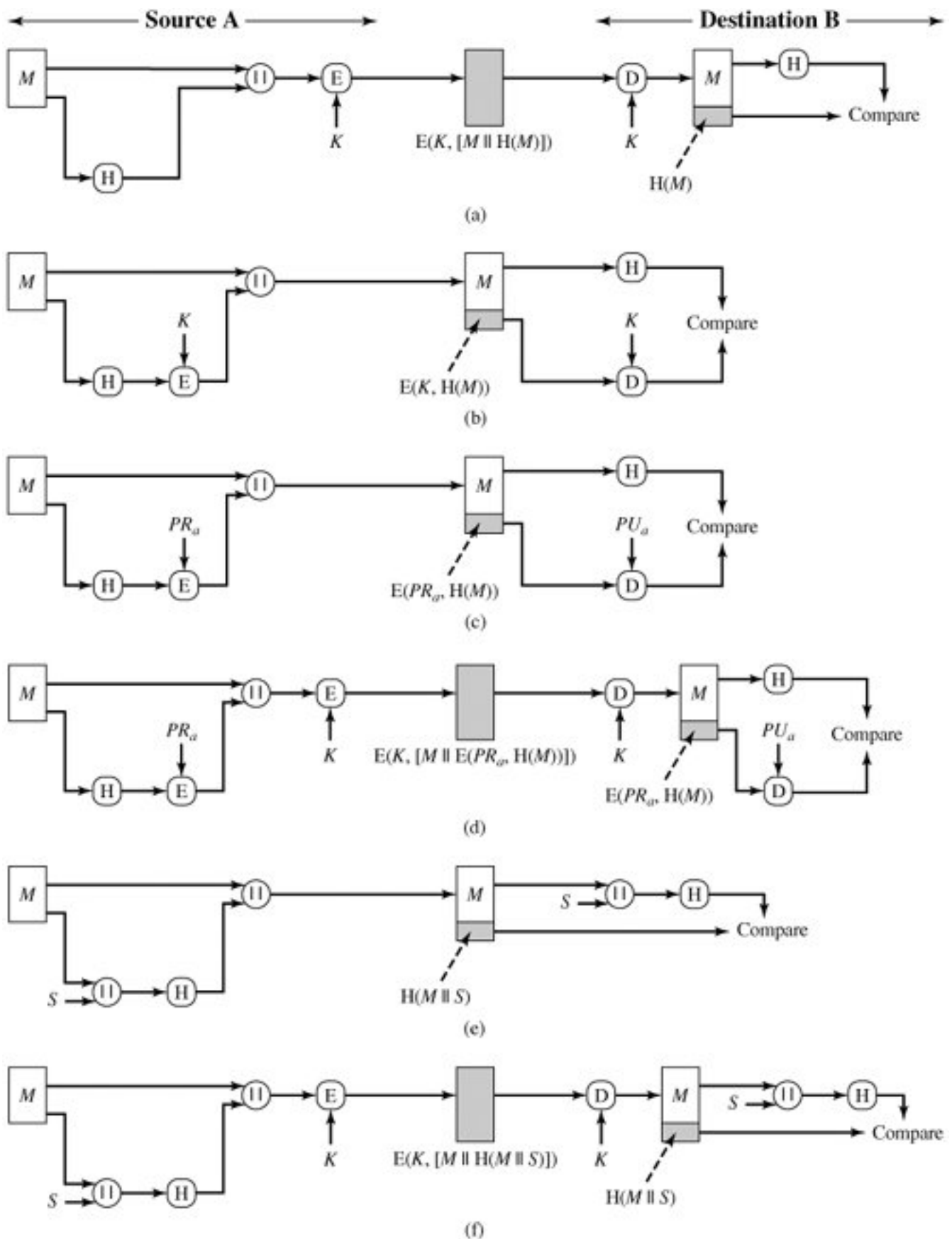
f.

Confidentiality can be added to the approach of (e) by encrypting the entire message plus the hash code.

Figure 11.5. Basic Uses of Hash Function

(This item is displayed on page 329 in the print version)

[\[View full size image\]](#)



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 11.5e](#)). Several reasons for this interest are pointed out in [\[TSUD92\]](#):

- Encryption software is relatively 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. For example, until the patent expired, RSA was patented and had to be licensed, adding a cost.

[Table 11.3](#) summarizes the confidentiality and authentication implications of the approaches illustrated in [Figure 11.5](#). We next examine MACs and hash codes in more detail.

Table 11.3. Basic Uses of Hash Function H (see [Figure 11.5](#))

$A \rightarrow B: E(K, [M H(M)])$	$A \rightarrow B: E(K, [M E(PR_a, H(M))])$
• Provides confidentiality	• Provides authentication and digital signature
Only A and B share K	• Provides confidentiality
• Provides authentication	Only A and B share K
$H(M)$ is cryptographically protected	
(a) Encrypt message plus hash code	(d) Encrypt result of (c) shared secret key
$A \rightarrow B: M E(K, H(M))$	$A \rightarrow B: M H(M S)$
• Provides authentication	• Provides authentication
$H(M)$ is cryptographically protected	Only A and B share S
(b) Encrypt hash code shared secret key	(e) Compute hash code of message plus secret value
$A \rightarrow B: M E(PR_a, H(M))$	$A \rightarrow B: E(K, [M H(M S)])$
• Provides authentication and digital signature	• Provides authentication
$H(M)$ is cryptographically protected	Only A and B share S
Only A could create $E(PR_a, H(M))$	• Provides confidentiality

	Only A and B share K
(c) Encrypt hash codesender's private <i>key</i>	(f) Encrypt result of (e)

[← PREV](#)

[NEXT →](#)

11.3. Message Authentication Codes

A MAC, also known as a cryptographic checksum, is generated by a function C of the form

$$\text{MAC} = C(K, M)$$

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. Other examples are discussed in [Chapter 12](#).

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, due to the many-to-one nature of the function. 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 $MAC_1 = C(K, M_1)$, 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 $2^{(k-n)}$ keys resulting from Round 1

Number of matches $\approx 2^{(k-2n)}$

and so on. On average, α 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

$$\Delta(M) = X_1 \oplus X_2 \oplus \dots \oplus X_m$$

$$C(K, M) = E(K, \Delta(M))$$

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 satisfy the following requirements:

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. The second requirement 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

The Data Authentication Algorithm, based on DES, has been one of the most widely used MACs for a number of years. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17). However, as we discuss in [Chapter 12](#), security weaknesses in this algorithm have been discovered and it is being replaced by newer and stronger algorithms.

The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES ([Figure 6.4](#)) with an initialization vector of zero. The data (e.g., message, record, file, or program) to be authenticated are 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 11.6](#)):

$$O_1 = E(K, D_1)$$

$$O_2 = E(K, [D_2 \oplus O_1])$$

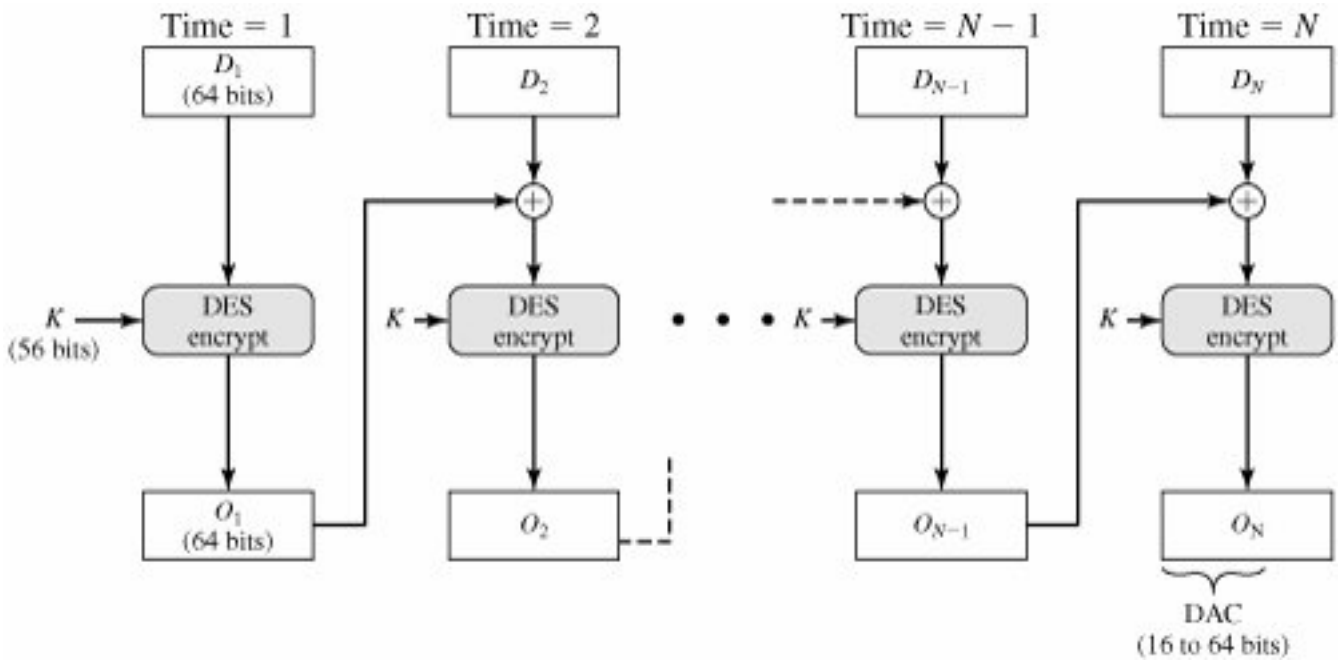
$$O_3 = E(K, [D_3 \oplus O_2])$$

-
-
-

$$O_N = E(K, [D_N \oplus O_{N-1}])$$

Figure 11.6. Data Authentication Algorithm (FIPS PUB 113)

[\[View full size image\]](#)



The DAC consists of either the entire block O_N or the leftmost M bits of the block, with $16 \leq M \leq 64$.

11.4. Hash Functions

A hash value h 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 11.5](#)).

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 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 value 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 $y \neq x$ such that $H(y) = H(x)$. 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**.^[2]

[2] 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.

The first three properties are requirements for the practical application of a hash function to message authentication.

The fourth property, the one-way property, states that it is easy to generate a code given a message but virtually impossible to generate a message given a code. This property is important if the authentication technique involves the use of a secret value ([Figure 11.5e](#)). 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 ([Figures 11.5b](#) and [c](#)). 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 type 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

C_i = i th bit of the hash code, $1 \leq i \leq n$

m = number of n -bit blocks in the input

b_{ij} = i th bit in j th block

\oplus = XOR operation

This operation 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:

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.

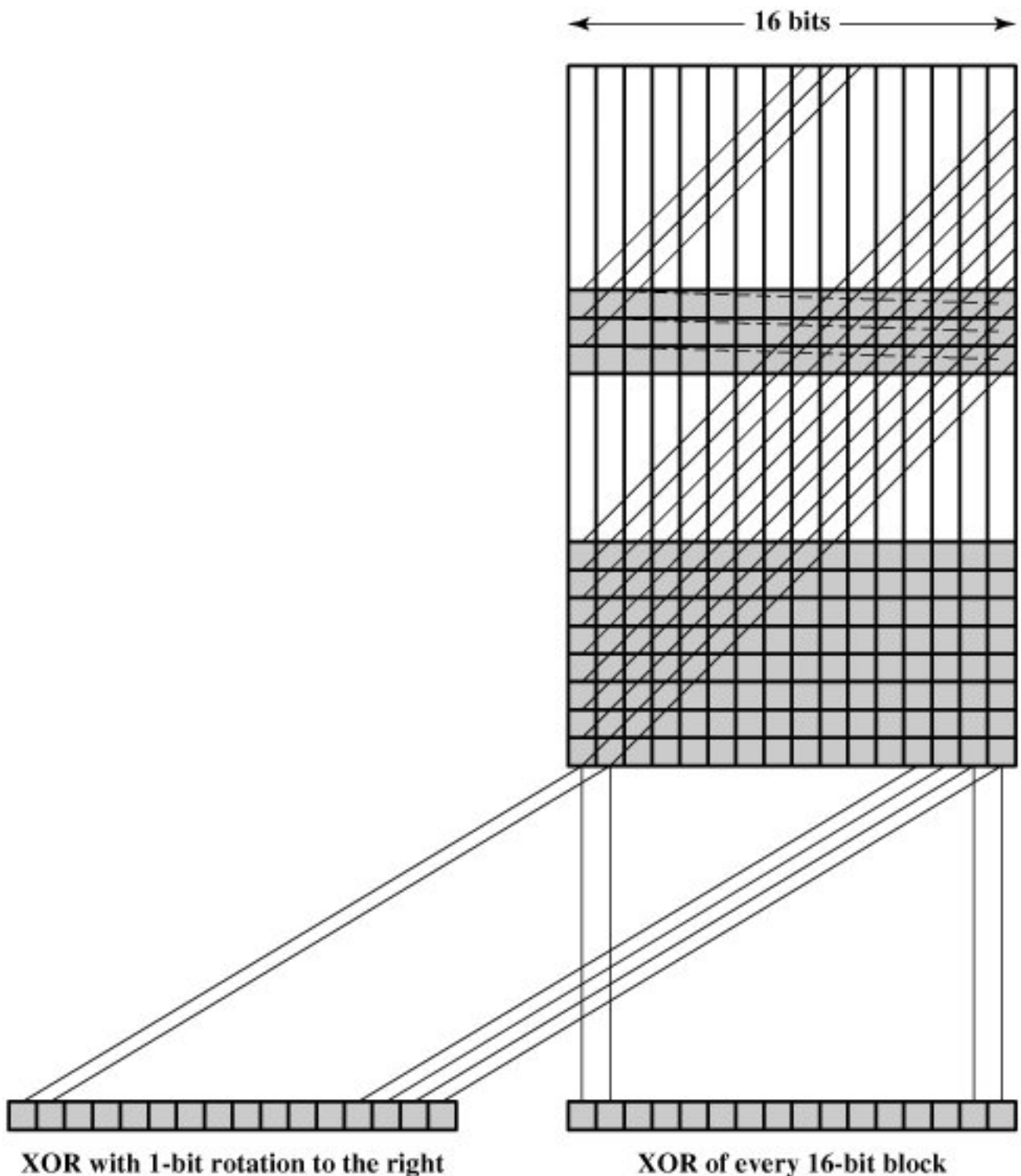
b.

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 11.7](#) illustrates these two types of hash functions for 16-bit hash values.

Figure 11.7. Two Simple Hash Functions

(This item is displayed on page 337 in the print version)



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 11.5b](#) and [c](#). 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 11.5a](#)). But you 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:

[Page 337]

$$C = X_{N+1} = X_1 \oplus X_2 \oplus \dots \oplus X_N$$

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 6.4), we have

$$X_1 = IV \oplus D(K, Y_1)$$

$$X_i = Y_{i-1} \oplus D(K, Y_i)$$

$$X_{N+1} = Y_N \oplus D(K, Y_{N+1})$$

[Page 338]

But X_{N+1} is the hash code:

$$\begin{aligned} X_{N+1} &= X_1 \oplus X_2 \oplus \dots \oplus X_N \\ &= [IV \oplus D(K, Y_1)] \oplus [Y_1 \oplus D(K, Y_2)] \oplus \dots \oplus [Y_{N-1} \oplus \dots \oplus D(K, Y_N)] \end{aligned}$$

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 11.5b or 11.5c), 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 11A, Equation (11.1)].

However, a different sort of attack is possible, based on the birthday paradox (Appendix 11A). Yuval proposed the following strategy [YUVA79]:

1.

The source, A, is prepared to "sign" a message by appending the appropriate m -bit hash code and encrypting that hash code with A's private key ([Figure 11.5c](#)).

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 11A, Equation \(11.7\)](#)].

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 11.8 \[DAVI89\]](#) provides an example.

Figure 11.8. A Letter in 2^{37} Variations [DAVI89]

(This item is displayed on page 339 in the print version)

[\[View full size image\]](#)

Dear Anthony,

{ This letter is } to introduce { you to } { Mr. } Alfred { P. }
{ I am writing } { to you } { -- } { -- }

Barton, the { newly appointed } { chief } jewellery buyer for { our }
{ the }

Northern { European } { area } . He { will take } over { the }
{ Europe } { division } . He { 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 } . Please { 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 }

The conclusion to be drawn from this is that the length of the hash code should be substantial. We discuss this further in [Section 11.5](#).

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 [[RAB178](#)]. Divide a message M into fixed-size blocks M_1, M_2, \dots, M_N and use a symmetric encryption system such as DES to compute the hash code G as follows:

H_0 = initial value

$H_i = E(M_i, H_i, H_{i-1})$

$G = H_N$

[Page 340]

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 for $H_i = E(Q_i, H_{i-1})$ 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 $E(X, H_{N^2}) = D(Y, G)$.

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 [DAVI89] describe the following variation:

$$H_i = E(M_i, H_{i-1}) \oplus H_{i-1}$$

Another variation, proposed in [MEYE88],

$$H_i = E(H_{i-1}, M_i) \oplus M_i$$

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 [JUEN87].

Thus, attention has been directed at finding other approaches to hashing. Many of these have also been shown to have weaknesses [MITC92]. We examine two strong hash functions in [Chapter 12](#).

11.5. Security of Hash Functions and Macs

Just as with symmetric 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 hash 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 [VANO94] 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. However, even 160 bits is now considered weak. We return to this topic in [Chapter 12](#).

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, we 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.

[Page 342]

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

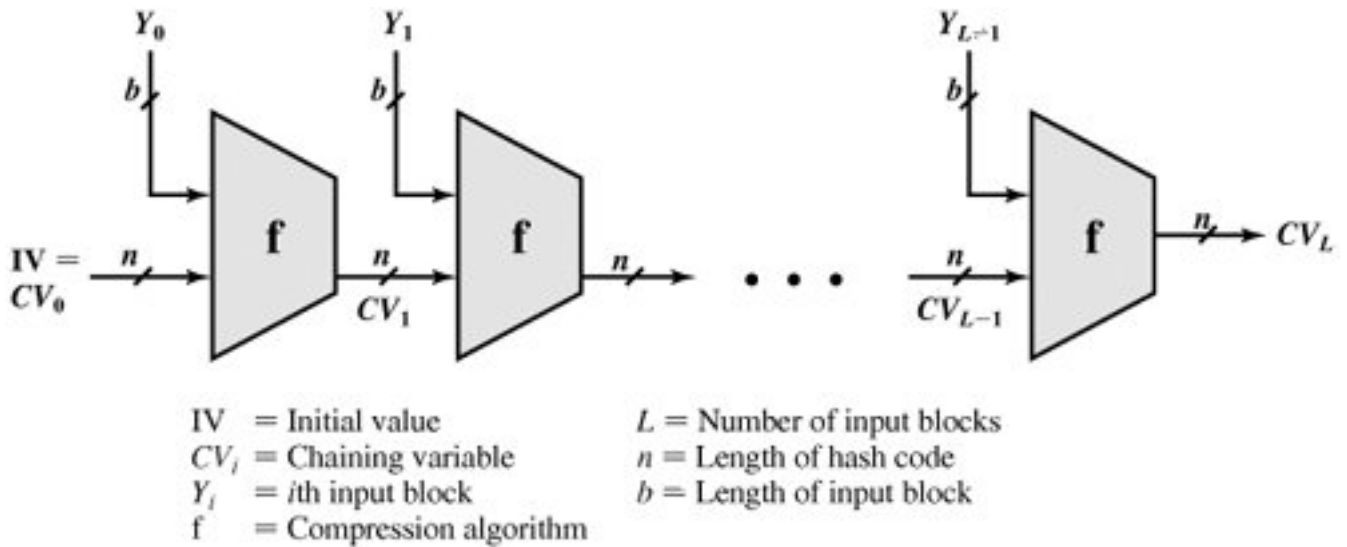
As with encryption algorithms, cryptanalytic attacks on hash functions and MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. The way to measure the resistance of a hash or MAC algorithm to cryptanalysis is to compare its strength to 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 11.9](#). 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 SHA and Whirlpool, which are discussed in [Chapter 12](#). The hash function takes an input message and partitions it into L 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.

Figure 11.9. General Structure of Secure Hash Code

[\[View full size image\]](#)



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. Often, $b > n$; hence the term *compression*. The hash function can be summarized as follows:

$$CV_0 = IV = \text{initial } n\text{-bit value}$$

$$CV_i = f(CV_{i-1}, Y_i) \quad 1 \leq i \leq L$$

$$H(M) = CV_L$$

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 Damgard

[DAMG89] that if the compression function is collision resistant, then so is the resultant iterated hash function. [3] 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.

[3] The converse is not necessarily true.

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 twice the block size b (because we must append a length field) into a hash code of length n , where $b \geq n$. What is required is that it is computationally infeasible to find collisions.

[Page 344]

The attacks that have been mounted on hash functions are rather complex and beyond our scope here. For the interested reader, [DOBB96] and [BELL97] are recommended.

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].

◀ PREY

NEXT ▶

11.6. Recommended Reading

[[JUN85](#)] and [[JUN87](#)] 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 [[STIN02](#)] and [[MENE97](#)]. A good recent survey is [[PREN99](#)].

[JUN85](#) Jueneman, R.; Matyas, S.; and Meyer, C. "Message Authentication." *IEEE Communications Magazine*, September 1988.

[JUN87](#) 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.

[PREN99](#) Preneel, B. "The State of Cryptographic Hash Functions." *Proceedings, EUROCRYPT '96*, 1996; published by Springer-Verlag.

[STIN02](#) Stinson, D. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 2002.

11.7. Key Terms, Review Questions, and Problems

Key Terms

[authenticator](#)

[birthday attack](#)

[birthday paradox](#)

[compression function](#)

[cryptographic checksum](#)

[hash code](#)

[hash function](#)

[hash value](#)

[message authentication](#)

[message authentication code \(MAC\)](#)

[message digest](#)

[one-way hash function](#)

[strong collision resistance](#)

[weak collision resistance](#)

Review Questions

- 11.1 What types of attacks are addressed by message authentication?
- 11.2 What two levels of functionality comprise a message authentication or digital signature mechanism?
- 11.3 What are some approaches to producing message authentication?

- 11.4 When a combination of symmetric encryption and an error control code is used for message authentication, in what order must the two functions be performed?

[Page 345]

- 11.5 What is a message authentication code?
- 11.6 What is the difference between a message authentication code and a one-way hash function?
- 11.7 In what ways can a hash value be secured so as to provide message authentication?
- 11.8 Is it necessary to recover the secret key in order to attack a MAC algorithm?
- 11.9 What characteristics are needed in a secure hash function?
- 11.10 What is the difference between weak and strong collision resistance?
- 11.11 What is the role of a compression function in a hash function?

Problems

- 11.1 If F is an error-detection function, either internal or external use ([Figure 11.2](#)) 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?
- 11.2 The data authentication algorithm, described in [Section 11.3](#), can be defined as using the cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero ([Figure 11.6](#)). Show that the same result can be produced using the cipher feedback mode.