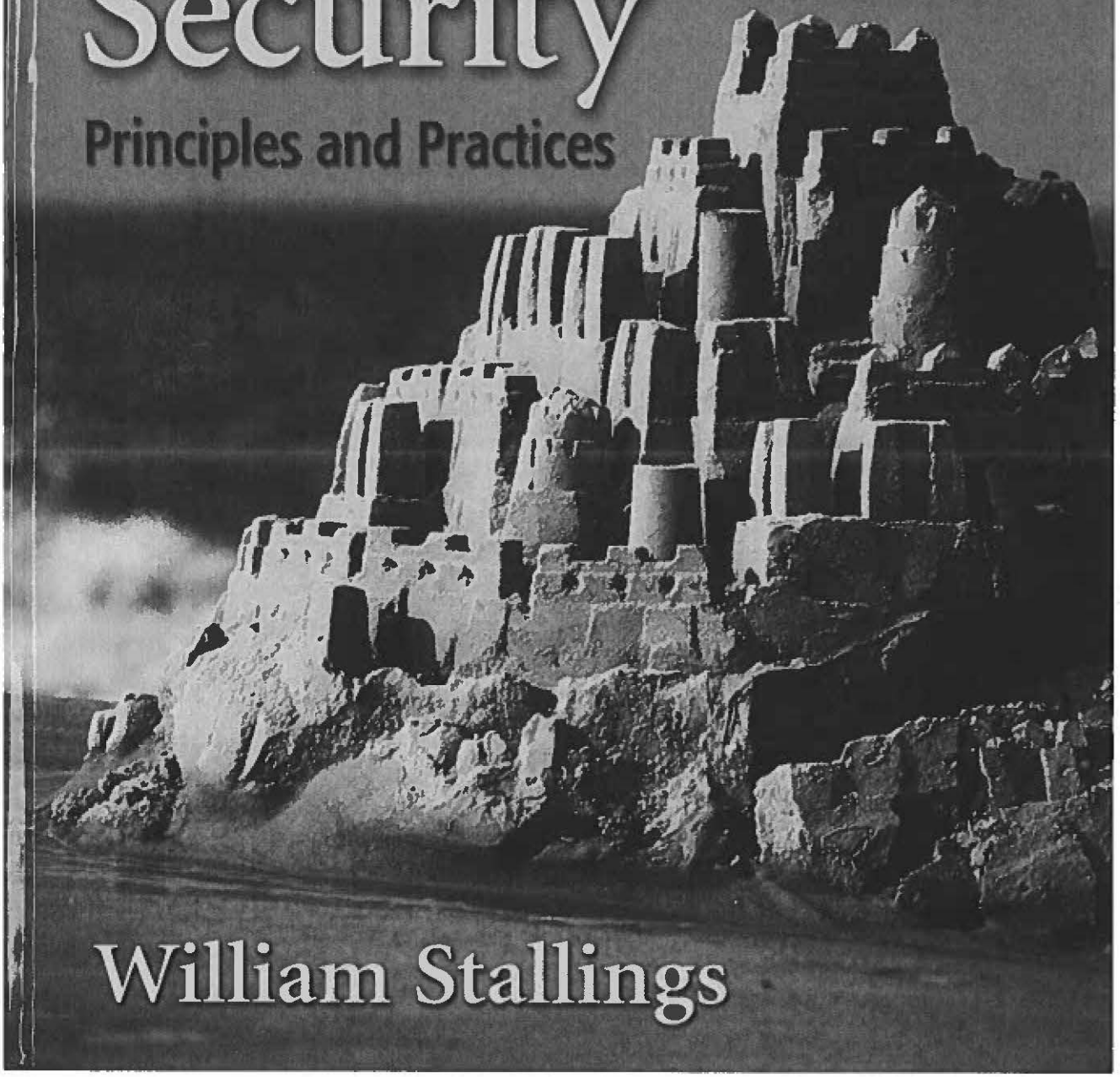
Cryptography and Network Security

Fourth Edition

Principles and Practices



William Stallings

CRYPTOGRAPHY AND NETWORK SECURITY

PRINCIPLES AND PRACTICES

FOURTH EDITION

William Stallings



Upper Saddle River, NJ 07458

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

ISBN: 0-13-187316-4

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 Mexico, S.A. de C.V.
Pearson Education—Japan, *Tokyo*
Pearson Education Malaysia, Pte. Ltd.
Pearson Education Inc., *Upper Saddle River, New Jersey*

CONTENTS

Notation xi

Preface xiii

Chapter 0 Reader's Guide 1

0.1 Outline of This Book 2
0.2 Roadmap 2
0.3 Internet and Web Resources 4

Chapter 1 Introduction 6

1.1 Security Trends 9
1.2 The OSI Security Architecture 12
1.3 Security Attacks 13
1.4 Security Services 16
1.5 Security Mechanisms 19
1.6 A Model for Network Security 22
1.7 Recommended Reading and Web Sites 24
1.8 Key Terms, Review Questions, and Problems 25

PART ONE SYMMETRIC CIPHERS 26

Chapter 2 Classical Encryption Techniques 28

2.1 Symmetric Cipher Model 30
2.2 Substitution Techniques 35
2.3 Transposition Techniques 49
2.4 Rotor Machines 51
2.5 Steganography 53
2.6 Recommended Reading and Web Sites 55
2.7 Key Terms, Review Questions, and Problems 56

Chapter 3 Block Ciphers and the Data Encryption Standard 62

3.1 Block Cipher Principles 64
3.2 The Data Encryption Standard 72
3.3 The Strength of DES 82
3.4 Differential and Linear Cryptanalysis 83
3.5 Block Cipher Design Principles 86
3.6 Recommended Reading 90
3.7 Key Terms, Review Questions, and Problems 90

Chapter 4 Finite Fields 95

4.1 Groups, Rings, and Fields 97
4.2 Modular Arithmetic 101
4.3 The Euclidean Algorithm 107
4.4 Finite Fields of the Form GF(p) 109
4.5 Polynomial Arithmetic 113
4.6 Finite Fields of the Form GF(2^n) 119

JPMC Ex. 1041
Page 4

4.7 Recommended Reading and Web Sites 129
4.8 Key Terms, Review Questions, and Problems 130

Chapter 5 Advanced Encryption Standard 134

5.1 Evaluation Criteria for AES 135
5.2 The AES Cipher 140
5.3 Recommended Reading and Web Sites 160
5.4 Key Terms, Review Questions, and Problems 161
Appendix 5A Polynomials with Coefficients in GF(2^8) 163
Appendix 5B Simplified AES 165

Chapter 6 More on Symmetric Ciphers 174

6.1 Multiple Encryption and Triple DES 175
6.2 Block Cipher Modes of Operation 181
6.3 Stream Ciphers and RC4 189
6.4 Recommended Reading and Web Site 194
6.5 Key Terms, Review Questions, and Problems 194

Chapter 7 Confidentiality Using Symmetric Encryption 199

7.1 Placement of Encryption Function 201
7.2 Traffic Confidentiality 209
7.3 Key Distribution 210
7.4 Random Number Generation 218
7.5 Recommended Reading and Web Sites 227
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

8.1 Prime Numbers 236
8.2 Fermat's and Euler's Theorems 238
8.3 Testing for Primality 242
8.4 The Chinese Remainder Theorem 245
8.5 Discrete Logarithms 247
8.6 Recommended Reading and Web Site 253
8.7 Key Terms, Review Questions, and Problems 254

Chapter 9 Public-Key Cryptography and RSA 257

9.1 Principles of Public-Key Cryptosystems 259
9.2 The RSA Algorithm 268
9.3 Recommended Reading and Web Site 280
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

10.1 Key Management 290
10.2 Diffie-Hellman Key Exchange 298
10.3 Elliptic Curve Arithmetic 301
10.4 Elliptic Curve Cryptography 310

10.5 Recommended Reading and Web Site 313
10.6 Key Terms, Review Questions, and Problems 314

Chapter 11 Message Authentication and Hash Functions 317

11.1 Authentication Requirements 319
11.2 Authentication Functions 320
11.3 Message Authentication Codes 331
11.4 Hash Functions 334
11.5 Security of Hash Functions and MACs 340
11.6 Recommended Reading 344
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

12.1 Secure Hash Algorithm 353
12.2 Whirlpool 358
12.3 HMAC 368
12.4 CMAC 372
12.5 Recommended Reading and Web Sites 374
12.6 Key Terms, Review Questions, and Problems 375

Chapter 13 Digital Signatures and Authentication Protocols 377

13.1 Digital Signatures 378
13.2 Authentication Protocols 382
13.3 Digital Signature Standard 390
13.4 Recommended Reading and Web Sites 393
13.5 Key Terms, Review Questions, and Problems 393

PART THREE NETWORK SECURITY APPLICATIONS 398

Chapter 14 Authentication Applications 400

14.1 Kerberos 401
14.2 X.509 Authentication Service 419
14.3 Public-Key Infrastructure 428
14.4 Recommended Reading and Web Sites 430
14.5 Key Terms, Review Questions, and Problems 431
Appendix 14A Kerberos Encryption Techniques 433

Chapter 15 Electronic Mail Security 436

15.1 Pretty Good Privacy 438
15.2 S/MIME 457
15.3 Recommended Web Sites 474
15.4 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

16.1 IP Security Overview 485
16.2 IP Security Architecture 487

16.3 Authentication Header 493
16.4 Encapsulating Security Payload 498
16.5 Combining Security Associations 503
16.6 Key Management 506
16.7 Recommended Reading and Web Site 516
16.8 Key Terms, Review Questions, and Problems 517
Appendix 16A Internetworking and Internet Protocols 518

Chapter 17 Web Security 527

17.1 Web Security Considerations 528
17.2 Secure Socket Layer and Transport Layer Security 531
17.3 Secure Electronic Transaction 549
17.4 Recommended Reading and Web Sites 560
17.5 Key Terms, Review Questions, and Problems 561

PART FOUR SYSTEM SECURITY 563

Chapter 18 Intruders 565

18.1 Intruders 567
18.2 Intrusion Detection 570
18.3 Password Management 582
18.4 Recommended Reading and Web Sites 591
18.5 Key Terms, Review Questions, and Problems 592
Appendix 18A The Base-Rate Fallacy 594

Chapter 19 Malicious Software 598

19.1 Viruses and Related Threats 599
19.2 Virus Countermeasures 610
19.3 Distributed Denial of Service Attacks 614
19.4 Recommended Reading and Web Sites 619
19.5 Key Terms, Review Questions, and Problems 620

Chapter 20 Firewalls 621

20.1 Firewall Design Principles 622
20.2 Trusted Systems 634
20.3 Common Criteria for Information Technology Security Evaluation 640
20.3 Recommended Reading and Web Sites 644
20.4 Key Terms, Review Questions, and Problems 645

APPENDICES 647

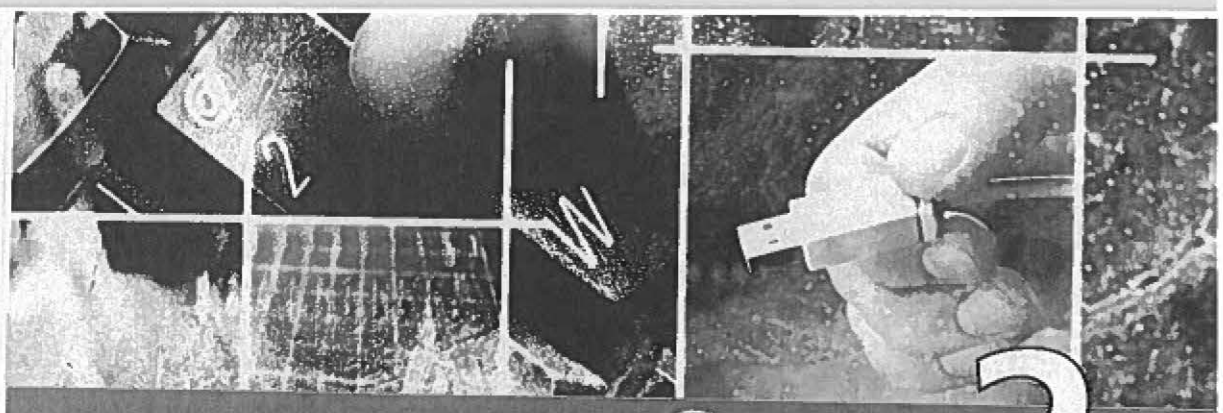
Appendix A Standards and Standards-Setting Organizations 647

A.1 The Importance of Standards 648
A.2 Internet Standards and the Internet Society 649
A.3 National Institute of Standards and Technology 652

Appendix B Projects for Teaching Cryptography and Network Security 653

B.1 Research Projects 654
B.2 Programming Projects 655
B.3 Laboratory Exercises 655

B.4 Writing Assignments 655
B.5 Reading/Report Assignments 656
Glossary 657
References 663
Index 673



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

JPMC Ex. 1041
Page 9

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

JPMC Ex. 1041
Page 10

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.

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

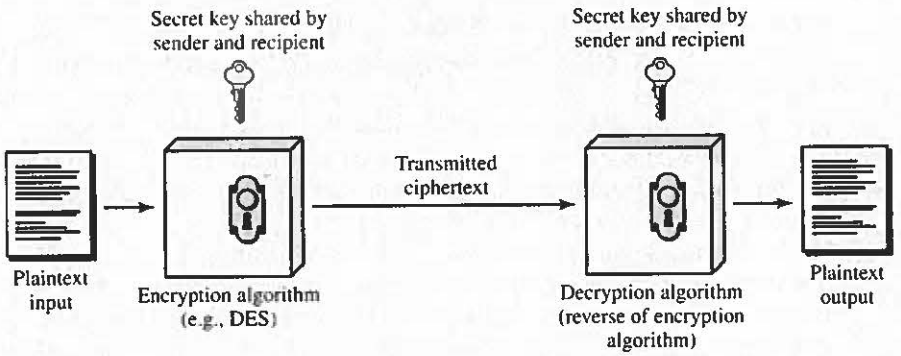


Figure 2.1 Simplified Model of Conventional Encryption

in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.

2. 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, $X = [X_1, X_2, \ldots, X_M]$. 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 $K = [K_1, K_2, \ldots, K_J]$ 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.

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

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

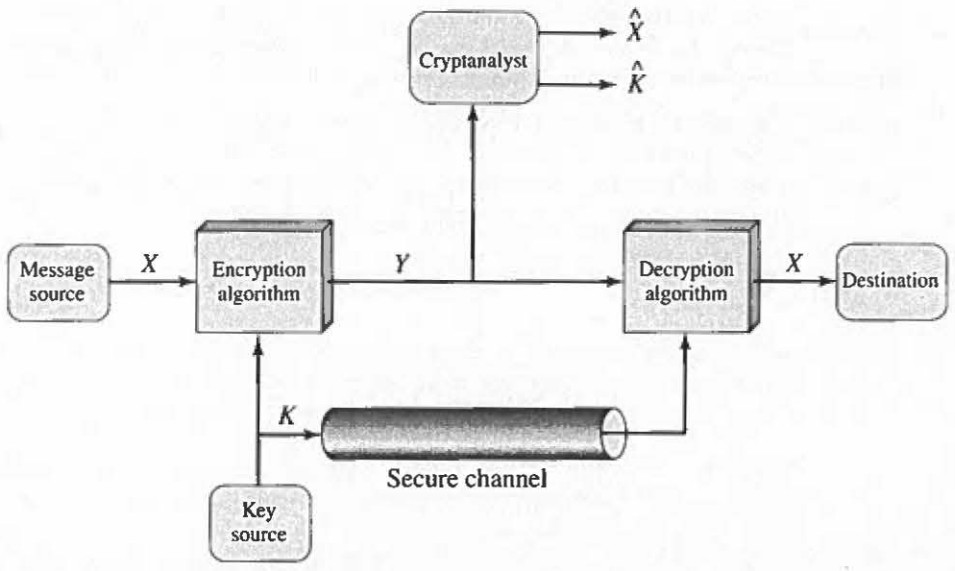


Figure 2.2 Model of Conventional Cryptosystem

JPMC Ex. 1041
Page 12

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 then 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.

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

Table 2.1 Types of Attacks on Encrypted Messages

Type of Attack	Known to Cryptanalyst
Ciphertext only	• Encryption algorithm • Ciphertext
Known plaintext	• Encryption algorithm • Ciphertext • One or more plaintext-ciphertext pairs formed with the secret key
Chosen plaintext	• Encryption algorithm • Ciphertext • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen ciphertext	• Encryption algorithm • Ciphertext • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen text	• 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

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.

Table 2.2 Average Time Required for Exhaustive Key Search

Key size (bits)	Number of alternative keys	Time required at 1 decryption/μs	Time required at 10^6 decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^{6} years

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 μs 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.

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: PHHW PH DIWHU WKH WRJD SDUWB
```

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

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

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]

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

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

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

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

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

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.

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.

[2]We define a mod n to be the remainder when a is divided by n. For example, 11 mod 7 = 4. See Chapter 4 for a further discussion of modular arithmetic.

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 ydrcp rfc rmey nyprw
6	jbbq jb xcqbo qeb qldx mxoqv
7	iaap ia wbpan pda pkcw lwnpu
8	hzzo hz vaozm ocz ojbv kvmot
9	gyyn gy uznyl nby niau julns
10	fxxm fx tymxk max mhzt itkmr
11	ewwl ew sxlwj lzw lgys hsjlq
12	dvvk dv rwkvi kyv kfxr grikp
13	cuuj cu qvjuh jxu jewq fqhjo
14	btti bt puitg iwt idvp epgin
15	assh as othsf hvs hcuo dofhm
16	zrrg zr nsgre gur gbtn cnegl
17	yqqf yq mrfqd ftq fasm bmdfk
18	xppe xp lqepc esp ezrl alcej
19	wood wo kpdob dro dyqk zkbdi
20	vnnc vn jocna cqn cxpj yjach
21	ummb um inbmz bpm bwoi xizbg
22	tlla tl hmaly aol avnh whyaf
23	skkz sk glzkx znk zumg vgxze
24	rjjy rj fkyjw ymj ytlf ufwyd
25	qiix qi ejxiv xli xske tevxc

Figure 2.3 Brute-Force Cryptanalysis of Caesar Cipher

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.

˜+Wμ"— Ω-O)≤4{∞‡¸ ë~Ω%ràu·¯Í ◊˜Z-
Ú≠2Ò#Åæð œ«q7¸Ωn·®3N◊Ú Œz'Y-ƒ∞Í[±Û_ èΩ,<NO¬±«ˇxã Åäfèü3Å
x}ö§kºÂ
_yÍ ^ΔÉ] ¸¤ J/˙iTê&ı 'c<uΩ-
ÄD(G WÄC-y_ïõÄW PÔı«ÎÜ†ç],¤¡ˇÌ^üÑπ˜≈ˇL˘9OgflO˜&Œ≤ ¬≤ ØÔ§″:
˘Œ!SGqèvo^ ú\¸S>h<-*6ø‡%x´˝|fiÓ#≈˜my%ˇ≥ñP<,ƒi Áj Å◊¿˝Zù-
Ω¨Õ¯6Œÿ{% „ΩÊó ¸ï π÷Áî˚úO2çSŷ´O-
2ÄflßiˇÍ /@^”∏K°ªPŒπ¸úé^´3∑˜ö˘ÔZÌ"Y¬ŸΩœY> Ω+eô/˙<K£¿*+˜"≤û˜
B ZøK˜Qßÿüƒ¸!ÒflÎzsS/]>ÈQ ü

Figure 2.4 Sample of Compressed Text

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				

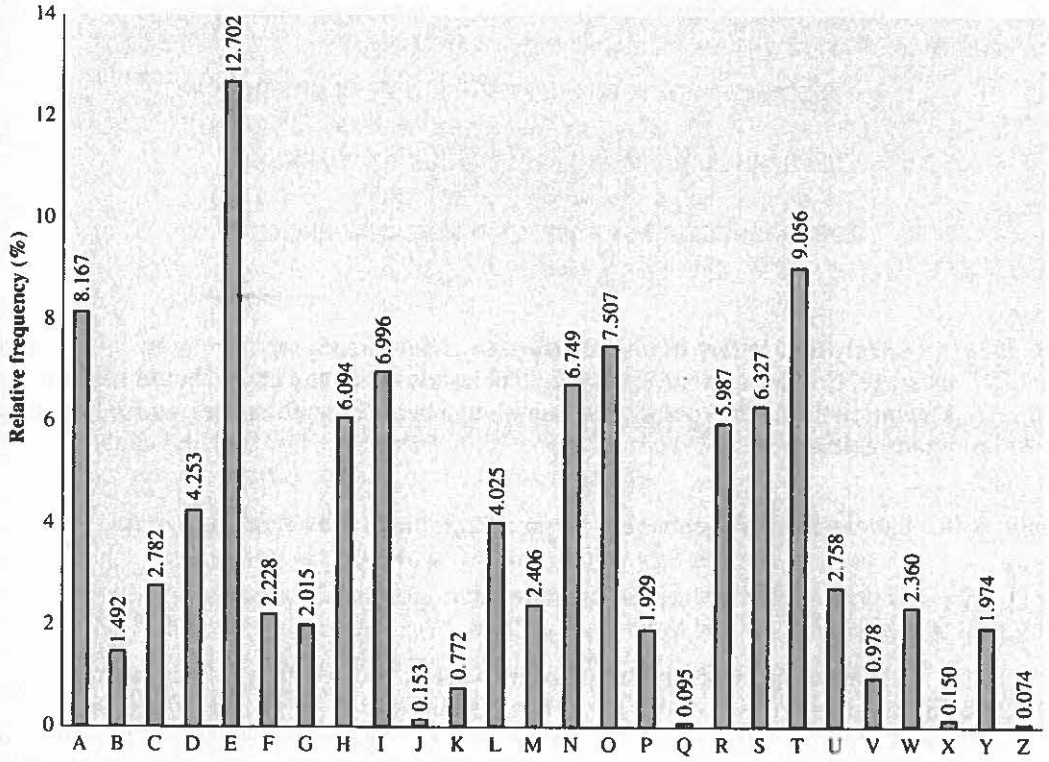


Figure 2.5 Relative Frequency of Letters in English Text

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.

JPMC Ex. 1041
Page 20

So far, then, we have

```
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
 t a        e  e te  a that e e a       a
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
   e t   ta t ha e ee  a e  th    t  a
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
 e  e e tat  e   the   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 × 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]

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 × 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 Brittanica* article on cryptology.[5] This is also the frequency distribution of any

[4]The book provides an absorbing account of a probable-word attack.

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

JPMC Ex. 1041
Page 22

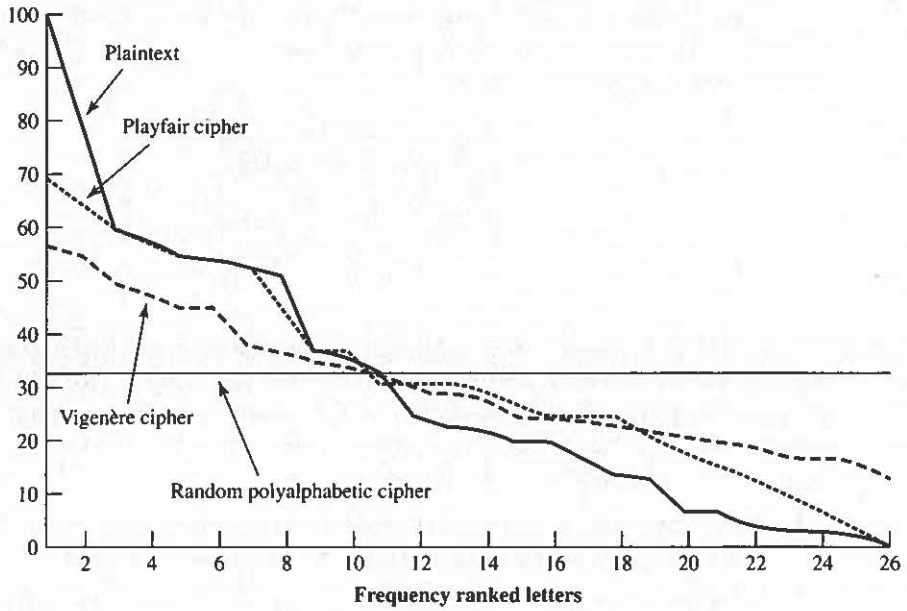


Figure 2.6 Relative Frequency of Occurrence of Letters

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.

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]

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

[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.

by m linear equations in which each character is assigned a numerical value (a = 0, b = 1, ... z = 25). For $m = 3$, the system can be described as follows:

$$c_1 = (k_{11}p_1 + k_{12}p_2 + k_{13}p_3) \bmod 26$$
$$c_2 = (k_{21}p_1 + k_{22}p_2 + k_{23}p_3) \bmod 26$$
$$c_3 = (k_{31}p_1 + k_{32}p_2 + k_{33}p_3) \bmod 26$$

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

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

or

$$\mathbf{C} = \mathbf{KP} \bmod 26$$

where **C** and **P** are column vectors of length 3, representing the plaintext and ciphertext, and **K** is a 3×3 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 $\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix}$. Then $\mathbf{K}\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix} = \begin{pmatrix} 375 \\ 819 \\ 486 \end{pmatrix} \bmod 26 = \begin{pmatrix} 11 \\ 13 \\ 18 \end{pmatrix}$ = LNS. Continuing in this fashion, the ciphertext for the entire plaintext is LNSHDLEWMTRW.

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

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

This is demonstrated as follows:

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

It is easily seen that if the matrix $\mathbf{K}^{-1}$ is applied to the ciphertext, then the plaintext is recovered. To explain how the inverse of a matrix is determined, we

JPMC Ex. 1041
Page 24

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

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

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 **A** has a nonzero determinant, then the inverse of the matrix is computed as $[\mathbf{A}^{-1}]_{ij} = (-1)^{i+j}(\mathrm{D}_{ji})/\det(\mathbf{A})$, where (D_{ij}) is the subdeterminant formed by deleting the ith row and the jth column of **A** and det(**A**) is the determinant of **A**. For our purposes, all arithmetic is done mod 26.

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

$$\mathbf{C} = \mathrm{E}(\mathbf{K}, \mathbf{P}) = \mathbf{KP} \bmod 26$$
$$\mathbf{P} = \mathrm{D}(\mathbf{K}, \mathbf{P}) = \mathbf{K}^{-1}\mathbf{C} \bmod 26 = \mathbf{K}^{-1}\mathbf{KP} = \mathbf{P}$$

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 $\mathbf{P}_j = \begin{pmatrix} p_{1j} \\ p_{2j} \\ \vdots \\ p_{mj} \end{pmatrix}$ and $\mathbf{C}_j = \begin{pmatrix} c_{1j} \\ c_{2j} \\ \vdots \\ c_{mj} \end{pmatrix}$ such that $\mathbf{C}_j = \mathbf{KP}_j$ for $1 \le j \le m$ and for some unknown key matrix **K**. Now define two $m \times m$ matrices $\mathbf{X} = (p_{ij})$ and $\mathbf{Y} = (c_{ij})$. Then we can form the matrix equation $\mathbf{Y} = \mathbf{KX}$. If **X** has an inverse, then we can determine $\mathbf{K} = \mathbf{YX}^{-1}$. If **X** is not invertible, then a new version of **X** can be formed with additional plaintext-ciphertext pairs until an invertible **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 PQCFKU. Thus, we know that $\mathbf{K}\begin{pmatrix} 5 \\ 17 \end{pmatrix} \bmod 26 = \begin{pmatrix} 15 \\ 16 \end{pmatrix}$; $\mathbf{K}\begin{pmatrix} 8 \\ 3 \end{pmatrix} \bmod 26 = \begin{pmatrix} 2 \\ 5 \end{pmatrix}$; and $\mathbf{K}\begin{pmatrix} 0 \\ 24 \end{pmatrix} \bmod 26 = \begin{pmatrix} 10 \\ 20 \end{pmatrix}$. Using the first two plaintext-ciphertext pairs, we have

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

[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.

The inverse of **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.

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:          deceptivedeceptivedeceptive
plaintext:    wearediscoveredsaveyourself
ciphertext:   ZICVTWQNGRZGVTWAVZHCQYGLMGJ
```

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

The strength of this cipher is that there are multiple ciphertext letters for each plaintext letter, one for each unique letter of the keyword. Thus, the letter frequency information is obscured. However, not all knowledge of the plaintext structure is lost. For example, Figure 2.6 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.

JPMC Ex. 1041
Page 26

Table 2.3 The Modern Vigenère Tableau

Key	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**
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

JPMC Ex. 1041
Page 27

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

JPMC Ex. 1041
Page 28

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]

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 = ith binary digit of plaintext
k_i = ith binary digit of key
c_i = ith 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 × 27. Consider the ciphertext

```
ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
```

[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.

JPMC Ex. 1041
Page 29

We now show two different decryptions using two different keys:

```
ciphertext: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
key:        pxlmvmsydofuyrvzwc tnlebnecvgdupahfzzlmnyih
plaintext:  mr mustard with the candlestick in the hall

ciphertext: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
key:        mfugpmiydgaxgoufhklllmhsqdqogtewbqfgyovuhwt
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
```

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]

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.

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

[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.

JPMC Ex. 1041
Page 32

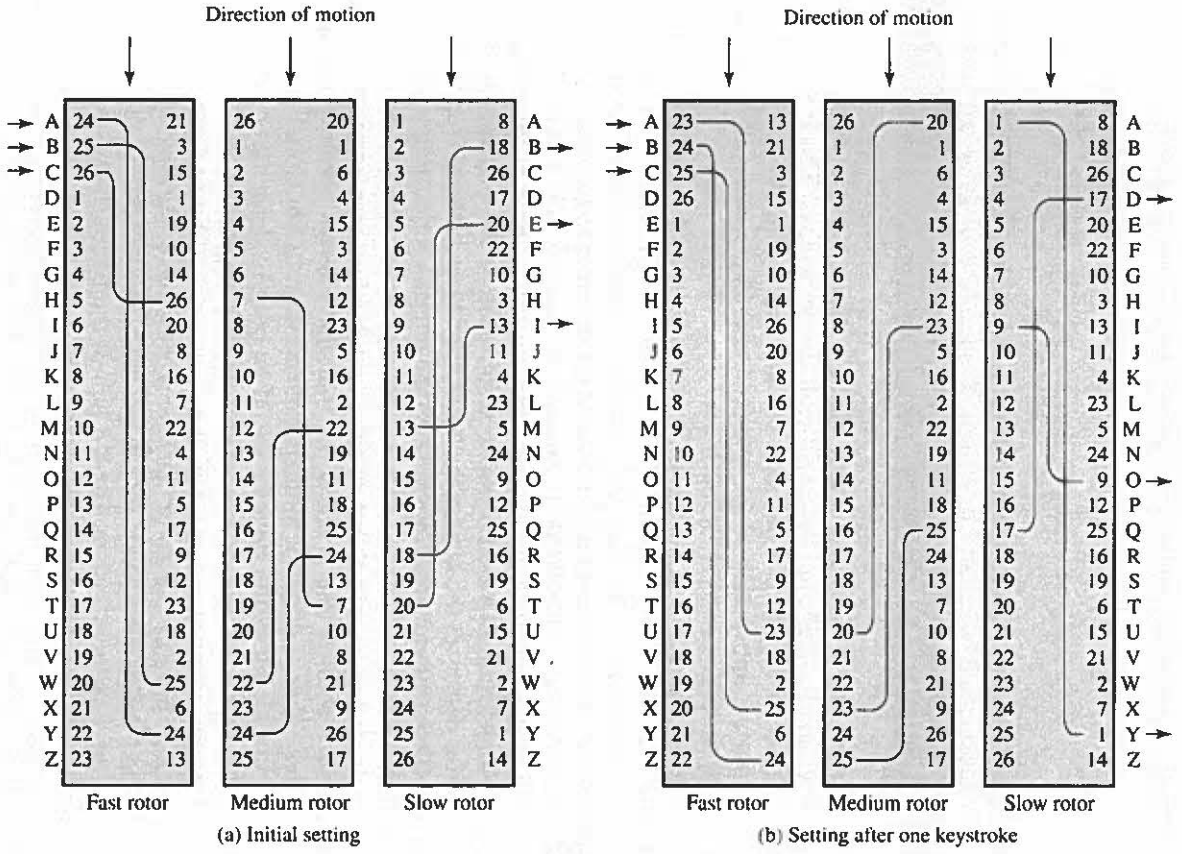


Figure 2.7 Three-Rotor Machine with Wiring Represented by Numbered Contacts

JPMC Ex. 1041
Page 33

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]:

> 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.

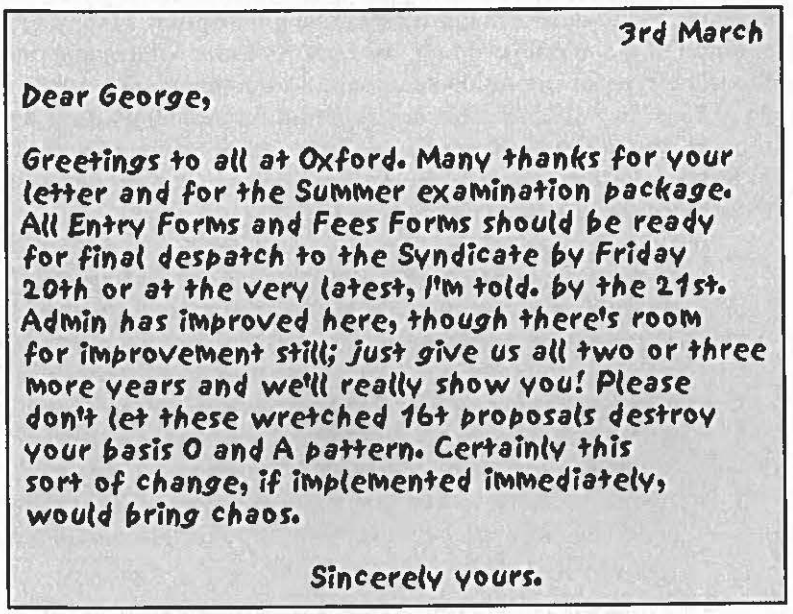
A plaintext message may be hidden in one of two ways. The methods of steganography conceal the existence of the message, whereas the methods of cryptography render the message unintelligible to outsiders by various transformations of the text.[10]

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.

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.

[10] *Steganography* was an obsolete word that was revived by David Kahn and given the meaning it has today [KAHN96].


3rd March

Dear George,

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

Sincerely yours.

Figure 2.8 A Puzzle for Inspector Morse
(From The Silent World of Nicholas Quinn, *by Colin Dexter)*

- **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.

2.6 RECOMMENDED READING AND WEB SITES

For anyone interested in the history of code making and code breaking, the book to read [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 [GARR01] 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.

GARR01 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.

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



Recommended Web Sites:

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

- **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.

2.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

block cipher	cryptology	polyalphabetic cipher
brute-force attack	deciphering	rail fence cipher
Caesar cipher	decryption	single-key encryption
cipher	enciphering	steganography
ciphertext	encryption	stream cipher
computationally secure	Hill cipher	symmetric encryption
conventional encryption	monoalphabetic cipher	transposition cipher
cryptanalysis	one-time pad	unconditionally secure
cryptographic system	plaintext	Vigenère cipher
cryptography	Playfair 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, knows as the affine Caesar cipher, has the following form: For each plaintext letter p, substitute the ciphertext letter C:

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

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‡.)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?

Chapter 13 (some)
Chapter 9 (some)
Chapter 11 (entire)

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

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 × 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 QZJKA 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 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 × 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*.

JPMC Ex. 1041
Page 40

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:

7876565434321123434565678788787656 54
34321123434565678788787656544332 11234

   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?

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

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

Figure 2.9 A Puzzle for Lord Peter

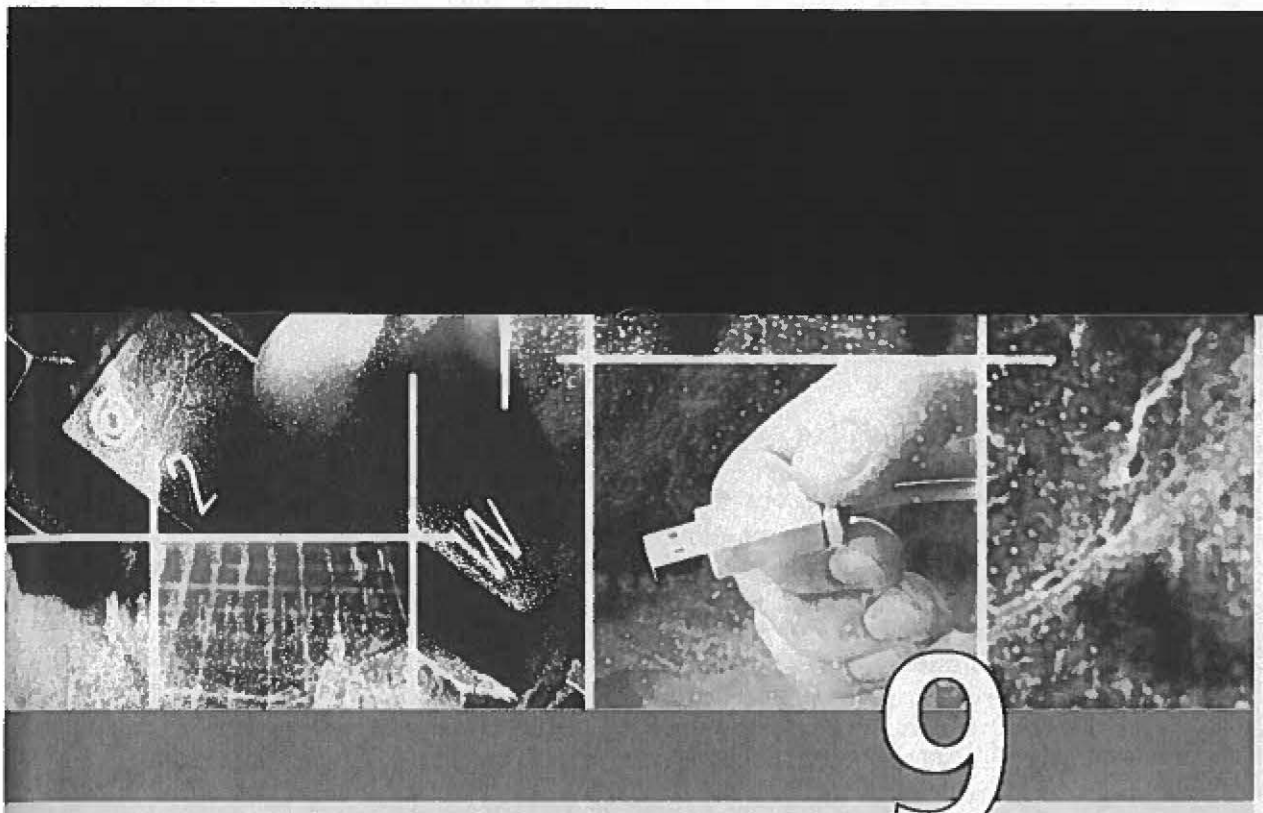
JPMC Ex. 1041
Page 41

order of likelihood. It would be good if your user interface allowed the user to specify "give me the top 10 possible plaintexts".

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 2×2 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 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

JPMC Ex. 1041
Page 43

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

JPMC Ex. 1041
Page 44

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.

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

JPMC Ex. 1041
Page 45

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]

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.

---

[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].

JPMC Ex. 1041
Page 46

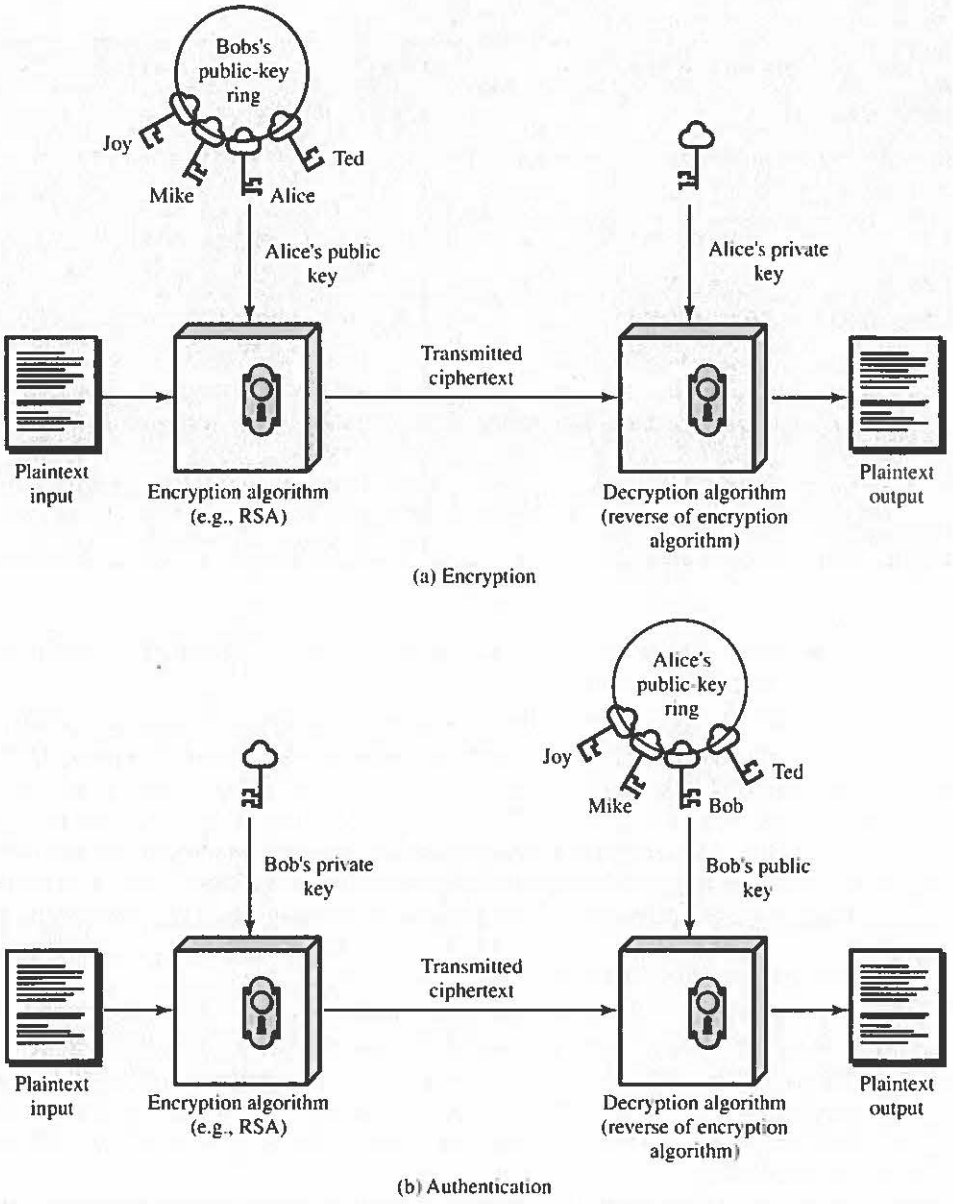


Figure 9.1 Public-Key Cryptography

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.

JPMC Ex. 1041
Page 47

Table 9.1 Conventional and Public-Key Encryption

Conventional Encryption	Public-Key Encryption
Needed to Work:	**Needed to Work:**
1. The same algorithm with the same key is used for encryption and decryption.	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 share the algorithm and the key.	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:**
1. The key must be kept secret.	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.	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.	3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

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.

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, $X = [X_1, X_2, \ldots, X_M]$. The M elements of X are letters in

[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.

JPMC Ex. 1041
Page 48

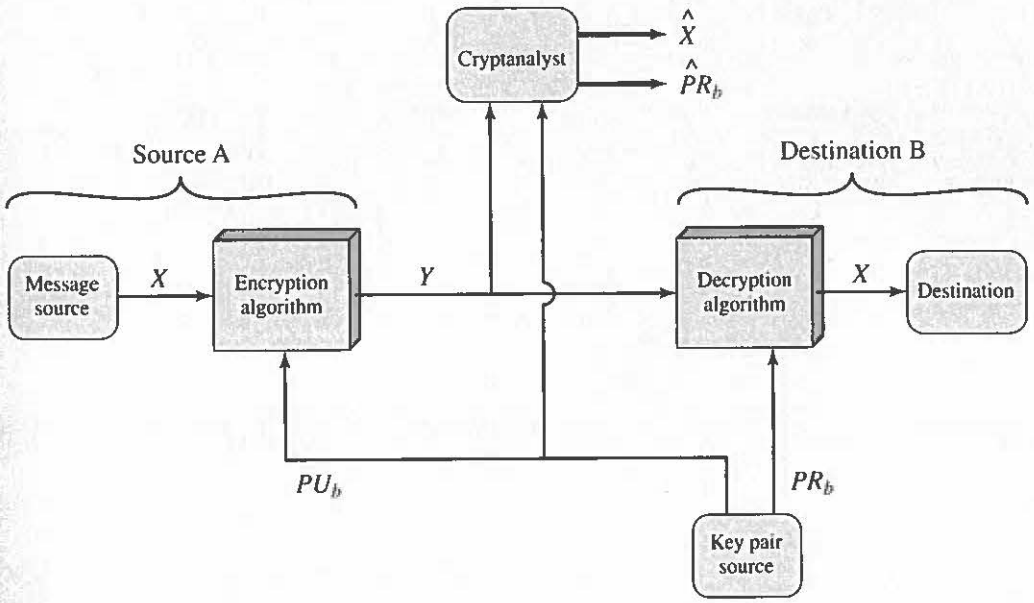


Figure 9.2 Public-Key Cryptosystem: Secrecy

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. PR_b is known only to B, whereas PU_b is publicly available and therefore accessible by A.

With the message X and the encryption key PU_b as input, A forms the ciphertext $Y = [Y_1, Y_2, \ldots, Y_N]$:

$$Y = \mathrm{E}(PU_b, X)$$

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

$$X = \mathrm{D}(PR_b, Y)$$

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

JPMC Ex. 1041
Page 49

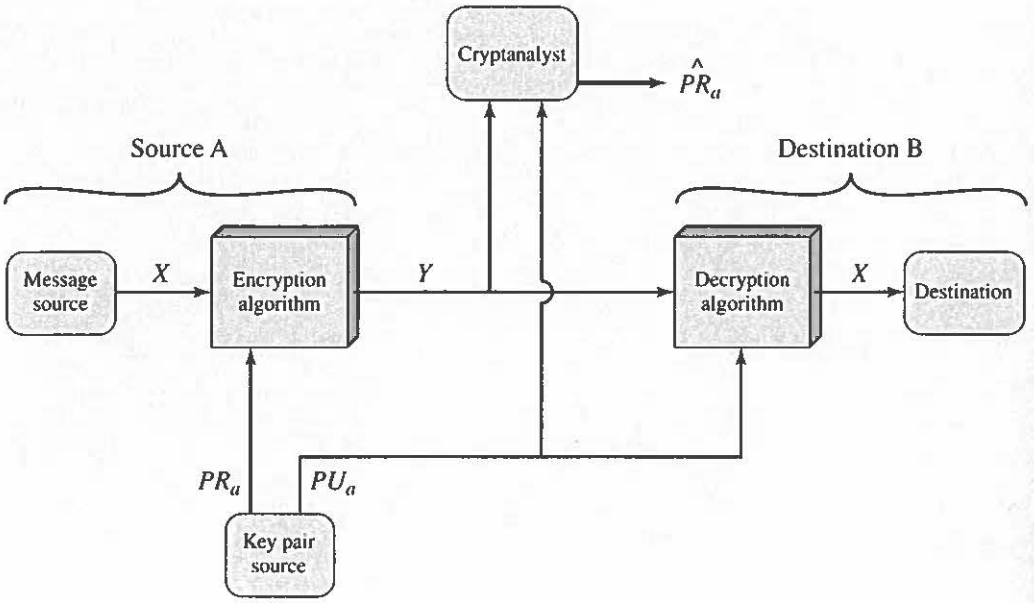


Figure 9.3 Public-Key Cryptosystem: Authentication

provides confidentiality, Figures 9.1b and 9.3 show the use of public-key encryption to provide authentication:

$$Y = \mathrm{E}(PR_a, X)$$
$$X = \mathrm{D}(PU_a, Y)$$

In this case, A prepares a message to B and encrypts it using A's private key before transmitting it. B can decrypt the message using A's public key. Because the message was encrypted using A's private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a *digital signature*. In addition, it is impossible to alter the message without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity.

In the preceding scheme, the entire message is encrypted, which, although validating both author and contents, requires a great deal of storage. Each document must be kept in plaintext to be used for practical purposes. A copy also must be stored in ciphertext so that the origin and contents can be verified in case of a dispute. A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted with the sender's private key, it serves as a signature that verifies origin, content, and sequencing. Chapter 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

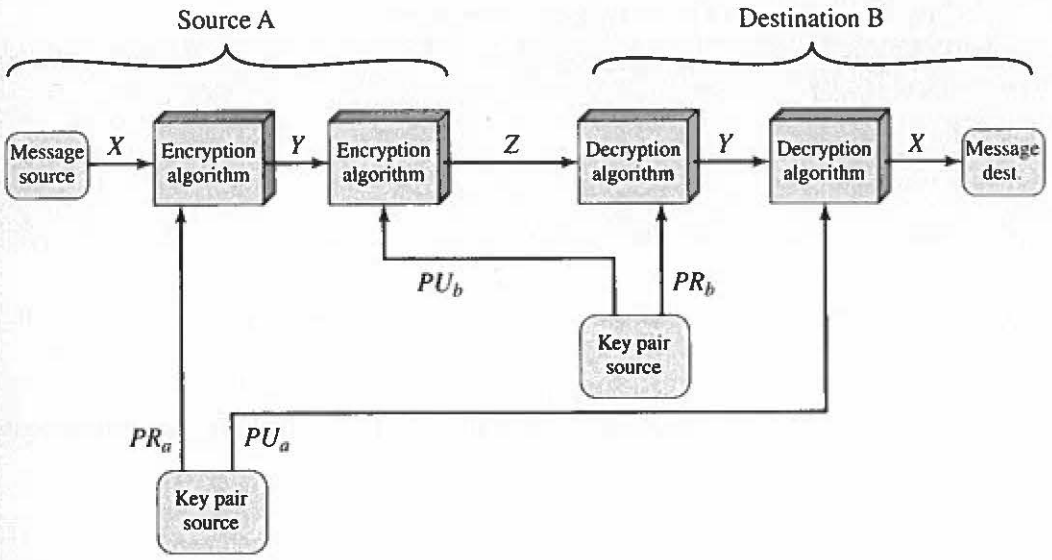


Figure 9.4 Public-Key Cryptosystem: Authentication and Secrecy

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):

$$Z = \mathrm{E}(PU_b, \mathrm{E}(PR_a, X))$$
$$X = \mathrm{D}(PU_a, \mathrm{D}(PR_b, Z))$$

In this case, we begin as before by encrypting a message, using the sender's private key. This provides the digital signature. Next, we encrypt again, using the receiver's public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided. The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

Applications for Public-Key Cryptosystems

Before proceeding, we need to clarify one aspect of public-key cryptosystems that is otherwise likely to lead to confusion. Public-key systems are characterized by the use of a cryptographic 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.

JPMC Ex. 1041
Page 51

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

- **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.

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:

$$C = \mathrm{E}(PU_b, M)$$

3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

$$M = \mathrm{D}(PR_b, C) = \mathrm{D}[PR_b, \mathrm{E}(PU_b, M)]$$

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:

$$M = \mathrm{D}[PU_b, \mathrm{E}(PR_b, M)] = \mathrm{D}[PR_b, \mathrm{E}(PU_b, M)]$$

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.

JPMC Ex. 1041
Page 52

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:

$$Y = f(X) \qquad \text{easy}$$
$$X = f^{-1}(Y) \qquad \text{infeasible}$$

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

$$Y = f_k(X) \qquad \text{easy, if } k \text{ and } X \text{ are known}$$
$$X = f_k^{-1}(Y) \qquad \text{easy, if } k \text{ and } Y \text{ are known}$$
$$X = f_k^{-1}(Y) \qquad \text{infeasible, if } Y \text{ is known but } k \text{ is not known}$$

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

[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).

JPMC Ex. 1041
Page 53

mentioned earlier, public-key encryption is currently confined to key management and signature applications.

Another form of attack is to find some way to compute the private key given the public key. To date, it has not been mathematically proven that this form of attack is infeasible for a particular public-key algorithm. Thus, any given algorithm, including the widely used RSA algorithm, is suspect. The history of cryptanalysis shows that a problem that seems insoluble from one perspective can be found to have a solution if looked at in an entirely different way.

Finally, there is a form of attack that is peculiar to public-key systems. This is, in essence, a probable-message attack. Suppose, for example, that a message were to be sent that consisted solely of a 56-bit DES key. An 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.

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

[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.

JPMC Ex. 1041
Page 54

algorithm with a public key of $PU = \{e, n\}$ and a private key of $PR = \{d, n\}$. 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 $C^d \bmod n$ 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

$$ed \bmod \phi(n) = 1 \tag{9.1}$$

This is equivalent to saying

$$ed \equiv 1 \bmod \phi(n)$$
$$d \equiv e^{-1} \bmod \phi(n)$$

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)
$d \equiv e^{-1} (\bmod \phi(n))$	(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 $n = pq = 17 \times 11 = 187$.

JPMC Ex. 1041
Page 55

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	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d = e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Figure 9.5 The RSA Algorithm

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).

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

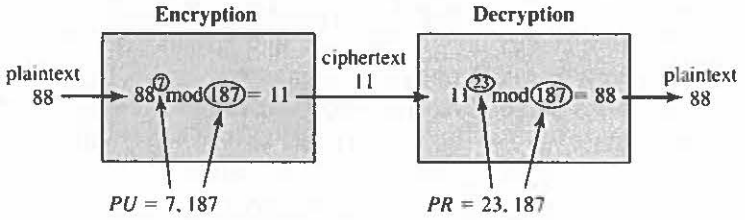


Figure 9.6 Example of RSA Algorithm

$M = 88$. For encryption, we need to calculate $C = 88^7 \bmod 187$. Exploiting the properties of modular arithmetic, we can do this as follows:

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$
$$88^1 \bmod 187 = 88$$
$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$
$$88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$$
$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894{,}432 \bmod 187 = 11$$

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 $x \bmod n$,

JPMC Ex. 1041
Page 57

```
c ← 0; f ← 1
for i ← k downto 0
    do  c ← 2 × c
        f ← (f × f) mod n
    if  b_i = 1
        then c ← c + 1
             f ← (f × a) mod n
return f
```

Note: The integer b is expressed as a binary number $b_k b_{k-1} \ldots b_0$

Figure 9.7 Algorithm for Computing $a^b \bmod n$

$x^2 \bmod n$, $x^4 \bmod n$, 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 $b_k b_{k-1} \ldots b_0$, then we have

$$b = \sum_{b_i \neq 0} 2^i$$

Therefore,

$$a^b = a^{\left(\sum_{b_i \neq 0} 2^i\right)} = \prod_{b_i \neq 0} a^{(2^i)}$$

$$a^b \bmod n = \left[\prod_{b_i \neq 0} a^{(2^i)}\right] \bmod n = \left(\prod_{b_i \neq 0} \left[a^{(2^i)} \bmod n\right]\right) \bmod n$$

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.

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

[5]The algorithm has a long history; this particular pseudocode expression is from [CORM01].

JPMC Ex. 1041
Page 58

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 $M^3 \bmod (n_1 n_2 n_3)$. 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 $M = C^d \bmod n$. Let us define the following intermediate results:

$$V_p = C^d \bmod p \qquad V_q = C^d \bmod q$$

Following the CRT, Equation (8.8), define the quantities:

$$X_p = q \times (q^{-1} \bmod p) \qquad X_q = p \times (p^{-1} \bmod q)$$

The CRT then shows, using Equation (8.9), that

$$M = (V_p X_p + V_q X_q) \bmod n$$

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:

$$V_p = C^d \bmod p = C^{d \bmod (p-1)} \bmod p \qquad V_q = C^d \bmod q = C^{d \bmod (q-1)} \bmod q$$

The quantities $d \bmod (p - 1)$ and $d \bmod (q - 1)$ can be precalculated. The end result is that the calculation is approximately four times as fast as evaluating $M = C^d \bmod n$ 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

JPMC Ex. 1041
Page 59

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

JPMC Ex. 1041
Page 60

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.

JPMC Ex. 1041
Page 61

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

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.

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.

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].

JPMC Ex. 1041
Page 62

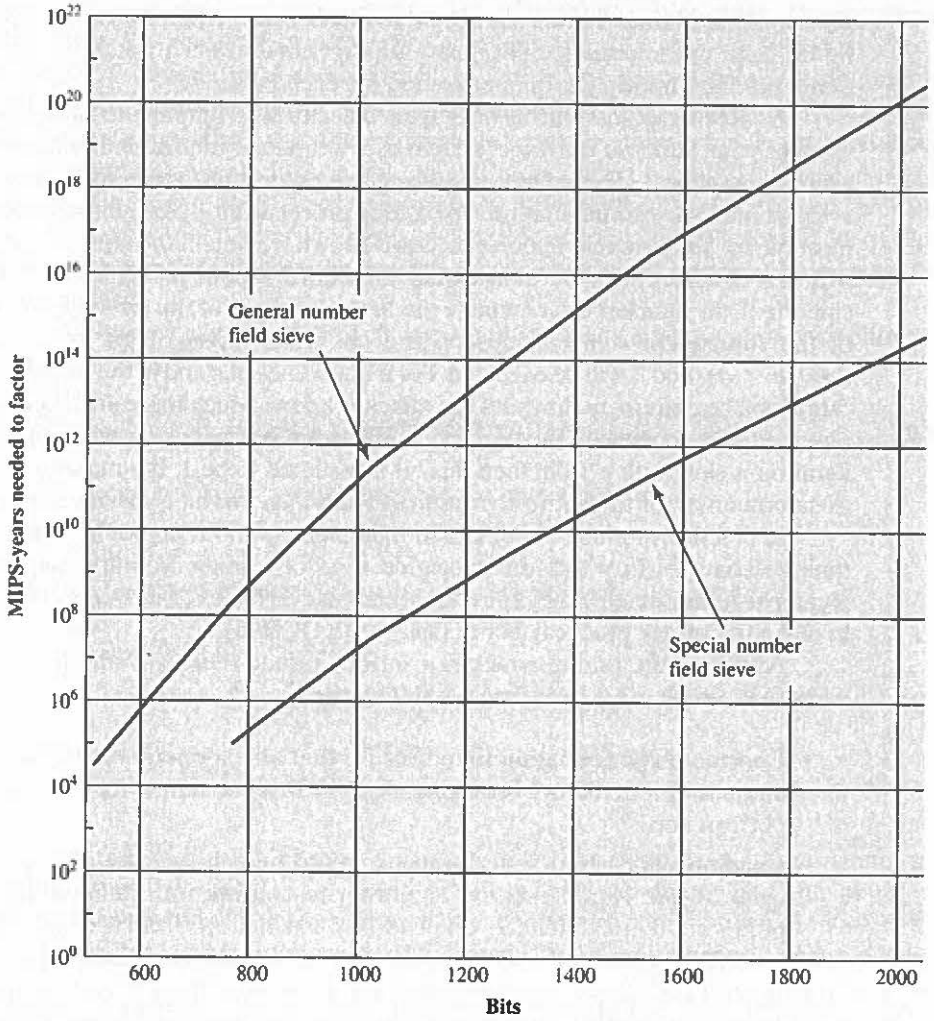


Figure 9.8 MIPS-years Needed to Factor

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

JPMC Ex. 1041
Page 63

not run in fixed time. In this algorithm, modular exponentiation is accomplished bit by bit, with one modular multiplication performed at each iteration and an additional modular multiplication performed for each 1 bit.

As Kocher points out in his paper, the attack is simplest to understand in an extreme case. Suppose the target system uses a modular multiplication function that is very fast in almost all cases but in a few cases takes much more time than an entire average modular exponentiation. The attack proceeds bit-by-bit starting with the leftmost bit, b_k. Suppose that the first j bits are known (to obtain the entire exponent, start with $j = 0$ and repeat the attack until the entire exponent is known). For a given ciphertext, the attacker can complete the first j iterations of the **for** loop. The operation of the subsequent step depends on the unknown exponent bit. If the bit is set, $d \leftarrow (d \times a) \bmod n$ will be executed. For a few values of a and d, the modular multiplication will be extremely slow, and the attacker knows which these are. Therefore, if the observed time to execute the decryption algorithm is always slow when this particular iteration is slow with a 1 bit, then this bit is assumed to be 1. If a number of observed execution times for the entire algorithm are fast, then this bit is assumed to be 0.

In practice, modular exponentiation implementations do not have such extreme timing variations, in which the execution time of a single iteration can exceed the mean execution time of the entire algorithm. Nevertheless, there is enough variation to make this attack practical. For details, see [KOCH96].

Although the timing attack is a serious threat, there are simple countermeasures that can be used, including the following:

- **Constant exponentiation time:** Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.
- **Random delay:** Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. Kocher points out that if defenders 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:

$$\mathrm{E}(PU, M_1) \times \mathrm{E}(PU, M_2) = \mathrm{E}(PU, [M_1 \times M_2]) \tag{9.2}$$

We can decrypt $C = M^e \bmod n$ 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

[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.

JPMC Ex. 1041
Page 65

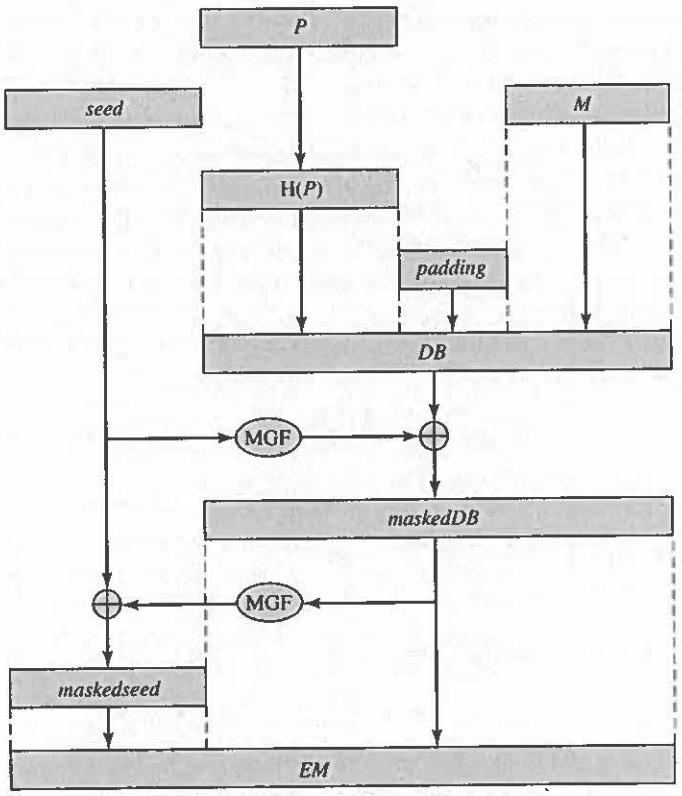


Figure 9.9 Encryption Using Optimal Assymetric Encryption Padding (OAEP)

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.

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].

JPMC Ex. 1041
Page 66

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)	private key	RSA
digital signature	public key	time complexity
key exchange	public key cryptography	timing attack
one-way function	public key cryptosystems	trapdoor one-way function
optimal asymmetric encryption padding (OAEP)	public key encryption	

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 $1 \leq x_i, y_i, z_i \leq N$. Function f_1 can be represented by a vector M1 of length N, in which the kth 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:

$$f_3(f_2(f_1(k), p), k) = p \qquad \text{for all } k, p \text{ with } 1 \leq k, p \leq N$$

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 ith element of M1 corresponds to $k = i$. The ith row of M2 corresponds to $x = i$; the jth column of M2 corresponds to $p = j$. The ith row of M3 corresponds to $z = i$; the jth 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:

a. $p = 3; q = 11, e = 7; M = 5$
b. $p = 5; q = 11, e = 3; M = 9$
c. $p = 7; q = 11, e = 17; M = 8$
d. $p = 11; q = 13, e = 11; M = 7$
e. $p = 17; q = 31, e = 7; M = 2$. *Hint:* Decryption is not as hard as you think; use some finesse.

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 trail 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

JPMC Ex. 1041
Page 68

25 (A → 0, . . . , Z → 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.

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, . . . , 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 $M = 2, 3, 4, 5, 6, \ldots$"

"But why, Holmes?" Dr. Watson apparently didn't understand. "Why do you think that the resulting sequence $2^K \bmod N$, $3^K \bmod N$, $4^K \bmod N$, . . . 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.

JPMC Ex. 1041
Page 69

9.13 Consider the following scheme:

1. Pick an odd number, E.
2. Pick two prime numbers, P and Q, where $(P - 1)(Q - 1) - 1$ is evenly divisible by E.
3. Multiply P and Q to get N.
4. Calculate $D = \dfrac{(P - 1)(Q - 1)(E - 1) + 1}{E}$.

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 $C = M^N \bmod N$.
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 M1, M2, and M3 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: $(A, \mathrm{E}(PU_b, [M, A]), B)$.
2. B acknowledges receipt by sending to A the following block: $(B, \mathrm{E}(PU_a, [M, B]), A)$.

"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: $(A, \mathrm{E}(PU_b, M), B)$.
2. B acknowledges receipt by sending to A the block: $(B, \mathrm{E}(PU_a, M), A)$.

"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.

JPMC Ex. 1041
Page 70

1. f ← 1; T ← a; E ← b
2. **if** odd(e) **then** f ← f × T
3. E ← ⌊E/2⌋
4. T ← T × 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 want 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:* $x^{i+1} = x^i \times x$.

b. Develop an algorithm that requires only $n + 1$ multiplications and $n + 1$ additions. *Hint:* $P(x) = a_0 + x \times q(x)$, where $q(x)$ is a polynomial of degree $(n - 1)$.

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 Why is it a requirement that $m > \sum_{i=1}^{n} a_i'$?

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, $\phi(pq) = (p - 1)(q - 1)$. 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

$$M^{k\phi(n)+1} \bmod n = M^{k(p-1)(q-1)+1} \bmod n = M \qquad (9.3)$$

JPMC Ex. 1041
Page 71

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, than $a^{\phi(n)} \bmod n = 1$.

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:

$$\begin{aligned}
M^{k(p-1)(q-1)+1} \bmod p &= [(M)M^{k(p-1)(q-1)}] \bmod p \\
&= [(M)(M^{(p-1)})^{k(q-1)}] \bmod p \\
&= [(M)(M^{\phi(p)})^{k(q-1)}] \bmod p \\
&= (M \bmod p) \times [(M^{\phi(p)}) \bmod p]^{k(q-1)} \\
&= (M \bmod p) \times (1)^{k(q-1)} \qquad \text{(by Euler's theorem)} \\
&= M \bmod p
\end{aligned}$$

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 $M^{k\phi(n)+1} \bmod n = M^{k(p-1)(q-1)+1} \bmod n = M$.

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

JPMC Ex. 1041
Page 72

the algorithm takes at most $f(n)$ steps. Thus, for a given size of input and a given processor speed, the time complexity is an upper bound on the execution time.

There are several ambiguities here. First, the definition of a step is not precise. A step could be a single operation of a Turing machine, a single processor machine instruction, a single high-level language machine instruction, and so on. However, these various definitions of step should all be related by simple multiplicative constants. For very large values of n, these constants are not important. What is important is how fast the relative execution time is growing. For example, if we are concerned about whether to use 50-digit ($n = 10^{50}$) or 100-digit ($n = 10^{100}$) keys for RSA, it is not necessary (or really possible) to know exactly how long it would take to break each size of key. Rather, we are interested in ballpark figures for level of effort and in knowing how much extra relative effort is required for the larger key size.

A second issue is that, generally speaking, we cannot pin down an exact formula for $f(n)$. We can only approximate it. But again, we are primarily interested in the rate of change of $f(n)$ as n becomes very large.

There is a standard mathematical notation, known as the "big-O" notation, for characterizing the time complexity of algorithms that is useful in this context. The definition is as follows: $f(n) = O(g(n))$ if and only if there exist two numbers a and M such that

$$|f(n)| \leq a \times |g(n)|, \qquad n \geq M \tag{9.4}$$

An example helps clarify the use of this notation. Suppose we wish to evaluate a general polynomial of the form

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

The following simple-minded algorithm is from [POHL81]:

```
algorithm P1;
   n, i, j: integer; x, polyval: real;
   a, S: array [0..100] of real;
   begin
      read(x, n);
      for i := 0 upto n do
      begin
         S[i] := 1; read(a[i]);
         for j := 1 upto i do S[i] := x × S[i];
         S[i] := a[i] × S[i]
      end;
      polyval := 0;
      for i := 0 upto n do polyval := polyval + S[i];
      write ('value at', x, 'is', polyval)
   end.
```

In this algorithm, each subexpression is evaluated separately. Each $S[i]$ requires $(i + 1)$ multiplications: i multiplications to compute $S[i]$ and one to multiply by $a[i]$. Computing all n terms requires

$$\sum_{i=0}^{n} (i + 1) = \frac{(n + 2)(n + 1)}{2}$$

multiplications. There are also $(n + 1)$ additions, which we can ignore relative to the much larger number of multiplications. Thus, the time complexity of this algorithm is $f(n) = (n + 2)(n + 1)/2$. We now show that $f(n) = O(n^2)$. From the definition of Equation (9.4), we

JPMC Ex. 1041
Page 73

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 $n = k + 1$,

$$\begin{aligned}\frac{(n+2)(n+1)}{2} &= \frac{(k+3)(k+2)}{2} \\ &= \frac{(k+2)(k+1)}{2} + k + 2 \\ &\leq k^2 + k + 2 \\ &\leq k^2 + 2k + 1 = (k+1)^2 = n^2\end{aligned}$$

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 $h(n)$

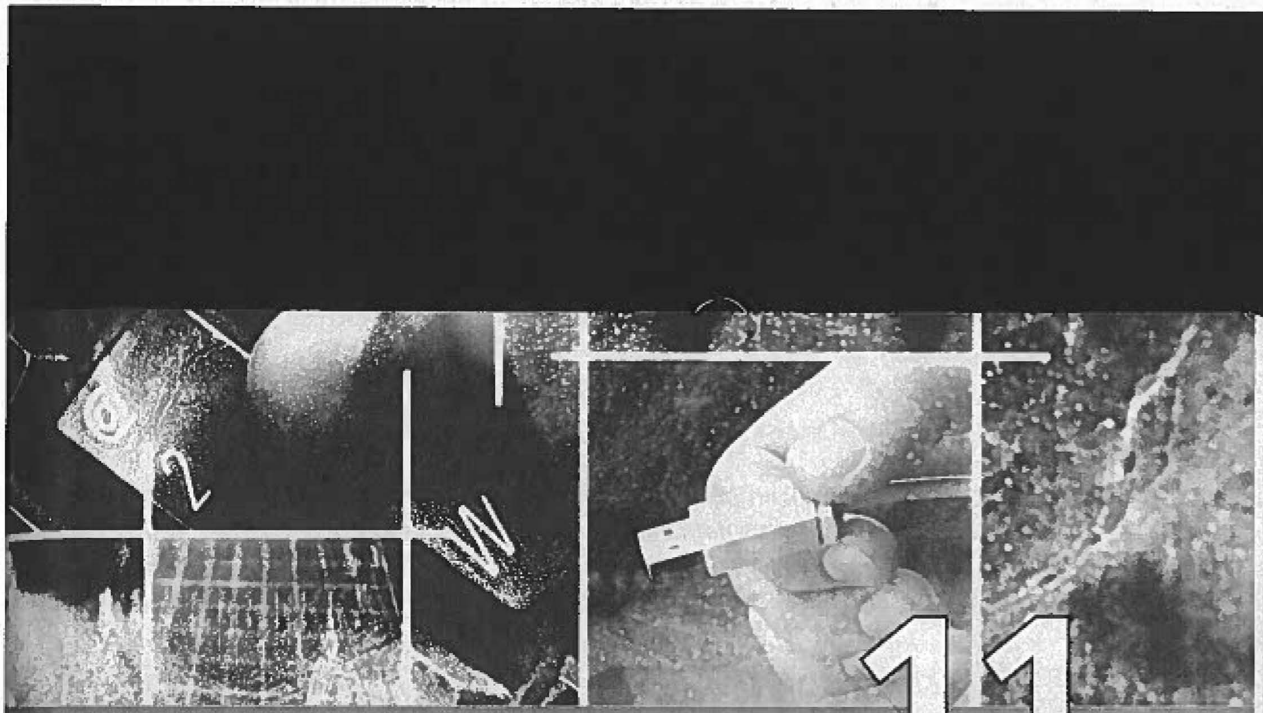
Generally, a problem that can be solved in polynomial time is considered feasible, whereas anything worse than polynomial time, especially exponential time, is considered infeasible. But you must be careful with these terms. First, if the size of the input is small enough, even very complex algorithms become feasible. Suppose, for example, that you have a system that can execute 10^{12} 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.

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 [$O(2^n)$].

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^n	39	10^{12}
$n!$	15	10^{12}

JPMC Ex. 1041
Page 74



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

JPMC Ex. 1041
Page 75

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.

JPMC Ex. 1041
Page 76

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.

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.

JPMC Ex. 1041
Page 77

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.

In addition, we may say that B is assured that the message was generated by A. Why? The message must have come from A because A is the only other party that possesses K and therefore the only other party with the information necessary to construct ciphertext that can be decrypted with K. Furthermore, if M is recovered, B knows that none of the bits of M have been altered, because an opponent that does not know K would not know how to alter bits in the ciphertext to produce desired changes in the plaintext.

So we may say that symmetric encryption provides authentication as well as confidentiality. However, this flat statement needs to be qualified. Consider exactly what is happening at B. Given a decryption function D and a secret key K, the destination will accept *any* input X and produce output $Y = D(K, X)$. If X is the ciphertext of a legitimate message M produced by the corresponding encryption function, then Y is some plaintext message M. Otherwise, Y will likely be a meaningless sequence of bits. There may need to be some automated means of determining at B whether Y is legitimate plaintext and therefore must have come from A.

The implications of the line of reasoning in the preceding paragraph are profound from the point of view of authentication. Suppose the message M can be any arbitrary bit pattern. In that case, there is no way to determine automatically, at the

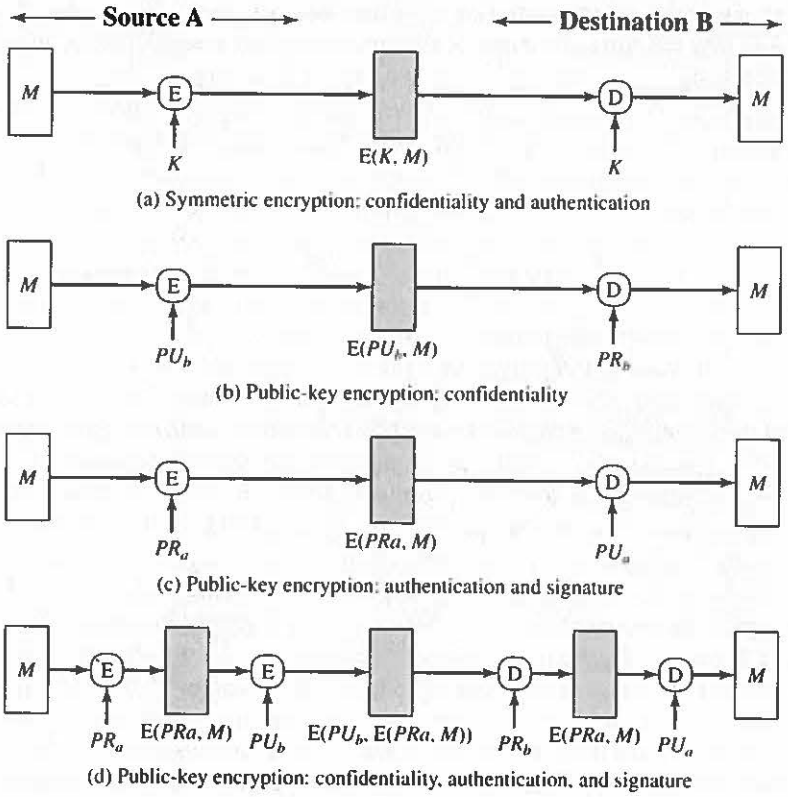


Figure 11.1 Basic Uses of Message Encryption

destination, whether an incoming message is the ciphertext of a legitimate message. This conclusion is incontrovertible: If M can be any bit pattern, then regardless of the value of X, the value $Y = D(K, X)$ is *some* bit pattern and therefore must be accepted as authentic plaintext.

Thus, in general, we require that only a small subset of all possible bit patterns 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
```

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:

```
ytuqrndufpwkyvhfzlcumlgolbbwhttqdnreabdaspwrwp
```

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.

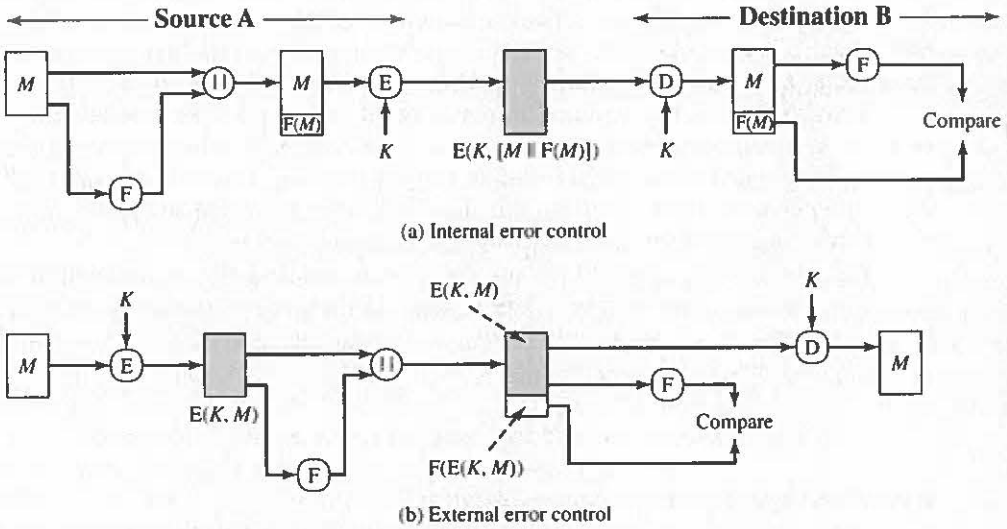


Figure 11.2 Internal and External Error Control

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.

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.

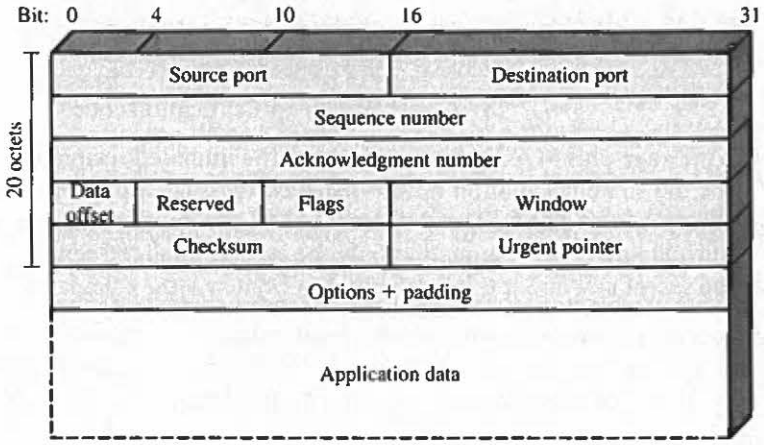


Figure 11.3 TCP Segment

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.

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.

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: $\text{MAC} = \text{C}(K, M)$, where

$$\begin{aligned} M &= \text{input message} \\ \text{C} &= \text{MAC function} \\ K &= \text{shared secret key} \\ \text{MAC} &= \text{message authentication code} \end{aligned}$$

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

[1] This is not the way in which digital signatures are constructed, as we shall see, but the principle is the same.

JPMC Ex. 1041
Page 82

Table 11.1 Confidentiality and Authentication Implications of Message Encryption (see Figure 11.1)

A → 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 → 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 → B: E(PR_a, M)

- Provides authentication and signature
  - —Only A has PR_a to encrypt
  - —Has not been altered in transit
  - —Requires some formatting/redundancy
  - —Any party can use PU_a to verify signature

(c) Public-key encryption: authentication and signature

A → B: E(PU_b, E(PR_a, M))

- Provides confidentiality because of PU_b
- Provides authentication and signature because of PR_a

(d) Public-key encryption: confidentiality, authentication, and signature

know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.

2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
3. If the message includes a sequence number (such as is used with HDLC, X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

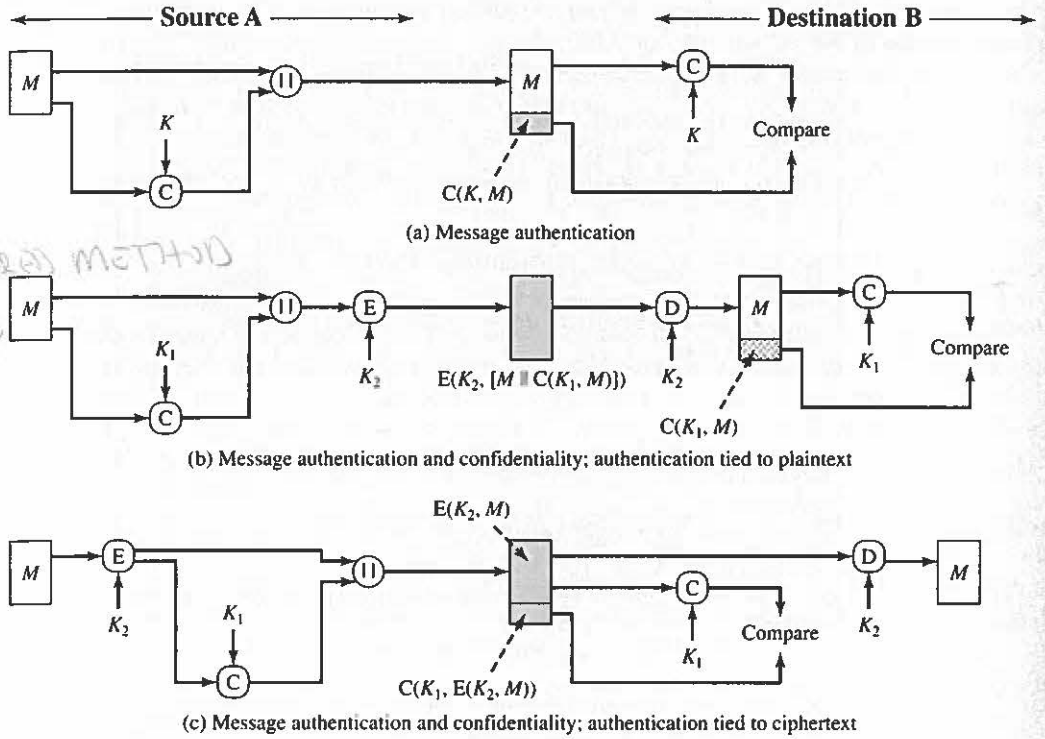


Figure 11.4 Basic Uses of Message Authentication Code (MAC)

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.

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.

JPMC Ex. 1041
Page 85

Table 11.2 Basic Uses of Message Authentication Code C (see Figure 11.4)

$A \rightarrow B: M \| C(K, M)$
- Provides authentication
  —Only A and B share K

(a) Message authentication

$A \rightarrow B: E(K_2, [M \| C(K, M)])$
- 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.

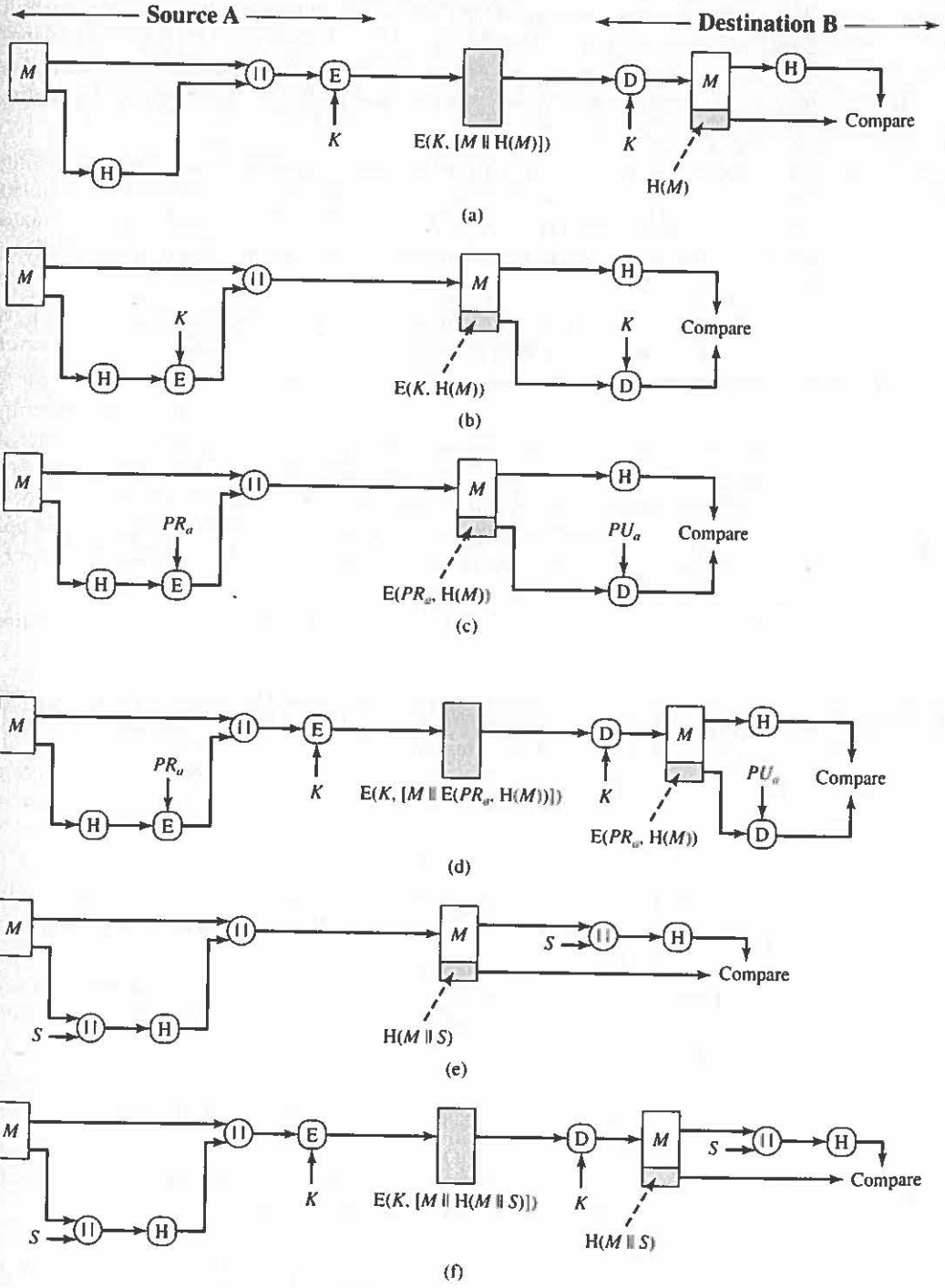


Figure 11.5 Basic Uses of Hash Function

JPMC Ex. 1041
Page 87

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.

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.

JPMC Ex. 1041
Page 88

Table 11.3 Basic Uses of Hash Function H (see Figure 11.5)

A → B: E(K, [M‖H(M)])	A → B: E(K, [M‖E(PR_a, H(M))])
• Provides confidentiality —Only A and B share K • Provides authentication —H(M) is cryptographically protected	• Provides authentication and digital signature • Provides confidentiality —Only A and B share K
(a) Encrypt message plus hash code	(d) Encrypt result of (c)—shared secret key
A → B: M‖E(K, H(M))	A → B: M‖H(M‖S)
• Provides authentication —H(M) is cryptographically protected	• Provides authentication —Only A and B share S
(b) Encrypt hash code—shared secret key	(e) Compute hash code of message plus secret value
A → B: M‖E(PR_a, H(M))	A → B: E(K, [M‖H(M‖S])
• Provides authentication and digital signature —H(M) is cryptographically protected —Only A could create E(PR_a, H(M))	• Provides authentication —Only A and B share S • Provides confidentiality —Only A and B share K
(c) Encrypt hash code—sender's private *key*	(f) Encrypt result of (e)

11.3 MESSAGE AUTHENTICATION CODES

A MAC, also known as a cryptographic checksum, is generated by a function C of the form

$$\text{MAC} = \text{C}(K, M)$$

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 $P_i = \text{D}(K_i, C)$ for all possible key values K_i until a P_i was produced that matched the form of acceptable plaintext.

JPMC Ex. 1041
Page 89

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 $\approx 2^{(k-2\times n)}$

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 $M = (X_1 \| X_2 \| \ldots \| X_m)$ be a message that is treated as a concatenation of 64-bit blocks X_i. Then define

$$\Delta(M) = X_1 \oplus X_2 \oplus \ldots \oplus X_m$$
$$C(K, M) = E(K, \Delta(M))$$

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:

$$Y_m = Y_1 \oplus Y_2 \oplus \ldots \oplus Y_{m-1} \oplus \Delta(M)$$

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: $D_1, D_2, \ldots, D_N$. If necessary, the final block is padded on the right with zeroes

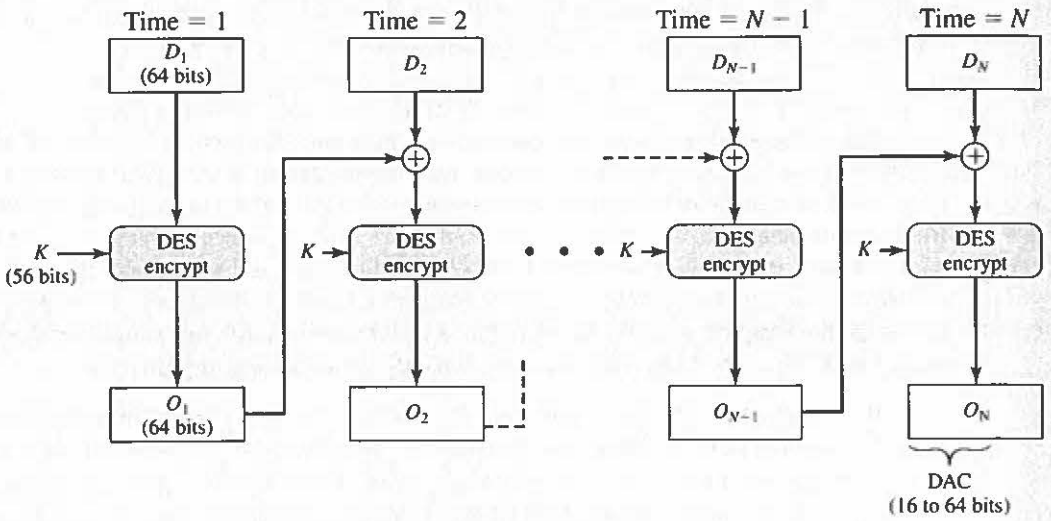


Figure 11.6 Data Authentication Algorithm (FIPS PUB 113)

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):

$$O_1 = \mathrm{E}(K, D_1)$$
$$O_2 = \mathrm{E}(K, [D_2 \oplus O_1])$$
$$O_3 = \mathrm{E}(K, [D_3 \oplus O_2])$$
$$\vdots$$
$$O_N = \mathrm{E}(K, [D_N \oplus O_{N-1}])$$

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

$$h = \mathrm{H}(M)$$

where M is a variable-length message and $\mathrm{H}(M)$ 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]

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.

---

[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.

JPMC Ex. 1041
Page 93

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:

$$C_i = b_{i1} \oplus b_{i2} \oplus \cdots \oplus b_{im}$$

where

C_i = ith bit of the hash code, $1 \leq i \leq$ n

m = number of n-bit blocks in the input

b_{ij} = ith bit in jth 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.

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

JPMC Ex. 1041
Page 94

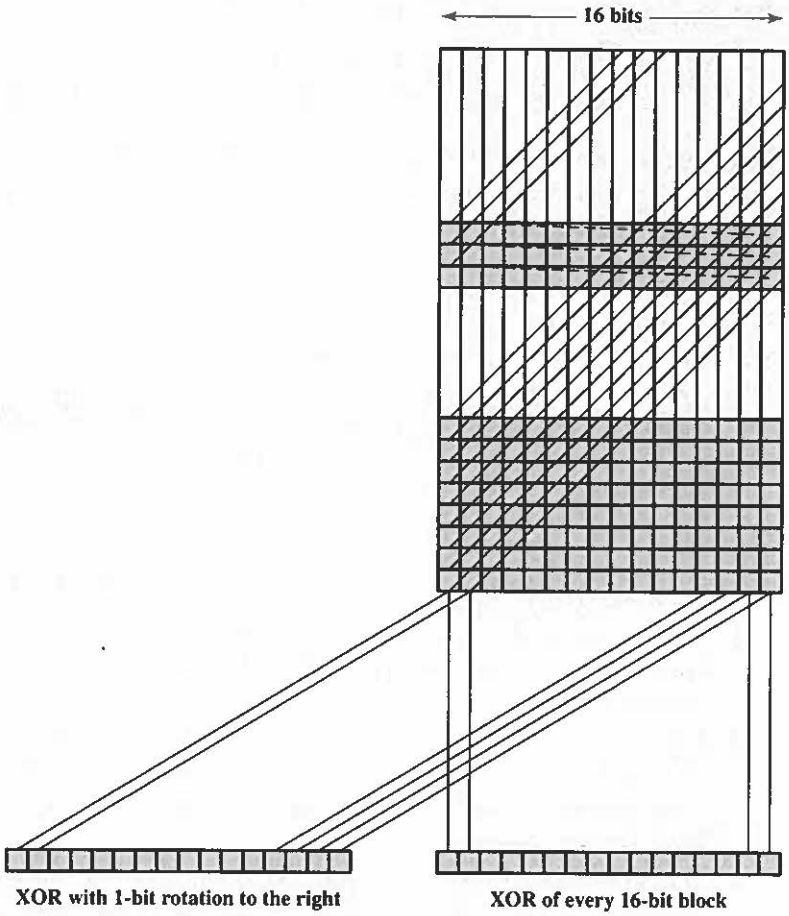


Figure 11.7 Two Simple Hash Functions

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 $X_1, X_2, \ldots, X_N$, define the hash code C as the block-by-block XOR of all blocks and append the hash code as the final block:

$$C = X_{N+1} = X_1 \oplus X_2 \oplus \ldots \oplus X_N$$

Next, encrypt the entire message plus hash code, using CBC mode to produce the encrypted message $Y_1, Y_2, \ldots, Y_{N+1}$. [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

$$X_1 = IV \oplus \mathrm{D}(K, Y_1)$$
$$X_i = Y_{i-1} \oplus \mathrm{D}(K, Y_i)$$
$$X_{N+1} = Y_N \oplus \mathrm{D}(K, Y_{N+1})$$

But X_{N+1} is the hash code:

$$X_{N+1} = X_1 \oplus X_2 \oplus \cdots \oplus X_N$$
$$= [IV \oplus \mathrm{D}(K, Y_1)] \oplus [Y_1 \oplus \mathrm{D}(K, Y_2)] \oplus \cdots \oplus [Y_{N-1} \oplus \mathrm{D}(K, Y_N)]$$

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 c), then an opponent would need to find an M' such that $\mathrm{H}(M') = \mathrm{H}(M)$ 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.

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.

JPMC Ex. 1041
Page 96

Dear Anthony,

{This letter is / I am writing} to introduce {you to / to you} {Mr. / --} Alfred {P. / --}

Barton, the {new / newly appointed} {chief / senior} jewellery buyer for {our / the}

Northern {European / Europe} {area / division}. He {will take / has taken} over {the / --}

responsibility for {all / the whole of} our interests in {watches and jewellery / jewellery and watches}

in the {area / region}. Please {afford / give} him {every / all the} help he {may need / needs}

to {seek out / find} the most {modern / up to date} lines for the {top / high} end of the

market. He is {empowered / authorized} to receive on our behalf {samples / specimens} of the

{latest / newest} {watch and jewellery / jewellery and watch} products, {up / subject} to a {limit / maximum}

of ten thousand dollars. He will {carry / hold} a signed copy of this {letter / document}

as proof of identity. An order with his signature, which is {appended / attached}

{authorizes / allows} you to charge the cost to this company at the {above / head office}

address. We {fully / --} expect that our {level / volume} of orders will increase in

the {following / next} year and {trust / hope} that the new appointment will {be / prove}

{advantageous / an advantage} to both our companies.

Figure 11.8 A Letter in 2^{37} Variations [DAVI89]

Block Chaining Techniques

A number of proposals have been made for hash functions based on using a cipher block chaining technique, but without the secret key. One of the first such proposals was that of Rabin [RABI78]. Divide a message M into fixed-size blocks $M_1, M_2, \ldots, M_N$ and use a symmetric encryption system such as DES to compute the hash code G as follows:

$$H_0 = \text{initial value}$$
$$H_i = \text{E}(M_i, H_{i-1})$$
$$G = H_N$$

JPMC Ex. 1041
Page 97

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 $Q_1, Q_2, \ldots, Q_{N-2}$.
3. Compute $H_i = \mathrm{E}(Q_i, H_{i-1})$ for $1 \leq i \leq (N-2)$.
4. Generate $2^{m/2}$ random blocks; for each block X, compute $\mathrm{E}(X, H_{N-2})$. 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 $\mathrm{E}(X, H_{N-2}) = \mathrm{D}(Y, G)$.
6. Form the message $Q_1, Q_2, \ldots, Q_{N-2}, X, Y$. 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:

$$H_i = \mathrm{E}(M_i, \mathrm{H}_{i-1}) \oplus H_{i-1}$$

Another variation, proposed in [MEYE88],

$$H_i = \mathrm{E}(H_{i-1}, M_i) \oplus M_i$$

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.

JPMC Ex. 1041
Page 98

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.

JPMC Ex. 1041
Page 99

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.

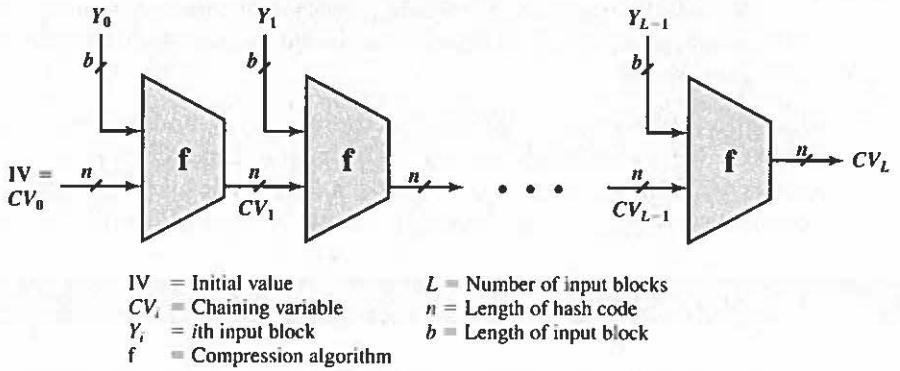


Figure 11.9 General Structure of Secure Hash Code

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:

$$\begin{aligned} CV_0 &= IV = \text{initial } n\text{-bit value} \\ CV_i &= \mathrm{f}(CV_{i-1}, Y_{i-1}) \qquad 1 \le i \le L \\ \mathrm{H}(M) &= CV_L \end{aligned}$$

where the input to the hash function is a message M consisting of the blocks $Y_0, Y_1, \ldots, Y_{L-1}$.

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.

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 $b \ge n$. What is required is that it is computationally infeasible to find collisions.

[3]The converse is not necessarily true.

JPMC Ex. 1041
Page 101

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].

11.6 RECOMMENDED READING

[JUEN85] and [JUEN87] 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].

JUEN85 Jueneman, R.; Matyas, S.; and Meyer, C. "Message Authentication." *IEEE Communications Magazine*, September 1988.

JUEN87 Jueneman, R. "Electronic Document Authentication." *IEEE Network Magazine*, April 1987.

MENE97 Menezes, A.; Oorshcot, 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	hash function	message digest
birthday attack	hash value	one-way hash function
birthday paradox	message authentication	strong collision resistance
compression function	message authentication code (MAC)	weak collision resistance
cryptographic checksum		
hash code		

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?

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.

11.3 The high-speed transport protocol XTP (Xpress Transfer Protocol) uses a 32-bit checksum function defined as the concatenation of two 16-bit functions: XOR and RXOR, defined in Section 11.4 as "two simple hash functions" and illustrated in Figure 11.7.

a. Will this checksum detect all errors caused by an odd number of error bits? Explain.
b. Will this checksum detect all errors caused by an even number of error bits? If not, characterize the error patterns that will cause the checksum to fail.
c. Comment on the effectiveness of this function for use as a hash function for authentication.

11.4 a. Consider the Davies and Price hash code scheme described in Section 11.4 and assume that DES is used as the encryption algorithm:

$$H_i = H_{i-1} \oplus \mathrm{E}(M_i, H_{i-1})$$

and recall the complementarity property of DES (Problem 3.14): If $Y = \mathrm{E}(K, X)$, then $Y' = \mathrm{E}(K', X')$. Use this property to show how a message consisting of blocks $M_1, M_2, \ldots, M_N$ can be altered without altering its hash code.

b. Show that a similar attack will succeed against the scheme proposed in [MEYE88]:

$$H_i = M_i \oplus \mathrm{E}(H_{i-1}, M_i)$$

11.5 a. Consider the following hash function. Messages are in the form of a sequence of decimal numbers, $M = (a_1, a_2, \ldots, a_t)$. The hash value h is calculated as $\left(\sum_{i=1}^{t} a_i\right) \bmod n$, for some predefined value n. Does this hash function satisfy any of the requirements for a hash function listed in Section 11.4? Explain your answer.

JPMC Ex. 1041
Page 103

b. Repeat part (a) for the hash function $h = \left(\sum_{i=1}^{t} (a_i)^2\right) \bmod n$.

c. Calculate the hash function of part (b) for $M = (189, 632, 900, 722, 349)$ and $n = 989$.

11.6 It is possible to use a hash function to construct a block cipher with a structure similar to DES. Because a hash function is one way and a block cipher must be reversible (to decrypt), how is it possible?

11.7 Now consider the opposite problem: using an encryption algorithm to construct a one-way hash function. Consider using RSA with a known key. Then process a message consisting of a sequence of blocks as follows: Encrypt the first block, XOR the result with the second block and encrypt again, etc. Show that this scheme is not secure by solving the following problem. Given a two-block message B1, B2, and its hash

$$\text{RSAH}(\text{B1}, \text{B2}) = \text{RSA}(\text{RSA}(\text{B1}) \oplus \text{B2})$$

Given an arbitrary block C1, choose C2 so that $\text{RSAH}(\text{C1}, \text{C2}) = \text{RSAH}(\text{B1}, \text{B2})$. Thus, the hash function does not satisfy weak collision resistance.

11.8 Suppose $\text{H}(m)$ is a collision resistant hash function that maps a message of arbitrary bit length into an n-bit hash value. Is it true that, for all messages x, x' with $x \neq x'$, we have $\text{H}(x) \neq \text{H}(x')$? Explain your answer.

APPENDIX 11A MATHEMATICAL BASIS OF THE BIRTHDAY ATTACK

In this appendix, we derive the mathematical justification for the birthday attack. We begin with a related problem and then look at the problem from which the name "birthday attack" is derived.

Related Problem

A general problem relating to hash functions is the following. Given a hash function H, with n possible outputs and a specific value $\text{H}(x)$, if H is applied to k random inputs, what must be the value of k so that the probability that at least one input y satisfies $\text{H}(y) = \text{H}(x)$ is 0.5?

For a single value of y, the probability that $\text{H}(y) = \text{H}(x)$ is just $1/n$. Conversely, the probability that $\text{H}(y) \neq \text{H}(x)$ is $[1 - (1/n)]$. If we generate k random values of y, then the probability that none of them match is just the product of the probabilities that each individual value does not match, or $[1 - (1/n)]^k$. Thus, the probability that there is at least one match is $1 - [1 - (1/n)]^k$.

The binomial theorem can be stated as follows:

$$(1 - a)^k = 1 - ka + \frac{k(k-1)}{2!}a^2 - \frac{k(k-1)(k-2)}{3!}a^3 \ldots$$

For very small values of a, this can be approximated as $(1 - ka)$. Thus, the probability of at least one match is approximated as $1 - [1 - (1/n)]^k \approx 1 - [1 - (k/n)] = k/n$. For a probability of 0.5, we have $k = n/2$.

In particular, for an m-bit hash code, the number of possible codes is 2^m and the value of k that produces a probability of one-half is

$$k = 2^{(m-1)} \tag{11.1}$$

JPMC Ex. 1041
Page 104

The Birthday Paradox

The birthday paradox is often presented in elementary probability courses to demonstrate that probability results are sometimes counterintuitive. The problem can be stated as follows: What is the minimum value of k such that the probability is greater than 0.5 that at least two people in a group of k people have the same birthday? Ignore February 29 and assume that each birthday is equally likely. To answer, let us define

$$\mathrm{P}(n, k) = \Pr[\text{at least one duplicate in } k \text{ items, with each item able to take on one of } n \text{ equally likely values between 1 and } n]$$

Thus, we are looking for the smallest value of k such that $\mathrm{P}(365, k) \geq 0.5$. It is easier first to derive the probability that there are no duplicates, which we designate as $\mathrm{Q}(365, k)$. If $k > 365$, then it is impossible for all values to be different. So we assume $k \leq 365$. Now consider the number of different ways, N, that we can have k values with no duplicates. We may choose any of the 365 values for the first item, any of the remaining 364 numbers for the second item, and so on. Hence, the number of different ways is

$$N = 365 \times 364 \times \ldots (365 - k + 1) = \frac{365!}{(365 - k)!} \tag{11.2}$$

If we remove the restriction that there are no duplicates, then each item can be any of 365 values, and the total number of possibilities is 365^k. So the probability of no duplicates is simply the fraction of sets of values that have no duplicates out of all possible sets of values:

$$\mathrm{Q}(365, k) = \frac{365!/(365 - k)!}{(365)^k} = \frac{365!}{(365 - k)!(365)^k}$$

and

$$\mathrm{P}(365, k) = 1 - \mathrm{Q}(365, k) = 1 - \frac{365!}{(365 - k)!(365)^k} \tag{11.3}$$

This function is plotted in Figure 11.10. The probabilities may seem surprisingly large to anyone who has not considered the problem before. Many people would guess that to have a probability greater than 0.5 that there is at least one duplicate, the number of people in the group would have to be about 100. In fact, the number is 23, with $\mathrm{P}(365, 23) = 0.5073$. For $k = 100$, the probability of at least one duplicate is 0.9999997.

Perhaps the reason that the result seems so surprising is that if you consider a particular person in a group, the probability that some other person in the group has the same birthday is small. But the probability that we are concerned with is the probability that *any* pair of people in the group has the same birthday. In a group of 23, there are $(23(23 - 1))/2 = 253$ different pairs of people. Hence the high probabilities.

Useful Inequality

Before developing a generalization of the birthday problem, we derive an inequality that will be needed:

$$(1 - x) \leq e^{-x} \qquad \text{for all } x \geq 0 \tag{11.4}$$

JPMC Ex. 1041
Page 105

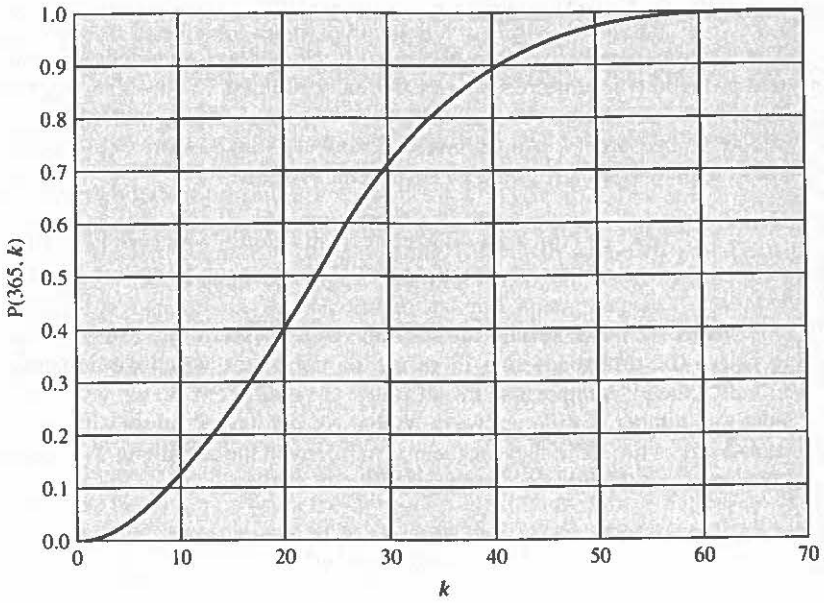


Figure 11.10 The Birthday Paradox

Figure 11.11 illustrates the inequality. To see that the inequality holds, note that the lower line is the tangent to e^{-x} at $x = 0$. The slope of that line is just the derivative of e^{-x} at $x = 0$:

$$f(x) = e^{-x}$$

$$f'(x) = \frac{d}{dx}e^{-x} = -e^{-x}$$

$$f'(0) = -1$$

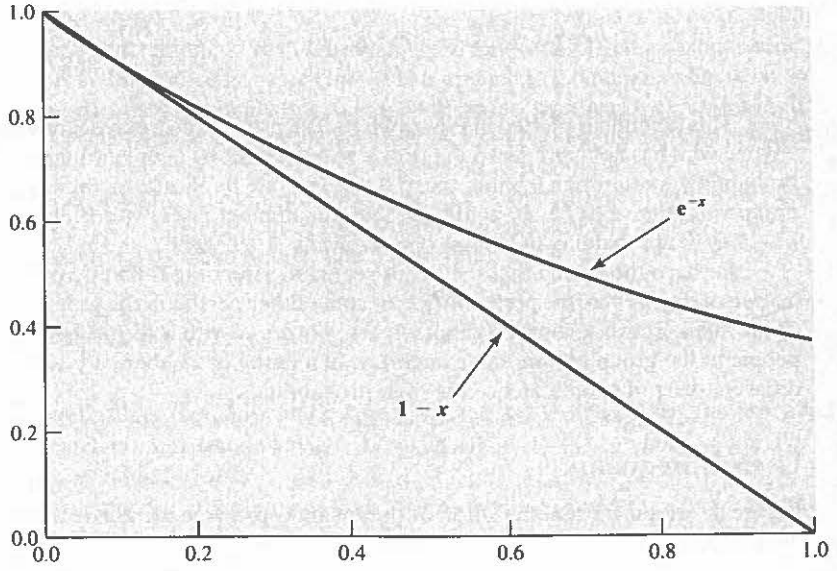


Figure 11.11 A Useful Inequality

The tangent is a straight line of the form $ax + b$, with $a = -1$, and the tangent at $x = 0$ must equal $e^{-0} = 1$. Thus, the tangent is the function $(1 - x)$, confirming the inequality of Equation (11.4). Further, note that for small x, we have $(1 - x) \approx e^{-x}$.

The General Case of Duplications

The birthday problem can be generalized to the following problem: Given a random variable that is an integer with uniform distribution between 1 and n and a selection of k instances ($k \leq n$) of the random variable, what is the probability, $P(n, k)$, that there is at least one duplicate? The birthday problem is just the special case with $n = 365$. By the same reasoning as before, we have the following generalization of Equation (11.3):

$$P(n, k) = 1 - \frac{n!}{(n - k)!n^k} \tag{11.5}$$

We can rewrite as

$$\begin{aligned} P(n, k) &= 1 - \frac{n \times (n - 1) \times \ldots \times (n - k + 1)}{n^k} \\ &= 1 - \left[\frac{n - 1}{n} \times \frac{n - 2}{n} \times \ldots \times \frac{n - k + 1}{n}\right] \\ &= 1 - \left[\left(1 - \frac{1}{n}\right) \times \left(1 - \frac{2}{n}\right) \times \ldots \times \left(1 - \frac{k - 1}{n}\right)\right] \end{aligned}$$

Using the inequality of Equation (11.4):

$$\begin{aligned} P(n, k) &> 1 - \left[(e^{-1/n}) \times (e^{-2/n}) \times \ldots \times (e^{-(k-1)/n})\right] \\ &> 1 - e^{-[(1/n)+(2/n)+\ldots+((k-1)/n)]} \\ &> 1 - e^{-(k\times(k-1))/2n} \end{aligned}$$

Now let us pose the question: What value of k is required such that $P(n, k) > 0.5$? To satisfy the requirement, we have

$$\begin{aligned} 1/2 &= 1 - e^{-(k\times(k-1))/2n} \\ 2 &= e^{(k\times(k-1))/2n} \\ \ln 2 &= \frac{k \times (k - 1)}{2n} \end{aligned}$$

For large k, we can replace $k \times (k - 1)$ by k^2, and we get

$$k = \sqrt{2(\ln 2)n} = 1.18\sqrt{n} \approx \sqrt{n} \tag{11.6}$$

As a reality check, for $n = 365$, we get $k = 1.18 \times \sqrt{365} = 22.54$, which is very close to the correct answer of 23.

We can now state the basis of the birthday attack in the following terms. Suppose we have a function H, with 2^m possible outputs (i.e., an m-bit output). If H is applied to k random

inputs, what must be the value of k so that there is the probability of at least one duplicate [i.e., $H(x) = H(y)$ for some inputs x, y)]? Using the approximation in Equation (11.6):

$$k = \sqrt{2^m} = 2^{m/2} \tag{11.7}$$

Overlap between Two Sets

There is a problem related to the general case of duplications that is also of relevance for our discussions. The problem is this: Given an integer random variable with uniform distribution between 1 and n and two sets of k instances ($k \leq n$) of the random variable, what is the probability, $R(n, k)$, that the two sets are not disjoint; that is, what is the probability that there is at least one value found in both sets?

Let us call the two sets X and Y, with elements $\{x_1, x_2, \ldots, x_k\}$ and $\{y_1, y_2, \ldots, y_k\}$, respectively. Given the value of x_1, the probability that $y_1 = x_1$ is just $1/n$, and therefore probability that y_1 does not match x_1 is $[1 - (1/n)]$. If we generate the k random values in Y, the probability that none of these values is equal to x_1 is $[1 - (1/n)]^k$. Thus, the probability that there is at least one match to x_1 is $1 - [1 - (1/n)]^k$.

To proceed, let us make the assumption that all the elements of X are distinct. If n is large and if k is also large (e.g., on the order of $\sqrt{n}$), then this is a good approximation. In fact, there may be a few duplications, but most of the values will be distinct. With that assumption, we can make the following derivation:

$$\Pr[\text{no match in } Y \text{ to } x_1] = \left(1 - \frac{1}{n}\right)^k$$

$$\Pr[\text{no match in } Y \text{ to } X] = \left(\left(1 - \frac{1}{n}\right)^k\right)^k = \left(1 - \frac{1}{n}\right)^{k^2}$$

$$R(n, k) = \Pr[\text{at least one match in } Y \text{ to } X] = 1 - \left(1 - \frac{1}{n}\right)^{k^2}$$

Using the inequality of Equation (11.4):

$$R(n, k) > 1 - (e^{-1/n})^{k^2}$$

$$R(n, k) > 1 - (e^{-k^2/n})$$

Let us pose the question: What value of k is required such that $R(n, k) > 0.5$? To satisfy the requirement, we have

$$1/2 = 1 - (e^{-k^2/n})$$

$$2 = e^{k^2/n}$$

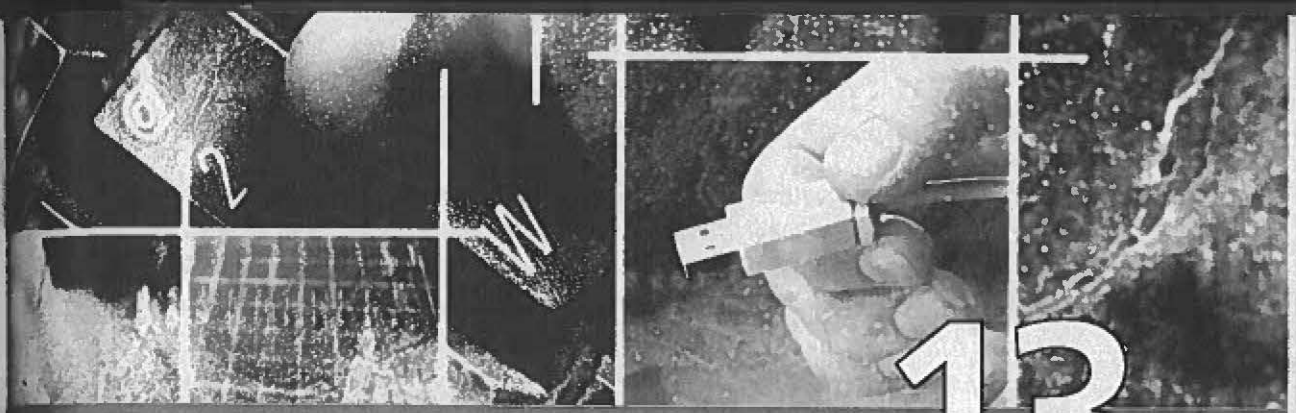
$$\ln(2) = \frac{k^2}{n}$$

$$k = \sqrt{(\ln(2))n} = 0.83\sqrt{n} \approx \sqrt{n} \tag{11.8}$$

We can state this in terms related to birthday attacks as follows. Suppose we have a function H, with 2^m possible outputs (i.e., an m-bit output). Apply H to k random inputs to produce the set X and again to k additional random inputs to produce the set Y. What must be the value of k so that there is the probability of at least 0.5 that there is a match between the two sets (i.e., $H(x) = H(y)$ for some inputs $x \in X$, $y \in Y$)? Using the approximation in Equation (11.8):

$$k = \sqrt{2^m} = 2^{m/2}$$

JPMC Ex. 1041
Page 108



CHAPTER 13

DIGITAL SIGNATURES AND AUTHENTICATION PROTOCOLS

13.1 **Digital Signatures**

Requirements
Direct Digital Signature
Arbitrated Digital Signature

13.2 **Authentication Protocols**

Mutual Authentication
One-Way Authentication

13.3 **Digital Signature Standard**

The DSS Approach
The Digital Signature Algorithm

13.4 **Recommended Reading and Web Sites**

13.5 **Key Terms, Review Questions, and Problems**

Key Terms
Review Questions
Problems

JPMC Ex. 1041
Page 109

To guard against the baneful influence exerted by strangers is therefore an elementary dictate of savage prudence. Hence before strangers are allowed to enter a district, or at least before they are permitted to mingle freely with the inhabitants, certain ceremonies are often performed by the natives of the country for the purpose of disarming the strangers of their magical powers, or of disinfecting, so to speak, the tainted atmosphere by which they are supposed to be surrounded.

—*The Golden Bough*, Sir James George Frazer

KEY POINTS

- A digital signature is an authentication mechanism that enables the creator of a message to attach a code that acts as a signature. The signature is formed by taking the hash of the message and encrypting the message with the creator's private key. The signature guarantees the source and integrity of the message.
- Mutual authentication protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys.
- In one-way authentication, the recipient wants some assurance that a message is from the alleged sender.
- The digital signature standard (DSS) is an NIST standard that uses the secure hash algorithm (SHA).

The most important development from the work on public-key cryptography is the digital signature. The digital signature provides a set of security capabilities that would be difficult to implement in any other way. We begin this chapter with an overview of digital signatures. Then we look at authentication protocols, many of which depend on the use of the digital signature. Finally, we introduce the Digital Signature Standard (DSS).

13.1 DIGITAL SIGNATURES

Requirements

Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. Several forms of dispute between the two are possible.

For example, suppose that John sends an authenticated message to Mary, using one of the schemes of Figure 11.4. Consider the following disputes that could arise:

1. Mary may forge a different message and claim that it came from John. Mary would simply have to create a message and append an authentication code using the key that John and Mary share.
2. John can deny sending the message. Because it is possible for Mary to forge a message, there is no way to prove that John did in fact send the message.

Both scenarios are of legitimate concern. Here is an example of the first scenario: An electronic funds transfer takes place, and the receiver increases the amount of funds transferred and claims that the larger amount had arrived from the sender. An example of the second scenario is that an electronic mail message contains instructions to a stockbroker for a transaction that subsequently turns out badly. The sender pretends that the message was never sent.

In situations where there is not complete trust between sender and receiver, something more than authentication is needed. The most attractive solution to this problem is the digital signature. The digital signature is analogous to the handwritten signature. It must have the following properties:

- It must verify the author and the date and time of the signature.
- It must to authenticate the contents at the time of the signature.
- It must be verifiable by third parties, to resolve disputes.

Thus, the digital signature function includes the authentication function.

On the basis of these properties, we can formulate the following requirements for a digital signature:

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information unique to the sender, to prevent both forgery and denial.
- It must be relatively easy to produce the digital signature.
- It must be relatively easy to recognize and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to retain a copy of the digital signature in storage.

A secure hash function, embedded in a scheme such as that of Figure 11.5c or d, satisfies these requirements.

A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated.

Direct Digital Signature

The direct digital signature involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source. A digital signature may be formed by encrypting the entire message with the sender's private

key (Figure 11.1c) or by encrypting a hash code of the message with the sender's private key (Figure 11.5c).

Confidentiality can be provided by further encrypting the entire message plus signature with either the receiver's public key (public-key encryption) or a shared secret key (symmetric encryption); for example, see Figures 11.1d and 11.5d. Note that it is important to perform the signature function first and then an outer confidentiality function. In case of dispute, some third party must view the message and its signature. If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message. However, if the signature is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.

All direct schemes described so far share a common weakness. The validity of the scheme depends on the security of the sender's private key. If a sender later wishes to deny sending a particular message, the sender can claim that the private key was lost or stolen and that someone else forged his or her signature. Administrative controls relating to the security of private keys can be employed to thwart or at least weaken this ploy, but the threat is still there, at least to some degree. One example is to require every signed message to include a timestamp (date and time) and to require prompt reporting of compromised keys to a central authority.

Another threat is that some private key might actually be stolen from X at time T. The opponent can then send a message signed with X's signature and stamped with a time before or equal to T.

Arbitrated Digital Signature

The problems associated with direct digital signatures can be addressed by using an arbiter.

As with direct signature schemes, there is a variety of arbitrated signature schemes. In general terms, they all operate as follows. Every signed message from a sender X to a receiver Y goes first to an arbiter A, who subjects the message and its signature to a number of tests to check its origin and content. The message is then dated and sent to Y with an indication that it has been verified to the satisfaction of the arbiter. The presence of A solves the problem faced by direct signature schemes: that X might disown the message.

The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly. The use of a trusted system, described in Chapter 20, might satisfy this requirement.

Table 13.1, based on scenarios described in [AKL83] and [MITC92], gives several examples of arbitrated digital signatures.[1] In the first, symmetric encryption is used. It is assumed that the sender X and the arbiter A share a secret key K_{xa} and that A and Y share secret key K_{ay}. X constructs a message M and computes its hash value H(M). Then X transmits the message plus a signature to A. The signature consists of an identifier ID_X of X plus the hash value, all encrypted using K_{xa}. A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y,

[1]The following format is used. A communication step in which P sends a message M to Q is represented as P → Q: M.

JPMC Ex. 1041
Page 112

Table 13.1 Arbitrated Digital Signature Techniques

(1) $X \rightarrow A: M \| E(K_{xa}, [ID_X \| H(M)])$
(2) $A \rightarrow Y: E(K_{ay}, [ID_X \| M \| E(K_{xa}, [ID_X \| H(M)]) \| T])$

(a) Conventional Encryption, Arbiter Sees Message

(1) $X \rightarrow A: ID_X \| E(K_{xy}, M) \| E(K_{xa}, [ID_X \| H(E(K_{xy}, M))])$
(2) $A \rightarrow Y: E(K_{ay}, [ID_X \| E(K_{xy}, M)]) \| E(K_{xa}, [ID_X \| H(E(K_{xy}, M)) \| T])$

(b) Conventional Encryption, Arbiter Does Not See Message

(1) $X \rightarrow A: ID_X \| E(PR_x, [ID_X \| E(PU_y, E(PR_x, M))])$
(2) $A \rightarrow Y: E(PR_a, [ID_X \| E(PU_y, E(PR_x, M)) \| T])$

(c) Public-Key Encryption, Arbiter Does Not See Message

Notation:

X = sender
Y = recipient
A = Arbiter
M = message
T = timestamp

encrypted with K_{ay}. The message includes ID_X, the original message from X, the signature, and a timestamp. Y can decrypt this to recover the message and the signature. The timestamp informs Y that this message is timely and not a replay. Y can store M and the signature. In case of dispute, Y, who claims to have received M from X, sends the following message to A:

$$E(K_{ay}, [ID_X \| M \| E(K_{xa}, [ID_X \| H(M)])])$$

The arbiter uses K_{ay} to recover ID_X, M, and the signature, and then uses K_{xa} to decrypt the signature and verify the hash code. In this scheme, Y cannot directly check X's signature; the signature is there solely to settle disputes. Y considers the message from X authentic because it comes through A. In this scenario, both sides must have a high degree of trust in A:

- X must trust A not to reveal K_{xa} and not to generate false signatures of the form $E(K_{xa}, [ID_X \| H(M)])$.
- Y must trust A to send $E(K_{ay}, [ID_X \| M \| E(K_{xa}, [ID_X \| H(M)]) \| T])$ only if the hash value is correct and the signature was generated by X.
- Both sides must trust A to resolve disputes fairly.

If the arbiter does live up to this trust, then X is assured that no one can forge his signature and Y is assured that X cannot disavow his signature.

The preceding scenario also implies that A is able to read messages from X to Y and, indeed, that any eavesdropper is able to do so. Table 13.1b shows a scenario that provides the arbitration as before but also assures confidentiality. In this case it is assumed that X and Y share the secret key K_{xy}. Now, X transmits an identifier, a copy of the message encrypted with K_{xy}, and a signature to A. The signature consists of the identifier plus the hash value of the encrypted message, all encrypted using K_{xa}. As before, A decrypts the signature and checks the hash value to validate the

JPMC Ex. 1041
Page 113

message. In this case, A is working only with the encrypted version of the message and is prevented from reading it. A then transmits everything that it received from X, plus a timestamp, all encrypted with K_{ay}, to Y.

Although unable to read the message, the arbiter is still in a position to prevent fraud on the part of either X or Y. A remaining problem, one shared with the first scenario, is that the arbiter could form an alliance with the sender to deny a signed message, or with the receiver to forge the sender's signature.

All the problems just discussed can be resolved by going to a public-key scheme, one version of which is shown in Table 13.1c. In this case, X double encrypts a message M first with X's private key, PR_x, and then with Y's public key, PU_y. This is a signed, secret version of the message. This signed message, together with X's identifier, is encrypted again with PR_x and, together with ID_X, is sent to A. The inner, double-encrypted message is secure from the arbiter (and everyone else except Y). However, A can decrypt the outer encryption to assure that the message must have come from X (because only X has PR_x). A checks to make sure that X's private/public key pair is still valid and, if so, verifies the message. Then A transmits a message to Y, encrypted with PR_a. The message includes ID_X, the double-encrypted message, and a timestamp.

This scheme has a number of advantages over the preceding two schemes. First, no information is shared among the parties before communication, preventing alliances to defraud. Second, no incorrectly dated message can be sent, even if PR_x is compromised, assuming that PR_a is not compromised. Finally, the content of the message from X to Y is secret from A and anyone else. However, this final scheme involves encryption of the message twice with a public-key algorithm. We discuss more practical approaches subsequently.

13.2 AUTHENTICATION PROTOCOLS

The basic tools described in Chapter 11 are used in a variety of applications, including the digital signature discussed in Section 13.1. Other uses are numerous and growing. In this section, we focus on two general areas (mutual authentication and one-way authentication) and examine some of the implications of authentication techniques in both.

Mutual Authentication

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. This topic was examined in Section 7.3 (symmetric techniques) and Section 10.1 (public-key techniques). There, the focus was key distribution. We return to this topic here to consider the wider implications of authentication.

Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness. To prevent masquerade and to prevent compromise of session keys, essential identification and session key information must be communicated in encrypted form. This requires the prior existence of secret or public keys that can be used for this purpose. The second issue, timeliness, is important because of the threat of message replays. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful

replay can disrupt operations by presenting parties with messages that appear genuine but are not.

[GONG93] lists the following examples of replay attacks:

- **Simple replay:** The opponent simply copies a message and replays it later.
- **Repetition that can be logged:** An opponent can replay a timestamped message within the valid time window.
- **Repetition that cannot be detected:** This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only the replay message arrives.
- **Backward replay without modification:** This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order. The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. Instead, one of the following two general approaches is used:

- **Timestamps:** Party A accepts a message as fresh only if the message contains a timestamp that, in A's judgment, is close enough to A's knowledge of current time. This approach requires that clocks among the various participants be synchronized.
- **Challenge/response:** Party A, expecting a fresh message from B, first sends B a nonce (challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

It can be argued (e.g., [LAM92a]) that the timestamp approach should not be used for connection-oriented applications because of the inherent difficulties with this technique. First, some sort of protocol is needed to maintain synchronization among the various processor clocks. This protocol must be both fault tolerant, to cope with network errors, and secure, to cope with hostile attacks. Second, the opportunity for a successful attack will arise if there is a temporary loss of synchronization resulting from a fault in the clock mechanism of one of the parties. Finally, because of the variable and unpredictable nature of network delays, distributed clocks cannot be expected to maintain precise synchronization. Therefore, any timestamp-based procedure must allow for a window of time sufficiently large to accommodate network delays yet sufficiently small to minimize the opportunity for attack.

On the other hand, the challenge-response approach is unsuitable for a connectionless type of application because it requires the overhead of a handshake before any connectionless transmission, effectively negating the chief characteristic of a connectionless transaction. For such applications, reliance on some sort of secure time server and a consistent attempt by each party to keep its clocks in synchronization may be the best approach (e.g., [LAM92b]).

JPMC Ex. 1041
Page 115

Symmetric Encryption Approaches As was discussed in Section 7.3, a two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment. In general, this strategy involves the use of a trusted key distribution center (KDC). Each party in the network shares a secret key, known as a master key, with the KDC. The KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys, and for distributing those keys using the master keys to protect the distribution. This approach is quite common. As an example, we look at the Kerberos system in Chapter 14. The discussion in this subsection is relevant to an understanding of the Kerberos mechanisms.

Figure 7.9 illustrates a proposal initially put forth by Needham and Schroeder [NEED78] for secret key distribution using a KDC that, as was mentioned in Chapter 7, includes authentication features. The protocol can be summarized as follows:

1. A → KDC: $ID_A \| ID_B \| N_1$
2. KDC → A: $E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. A → B: $E(K_b, [K_s \| ID_A])$
4. B → A: $E(K_s, N_2)$
5. A → B: $E(K_s, f(N_2))$

Secret keys K_a and K_b are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key K_s to A and B. A securely acquires a new session key in step 2. The message in step 3 can be decrypted, and hence understood, only by B. Step 4 reflects B's knowledge of K_s, and step 5 assures B of A's knowledge of K_s and assures B that this is a fresh message because of the use of the nonce N_2. Recall from our discussion in Chapter 7 that the purpose of steps 4 and 5 is to prevent a certain type of replay attack. In particular, if an opponent is able to capture the message in step 3 and replay it, this might in some fashion disrupt operations at B.

Despite the handshake of steps 4 and 5, the protocol is still vulnerable to a form of replay attack. Suppose that an opponent, X, has been able to compromise an old session key. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3. Nevertheless, it is a potential security risk. X can impersonate A and trick B into using the old key by simply replaying step 3. Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay. If X can intercept the handshake message, step 4, then it can impersonate A's response, step 5. From this point on, X can send bogus messages to B that appear to B to come from A using an authenticated session key.

Denning [DENN81, DENN82] proposes to overcome this weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3. Her proposal assumes that the master keys, K_a and K_b, are secure, and it consists of the following steps:

1. A → KDC: $ID_A \| ID_B$
2. KDC → A: $E(K_a, [K_s \| ID_B \| T \| E(K_b, [K_s \| ID_A \| T])])$
3. A → B: $E(K_b, [K_s \| ID_A \| T])$

JPMC Ex. 1041
Page 116

4. $B \rightarrow A$: $E(K_s, N_1)$
5. $A \rightarrow B$: $E(K_s, f(N_1))$

T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange. A and B can verify timeliness by checking that

$$|\text{Clock} - T| < \Delta t_1 + \Delta t_2$$

where Δt_1 is the estimated normal discrepancy between the KDC's clock and the local clock (at A or B) and Δt_2 is the expected network delay time. Each node can set its clock against some standard reference source. Because the timestamp T is encrypted using the secure master keys, an opponent, even with knowledge of an old session key, cannot succeed because a replay of step 3 will be detected by B as untimely.

A final point: Steps 4 and 5 were not included in the original presentation [DENN81] but were added later [DENN82]. These steps confirm the receipt of the session key at B.

The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network. [GONG92] points out a risk involved. The risk is based on the fact that the distributed clocks can become unsynchronized as a result of sabotage on or faults in the clocks or the synchronization mechanism.[2] The problem occurs when a sender's clock is ahead of the intended recipient's clock. In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site. This replay could cause unexpected results. Gong refers to such attacks as **suppress-replay attacks**.

One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC's clock. The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonces. This latter alternative is not vulnerable to a suppress-replay attack because the nonces the recipient will choose in the future are unpredictable to the sender. The Needham/Schroeder protocol relies on nonces only but, as we have seen, has other vulnerabilities.

In [KEHN92], an attempt is made to respond to the concerns about suppress-replay attacks and at the same time fix the problems in the Needham/Schroeder protocol. Subsequently, an inconsistency in this latter protocol was noted and an improved strategy was presented in [NEUM93a].[3] The protocol is as follows:

1. $A \rightarrow B$: $ID_A \| N_a$
2. $B \rightarrow KDC$: $ID_B \| N_b \| E(K_b, [ID_A \| N_a \| T_b])$
3. $KDC \rightarrow A$: $E(K_a, [ID_B \| N_a \| K_s \| T_b]) \| E(K_b, [ID_A \| K_s \| T_b]) \| N_b$
4. $A \rightarrow B$: $E(K_b, [ID_A \| K_s \| T_b]) \| E(K_s, N_b)$

[2]Such things can and do happen. In recent years, flawed chips were used in a number of computers and other electronic systems to track the time and date. The chips had a tendency to skip forward one day [NEUM90].

[3]It really is hard to get these things right.

JPMC Ex. 1041
Page 117

Let us follow this exchange step by step.

1. A initiates the authentication exchange by generating a nonce, N_a, and sending that plus its identifier to B in plaintext. This nonce will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.
2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce, N_b. This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness. B's message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDC to issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.
3. The KDC passes on to A B's nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a "ticket" that can be used by A for subsequent authentications, as will be seen. The KDC also sends to A a block encrypted with the secret key shared by A and the KDC. This block verifies that B has received A's initial message (ID_B) and that this is a timely message and not a replay (N_a), and it provides A with a session key (K_s) and the time limit on its use (T_b).
4. A transmits the ticket to B, together with the B's nonce, the latter encrypted with the session key. The ticket provides B with the secret key that is used to decrypt $E(K_s, N_b)$ to recover the nonce. The fact that B's nonce is encrypted with the session key authenticates that the message came from A and is not a replay.

This protocol provides an effective, secure means for A and B to establish a session with a secure session key. Furthermore, the protocol leaves A in possession of a key that can be used for subsequent authentication to B, avoiding the need to contact the authentication server repeatedly. Suppose that A and B establish a session using the aforementioned protocol and then conclude that session. Subsequently, but within the time limit established by the protocol, A desires a new session with B. The following protocol ensues:

1. $A \rightarrow B$: $E(K_b, [ID_A \| K_s \| T_b]) \| N'_a$
2. $B \rightarrow A$: $N'_b \| E(K_s, N'_a)$
3. $A \rightarrow B$: $E(K_s, N'_b)$

When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces N'_a and N'_b assure each party that there is no replay attack.

In all the foregoing, the time specified in T_b is a time relative to B's clock. Thus, this timestamp does not require synchronized clocks because B checks only self-generated timestamps.

Public-Key Encryption Approaches In Chapter 10, we presented one approach to the use of public-key encryption for the purpose of session key distribution (Figure 10.6). This protocol assumes that each of the two parties is in possession of the current public key of the other. It may not be practical to require this assumption.

A protocol using timestamps is provided in [DENN81]:

1. A → AS: $ID_A \| ID_B$
2. AS → A: $E(PR_{as}, [ID_A \| PU_a \| T]) \| E(PR_{as}, [ID_B \| PU_b \| T])$
3. A → B: $E(PR_{as}, [ID_A \| PU_a \| T]) \| E(PR_{as}, [ID_B \| PU_b \| T]) \| E(PU_b, E(PR_a, [K_s \| T]))$

In this case, the central system is referred to as an authentication server (AS), because it is not actually responsible for secret key distribution. Rather, the AS provides public-key certificates. The session key is chosen and encrypted by A; hence, there is no risk of exposure by the AS. The timestamps protect against replays of compromised keys.

This protocol is compact but, as before, requires synchronization of clocks. Another approach, proposed by Woo and Lam [WOO92a], makes use of nonces. The protocol consists of the following steps:

1. A → KDC: $ID_A \| ID_B$
2. KDC → A: $E(PR_{auth}, [ID_B \| PU_b])$
3. A → B: $E(PU_b, [N_a \| ID_A])$
4. B → KDC: $ID_A \| ID_B \| E(PU_{auth}, N_a)$
5. KDC → B: $E(PR_{auth}, [ID_A \| PU_a]) \| E(PU_b, E(PR_{auth}, [N_a \| K_s \| ID_B]))$
6. B → A: $E(PU_a, E(PR_{auth}, [(N_a \| K_s \| ID_B) \| N_b]))$
7. A → B: $E(K_s, N_b)$

In step 1, A informs the KDC of its intention to establish a secure connection with B. The KDC returns to A a copy of B's public-key certificate (step 2). Using B's public key, A informs B of its desire to communicate and sends a nonce N_a (step 3). In step 4, B asks the KDC for A's public-key certificate and requests a session key; B includes A's nonce so that the KDC can stamp the session key with that nonce. The nonce is protected using the KDC's public key. In step 5, the KDC returns to B a copy of A's public-key certificate, plus the information $\{N_a, K_s, ID_B\}$. This information basically says that K_s is a secret key generated by the KDC on behalf of B and tied to N_a; the binding of K_s and N_a will assure A that K_s is fresh. This triple is encrypted, using the KDC's private key, to allow B to verify that the triple is in fact from the KDC. It is also encrypted using B's public key, so that no other entity may use the triple in an attempt to establish a fraudulent connection with A. In step 6, the triple $\{N_a, K_s, ID_B\}$, still encrypted with the KDC's private key, is relayed to A, together with a nonce N_b generated by B. All the foregoing are encrypted using A's public key. A retrieves the session key K_s and uses it to encrypt N_b and return it to B. This last message assures B of A's knowledge of the session key.

This seems to be a secure protocol that takes into account the various attacks. However, the authors themselves spotted a flaw and submitted a revised version of the algorithm in [WOO92b]:

1. A → KDC: $ID_A \| ID_B$
2. KDC → A: $E(PR_{auth}, [ID_B \| PU_b])$
3. A → B: $E(PU_b, [N_a \| ID_A])$

JPMC Ex. 1041
Page 119

4. B → KDC: $ID_A \| ID_B \| E(PU_{auth}, N_a)$
5. KDC → B: $E(PR_{auth}, [ID_A \| PU_a]) \| E(PU_b, E(PR_{auth}, [N_a \| K_s \| ID_A \| ID_B]))$
6. B → A: $E(PU_a, E(PR_{auth}, [(N_a \| K_s \| ID_A \| ID_B) \| N_b]))$
7. A → B: $E(K_s, N_b)$

The identifier of A, ID_A, is added to the set of items encrypted with the KDC's private key in steps 5 and 6. This binds the session key K_s to the identities of the two parties that will be engaged in the session. This inclusion of ID_A accounts for the fact that the nonce value N_a is considered unique only among all nonces generated by A, not among all nonces generated by all parties. Thus, it is the pair $\{ID_A, N_a\}$ that uniquely identifies the connection request of A.

In both this example and the protocols described earlier, protocols that appeared secure were revised after additional analysis. These examples highlight the difficulty of getting things right in the area of authentication.

One-Way Authentication

One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be online at the same time. Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.

The "envelope" or header of the e-mail message must be in the clear, so that the message can be handled by the store-and-forward e-mail protocol, such as the Simple Mail Transfer Protocol (SMTP) or X.400. However, it is often desirable that the mail-handling protocol not require access to the plaintext form of the message, because that would require trusting the mail-handling mechanism. Accordingly, the e-mail message should be encrypted such that the mail-handling system is not in possession of the decryption key.

A second requirement is that of authentication. Typically, the recipient wants some assurance that the message is from the alleged sender.

Symmetric Encryption Approach Using symmetric encryption, the decentralized key distribution scenario illustrated in Figure 7.11 is impractical. This scheme requires the sender to issue a request to the intended recipient, await a response that includes a session key, and only then send the message.

With some refinement, the KDC strategy illustrated in Figure 7.9 is a candidate for encrypted electronic mail. Because we wish to avoid requiring that the recipient (B) be on line at the same time as the sender (A), steps 4 and 5 must be eliminated. For a message with content M, the sequence is as follows:

1. A → KDC: $ID_A \| ID_B \| N_1$
2. KDC → A: $E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. A → B: $E(K_b, [K_s \| ID_A]) \| E(K_s, M)$

This approach guarantees that only the intended recipient of a message will be able to read it. It also provides a level of authentication that the sender is A. As specified, the protocol does not protect against replays. Some measure of defense could be

provided by including a timestamp with the message. However, because of the potential delays in the e-mail process, such timestamps may have limited usefulness.

Public-Key Encryption Approaches We have already presented public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for confidentiality (Figure 11.1b), authentication (Figure 11.1c), or both (Figure 11.1d). These approaches require that either the sender know the recipient's public key (confidentiality) or the recipient know the sender's public key (authentication) or both (confidentiality plus authentication). In addition, the public-key algorithm must be applied once or twice to what may be a long message.

If confidentiality is the primary concern, then the following may be more efficient:

$$A \rightarrow B: E(PU_b, K_s) \| E(K_s, M)$$

In this case, the message is encrypted with a one-time secret key. A also encrypts this one-time key with B's public key. Only B will be able to use the corresponding private key to recover the one-time key and then use that key to decrypt the message. This scheme is more efficient than simply encrypting the entire message with B's public key.

If authentication is the primary concern, then a digital signature may suffice, as was illustrated in Figure 11.5c:

$$A \rightarrow B: M \| E(PR_a, H(M))$$

This method guarantees that A cannot later deny having sent the message. However, this technique is open to another kind of fraud. Bob composes a message to his boss Alice that contains an idea that will save the company money. He appends his digital signature and sends it into the e-mail system. Eventually, the message will get delivered to Alice's mailbox. But suppose that Max has heard of Bob's idea and gains access to the mail queue before delivery. He finds Bob's message, strips off his signature, appends his, and requeues the message to be delivered to Alice. Max gets credit for Bob's idea.

To counter such a scheme, both the message and signature can be encrypted with the recipient's public key:

$$A \rightarrow B: E(PU_b, [M \| E(PR_a, H(M))])$$

The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate, described in Chapter 10. Now we have

$$A \rightarrow B: M \| E(PR_a, H(M)) \| E(PR_{as}, [T \| ID_A \| PU_a])$$

In addition to the message, A sends B the signature, encrypted with A's private key, and A's certificate, encrypted with the private key of the authentication server. The recipient of the message first uses the certificate to obtain the sender's public key and verify that it is authentic and then uses the public key to verify the message itself. If confidentiality is required, then the entire message can be encrypted with B's public key. Alternatively, the entire message can be encrypted with a one-time secret key; the secret key is also transmitted, encrypted with B's public key. This approach is explored in Chapter 15.

JPMC Ex. 1041
Page 121

13.3 DIGITAL SIGNATURE STANDARD

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Hash Algorithm (SHA) described in Chapter 12 and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme. There was a further minor revision in 1996. In 2000, an expanded version of the standard was issued as FIPS 186-2. This latest version also incorporates digital signature algorithms based on RSA and on elliptic curve cryptography. In this section, we discuss the original DSS algorithm.

The DSS Approach

The DSS uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

Figure 13.1 contrasts the DSS approach for generating digital signatures to that used with RSA. In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.

The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number k generated for this particular signature. The signature function also depends on the sender's private key

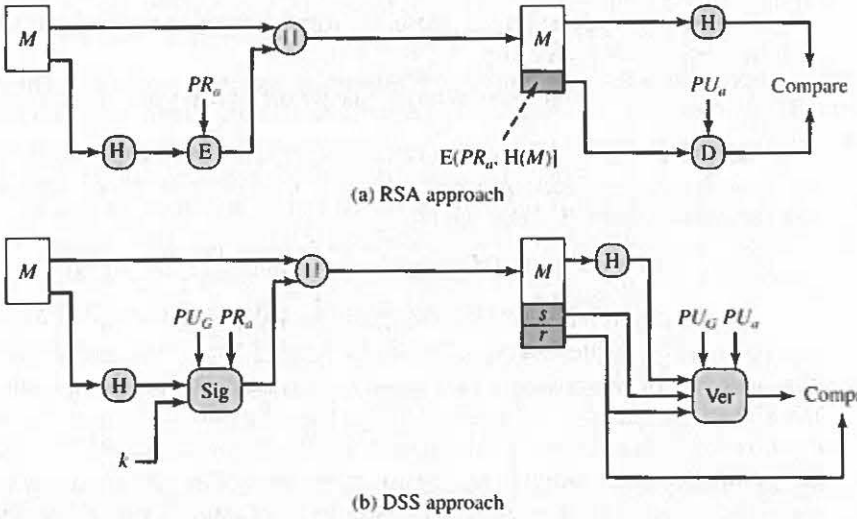


Figure 13.1 Two Approaches to Digital Signatures

(PR_a) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PU_G).[4] The result is a signature consisting of two components, labeled s and r.

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender's public key (PU_a), which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component r if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

We turn now to the details of the algorithm.

The Digital Signature Algorithm

The DSA is based on the difficulty of computing discrete logarithms (see Chapter 8) and is based on schemes originally presented by ElGamal [ELGA85] and Schnorr [SCHN91].

Figure 13.2 summarizes the algorithm. There are three parameters that are public and can be common to a group of users. A 160-bit prime number q is chosen.

Global Public-Key Components

p prime number where $2^{L-1} < p < 2^L$ for $512 \le L \le 1024$ and L a multiple of 64; i.e., bit length of between 512 and 1024 bits in increments of 64 bits

q prime divisor of $(p - 1)$, where $2^{159} < q < 2^{160}$; i.e., bit length of 160 bits

g $= h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < (p - 1)$ such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x random or pseudorandom integer with $0 < x < q$

User's Public Key

$y = g^x \bmod p$

User's Per-Message Secret Number

k = random or pseudorandom integer with $0 < k < q$

Signing

$r = (g^k \bmod p) \bmod q$

$s = [k^{-1}(\mathrm{H(M)} + xr)] \bmod q$

Signature = (r, s)

Verifying

$w = (s')^{-1} \bmod q$

$u_1 = [\mathrm{H(M')}w] \bmod q$

$u_2 = (r')w \bmod q$

$v = [(g^{u1}\, y^{u2}) \bmod p] \bmod q$

TEST: $v = r'$

M = message to be signed

$\mathrm{H}(M)$ = hash of M using SHA-1

M', r', s' = received versions of M, r, s

Figure 13.2 The Digital Signature Algorithm (DSA)

[4]It is also possible to allow these additional parameters to vary with each user so that they are a part of a user's public key. In practice, it is more likely that a global public key will be used that is separate from each user's public key.

Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides $(p - 1)$. Finally, g is chosen to be of the form $h^{(p-1)/q} \bmod p$ where h is an integer between 1 and $(p - 1)$ with the restriction that g must be greater than 1.[5]

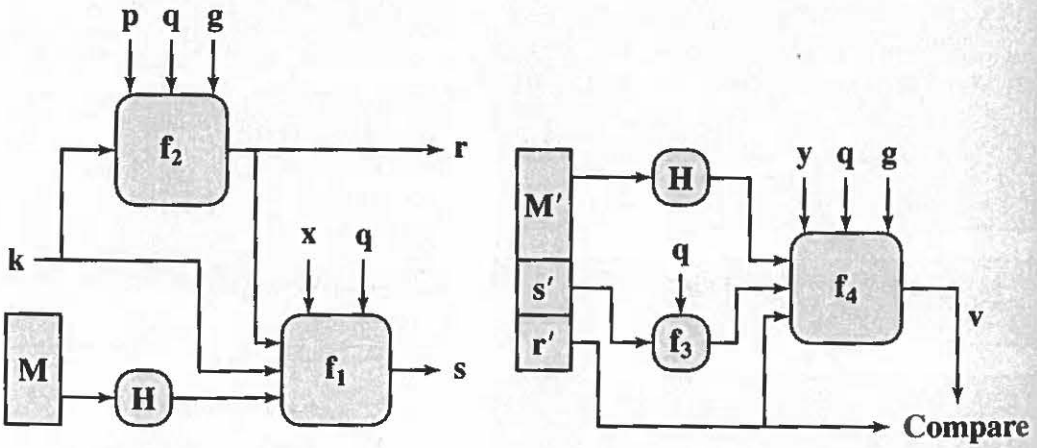
With these numbers in hand, each user selects a private key and generates a public key. The private key x must be a number from 1 to $(q - 1)$ and should be chosen randomly or pseudorandomly. The public key is calculated from the private key as $y = g^x \bmod p$. The calculation of y given x is relatively straightforward. However, given the public key y, it is believed to be computationally infeasible to determine x, which is the discrete logarithm of y to the base g, mod p (see Chapter 8).

To create a signature, a user calculates two quantities, r and s, that are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message, $H(M)$, and an additional integer k that should be generated randomly or pseudorandomly and be unique for each signing.

At the receiving end, verification is performed using the formulas shown in Figure 13.2. The receiver generates a quantity v that is a function of the public key components, the sender's public key, and the hash code of the incoming message. If this quantity matches the r component of the signature, then the signature is validated.

Figure 13.3 depicts the functions of signing and verifying.

The structure of the algorithm, as revealed in Figure 13.3, is quite interesting. Note that the test at the end is on the value r, which does not depend on the



$s = f_1(H(M), k, x, r, q) = (k^{-1} (H(M) + xr)) \bmod q$

$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$

(a) Signing

$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M'), w, r')$

$= ((g^{(H(M')w) \bmod q} y^{r'w \bmod q}) \bmod p) \bmod q$

(b) Verifying

Figure 13.3 DSS Signing and Verifying

[5]In number-theoretic terms, g is of order q mod p; see Chapter 8.

JPMC Ex. 1041
Page 124

message at all. Instead, r is a function of k and the three global public-key components. The multiplicative inverse of k (mod q) is passed to a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover r using the incoming message and signature, the public key of the user, and the global public key. It is certainly not obvious from Figure 13.2 or Figure 13.3 that such a scheme would work. A proof is provided at this book's Web site.

Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover k from r or to recover x from s.

Another point worth noting is that the only computationally demanding task in signature generation is the exponential calculation $g^k \bmod p$. Because this value does not depend on the message to be signed, it can be computed ahead of time. Indeed, a user could precalculate a number of values of r to be used to sign documents as needed. The only other somewhat demanding task is the determination of a multiplicative inverse, k^{-1}. Again, a number of these values can be precalculated.

13.4 RECOMMENDED READING AND WEB SITES

[AKL83] is the classic paper on digital signatures and is still highly relevant. A more recent, and excellent, survey is [MITC92].

AKL83 Akl, S. "Digital Signatures: A Tutorial Survey." *Computer*, February 1983.
MITC92 Mitchell, C.; Piper, F.; and Wild, P. "Digital Signatures." In [SIMM92a].



Recommended Web Sites:

- **Digital Signatures:** NIST page with information on NIST-approved digital signature options

13.5 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

arbiter	digital signature algorithm (DSA)	nonce
arbitrated digital signature	digital signature standard (DSS)	one-way authentication
direct digital signature	mutual authentication	replay attack
digital signature		suppress-replay attack
		timestamp

Review Questions

13.1 List two disputes that can arise in the context of message authentication.
13.2 What are the properties a digital signature should have?
13.3 What requirements should a digital signature scheme satisfy?
13.4 What is the difference between direct and arbitrated digital signature?
13.5 In what order should the signature function and the confidentiality function be applied to a message, and why?
13.6 What are some threats associated with a direct digital signature scheme?
13.7 Give examples of replay attacks.
13.8 List three general approaches to dealing with replay attacks.
13.9 What is a suppress-replay attack?

Problems

13.1 Modify the digital signature techniques of Table 13.1a and b to enable the receiver to verify the signature.
13.2 Modify the digital signature technique of Table 13.1c to avoid triple encryption of the entire message.
13.3 In discussing Table 13.1c, it was stated that alliances to defraud were impossible. In fact, there is one possibility. Describe it and explain why it would have so little credibility that we can safely ignore it.
13.4 In Section 13.2, we outlined the public-key scheme proposed in [WOO92a] for the distribution of secret keys. The revised version includes ID_A in steps 5 and 6. What attack, specifically, is countered by this revision?
13.5 The protocol referred to in Problem 13.1 can be reduced from seven steps to five, having the following sequence:

(1) A → B:
(2) B → KDC:
(3) KDC → B:
(4) B → A:
(5) A → B:

Show the message transmitted at each step. *Hint:* The final message in this protocol is the same as the final message in the original protocol.

13.6 With reference to the suppress-replay attack described in Section 13.2:
a. Give an example of an attack when a party's clock is ahead of that of the KDC.
b. Give an example of an attack when a party's clock is ahead of that of another party.

13.7 There are three typical ways to use nonces as challenges. Suppose N_a is a nonce generated by A, A and B share key K, and f() is a function such as increment. The three usages are

Usage 1	Usage 2	Usage 3
(1) A → B: N_a	(1) A → B: E(K, N_a)	(1) A → B: E(K, N_a)
(2) B → A: E(K, N_a)	(2) B → A: N_a	(2) B → A: E(K, f(N_a))

Describe situations for which each usage is appropriate.

13.8 Dr. Watson patiently waited until Sherlock Holmes finished. "Some interesting problem to solve, Holmes?" he asked when Holmes finally logged out.

"Oh, not exactly. I merely checked my e-mail and then made a couple of network experiments instead of my usual chemical ones. I have only one client now and

I have already solved his problem. If I remember correctly, you once mentioned cryptology among your other hobbies, so it may interest you."

"Well, I am only an amateur cryptologist, Holmes. But of course I am interested in the problem. What is it about?"

"My client is Mr. Hosgrave, director of a small but progressive bank. The bank is fully computerized and of course uses network communications extensively. The bank already uses RSA to protect its data and to digitally sign documents that are communicated. Now the bank wants to introduce some changes in its procedures; in particular, it needs to digitally sign some documents by *two* signatories so that

1. The first signatory prepares the document, forms its signature, and passes the document to the second signatory.
2. The second signatory as a first step must verify that the document was really signed by the first signatory. She then incorporates her signature into the document's signature so that the recipient, as well as any member of the public, may verify that the document was indeed signed by both signatories. In addition only the second signatory has to be able to verify the document's signature after the step (1); that is the recipient (or any member of the public) should be able to verify only the complete document with signatures of both signatories, but not the document in its intermediate form where only one signatory has signed it. Moreover, the bank would like to make use of its existing modules that support RSA-style digital signatures."

"Hm, I understand how RSA can be used to digitally sign documents by *one* signatory, Holmes. I guess you have solved the problem of Mr. Hosgrave by appropriate generalization of RSA digital signatures."

"Exactly, Watson," nodded Sherlock Holmes. "Originally, the RSA digital signature was formed by encrypting the document by the signatory's private decryption key 'd', and the signature could be verified by anyone through its decryption using publicly known encryption key 'e'. One can verify that the signature S was formed by the person who knows d, which is supposed to be the only signatory. Now the problem of Mr. Hosgrave can be solved in the same way by slight generalization of the process, that is . . . "

Finish the explanation.

13.9 DSA specifies that if the signature generation process results in a value of $s = 0$, a new value of k should be generated and the signature should be recalculated. Why?

13.10 What happens if a k value used in creating a DSA signature is compromised?

13.11 The DSS document includes a recommended algorithm for testing a number for primality, as follows:

(1) **[Choose *w*]** Let w be a random odd integer. Then $(w - 1)$ is even and can be expressed in the form $2^a m$ with m odd. That is, 2^a is the largest power of 2 that divides $(w - 1)$.
(2) **[Generate *b*]** Let b be a random integer in the range $1 < b < w$.
(3) **[Exponentiate]** Set $j = 0$ and $z = b^m \bmod w$.
(4) **[Done?]** If $j = 0$ and $z = 1$, or if $z = w - 1$, then w passes the test and may be prime; go to step 8.
(5) **[Terminate?]** If $j > 0$ and $z = 1$, then w is not prime; terminate algorithm for this w.
(6) **[Increase *j*]** Set $j = j + 1$. If $j < a$, set $z = z^2 \bmod w$ and go to step 4.
(7) **[Terminate]** w is not prime; terminate algorithm for this w.
(8) **[Test again?]** If enough random values of b have been tested, then accept w as prime and terminate algorithm; otherwise, go to step 2.

a. Explain how the algorithm works.
b. Show that it is equivalent to the Miller-Rabin test described in Chapter 8.

13.12 With DSS, because the value of k is generated for each signature, even if the same message is signed twice on different occasions, the signatures will differ. This is not true of RSA signatures. What is the practical implication of this difference?

13.13 Consider the problem of creating domain parameters for DSA. Suppose we have already found primes p and q such that $q|(p-1)$. Now we need to find $g \in Z_p$ with g of order q mod p. Consider the following two algorithms:

Algorithm 1	Algorithm 2
repeat	**repeat**
select $g \in Z_p$	**select** $h \in Z_p$
$h \leftarrow g^q \bmod p$	$g \leftarrow h^{(p-1)/p} \bmod p$
until ($h = 1$ and $g \neq 1$)	**until** ($g \neq 1$)
return g	**return** g

a. Prove that the value returned by Algorithm 1 has order q.
b. Prove that the value returned by Algorithm 1 has order q.
c. Suppose $p = 40193$ and $q = 157$. How many loop iterations do you expect Algorithm 1 to make before it finds a generator?
d. If p is 1024 bits and q is 160 bits, would you recommend using Algorithm 1 to find g? Explain.
e. Suppose $p = 40193$ and $q = 157$. What is the probability that Algorithm 2 computes a generator in its very first loop iteration? (if it is helpful, you may use the fact that $\sum_{d|n} \varphi(d) = n$ when answering this question).

13.14 It is tempting to try to develop a variation on Diffie-Hellman that could be used as a digital signature. Here is one that is simpler than DSA and that does not require a secret random number in addition to the private key.

Public elements:

q prime number
α $\alpha < q$ and α is a primitive root of q

Private key:

X $X < q$

Public key:

$$Y = \alpha^X \bmod q$$

To sign a message M, compute $h = H(M)$, the hash code of the message. We require that $\gcd(h, q-1) = 1$. If not, append the hash to the message and calculate a new hash. Continue this process until a hash code is produced that is relatively prime to $(q-1)$. Then calculate Z to satisfy $Z \times h \equiv X \pmod{q-1}$. The signature of the message is α^Z. To verify the signature, a user verifies that $Y = (\alpha^Z)^h = \alpha^X \bmod q$.

a. Show that this scheme works. That is, show that the verification process produces an equality if the signature is valid.
b. Show that the scheme is unacceptable by describing a simple technique for forging a user's signature on an arbitrary message.

13.15 An early proposal for a digital signature scheme using symmetric encryption is based on the following: To sign an n-bit message, the sender randomly generates in advance 2n 56-bit cryptographic keys:

$$k1, K1, k2, K2, \ldots, kn, Kn$$

which are kept secret. The sender prepares in advance two sets of corresponding nonsecret 64-bit validation parameters, which are made public:

$$u1, V1, u2, V2, \ldots, un, Vn \quad \text{and} \quad v1, V1, v2, V2, \ldots, vn, Vn$$

JPMC Ex. 1041
Page 128

where

$$vi = \mathrm{E}(ki, ui), Vi = \mathrm{E}(ki, Ui)$$

The message M is signed as follows. For the ith bit of the message, either ki or Ki is attached to the message, depending on whether the message bit is 0 or 1. For example, if the first three bits of the message are 011, then the first three keys of the signature are $k1, K2, K3$.

a. How does the receiver validate the message?
b. Is the technique secure?
c. How many times can the same set of secret keys be safely used for different messages?
d. What, if any, practical problems does this scheme present?

JPMC Ex. 1041
Page 129