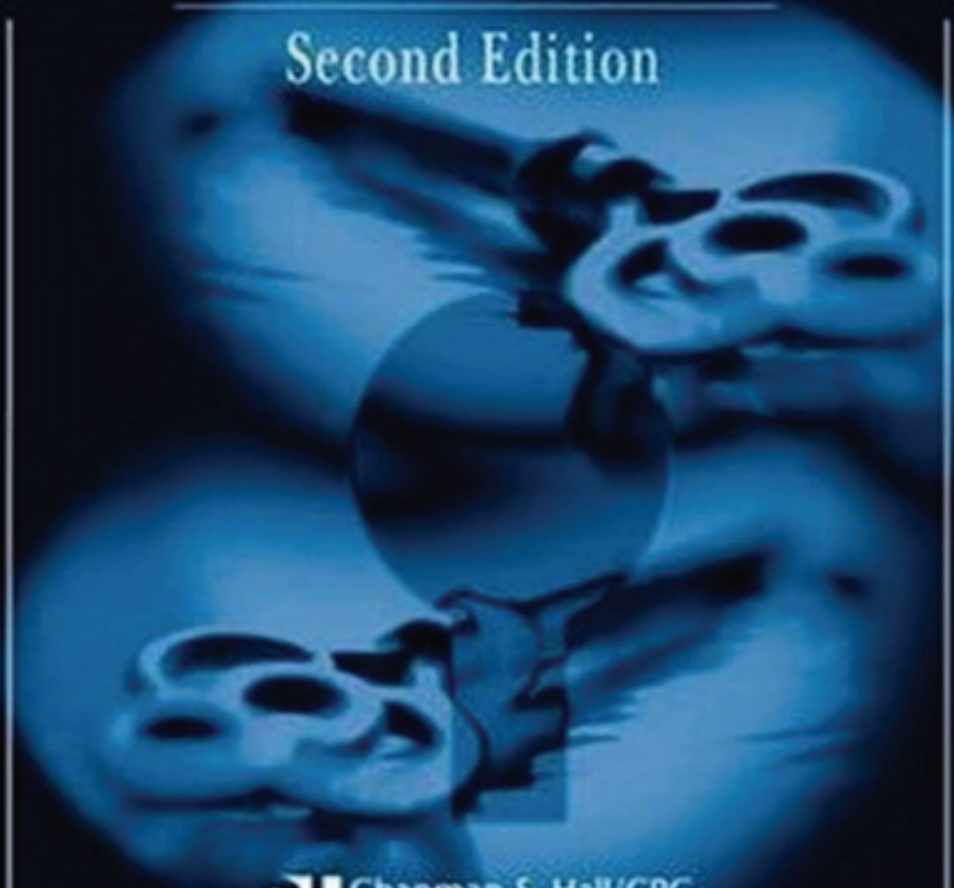
DISCRETE MATHEMATICS AND ITS APPLICATIONS
Series Editor KENNETH H. ROSEN

RICHARD A. MOLLIN

An INTRODUCTION to CRYPTOGRAPHY

Second Edition



Chapman & Hall/CRC
Taylor & Francis Group

DISCRETE MATHEMATICS AND ITS APPLICATIONS
Series Editor KENNETH H. ROSEN

An INTRODUCTION to CRYPTOGRAPHY

Second Edition

DISCRETE MATHEMATICS AND ITS APPLICATIONS

Series Editor
Kenneth H. Rosen, Ph.D.

Juergen Bierbrauer, Introduction to Coding Theory

Kun-Mao Chao and Bang Ye Wu, Spanning Trees and Optimization Problems

Charalambos A. Charalambides, Enumerative Combinatorics

Henri Cohen, Gerhard Frey, et al., Handbook of Elliptic and Hyperelliptic Curve Cryptography

Charles J. Colbourn and Jeffrey H. Dinitz, The CRC Handbook of Combinatorial Designs

Steven Furino, Ying Miao, and Jianxing Yin, Frames and Resolvable Designs: Uses, Constructions, and Existence

Randy Goldberg and Lance Riek, A Practical Handbook of Speech Coders

Jacob E. Goodman and Joseph O'Rourke, Handbook of Discrete and Computational Geometry, Second Edition

Jonathan L. Gross and Jay Yellen, Graph Theory and Its Applications, Second Edition

Jonathan L. Gross and Jay Yellen, Handbook of Graph Theory

Darrel R. Hankerson, Greg A. Harris, and Peter D. Johnson, Introduction to Information Theory and Data Compression, Second Edition

Daryl D. Harms, Miroslav Kraetzl, Charles J. Colbourn, and John S. Devitt, Network Reliability: Experiments with a Symbolic Algebra Environment

Leslie Hogben, Handbook of Linear Algebra

Derek F. Holt with Bettina Eick and Eamonn A. O'Brien, Handbook of Computational Group Theory

David M. Jackson and Terry I. Visentin, An Atlas of Smaller Maps in Orientable and Nonorientable Surfaces

Richard E. Klima, Neil P. Sigmon, and Ernest L. Stitzinger, Applications of Abstract Algebra with Maple™ and MATLAB® , Second Edition

Patrick Knupp and Kambiz Salari, Verification of Computer Codes in Computational Science and Engineering

William Kocay and Donald L. Kreher, Graphs, Algorithms, and Optimization

Donald L. Kreher and Douglas R. Stinson, Combinatorial Algorithms: Generation Enumeration and Search

Continued Titles

Charles C. Lindner and Christopher A. Rodgers, Design Theory

Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, Handbook of Applied Cryptography

Richard A. Mollin, Algebraic Number Theory

Richard A. Mollin, Codes: The Guide to Secrecy from Ancient to Modern Times

Richard A. Mollin, Fundamental Number Theory with Applications

Richard A. Mollin, An Introduction to Cryptography, Second Edition

Richard A. Mollin, Quadratics

Richard A. Mollin, RSA and Public-Key Cryptography

Carlos J. Moreno and Samuel S. Wagstaff, Jr., Sums of Squares of Integers

Dingyi Pei, Authentication Codes and Combinatorial Designs

Kenneth H. Rosen, Handbook of Discrete and Combinatorial Mathematics

Douglas R. Shier and K.T. Wallenius, Applied Mathematical Modeling: A Multidisciplinary Approach

Jörn Steuding, Diophantine Analysis

Douglas R. Stinson, Cryptography: Theory and Practice, Third Edition

Roberto Togneri and Christopher J. deSilva, Fundamentals of Information Theory and Coding Design

Lawrence C. Washington, Elliptic Curves: Number Theory and Cryptography

DISCRETE MATHEMATICS AND ITS APPLICATIONS
Series Editor KENNETH H. ROSEN

An INTRODUCTION to CRYPTOGRAPHY

Second Edition

RICHARD A. MOLLIN

Chapman & Hall/CRC
Taylor & Francis Group
Boca Raton London New York

Chapman & Hall/CRC is an imprint of the
Taylor & Francis Group, an informa business

Chapman & Hall/CRC
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-10: 1-58488-618-8 (Hardcover)
International Standard Book Number-13: 978-1-58488-618-1 (Hardcover)

Library of Congress Cataloging-in-Publication Data

Mollin, Richard A., 1947-
An Introduction to Cryptography / Richard A. Mollin. -- 2nd ed.
p. cm. -- (Discrete mathematics and its applications)
Includes bibliographical references and index.
ISBN-13: 978-1-58488-618-1 (acid-free paper)
ISBN-10: 1-58488-618-8 (acid-free paper)
1. Coding theory--Textbooks. I. Title. II. Series.

QA268.M65 2007
003'.54--dc22 2006049639

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

To Kathleen Ellen — my Soul Mate.

Contents

Preface

The second edition of the original introductory undergraduate text for a one-semester course in cryptography is redesigned to be more accessible. This includes the decision to include many items of contemporary interest not contained in the first edition, such as electronic mail and Internet security, and some leading-edge applications. The former comprises the history of the WWW, PGP, protocol layers, SSL, firewalls, client-server models, and cookies, all contained in Chapter 7. The latter encompasses login and network security, viruses and other computer infections, as well as smart cards and biometrics, making up the closing Chapter 8 of the main text. In the appendices, we retained the data on fundamental mathematical facts. However, instead of leading each chapter with mathematical background to each of the cryptographic concepts, we have placed all mathematical basics in Chapter 1, and we have placed all cryptographic basics in Chapter 2. In this fashion, all essential background material is grounded at the outset.

Symmetric and public-key cryptosystems comprise Chapters 3 and 4, respectively, with the addition of the digital signature standard at the end of Chapter 4, not contained in the first edition. In order to make the presentation of DES more palatable to the reader, we have included a new discussion of S-DES ("baby DES") as a preamble to DES in Chapter 3.

We maintain the coverage of factoring and primality testing in Chapters 5 and 6, respectively. However, we include a wealth of new aspects of "recognizing" primes in Chapter 5, including the recent discovery of an unconditional deterministic polynomial-time algorithm for primality testing. Furthermore, instead of the more advanced number field sieve, which we have excluded in this edition, we have placed the elliptic curve method in Chapter 6. We have, nevertheless, excluded the chapter on advanced topics — the more advanced elliptic curve cryptography, the coverage of zero knowledge — and have placed quantum cryptography in an appendix but deleted the more advanced exposition on quantum computing. This has reduced the number of entries in the bibliography because the first edition had a large number of references to those advanced topics and points to the greater accessibility of this edition. We have added Pollard's two algorithms, the $p-1$ and rho factoring methods in Chapter 6, and lead the chapter with classical factoring methods with more breadth than the first edition.

Other than Appendix A on mathematical facts, we have included eight other appendices on computer arithmetic, which was part of Chapter 1 of the first edition; the Rijndael S-Box, also an appendix in the first edition; knapsack ciphers, which was part of Chapter 3 of the first edition; the Silver-Pohlig-Hellman Algorithm; the SHA-1 algorithm; and radix-64 encoding, the latter three not included in the first edition, and quantum cryptography in the concluding Appendix H.

The numbering system has been changed from the global approach in the first edition to the standard numbering found in most texts. The use of footnotes has been curtailed in this edition. For instance, the mini-biographies are placed

in highlighted boxes as sidebars to reduce distraction and impinging on text of footnote usage. Footnotes are employed only when no other mechanisms will work. Also, the bibliography contains the page(s) where each entry is cited, another new inclusion.

A course outline for the second edition would be to cover the Chapters 1–6 and, if time allows, include topics of interest from Chapters 7–8. The instructor may include or exclude material, depending upon the needs and background of the students, that is deemed to be more advanced, as flagged by the symbol: ☞. Use of the material from the appendices, as needed, is advised.

There are more than 300 exercises in this edition, and there are nearly sixty mini-biographies, both of which exceed the first edition. (As with the first edition, the more challenging exercises are marked with the ✯ symbol.) Similarly the index, consisting of roughly 2,600 entries, surpasses the first edition. As with the first edition, solutions of the odd-numbered exercises are included at the end of the text, and a solutions manual for the even-numbered exercises is available to instructors who adopt the text for a course. As usual, the website below is designed for the reader to access any updates and the e-mail address below is available for any comments.

◆ **Acknowledgments** The author is grateful for the proofreading done by the following people, each of whom lent their own valuable time: John Burke (U.S.A.) Jacek Fabrykowski (U.S.A.) Bart Goddard (U.S.A.) and Thomas Zaplachinski (Canada) a former student, now cryptographer. Thanks also to John Callas of PGP corporation for comments on Section 7.2, which helped update the presentation of PGP.

August 10, 2006

website: http://www.math.ucalgary.ca/~ramollin/

e-mail: ramollin@math.ucalgary.ca

Chapter 1

Mathematical Basics

In this introductory chapter, we set up the basics for number theoretic concepts in the first seven sections and the basics for complexity in the last section. This will provide us with the foundations to study the cryptographic notions later in the book. Indeed, this material, together with Appendices A–B, comprise all the requisite background material in number theory and algorithmic complexity needed throughout the text.

1.1 Divisibility

For background on notation, sets, number systems, and other fundamental facts, the reader should consult Appendix A.

Definition 1.1 **Division**

If $a, b \in \mathbb{Z}$, $b \neq 0$, *then to say that* b divides a, *denoted by* $b|a$, *means that* $a = bx$ *for a* unique $x \in \mathbb{Z}$, *denoted by* $x = a/b$. *Note that the existence and uniqueness of* x *implies that* b *cannot be* 0. *We also say that* a *is* divisible *by* b. *If* b *does* not *divide* a, *then we write* $b \nmid a$ *and say that* a *is* not divisible *by* b. *We say that* division by zero is undefined.

We may classify integers according to whether they are divisible by 2, as follows.

Definition 1.2 **Parity**

If $a \in \mathbb{Z}$, *and* $a/2 \in \mathbb{Z}$, *then we say that* a *is an* even integer. *In other words, an even integer is one which is divisible by* 2. *If* $a/2 \notin \mathbb{Z}$, *then we say that* a *is an* odd *integer. In other words, an odd integer is one which is not divisible by* 2. *If two integers are either both even or both odd, then they are said to have the same* parity. *Otherwise they are said to have* opposite *or* different parity.

In order to prove our first result, we need a concept that will be valuable throughout.

Definition 1.3 **The Floor Function**

If $x \in \mathbb{R}$, then there is a unique integer n such that $n \leq x < n+1$. We say that n is the greatest integer less than or equal to x, sometimes called the floor of x, denoted by $\lfloor x \rfloor = n$.

The reader may test understanding of the floor function by solving Exercises 1.12–1.19 on pages 4–5. Indeed, we will need one of those exercises to establish the following algorithm, which is of particular importance for divisibility.

Theorem 1.1 **The Division Algorithm**

If $a \in \mathbb{N}$ and $b \in \mathbb{Z}$, then there exist unique integers $q, r \in \mathbb{Z}$ with $0 \leq r < a$, and $b = aq + r$.

Proof. There are two parts to prove, the first of which is existence, and the second of which is uniqueness.

Given $a \in \mathbb{N}$, $b \in \mathbb{Z}$, we may form $\lfloor b/a \rfloor = q \in \mathbb{Z}$. Therefore, $b = aq + r$ with $q, r \in \mathbb{Z}$. If $r \geq a$, then $b = a\lfloor b/a \rfloor + r \geq a\lfloor b/a \rfloor + a > a(b/a - 1) + a = b$, where the last inequality follows from Exercise 1.15 (which says that $x - 1 < \lfloor x \rfloor \leq x$). This is a contradiction, which establishes that $r < a$.

If $r < 0$, then $b = a\lfloor b/a \rfloor + r \leq a(b/a) + r = b + r < b$, where the first inequality follows from Exercise 1.15 again. This contradiction establishes that $0 \leq r < a$. We have shown the existence of the integers q and r as required. The final step is to show uniqueness.

If $b = aq_i + r_i$ for $i = 1, 2$ with $0 \leq r_i < a$, then we may subtract the two equations to get $a(q_1 - q_2) = r_2 - r_1$. Since $-a < -r_1 < 0 < r_2 < a$, $r_2 - a < r_2 - r_1 < a - r_1$. Dividing through the inequality by a, we deduce that $-1 < (r_2 - r_1)/a < 1$. Since $(r_2 - r_1)/a = q_1 - q_2 \in \mathbb{Z}$, $q_1 - q_2 = 0$. In other words, $q_1 = q_2$ from which it follows that $r_1 = r_2$. This establishes uniqueness, and we have the division algorithm. □

Now we look more closely at our terminology. To say that b divides a is to say that a is a *multiple* of b and that b is a *divisor* of a. Also, note that b dividing a is equivalent to the remainder upon dividing a by b is zero. Any divisor $b \neq a$ of a is called a *proper divisor* of a. If we have two integers a and b, then a *common divisor* of a and b is a natural number n which is a divisor of *both* a and b. There is a special kind of common divisor that deserves singular recognition. Properties of the following are developed in Exercises 1.20–1.30 on page 5.

Definition 1.4 **The Greatest Common Divisor**

If $a, b \in \mathbb{Z}$ are not both zero, then the[1.1] greatest common divisor *or* gcd *of a and b is the natural number g such that $g|a$, $g|b$, and g is divisible by any common divisor of a and b, denoted by $g = \gcd(a, b)$.*

[1.1] The word "the" is valid here since g is indeed unique. See Exercise 1.23.

We have a special term for the case where the gcd is 1.

Definition 1.5 **Relative Primality**

If $a, b \in \mathbb{Z}$, and $\gcd(a, b) = 1$*, then a and b are said to be* relatively prime *or* coprime*. Sometimes the phrase a* is prime to *b is also used.*

By applying the Division Algorithm, we get the following. The reader should solve Exercise 1.20 on page 5 first, since we use it in the proof.

Theorem 1.2 **The Euclidean Algorithm**

Let $a, b \in \mathbb{Z} \quad (a \geq b > 0)$, and set $a = r_{-1}, b = r_0$. By repeatedly applying the Division Algorithm, we get $r_{j-1} = r_j q_{j+1} + r_{j+1}$ with $0 < r_{j+1} < r_j$ for all $0 \leq j < n$, where n is the least nonnegative number such that $r_{n+1} = 0$, in which case $\gcd(a, b) = r_n$.

Proof. The sequence $\{r_i\}$, produced by repeated application of the division algorithm, is a strictly decreasing sequence bounded below, and so stops for some nonnegative integer n with $r_{n+1} = 0$. By Exercise 1.20,

$$\gcd(a, b) = \gcd(r_i, r_{i+1})$$

for any $i \geq 0$, so in particular, $\gcd(a, b) = \gcd(r_n, r_{n+1}) = r_n$. □

It is easily seen that any common divisor of $a, b \in \mathbb{Z}$ is also a divisor of an expression of the form $ax + by$ for $x, y \in \mathbb{Z}$. Such an expression is called a *linear combination* of a and b. The greatest common divisor is a special kind of linear combination. By Exercise 1.22, the least positive value of $ax + by$ for any $x, y \in \mathbb{Z}$, is $\gcd(a, b)$.

We will also need a concept, closely related to the gcd, as follows.

Biography 1.1 Euclid of Alexandria (ca. 300 B.C.) *is the author of the* Elements*. Next to the Bible, the* Elements *is the most reproduced book in recorded history. Little is known about Euclid's life, other than that he lived and taught in Alexandria. However, the folklore is rich with quotes attributed to Euclid. For instance, he is purported to have been a teacher of the ruler Ptolemy I, who reigned from 306 to 283 B.C. When Ptolemy asked if there were an easier way to learn geometry, Euclid ostensibly responded that* there is no royal road to geometry*. His nature as a* purist *is displayed by another quotation. A student asked Euclid what use could be made of geometry, to which Euclid responded by having the student handed some coins, saying that the student had to* make gain from what he learns.

Definition 1.6 **The Least Common Multiple**

If $a, b \in \mathbb{Z}$, then the[1.2] *smallest natural number which is a multiple of both a and b is the* least common multiple *of a and b, denoted by* $\operatorname{lcm}(a, b)$.

[1.2]Here the uniqueness of the lcm follows from the uniqueness of the gcd via Exercise 1.36.

For instance, if $a = 22$ and $b = 14$, then $\gcd(a, b) = 2$, and $\text{lcm}(a, b) = 154$.

Properties of the lcm are developed in Exercises 1.31–1.34 and relative properties of the gcd and lcm are explored in Exercises 1.35–1.36.

Exercises

1.1. Prove that if $a, b \in \mathbb{Z}$ and $ab = 1$, then either $a = b = 1$ or $a = b = -1$.

1.2. Prove that if $a \in \mathbb{Z}$ and $a|1$, then either $a = 1$ or $a = -1$.

1.3. Prove that if $a, b \in \mathbb{Z}$ are nonzero with $a|b$ and $b|a$, then $a = \pm b$.

1.4. Prove each of the following.

(a) If $a, b, c \in \mathbb{Z}$ with $a \neq 0$, and $a|b$, $a|c$, then $a|(bx+cy)$ for any $x, y \in \mathbb{Z}$.

(b) If $a|b$ and $b|c$, then $a|c$ for $a, b, c \in \mathbb{Z}$, $(a, b \neq 0)$, called the *Transitive Law for Division*.

1.5. Prove that the square of an odd integer bigger than 1 is of the form $8n+1$ for some $n \in \mathbb{N}$.

1.6. Prove that if $a, b \in \mathbb{Z}$ with $a|b$, then $a^n|b^n$ for any $n \in \mathbb{N}$.

1.7. Prove that if $a, b, c \in \mathbb{Z}$ with $a, c \neq 0$, then $a|b$ if and only if $ca|cb$.

1.8. Prove that if $a, b, c, d \in \mathbb{Z}$ with $a, c \neq 0$, $a|b$, and $c|d$, then $ac|bd$.

1.9. Find integers x, y such that $3x + 7y = 1$.

1.10. Find the gcd of each of the following pairs.

(a) $a = 22$, $b = 55$. (b) $a = 15$, $b = 113$.

1.11. Find the least common multiple (lcm) of the following pairs.

(a) $a = 15$, $b = 385$.

(b) $a = 28$, $b = 577$.

(c) $a = 73$, $b = 561$.

(d) $a = 110$, $b = 5005$.

1.12. There is a function that is a close cousin of the greatest integer function (see Definition 1.3 on page 2). It is the *ceiling* defined for all $x \in \mathbb{R}$, as that unique integer $m \in \mathbb{Z}$ such that $x \leq m < x + 1$, denoted by $\lceil x \rceil$. It is also called the *least integer function*. Prove that, if $x \in \mathbb{R}$, then $-\lfloor -x \rfloor = \lceil x \rceil$.

1.13. With reference to Exercise 1.12, prove each of the following.

(a) For any $x \in \mathbb{R}$, $\lceil x \rceil = \lfloor x \rfloor + 1$ if and only if $x \notin \mathbb{Z}$.

(b) $\lfloor x + 1/2 \rfloor$ is the nearest integer to x. (When two integers are equally near each other we choose the larger of the two as the nearest. The function $Ne(x) = \lfloor x + 1/2 \rfloor$ is the *nearest integer function*.)

1.14. Prove that, if $n, m \in \mathbb{N}$ with $n \geq m$, then $\lfloor n/m \rfloor$ is the number of natural numbers that are less than or equal to n and divisible by m.

1.15. Establish the inequality $x - 1 < \lfloor x \rfloor \leq x$.

1.16. Prove that $\lfloor x + n \rfloor = \lfloor x \rfloor + n$ for any $n \in \mathbb{Z}$.

1.17. Prove that $\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + 1$.

1.18. Establish that $\lfloor x \rfloor + \lfloor -x \rfloor = \begin{cases} 0 & \text{if } x \in \mathbb{Z}, \\ -1 & \text{otherwise.} \end{cases}$

1.19. Prove that, if $n \in \mathbb{N}$ and $x \in \mathbb{R}$, then $\lfloor \lfloor x \rfloor / n \rfloor = \lfloor x/n \rfloor$.

1.20. Prove that if $a, b \in \mathbb{Z}$ with $b = aq + r$, then $\gcd(a, b) = \gcd(a, r)$.

1.21. Prove that if $a, b \in \mathbb{Z}$ and $c \in \mathbb{N}$, c divides both a and b, and c is divisible by every common divisor of a and b, then $c = \gcd(a, b)$.

1.22. If $a, b \in \mathbb{Z}$, $g = \gcd(a, b)$, then the least positive value of $ax + by$ for any $x, y \in \mathbb{Z}$ is g. (*Hint: Use the* Well-Ordering Principle *cited on page 8.*)

1.23. Given $a, b \in \mathbb{Z}$, prove that $\gcd(a, b)$ is unique.

1.24. Show that for any $m \in \mathbb{N}$, $mg = \gcd(ma, mb)$.

1.25. If $a, b \in \mathbb{Z}$, prove that $\gcd(a, b) = a$ if and only if $a|b$.

1.26. Let $a, b, c \in \mathbb{Z}$. Prove that if $c|ab$ and $\gcd(b, c) = 1$, then $c|a$. (This is called *Euclid's Lemma.*)

1.27. Given $a, b \in \mathbb{Z}$, $c \in \mathbb{N}$ where c is a common divisor of a and b, prove that $\gcd(a/c, b/c) = g/c$.

1.28. If $a, b \in \mathbb{Z}$, and $g = \gcd(a, b)$, show that $\gcd(a/g, b/g) = 1$.

1.29. If $a, b \in \mathbb{Z}$, prove that for any $m \in \mathbb{Z}$, $\gcd(a, b) = \gcd(b, a) = \gcd(a, b+am)$.

☆ 1.30. If $k, \ell, n \in \mathbb{N}$ with $n > 1$, prove that $\gcd(n^k - 1, n^\ell - 1) = n^{\gcd(k,\ell)} - 1$.

1.31. Let $\ell = \operatorname{lcm}(a, b)$ for $a, b \in \mathbb{Z}$. Prove that $\ell = b$ if and only if $a|b$.

1.32. Prove that $\operatorname{lcm}(a, b)$ is a divisor of all common multiples of a and b.

1.33. With the same notation as in Exercise 1.31, prove that $\ell \leq ab$.

1.34. If $a, b, c \in \mathbb{Z}$ and $\operatorname{lcm}(a, b) = \ell$, show that If $c|a$ and $c|b$, then $\operatorname{lcm}(a/c, b/c) = \ell/c$.

1.35. With the same notation as in Exercise 1.31, prove that if $\gcd(a, b) = g = 1$, then $\ell = ab$.

1.36. Let $a, b \in \mathbb{N}, \ell = \operatorname{lcm}(a, b)$, and $g = \gcd(a, b)$. Prove that $\ell g = ab$.

1.2 Primes, Primality Testing, and Induction

Two of the features of this text are the roles played by primality testing and factoring in cryptography, which we will study in detail later in Chapters 5 and 6. In this section, we set out the basic notions behind these important areas, as well as one of the fundamental tools of study, the Principle of Mathematical Induction.

The definition of a *prime number* (or simply a *prime*) is a natural number bigger than 1, that is not divisible by any natural number except itself and 1. The first recorded definition of a prime was given by Euclid around 300 B.C. in his *Elements*. However, there is some indirect evidence that the concept of primality must have been known earlier to Aristotle (ca. 384–322 B.C.), for instance, and probably to Pythagoras (see Biography 1.3 on page 7). If $n \in \mathbb{N}$ and $n > 1$ is *not* prime, then n is called *composite*.

Biography 1.2 *The* Greeks of antiquity *used the term* arithmetic *to mean what we consider today to be* number theory*, namely the study of the properties of the natural numbers and the relationships between them. They reserved the word* logistics *for the study of ordinary computations using the standard operations of addition/subtraction and multiplication/division, which we call arithmetic today. The Pythagoreans (see Biography 1.3 on page 7) introduced the term* mathematics*, which to them meant the study of arithmetic, astronomy, geometry, and music. These became known as the* quadrivium *in the Middle Ages. See Appendix A for the* Fundamental Laws of Arithmetic.

The *Factoring Problem* is the determination of the prime factorization of a given $n \in \mathbb{N}$ guaranteed by The Fundamental Theorem of Arithmetic (see Theorem 1.3 on page 9). This theorem says that the primes in the factorization of a given natural number n are unique to n up to order of the factors. Thus, the primes are the fundamental *atoms* or multiplicative building blocks of arithmetic as well as its more elevated relative *the higher arithmetic*, also known as *number theory*.

Eratosthenes (ca. 284–204 B.C.) gave us the first notion of a *sieve*, which was what he called his method for finding primes. The following example illustrates the *Sieve of Eratosthenes*. (In general, we may think of a *sieve* as any process whereby we find numbers by searching up to a prescribed bound and eliminating candidates as we proceed until only the desired solution set remains.)

Example 1.1 Suppose that we want to find all primes less than 30. First, we write down all natural numbers less than 30 and bigger than 1, and cross out all numbers (bigger than 2) that are multiples of 2, the smallest prime:

$$\{2, 3, \not{4}, 5, \not{6}, 7, \not{8}, 9, \not{10}, 11, \not{12}, 13, \not{14}, 15, \not{16}, 17, \not{18}, 19, \not{20}, 21, \not{22},$$
$$23, \not{24}, 25, \not{26}, 27, \not{28}, 29, \not{30}\}.$$

Next, we cross out all numbers (bigger than 3) that are multiples of 3, the

next prime: $\{2, 3, 5, 7, \not{9}, 11, 13, \not{15}, 17, 19, \not{21}, 23, 25, \not{27}, 29\}$. Then we cross out all numbers (bigger than 5) that are multiples of 5, the next prime:[1.3]

$$\{2, 3, 5, 7, 11, 13, 17, 19, 23, \not{25}, 29\}.$$

What we have left is the set of primes less than 30.

$$\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29\}.$$

The sieve of Eratosthenes illustrated in Example 1.1 clearly works well, but it is highly inefficient. This sieve represents the only known algorithm from antiquity that could come remotely close to what we call primality testing today. We should agree upon what we mean by *primality testing*. A *primality test* is an algorithm the steps of which verify the hypothesis of a theorem the conclusion of which is: "n is prime." (For now, we may think loosely of an algorithm as any methodology following a set of rules to achieve a goal. More precisely, later, when we discuss complexity theory, we will need the definition of an algorithm as a well-defined [see page 298 in Appendix A] computational procedure, which takes a variable input and halts with an output.)

Arab scholars helped enlighten the exit from Europe's Dark Ages, and they were primarily responsible for preserving much of the mathematics from antiquity, as well as for extending some of the ideas. For instance, Eratosthenes did not address the issue of termination in his algorithm. However, Ibn al-Banna (ca. 1258–1339) appears to have been the first to observe that, in order to find the primes less than n using the sieve of Eratosthenes, one can restrict attention to prime divisors less than $\sqrt{n}$.

Biography 1.3 Pythagorus *lived from roughly 580 to 500 B.C., although little is known about his life with any degree of accuracy. He is not known to have written any books, but his followers carried on his legacy. The most famous result bearing his name, although known to the Babylonians, is the theorem that says that the square of the hypotenuse of a right-angled triangle is equal to the sum of the squares of the other two sides. Nevertheless, Pythagorus is undoubtedly the first to* prove *this. He is thought to have traveled to Egypt and Babylonia and settled in Crotona on the southeastern coast of* Magna Graecia*, now Italy, where he founded a secret society that became known as the* Pythagoreans*. Their motto,* number rules the universe*, reflected the mysticism embraced by Pythagorus, who was more of a mystic and a prophet than a scholar. The Pythagoreans' belief that everything was based on the natural numbers was deeply rooted. The degree of their commitment to this belief is displayed by an anecdote about* $\sqrt{2}$*. Hippasus was a Pythagorean who revealed to outsiders the* secret *that* $\sqrt{2}$ *is irrational. For this indiscretion, he was drowned by his comrades.*

[1.3] We need not check any primes bigger than 5 since such primes are larger than $\sqrt{30}$. See the above paragraph for the historical description of this fact.

The resurrection of mathematical interest in Europe during the thirteenth century is perhaps best epitomized by the work of Fibonacci.

Biography 1.4 Fibonacci (ca.1180–1250) *was known as* Leonardo of Pisa, *the son of an Italian merchant named Bonaccio. He had an Arab scholar as his tutor while his father served as consul in North Africa. Thus, he was well educated in the mathematics known to the Arabs. Fibonacci's first and certainly his best-known book is* Liber Abaci *or* Book of the Abacus *first published in 1202, which was one of the means by which the Hindu-Arabic number system was transmitted into Europe (see also Biography 1.9 on page 34). However, only the second edition, published in 1228, has survived. In this work, Fibonacci gave an algorithm to determine if n is prime by dividing n by natural numbers up to $\sqrt{n}$. This represents the first recorded instance of a* Deterministic Algorithm *for primality testing, where* deterministic *means that the algorithm always terminates with either a* yes *answer or a* no *answer. Also included in his book was the rabbit problem described below.*

◆ The Rabbit Problem

Suppose that a male rabbit and a female rabbit have just been born. Assume that any given rabbit reaches sexual maturity after one month and that the gestation period for a rabbit is one month. Furthermore, once a female rabbit reaches sexual maturity, it will give birth every month to exactly one male and one female. Assuming that no rabbits die, how many male/female pairs are there after n months?

The answer is given by the *Fibonacci Sequence* $\{F_n\}$:

$$F_1 = F_2 = 1,$$

$$F_n = F_{n-1} + F_{n-2} \quad (n \geq 3)$$

where F_n is the *nth Fibonacci Number*. (A research journal devoted entirely to the study of such numbers is the *Fibonacci Quarterly*.) The answer to the rabbit problem is F_n pairs of rabbits (see Exercise 1.37 on page 15). Later, we will see the influence of Fibonacci Numbers in the history of primality testing.

Before we turn to the notion of induction, we need the following important topic.

◆ The Well-Ordering Principle

Every nonempty subset of $\mathbb{N}$ contains a least element.

This proof of the following fundamental result, which is sometimes called the *Unique Factorization Theorem* for integers, demonstrates the power of the Well-Ordering Principle. In advance, the reader should solve Exercise 1.38 on page 15, which we use in the following proof.

Theorem 1.3 **The Fundamental Theorem of Arithmetic**

Let $n \in \mathbb{N}, n > 1$. Then n has a factorization into a product of prime powers (existence). Moreover, if $n = \prod_{i=1}^{r} p_i = \prod_{i=1}^{s} q_i$, where the p_i and q_i are primes, then $r = s$, and the factors are the same if their order is ignored (uniqueness).

Proof. We must first show that every natural number $n > 1$ can be written as a product of primes. If there exists a natural number (bigger than 1) that is not a product of primes, then there exists a smallest such one, by the Well-Ordering Principle. If n is this number, then n must be composite since any prime is trivially a product of a set of primes, namely itself. Let $n = rs$ with $1 < r < n$ and $1 < s < n$. Since n is the smallest, r and s are products of primes. However, $n = rs$, so n is a product of primes, a contradiction.

Now we establish the uniqueness of such factorizations. Again we use proof by contradiction to establish it. Let $n > 1$, and $n = \prod_{i=1}^{r} p_i = \prod_{i=1}^{s} q_i$ be the smallest natural number (bigger than 1) that does not have unique factorization. Suppose that $p_i = q_j$ for some i, j, then since the order of the factors does not matter, we may let $p_1 = q_1$. If $n = p_1$, then we are done, so assume that $n > p_1$. Since $1 < n/p_1 < n$, n/p_1 has unique factorization, and so $n/p_1 = \prod_{i=2}^{r} p_i = \prod_{i=2}^{s} q_i$, with $r = s$ and $p_i = q_i$ for all $i = 1, 2, \ldots, r = s$. Since $n = p_1 \prod_{i=2}^{r} p_i = q_1 \prod_{i=2}^{s} q_i$, n has unique factorization, a contradiction. Hence, $p_i \neq q_j$ for all i, j. However, by Exercise 1.38, since $p_1 | \prod_{i=1}^{s} q_i$, then $p_1 | q_j$ for some j. Therefore, $p_1 = q_j$, a contradiction, so we have established unique factorization. □

For example, $617,400 = 2^3 \cdot 3^2 \cdot 5^2 \cdot 7^3$. Before leaving the discussion of primes it is worthy of note that one of the most elegant proofs to remain from antiquity is Euclid's proof of the infinitude of primes. Suppose that $p_1, p_2, \ldots, p_n$ for $n \in \mathbb{N}$ are all of the primes. Then set $N = \prod_{j=1}^{n} p_j$. Since $N + 1 > p_j$ for any natural number $j \leq n$, then $N + 1$ must be composite. Hence, $p_j \mid (N + 1)$ for some such j by the Fundamental Theorem of Arithmetic. Since $p_j \mid N$, then $p_j \mid N + 1 - N = 1$, a contradiction.

Any nonempty set, denoted by $\mathcal{S} \neq \varnothing$, with $\mathcal{S} \subseteq \mathbb{Z}$, having a least element is said to be *well-ordered*. For instance, $\mathbb{N}$ is *well-ordered*. The Well-Ordering Principle is sometimes called the *Principle of the Least Element*.

Later we will show that the Well-Ordering Principle is equivalent to the following important principle.

◆ **The Principle of Mathematical Induction**

Suppose that $\mathcal{S} \subseteq \mathbb{N}$. If

(a) $1 \in \mathcal{S}$, and

(b) If $n > 1$ and $n - 1 \in \mathcal{S}$, then $n \in \mathcal{S}$,

then $\mathcal{S} = \mathbb{N}$.

In other words, the Principle of Mathematical Induction says that any subset of the natural numbers that contains 1 and can be shown to contain $n > 1$

whenever it contains $n-1$ must *be* $\mathbb{N}$. Part (a) is called the *induction step*, and the assumption that $n \in \mathcal{S}$ is called the *induction hypothesis*. Typically, one establishes the induction step, then assumes the induction hypothesis and proves the conclusion, that $n \in \mathcal{S}$. Then we simply say that *by induction*, $n \in \mathcal{S}$ for all $n \in \mathbb{N}$ (so $\mathcal{S} = \mathbb{N}$).

Induction, in practice, is illustrated in the following two results.

Theorem 1.4 **A Summation Formula**

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}.$$

Proof. If $n = 1$, then $\sum_{j=1}^{n} j = 1 = n(n+1)/2$, and the induction step is secured. Assume that

$$\sum_{j=1}^{n-1} j = (n-1)n/2,$$

the induction hypothesis. Now consider

$$\sum_{j=1}^{n} j = n + \sum_{j=1}^{n-1} j = n + (n-1)n/2,$$

by the induction hypothesis. Hence,

$$\sum_{j=1}^{n} j = [2n + (n-1)n]/2 = (n^2+n)/2 = n(n+1)/2,$$

as required. Hence, by induction, this must hold for all $n \in \mathbb{N}$. □

Theorem 1.5 **A Geometric Formula**

If $a, r \in \mathbb{R}$, $r \neq 0, 1$, $n \in \mathbb{N}$, then

$$\sum_{j=0}^{n} ar^j = \frac{a(r^{n+1}-1)}{r-1}.$$

Proof. If $n = 1$, then

$$\sum_{j=0}^{n} ar^j = a + ar = a(1+r) = a(1+r)(r-1)/(r-1) = a(r^2-1)/(r-1) =$$

$$a(r^{n+1}-1)/(r-1),$$

which is the induction step. By the induction hypothesis, we get,

$$\sum_{j=0}^{n+1} ar^j = ar^{n+1} + \sum_{j=0}^{n} ar^j = ar^{n+1} + a(r^{n+1}-1)/(r-1) = a(r^{n+2}-1)/(r-1),$$

as required. □

The sum in Theorem 1.5 is called a *geometric sum* where a is the *initial term* and r is called the *ratio*.

There is another form of induction given in the following. We will show that this form is actually equivalent to the first, but this is not obvious at first glance. Moreover, perhaps even less obvious, both forms of induction will be shown to be equivalent to the Well-Ordering Principle.

◆ The Principle of Mathematical Induction (Second Form)

Suppose that $\mathcal{S} \subseteq \mathbb{Z}$, and $m \in \mathbb{Z}$ with

(a) $m \in \mathcal{S}$, and

(b) If $m < n$ and $\{m, m+1, \ldots, n-1\} \subseteq \mathcal{S}$, then $n \in \mathcal{S}$.

Then $k \in \mathcal{S}$ for all $k \in \mathbb{Z}$ such that $k \geq m$.

An illustration of the use of this form of induction is as follows where we employ Fibonacci numbers defined on page 8. In what follows,

$$\mathfrak{g} = \frac{1+\sqrt{5}}{2},$$

called the *golden ratio*. Since we use Exercise 1.39 on page 15 in the following, the reader should solve it in advance.

Theorem 1.6 **Fibonacci Dominates the Golden Ratio**

For any $n \in \mathbb{N}$, $F_n \geq \mathfrak{g}^{n-2}$.

Proof. We use the Principle of Induction in its second form. We need to handle $n = 1, 2$ separately since $F_n = F_{n-1} + F_{n-2}$ only holds for $n \geq 3$. If $n = 1$, then

$$F_n = 1 > \frac{1}{\mathfrak{g}} = \mathfrak{g}^{n-2} = \frac{2}{1+\sqrt{5}}.$$

Also, if $n = 2$, then $F_2 = 1 = \mathfrak{g}^0 = \mathfrak{g}^{n-2}$. This establishes the induction step. Now assume that $F_m \geq \mathfrak{g}^{m-2}$ for all $m \in \mathbb{N}$ with $m \leq n$, the induction hypothesis. By the induction hypothesis

$$F_{n+1} = F_n + F_{n-1} \geq \mathfrak{g}^{n-2} + \mathfrak{g}^{n-3} = \mathfrak{g}^{n-3}(\mathfrak{g}+1).$$

By Exercise 1.39, $(\mathfrak{g}+1) = \mathfrak{g}^2$, so

$$F_{n+1} \geq \mathfrak{g}^{n-3}\mathfrak{g}^2 = \mathfrak{g}^{n-1}.$$

By the Principle of Induction (second form) we have proved that this holds for all $n \in \mathbb{N}$. □

Another application of induction is the following more general version of the Euclidean algorithm (Theorem 1.2 presented on page 3).

Theorem 1.7 **Extended Euclidean Algorithm**.

Let $a, b \in \mathbb{N}$, and let q_i for $i = 1, 2, \ldots, n+1$ be the quotients obtained from the application of the Euclidean Algorithm to find $g = \gcd(a, b)$, where n is the least nonnegative integer such that $r_{n+1} = 0$. If $s_{-1} = 1$, $s_0 = 0$, and

$$s_i = s_{i-2} - q_{n-i+2} s_{i-1},$$

for $i = 1, 2, \ldots, n+1$, then

$$g = s_{n+1} a + s_n b.$$

Proof. We use induction to prove that the remainders obtained by application of the Euclidean algorithm satisfy

$$r_n = s_i r_{n-i+1} + s_{i-1} r_{n-i} \text{ for all } i = 0, 1, \ldots, n+1.$$

If $i = 0$, then

$$s_i r_{n-i+1} + s_{i-1} r_{n-i} = s_0 r_{n+1} + s_{-1} r_n = r_n.$$

This is the induction step. The induction hypothesis for $i > 0$ is

$$r_n = s_i r_{n-i+1} + s_{i-1} r_{n-i}.$$

Now, by the definition of s_{i+1}

$$r_{n-i} s_{i+1} + s_i r_{n-i-1} = r_{n-i}(s_{i-1} - s_i q_{n-i+1}) + s_i r_{n-i-1}.$$

By rearranging, this equals

$$s_i(r_{n-i-1} - r_{n-i} q_{n-i+1}) + s_{i-1} r_{n-i},$$

and by the Euclidean algorithm, this equals

$$s_i r_{n-i+1} + s_{i-1} r_{n-i},$$

which is r_n by the induction hypothesis. This completes the induction. Thus, in particular, if $i = n+1$, then

$$g = r_n = s_{n+1} r_0 + s_n r_{-1} = s_{n+1} a + s_n b.$$

□

It may seem that this second form of induction is stronger than the first, but they are equivalent.

Theorem 1.8 **Equivalence of the Forms of Induction**

The first and second forms of the Principle of Mathematical Induction are equivalent.

Proof. The easy part is to show that the second form implies the first form. Assume the validity of the second form. Suppose that we have a set $\mathcal{S} \subseteq \mathbb{N}$ such that $1 \in \mathcal{S}$, and $n+1 \in \mathcal{S}$ whenever $n \in \mathcal{S}$. In other words, we are assuming the hypothesis of the first form. We must show that $\mathcal{S} = \mathbb{N}$, namely the conclusion of the first form. Take $m = 1$ in part 1 of the hypothesis of the second form. Therefore, part 2 of its hypothesis says that if $n \geq 1$ and $\{1, 2, \ldots, n\} \subseteq \mathcal{S}$, then $n+1 \in \mathcal{S}$. Since we are assuming the validity of the second form, we may conclude that $k \in \mathcal{S}$ for all $k \in \mathbb{Z}$ such that $k \geq 1$. In other words, $\mathcal{S} = \mathbb{N}$. We have shown that the validity of the second form implies the validity of the first form.

Conversely, we now assume the validity of the first form. Suppose that parts (a)–(b), namely the hypotheses of the second form, hold. Thus,

(a) $m \in \mathcal{S}$, and

(b) If $m \leq n$ and $\{m, m+1, \ldots, n\} \subseteq \mathcal{S}$, then $n+1 \in \mathcal{S}$.

We must show that $k \in \mathcal{S}$ for all $k \in \mathbb{Z}$ such that $k \geq m$. To do this, we make some identifications. Consider the following schematic diagram. We may think of each element in this schematic as a carrying or a *mapping* of each element listed on the left to a single element on the right, namely a *function* (see Definition A.6 on page 300).

$$\begin{aligned} m &\longmapsto 1 \\ m+1 &\longmapsto 2 \\ &\vdots \\ k &\longmapsto k-m+1. \end{aligned}$$

We may call this mapping f, and we say that f maps k to $k-m+1$ for any integer $k \geq m$, denoted by

$$f(k) = k - m + 1.$$

Also, we write $f(\mathcal{S}) = \mathcal{T}$ to represent the set $\mathcal{T}$ which is *identified* by f with the subset of $\mathcal{S}$ containing all those integers $k \geq m$. We have also symbolically identified the set

$$\mathcal{S}_k = \{m, m+1, \ldots, k\} \subseteq \mathcal{S}$$

with the set

$$\mathcal{T}_{k-m+1} = \{1, 2, \ldots, k-m+1\} \subseteq \mathbb{N}.$$

Now we may translate what parts 1 and 2 of the second form of induction say under this map. If we set $N = n - m + 1$ for any given $n \geq m$, then part 1 says that $1 = f(m) \in \mathcal{T}$ (since $m \in \mathcal{S}$). Part 2 says that if $\{1, 2, \ldots, N\} \subseteq \mathcal{T}$, then

$$f(n+1) = N + 1 \in \mathcal{T}$$

(since $\mathcal{S}_n = \{m, m+1, \ldots, n\} \subseteq \mathcal{S}$ implies that $n+1 \in \mathcal{S}$). In other words, $1 \in \mathcal{T}$, and $N+1 \in \mathcal{T}$ whenever $N \in \mathcal{T}$. Thus, the Principle of Induction (first form) allows us to conclude that $\mathcal{T} = \mathbb{N}$. We have shown that $f(\mathcal{S}) = \mathcal{T}$ is *identified* with $\mathbb{N}$, and we recall that $f(\mathcal{S}) = \mathcal{T}$ is just that set identified with the subset of $\mathcal{S}$ consisting of all integers $k \geq m$, namely f is a bijection between them. In other words, since $\mathcal{T} = \mathbb{N}$, then we have the following schematic:

$$m \in \mathcal{S} \longleftrightarrow 1 \in \mathcal{T} = \mathbb{N}$$

$$m+1 \in \mathcal{S} \longleftrightarrow 2 \in \mathcal{T} = \mathbb{N}$$

$$\vdots$$

$$k \in \mathcal{S} \longleftrightarrow k-m+1 \in \mathcal{T} = \mathbb{N}$$

for all $k \geq m$. Also, since the double arrows represent bijections, then the elements of $\mathcal{S}$ on the left are *identified* with the elements of $\mathcal{T} = \mathbb{N}$ on the right. Hence, via this bijection, $k \in \mathcal{S}$ for all $k \geq m$. We have now demonstrated the logical equivalence of the two forms of the Principle of Mathematical Induction.□

Remark 1.1 *To understand why a seemingly stronger version of induction is no more powerful than the original version, we must keep in mind the basic principle behind induction. Once we have a* beginning element *m, in a set of integers $\mathcal{S}$, and once we show that $n+1 \in \mathcal{S}$ for* any given *$n \in \mathcal{S}$, with $n \geq m$, then all successors of m are in $\mathcal{S}$. It does not matter if we start with $m = 1$ or $m = -1,000$. The fact remains that the Principle of Mathematical Induction, in any of its forms (first, second or, via Theorem 1.9 below, the Well-Ordering Principle) guarantees that all successors are also there.*

Now we demonstrate that not only are the forms of induction equivalent but also they are equivalent to the Well-Ordering Principle.

Theorem 1.9 **Equivalence of Induction and Well Ordering**

The Well-Ordering Principle is Equivalent to the Principle of Mathematical Induction.

Proof. Assume that the Principle of Mathematical Induction holds. Let $\mathcal{S} \neq \varnothing$, and $\mathcal{S} \subseteq \mathbb{N}$. Suppose that $\mathcal{S}$ has no least element. Then $1 \notin \mathcal{S}$, so $2 \notin \mathcal{S}$, and similarly $3 \notin \mathcal{S}$, and so on, which implies that $\mathcal{S} = \varnothing$ by induction, a contradiction.

Conversely, assume the Well-Ordering Principle holds. Also, assume that $1 \in \mathcal{S}$, and that $k \in \mathcal{S}$, whenever $k-1 \in \mathcal{S}$. If $\mathcal{S} \neq \mathbb{N}$, then the Well-Ordering Principle says that there is a least $n \in \mathbb{N} \setminus \mathcal{S}$. Thus, $n-1 \in S$. However, by assumption $n \in \mathcal{S}$, a contradiction. Therefore, $\mathcal{S} = \mathbb{N}$, so the Principle of Mathematical Induction holds. □

Exercises

1.37. Prove that the solution to the rabbit problem on page 8 is F_n pairs of rabbits.

1.38. If p is a prime and $p|ab$, prove that either $p|a$ or $p|b$.

1.39. Let $\mathfrak{g}$ be the golden ratio defined on page 11. Prove that $\mathfrak{g}^2 = \mathfrak{g} + 1$.

1.40. Prove that if $n \in \mathbb{N}$ is composite, then n has a prime divisor p such that $p \leq \sqrt{n}$.

1.41. Prove that all odd primes are either of the form $4n+1$ or $4n-1$ for some $n \in \mathbb{N}$.

1.42. Prove that if $n \in \mathbb{N}$ is a product of primes of the form $4m+1$, then n must also be of that form.

1.43. Let $a = \prod_{i=1}^{r} p_i^{m_i}, b = \prod_{i=1}^{r} p_i^{n_i}$ for integers $m_i, n_i \geq 0$ and distinct primes p_i with $1 \leq i \leq r$. Let $t_i = \min\{m_i, n_i\}$ denote the minimum value of m_i and n_i.

(a) Prove that $\gcd(a, b) = \prod_{i=1}^{r} p_i^{t_i}$.

(b) Prove that $a|b$ if and only if $m_i \leq n_i \quad (1 \leq i \leq r)$.

1.44. If p is prime and $p|a^n$. Prove that $p^n|a^n$, where $a \in \mathbb{Z}$ and $n \in \mathbb{N}$.

1.45. Suppose that there are no primes p such that p divides both $a, b \in \mathbb{Z}$. Prove that $\gcd(a, b) = 1$.

1.46. For each $n \in \mathbb{N}$ the sum of the positive divisors of n is denoted by $\sigma(n)$, called the *sum of divisors function*. Prove that for a prime p and $k \in \mathbb{N}$, $\sigma(p^k) = (p^{k+1} - 1)/(p - 1)$.

1.47. With reference to Exercise 1.46, a number $n \in \mathbb{N}$ is called *almost perfect* if $\sigma(n) = 2n - 1$. Prove that all powers of 2 are almost perfect. (*It is unknown if there are other almost perfect numbers.*)

1.48. A natural number n is called *perfect* if it equals the sum of its proper divisors (see page 2) (namely if $\sigma(n) = 2n$ in the notation of Exercise 1.46). Prove that if $2^n - 1$ is prime, then n is prime and $2^{n-1}(2^n - 1)$ is a perfect number. (See Biography 1.5 on page 16.)

1.49. Calculate $\sigma(n)$ for each of the following n.

(a) 69. (b) 96.

(c) 100. (d) 64.

(e) 2^k for $k \in \mathbb{N}$. (f) 10000.

Biography 1.5 Saint Augustine of Hippo (354–430) *is purported to have said that God created the universe in six days since the perfection of the work is signified by the perfect number* 6*, which is the smallest perfect number. Augustine, who was considered to be the greatest Christian philosopher of antiquity, merged the religion of the new testament with Platonic philosophy. Perfect numbers were known to the ancient Greeks in Euclid's time* (*see Biography 1.1 on page 3*)*, although they knew of only the four smallest ones:* $6, 28, 496$*, and* 8128*. They also attributed mystical properties to these numbers. Also note that the moon orbits the earth every 28 days, another perfect number.*

1.50. Numbers of the form $M_n = 2^n - 1$ for $n \in \mathbb{N}$, are called *Mersenne numbers* — see Biography 1.6. Prove that if M_n is prime, then n is prime. (Compare with Exercise 1.48.)[1.4]

1.51. Let $\mathfrak{g}' = (1 - \sqrt{5})/2$. Prove that the nth Fibonacci number (defined on page 8) has an alternative definition in terms of the golden ratio (defined on page 11), given by

$$F_n = \frac{1}{\sqrt{5}} \left[\mathfrak{g}^n - \mathfrak{g}'^n \right].$$

1.52. Prove that the golden ratio has an alternative representation given by

$$\mathfrak{g} = \sqrt{1 + \sqrt{1 + \sqrt{1 + \cdots}}}.$$

(*Hint: Use Exercise 1.39.*)

Biography 1.6 Marin Mersenne (1588–1648) *was born in Paris on September 8, 1588. He studied at the new Jesuit college at La Fleche* (*1604–1609*) *and at the Sorbonne* (*1609–1611*)*. He joined the mendicant religious order of the Minims in 1611, and on October 28, 1613, he celebrated his first mass. After teaching philosophy and theology at Nevers, he returned to Paris in 1619 to the Minim Convent de l'Annociade near Place Royale where he was elected Correcteur. This became his home base for the rest of his life. He died on September 1, 1648, in Paris.*

1.53. Let $n = pq$ where $p > q$ are odd primes. Prove that there are exactly two ordered pairs of natural numbers (x, y) for which $n = x^2 - y^2$, namely

$$(x, y) \in \{((p + q)/2, (p - q)/2), ((pq + 1)/2, (pq - 1)/2)\}.$$

[1.4] See http://www.mersenne.org/ for the largest Mersennne prime, which is updated on a regular basis.

1.3 An Introduction to Congruences

We now turn to a concept called *congruences*, invented by Gauss (see Biography 1.7 on page 18). The stage is set by the discussion of divisibility given in Section 1.1.

Gauss sought a convenient tool for abbreviating the family of expressions $a = b + nk$, called an *arithmetic progression* with modulus n, wherein k varies over all natural numbers, $n \in \mathbb{N}$ is fixed, as are $a, b \in \mathbb{Z}$. He did this as follows.

Definition 1.7 **Congruences**

If $n \in \mathbb{N}$, then we say that a is congruent *to b* modulo *n if $n|(a-b)$, denoted by*

$$a \equiv b \,(\text{mod } n).$$

On the other hand, if $n \nmid (a-b)$, then we write

$$a \not\equiv b \,(\text{mod } n)$$

and say that a and b are incongruent *modulo n, or that a is* not congruent *to b modulo n. The integer n is the* modulus *of the congruence. The set of all integers that are congruent to a given integer m modulo n, denoted by $\overline{m}$, is called the* congruence class *or* residue class *of m modulo n. (Note that since the notation $\overline{m}$ does not specify the modulus n, then the bar notation will always be taken in context.)*

Example 1.2 (a) Since $3|(82-1)$, $82 \equiv 1 \,(\text{mod } 3)$.

(b) Since $11|(16-(-6))$, $16 \equiv -6 \,(\text{mod } 11)$.

(c) Since $7 \nmid (10-2)$, $10 \not\equiv 2 \,(\text{mod } 7)$.

(d) For any $a, b \in \mathbb{Z}$, $a \equiv b \,(\text{mod } 1)$, since $1|(a-b)$.

Now we develop results for *modular arithmetic*, namely an arithmetic for congruences. The first result shows that congruences are a special kind of relation, which behaves much like equality.

Proposition 1.1 *Let $n \in \mathbb{N}$. Then each of the following holds.*

(a) For each $a \in \mathbb{Z}$, $a \equiv a \,(\text{mod } n)$, called the *reflexive property.*

(b) For any $a, b \in \mathbb{Z}$, if $a \equiv b \,(\text{mod } n)$, then $b \equiv a \,(\text{mod } n)$, called the *symmetric property.*

(c) For any $a, b, c \in \mathbb{Z}$, if $a \equiv b \,(\text{mod } n)$, and $b \equiv c \,(\text{mod } n)$, then $a \equiv c \,(\text{mod } n)$, called the *transitive property.*

Proof. (a) If $n \in \mathbb{N}$, then $n|0 = a - a$, so $a \equiv a \,(\text{mod } n)$, which establishes the reflexive property.

(b) Let $n \in \mathbb{N}$, $a, b, c \in \mathbb{Z}$, $a \equiv b \,(\mathrm{mod}\ n)$, so $a - b = kn$ for some $k \in \mathbb{Z}$. By rewriting, $b - a = (-k)n$, implying $b \equiv a \,(\mathrm{mod}\ n)$, which establishes the symmetric property.

To prove part (c), we use Definition 1.7. Since $a \equiv b \,(\mathrm{mod}\ n)$, and $b \equiv c \,(\mathrm{mod}\ n)$, then $n \mid (a - b)$ and $n \mid (b - c)$. Therefore,

$$n \mid (a - b) + (b - c) = (a - c),$$

which is to say

$$a \equiv c \,(\mathrm{mod}\ n).$$

□

Remark 1.2 *Proposition 1.1 shows that congruence modulo n is an* equivalence relation*, which is defined to be a set R of ordered pairs on* $\mathbb{S} \times \mathbb{S}$ *for a given set* $\mathbb{S}$ *satisfying the reflexive, symmetric, and transitive properties. Moreover, the set* $\{x : (x, a) \in R\}$ *is called the* equivalence class *containing a. In the case of congruences, this latter notion coincides with that of a* congruence class*.*

The next result tells us that we can perform the basic operations of addition, subtraction, and multiplication with congruences.

Proposition 1.2 *Let* $n \in \mathbb{N}$ *and* $a, b, c, d \in \mathbb{Z}$*. If* $a \equiv b \,(\mathrm{mod}\ n)$ *and* $c \equiv d \,(\mathrm{mod}\ n)$*, then* $a + c \equiv b + d \,(\mathrm{mod}\ n)$*,* $a - c \equiv b - d \,(\mathrm{mod}\ n)$*, and* $ac \equiv bd \,(\mathrm{mod}\ n)$*.*

Proof. Since there exist integers $k, \ell \in \mathbb{Z}$ such that $a = b + kn$ and $c = d + \ell n$, then (grouping the $\pm$ into a single proof)

$$a \pm c = b + kn \pm (d + \ell n) = b \pm d + (k \pm \ell)n,$$

so

$$a \pm c \equiv b \pm d \,(\mathrm{mod}\ n).$$

Similarly,

$$ac \equiv (b + kn)(d + \ell n) \equiv bd \,(\mathrm{mod}\ n).$$

□

The next result tells us how to divide using congruences.

Biography 1.7 *One of the greatest mathematicians who ever lived was* Carl Friedrich Gauss (1777–1855)*. At the age of eight, he astonished his teacher, Büttner, by rapidly adding the integers from* 1 *to* 100 *via the observation that the fifty pairs* $(j + 1, 100 - j)$ *for* $j = 0, 1, \ldots, 49$ *each sum to* 101 *for a total of* 5050*. At the age of eleven, Gauss entered a preparatory school for university called a* Gymnasium *in Germany. By the age of fifteen, Gauss entered Brunswick Collegium Carolinum funded by the Duke of Brunswick. In 1795, Gauss was accepted to Göttingen University and by the age of twenty achieved his doctorate. The reader is referred to* [61] *and* [62] *for a discussion of his multitudinous achievements. Gauss was married twice. He married his first wife, Johanna Ostoff, on October 9, 1805. She died in 1809 after giving birth to their second son. His second wife was Johanna's best friend Minna, whom he married in 1810. She bore him three children. Gauss remained a professor at Göttingen until the early morning of February 23, 1855, when he died in his sleep.*

Proposition 1.3 *If* $\gcd(c, n) = g$, *then*

$$ac \equiv bc \,(\mathrm{mod}\ n)$$

if and only if

$$a \equiv b \,(\mathrm{mod}\ n/g).$$

Proof. If $ac - bc = kn$ for some $k \in \mathbb{Z}$, then $(a - b)c/g = kn/g$. By Exercise 1.28 on page 5, $\gcd(c/g, n/g) = 1$. Therefore, (n/g) divides $(a - b)$, namely

$$a \equiv b \,(\mathrm{mod}\ n/g).$$

Conversely, if $a \equiv b \,(\mathrm{mod}\ n/g)$, then there exists an integer $d \in \mathbb{Z}$ such that $a = b + dn/g$, so $ac = bc + d(c/g)n$. Hence, $ac \equiv bc \,(\mathrm{mod}\ n)$. □

Notice that Proposition 1.3 tells us that we cannot simply divide through by c if $\gcd(c, n) = g > 1$, since the modulus must be taken into consideration. Only when $g = 1$ may we divide through and leave the modulus unchanged.

Some additional properties of congruences are given in the next result.

Proposition 1.4 *Let* $a, b, c \in \mathbb{Z}$, $m, n \in \mathbb{N}$, *and* $a \equiv b \,(\mathrm{mod}\ n)$. *Then each of the following holds.*

(a) $am \equiv bm \,(\mathrm{mod}\ mn)$.

(b) $a^m \equiv b^m \,(\mathrm{mod}\ n)$.

(c) If m divides n, then $a \equiv b \,(\mathrm{mod}\ m)$.

Proof. (a) Given that $a \equiv b \,(\mathrm{mod}\ n)$, $a - b = kn$ for some integer k. Multiplying by m, we get $(a - b)m = knm$, so $am - bm = (km)n$, namely $am \equiv bm \,(\mathrm{mod}\ n)$.

(b) Since $n|(a - b)$, then

$$n|(a - b)(a^{m-1} + a^{m-2}b + \cdots + b^m) = a^m - b^m.$$

In other words,

$$a^m \equiv b^m \,(\mathrm{mod}\ n).$$

(c) Since $a = b + kn$ for some $k \in \mathbb{Z}$ and $n = \ell m$ for some $\ell \in \mathbb{N}$, then $a = b + k\ell m$, so $a - b = (k\ell)m$, whence $a \equiv b \,(\mathrm{mod}\ m)$. □

Propositions 1.1–1.4 can be employed to establish a modular arithmetic. First we need another couple of notions.

Gauss used the congruence notation to replace the assertion: *a and b are in the same arithmetic progression with difference a multiple of n* by the statement: *a is congruent to b modulo n*, denoted by $a \equiv b \,(\mathrm{mod}\ n)$, which is the content of Definition 1.7, namely $a \equiv b \,(\mathrm{mod}\ n)$, if and only if $a = b + nk$ for some $k \in \mathbb{Z}$. Thus, $a \equiv b \,(\mathrm{mod}\ n)$ if and only if $\overline{a} = \overline{b}$ with modulus n. Therefore, it makes sense to have a canonical representative.

Definition 1.8 **Least Residues**

If $n \in \mathbb{N}$, $a \in \mathbb{Z}$, and $a = nq + r$ where $0 \leq r < n$ is the remainder when a is divided by n, given by Theorem 1.1, the Division Algorithm, then r is called the least (nonnegative) residue of a modulo n, *and the set $\{0, 1, 2, \ldots, n-1\}$ is called the set of* least nonnegative residues modulo n.

We now show that for all $n \in \mathbb{N}$, congruence modulo n partitions the integers $\mathbb{Z}$ into disjoint subsets (see Definition A.4 on page 299). We need to show that every $m \in \mathbb{Z}$ is in *exactly* one residue class modulo n. (Note that Definition 1.8 justifies the use of the term *residue* class, given in Definition 1.7.) Since $m \in \overline{m}$, then m is in *some* congruence class. We must prove that it is in *no more than one* such class.

If $m \in \overline{m}_1$ and $m \in \overline{m}_2$, both $m \equiv m_1 \pmod{n}$ and $m \equiv m_2 \pmod{n}$. Thus, $m_1 \equiv m_2 \pmod{n}$ by Proposition 1.1 (c), so $\overline{m}_1 = \overline{m}_2$, and we are done.

As well as the above being true, it is also true that for any $n \in \mathbb{N}$, and $0 \leq i \leq j \leq n-1$, $i \equiv j \pmod{n}$ if and only if $i = j$. To see this, we observe that $j - i = mn$ for some $m \in \mathbb{Z}$ by definition, so $n \mid (j - i)$. If $j - i > 0$, then $n \leq (j - i)$. Since $j < n$ and $-i \leq 0$, it follows that $j - i = j + (-i) < n$, contradicting that $n \leq (j - i)$. Hence, $i = j$. We have shown that there are exactly n congruence classes for each $n \in \mathbb{N}$.

Example 1.3 There are four congruence classes modulo 4, namely

$$\overline{0} = \{\ldots, -4, 0, 4, \ldots\},$$
$$\overline{1} = \{\ldots, -3, 1, 5, \ldots\},$$
$$\overline{2} = \{\ldots, -2, 2, 6, \ldots\},$$

and

$$\overline{3} = \{\ldots, -1, 3, 7, \ldots\},$$

since each element of $\mathbb{Z}$ is in exactly one of these sets.

In order to motivate the next notion we let $r \in \mathbb{Z}$, $n \in \mathbb{N}$, and consider the set $\{r, r+1, \ldots, r+n-1\}$. If $r + i \equiv r + j \pmod{n}$ for $0 \leq i \leq j \leq n-1$, then $i \equiv j \pmod{n}$, so by the same argument as above $i = j$. This shows that the $\overline{r+j}$ for $0 \leq j \leq n-1$ are n distinct congruences classes. Moreover, if $m \in \mathbb{Z}$, then by the argument given after Definition 1.8, m must be in exactly one of the n congruence classes. In other words, $m \equiv r + j \pmod{n}$ for some nonnegative integer $j < n$. This motivates the following.

Definition 1.9 **Complete Residue System**

Suppose that $n \in \mathbb{N}$ is a modulus. A set of integers

$$\mathcal{T} = \{r_1, r_2, \ldots, r_n\}$$

such that every integer is congruent to exactly one element of $\mathcal{T}$ *modulo* n *is called a* complete residue system modulo n. *In other words, for any* $a \in \mathbb{Z}$, *there exists a unique* $r_i \in \mathcal{T}$ *such that* $a \equiv r_i \pmod{n}$. *The set*

$$\{0, 1, \ldots, n-1\}$$

is a complete residue system, called the least residue system modulo n.

For example, $\mathcal{T} = \{-4, -3, -2, -1\}$ is a complete residue system modulo 4. Also, $\mathcal{T} = \{0, 1, 2, 3\}$ is the least residue system modulo 4. In fact, as proved in the discussion preceding Definition 1.9, any set of n consecutive integers forms a complete residue system modulo n. By choosing $r = 0$ in that discussion, we get the least residues.

Example 1.4 The least residue system modulo 4 is $\mathcal{T} = \{0, 1, 2, 3\}$. Suppose that we want to calculate the addition of $\overline{3}$ and $\overline{2}$ in $\{\overline{0}, \overline{1}, \overline{2}, \overline{3}\}$. First, we must define what we mean by this addition. Define

$$\overline{a} \oplus \overline{b} = \overline{a+b}$$

where $+$ is the ordinary addition of integers. Since $\overline{3}$ represents all integers of the form $3 + 4k$, $k \in \mathbb{Z}$ and $\overline{2}$ represents all integers of the form $2 + 4\ell$, $\ell \in \mathbb{Z}$,

$$3 + 4k + 2 + 4\ell = 5 + 4(k + \ell) = 1 + 4(1 + k + \ell).$$

Hence, $\overline{3} \oplus \overline{2} = \overline{1} = \overline{3+2}$.

Similarly, we may define

$$\overline{a} \otimes \overline{b} = \overline{a \cdot b},$$

where $\cdot$ is the ordinary multiplication of integers. The reader may verify that $\overline{2} \otimes \overline{3} = \overline{2} = \overline{2 \cdot 3}$. Notice as well that since

$$\overline{a-b} = \overline{a + (-b)} = \overline{a} \oplus \overline{-b},$$

then $\overline{2} \oplus \overline{-3} = \overline{3} = \overline{2-3}$, for instance.

Example 1.4 illustrates the basic operations of addition and multiplication in $\{\overline{0}, \overline{1}, \ldots, \overline{n-1}\}$ for any $n \in \mathbb{N}$, namely

$$\overline{a} \oplus \overline{b} = \overline{a+b} \text{ and } \overline{a} \otimes \overline{b} = \overline{a \cdot b},$$

where $\oplus$ and $\otimes$ are well defined since $+$ and $\cdot$ are well defined. Since it would be cumbersome to use the notations of $\oplus$, and $\otimes$ in general, we maintain the usage of $+$ for $\oplus$ and $\cdot$ for $\otimes$,where we will understand that the the result of the given operation is in the appropriate residue class. The following result formalizes this for us in general. The reader is encouraged to review the fundamental laws for arithmetic beginning on page 302, so that we will see that these seemingly trivial laws have a generalization to the following important scenario.

Theorem 1.10 **Modular Arithmetic**

Let $n \in \mathbb{N}$ and suppose that for any $x \in \mathbb{Z}$, $\overline{x}$ denotes the congruence class of x modulo n. Then for any $a, b, c \in \mathbb{Z}$ the following hold.

(a) $\overline{a} \pm \overline{b} = \overline{a \pm b}$. (Modular additive closure)

(b) $\overline{a}\overline{b} = \overline{ab}$. (Modular multiplicative closure)

(c) $\overline{a} + \overline{b} = \overline{b} + \overline{a}$. (Commutativity of modular addition)

(d) $(\overline{a} + \overline{b}) + \overline{c} = \overline{a} + (\overline{b} + \overline{c})$. (Associativity of modular addition)

(e) $\overline{0} + \overline{a} = \overline{a} + \overline{0} = \overline{a}$. (Additive modular identity)

(f) $\overline{a} + \overline{-a} = \overline{-a} + \overline{a} = -\overline{a} + \overline{a} = \overline{0}$. (Additive modular inverse)

(g) $\overline{a}\overline{b} = \overline{b}\overline{a}$. (Commutativity of modular multiplication)

(h) $(\overline{a}\overline{b})c = \overline{a}(\overline{b}\overline{c})$. (Associativity of modular multiplication)

(i) $\overline{1}\overline{a} = \overline{a}\overline{1} = \overline{a}$. (Multiplicative modular identity)

(j) $\overline{a}(\overline{b} + \overline{c}) = \overline{a}\overline{b} + \overline{a} \cdot \overline{c}$. (Modular Distributivity)

Proof. Part (a) is a consequence of Proposition 1.2, and part (b) is a consequence of Proposition 1.4 part (a). Part (c) can be established using part (a) which we just proved since

$$\overline{a} + \overline{b} = \overline{a + b} = \overline{b + a} = \overline{b} + \overline{a}.$$

In other words, the commutativity property is inherited from the integers $\mathbb{Z}$. Part (d) also follows from part (a) since

$$(\overline{a} + \overline{b}) + \overline{c} = \overline{a + b} + \overline{c} = \overline{a + b + c} = \overline{a} + (\overline{b + c}) = \overline{a} + (\overline{b} + \overline{c}).$$

Part (e) is a consequence of parts (c) and (a) since

$$\overline{0} + \overline{a} = \overline{a} + \overline{0} = \overline{a + 0} = \overline{a},$$

where the first equality holds by part (c) and the second equality holds by part (a). The first equality of part (f) follows from parts (a) and (c) in exactly the same fashion, whereas the second part follows from part (b). Part (g) follows from the ordinary commutativity of multiplication of integers and part (b), since

$$\overline{a}\overline{b} = \overline{ab} = \overline{ba} = \overline{b}\overline{a}.$$

Part (h) may now be deduced from part (b) and ordinary associativity of the integers since

$$(\overline{a}\overline{b})\overline{c} = \overline{(ab)}\overline{c} = \overline{(ab)c} = \overline{a(bc)} = \overline{a}\overline{(bc)} = \overline{a}(\overline{b}\overline{c}).$$

Part (i) is a simple consequence of parts (b), (g), and the multiplicative identity of the integers since

$$\overline{1}\overline{a} = \overline{1 \cdot a} = \overline{a \cdot 1} = \overline{a}.$$

Lastly, part (j) is a consequence of parts (a), (b), and the ordinary distributivity of multiplication over addition given that

$$\overline{a}(\overline{b} + \overline{c}) = \overline{a}(\overline{b + c}) = \overline{a(b + c)} = \overline{ab + ac} = \overline{ab} + \overline{ac} = \overline{a}\overline{b} + \overline{a} \cdot \overline{c}.$$

□

Parts (a)–(b) of Theorem 1.10 tell us that the bar operation is well defined under addition and multiplication. The remaining properties of this theorem tell us that there is an underlying structure. Any set that satisfies the (named) properties (a)–(j) of Theorem 1.10 is called a *commutative ring with identity.* Now we look at a specific such ring that has important consequences.

Definition 1.10 **The Ring $\mathbb{Z}/n\mathbb{Z}$**

For $n \in \mathbb{N}$, the set

$$\mathbb{Z}/n\mathbb{Z} = \{\overline{0}, \overline{1}, \overline{2}, \ldots, \overline{n-1}\}$$

is called the Ring of Integers Modulo *n, where $\overline{m}$ denotes the congruence class of m modulo n. (Occasionally, when the context is clear and no confusion can arise when talking about elements of $\mathbb{Z}/n\mathbb{Z}$, we will eliminate the* overline bars.)

Notice that since $\{0, \ldots, n-1\}$ is the least residue system modulo n, then every $z \in \mathbb{Z}$ has a *representative* in the ring of integers modulo n, namely an element $j \in \{0, \ldots, n-1\}$ such that $z \equiv j \pmod{n}$. The ring $\mathbb{Z}/n\mathbb{Z}$ will play an important role in the cryptographic applications that we study later in the text. There are other structures hidden within the properties listed in Theorem 1.10 that are worth mentioning, since we will also encounter them in our cryptographic travels. Any set satisfying the properties (a), (d)–(f) is called an *additive group*, and if additionally it satisfies (c), then it is called an *additive abelian group.* A fortiori, $\mathbb{Z}/n\mathbb{Z}$ is an additive abelian group as is $\mathbb{Z}$. Any set satisfying (a)–(f), (h) and (j) is called a *ring*, and if in addition it satisfies (g), then it is a *commutative ring.* As we have seen, any set satisfying all of the conditions (a)–(j) is a commutative ring with identity. In general, we would use symbols other than the bar operation and possibly binary symbols other than the multiplication and addition symbols, but the listed properties in Theorem 1.10 would remain essentially the same for the algebraic structures defined above.

There is a multiplicative property of $\mathbb{Z}$ that $\mathbb{Z}/n\mathbb{Z}$ does not have. On page 303, the Cancellation Law for $\mathbb{Z}$ is listed. This is not the case for $\mathbb{Z}/n\mathbb{Z}$ in general. For instance, $2 \cdot 3 \equiv 2 \cdot 8 \pmod{10}$, but $3 \not\equiv 8 \pmod{10}$. In other words, $2 \cdot 3 = 2 \cdot 8$ in $\mathbb{Z}/10\mathbb{Z}$, but $3 \neq 8$ in $\mathbb{Z}/10\mathbb{Z}$. We may ask for conditions on n

under which a modular law for cancellation would hold. In other words, for which $n \in \mathbb{N}$ does it hold that:

$$\text{for any } a, b, c \in \mathbb{Z}/n\mathbb{Z} \text{ with } a \neq 0,\ ab = ac \text{ if and only if } b = c? \tag{1.1}$$

By Proposition 1.3, (1.1) cannot hold if $\gcd(a, n) > 1$. When $\gcd(a, n) = 1$, there is a solution $x \in \mathbb{Z}$ to $ax \equiv 1 \pmod{n}$ (see Exercise 1.64 on page 32). This motivates the following.

Definition 1.11 **Modular Multiplicative Inverses**

Suppose that $a \in \mathbb{Z}$, and $n \in \mathbb{N}$. A multiplicative inverse of the integer a modulo n *is an integer x such that $ax \equiv 1 \pmod{n}$. If x is the least positive such inverse, then we call it* the least multiplicative inverse of the integer a modulo n, *denoted by $x = a^{-1}$.*

In the illustration of the Egg Basket problem on page 30, the linear congruences all have coefficient 1 for x. However, by using modular multiplicative inverses, we can solve more general systems of linear congruences.

Example 1.5 Suppose that we wish to solve the system of linear congruences

$$2x \equiv 1 \pmod{3},\ 3x \equiv 1 \pmod{5}, \text{ and } 3x \equiv 2 \pmod{7}.$$

Since $2^{-1} \equiv 2 \pmod{3}$, $3^{-1} \equiv 2 \pmod{5}$, and $3^{-1} \equiv 5 \pmod{7}$, then the system of congruences becomes

$$x \equiv 2 \pmod{3}, x \equiv 2 \pmod{5}, \text{ and } x \equiv 3 \pmod{7},$$

for which $x = 17$ is clearly seen to be the least nonnegative solution modulo 105.

Example 1.6 Consider $n = 11$ and $a = -3$, and suppose that we want to find the least multiplicative inverse of a modulo n. Since $-3 \cdot 7 \equiv 1 \pmod{11}$ and no smaller natural number than 7 satisfies this congruence, then $a^{-1} = 7$ modulo 11.

Example 1.7 If $n = 22$ and $a = 6$, then no multiplicative inverse of a modulo n exists since $\gcd(a, n) = 2$. Asking for a multiplicative inverse of such a value a modulo n is similar to asking for division by 0 with ordinary division of integers. In other words, this is undefined.

Since any composite n has a prime $p < n$ dividing it, then this means that (1.1) holds for all $a \in \mathbb{Z}/n\mathbb{Z}$, $a \neq 0$, if and only if n is prime. Another way of stating this is as follows. Every nonzero $z \in \mathbb{Z}/n\mathbb{Z}$ has a multiplicative inverse if and only if n is prime.

If the existence of multiplicative inverses is satisfied for any given element along with (b), (h)–(i) of Theorem 1.10 for a given set, then that set is called a

multiplicative group. In addition, if the set satisfies (g) of Theorem 1.10, then it is called an *abelian multiplicative group*. Hence, $(\mathbb{Z}/n\mathbb{Z})^*$ is a multiplicative abelian group if and only if n is prime. Notice that $\mathbb{Z}$ is *not* a multiplicative group since any nonzero $a \in \mathbb{Z}$ with $a \neq \pm 1$ has no multiplicative inverse.

There is one property that is held by $\mathbb{Z}$ that is of particular importance to the ring $\mathbb{Z}/n\mathbb{Z}$. There are mathematical structures $\mathcal{S}$ that have what are called *zero divisors*. These are elements $s, t \in \mathcal{S}$ such that both s and t are nonzero, yet $st = 0$. For instance, in the ring $\mathbb{Z}/6\mathbb{Z}$, $2 \cdot 3 = 0$, so this ring has zero divisors. The integers $\mathbb{Z}$ have no zero divisors. What is the situation for $\mathbb{Z}/n\mathbb{Z}$ with respect to zero divisors? If n is composite, then there are natural numbers $n > n_1 > 1$ and $n > n_2 > 1$ such that $n = n_1 n_2$. Hence, $n_1 n_2 = 0$ in $\mathbb{Z}/n\mathbb{Z}$. Therefore, $\mathbb{Z}/n\mathbb{Z}$ has *no zero divisors* if and only if n is prime. Any set that satisfies all the conditions (a)–(j) of Theorem 1.10 together with having no zero divisors and having multiplicative inverses for all of its nonzero elements is called a *field*. Hence, we have established the following.

Theorem 1.11 **The Field $\mathbb{Z}/p\mathbb{Z}$**

If $n \in \mathbb{N}$, then $\mathbb{Z}/n\mathbb{Z}$ is a field if and only if n is prime.

In Theorem A.7 on page 313, we employed the notation F^* to denote the multiplicative group of nonzero elements of a given field F. In particular, when we have a finite field $\mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p$ of p elements for a given prime p, then

$$(\mathbb{Z}/p\mathbb{Z})^* \text{ denotes the multiplicative group of nonzero elements of } \mathbb{F}_p.$$

This is tantamount to saying that $(\mathbb{Z}/p\mathbb{Z})^*$ is the group of units in $\mathbb{F}_p$, and $(\mathbb{Z}/p\mathbb{Z})^*$ is cyclic by Theorem A.7. Thus, this notation and notion may be generalized as follows. Let $n \in \mathbb{N}$ and let the group of units of $\mathbb{Z}/n\mathbb{Z}$ be denoted by $(\mathbb{Z}/n\mathbb{Z})^*$. Then

$$(\mathbb{Z}/n\mathbb{Z})^* = \{\overline{a} \in \mathbb{Z}/n\mathbb{Z} : 0 \leq a < n \text{ and } \gcd(a, n) = 1\}. \tag{1.2}$$

The structure of $(\mathbb{Z}/n\mathbb{Z})^*$ is going to be of vital importance as we move through the text. Moreover, we will be interested only in *finite* groups, rings and fields, except for the obvious infinite cases such as $\mathbb{Z}$ and $\mathbb{Q}$.

Now we go on to look at some of the consequences of this notion of modular division, which is implicit in the above. Definition 1.11 gives us the means to do modular division since multiplication by a^{-1} is equivalent to division in $\mathbb{Z}/n\mathbb{Z}$.

A classic example in the use of multiplicative inverses is the following.

◆ The Coconut Problem

Three sailors and a monkey are shipwrecked on an island. The sailors pick n coconuts as a food supply and place them in a pile. During the night, one of the sailors wakes up and goes to the pile to get his *fair share*. He divides the pile into three, and there is a coconut left over, which he gives to the monkey.

He then hides his third and goes back to sleep. Each of the other two sailors does the exact same thing, by dividing the remaining pile into three, giving the leftover coconut to the monkey and hiding his third. In the morning, the sailors divide the remaining pile into three and give the monkey its fourth coconut. What is the *minimum* number of coconuts that could have been in the original pile?

We begin by observing that the first sailor began with a pile $n \equiv 1 \pmod 3$ coconuts. The second sailor began with a pile of

$$m_1 = \frac{2(n-1)}{3} \equiv 1 \pmod 3$$

coconuts, and the third sailor began with a pile of

$$m_2 = \frac{2(m_1-1)}{3} \equiv 1 \pmod 3$$

coconuts, after which the three of them divided up the remaining pile of

$$m_3 = \frac{2(m_2-1)}{3} \equiv 1 \pmod 3$$

coconuts. We calculate m_3 and get

$$m_3 = \frac{8}{27}n - \frac{38}{27} \equiv 1 \pmod 3.$$

We now solve for n by multiplying through both sides and the modulus by 27, then simplifying to get $8n \equiv 65 \pmod{81}$. (Note that each of n, m_1, m_2, and m_3 must be natural numbers.) Since the multiplicative inverse of 8 modulo 81 is 71, namely $8^{-1} \equiv 71 \pmod{81}$, then $n \equiv 8^{-1} \cdot 65 \equiv 71 \cdot 65 \equiv 79 \pmod{81}$, and the smallest solution is 79.

The reader may now solve Exercise 1.101 on page 43 for another version of the coconut problem.

Theorem 1.12 **Chinese Remainder Theorem**

Let $n_i \in \mathbb{N}$ for natural numbers $i \leq k \in \mathbb{N}$ be pairwise relatively prime, set

$$n = \prod_{j=1}^{k} n_j$$

and let $r_i \in \mathbb{Z}$ for $i \leq k$. Then the system of k simultaneous linear congruences given by

$$\begin{aligned} x &\equiv r_1 \pmod{n_1}, \\ x &\equiv r_2 \pmod{n_2}, \\ &\vdots \\ x &\equiv r_k \pmod{n_k}, \end{aligned}$$

has a unique solution modulo n.

Proof. If $N_j = n/n_j \quad (1 \le j \le k)$, then $\gcd(N_j, n_j) = 1$. Also, by Definition 1.11 on page 24, there is a multiplicative inverse M_j of N_j modulo n_j. Therefore,

$$M_j N_j \equiv 1 \,(\text{mod } n_j).$$

Hence for any $m \le k$,

$$x \equiv \sum_{j=1}^{k} r_j M_j N_j \equiv r_m \,(\text{mod } n_m),$$

which means that x is a solution of the system of linear congruences modulo n. Furthermore, if x_1 and x_2 are solutions of this system, then for each $j \le k$, $x_1 \equiv x_2 \equiv r_j \,(\text{mod } n_j)$, so $n_j | (x_1 - x_2)$ and since $\gcd(n_i, n_j) = 1$, $n | (x_1 - x_2)$. In other words, $x_1 \equiv x_2 \,(\text{mod } n)$. Therefore, the simultaneous solution is unique modulo n. □

Example 1.8 In the example given by Sun Tsŭ, Example 1.5, we set $n = n_1 n_2 n_3 = 105$ with $n_1 = 3$, $n_2 = 5$, and $n_3 = 7$. Also, let $r_1 = 2$, $r_2 = 2$, and $r_3 = 3$. Then the least multiplicative inverse of $N_1 = n/n_1 = 35$ modulo $n_1 = 3$ is $M_1 = 2$. The least multiplicative inverse of $N_2 = n/n_2 = 21$ modulo $n_2 = 5$ is $M_2 = 1$, and the least multiplicative inverse of $N_3 = n/n_3 = 15$ modulo $n_3 = 7$ is $M_3 = 1$. Hence,

$$x = \sum_{j=1}^{3} r_j M_j N_j = 2 \cdot 2 \cdot 35 + 2 \cdot 1 \cdot 21 + 3 \cdot 1 \cdot 15 = 227,$$

as calculated by Sun Tsŭ. By reducing $x = 227$ modulo $n = 105$, we get $x_0 = 17$, as in Example 1.5, the unique solution modulo n.

One may wonder about the situation where the moduli are *not* relatively prime. In 717 A.D. a priest named Yih-hing generalized Theorem 1.12 in his book *t'ai-yen-lei-schu* as follows. The reader should solve Exercises 1.22 on page 5 and 1.75 on page 33, which are used in the following proof.

Theorem 1.13 **Generalized Chinese Remainder Theorem**

Let $n_j \in \mathbb{N}$, set $\ell = \text{lcm}(n_1, n_2, \ldots, n_k)$, and let $r_j \in \mathbb{Z}$ be any integers for $j = 1, 2, \ldots, k$. Then the system of k simultaneous linear congruences given by

$$x \equiv r_1 \,(\text{mod } n_1),$$

$$x \equiv r_2 \,(\text{mod } n_2),$$

$$\vdots$$

$$x \equiv r_k \,(\text{mod } n_k),$$

has a solution if and only if

$$\gcd(n_i, n_j) \mid (r_i - r_j) \text{ for each pair of natural number } i, j \leq k.$$

Moreover, if a solution exists, then it is unique modulo ℓ. Additionally, if there exist integer divisors $m_j \geq 1$ of n_j with $\ell = m_1 \cdot m_2 \cdots m_k$ such that the m_j are pairwise relatively prime, and there exist integers

$$s_j \equiv 0 \pmod{\ell/m_j} \text{ and } s_j \equiv 1 \pmod{m_j} \text{ for } 1 \leq j \leq k,$$

then

$$x = \sum_{j=1}^{k} s_j r_j$$

is a solution of the above congruence system.

Proof. In view of Exercise 1.75, and induction, we need only prove the result for $k = 2$. If $x \equiv r_j \pmod{n_j}$, for $j = 1, 2$, then $x = r_j + u_j n_j \quad (j = 1, 2)$. Therefore,

$$r_1 - r_2 = u_2 n_2 - u_1 n_1.$$

Thus, if $g = \gcd(n_1, n_2)$, then

$$r_1 - r_2 = g(u_2 n_2/g - u_1 n_1/g).$$

We have shown that if a solution exists, then $g \mid (r_1 - r_2)$. Conversely if $g \mid (r_1 - r_2)$, then there is an integer z such that $r_1 = r_2 + gz$. Also, by Exercise 1.22, there are $a, b \in \mathbb{Z}$ such that $g = an_1 + bn_2$. Thus,

$$r_1 = r_2 + z(an_1 + bn_2) = r_2 + zan_1 + zbn_2.$$

If we set

$$x = r_1 - zan_1 = r_2 + zbn_2,$$

then

$$r_j \equiv x \pmod{n_j} \text{ for } j = 1, 2.$$

This establishes the necessary and sufficient condition for existence. We now establish uniqueness.

Suppose that

$$x \equiv r_j \pmod{n_j} \text{ and } y \equiv r_j \pmod{n_j} \text{ for } 1 \leq j \leq k.$$

Then $x - y \equiv 0 \pmod{n_j}$ for each such j. This means that $\ell \mid (x - y)$. Hence, any solution x is unique modulo ℓ.

The last statement of the theorem is clear since if such m_j and s_j exist, then

$$x = \sum_{j=1}^{k} s_j r_j \equiv r_j \pmod{m_j} \text{ for } 1 \leq j \leq k$$

has a unique solution modulo ℓ by the Chinese Remainder Theorem 1.12, and the proof is secured. □

Yin-hing designed Theorem 1.13 to solve the following problem.

◆ The Units of Work Problem

Determine the number of completed units of work when the same number x of units to be performed by each of four sets of $2, 3, 6$, and 12 workers performing their duties for certain numbers of whole days such that there remain $1, 2, 5$, and 5 units of work not completed by the respective sets. We assume further that no set of workers is lazy, namely each completes a nonzero number of units of work.

Here we are looking to solve

$$x \equiv 1 \,(\mathrm{mod}\ 2),\ x \equiv 2 \,(\mathrm{mod}\ 3),\ x \equiv 5 \,(\mathrm{mod}\ 6), \text{ and } x \equiv 5 \,(\mathrm{mod}\ 12).$$

Since $\ell = \mathrm{lcm}(2, 3, 6, 12) = 12$, then we let

$$m_1 = m_2 = 1,\ m_3 = 3, \text{ and } m_4 = 4.$$

Thus, $s_1 = s_2 = 0$ since $m_1 = m_2 = 1$. Also, $s_3 = 4$ since $s_3 \equiv 0 \,(\mathrm{mod}\ 4)$ and $s_3 \equiv 1 \,(\mathrm{mod}\ 3)$; and $s_4 = 9$, since $s_4 \equiv 0 \,(\mathrm{mod}\ 3)$ and $s_4 \equiv 1 \,(\mathrm{mod}\ 4)$. Since $(r_1, r_2, r_3, r_4) = (1, 2, 5, 5)$, then $x = \sum_{j=1}^{4} r_j s_j = 5 \cdot 4 + 5 \cdot 9 = 65 \equiv 17 \,(\mathrm{mod}\ 12)$.

Note that we cannot choose $x = 5$ since this would mean that no units of work had been completed by the last two sets of workers. For $x = 17$, the completed units of work must be $8 \cdot 2 = 16$ for the first set since they do not complete one unit, $5 \cdot 3 = 15$ for the second set since they do not complete two units, $2 \cdot 6 = 12$ for the third set since they do not complete five units, and $1 \cdot 12 = 12$ for the fourth set for the same reason. Hence, the total completed units of work is 55, and Yin-hing's problem is solved.

Biography 1.8 Brahmagupta was considered to be the greatest of the Hindu mathematicians. In 628 he wrote his masterpiece on astronomy *Brahma-sphuta-siddhanta* or *The revised system of Brahma*, which had two chapters devoted to mathematics. He is also credited with first studying the equation $x^2 - py^2 = 1$ for a prime p. The Arab mathematician al-Khowarizmi based some of his work on the Arabic translation of Brahmagupta's work (see Biography 1.9 on page 34).

Another classic illustration of Theorem 1.13 is the following, which is due to the Hindu mathematician Brahmagupta.

◆ The Egg Basket Problem

Suppose that a basket has n eggs in it. If the eggs are taken from the basket $2, 3, 4, 5$, and 6 at a time, there remain $1, 2, 3, 4$, and 5 eggs in the basket, respectively. If the eggs are removed from the basket 7 at a time, then no eggs

remain in the basket. What is the smallest value of n such that the above could occur?

Essentially, this problem asks for a value of x such that

$$x \equiv j = r_j \,(\text{mod } j+1) \text{ for } j = 1, 2, 3, 4, 5 \text{ and } x \equiv 0 \,(\text{mod } 7).$$

Since $\ell = \text{lcm}(2, 3, 4, 5, 6, 7) = 420$, then we may choose

$$m_1 = 1, m_2 = 3, m_3 = 4, m_4 = 5, m_5 = 1, \text{ and } m_6 = 7.$$

Thus, $s_1 = s_5 = 0$, since $m_1 = m_5 = 1$. Also,

$$s_2 = 280 \text{ since } s_2 \equiv 0 \,(\text{mod } 140) \text{ and } s_2 \equiv 1 \,(\text{mod } 3).$$

Similarly, we calculate that

$$s_3 = 105 \text{ since } s_3 \equiv 0 \,(\text{mod } 105) \text{ and } s_3 \equiv 1 \,(\text{mod } 4).$$

and

$$s_4 = 336 \text{ since } s_4 \equiv 0 \,(\text{mod } 84) \text{ and } s_4 \equiv 1 \,(\text{mod } 5).$$

We need not calculate s_6 since $r_6 = 0$ given that $x \equiv 0 \,(\text{mod } 7)$. Hence, by Theorem 1.13,

$$x_0 = \sum_{j=1}^{6} r_j s_j = 2 \cdot 280 + 3 \cdot 105 + 4 \cdot 336 = 2219.$$

To get the smallest value modulo ℓ, we reduce 2219 modulo 420 to get

$$2219 - 420\lfloor 2219/420 \rfloor = 2219 - 420 \cdot 5 = 119,$$

which is the solution to Brahmagupta's Problem.

The reader may now go to Exercises 1.64–1.66 to test understanding of the solutions of systems of *linear congruences*.

The next aspect of modular arithmetic that we will need later in the text is called *modular exponentiation*. For $b, r \in \mathbb{N}$, this involves the finding of a least nonnegative residue of b^r modulo a given $n \in \mathbb{N}$, especially when the given natural numbers r and n are large. There is an algorithm for doing this that is far more efficient than repeated multiplication of b by itself.

The Repeated Squaring Method

Given $d, n \in \mathbb{N}$, $d > 1$, $x \in \mathbb{Z}$, and

$$d = \sum_{j=0}^{k} d_j 2^j,\ d_j \in \{0, 1\},$$

the goal is to find $x^d \pmod{n}$.
First, we initialize by setting $c_0 = x$ if $d_0 = 1$ and set $c_0 = 1$ if $d_0 = 0$. Also, set $x_0 = x$, $j = 1$, and execute the following steps:

(1) Compute $x_j \equiv x_{j-1}^2 \pmod{n}$.

(2) If $d_j = 1$, set $c_j = x_j \cdot c_{j-1} \pmod{n}$.

(3) If $d_j = 0$, then set $c_j \equiv c_{j-1} \pmod{n}$.

(4) Reset j to $j + 1$. If $j = k + 1$, output

$$c_k \equiv x^d \pmod{n},$$

and terminate the algorithm. Otherwise, go to step (1).

The above algorithm will be valuable in Section 6.5 where we look at the elliptic curve factoring method, for instance, as well as in other areas. The reader may conclude this section by looking at Exercise 1.83 on page 34, which is a practical application of the repeated squaring method for modular exponentiation.

Exercises

1.54. Prove that if $a \in \mathbb{Z}$ is odd, then $a^2 \equiv 1 \pmod{8}$.

1.55. Prove that if $a \in \mathbb{Z}$ is even, then $a^2 \equiv 0 \pmod{4}$.

In Exercises 1.56–1.62, find the unique solution for each modulus in the given systems of linear congruences.

1.56. $x \equiv 2 \pmod{3}$, $x \equiv 3 \pmod{5}$, $x \equiv 1 \pmod{11}$.

1.57. $x \equiv 1 \pmod{7}$, $x \equiv 2 \pmod{13}$, $x \equiv 3 \pmod{17}$.

1.58. $x \equiv 1 \pmod{3}$, $x \equiv 7 \pmod{7}$, $x \equiv 1 \pmod{10}$.

1.59. $x \equiv 1 \pmod{3}$, $x \equiv 5 \pmod{19}$, $x \equiv 1 \pmod{7}$.

1.60. $x \equiv 2 \pmod{3}$, $x \equiv 2 \pmod{5}$, $x \equiv 3 \pmod{7}$, $x \equiv 4 \pmod{11}$.

1.61. $x \equiv 1 \pmod{13}$, $x \equiv 2 \pmod{17}$, $x \equiv 3 \pmod{19}$, $x \equiv 4 \pmod{23}$.

1.62. $x \equiv 5 \pmod{29}$, $x \equiv 4 \pmod{31}$, $x \equiv 3 \pmod{37}$, $x \equiv 2 \pmod{41}$, $x \equiv 1 \pmod{43}$.

1.63. Let $a \in \mathbb{Z}$ and $n \in \mathbb{N}$. Exactly one of the following holds. Prove it and provide counterexamples for the other two.

(a) If $a \equiv \pm 1 \pmod{p}$ for all primes p dividing n, then $a^2 \equiv 1 \pmod{n}$.

(b) If $a^2 \equiv 1 \pmod{n}$, then $a \equiv \pm 1 \pmod{p}$ for all primes p dividing n.

(c) The congruence $a \equiv \pm 1 \pmod{p}$ for all primes p dividing n holds if and only if $a^2 \equiv 1 \pmod{n}$.

1.64. Let $a, b \in \mathbb{Z}$, $n \in \mathbb{N}$. Prove that $ax \equiv b \pmod{n}$ has a solution if and only if $\gcd(a, n) \mid b$.

(*Hint: Use Theorem 1.7 on page 12.*)

1.65. With reference to Exercise 1.64, let $g = \gcd(a, n)$, and suppose that $x = x_0$ is a solution of $ax \equiv b \pmod{n}$. Prove that a given $y \in \mathbb{Z}$ is a solution of $ay \equiv b \pmod{n}$ if and only if $x_0 \equiv y \pmod{n/g}$.

1.66. If x is a solution to $ax \equiv b \pmod{n}$, show that exactly one solution x_0 of the congruence is in the least residue system modulo n/g, and that x_0 is the unique solution modulo n/g of that congruence.

(*Exercises 1.64–1.66 provide necessary and sufficient conditions for the existence of solutions to congruences $ax \equiv b \pmod{n}$, called* linear congruences.)

1.67. Find all $n \in \mathbb{N}$ for which the following congruences hold.

(a) $25 \equiv 2 \pmod{n}$. (b) $-1 \equiv 5 \pmod{n}$.

(c) $1 \equiv -7 \pmod{n}$. (d) $10 \equiv 6 \pmod{n}$.

1.68. Prove that if $a, b \in \mathbb{Z}, n \in \mathbb{N}$ and $a \equiv b \pmod{n}$, then $\gcd(a, n) = \gcd(b, n)$.

1.69. Prove that if $a, b \in \mathbb{Z}$, $n, m \in \mathbb{N}$, and $m|n$ with $a \equiv b \pmod{n/m}$, then there exists a nonnegative integer $j \leq m$ such that

$$a \equiv b + (m - j)n/m \pmod{n}.$$

1.70. Prove that for all $n \in \mathbb{N}$, $\mathfrak{T} = \{\overline{0}, \overline{1}, \ldots, \overline{n-1}\}$ is a complete residue system modulo n. Furthermore, prove that any set of n consecutive integers determines a complete residue system modulo n.

1.71. Let $\mathfrak{T} = \{\overline{r_1}, \ldots, \overline{r_n}\}$ be a complete residue system modulo $n \in \mathbb{N}$.

(a) Suppose $a \in \mathbb{N}$ with $\gcd(a, n) = 1$. Prove that for any $b \in \mathbb{Z}$,

$$\{\overline{ar_1 + b}, \ldots, \overline{ar_n + b}\}$$

is a complete residue system modulo n.

(b) If $\mathfrak{S} = \{\overline{s_1}, \ldots, \overline{s_m}\}$ is a complete residue system modulo $m \in \mathbb{N}$, prove that if $\gcd(m, n) = 1$, then $\{\overline{mr_i + ns_j} : r_i \in \mathfrak{T}, s_j \in \mathfrak{S}\}$ forms a complete residue system modulo mn.

1.72. Prove that there exist arbitrarily long blocks of consecutive natural numbers, no one of which is squarefree. (For n to be *squarefree* means that there is no prime p such that p^2 divides n. Thus, in particular, 1 is squarefree.)

(*Hint: Use the Chinese Remainder Theorem.*)

1.73. Given two relatively prime integers x_1 and x_2, and any two arbitrary integers y_1 and y_2, prove that there exists an integer z such that dividing z by x_1 and x_2, we obtain the remainders y_1 and y_2, respectively.

(*Hint: Use the Chinese Remainder Theorem. This result is a reason that it is called a* remainder *theorem.*)

1.74. Prove that each natural number n can be uniquely represented in the form $n = m^2\ell$ where $m, \ell \in \mathbb{N}$ and ℓ is squarefree. (See Exercise 1.72.)

1.75. If $a, b, n_i \in \mathbb{N}$ for $i = 1, 2, \ldots, k$, and $\ell = \operatorname{lcm}(n_1, n_2, \ldots, n_k)$, prove that $a \equiv b \pmod{n_j}$ for each i if and only if $a \equiv b \pmod{\ell}$.

1.76. A prime p is called a Wilson prime if $(p-1)! \equiv -1 \pmod{p^2}$ (see Biography 1.10). Find all Wilson primes less than 564.

(*Other than the ones that the reader will find in the solution to this exercise, there are no other known ones less than* $5 \cdot 10^8$. *See* [20]. It is conjectured that there are infinitely many Wilson primes.)

1.77. Find all incongruent solutions modulo the given modulus in each of the following.

(a) $5x \equiv 1 \pmod{9}$.
(b) $4x \equiv 5 \pmod{27}$.
(c) $10x \equiv 3 \pmod{11}$.
(d) $-x \equiv 9 \pmod{49}$.
(e) $3x \equiv 17 \pmod{103}$.
(f) $103x \equiv 3 \pmod{211}$.

1.78. (a) Find $x, y \in \mathbb{Z}$ such that $19x + 31y = 1$.

(b) Find $19^{-1} \pmod{31}$.

1.79. If $a \in \mathbb{Z}$ and p is prime, prove that a is its own multiplicative inverse modulo p if and only if $a \equiv 1 \pmod{p}$ or $a \equiv -1 \pmod{p}$.

(In the above case a is said to be a *self-multiplicative inverse.*)

1.80. Prove that $\sum_{j=1}^{n-1} j^3 \equiv 0 \pmod{n}$ for $n \in \mathbb{N}$ if and only if $n \not\equiv 2 \pmod{4}$.

1.81. Let a^{-1} be the inverse of the integer a modulo $n \in \mathbb{N}$, and let b^{-1} be the inverse of the integer b modulo n. Prove that $a^{-1}b^{-1}$ is the inverse of ab modulo n.

1.82. Let $b \in \mathbb{N}$ and let m be the product of all natural numbers less than b and relatively prime to b (for instance if b is prime, then $m = (b-1)!$). Prove that if b is one of the forms 4, p^t, or $2p^t$, where $t \in \mathbb{N}$ and $p > 2$ is prime, then $m \equiv -1 \pmod{b}$. (*Hint: Use Exercise 1.79.*)

1.83. Use the repeated squaring method on page 31 to find the least nonnegative residue of 3^{61} modulo 101.

☆ 1.84. For given pairwise relatively prime natural numbers $n_1, n_2, \ldots, n_\ell$, prove that

$$\mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}/n_1\mathbb{Z} \oplus \cdots \oplus \mathbb{Z}/n_\ell\mathbb{Z},$$

where $n = n_1 \cdot n_2 \cdots n_\ell$.

(See Appendix A for a discussion of the abstract algebra needed for this exercise. This exercise is a restatement of the Chinese Remainder Theorem 1.12, which may be used to prove this result.)

Biography 1.9 Mohammed ibn Musa al-Khowarizmi *was an Arab scholar to whom we owe the introduction of the Hindu-Arabic number system. In around 825 A.D. he completed a book on arithmetic, which was later translated into Latin in the twelfth century under the title* Algorithmi de numero Indorum. *This book is one of the best-known means by which the Hindu-Arabic number system was introduced to Europe after being introduced into the Arab world* (*also see Biography 1.4 on page 8*). *This may account for the widespread, although mistaken, belief that our numerals are Arabic in origin. Not long after Latin translations of his book began appearing in Europe, readers began to attribute the new numerals to al-Khowarizmi and began contracting his name, concerning the use of these numerals, to* algorism, *and ultimately to* algorithm. *Also, al-Khowarizmi wrote a book on algebra,* Hisab al-jabr wa'lmuqābala. *The word* algebra *is derived from* al-jabr *or* restoration. *In the Spanish work* Don Quixote, *which came much later, the term* algebrist *is used for a* bone-setter *or* restorer. *Al-Khowarizmi lived during the caliphate of al-Mamun* (809–833 A.D.) *who had a vision in which he was visited by Aristotle. After this encounter he was driven to have the Greek classics translated into Arabic. Among them were Ptolemy's* Almagest *and the complete volumes of Euclid's* Elements.

Biography 1.10 John Wilson *was born on August 6, 1741, in Applethwaite, Westmoreland, England. On July 7, 1764, he was elected as a Fellow of Peterhouse, Cambridge, where he studied. On March 13, 1782, he was elected Fellow of the Royal Society, and was appointed king's counsel on April 24 of that year. The latter was part of his legal career, which he began on January 22, 1763. On November 15, 1786, he was knighted for his numerous accomplishments. Wilson married Mary Ann Adair on April 7, 1788, and the marriage produced a son and two daughters. However, he died only five years later, on October 18, 1793, in Kendal, Westmoreland, where grew up.*

1.4 Euler, Fermat, and Wilson

The following famous result was conjectured by John Wilson (see Biography 1.10 on page 34) on the basis of some heuristic evidence. See Exercise 1.76 on page 33 for Wilson's other claim to fame. The reader, in advance of reading the next result, should solve Exercise 1.79 on page 33, which is used in the proof.

Theorem 1.14 **Wilson's Theorem**

If p is a prime, then

$$(p-1)! \equiv -1 \pmod{p}.$$

Proof. The result is trivial if $p \leq 3$, so we assume that $p > 3$. By Definition 1.11 on page 24, any $a \in \mathbb{Z}$ with $2 \leq a \leq p-2$ has a unique $a^{-1} \in \mathbb{Z}$ with $2 \leq a^{-1} \leq p-2$ where

$$aa^{-1} \equiv 1 \pmod{p}.$$

By Exercise 1.79, $a = a^{-1}$ if and only if $a = 1$ or $a = p-1$. Therefore, the product of the values a^{-1} for $2 \leq a^{-1} \leq p-2$ is just the product of the integers $a = 2, 3, \ldots, p-2$ in some order. Therefore, after possible rearrangement of those values of a, we have

$$(p-2)! \equiv 1 \pmod{p}.$$

Thus,

$$(p-1)! \equiv (p-1) \equiv -1 \pmod{p},$$

which secures the result. □

The first to actually prove Theorem 1.14 was Lagrange (see Biography 1.12 on page 36).

Biography 1.11 *The Swiss mathematician* Leonard Euler (1707–1783) *studied under Jean Bernoulli (1667–1748). Euler was extremely prolific. He published over five hundred papers during his lifetime, and another three hundred and fifty have appeared posthumously. It took almost fifty years for the Imperial Academy to finish publication of his works after his death. Euler had spent the years 1727–1741 and 1766–1783 at the Imperial Academy in St. Petersburg under the invitation of Peter the Great. Euler lost the sight in his right eye in 1735, and he was totally blind for the last seventeen years of his life. Nevertheless, he had a phenomenal memory, and so his mathematical output remained high. In fact, about half of his works were written in those last seventeen years. He contributed not only to number theory, but also to other areas of mathematics such as graph theory. It may even be argued that he essentially founded that branch of mathematics. He died on September 18, 1783.*

Example 1.9 If $p = 17$, then for each $a \in \mathbb{N}$ with $2 \leq a \leq 15$, we have

$$2 \cdot 9 \equiv 3 \cdot 6 \equiv 4 \cdot 13 \equiv 5 \cdot 7 \equiv 8 \cdot 15 \equiv 10 \cdot 12$$
$$\equiv 11 \cdot 14 \equiv 1 \pmod{17}.$$

Therefore, $16! \equiv 16 \equiv -1 \pmod{17}$.

Lagrange also proved the following.

Theorem 1.15 **The Converse of Wilson's Theorem**

If $n \in \mathbb{N}$ and $(n-1)! \equiv -1 \pmod{n}$, then n is a prime.

Proof. If $p \mid n$ is a prime and $p < n$, then $p \mid (n-1)!$. Thus, given that $(n-1)! \equiv -1 \pmod{n}$, we have

$$0 \equiv (n-1)! \equiv -1 \pmod{p},$$

a contradiction. □

Biography 1.12 Joseph-Louis Lagrange (1736–1813) *was born on January 25, 1736, in Turin, Sardinia-Piedmont (now Italy). Although Lagrange's primary interests as a young student were in classical studies, his reading of an essay by Edmund Halley (1656–1743) on the calculus converted him to mathematics. While still in his teens, Lagrange became a professor at the Royal Artillery School in Turin in 1755 and remained there until 1766 when he succeeded Euler (see Biography 1.11 on page 35) as director of mathematics at the Berlin Academy of Science. Lagrange left Berlin in 1787 to become a member of the Paris Academy of Science, where he remained for the rest of his professional life. In 1788 he published his masterpiece* Mécanique Analytique, *which may be viewed as both a summary of the entire field of mechanics to that time and an establishment of mechanics as a branch of analysis, mainly through the use of the theory of differential equations. When he was fifty-six, he married a young woman almost forty years younger than he, the daughter of the astronomer Lemonnier. She became his devoted companion until his death in the early morning of April 10, 1813, in Paris.*

Another famous result that is linked to Exercise 1.79 on page 33 in the proof of Theorem 1.14 is the following proved by Fermat (see Biography 1.13 on page 37). Moroever, since Exercise 1.71 on page 32 is employed in the following proof, the reader should solve it in advance.

Theorem 1.16 **Fermat's Little Theorem**

If $a \in \mathbb{Z}$, and p is a prime such that $\gcd(a, p) = 1$, then

$$a^{p-1} \equiv 1 \pmod{p}.$$

Proof. By part (a) of Exercise 1.71,

$$\prod_{k=1}^{p-1} ak \equiv \prod_{k=1}^{p-1} k \pmod{p}.$$

However,

$$\prod_{k=1}^{p-1} ak \equiv a^p \prod_{k=1}^{p-1} k \equiv a^{p-1}(p-1)! \pmod{p}.$$

Hence, $a^{p-1} \equiv 1 \pmod{p}$. □

When $p > 2$ and $p \nmid a \in \mathbb{Z}$, we see that $b = a^{(p-1)/2}$ is its own multiplicative inverse modulo p, since $b^2 = a^{p-1} \equiv 1 \pmod{p}$.

Notice that Theorem 1.16 tells us that $a^{-1} \equiv a^{p-2} \pmod{p}$, when $p \nmid a$, so this provides a means for computing inverses in $\mathbb{Z}/p\mathbb{Z}$.

Biography 1.13 Pierre Fermat (1607–1665) *is most often listed in the historical literature as having been born on August 17, 1601, which was actually the baptismal date of an elder brother, also named Pierre Fermat, born to Fermat's father's first wife, who died shortly thereafter. Fermat, the mathematician, was a son of Fermat's father's second wife. Note also that Fermat's son gave Fermat's age as fifty-seven on his tombstone — see* http://library.thinkquest.org/27694/Pierre%20de%20Fermat.htm*, for instance. Fermat attended the University of Toulouse and later studied law at the University of Orléans where he received his degree in civil law. By 1631, Fermat was a lawyer as well as a government official in Toulouse. This entitled him to change his name to Pierre* de *Fermat. He was ultimately promoted to the highest chamber of the criminal court in 1652. Throughout his life Fermat had a deep interest in number theory and incisive ability with mathematics. There is little doubt that he is best remembered for Fermat's Last Theorem (FLT). (FLT says that*

$$x^n + y^n = z^n$$

has no solutions $x, y, z, n \in \mathbb{N}$ for $n > 2$. This has recently been solved after centuries of struggle by Andrew Wiles. See [73].*) However, Fermat published none of his discoveries. It was only after Fermat's son Samuel published an edition of Bachet's translation of Diophantus's* Arithmetica *in 1670 that his father's margin notes, claiming to have had a proof, came to light. The attempts to prove FLT for over three centuries have led to discoveries of numerous results and the creation of new areas of mathematics. Fermat died on January 12, 1665, in Castres, France.*

Fermat's Little Theorem, which is worthy of the description *a gem*, was generalized by Euler. In order to understand how he did this, we need to introduce another concept that bears Euler's name.

Definition 1.12 **Euler's ϕ-Function**

For any $n \in \mathbb{N}$ the Euler ϕ-function*, also known as* Euler's Totient (*see Biography 1.14 on the following page*)*, $\phi(n)$ is defined to be the number of $m \in \mathbb{N}$ such that $m < n$ and $\gcd(m, n) = 1$.*

Note that Gauss introduced the symbol $\phi(n)$ (see [35, Articles 38–39, pp. 20–21]), and Euler used the symbol πn to denote $\phi(n)$—the Totient

Example 1.10 If p is prime, then all $j \in \mathbb{N}$ with $j < p$ is relatively prime to p, so $\phi(p) = p - 1$.

Example 1.11 Let $n \in \mathbb{N}$. Then the cardinality of $(\mathbb{Z}/n\mathbb{Z})^*$ is $\phi(n)$. See (1.2) on page 25 and Exercise 1.84 on page 34.

Biography 1.14 James Joseph Sylvester (1814–1897) *gave the name* totient *to the function* $\phi(n)$. *He defined the* totatives *of* n *to be the natural numbers* $m < n$ *relatively prime to* n. *Sylvester was born in London, England, on September 3, 1814. He taught at University of London from 1838 to 1841 with his former teacher Augustus De Morgan (1806–1871). Later he left mathematics to work as an actuary and a lawyer. This brought him into contact with Arthur Cayley (1821–1895) who also worked the courts of Lincoln's Inn in London, and thereafter they remained friends. Sylvester returned to mathematics, being appointed professor of mathematics at the Military Academy at Woolrich in 1854. In 1876 he accepted a position at the newly established Johns Hopkins University. He founded the first mathematical journal in the U.S.A., the* American Journal of Mathematics. *In 1883, he was offered a professorship at Oxford University. This position was to fill the chair left vacant by the death of the Irish number theorist Henry John Stephen Smith (1826–1883). When his eyesight began to deteriorate in 1893, he retired to live in London. Nevertheless, his enthusiasm for mathematics remained until the end as evidenced by the fact that in 1896 he began work on Goldbach's Conjecture (which says that every even integer* $n > 2$ *is a sum of two primes). He died in London on March 15, 1897, from complications involving a stroke.*

◆ Applications of Euler's Totient

Theorem 1.17 **The Arithmetic of the Totient**

If $n = \prod_{j=1}^{k} p_j^{a_j}$ *where the* p_j *are distinct primes, then*

$$\phi(n) = \prod_{j=1}^{k} (p_j^{a_j} - p_j^{a_j - 1}) = \prod_{j=1}^{k} \phi(p_j^{a_j}).$$

Proof. We perform induction on k, where

$$n = \prod_{j=1}^{k} p_j^{a_j}.$$

First we prove that the result holds for $k = 1$. Those natural numbers less than or equal to p^a and divisible by p are precisely those $j = ip$ for $i = 1, 2, \ldots, p^{a-1}$, so there are p^{a-1} of them. Hence,

$$\phi(p^a) = p^a - p^{a-1}.$$

Now we may assume that $k > 1$, and

$$\phi(M) = \prod_{j=1}^{k-1} (p_j^{a_j} - p_j^{a_j - 1}),$$

where

$$M = \prod_{j=1}^{k-1} p_j^{a_j}.$$

Claim 1.1 *If $n \in \mathbb{N}$ and p is prime, then*

$$\phi(pn) = \begin{cases} p\phi(n) & \text{if } p|n, \\ (p-1)\phi(n) & \text{otherwise.} \end{cases}$$

In order to calculate the value $\phi(pn)$, we look at each of the range of numbers

$$in+1, in+2, \ldots, in+n, \text{ for } i = 0, 1, \ldots, p-1.$$

If we eliminate all of the values j from these intervals that satisfy $\gcd(n, j) > 1$, then we have $p\phi(n)$ integers left. If $p|n$, then this is all of those values relatively prime to pn. However, if $p \nmid n$, then we must also eliminate all those values ip for $i = 1, 2, \ldots, n$. Of these, those ip with $\gcd(i, n) > 1$ have already been eliminated. Hence, there are just $\phi(n)$ more to eliminate, namely

$$\phi(pn) = p\phi(n) - \phi(n) = (p-1)\phi(n),$$

and we have Claim 1.1. Therefore, it follows that

$$\phi(p_k^{a_k} M) = p_k \phi(p_k^{a_k - 1} M) = p_k^2 \phi(p_k^{a_k - 2} M) = \ldots$$

$$= p_k^{a_k - 1} \phi(p_k M) = p_k^{a_k - 1}(p_k - 1)\phi(M)$$

and by the induction hypothesis, this equals

$$p_k^{a_k - 1}(p_k - 1) \prod_{j=1}^{k-1} (p_j^{a_j} - p_j^{a_j - 1}) = \prod_{j=1}^{k} (p_j^{a_j} - p_j^{a_j - 1}) = \prod_{j=1}^{k} \phi(p_j^{a_j}).$$

This completes the induction and secures the result. □

In order to get Euler's generalization of Fermat's Little Theorem, we need another concept.

Definition 1.13 **Reduced Residue Systems**

If $n \in \mathbb{N}$, then a set

$$\mathcal{R} = \{m_j \in \mathbb{N} : \gcd(m_j, n) = 1 \text{ and } m_j \not\equiv m_k \pmod{n} \text{ where } 1 \leq j \neq k \leq \phi(n)\}$$

is called a reduced residue system modulo n.

Remark 1.3 *If the set*

$$\mathcal{R} = \{r_1, \ldots, r_{\phi(n)}\}$$

is a reduced residue system modulo n, then so is

$$\mathfrak{R} = \{mr_1, \ldots, mr_{\phi(n)}\}$$

for $m \in \mathbb{N}$ with $\gcd(m, n) = 1$. To see this, note that since

$$\gcd(m, n) = \gcd(r_j, n) = 1,$$

then

$$\gcd(mr_j, n) = 1 \text{ for all natural numbers } j \leq \phi(n).$$

If

$$mr_j \equiv mr_k \,(\mathrm{mod}\ n)$$

for some $j \neq k$ with $1 \leq j, k \leq \phi(n)$, then

$$r_j \equiv r_k \,(\mathrm{mod}\ n),$$

by Proposition 1.3 on page 19, a contradiction.

Theorem 1.18 **Euler's Generalization of Fermat's Little Theorem**

If $n \in \mathbb{N}$ and $m \in \mathbb{Z}$ such that $\gcd(m, n) = 1$, then

$$m^{\phi(n)} \equiv 1 \,(\mathrm{mod}\ n).$$

Proof. By the discussion immediately preceding the theorem, each element in $\mathcal{R}$ is congruent to a unique element in $\mathfrak{R}$ modulo n. Hence,

$$\prod_{j=1}^{\phi(n)} r_j \equiv \prod_{j=1}^{\phi(n)} mr_j \equiv m^{\phi(n)} \prod_{j=1}^{\phi(n)} r_j \,(\mathrm{mod}\ n),$$

and $\gcd(\prod_{j=1}^{\phi(n)} r_j, n) = 1$, so

$$m^{\phi(n)} \equiv 1 \,(\mathrm{mod}\ n),$$

by Proposition 1.3 on page 19. □

Example 1.12 By Euler's Theorem, $3^{\phi(7)-1} \equiv 3^{6-1} = 3^5 \equiv 5 \,(\mathrm{mod}\ 7)$, and 5 is a (least) multiplicative inverse of 3 modulo 7.

Example 1.12 is a special case of a result that is the content of Exercise 1.97 on page 43, which is in turn a simple application of Theorem 1.18.

Exercises

1.85. Let $n \in \mathbb{N}$ such that $n \equiv 3 \pmod 4$. Prove that $x^2 \equiv -1 \pmod n$ is not solvable.

1.86. Let p be an odd prime. Establish the binomial coefficient congruence,

$$\binom{p}{j} \equiv 0 \pmod p$$

for all natural numbers $j \leq p-1$.

1.87. Let $b \in \mathbb{N}$ and let q be a prime such that q does not divide b. Prove that there exists an $n \in \mathbb{N}$ such that $n \mid (q-1)$ and $q \mid (b^{(q-1)/n} - 1)$. (This is called *Fermat's divisibility test* — see Biography 1.13 on page 37.)

(*Hint: Use the Binomial Theorem A.3 on page 307 and Fermat's Little Theorem.*)

1.88. If p is an odd prime, prove that any prime divisor q of $2^p - 1$ must be the form $q = 2mp + 1$ for some $m \in \mathbb{N}$. Also, prove that if $m \in \mathbb{N}$ is the smallest such that $q \mid (b^m - 1)$, then $q \mid (b^t - 1)$ whenever $m|t$.

(*Hint: Use Exercise 1.87.*)

1.89. Generalize the Fibonacci sequence (defined on page 8) by setting $g_1 = a \in \mathbb{Z}$, $g_2 = b \in \mathbb{Z}$, and

$$g_j = g_{j-1} + g_{j-2} \text{ for } j \geq 3.$$

Prove that $g_j = aF_{j-2} + bF_{j-1}$.

☆ 1.90. The nth Fermat number for $n \in \mathbb{N}$ is given by $\mathfrak{F}_n = 2^{2^n} + 1$. Prove that every prime divisor of $\mathfrak{F}_n$ is of the form $2^{n+1}k + 1$ for some $k \in \mathbb{N}$.

(*Hint: Use Exercise 1.87 and the Binomial Theorem.*)

(*The above exercise is Euler's result on Fermat numbers — see Biography 1.11 on page 35.*)

☆ 1.91. The following is called *Legendre's Divisibility Criterion.* (See Biography 1.15 on page 42.)

Let p be a prime and $n \in \mathbb{N}$. Then

(a) If $p \mid (a^n+1)$, then either $p = 2nm+1$ for some $m \in \mathbb{N}$, or $p \mid (a^{n/k}+1)$ where k is an odd divisor of n.

(b) If $p \mid (a^n - 1)$, then either $p = nb + 1$ for some $b \in \mathbb{N}$, or $p \mid a^k - 1$ where $k \mid n$.

1.92. Let $\mathfrak{F}_n$ be as in Exercise 1.90. Prove that if p is a prime dividing $\mathfrak{F}_n$, then the smallest $m \in \mathbb{N}$ such that $p \mid (2^m - 1)$ is $m = 2^{n+1}$.

(*Hint: Use the division algorithm and the Binomial Theorem.*)

1.93. Prove that

$$\sum_{j=1}^{p-1} j^{p-1} \equiv -1 \,(\mathrm{mod}\ p)$$

for any prime p. (*It is an open question as to whether*

$$\sum_{j=1}^{n-1} j^{n-1} \equiv -1 \,(\mathrm{mod}\ n)$$

for a given $n \in \mathbb{N}$ implies that n is prime. However, it has been verified up to 10^{1700}. See [40, p. 37]).

Biography 1.15 Adrien-Marie Legendre (1752–1833) *was born on September 18, 1752, in Paris, France. He was educated at the Collège Mazarin in Paris. During the half decade 1775–1780, he taught along with Laplace (1749–1827) at Ecole Militaire. He also took a position at the Académie des Sciences, becoming first* adjoint *in 1783, then* associé *in 1785, and his work finally resulted in his election to the Royal Society of London in 1787. In 1793, the Académie was closed due to the Revolution, but Legendre was able to publish his phenomenally successful book* Eléments de Géométrie *in 1794, which remained the leading introductory text in the subject for over a century. In 1795, the Académie was reopened as the* Institut National des Sciences et des Arts *and met in the Louvre until 1806. In 1808, Legendre published his second edition of* Théorie des Nombres*, which included Gauss's proof of the Quadratic Reciprocity Law (about which we will learn in Chapter 4). Legendre also published his three-volume work* Exercises du Calcul Intégral *during 1811–1819. Then his three-volume work* Traité des Fonctions Elliptiques *was published during the period 1825–1832. Therein he introduced the name "Eulerian Integrals" for beta and gamma functions. This work also provided the fundamental analytic tools for mathematical physics, and today some of these tools bear his name, such as* Legendre Functions*. In 1824, Legendre had refused to vote for the government's candidate for the Institute National, and for taking this position his pension was terminated. He died in poverty on January 10, 1833, in Paris.*

1.94. Let $b \in \mathbb{N}$ and let m be the product of all natural numbers less than b and relatively prime to b. Prove that if b is *not* of one of the forms: 4, p^t, or $2p^t$ where $t \in \mathbb{N}$ and $p > 2$ is prime, then $m \equiv 1 \,(\mathrm{mod}\ b)$.

(This result, in conjunction with Exercise 1.82 on page 34, is Gauss' generalization of Wilson's Theorem presented in [35, Article 78, p. 51].)

1.95. Suppose that $p \equiv 3 \,(\mathrm{mod}\ 4)$ is prime. Prove that $\left(\frac{p-1}{2}\right)! \equiv \pm 1 \,(\mathrm{mod}\ p)$.

(*Hint: Use Wilson's Theorem and Exercise 1.79 on page 33.*)

1.96. With reference to Exercise 1.94, solve the following. If $b \in \mathbb{N}$ is composite and $m \equiv \pm 1 \,(\mathrm{mod}\ b^2)$, then b is called a *Wilson composite*. The only

Wilson composite less than $5 \cdot 10^4$ is 5971. Find a Wilson composite bigger than $5 \cdot 10^5$.

1.97. Suppose that $m \in \mathbb{Z}$, $n \in \mathbb{N}$ and $\gcd(m, n) = 1$. Prove that $m^{\phi(n)-1}$ is a multiplicative inverse of m modulo n.

1.98. Prove that $\phi(mn) = \phi(m)\phi(n)$ for any relatively prime $m, n \in \mathbb{N}$.

1.99. Use Exercise 1.98 to prove the following. If $m, n \in \mathbb{N}$ with $g = \gcd(m, n)$, then

$$\phi(mn) = g\phi(m)\phi(n)/\phi(g).$$

1.100. Prove that if $d \mid n \in \mathbb{N}$, then $\phi(d) \mid \phi(n)$.

1.101. Solve for minimum $n \in \mathbb{N}$ in the coconut problem on page 25 for the case of five sailors who subdivide into five piles, each time giving the monkey one coconut.

1.102. Prove that any prime divisor of $M_p = 2^p - 1$ for $p > 2$ is of the form $2kp + 1$ for some $k \in \mathbb{N}$. (See Exercise 1.50 on page 16.)

1.103. If n is composite, then n is a *Carmichael number* if

$$b^{n-1} \equiv 1 \pmod{n} \text{ for all } b \in \mathbb{N} \text{ such that } \gcd(b, n) = 1.$$

Suppose that $n = \prod_{j=1}^{r} p_j$ $(r \geq 2)$ for distinct odd primes p_j. Prove that $(p_j - 1)|(n - 1)$ for all nonnegative integers $j \leq r$ if and only if n is a Carmichael number.

(It has been observed that, if the converse to Exercise 1.93 on page 42 *fails* to hold for some n, then that number would be a Carmichael number, and that for any prime $p|n$, we would have that $(p - 1)|(n - 1)$.)

Biography 1.16 Robert Daniel Carmichael (1879–1967) *was born in Goodwater, Alabama. In 1911, he received his doctorate from Princeton under the direction of G.D. Birkhoff. In 1912, he conjectured that there are infinitely many of the numbers that now bear his name. In 1992, W. Alford, A. Granville, and C. Pomerance proved his conjecture, see* [40, p. 30]. *Carmichael Numbers were generalized to Lucas Sequences by Williams* [93] *in 1977.*

1.104. Prove that if n is composite and $\phi(n) \mid (n - 1)$, then n is squarefree. (See Exercise 1.72 on page 33.)

☆ 1.105. Let $n \in \mathbb{N}$. Prove that for all $a \in \mathbb{Z}$, $b^b \equiv a \pmod{n}$ for some $b \in \mathbb{N}$ if and only if $\gcd(n, \phi(n)) = 1$.

1.106. Let $a \in \mathbb{Z}$, $n > 1$ a natural number with $\gcd(a, n) = 1$, and let r be the smallest positive integer such that $a^r \equiv 1 \pmod{n}$. Prove that $r|\phi(n)$.

(The notion in this exercise is the main topic of Section 1.5.)

1.5 Primitive Roots

In order to study the primality testing algorithms and related phenomena in the text, we need to acquaint ourselves with the notion mentioned in the section header. Toward this end, we first need the following concept related to Euler's Theorem 1.18, which tells us that for $m \in \mathbb{Z}$ and $n \in \mathbb{N}$ with $\gcd(m,n) = 1$, we have $m^{\phi(n)} \equiv 1 \pmod{n}$. One may naturally ask for the *smallest* exponent $e \in \mathbb{N}$ such that $m^e \equiv 1 \pmod{n}$.

Definition 1.14 **Modular Order of an Integer**

Let $m \in \mathbb{Z}$, $n \in \mathbb{N}$ and $\gcd(m,n) = 1$. Then the order of m modulo n *is the smallest $e \in \mathbb{N}$ such that $m^e \equiv 1 \pmod{n}$, denoted by $e = \mathrm{ord}_n(m)$, and we say that m* belongs to the exponent e *modulo n.*

Note that the modular order of an integer given in Definition 1.14 is the same as the element order in the group $(\mathbb{Z}/n\mathbb{Z})^*$.

Example 1.13 Clearly 2 has order 2 modulo 3, so $\mathrm{ord}_3(2) = 2 = \phi(3)$. However, 7 has order 1 modulo 3, so $\mathrm{ord}_3(7) = 1$. A more substantial instance is for the prime $p = 3677$, where $7^{1838} \equiv 1 \pmod{p}$ but $7^e \not\equiv 1 \pmod{p}$ for any $e < 1838$, so $\mathrm{ord}_p(7) = 1838$.

Notice in Example 1.13 that the order of each integer divides $\phi(n)$.

Proposition 1.5 **Divisibility by the Order of an Integer**

If $m \in \mathbb{Z}$, $d, n \in \mathbb{N}$ such that $\gcd(m,n) = 1$, then $m^d \equiv 1 \pmod{n}$ if and only if $\mathrm{ord}_n(m) \mid d$. In particular, $\mathrm{ord}_n(m) \mid \phi(n)$.

Proof. If $\mathfrak{d} = \mathrm{ord}_n(m)$, and $d = \mathfrak{d}x$ for some $x \in \mathbb{N}$, then

$$m^d = (m^{\mathfrak{d}})^x \equiv 1 \pmod{n}.$$

Conversely, if $m^d \equiv 1 \pmod{n}$, then $d \geq \mathfrak{d}$ so there exist integers q and r with $d = q \cdot \mathfrak{d} + r$ where $0 \leq r < \mathfrak{d}$ by the Division Algorithm. Thus, $1 \equiv m^d \equiv (m^{\mathfrak{d}})^q m^r \equiv m^r \pmod{n}$, so by the minimality of $\mathfrak{d}$, $r = 0$. In other words, $\mathfrak{d} \mid d$. In particular (also the content of Exercise 1.106 on page 43) we have that $\mathfrak{d} \mid \phi(n)$. □

Note that we may rephrase Proposition 1.5 in terms of the group theoretic language surrounding $(\mathbb{Z}/n\mathbb{Z})^*$, namely that if $\mathfrak{d}$ is the order of an element $m \in (\mathbb{Z}/n\mathbb{Z})^*$, then for any $d \in \mathbb{N}$, if $m^d = 1 \in (\mathbb{Z}/n\mathbb{Z})^*$, d must be a multiple of $\mathfrak{d}$. We use this language to prove the next fact.

Corollary 1.1 *If $d, n \in \mathbb{N}$, and $m \in \mathbb{Z}$ with $\gcd(m,n) = 1$, then*

$$\mathrm{ord}_n(m^d) = \frac{\mathrm{ord}_n(m)}{\gcd(d, \mathrm{ord}_n(m))}.$$

Proof. With $\mathfrak{d}$ as above, set $f = \mathrm{ord}_n(m^d)$ (the order of m^d in $(\mathbb{Z}/n\mathbb{Z})^*$) and $g = \gcd(d, \mathfrak{d})$. Thus, by Proposition 1.5, $\mathfrak{d} \mid df$, so $(\mathfrak{d}/g) \mid fd/g$. Therefore, by Exercise 1.28 on page 5, $(\mathfrak{d}/g) \mid f$. Also, since

$$(m^{\mathfrak{d}})^{d/g} = (m^d)^{\mathfrak{d}/g} = 1 \in (\mathbb{Z}/n\mathbb{Z})^*,$$

then by our above proposition applied to m^d this time, $f \mid (\mathfrak{d}/g)$. Hence, $f = (\mathfrak{d}/g)$, which is the intended result. □

Those integers m for which $\mathrm{ord}_n(m) = \phi(n)$ are of special importance and are the main topic of this section.

Definition 1.15 **Primitive Roots**

If $m \in \mathbb{Z}$, $n \in \mathbb{N}$ and

$$\mathrm{ord}_n(m) = \phi(n),$$

then m is called a primitive root modulo n. *In other words, m is a primitive root if it belongs to the exponent $\phi(n)$ modulo n.*

Example 1.14 We calculate that

$$\mathrm{ord}_{37}(2) = 36,$$

so 2 is a primitive root modulo the prime 37. Also, we calculate that

$$\mathrm{ord}_{1777}(5) = 1776,$$

so 5 is a primitive root modulo the prime 1777. Also, we see that

$$\mathrm{ord}_{3677}(2) = 3676,$$

so 2 is a primitive root modulo the prime 3677. However, 15 has no primitive roots (see Theorem 1.19 on page 49).

The following proposition contains important consequences of the above.

Proposition 1.6 (a) *Let $m \in \mathbb{Z}$, e, $n \in \mathbb{N}$ and* $\gcd(m, n) = 1$. *Then*

$$\mathrm{ord}_n(m^e) = \mathrm{ord}_n(m)$$

if and only if

$$\gcd(e, \mathrm{ord}_n(m)) = 1.$$

(*In particular, this result says that if m is a primitive root modulo n, then m^e is a primitive root modulo n if and only if* $\gcd(e, \phi(n)) = 1$.)

(b) *Let $m \in \mathbb{Z}$ and $n \in \mathbb{N}$ relatively prime to m. If m is a primitive root modulo n, then $\{m^j\}_{j=1}^{\phi(n)}$ is a complete set of reduced residues modulo n.*

(c) *If $n \in \mathbb{N}$ has a primitive root, there are $\phi(\phi(n))$ incongruent primitive roots modulo n.*

(d) *Let $t, n \in \mathbb{N}$ where $n > 1$ has a primitive root, and $t|\phi(n)$. Then $x^t \equiv 1 \pmod{n}$ has exactly t incongruent roots modulo n.*

Proof. (a) By Corollary 1.1 on page 44,

$$\operatorname{ord}_n(m^e) = \operatorname{ord}_n(m)/\gcd(e, \operatorname{ord}_n(m)).$$

Therefore, $\operatorname{ord}_n(m^e) = \operatorname{ord}_n(m)$ if and only if $\gcd(e, \operatorname{ord}_n(m)) = 1$. In particular, if m is a primitive root modulo n, then $\operatorname{ord}_n(m^e) = \operatorname{ord}_n(m)$ if and only if $\gcd(e, \phi(n)) = 1$.

(b) It suffices to show that $m^i \not\equiv m^j \pmod{n}$ for any $i \neq j$. Suppose to the contrary that $m^i \equiv m^j \pmod{n}$ for $1 \leq i \leq j \leq \phi(n)$, then $m^{i-j} \equiv 1 \pmod{n}$, so $i = j$ by the minimality of $\phi(n)$.

(c) Let m be a primitive root modulo n. By part (b), another primitive root must be of the form m^e with $1 \leq e \leq \phi(n)$. Thus, by part (a), $\operatorname{ord}_n(m) = \operatorname{ord}_n(m^e)$ if and only if $\gcd(e, \phi(n)) = 1$, and there are precisely $\phi(\phi(n))$ such integers e.

(d) Let a be a primitive root modulo n. Then $a, a^2, \ldots, a^t$ are incongruent modulo n for any $t|\phi(n)$, by part (a). If $a^s \equiv x \pmod{n}$ for some $x \in \mathbb{Z}$ and $x^t \equiv 1 \pmod{n}$, then

$$1 \equiv a^{st} \equiv x^t \pmod{n}.$$

However, by Proposition 1.5, $\phi(n)|st$, so s is a multiple of $\phi(n)/t$. Hence, there are exactly t incongruent solutions modulo n. □

It is handy to have a methodology for computing primitive roots. In [35, Articles 73–74, pp. 47–49], Gauss developed a method for computing primitive roots modulo a prime p as follows.

◆ Gauss's Algorithm for Computing Primitive Roots Modulo p

(1) Let $m \in \mathbb{N}$ such that $1 < m < p$ and compute m^t for $t = 1, 2, \ldots$ until $m^t \equiv 1 \pmod{p}$. In other words, compute powers until $\operatorname{ord}_p(m)$ is achieved. If $t = \operatorname{ord}_p(m) = p - 1$, then m is a primitive root and the algorithm terminates. Otherwise, go to step (2).

(2) Choose $b \in \mathbb{N}$ such that $1 < b < p$ and $b \not\equiv m^j \pmod{p}$ for any $j = 1, 2, \ldots, t$. Let $u = \operatorname{ord}_p(b)$.[1.5] If $u \neq p - 1$, then let $v = \operatorname{lcm}(t, u)$. Therefore, $v = ac$ where $a \mid t$ and $c \mid u$ with $\gcd(a, c) = 1$. Let m_1 and b_1 be the least nonnegative residues of $m^{t/a}$ and $b^{u/c}$ modulo p, respectively. Thus, $g = m_1 b_1$ has order $ac = v$ modulo p. If $v = p - 1$, then g is a primitive root and the algorithm is terminated. Otherwise, go to step (3).

(3) Repeat step (2) with v taking the role of t and $m_1 b_1$ taking the role of m. (Since $v > t$ at each step, the algorithm terminates after a finite number of steps with a primitive root modulo p.)

[1.5] Observe that we cannot have $u|t$; since if it did then $b^t \equiv 1 \pmod{p}$. However, it follows from (1) and part (d) of Proposition 1.6 that m^j for $0 \leq j \leq t - 1$ are all the incongruent solutions of $x^t \equiv 1 \pmod{p}$, so $b \equiv m^j \pmod{p}$ for some such j, a contradiction to the choice of b.

Gauss used the following to illustrate his algorithm.

Example 1.15 Let $p = 73$. Choose $m = 2$ in step (1), and we compute $t = \text{ord}_p(m) = 9$ with

$$m^j \equiv 1, 2, 4, 8, 16, 32, 64, 55, 37, 1 \,(\text{mod } p)$$

for $j = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 = t = \text{ord}_p(m)$, respectively. Now we go to step (2) since $m = 2$ is not a primitive root modulo $p = 73$. Since $3 \not\equiv 2^j \,(\text{mod } 73)$ for any natural number $j \leq 9$, we choose $b = 3$. Compute b^j for $j = 1, 2, \ldots u$, where $3^u = 3^{12} \equiv 1 \,(\text{mod } 73)$, where

$$3^j \equiv 3, 9, 27, 8, 24, 72, 70, 64, 46, 65, 49, 1 \,(\text{mod } p)$$

for $j = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 = u = \text{ord}_p(b) = \text{ord}_{73}(3)$, respectively. Since $u \neq p - 1$, then set $v = \text{lcm}(t, u) = 36 = ac = 9 \cdot 4$. Then $m_1 = 2^{t/a} = 2$ and $b_1 = 3^{u/c} = 3^3 = 27$, so $m_1 b_1 = 54$, but $v = \text{ord}_{73}(54) = 36 \neq p - 1$. Thus, we repeat step (2) with $v = 36$ replacing t and choose a value of b not equivalent to any power of the new $m = 54 = m_1 b_1$ modulo 73. Since $b = 5$ qualifies for the role and it is a primitive root modulo 73, the algorithm terminates.

Example 1.14 suggests a natural question. Is 2 a primitive root of infinitely many primes? This is unknown, but a positive answer is conjectured. In fact, there is a more general famous conjecture as follows.

Conjecture 1.1 **Artin's Conjecture**

Every nonsquare integer $m \neq -1$ is a primitive root modulo infinitely many primes.

Biography 1.17 Emil Artin (1898–1962) *was born in Vienna, Austria, in 1898. In World War I, he served in the Austrian army. After the war, he obtained his Ph.D. from the University of Leipzig in 1921. In 1937, he emigrated to the U.S.A. and taught at the University of Notre Dame for one year. Then he spent eight years at Indiana, and in 1946 went to Princeton where he remained for the next twelve years. In 1958, he returned to Germany where he remained for the rest of his life. He was reappointed to the University of Hamburg, which he had left two decades before. Artin contributed to finite group theory, the theory of associative algebras, as well as number theory. His name is attached to numerous deep mathematical entities. For instance, there are the Artin Reciprocity Law, Artin L-functions, and Artinian Rings* (see [62]). *Among Artin's students were Serge Lang, John Tate, and Max Zorn. Artin had interests outside of mathematics, including astronomy, biology, chemistry, and music. In the latter, he excelled as an accomplished musician in his own right, playing not only the flute but also the harpsichord and the clavichord. He died in Hamburg on December 20, 1962.*

Although this conjecture remains open, Heath-Brown proved in 1986 that, with the possible exception of at most two primes, it is true that for each prime p there are infinitely primes q such that p is a primitive root modulo q. For example, there are infinitely many primes q such that one of 2, 3, or 5 is a primitive root modulo q (see [40, p. 249]).

In order to prove a prelude to our first major goal in this section, which is the determination of precisely which integers actually have primitive roots, we need not only Proposition 1.6 on page 45 but also the following, which is an elegant result in its own right. If the reader has not already done so, Exercise 1.27 on page 5 should first be solved since it is used in the following proof.

Proposition 1.7 *For any $n \in \mathbb{N}$,*

$$\sum_{d|n} \phi(d) = n,$$

where d runs over all positive divisors of n.

Proof. By Exercise 1.27, $\gcd(m, n) = d$ if and only if $\gcd(m/d, n/d) = 1$. Therefore, $\phi(n/d)$ is the cardinality $|\mathcal{T}_d|$ of

$$\mathcal{T}_d = \{m \in \mathbb{N} : m \leq n, \text{ and } \gcd(m, n) = d\}.$$

Since

$$\{1, 2, \ldots, n\} = \cup_{d|n} \mathcal{T}_d,$$

then

$$n = \left| \bigcup_{d|n} \mathcal{T}_d \right| = \sum_{d|n} |\mathcal{T}_d| = \sum_{d|n} \phi(n/d) = \sum_{d|n} \phi(d),$$

where the second equality follows from the fact that the $\mathcal{T}_d$ are disjoint, and the last equality follows from the fact that n/d runs over divisors of n as does d. □

Lemma 1.1 **Primitive Roots Modulo a Prime**

Let p be a prime and let $e \in \mathbb{N}$ such that $e \mid (p-1)$. In any reduced residue system modulo p, there exist either 0 or $\phi(e)$ distinct $m \in \mathbb{Z}$, $0 \leq m \leq p-1$, with $\operatorname{ord}_p(m) = e$. In particular, there exist $\phi(p-1)$ primitive roots modulo p.

Proof. Assume that there exists an integer with order e modulo p, and let $r(e)$ be the number of incongruent integers that belong to the exponent e modulo p. Since every natural number $e < p$ must belong to *some* exponent modulo p, then

$$\sum_{e|(p-1)} r(e) = p - 1.$$

By part (d) of Proposition 1.6, $x^e \equiv 1 \pmod{p}$ has exactly e incongruent solutions modulo p, and by part (b) of Proposition 1.6 on page 45, these solutions

are m, m^2, ..., m^{p-1}, where m is a primitive root modulo p. Of these, the ones with $\text{ord}_p(m^j) = e$ are those for which $\gcd(j, e) = 1$ by part (a) of Proposition 1.6, which means there are $\phi(e)$ of them. By Proposition 1.7,

$$\sum_{e|(p-1)} \phi(e) = p - 1,$$

but

$$\sum_{e|(p-1)} r(e) = p - 1,$$

and $r(e) \leq \phi(e)$, so $r(e) = \phi(e)$ for all $e \mid (p-1)$.

The last statement of the lemma follows from Theorem A.7 on page 313, which guarantees that p has a primitive root, and part (c) of Proposition 1.6, which tells us that there exist $\phi(p-1)$ incongruent integers of order $p-1$ modulo p. □

The reader who solves Exercises 1.100 on page 43, as well as 1.107 and 1.112 on page 50, will have set the stage for the following existence result.

Theorem 1.19 **Primitive Root Theorem**

An integer $m > 1$ has a primitive root if and only if m is of the form $2^a p^b$ where p is an odd prime, $0 \leq a \leq 1$, and $b \geq 0$ or $m = 4$. Also, if m has a primitive root, then it has $\phi(\phi(m))$ of them.

Proof. Suppose that $m > 1$ has a primitive root, and let $p^b || m$ where p is prime,[1.6] and $b \in \mathbb{N}$. By Lemma 1.1, and Exercise 1.107, if g is a primitive root modulo p, then either g or $g + p$ is a primitive root modulo p^b. If g is a primitive root modulo p^b, then either g or $g + p$ is odd, so there is an odd primitive root modulo p^b. If h is such a primitive root, then h must also be a primitive root modulo $2p^b$. To see this, let $\text{ord}_{2p^b}(h) = c$, then $c|\phi(2p^b) = \phi(p^b)$, by Proposition 1.5 on page 44. Hence, $c = \phi(p^b) = \phi(2p^b)$.

We have shown that each of $m = 2, 4$ or $m = 2^a p^b$ where p is an odd prime, $0 \leq a \leq 1$, and $b \geq 0$ has a primitive root. To show that no other moduli have primitive roots, suppose that $m = m_1 m_2$ where $m_1 > 2$ and $m_2 > 2$, and $\gcd(m_1, m_2) = 1$. By Exercise 1.100, $\phi(m_j)$ is even for $j = 1, 2$. Therefore, for any $n \in \mathbb{N}$, $n^{\phi(m)/2} \equiv (n^{\phi(m_1)})^{\phi(m_2)/2} \equiv 1 \pmod{m_1}$, and similarly, $n^{\phi(m)/2} \equiv 1 \pmod{m_2}$. Therefore, $n^{\phi(m)/2} \equiv 1 \pmod{m}$, so no $n \in \mathbb{N}$ can be a primitive root modulo m. The last type of modulus to consider is $m = 2^a$. For $a \geq 3$, $n^{2^{a-2}} \equiv 1 \pmod{2^a}$ for all odd $n \in \mathbb{Z}$, by Exercise 1.112. Lastly, by part (c) of Proposition 1.6, there are exactly $\phi(\phi(m))$ primitive roots modulo m. □

Example 1.16 If $m = 22$, then $7, 13, 17, 19$ are the four incongruent primitive roots modulo m. Note that $\phi(\phi(m)) = 4$.

[1.6] This symbol || means that $p^b | m$, but p^{b+1} does not. We say that p^b *exactly* divides m.

Exercises

1.107. Let g be a primitive root modulo a prime $p > 2$. Prove that one of g or $g + p$ is a primitive root modulo p^a for all $a \in \mathbb{N}$.

1.108. Prove that if $\text{ord}_p(m)$ is odd and $p > 2$ is prime, then $m^e \equiv -1 \pmod{p}$ has no solution $e \in \mathbb{N}$.

1.109. Let $m, n \in \mathbb{N}$ be relatively prime. Prove that if a is a primitive root modulo mn, then a is a primitive root modulo both m and n.

1.110. Let m be a primitive root modulo an odd prime p. Prove that, for any prime $q \mid (p-1)$, we must have that $m^{(p-1)/q} \not\equiv 1 \pmod{p}$.

1.111. Let $m \in \mathbb{N}$ and $p > 2$ a prime. Prove that if $m^{(p-1)/q} \not\equiv 1 \pmod{p}$ for all primes $q \mid (p-1)$, then m is a primitive root modulo p.

1.112. Prove that for any integer for $a \geq 3$,

$$n^{2^{a-2}} \equiv 1 \pmod{2^a}$$

for all odd $n \in \mathbb{Z}$.

1.113. If $m, n \in \mathbb{N}$ and $x^n \equiv 1 \pmod{m}$ for every $x \in \mathbb{Z}$ with $\gcd(x, m) = 1$, then n is called a *universal exponent modulo m*. Prove that if n is odd, then n is a universal exponent modulo m if and only if $m|2$.

1.114. Which of the following have primitive roots? Provide an example if such a root exists, and provide a proof that it does not otherwise.

(a) 49. (b) 85.
(c) 14. (d) 2^9.
(e) 202. (f) 3677.

1.115. Find all incongruent primitive roots of the following.

(a) 5. (b) 11.
(c) 10. (d) 19.

1.116. If p is prime, f is called a *Fibonacci primitive root modulo p* if

$$f^2 \equiv f + 1 \pmod{p}.$$

Prove that if f is a Fibonacci primitive root modulo p, then

$$f^{j+1} \equiv F_{j+1} f + F_j \pmod{p},$$

where F_j is the jth Fibonacci number for $j \in \mathbb{N}$, defined on page 8.

1.6 The Index Calculus and Power Residues

The preceding sections have put us in a position to introduce the next important concept, which will have cryptographic applications in the text. If $n \in \mathbb{N}$ has a primitive root m, then by part (b) of Proposition 1.6 on page 45, the values $1, m, m^2, \ldots, m^{\phi(n)-1}$ form a complete set of reduced residues modulo n. It follows from part (c) of Proposition 1.6 that, given any $b \in \mathbb{N}$, there is exactly one nonnegative integer $e \leq \phi(n)$ for which $b \equiv m^e \pmod{n}$. This value has a distinguished name.

Definition 1.16 **Index**

Let $n \in \mathbb{N}$ with primitive root m, and $b \in \mathbb{N}$ with $\gcd(b, n) = 1$. Then for exactly one of the values $e \in \{0, 1, \ldots, \phi(n) - 1\}$, $b \equiv m^e \pmod{n}$ holds. This unique value e modulo $\phi(n)$ is the index of b to the base m modulo n*, denoted by* $\operatorname{ind}_m^n(b)$.

Example 1.17 If $n = 29$, then $m = 2$ is a primitive root modulo n. Also, $\operatorname{ind}_2^{29}(5) = 22$, since $5 \equiv 2^{22} \pmod{29}$, so 5 has index 22 to base 2 modulo 29.

Definition 1.16 gives rise to an arithmetic of its own, the *index calculus*. The following are some of the properties.

Theorem 1.20 **Index Calculus**

If $n \in \mathbb{N}$ and m is a primitive root modulo n, then for any $c, d \in \mathbb{Z}$ each of the following holds.

(1) $\operatorname{ind}_m^n(cd) \equiv \operatorname{ind}_m^n(c) + \operatorname{ind}_m^n(d) \pmod{\phi(n)}$.

(2) *For any $t \in \mathbb{N}$,* $\operatorname{ind}_m^n(c^t) \equiv t \cdot \operatorname{ind}_m^n(c) \pmod{\phi(n)}$.

(3) $\operatorname{ind}_m^n(1) = 0$.

(4) $\operatorname{ind}_m^n(m) = 1$.

(5) $\operatorname{ind}_m^n(-1) = \phi(n)/2$ *for $n > 2$.*

(6) $\operatorname{ind}_m^n(n - c) \equiv \operatorname{ind}_m^n(-c) \equiv \phi(n)/2 + \operatorname{ind}_m^n(c) \pmod{\phi(n)}$.

Proof. Let $\operatorname{ind}_m^n(cd) = x$, $\operatorname{ind}_m^n(c) = y$, and $\operatorname{ind}_m^n(d) = z$.
(1) Since $cd \equiv m^x \pmod{n}$, $c \equiv m^y \pmod{n}$ and $d \equiv m^z \pmod{n}$, then

$$m^{y+z} \equiv cd \equiv m^x \pmod{n},$$

so the result follows from Euler's Theorem 1.18 on page 40.
(2) Since $c \equiv m^y \pmod{n}$, then

$$c^t \equiv m^{yt} \pmod{n},$$

the result follows by applying ind_m^n to both sides.

(3) If $\text{ind}_m^n(1) = w$, then $1 \equiv m^w \pmod{n}$. Since m is a primitive root modulo n, and $0 \leq w < \phi(n)$, by Definition 1.16, then $w = 0$.

(4) Let $\text{ind}_m^n(m) = v$. Since $m \equiv m^v \pmod{n}$, then $v = 1$ by Definition 1.16.

(5) Since $m^{\phi(n)/2} \equiv -1 \pmod{n}$ for $n > 2$, then the result follows as in (2).

(6) Since $m^{\phi(n)/2} \equiv -1 \pmod{n}$, then

$$-c \equiv n - c \equiv -m^y \equiv m^{\phi(n)/2} m^y \equiv m^{\phi(n)/2+y} \pmod{n},$$

so the result follows from Euler's Theorem. □

The reader will recognize that the properties of the index mimic those of logarithms. Hence, if n is a prime p, the index of b to the base p is often called the *discrete logarithm* of b to the base p. For further such properties, see Exercises 1.119–1.122 on page 56. Moreover, part (1) of Theorem 1.20 provides us with a tool for finding indices by reducing it to solving linear congruences

$$cx \equiv b \pmod{n}$$

for $x \in \mathbb{Z}$. To see this note that when this congruence holds, then

$$\text{ind}_m^n(c) + \text{ind}_m^n(x) \equiv \text{ind}_m^n(b) \pmod{\phi(n)},$$

for any primitive root m modulo n. The process is illustrated as follows.

Example 1.18 Suppose that we wish to solve the congruence $7x \equiv 3 \pmod{29}$. Since 2 is a primitive root modulo 29, then

$$\text{ind}_2^{29}(7) + \text{ind}_2^{29}(x) \equiv \text{ind}_2^{29}(3) \pmod{\phi(29)},$$

so

$$\text{ind}_2^{29}(x) \equiv \text{ind}_2^{29}(3) - \text{ind}_2^{29}(7) = 5 - 12 = -7 \equiv 21 \pmod{28}.$$

Therefore, by raising to appropriate exponents we get, $x \equiv 2^{21} \equiv 17 \pmod{29}$.

Part (2) of Theorem 1.20 also provides us with a mechanism for solving power congruences

$$c^x \equiv b \pmod{n}$$

for $x \in \mathbb{Z}$. To see this, note that if this congruence holds, then

$$x \cdot \text{ind}_m^n(c) \equiv \text{ind}_m^n(b) \pmod{\phi(n)}.$$

This methodology is illustrated as follows.

Example 1.19 Suppose we want to solve $3^x \equiv 7 \pmod{29}$. Since

$$x \cdot \text{ind}_2^{29}(3) \equiv \text{ind}_2^{29}(7) \pmod{28},$$

then $5x \equiv 12 \pmod{28}$. Therefore,

$$x \equiv 5^{-1} \cdot 12 \equiv 17 \cdot 12 \equiv 8 \pmod{28}.$$

Thus, $3^8 \equiv 7 \pmod{29}$.

Exercise 1.118 on page 56 is designed to test the methods illustrated in Examples 1.18–1.19.

The above is related to the following notion.

Definition 1.17 **Modular Roots and Power Residues**

If $m, n \in \mathbb{N}$, $c \in \mathbb{Z}$, $\gcd(c, n) = 1$, then c is called an m^{th} power residue *modulo n if $x^m \equiv c \pmod{n}$ for some $x \in \mathbb{Z}$, and x is called an m^{th}* root modulo *n.*

For instance, if $m = 2$, then x is called a *square root modulo n*, and c is called a *quadratic residue modulo n*; if $m = 3$, then c is called a *cubic residue modulo n*, and x is called a *cube root modulo n*, and so on.

If we know a factorization of n into prime powers, then we may find m^{th} roots modulo each prime power and use the Chinese Remainder Theorem to obtain an m^{th} root modulo n.

It is valuable to have a criterion for the solvability of such congruences. With the index calculus as a tool, we may now present such a result. It is essential that the reader solve Exercise 1.121 since this tells us *when* the following power congruence actually has a solution and how many there are.

Theorem 1.21 **Euler's Criterion for Power Residue Congruences**

Let $e, c \in \mathbb{N}$ with $e \geq 2$, $b \in \mathbb{Z}$, $p > 2$ is prime with $p \nmid b$, $g = \gcd(e, \phi(p^c))$, and $g \mid b$. Then the congruence

$$x^e \equiv b \pmod{p^c} \tag{1.3}$$

is solvable if and only if

$$b^{\phi(p^c)/g} \equiv 1 \pmod{p^c}.$$

Proof. Suppose that

$$x^e \equiv b \pmod{p^c}.$$

Then

$$b^{\phi(p^c)/g} \equiv (x^e)^{\phi(p^c)/g} \equiv (x^{e/g})^{\phi(p^c)} \equiv 1 \pmod{p^c},$$

by Euler's Theorem 1.18 on page 40.

Conversely, assume that

$$b^{\phi(p^c)/g} \equiv 1 \pmod{p^c}.$$

By Exercise 1.107 on page 50, there exists a primitive root m modulo p^c. Therefore,

$$b \equiv m^n \pmod{p^c}, \tag{1.4}$$

for some $n \in \mathbb{N}$, by part (b) of Proposition 1.6 on page 45. Hence,

$$1 \equiv b^{\phi(p^c)/g} \equiv m^{n\phi(p^c)/g} \pmod{p^c}.$$

Since m is a primitive root modulo p^c, this implies that

$$\phi(p^c)|n\phi(p^c)/g,$$

so $g|n$. Thus, there exists some $k \in \mathbb{N}$ such that $n = kg$. By Exercise 1.22 on page 5, there exist $x, y \in \mathbb{Z}$ such that

$$ye - x\phi(p^c) = g,$$

so

$$n = kg = kye - kx\phi(p^c). \tag{1.5}$$

By Euler's Theorem again,

$$m^{kx\phi(p^c)} \equiv (m^{kx})^{\phi(p^c)} \equiv 1 \,(\mathrm{mod}\ p^c),$$

so by (1.4)–(1.5),

$$b \equiv bm^{kx\phi(p^c)} \equiv m^{n+kx\phi(p^c)} \equiv m^{kye-kx\phi(p^c)+kx\phi(p^c)} \equiv m^{kye} \equiv (m^{ky})^e \,(\mathrm{mod}\ p^c).$$

Hence, b is an e^{th} power residue modulo p^c, namely $b \equiv x^e \,(\mathrm{mod}\ p^c)$ with $x = m^{ky}$. □

Example 1.20 If $x^5 \equiv 5 \,(\mathrm{mod}\ 27)$, with $g = \gcd(e, \phi(p^c)) = \gcd(5, 18) = 1$, then $\mathrm{ind}_2^{27}(5) = 5$, and

$$5 \cdot \mathrm{ind}_2^{27}(x) \equiv \mathrm{ind}_2^{27}(5) \equiv 5 \,(\mathrm{mod}\ 18),$$

since $m = 2$ is a primitive root mod 3^3. Therefore, $\mathrm{ind}_2^{27}(x) \equiv 1 \,(\mathrm{mod}\ 18)$, so either $\mathrm{ind}_2^{27}(x) \equiv 1 \,(\mathrm{mod}\ 27)$, or $\mathrm{ind}_2^{27}(x) \equiv 19 \,(\mathrm{mod}\ 27)$. Thus, $x \equiv 2 \,(\mathrm{mod}\ 27)$ is the only distinct congruence class that satisfies the given congruence, since $2^{19} \equiv 2 \,(\mathrm{mod}\ 27)$.

Example 1.21 Consider $x^3 \equiv 4 \,(\mathrm{mod}\ 27)$. Then since $3\mathrm{ind}_2^{27}x \equiv \mathrm{ind}_2^{27}4 \equiv 2 \,(\mathrm{mod}\ 27)$, and $\gcd(3, 2) = 1$, there are no solutions to this congruence.

The above examples suggest the following consequence of Theorem 1.21.

Corollary 1.2 **Power Residues at Prime Powers**

Suppose that p is an odd prime, $b \in \mathbb{Z}$, $p \nmid b$ and $e \in \mathbb{N}$. If

$$x^e \equiv b \,(\mathrm{mod}\ p)$$

has a solution, then so does

$$x^e \equiv b \,(\mathrm{mod}\ p^c)$$

for all $c \in \mathbb{N}$.

Proof. We prove this by induction on c. We are given $x^e \equiv b \pmod{p}$, and we assume that $x^e \equiv b \pmod{p^c}$ has a solution. We need only establish that $x^e \equiv b \pmod{p^{c+1}}$ has a solution. If $g = \gcd(e, \phi(p^c))$, then

$$b^{\phi(p^c)/g} \equiv 1 \pmod{p^c},$$

by Theorem 1.21. Thus, there is an integer f such that

$$b^{\phi(p^c)/g} = 1 + fp^c.$$

Therefore, by Claim 1.1 in the proof of Theorem 1.17 on page 39,

$$(1 + p^c f)^p = b^{p\phi(p^c)/g} = b^{\phi(p^{c+1})/g}.$$

However, by the Binomial Theorem,

$$(1 + p^c f)^p = \sum_{j=0}^{p} p^{cj} f^j \binom{p}{j} = 1 + p^{c+1} h,$$

for some $h \in \mathbb{Z}$ since $cj \geq c + 1$ for all $j \geq 2$. Hence, $b^{\phi(p^{c+1})/g} \equiv 1 \pmod{p^{c+1}}$, so by Theorem 1.21, $x^e \equiv b \pmod{p^{c+1}}$ has a solution as required. □

Furthermore, we have the following consequence.

Corollary 1.3 **The Number of Power Residues**

Under the hypothesis of Theorem 1.21, there are $\phi(p^c)/g$ incongruent (nonzero) c^{th}-power residues modulo p^c.

Proof. By Theorem 1.21, we want the number of incongruent solutions of

$$x^{\phi(p^c)/g} \equiv 1 \pmod{p^c}.$$

By part (d) of Proposition 1.6 on page 45, this congruence has exactly $\phi(p^c)/g$ incongruent solutions. □

Example 1.22 Suppose that we want to determine the number of incongruent fourth power residues modulo 27, namely the number of incongruent $b \in \mathbb{N}$ such that $x^4 \equiv b \pmod{27}$. Since

$$g = \gcd(\phi(27), e) = \gcd(18, 4) = 2 \mid \operatorname{ind}_2^{27}(7) = 16,$$

then we know there are such solutions by Exercise 1.121. By Corollary 1.3, there must be $\phi(p^c)/g = 18/2 = 9$ incongruent such solutions. They are

$$b \in \{1, 4, 7, 10, 13, 16, 19, 22, 25\}$$

given by $x \in \{1, 5, 11, 10, 4, 2, 8, 13, 7\}$, respectively.

An important consideration in complexity theory (which we will study in Section 1.8) is the search for an efficient algorithm which, given a prime p and a primitive root m modulo p, computes $\text{ind}_m^p(x)$ for any given $x \in \mathbb{F}_p^*$. These algorithms have significant ramifications for the construction of secure pseudorandom number generators. There is no such algorithm known in general. However, the Silver-Pohlig-Hellman Algorithm for computing discrete logs, described in Appendix E, is an efficient such algorithm in the case where the prime factors of $p-1$ are known and are "small" with respect to p. Note that the α that generates $\mathbb{F}_p^*$ in the description of the Silver-Pohlig-Hellman Algorithm in Appendix E is a primitive root modulo p. Also, the computation of a modulo $p_j^{a_j}$ described in Appendix E is merely the computation of

$$\text{ind}_\alpha^{p_j^{a_j}}(\beta).$$

Exercises

1.117. Find each of the following.

(a) $\text{ind}_2^{13}(7)$ (b) $\text{ind}_2^{13}(11)$
(c) $\text{ind}_2^{19}(5)$ (d) $\text{ind}_2^{19}(10)$
(e) $\text{ind}_3^{17}(6)$ (f) $\text{ind}_3^{17}(8)$

1.118. Using the index calculus, find solutions to each of the following.

(a) $3x^4 \equiv 4 \pmod 7$ (b) $x^3 - 5 \equiv 0 \pmod{49}$
(c) $3x^5 \equiv 4 \pmod{11}$ (d) $2x^3 \equiv 5 \pmod{11}$
(e) $5x^5 \equiv 3 \pmod 7$ (f) $3x^7 \equiv 5 \pmod{11}$

1.119. Let p be a prime with primitive roots a and b. Also, let c be an integer relatively prime to p. Prove that

$$\text{ind}_a^p(c) \equiv \text{ind}_b^p(c) \cdot \text{ind}_a^p(b) \pmod{p-1}.$$

(This property mimics the change of base formula for logarithms.)

1.120. With the same assumptions as in Exercise 1.119, prove that

$$\text{ind}_a^p(p-c) \equiv \text{ind}_a^p(c) + (p-1)/2 \pmod{p-1}.$$

1.121. Show that congruence (1.3) has $g = \gcd(e, \phi(p^c))$ incongruent solutions modulo p^c if $g \mid \text{ind}_a^{p^c}(b)$ for any primitive root a modulo p^c and has none otherwise.

1.122. Let $c \in \mathbb{Z}$, p an odd prime, and $\gcd(p, c) = 1$. If a is a primitive root modulo p, prove that $c \equiv x^2 \pmod p$ for some $x \in \mathbb{Z}$ if and only if $2|\text{ind}_a^p(c)$.

1.123. The *Lucas-Lehmer primality test for Mersenne numbers* is given as follows (see Exercise 1.50 on page 16).

Input $M_n = 2^n - 1$ and execute the following steps.

(1) Set $s_1 = 4$ and compute $s_j \equiv s_{j-1}^2 - 2 \pmod{M_n}$ for $j = 1, 2, \ldots, n-1$.

(2) If $s_{n-1} \equiv 0 \pmod{M_n}$, then conclude that M_n is prime. Otherwise, conclude that M_n is composite.

Use the above algorithm to prove that M_{13} is prime.

Biography 1.18 François Édouard Anatole Lucas (1842–1891) *was born on April 4, 1842, in Amiens, France. In 1864, he graduated from École Normale as Agrégé des sciences mathématiques, meaning that he had passed the state agrégation examination required for a teaching position at French lycées (high schools). However, his first position was assistant astronomer at the Observatory of Paris. He remained there until the Franco-Prussian war in 1870 in which he served as an auxiliary artillery officer. After the war, he became a mathematics teacher at various high schools in Paris. He had interests in recreational mathematics, but his serious interest was in number theory, especially Diophantine analysis. Although he spent only the years 1875–1878 on the problems of factoring and primality testing, his contribution was impressive. Some of the ideas developed by Lucas may be interpreted today as the beginnings of computer design. His death was untimely and unfortunate. While attending a social function, a plate fell and a chip from it cut his face. Later he died from an infection that developed from that cut.*

Biography 1.19 Derrick Henry Lehmer (1905–1991) *was born in Berkeley, California, on February 23, 1905. After graduating with his bachelor's degree from Berkeley in 1927, he went to the University of Chicago to study under L. E. Dickson, but he left after only a few months. Neither the Chicago weather nor the working environment suited him. Brown University offered him a better situation with an instructorship, and he completed both his master's degree and his Ph.D., the latter in 1930. During the period 1930–1940, he had brief stints at the California Institute of Technology; Stanford University; Lehigh University; and Cambridge, England, the latter on a Guggenheim Fellowship. In 1940, he accepted a position at the University of California at Berkeley where he remained until his retirement in 1972. He was a pioneering giant in the world of computational number theory and was widely respected in the mathematical community. The reader is advised to look into his contributions given in his selected works* [51]. *He was also known for his valued sense of humour, as attested by John Selfridge in the forward to the aforementioned selected works, as well as by one of Lehmer's students, Ron Graham. In particular, Selfridge concludes with an apt description of Lehmer's contributions, saying that he "has shown us this beauty with the sure hand of a master."*

1.7 Legendre, Jacobi, & Quadratic Reciprocity

We now proceed to establish an important result that we will need, namely Gauss's *Quadratic Reciprocity Law* (see Biography 1.7 on page 18). After we introduced power residues in Definition 1.17 on page 53, we talked about various types of residues including *quadratic residues modulo* $n \in \mathbb{N}$, which are those integers c for which there exists an integer x with

$$x^2 \equiv c \,(\text{mod } n).$$

If no such integer x exists, then c is called a *quadratic nonresidue modulo* n. Nonresidues for the cubic, quartic, and higher cases are similarly defined. Whether an integer is a residue or nonresidue modulo $n \in \mathbb{N}$ is called its *residuacity* modulo n. There are symbols for representing the residuacity of a given integer. In particular, we will be interested in the quadratic residuacity for which we introduce the following symbol (see Biography 1.15 on page 42).

Definition 1.18 **Legendre's Symbol**

If $c \in \mathbb{Z}$ and $p > 2$ is prime, then

$$\left(\frac{c}{p}\right) = \begin{cases} 0 & \text{if } p \mid c, \\ 1 & \text{if } c \text{ is a quadratic residue modulo } p, \\ -1 & \text{otherwise,} \end{cases}$$

and $\left(\frac{c}{p}\right)$ is called the Legendre Symbol *of c with respect to p.*

Example 1.23 A corollary of Euler's Criterion, Theorem 1.21 on page 53, may now be stated as follows (see Biography 1.11 on page 35). Let p be an odd prime. Then

$$\text{if } c^{(p-1)/2} \equiv 1 \,(\text{mod } p), \text{ then } \left(\frac{c}{p}\right) = 1,$$

and

$$\text{if } c^{(p-1)/2} \equiv -1 \,(\text{mod } p), \text{ then } \left(\frac{c}{p}\right) = -1.$$

This is called Euler's Criterion for quadratic residuacity.

Theorem 1.22 **Properties of the Legendre Symbol**

If $p > 2$ is prime and $b, c \in \mathbb{Z}$ with $p \nmid bc$, then

(1) $\left(\frac{c}{p}\right) \equiv c^{(p-1)/2} \,(\text{mod } p)$.

(2) $\left(\frac{b}{p}\right)\left(\frac{c}{p}\right) = \left(\frac{bc}{p}\right)$.

(3) $\left(\frac{b}{p}\right) = \left(\frac{c}{p}\right)$, *provided* $b \equiv c \,(\text{mod } p)$.

Proof. As seen in Example 1.23, part (1) is a corollary of Euler's Criterion. This may now be used to establish part (2) as follows.

$$\left(\frac{b}{p}\right)\left(\frac{c}{p}\right) \equiv b^{(p-1)/2}c^{(p-1)/2} \equiv (bc)^{(p-1)/2} \equiv \left(\frac{bc}{p}\right) \pmod{p}.$$

Part (3) is an immediate consequence of the definition of a quadratic residue.□

To establish Gauss's Quadratic Reciprocity Law, we first need a technical result proved by him.

☞ **Lemma 1.2** **Gauss's Lemma on Residues**

Let $p > 2$ be a prime and $c \in \mathbb{Z}$ such that $p \nmid c$. Suppose that $\mathfrak{c}$ denotes the cardinality of the set

$$\{\overline{jc} : 1 \le j \le (p-1)/2, \overline{jc} > p/2\},$$

where the $\overline{jc}$ denotes reduction of jc to its least positive residue modulo p. Then

$$\left(\frac{c}{p}\right) = (-1)^{\mathfrak{c}}.$$

Proof. For each natural number $j \le (p-1)/2$, define

$$c_j = \begin{cases} \overline{jc} & \text{if } \overline{jc} < p/2, \\ p - \overline{jc} & \text{if } \overline{jc} > p/2. \end{cases}$$

If $1 \le j, k \le (p-1)/2$, then it is a simple verification that $c_j \equiv c_k \pmod{p}$ if and only if $j = k$. Hence, $c_j \not\equiv c_k \pmod{p}$ for all $j \ne k$ with $1 \le j, k \le (p-1)/2$. Thus, we have $(p-1)/2$ incongruent natural numbers, all less than $p/2$. Therefore,

$$\prod_{j=1}^{(p-1)/2} c_j \equiv \left(\frac{p-1}{2}\right)! \pmod{p}. \tag{1.6}$$

Also, since $p - \overline{jc} \equiv (-1)(\overline{jc}) \pmod{p}$, then

$$\prod_{j=1}^{(p-1)/2} c_j \equiv (-1)^{\mathfrak{c}} \cdot c^{(p-1)/2} \cdot \left(\frac{p-1}{2}\right)! \pmod{p}. \tag{1.7}$$

By equating the two versions of $\prod_{j=1}^{(p-1)/2} c_j$ in (1.6)–(1.7), and dividing through by $(-1)^{\mathfrak{c}} \cdot (\frac{p-1}{2})!$, we get

$$c^{(p-1)/2} \equiv (-1)^{\mathfrak{c}} \pmod{p},$$

and by Euler's Criterion in Example 1.23,

$$c^{(p-1)/2} \equiv \left(\frac{c}{p}\right) \pmod{p},$$

so the result follows. □

An important consequence of Gauss's Lemma that we will need to prove his quadratic reciprocity law is contained in the following.

☞ **Corollary 1.4** *Let $c \in \mathbb{Z}$ be odd, and $p > 2$ prime such that $p \nmid c$. Then*

$$\left(\frac{c}{p}\right) = (-1)^M,$$

where

$$M = \sum_{j=1}^{(p-1)/2} \lfloor jc/p \rfloor.$$

Proof. For each natural number $j \leq (p-1)/2$, we have $jc = q_j p + r_j$, where $r_j \in \mathbb{N}$ with $r_j < p$, by the Division Algorithm. In the notation of the proof of Gauss's Lemma, this means that $r_j = \overline{jc}$, so

$$c_j = \begin{cases} r_j & \text{if } r_j < p/2, \\ p - r_j & \text{if } r_j > p/2, \end{cases}$$

and $q_j = \lfloor jc/p \rfloor$. Arrange the r_j so that $r_j > p/2$ for $j = 1, 2, \ldots, \mathfrak{c}$, and $r_j < p/2$ for $j = \mathfrak{c}+1, \mathfrak{c}+2, \ldots, (p-1)/2$, which is allowed since we know from the proof of Gauss's Lemma that the c_j are just the values $1, 2, \ldots, (p-1)/2$ in some order. Thus, we have

$$\sum_{j=1}^{(p-1)/2} jc = \sum_{j=1}^{(p-1)/2} p\lfloor jc/p \rfloor + \sum_{j=1}^{(p-1)/2} r_j. \tag{1.8}$$

Also, since the c_j are just a rearrangement of the numbers $1, 2, \ldots, (p-1)/2$, then

$$\sum_{j=1}^{(p-1)/2} j = \sum_{j=1}^{\mathfrak{c}} (p - r_j) + \sum_{j=\mathfrak{c}+1}^{(p-1)/2} r_j = p\mathfrak{c} - \sum_{j=1}^{\mathfrak{c}} r_j + \sum_{j=\mathfrak{c}+1}^{(p-1)/2} r_j. \tag{1.9}$$

Subtracting (1.9) from (1.8), we get

$$(c-1) \sum_{j=1}^{(p-1)/2} j = p \left(\sum_{j=1}^{(p-1)/2} \lfloor jc/p \rfloor - \mathfrak{c} \right) + 2 \sum_{j=1}^{\mathfrak{c}} r_j. \tag{1.10}$$

Now we reduce (1.10) modulo 2 to get

$$0 \equiv \left(\sum_{j=1}^{(p-1)/2} \lfloor jc/p \rfloor - \mathfrak{c} \right) \pmod 2,$$

since $c \equiv p \equiv 1 \,(\mathrm{mod}\ 2)$, which means that

$$\mathfrak{c} \equiv \sum_{j=1}^{(p-1)/2} \lfloor jc/p \rfloor \,(\mathrm{mod}\ 2).$$

By Gauss's Lemma, we are now done. □

We are now in a position to establish Gauss's famous result, which he first proved in his masterpiece [35].

Theorem 1.23 **The Quadratic Reciprocity Law**

If $p \neq q$ are odd primes, then

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}.$$

Equivalently,

$$\left(\frac{q}{p}\right) = -\left(\frac{p}{q}\right) \text{ if } p \equiv q \equiv 3 \,(\mathrm{mod}\ 4), \text{ and } \left(\frac{q}{p}\right) = \left(\frac{p}{q}\right) \text{ otherwise.}$$

Proof. First we establish that

$$\frac{p-1}{2}\cdot\frac{q-1}{2} = \sum_{k=1}^{(p-1)/2} \lfloor kq/p \rfloor + \sum_{j=1}^{(q-1)/2} \lfloor jp/q \rfloor. \tag{1.11}$$

Let

$$\mathcal{S} = \{(jp, kq) : 1 \leq j \leq (q-1)/2; 1 \leq k \leq (p-1)/2\}.$$

The cardinality of $\mathcal{S}$ is $\frac{p-1}{2}\cdot\frac{q-1}{2}$. Also, it is an easy check to verify that $jp \neq kq$ for any $1 \leq j \leq (q-1)/2$, or $1 \leq k \leq (p-1)/2$. Furthermore, set

$$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2,$$

where

$$\mathcal{S}_1 = \{(jp, kq) \in \mathcal{S} : jp < kq\},$$

and

$$\mathcal{S}_2 = \{(jp, kq) \in \mathcal{S} : jp > kq\}.$$

If $(jp, kq) \in \mathcal{S}_1$, then $j < kq/p$. Also, $kq/p \leq (p-1)q/(2p) < q/2$. Therefore, $\lfloor kq/p \rfloor < q/2$, from which it follows that

$$\lfloor kq/p \rfloor \leq (q-1)/2.$$

Hence, the cardinality of $\mathcal{S}_1$ is $\sum_{k=1}^{(p-1)/2} \lfloor kq/p \rfloor$. Similarly, the cardinality of $\mathcal{S}_2$ is $\sum_{j=1}^{(q-1)/2} \lfloor jp/q \rfloor$. This establishes (1.11).

Now set $M = \sum_{k=1}^{(p-1)/2} \lfloor kq/p \rfloor$, and $N = \sum_{j=1}^{(q-1)/2} \lfloor jp/q \rfloor$. If we let $q = c$ in Corollary 1.4, then

$$\left(\frac{q}{p}\right) = (-1)^M.$$

Similarly,

$$\left(\frac{p}{q}\right) = (-1)^N.$$

Hence,

$$\left(\frac{q}{p}\right)\left(\frac{p}{q}\right) = (-1)^{M+N}.$$

The result now follows from (1.11). □

Example 1.24 Let $p = 7$ and $q = 991$. Then by the Quadratic Reciprocity Law,

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = \left(\frac{7}{991}\right)\left(\frac{991}{7}\right) = (-1)^{3\cdot 495} = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}} = -1,$$

so

$$\left(\frac{7}{991}\right) = -\left(\frac{991}{7}\right) = -\left(\frac{4}{7}\right) = -\left(\frac{2}{7}\right)^2 = -1.$$

Hence, $x^2 \equiv 7 \pmod{991}$ has no solutions $x \in \mathbb{Z}$.

Exercises 1.124–1.125 on page 66 are applications of the Quadratic Reciprocity Law. We conclude this section with a generalization of the Legendre symbol, which we will require later in the text.

Definition 1.19 **The Jacobi Symbol**

Let $n > 1$ be an odd natural number with $n = \prod_{j=1}^{k} p_j^{e_j}$ where $e_j \in \mathbb{N}$ and the p_j are distinct primes. Then the Jacobi Symbol *of a with respect to n is given by*

$$\left(\frac{a}{n}\right) = \prod_{j=1}^{k} \left(\frac{a}{p_j}\right)^{e_j},$$

for any $a \in \mathbb{Z}$, where the symbols on the right are Legendre Symbols.

Biography 1.20 Carl Gustav Jacob Jacobi (1804–1851) *was born in Potsdam in Prussia on December 10, 1804, to a wealthy German banking family. In August of 1825, Jacobi obtained his doctorate from the University of Berlin on an area involving partial fractions. The next year he became a lecturer at the University of Königsberg and was appointed as a professor there in 1831. Jacobi's first major work was his application of elliptic functions to number theory. Also, he made contributions to analysis, geometry, and mechanics. He died of smallpox on February 18, 1851.*

We will require the following result to establish properties of the Jacobi Symbol. This result is known as the *Supplement to the Quadratic Reciprocity Law*. Such supplements also exist for the higher reciprocity laws (see [62, pp. 273–332].

Proposition 1.8 *Let p be an odd prime. Then the following Legendre Symbol identity holds,*

$$\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}.$$

Proof. Let $M = \sum_{j=1}^{(p-1)/2} \lfloor jc/p \rfloor$ where $c \in \mathbb{Z}$ such that $p \nmid c$, and note that $\sum_{j=1}^{(p-1)/2} j = (p^2-1)/8$ by Theorem 1.4 on page 10.

From (1.10) in the proof of Corollary 1.4 on page 60,

$$(c-1)(p^2-1)/8 = p(M - \mathfrak{c}) + 2\sum_{j=1}^{\mathfrak{c}} r_j.$$

Therefore,

$$\mathfrak{c} \equiv M + \frac{(p^2-1)}{8}(c-1) \,(\text{mod } 2).$$

If $c = 2$, then $M = 0$, since $\lfloor 2j/p \rfloor = 0$ for all $j \in \mathbb{N}$ with $j < p/2$, so

$$\mathfrak{c} \equiv \frac{(p^2-1)}{8} \,(\text{mod } 2).$$

This establishes the result via Gauss's Lemma 1.2 on page 59. □

Corollary 1.5 *Let p be an odd prime. Then*

$$\left(\frac{2}{p}\right) = \begin{cases} 1 & \text{if } p \equiv \pm 1 \,(\text{mod } 8), \\ -1 & \text{if } p \equiv \pm 3 \,(\text{mod } 8). \end{cases}$$

Proof. By Proposition 1.8, 2 is a quadratic residue modulo p if and only if $(p^2-1)/8$ is even, namely whenever $p^2 \equiv 1 \,(\text{mod } 16)$. This occurs precisely when $p \equiv \pm 1 \,(\text{mod } 8)$. Similarly, 2 is a quadratic nonresidue modulo p when $(p^2-1)/8$ is odd, namely whenever $p^2 \equiv 9 \,(\text{mod } 16)$, and this occurs precisely when $p \equiv \pm 3 \,(\text{mod } 8)$. □

The Jacobi Symbol satisfies the following properties.

Theorem 1.24 **Properties of the Jacobi Symbol**

Let $m, n \in \mathbb{N}$, with n odd, and $a, b \in \mathbb{Z}$. Then

(1) $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$.

(2) $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$ *if* $a \equiv b \pmod{n}$.

(3) *If* m *is odd, then* $\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right)$.

(4) $\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}$.

(5) $\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}$.

(6) *If* $\gcd(a, n) = 1$ *where* $a \in \mathbb{N}$ *is odd, then*

$$\left(\frac{a}{n}\right)\left(\frac{n}{a}\right) = (-1)^{\frac{a-1}{2}\cdot\frac{n-1}{2}},$$

which is the Quadratic Reciprocity Law for the Jacobi Symbol.

Proof. Properties (1)–(2) follow from the results for the Legendre Symbol given in Theorem 1.22 on page 58. Property (3) is an easy consequence of Definition 1.19. For part (4), observe that if $n = \prod_{j=1}^{\ell} p_j$ where the p_j are (not necessarily distinct) primes, then

$$n = \prod_{j=1}^{\ell}(p_j - 1 + 1) \equiv 1 + \sum_{j=1}^{\ell}(p_j - 1) \pmod{4},$$

since all $p_j - 1$ are even. Thus,

$$\frac{n-1}{2} \equiv \sum_{j=1}^{\ell}(p_j - 1)/2 \pmod{2}. \tag{1.12}$$

For convenience's sake, we set $S = \sum_{j=1}^{\ell}(p_j - 1)/2$. Therefore, by part (3) proved above and part (1) of Theorem 1.22 in conjunction with (1.12),

$$\left(\frac{-1}{n}\right) = \prod_{j=1}^{\ell}\left(\frac{-1}{p_j}\right) = \prod_{j=1}^{\ell}(-1)^{(p_j-1)/2} = (-1)^S = (-1)^{(n-1)/2},$$

which is part (4). For part (5), first observe that

$$n^2 = \prod_{j=1}^{\ell} p_j^2 = \prod_{j=1}^{\ell}(p_j^2 - 1 + 1) \equiv 1 + \sum_{j=1}^{\ell}(p_j^2 - 1) \pmod{16},$$

since $p_j^2 \equiv 1 \pmod{8}$ for all such j. Therefore,

$$\frac{n^2-1}{8} \equiv \frac{\sum_{j=1}^{\ell}(p_j^2 - 1)}{8} \pmod{2},$$

and we set $T = \sum_{j=1}^{\ell}(p_j^2-1)/8$ for convenience. By Proposition 1.8 on page 63,

$$\left(\frac{2}{n}\right) = \prod_{j=1}^{\ell}\left(\frac{2}{p_j}\right) = \prod_{j=1}^{\ell}(-1)^{(p_j^2-1)/8} = (-1)^T = (-1)^{(n^2-1)/8},$$

which secures part (5). For part (6), let $a = \prod_{j=1}^{t} q_j$, where the q_j are (not necessarily distinct) primes. Since $\gcd(a,n) = 1$, then $p_j \neq q_k$ for any j,k. Thus, by properties (1) and (3), established above,

$$\left(\frac{a}{n}\right)\left(\frac{n}{a}\right) = \prod_{j=1}^{\ell}\left(\frac{a}{p_j}\right)\prod_{k=1}^{t}\left(\frac{n}{q_k}\right) =$$

$$\prod_{j=1}^{\ell}\prod_{k=1}^{t}\left(\frac{q_k}{p_j}\right)\prod_{k=1}^{t}\prod_{j=1}^{\ell}\left(\frac{p_j}{q_k}\right) = \prod_{j=1}^{\ell}\prod_{k=1}^{t}\left(\frac{p_j}{q_k}\right)\left(\frac{q_k}{p_j}\right),$$

and by Theorem 1.23, this equals

$$\prod_{j=1}^{\ell}\prod_{k=1}^{t}(-1)^{\frac{p_j-1}{2}\cdot\frac{q_k-1}{2}} = (-1)^U,$$

where

$$U = \sum_{j=1}^{\ell}\sum_{k=1}^{t}\frac{p_j-1}{2}\cdot\frac{q_k-1}{2} = \sum_{j=1}^{\ell}\frac{p_j-1}{2}\sum_{k=1}^{t}\frac{q_k-1}{2}.$$

However, as shown for the p_j in (1.12),

$$\sum_{k=1}^{t}\frac{q_k-1}{2} \equiv \frac{a-1}{2} \pmod 2,$$

so the result follows. □

Example 1.25 We have the following for $n = 15$,

$$\left(\frac{2}{15}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{5}\right) = (-1)(-1) = 1.$$

However, 2 is not a quadratic residue modulo 15. Thus, more caution must be exercised with the interpretation of the use of the Jacobi Symbol. See Exercises 1.126–1.127.

The use of the Jacobi Symbol for primality testing will become apparent as we travel through the text.

Exercises

1.124. Prove that if $p > 2$ is prime, then

$$\left(\frac{3}{p}\right) = \begin{cases} 1 & \text{if } p \equiv \pm 1 \,(\text{mod } 12), \\ -1 & \text{if } p \equiv \pm 5 \,(\text{mod } 12). \end{cases}$$

1.125. Verify the Legendre Symbol identity,

$$\sum_{j=1}^{p-1} \left(\frac{j}{p}\right) = 0,$$

where p is an odd prime by showing that there are $(p-1)/2$ quadratic residues and $(p-1)/2$ quadratic nonresidues modulo p. Then use this fact to establish the Legendre Symbol identity,

$$\sum_{j=0}^{p-1} \left(\frac{(j-a)(j-b)}{p}\right) = \begin{cases} p-1 & \text{if } a \equiv b \,(\text{mod } p), \\ -1 & \text{if } a \not\equiv b \,(\text{mod } p). \end{cases}$$

1.126. Let $a \in \mathbb{Z}$, and $n \in \mathbb{N}$ odd. Prove that if $\left(\frac{a}{p^t}\right) = 1$ for all primes p such that $p^t || n$ for some $t \in \mathbb{N}$, then a is a quadratic residue modulo n (see Footnote 1.6 on page 49).

1.127. Let $n \in \mathbb{N}$ be odd. Prove that $\left(\frac{m}{n}\right) = 1$ for all $m \in \mathbb{N}$ with $m < n$ such that $\gcd(m, n) = 1$ if and only if n is a perfect square.

1.128. Let $f(x) = ax^2 + bx + c$ where $a, b, c \in \mathbb{Z}$, and set $\Delta = b^2 - 4ac$. Suppose that $p > 2$ is a prime not dividing Δ. Prove that $\sum_{x=0}^{p-1} \left(\frac{f(x)}{p}\right) = -\left(\frac{a}{p}\right)$.

1.129. Let $f(x) = ax^2 + bx + c$ where $a, b, c \in \mathbb{Z}$, and set $\Delta = b^2 - 4ac$. Suppose that $p > 2$ is a prime such that $p|\Delta$. Prove that

$$\sum_{x=0}^{p-1} \left(\frac{f(x)}{p}\right) = (p-1)\left(\frac{a}{p}\right).$$

Sums of the form $\sum(\frac{f(x)}{p})$ *are called* Jacobsthal sums.

Biography 1.21 Ernst Jacobsthal (1882–1965) *was born in Berlin on October 16, 1882. He received his doctorate in Berlin in 1906 with his doctoral advisors being* Georg Frobenius *and* Issai Schur. *From 1913 he was a lecturer at the technical university of Berlin. In 1934, he emigrated to Norway where he took a position as a professor at the technical university of Trondheim. After the occupation of Norway, he fled in January of 1943 to Sweden. When the war ended, he returned to Trondheim to resume his position there. He died in Ueberlingen in 1965.*

1.8 Complexity

The amount of time required for the execution of an algorithm on a computer is measured in terms of *bit operations*, which are defined as follows: addition, subtraction, or multiplication of two binary digits; the division of a two-bit integer by a one-bit integer; or the shifting of a binary digit by one position. (The reader unfamiliar with computer arithmetic should consult Appendix B.) The number of bit operations necessary to complete the performance of an algorithm is called its *computational complexity* or simply its *complexity*. This method of estimating the amount of time taken to execute a calculation does not take into account such things as memory access or time to execute an instruction. However, these executions are very fast compared with a large number of bit operations, so we can safely ignore them. These comments are made more precise by the introduction of the following notation introduced by Edmund Landau.

Biography 1.22 Edmund Georg Hermann Landau (1877–1938) *was born on February 14, 1877, in Berlin, Germany. He attended the French Lycée in Berlin, then entered the University of Berlin to study mathematics at the age of sixteen. He received his doctorate in 1899 in number theory, having studied under Frobenius.* (*See Biography 1.21 on page 66.*) *In 1901, he submitted his Habilitation on analytic number theory. Then he taught at the University of Berlin from 1899 until 1909, when he was appointed to Göttingen as a successor to Minkowski (1864–1909). In 1909, he published the first systematic presentation of analytic number theory. In 1933, he was forced out of Göttingen by the National Socialist regime. After this he lectured only outside Germany. He died on February 19, 1938, in Berlin.*

Definition 1.20 **Big O Notation**

Suppose that f and g are positive real-valued functions. If there exists a positive real number c such that

$$f(x) < cg(x) \tag{1.13}$$

for all sufficiently large[1.7] *x, then we write*

$$f(x) = O(g(x)) \text{ or simply } f = O(g). \tag{1.14}$$

(*Mathematicians also write $f << g$ to denote $f = O(g)$ — see Biography 1.23 on page 68.*)

Big O is *the order of magnitude of the complexity*, an *upper bound* on the number of bit operations required for execution of an algorithm in the *worst-case scenario*, namely in the case where even the trickiest or the nastiest inputs

[1.7] Here sufficiently large means that there exists some bound $B \in \mathbb{R}^+$ such that $f(x) < cg(x)$ for all $x > B$. We just may not know explicitly the value of B. Often f is defined on $\mathbb{N}$ rather than $\mathbb{R}$ and occasionally over any subset of $\mathbb{R}$.

are given. It is possible that most often for a given algorithm even less time will be used, but we must always account for the worst-case scenario.

The comments made before Definition 1.20 may now be put into perspective. The definition of the time taken to perform a given algorithm does not take into consideration time spent *reading and writing* such as memory access, timings of instructions, even the speed or amount of memory of a computer, all of which are negligible in comparison with the order of magnitude complexity. The greatest merit of this method for estimating execution time is that it is machine independent. In other words, it does not rely upon the specifics of a given computer, so the order of magnitude complexity remains the same, irrespective of the computer being used. In the analysis of the complexity of an algorithm, we need not know *exactly* how long it takes (namely, the *exact* number of bit operations required to execute the algorithm), but rather it suffices to compare with other objects, and these comparisons need not be immediate but rather long-term. In other words, what Definition 1.20 says is that if f is $O(g)$, then *eventually* $f(x)$ is bounded by *some* constant multiple $cg(x)$ of $g(x)$. We do not know exactly *what* c happens to be or just *how big* x must be before (1.13) occurs. However, for reasons given above, it is enough to account for the efficiency of the given algorithm in the worst-case scenario.

Biography 1.23 *The notation* $<<$ *was introduced by* I.M. Vinogradov*, a Russian mathematician who proved in 1937 that every sufficiently large positive integer is the sum of at most four primes. This is related to Goldbach's Conjecture, which says that every even* $n \in \mathbb{N}$ *with* $n > 2$ *is a sum of two primes. The "=" in* (1.14) *should be considered as a* $<$ *and the "O" should be considered as a constant multiple. The equality is a means of saying that* f *is a member of the family satisfying* (1.13)*.*

Example 1.26 A simple illustration of the use of Big O is to determine the number of bits in a base b integer. If n is a t_n-bit base b integer, then

$$b^{t_n-1} \leq n < b^{t_n}.$$

Therefore, $t_n = \lfloor \log_b n \rfloor + 1$, so an estimate on the size of t_n is, in general, $t_n = O(\log_b n)$. Shortly, we will demonstrate that the base b of the logarithm is irrelevant in determining complexity.

Another simple illustration of the use of the Big O notation is to refer to Appendix B, where we introduced the algorithms for adding, subtracting, multiplying, and dividing two s-bit integers. Review of these algorithms shows us that addition or subtraction take $O(s)$ bit operations, which is also the number of bit operations required to compare them (determine which is larger, or whether they are equal). On the other hand, the multiplication of an s-bit integer with an t-bit integer requires $O(st)$ bit operations (see Exercise 1.130 on

page 78). By Exercise 1.140, division of an s-bit integer by an t-bit integer, with $s \leq t$, takes $O(st)$ bit operations.

If a number n has no more than s bits, then $n \leq 2^s$, so if we wish to describe complexity in terms of the numbers themselves rather than their respective bit sizes, then we can rephrase the above as follows. The addition, subtraction or comparison of two integers less than n takes $O(\log_2(n))$ bit operations, and the multiplication of two such integers takes $O(\log_2^2(n))$ bit operations, while division of n by $m \leq n$ takes $O(\log_2 m \log_2 n)$ bit operations.

The amount of time taken by a computer to perform a task is (essentially) *proportional* to the number of bit operations. In the simplest possible terms, the constant of proportionality, which is the number of nanoseconds per bit operation, depends upon the computer being used. (A nanosecond is $1/10^9$ of a second — a billionth of a second.) This accounts for the machine independence of the Big O method of estimating complexity since the constant of proportionality is of no consequence in the determination of Big O.

Some Basic Facts: (1) Recall that $\ln n$ means $\log_e n$, the logarithm to the base e, the *natural* or *canonical* base, where we often use $\exp(x)$ in place of e^x for convenience. In the mathematical literature, $\log x$ is often used for $\log_e x$. Also, recall that $\log_b x = \ln x / \ln b$.
(2) To say that a is proportional to b means that $a/b = c$, a constant, called the *constant of proportionality*. This relationship is often written as $a \propto b$ in the literature.
(3) Recall that a (nonconstant) polynomial is a function of the form $\sum_{i=0}^{n} a_i x^i$ for $n \in \mathbb{N}$, where the a_i are the coefficients (see page 311).

◆ Time Estimates

A fundamental *time estimate* in executing an algorithm is *polynomial time* (or simply *polynomial*). In other words, an algorithm is polynomial when its complexity is $O(n^c)$ for some constant $c \in \mathbb{R}^+$, where n is the bitlength of the input to the algorithm, and c is independent of n. (Observe that any polynomial of degree c is $O(n^c)$.) In general, these are the desirable algorithms, since they are the fastest. Therefore, roughly speaking, the polynomial time algorithms are the *good* or *efficient* algorithms. For instance, the algorithm is constant if $c = 0$; if $c = 1$, it is linear; if $c = 2$, it is quadratic; and so on. Examples of polynomial time algorithms are those for the ordinary arithmetic operations of addition, subtraction, multiplication, and division. On the other hand, those algorithms with complexity $O(c^{f(n)})$ where c is constant and f is a polynomial on $n \in \mathbb{N}$ are *exponential time algorithms* or simply *exponential*. A *subexponential* time algorithm is one for which the complexity for input $n \in \mathbb{N}$ is

$$O(\exp((c + o(1))(\ln n)^r (\ln \ln n)^{1-r})$$

where $r \in \mathbb{R}$ with $0 < r < 1$ and c is a constant, where $o(1)$ denotes a function $f(n)$ such that $\lim_{n\to\infty} f(n) = 0$. (In general, $f(n) = o(g(n))$ means that $\lim_{n\to\infty} f(n)/g(n) = 0$. Thus, $o(1)$ is used to symbolize a function whose limit

as n approaches infinity is 0.) Subexponential time algorithms are faster than exponential time algorithms but slower than polynomial time algorithms. These are, again roughly speaking, the *inefficient* algorithms. For instance, the method of trial division as a test for primality of $n \in \mathbb{N}$ uses $\sqrt{n}$ steps to prove that n is prime, if indeed it is. If we take the maximum bitlength $N = \log_2 n$ as input, then

$$\sqrt{n} = 2^{(\log_2 n)/2} = 2^{N/2},$$

which is exponential. Algorithms with complexity $O(c^{f(n)})$ where c is constant and $f(n)$ is more than constant but less than linear are called *superpolynomial*. It is generally accepted that modern-day cryptanalytic techniques for breaking known ciphers are of superpolynomial time complexity, but nobody has been able to prove that polynomial time algorithms for cryptanalyzing ciphers do not exist.

In calculating complexity using the Big O notation, the following properties are essential.

Theorem 1.25 **Properties of the Big O Notation**

Suppose that f, g are positive real-valued functions.

(a) *If $c \in \mathbb{R}^+$, then $cO(g) = O(g)$.*

(b) $O(\max\{f, g\}) = O(f) + O(g)$.

(c) $O(fg) = O(f)O(g)$.

Proof. (a) If $f = O(g)$, then there is a constant $k \in \mathbb{R}^+$ such that $f(x) < kg(x)$ for all sufficiently large x. Therefore,

$$cf(x) < (ck)g(x),$$

from which we get $cf = O(g)$. In other words,

$$O(g) = cO(g).$$

(b) Let $h_1 = O(f)$, and $h_2 = O(g)$, then there exist $c_1, c_2 \in \mathbb{R}^+$ such that $h_1(x) < c_1 f(x)$ and $h_2(x) < c_2 g(x)$ for sufficiently large x. Therefore,

$$h_1(x) + h_2(x) < \max\{c_1 f(x), c_2 g(x)\}.$$

Hence, $O(f) + O(g) = O(\max\{f, g\})$.

(c) Let $h_1 = O(f)$, and $h_2 = O(g)$, then $h_1(x) < c_1 f(x)$ and $h_2(x) < c_2 g(x)$, for some $c_1, c_2 \in \mathbb{R}^+$, and for sufficiently large x. Therefore,

$$h_1(x)h_2(x) < c_1 c_2 f(x)g(x),$$

for sufficiently large x. This implies that

$$O(f)O(g) = h_1 h_2 = O(fg),$$

which completes the proof. (*Note that part* (a) *is now a special case of part* (c) *with* $f = 1$. Also, note that if $f = g$, then this provides the induction step for the more general fact that $O(f^n) = O(f)^n$ for any $n \in \mathbb{N}$.) □

The following illustration involves the factorial notation (see Appendix A).

Example 1.27 Suppose that we wish to calculate the number of bit operations required to evaluate $n!$ for $n \in \mathbb{N}$ using only standard techniques. Assume that each natural number less than n has at most t bits. Then $n!$ has at most $n(t+1)$ bits and $n(t+1) = O(nt)$. Therefore, in the $n-1 = O(n)$ multiplications involved in computing $n!$, we multiply an integer with at most t bits by an integer with $O(nt)$ bits. This requires $O(nt^2)$ bit operations. Since we do this $O(n)$ times, the total number of bit operations required is $O(nt^2)O(n) = O(n^2t^2)$, by part (c) of Theorem 1.25. However, we know that $t = O(\log_2 n)$ from above, so the number of bit operations to compute $n!$ is $O(n^2t^2) = O(n^2 \log_2^2 n)$, by part (c) of the theorem again. This is exponential in the number of bits of n.

Note that Theorem 1.25 shows us that in the complexity analysis of such algorithms as the division of two integers discussed above, where division of n by $m \leq n$ takes $O(\log_2 m \log_2 n)$ bit operations, it is irrelevant which logarithm we use, since it does not change the Big O estimate. To see this we note that

$$O(\log_b(n)) = O(\ln n / \ln b) = O(\ln n)/\ln b = O(\ln n), \tag{1.15}$$

by Theorem 1.25. For this reason, we omit any subscripts on logarithms in Big O notation henceforth, unless specified otherwise for a given reason.

◆ Real-World Complexity of Algorithms

To get some idea of what the various classes of complexity analysis mean in "real-world" terms, let us look at times related to some of these classes. Suppose that the unit of time on the computer at our disposal is a microsecond (a millionth $(1/10^6)$ of a second). Assuming an input of $n = 10^6$ bits, then a constant algorithm (complexity $O(1)$) would take a microsecond to execute, since the number of bit operations is one. A linear algorithm (complexity $O(n)$) would take a second, since the number of bit operations is 10^6. A quadratic algorithm (complexity $O(n^2)$) would take $11.5741 = 10^{12}/(10^6 \cdot 24 \cdot 3600)$ days, since there are 10^{12} bit operations, and a cubic algorithm (complexity $O(n^3)$) would take $31,709 = 10^{18}/(10^6 \cdot 24 \cdot 3600 \cdot 365)$ years, since the number of bit operations is 10^{18}. By the time we get to exponential algorithms, we are looking at times astronomically larger than the age of the known universe. Hence, a problem is called *intractable* if no polynomial time algorithm could possibly solve it, whereas one that can be solved using a polynomial time algorithm is called *tractable*. (By a *problem*, we mean a general question to be answered. A *decision problem* is one whose solution is "yes" or "no." A problem may possess *parameters* whose values are left unspecified, and an *instance* of a problem is achieved by specifying values for those parameters.)

◆ Turing Machines

To understand how complexity theory divides problems into classes, we must imagine a theoretical computer, called a *Turing Machine* (see Biography 1.24 on page 73), which is a finite state machine having an infinite read-write tape. In other words, our theoretical computer has infinite memory and the ability to search for and retrieve any data from memory. *Church's Thesis* essentially says that the Turing Machine as a model of computation is equivalent to any other model for computation. (Here we may think of a "model" naively as a simplified mathematical description of a computer system.) Therefore, Turing Machines are realistic models for simulating the running of algorithms, and they provide a powerful computational model. However, a Turing Machine is not meant to be a practical design for any actual machine but rather is a sufficiently simple model to allow us to prove theorems about its computational capabilities while at the same time being sufficiently complex to include any digital computer irrespective of implementation.

◆ Complexity Problem Classes

Complexity theory designates a decision problem to be in class **P** if it can be solved in polynomial time, whereas a decision problem is said to be in class **NP** if it can be solved in polynomial time on a *nondeterministic* Turing Machine, which is a variant of the normal Turing Machine in that it *guesses* solutions to a given problem and checks its guess in polynomial time. Another way to look at the class **NP** is to think of these problems as those for which the *correctness of a guess* at an answer to a question can be proven in polynomial time. Those problems that can be *disproved* or for which a *no* answer can be guessed in polynomial time are said to be in $\mathbf{Co}-\mathbf{NP}$. In other words, $\mathbf{Co}-\mathbf{NP}$ consists of the problems whose complement is in **NP**. For example, the complement of "Is $n \in \mathbb{N}$ composite?" is "Is $n \in \mathbb{N}$ prime?". Note that the complement of problems in **P** are also in **P**. However, this is not known for **NP**. It is generally held that $\mathbf{NP} \neq \mathbf{Co}-\mathbf{NP}$, but this has not been proved.

The class **P** is a subset of the class **NP** since a problem that can be solved in polynomial time on a *deterministic* machine can also be solved, by eliminating the guessing stage, on a nondeterministic Turing Machine. It is an open problem in complexity theory to resolve whether or not $\mathbf{P} = \mathbf{NP}$. However, virtually everyone believes that they are unequal. It is generally held that most modern ciphers can be cryptanalyzed in nondeterministic polynomial time. However, in practice it is the deterministic polynomial time algorithm that is the end goal of modern-day cryptanalysis. Defining what it means to be a "computationally hard" problem is a *hard problem*. One may say that problems in **P** are *easy*, and those not in **P** are considered to be *hard*. (For instance, see [83, pp. 195–196].) However, there are problems that are regarded as computationally easy yet are not known to be in **P**. (For instance, the Miller-Rabin-Selfridge Test, which we will study in Section 3 of Chapter 4, is such a problem. It is in the class **RP**, called *randomized polynomial time* or *probabilistic polynomial time*. Here, $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$.) A practical (but mathematically less satisfying) way to

define "hard" problems is to view them as those which have continued to resist solutions after a concerted attack by competent investigators for a long time up to the present.

Biography 1.24 Alan Mathison Turing (1912–1954) *was born on June 23, 1912, in London, England. He studied under Alonzo Church (1903–1995) at Princeton and received his doctorate in 1938 for his thesis entitled Systems of Logic Based on Ordinals. During World War II, he worked in the British Foreign Office and was a major player in cryptanalyzing enemy codes. In 1945, he began work at the National Physical Laboratory in London, where he helped design the Automatic Computing Engine, which led the world at the time as a design for a modern computer. In 1948, Turing became the deputy director of the Computing Laboratory at Manchester, where the first running example of a computer using electronically stored programs was being built. His contributions include pioneering efforts in artificial intelligence. In 1952, he was arrested for violation of the British homosexuality statutes. His death from potassium cyanide poisoning occurred while he was doing experiments in electrolysis, and it is uncertain whether this was an accident or self-inflicted.*
For the reader interested in more detail and background, a (deterministic one-tape) Turing Machine has an infinitely long magnetic tape on which instructions can be written and erased. It also has a single bit register of memory and a processor that carries out the instructions: (1) *move the tape right,* (2) *move the tape left,* (3) *change the state of the register based upon its current value and a value on the tape, and write or erase on the tape. The Turing Machine runs until it reaches a desired state causing it to halt. A famous problem in theoretical computer science is to determine when a Turing Machine will halt for a given set of input and rules. This is called the Halting Problem. Turing proved that this problem is undecidable, meaning that it is neither formally provable nor unprovable.*

Another aspect of problem classification in complexity theory is the **NP**-complete problem, which is a problem in the class **NP** that can be proved to be as difficult as any problem in the class. Should an **NP**-complete problem be discovered to have a deterministic polynomial time algorithm for its solution, this would prove that $\mathbf{NP} \subseteq \mathbf{P}$, so $\mathbf{P} = \mathbf{NP}$. Hence, we are in the position that there is no proof that there are *any* hard problems in this sense, namely those in **NP** but not in **P**. Nevertheless, this has not prevented the flourishing of research into complexity theory. (The classical **NP**-complete problem is the *Travelling Salesman Problem*: A travelling salesman wants to visit $n \in \mathbb{N}$ cities. Is there a round trip that he can map out that allows him to visit each city exactly once? This has been shown to be equivalent to the *Knapsack Problem*, which we will study in Section 4 of Chapter 3.) There are other distinctions up to and including the set **EXPTIME** of problems that can be solved in exponential time. However, the focus of this book will not need to address the finer distinctions.

◆ Applications of Complexity Theory

We now look at some applications of complexity theory and the use of the Big O symbol (see Biography 1.25).

Theorem 1.26 **Lamé**

If $a, b \in \mathbb{N}$ such that $a > b$, and it takes $n+1$ iterations (divisions) to find $\gcd(a,b)$ using the Euclidean Algorithm, then $n < \log_{\mathfrak{g}} b$, where $\mathfrak{g} = (1+\sqrt{5})/2$ is the golden ratio (initially defined on page 11).

Proof. If $a = r_{-1}$, $b = r_0$, then by Euclid's Algorithm,

$$r_{j-1} = r_j q_{j+1} + r_{j+1} \quad (0 < r_{j+1} < r_j)$$

for all nonnegative integers $j < n$, where n is the smallest value such that $r_{n+1} = 0$ (namely, we have $n+1$ iterations). Therefore, if F_j denotes the j^{th} Fibonacci number, then

$$r_n \geq 1 = F_2$$

$$r_{n-1} = r_n q_{n+1} \geq 2 = F_3$$

$$r_{n-2} \geq r_{n-1} + r_n \geq F_3 + F_2 = F_4$$

$$\vdots$$

$$b = r_0 \geq r_1 + r_2 \geq F_{n+1} + F_n = F_{n+2}$$

Therefore, $b \geq F_{n+2}$. By Theorem 1.6 on page 11,

$$F_{n+2} \geq \mathfrak{g}^n,$$

so

$$b \geq \mathfrak{g}^n.$$

Hence,

$$n < \log_{\mathfrak{g}} b,$$

so we have established the theorem of Lamé. □

Biography 1.25 Gabriel Lamé (1795–1870) *was born on July 22, 1795, in Tours, France. He both studied at École Polytechnique and was later a professor there. His primary contributions were to mathematical physics, but he also worked in differential geometry, diffusion in crystalline material, and elasticity. In fact, two elastic constants are named after him. His contributions to number theory were this result, and his proof of Fermat's Last Theorem for the exponent $n = 7$, which he gave in 1839. He died on May 1, 1870, in Paris, where there is now a street named after him.*

Corollary 1.6 *With the hypothesis of Lamé's Theorem holding,*

$$n < 5 \log_{10} b.$$

Proof. An easy check shows that $\log_{10} \mathfrak{g} > 1/5$, so by Lamé's Theorem,

$$\frac{n}{5} < \log_{\mathfrak{g}} b \log_{10} \mathfrak{g} = \log_{10} b.$$

(For the equality, see the basic facts box on page 69.) Hence, $n < 5 \log_{10} b$. □

In common language, what Corollary 1.6 says is the following.

The number of iterations required to find $\gcd(a, b)$ is at most five times the number of decimal digits in the smaller value b.

To see this, suppose that b has s decimal digits, so $b < 10^s$, namely $\log_{10} b < s$. Therefore, $5s > n$ by Corollary 1.6, so $5s \geq n + 1$, which is the number of iterations required to find the $\gcd(a, b)$. What is implicit in this discussion is that the complexity of the given algorithm depends on b and not on a. Moreover, in computing the gcd of two consecutive Fibonacci numbers, the upper bound on complexity is reached. In other words, the worst-case scenario does indeed occur.

Now we use Lamé's Theorem to establish the following.

Theorem 1.27 **Computational Complexity of the GCD**

If $a, b \in \mathbb{N}$ such that $a > b$, then the number of bit operations required to find $\gcd(a, b)$ *using Euclid's Algorithm, is* $O(\ln^3 a)$.[1.8]

Proof. From the proof of Theorem 1.26, $r_j \geq F_{n-j+2}$. In particular, we have $b \geq F_{n+2}$ and $a \geq F_{n+3}$. Since

$$F_{n+3} \geq \mathfrak{g}^{n+1},$$

then

$$\log_{\mathfrak{g}} a \geq n + 1.$$

By the Euclidean Algorithm, Theorem 1.2, the number of divisions required to find $\gcd(a, b)$ is $n + 1$, and

$$n + 1 = O(\ln a)$$

by (1.15) on page 71. This is the number of bit operations required to perform the $n + 1$ divisions. Since each iteration of the algorithm computes a quotient

[1.8] This can be improved by a more refined analysis to show that the *running time* (defined as the number of bit operations executed for a given input) of the Euclidean Algorithm is $O(\ln^2 a)$. See [58, 2.105 Fact, p. 66].

and remainder involving numbers no bigger than a, then each iteration can be done in time $O(\ln^2 a)$. Therefore, by part (c) of Theorem 1.25, $\gcd(a, b)$ may be found using $O(\ln^3 a)$ bit operations. □

The following is a variant of the Euclidean Algorithm.

◆ The Least Remainder Algorithm

Let $a, b \in \mathbb{Z}$ with $a \geq b > 0$ and set $a = s_{-1}$, $b = s_0$. As with the Euclidean Algorithm, we repeatedly apply the Division Algorithm according to the recursive formula for each $j \geq 0$:

$$|s_{j-1}| = |s_j|t_{j+1} + s_{j+1} \text{ where } -|s_j|/2 < s_{j+1} \leq |s_j|/2.$$

Note that an analogue of the Division Algorithm (Theorem 1.1) holds for the least remainder algorithm. This guarantees the existence and uniqueness of the s_j and t_j.

Example 1.28 If $1001 = s_{-1}$ and $s_0 = 221$, then to apply the Least Remainder Algorithm we proceed as follows.

$$s_{-1} = 1001 = 221 \cdot 5 - 104 = s_0 t_1 + s_1,$$

$$s_0 = 221 = 104 \cdot 2 + 13 = |s_1|t_2 + s_2,$$

and $|s_1| = 104 = 13 \cdot 8 = s_2 t_3$, so $\gcd(1001, 221) = 13$.

If we now compare this with the ordinary Euclidean Algorithm, we get,

$$r_{-1} = 1001 = 221 \cdot 4 + 117 = r_0 q_1 + r_1,$$

$$r_0 = 221 = 117 \cdot 1 + 104 = r_1 q_2 + r_2,$$

$$r_1 = 117 = 104 \cdot 1 + 13 = r_2 q_3 + r_3,$$

and $r_2 = 104 = 13 \cdot 8 + 0 = r_3 q_4 + r_4$, so $\gcd(1001, 221) = 13 = r_3 = r_{n-1}$.

In general, the Euclidean Algorithm is less efficient than the Least Remainder Algorithm. It can be shown that we *save* approximately

$$1 - \log_2 \mathfrak{g} \approx 0.306$$

of the division steps by using the Least Remainder Algorithm over the Euclidean Algorithm (see [46, Exercise 30, p. 376]). The Least Remainder Algorithm allows for negative remainders, the *least* in absolute value, which accounts for its increased efficiency.

This completes the introductory material and thus we conclude Chapter 1 with some general remarks concerning complexity theory.

◆ Summary of Complexity Theory

Roughly speaking, complexity theory can be subdivided into two categories: (a) structural complexity theory, and (b) the design and analysis of algorithms. Essentially, category (a) is concerned with lower bounds, and category (b) deals with upper bounds. Basically, the primary goal of structural complexity theory is to classify problems into classes determined by their intrinsic computational difficulty. In other words, how much computing time (and resources) does it take to solve a given problem? As we have seen in this section, the fundamental question in structural complexity theory remains unanswered, namely does $\mathbf{P} = \mathbf{NP}$? In this section, we have been primarily concerned with the analysis of algorithms, which is of the most practical importance to cryptography.

The foundations of complexity theory were laid by the work done starting in the 1930's by Turing and Church, among others (see Biography 1.24 on page 73). As we have seen in this section, the first goal was to formalize the notion of a computer (or realistic model thereof such as the Turing Machine). Then the goal was whether such devices could solve various mathematical problems. One of the outcomes of this research, again as we have seen, is that there are problems that cannot be solved by a computer. This dashed the program, set out by Hilbert (see Biography 1.26) at the turn of the twentieth century, which sought to show that all mathematical problems could, at least in principle, be answered in some deterministic or mechanical way.

Although the design of better and more efficient algorithms has been a goal of mathematicians and scientists in general for some time, it was not until the late 1960's that complexity theory began to be recognized as a formal discipline. The establishment of the theory may be credited to the pioneering work of Stephen Cook, Richard Karp, Donald Knuth, and Michael Rabin. Each of these individuals has since been awarded the highest honour in computer science research — the Turing Award.

Biography 1.26 David Hilbert (1862–1943) *was born in Königsberg, Prussia (now Kaliningrad, Russia). In 1895, Hilbert was appointed to the chair of mathematics at the University of Göttingen where he remained until his retirement in 1930. Among his students were* Hermann Weyl (1885–1955) *and* E.F.E. Zermelo (1871–1953). *Hilbert's contributions to twentieth-century mathematics were deep indeed. Perhaps this is best epitomized in the speech he delivered to the Second International Congress of Mathematicians held in 1900 in Paris, where he presented his now-famous list of twenty-three problems, many of which remain unsolved. Among these problems was the aforementioned one, namely that a finite number of logical steps based upon the axioms of arithmetic can never lead to contradictory results. However, the work of a mathematician named* Kurt Gödel (1906–1978) *destroyed any hope of that in 1931, when he proved that, given the axioms of arithmetic, statements can be made that can neither be proved nor disproved, namely they are undecidable. Hilbert died on February 14, 1943. See* [61, p. 290] *and* [62, p. 10] *for more information.*

Exercises

1.130. Prove that the multiplication of an m-bit integer with an n-bit integer, using the algorithms of Section 3, takes $O(mn)$ bit operations.

1.131. Prove that for any $n \in \mathbb{N}$,

$$n! = O(n^n).$$

1.132. Show that if

$$f(x) = \sum_{j=0}^{n} a_j x^j,$$

where $a_j \in \mathbb{N}$ for each $j \geq 0$, then

$$O(f) = O(x^n).$$

1.133. Find a function $f(b, n)$, for $b, n \in \mathbb{N}$, such that $O(f)$ is the largest n-digit number to base b.

1.134. Estimate the number of bit operations required to compute $\sum_{j=1}^{n} j^2$.

1.135. Prove that for all $r \in \mathbb{R}^+$,

$$\ln(n) = O(n^r).$$

1.136. Find a counterexample to the following assertion.
If $f(x) = O(g(x))$, then $e^{f(x)} = O(e^{g(x)})$.

1.137. Prove that if $f(x) = O(g(x))$, then $\ln(f(x)) = O(\ln(g(x))$.

1.138. Find a counterexample to the assertion:
If $f(x) = O(g(x))$, then $h(f(x)) = O(h(g(x))$.

1.139. Given $n \in \mathbb{N}$, prove that the number of bit operations required to compute $n!$ is $O(n^2 \ln^2(n))$.

1.140. Prove that the number of bit operations required to divide n by m, using the algorithm given on page 333 of Appendix B on basic computer arithmetic, is $O(mn)$.

1.141. Prove that if $f(x) = O(g(x))$ where $g(x)$ is a polynomial, then $f^n(x) = O(g^n(x))$.

1.142. Prove that for any $n \in \mathbb{N}$,

$$n \ln(n) = O(\ln(n!)).$$

Chapter 2

Cryptographic Basics

2.1 Definitions and Illustrations

◆ Cryptography

First we will settle upon the meaning of *cryptography*, which is the study of methods for sending messages in *secret* (namely, in *enciphered* or *disguised* form) so that only the intended recipient can remove the disguise and read the message (or *decipher* it). Cryptography has, as its etymology, *kryptos* from the Greek, meaning *hidden*, and *graphein*, meaning *to write*. The original message is called the *plaintext*, and the disguised message is called the *ciphertext*. The final message, encapsulated and sent, is called a *cryptogram*. The process of transforming plaintext into ciphertext is called *encryption* or *enciphering*. The reverse process of turning ciphertext into plaintext, which is accomplished by the recipient who has the knowledge to remove the disguise, is called *decryption* or *deciphering*. Anyone who engages in cryptography is called a *cryptographer*. On the other hand, the study of mathematical techniques for attempting to defeat cryptographic methods is called *cryptanalysis*. Those practicing cryptanalysis (usually termed the "enemy") are called *cryptanalysts*. The term *cryptology* (see Biography 2.1 on page 80) is used to embody the study of both cryptography and cryptanalysis, and the practitioners of cryptology are *cryptologists*. The etymology of cryptology is the greek *kryptos* meaning *hidden* and *logos* meaning *word*. Also, the term *cipher* (which we will use interchangeably with the term *cryptosystem*) is a method for enciphering and deciphering. Later in this section we will mathematically formalize the notion of a cryptosystem, but this will suffice for now.

◆ Steganography

Cryptography may be viewed as *overt secret writing* in the sense that the writing is clearly seen to be disguised. This is different from *steganography*, which conceals the very *existence* of the message, namely *covert secret writing*. (The etymology is *steganos* from the Greek meaning *impenetrable*.) For

instance, *invisible ink* would be called a *technical* steganographic method as would suitcases with false bottoms containing a secret message. One of the most famous such methods, employed by the Germans during World War II, was the use of the microdot (invented by Emanuel Goldberg in the 1920's) as a period in typewritten documents. An example of *linguistic* steganography, the other branch of steganography, is the use of two typefaces to convey a secret message, a technique used by Francis Bacon (see Biography 2.2 on page 81), which is described in his publication of 1623: *De Augmentis Scientarium*. Today, hiding (subliminal) messages in television commercials would also qualify. Generally speaking, steganography involves the hiding of secret messages in other messages or devices. The modern convention is to break cryptography into two parts: *cryptography proper* or *overt secret writing*, and *steganography* or *covert secret writing*. The term *steganography* first appeared in the work *Steganographia*, by Johannes Trithemius (see Biography 2.3 on page 82). (See [45] for more details on Steganography.)

Biography 2.1 *The* (English) *term* cryptography *was coined in 1658 by* Thomas Browne*, a British physician and writer. The term* cryptology *was coined by* James Howell *in 1645. However,* John Wilkins (1614–1672) *in his book* Mercury, or the Secret and Swift Messenger*, introduced into the English language the terms* cryptologia *or* secrecy in speech*, and* cryptographia *or* secrecy in writing*. Wilkins, who was a cofounder of the Royal Society along with* John Wallis (1616–1703) *later married Oliver Cromwell's sister and became Bishop of Chester. The modern incarnation of the use of the word* cryptology *is probably due to the advent of Kahn's encyclopedic book* [43], The Codebreakers *published in 1967, after which the term became accepted and established as that area of study embracing both cryptography and cryptanalysis.*

◆ Codes

Some comments on the term "codes" are appropriate at this juncture. Throughout history the term "code" has become blurred with that of "cipher" and has come often to mean any kind of disguised secret. However, today the word "code" has a very specific meaning in various contexts. If we have a column of plaintext symbols next to a corresponding column of ciphertext symbols, that is an example of a *code-book* since you can look up the "code" and find the plaintext next to it. It is this usage that usually is reserved for the term "code" — a dictionary-like listing of plaintext and corresponding ciphertext. A *cryptographic code* means the replacement of linguistic groups (such as groups of words, or phrases) with numbers, designated words, or phrases, called *codegroups*. This is the meaning that we shall use throughout. Moreover, today there are *error-correcting codes*, which have nothing to do with secrecy. These codes rather refer to the removal of "noise" from, say, a telephone line or satellite signal; namely, these codes provide a means of fixing portions of a message that were corrupted during transmission. The codes with which we are concerned here are the ones defined above, which are *cryptographic* codes,

since they have to do with secrecy.

◆ Substitution and Transposition Ciphers

A "substitution" cipher replaces plaintext symbols with other symbols to produce ciphertext. As a simple example, the plaintext might be *palace*, and the ciphertext might be *QZYZXW* when *a,c,e,l,p* are replaced by *Z,X,W,Y,Q*, respectively. (The cryptographic convention is to use *lower-case* letters for *plaintext* and *UPPER-CASE* letters for *CIPHERTEXT*.) With a transposition cipher (see Biography 2.4 on page 82), we permute the *places* where the plaintext letters sit. What this means is that we do not change the letters but rather move them around, transpose them, without introducing any *new* letters. Here is a simple illustration. Suppose that we have thirteen letters in our plaintext, and the following is a permutation that tells us how to move the thirteen positions around. The way to read the following is that the symbol in the position number in the top row gets replaced by the symbol in the position number below it in the second row.

Biography 2.2 Francis Bacon (1561–1626) *was born on January 22, 1561, in London, England. He was educated at Trinity College in Cambridge and ultimately became a lawyer in 1582. After a brief unsuccessful stint in politics, he became Queen Elizabeth's counsel. In that capacity, he helped convict Robert Devereux (1566–1601) the second earl of Essex, for treason, after which Essex was executed. After James I assumed the throne, Bacon again went the political route. Then after a sequence of subsequent legal positions, he was appointed lord chancellor and Baron Verulam in 1618, and by 1621 he was made Viscount St. Albans. In the intervening years 1608–1620, he wrote numerous philosophical works and wrote several versions of his best-known scientific work,* Novum Organum. *In 1621, Bacon was accused of bribery and fell from power. He spent his final years writing his most valuable and respected works. He died on April 9, 1626, in London.*

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 1 & 2 & 3 & 4 & 10 & 7 & 8 & 9 & 5 & 6 & 11 & 12 & 13 \end{pmatrix}$$

Now, suppose that our plaintext is *they flung hags*. Then the ciphertext will be *THEY HUNG FLAGS*. Notice that the first four and last three plaintext letters remain in the same position as dictated by the above permutation, but the f in position 5 gets replaced by the H in position 10, the l in position 6 gets replaced by the U in position 7, the u in position 7 gets replaced by the N in position 8, the n in position 8 gets replaced by the G in position 9, the g in position 9 gets replaced by the F in position 5, and the h in position 10 gets replaced by the L in position 6. So this is an easy-to-understand method of depicting transposition ciphers that we will use throughout the book. We can see that transposition ciphers depend upon the permutation given, such as the one above, so often transposition ciphers are called *permutation ciphers*. In Section 3.1 we will give a more formal mathematical description of substitution

and transposition ciphers, which will be necessary for a description of the DES cryptosystem. However, the above description is mathematically sufficient for our purposes now.

Biography 2.3 Steganographia *was written by* Johannes Trithemius (1462–1516) *in 1499. Trithemius's manuscript was circulated for over a century and not published until 1606. In 1609, the Roman Catholic Church placed it on its index of Prohibited Books, where it remained for over two centuries. Nevertheless, it was reprinted numerous times, including as late as 1721. During Trithemius's lifetime, his* Steganographia *caused him to be known as a sorcerer, which did not sit well since he was an abbot at the abbey of Saint Martin at Spanheim, Germany. In fact, his fellow monks were so incensed that Trithemius was transferred to the monastery of Saint Jacob in Wurzberg, where he remained (writing and studying) until his death on December 15, 1516.*

◆ Julius Caesar

Although the ancient Greeks made no claim to actually using any of the substitution ciphers that they invented, the first use in both military and domestic affairs of such a cipher is well documented by the Romans. In *The Lives of the Twelve Caesars* [88, p. 45], Suetonius writes of Julius Caesar: ".... if there was occasion for secrecy, he wrote in cyphers; that is, he used the alphabet in such a manner, that not a single word could be made out. The way to decipher those epistles was to substitute the fourth for the first letter, as d for a, and so for the other letters respectively." What is being described here is a simple substitution cipher used by Julius Caesar. He used them not only in his domestic affairs as noted above by Seutonius but also in his military affairs as he documented in his own writing of the *Gallic Wars*.

Biography 2.4 *The first to use* military *cryptography for correspondence were the* Spartans*, the great warriors of the Greek states. The Spartans used a transposition cipher device called a* skytale (*also spelled* scytale *in some sources*)*. This consisted of a tapered wooden staff around which a strip of parchment (leather or papyrus were also used) was spirally wrapped, layer upon layer. The secret message was written on the parchment lengthwise down the staff. Then the parchment was unwrapped and sent. By themselves, the letters on the parchment were disconnected and made no sense until rewrapped around a staff of equal proportions, at which time the letters would realign to once again make sense. One use of the skytale was documented to have occurred around 475 B.C. with the recalling of General Pausanius, who was a Spartan prince. He was attempting to make alliances with the Persians, an act the Spartans regarded as treasonous. Over one hundred years later, a skytale was used to recall General Lysander to face charges of sedition. Thus, the Greeks have been credited with the first use of a device employing a transposition cipher.*

Caesar's substitution cipher is even easier to use than that invented by Polybius, which we describe in Exercise 2.3 on pages 88–89. With Caesar's cipher there is merely a shift to the right of three places of each plaintext letter to achieve the ciphertext letters. This is best illustrated by Table 2.1.

Table 2.1

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

Table 2.1 is an example of a *cipher table*, which is defined to be a table of (ordered) pairs of symbols (p, c), where p is a plaintext symbol and c is its ciphertext equivalent. For instance, in the Caesar cipher table, (b, E) is the pair consisting of the plaintext letter b together with its ciphertext equivalent E. An example of a cryptogram made with the Caesar cipher is: *brutus* becomes *EUXWXV*. Also, this simple type of substitution cipher is called a *shift cipher*. Moreover, the mechanism for enciphering in the Caesar cipher is a shift to the right of three letters. So the value 3 is an example of a *key*, which we may regard, in general, as a *shared secret* between the sender and the recipient, which *unlocks* the cipher. So 3, in this case, is the *enciphering key*. Since shifting three units left unlocks the cipher, then 3 is also the *deciphering key*. This is an example of a *symmetric-key cryptosystem*, namely, where one can "easily determine" the deciphering key from the enciphering key and vice versa. (We will formalize these notions later in this section, but for now, these are adequate descriptions.) Thus, the key must be kept secret from all unauthorized parties. (This is distinct from a cryptosystem, about which we will learn later, where the enciphering key can be made publicly known! Yet, nobody can determine the deciphering key from it.) There is a method of employing the Caesar cipher with numbers that simplifies the process. Consider Table 2.2 that gives numerical values to the English alphabet.

Table 2.2

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

Now, if we take *zebra* as the plaintext, the numerical equivalent is $25, 4, 1, 17, 0$, and using the Caesar cipher we add 3 to each number to get the ciphertext. However, notice that when we get to *x, y, z*, adding 3 will take us beyond the highest value of 25. The Caesar cipher, Table 2.1, actually loops these three letters back to *A, B, C*. Hence, what we have to do here is to throw away any multiples of 26 and treat them as zeroes in our addition, namely we are performing addition modulo 26 (see Section 1.3 for an introduction to congruences). Thus, the plaintext numerical equivalents $25, 4, 1, 17, 0$ become $2, 7, 4, 20, 3$, and

using Table 2.2, the ciphertext message becomes *CHEUD*. Once sent, the recipient uses the key 3 to decipher by first converting the ciphertext to letters via Table 2.2, then calculating, for instance $2 - 3 \equiv 25 \pmod{26}$. Similarly, all other numbers are decrypted to yield $25, 4, 1, 17, 0$, which, via Table 2.1 becomes *zebra*.

◆ Messages and Tranformations

In order to discuss even the most basic ciphers in depth, we need to use a rigorous mathematical language in which to carry on this discussion. Earlier in this section, we defined certain basic notions such as the plaintext and ciphertext. Both the plaintext and the ciphertext are written in terms of elements from a finite set $\mathcal{A}$, called an *alphabet of definition*. The alphabet of definition may consist of numbers, letters from an alphabet such as the English, Greek, or Russian alphabets, or symbols such as !, @, *, or any other symbols that we choose to use when sending messages. The alphabet of definition for the plaintext and ciphertext may differ, but the usual convention is to use the same for both. For instance, a commonly used one is $\mathcal{A} = \{0, 1\}$, called the *binary alphabet* of definition, in terms of which any given alphabet may be given binary equivalents. An example is the English alphabet, in which each letter may be assigned a unique binary string of length five since there are $2^5 = 32$ binary strings of length five. Another example is $\mathcal{A} = \{0, 1, 2\}$, called the ternary alphabet, in which each letter of the English alphabet may be replaced by a unique ternary string of length three since there exist $3^3 = 27$ ternary strings of length three (see page 326). Once we have agreed upon an alphabet of definition, we choose a *message space*, $\mathcal{M}$, which is defined to be a finite set consisting of strings of symbols from the alphabet of definition. Elements of $\mathcal{M}$, which may be anything from binary strings to English text, are called *plaintext message units*. Any block of $n \in \mathbb{N}$ letters may be used. A finite set $\mathcal{C}$, consisting of strings of symbols from an alphabet of definition for the ciphertext, is called the *ciphertext space*, and elements from $\mathcal{C}$ are called *ciphertext message units*. Most often it is convenient to let $\mathcal{M}$ be the message space consisting of *all possible* plaintext message units and $\mathcal{C}$ be the set of *all possible* ciphertext message units. It is within this context that we will choose to work below.

To make cryptanalysis more difficult, we need a set of parameters $\mathcal{K}$, called the *keyspace*, whose elements are called *keys*. For instance, we learned above about the Caesar cipher. In terms of the above definitions, we may restate the cipher as follows. Given the alphabet of definition as the numbers 0 through 25, corresponding to the letters A through Z, respectively, any $m \in \mathcal{M}$ is enciphered as $c \in \mathcal{C}$, where

$$c = m + 3 \in \mathbb{Z}/26\mathbb{Z}.$$

Thus, the (enciphering) key is $k = 3 \in \mathcal{K}$, since we are using the parameter 3 as the shift from $m \in \mathcal{M}$ to achieve $c \in \mathcal{C}$. Also, the (deciphering) key is also the parameter 3 since we achieve $m \in \mathcal{M}$ from $c \in \mathcal{C}$ by

$$c - 3 = m \in \mathbb{Z}/26\mathbb{Z}.$$

We formalize the above in the following.

Definition 2.1 **Enciphering and Deciphering Transformations**

An enciphering transformation (*also called an* enciphering function) *is a bijective function*

$$E_e : \mathcal{M} \mapsto \mathcal{C},$$

where the key $e \in \mathcal{K}$ *uniquely determines* E_e *acting upon plaintext message units* $m \in \mathcal{M}$ *to get ciphertext message units*

$$E_e(m) = c \in \mathcal{C}.$$

A deciphering transformation (*or* deciphering function) *is a bijective function*

$$D_d : \mathcal{C} \mapsto \mathcal{M},$$

which is uniquely determined by a given key $d \in \mathcal{K}$*, acting upon ciphertext message units* $c \in \mathcal{C}$ *to get plaintext message units*

$$D_d(c) = m.$$

The application of E_e *to* m*, namely the operation* $E_e(m)$*, is called* enciphering*,* encoding*, or* encrypting $m \in \mathcal{M}$*, whereas the application of* D_d *to* c *is called* deciphering*,* decoding*, or* decrypting $c \in \mathcal{C}$*.*

In Definition 2.1 we have mathematically formalized the two notions of enciphering and deciphering, which we informally discussed earlier in this section, as a motivator for this formal setup. For instance, returning to the Caesar cipher, it may be defined as that transformation E_e uniquely determined by the key e, which is addition of 3 modulo 26. Thus,

$$E_e(m) = c \equiv m + 3 \,(\text{mod } 26),$$

or simply

$$E_e(m) = c = m + 3 \in \mathcal{C} = \mathbb{Z}/26\mathbb{Z}.$$

Also, $m \in \mathcal{M} = \mathbb{Z}/26\mathbb{Z}$ is the numerical equivalent of the plaintext letter as described above. Similarly, $D_d(c)$ is that deciphering transformation uniquely defined by the key d, which is modular subtraction of 3 modulo 26. In other words,

$$D_d(c) = m \equiv c - 3 \,(\text{mod } 26),$$

or simply

$$D_d(c) = m = c - 3 \in \mathbb{Z}/26\mathbb{Z},$$

and $c \in \mathcal{C} = \mathbb{Z}/26\mathbb{Z}$ is the numerical equivalent of the ciphertext letter. Notice that

$$D_d(E_e(m)) = m$$

for each $m \in \mathcal{M}$. In other words, $D_d = E_e^{-1}$ is the inverse function of E_e. This is formalized as follows.

Definition 2.2 **Cryptosystems/Ciphers**

A cryptosystem *is composed of a set*

$$\{E_e : e \in \mathcal{K}\}$$

consisting of enciphering transformations and the corresponding set

$$\{E_e^{-1} : e \in \mathcal{K}\} = \{D_d : d \in \mathcal{K}\}$$

of deciphering transformations. In other words, for each $e \in \mathcal{K}$*, there exists a*

unique $d \in \mathcal{K}$ *such that* $D_d = E_e^{-1}$*, so that* $D_d(E_e(m)) = m$ *for all* $m \in \mathcal{M}$*.*

The keys (e, d) *are called a* key pair *where possibly* $e = d$*. A cryptosystem is also called a* cipher*. We reserve the term* Cipher Table *for the pairs of plaintext symbols and their ciphertext equivalents*

$$\{(m, E_e(m)) : m \in \mathcal{M}\}.$$

Definition 2.2 mathematically formalizes the notions of the terms *cryptosystem/cipher*, *cipher table*, and *key*, which were informally discussed earlier in this section. The case where $e = d$ or where one of them may be "easily" determined from the other in the key pair has a special name, which is the simplest of the possibilities for cryptosystems, and so has the longest history.

Definition 2.3 **Symmetric-Key Ciphers**

A cryptosystem is called symmetric-key (*also called single-key,* one-key*, and* conventional) *if for each key pair* (e, d)*, the key* d *is "computationally easy" to determine knowing only* e *and similarly to determine* e *knowing only* d*.* [2.1]

Usually $e = d$ *with practical symmetric-key ciphers, thereby justifying the use of the term symmetric-key.*

There are two kinds of symmetric-key cryptosystems about which we will learn in Section 2.2.

[2.1] We will use the term "computationally easy problem" to mean one that can be solved in expected (in the probability sense) polynomial time and can be attacked using available resources. (The reason for adding the latter caveat is to preclude problems that are of polynomial time complexity but for which the degree is "large.") The antithesis of this would be a *computationally infeasible problem*, which means that, given the enormous amount of computer time that would be required to solve the problem, this task cannot be carried out in *realistic* computational time. Thus, "computationally infeasible" means that, although there (theoretically) exists a unique answer to our problem, we cannot find it even if we devoted every scintilla of the time and resources available. This is distinct from a problem that is unsolvable in any amount of time or resources. For example, an unsolvable problem would be to cryptanalyze XYZ assuming that it was enciphered using a monoalphabetic substitution. There is simply no unique verifiable answer without more information. However, it should be stressed here that there is no proved example of a computationally infeasible problem.

We close this section with the following distinction between ciphers based on the number of cipher alphabets.

◆ Monoalphabetic and Polyalphabetic Ciphers

A *homophone* is a ciphertext symbol that always represents the same plaintext symbol. For instance, with the Caesar cipher in Table 2.1 (page 83), the letter D is always the ciphertext for the plaintext letter a, so D is a homophone in the *monoalphabetic* cipher known as the Caesar cipher. Here "monoalphabetic" means that there is only one *cipher alphabet*, which means the set of ciphertext equivalents used to transform the plaintext. The row of ciphertext equivalents below the plaintext in Table 2.1, for instance, is the cipher alphabet for the Caesar cipher.

A *polyphone* is a ciphertext symbol that always represents the same *set* of plaintext symbols, typically a set consisting of at most three plaintext symbols. With homophones or polyphones, there is no option for change since the relationship between plaintext and ciphertext is fixed. However, a cipher is called *polyalphabetic* if it has more than one cipher alphabet. In this type of cipher, the relationship between the ciphertext substitution for plaintext symbols is variable. Thus, since each cipher alphabet (usually) employs the same symbols, a given symbol may represent several plaintexts.

An example of a polyalphabetic cipher, the first in history, is that invented by Leon Battista Alberti (see Biography 2.5).

Alberti conceived of a disk with plaintext letters and numbers on the outer ring and ciphertext symbols on an inner movable circle. Alberti divided his ring and corresponding circle into twenty-four equal segments, called *cells*, each containing a symbol. A representation of Alberti's disk is pictured in Figure 2.1 on page 88. We have altered his original presentation since he had ciphertext in lower case and plaintext in upper case, the reverse of what we have as a convention.

Biography 2.5 *The title* Father of Western Cryptography *must go to* Leon Battista Alberti (1404–1472). *Alberti was born on February 14, 1404, in Genoa, Italy, and was the son of a wealthy banker, Lorenzo di Benedetto Alberti. Alberti was raised as Battista in Venice where the family moved shortly after he was born. (He adopted the name Leon later in life.) At the age of ten, he had already learned Latin and his father was teaching him mathematics. His formal education was at the University of Bologna, where he ultimately earned a degree in law. However, he quickly turned his interests to artistic and ultimately scientific thought. Alberti not only taught himself music, became an expert at playing the organ, and wrote sonnets, but also wrote on art, criminology, sculpture, architecture, and mathematics. In 1432, he went to Rome where he became a secretary in the Papal Chancery, and he remained in the arms of church for the rest of his life. In 1434, he went to Florence as part of the papal court of Eugenius IV. It was in the papal secretariat that he became a cryptographer.*

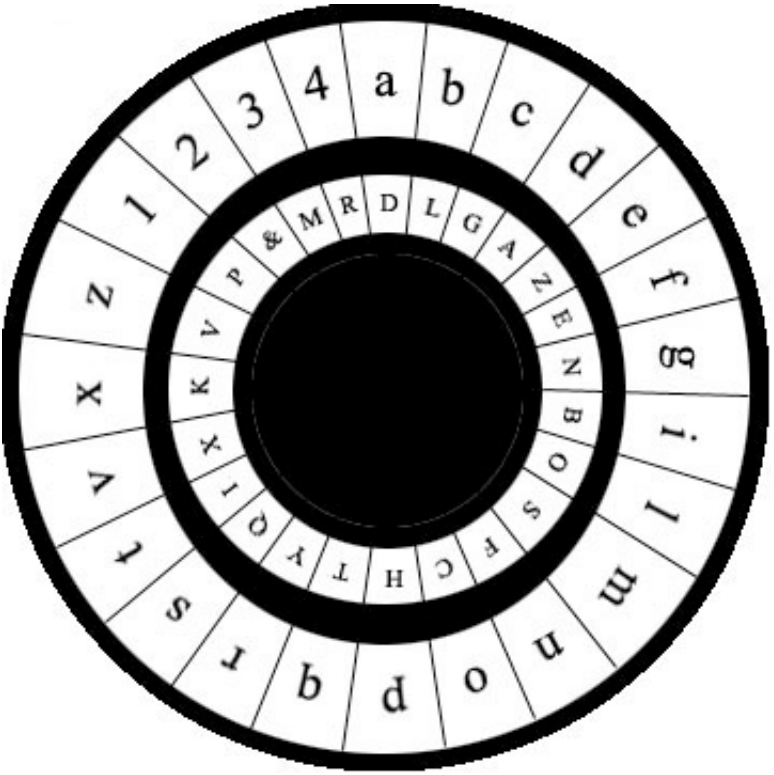


Figure 2.1: Alberti disk.

In Figure 2.1 the plaintext letter z is enciphered as V, so in this setting (one of the twenty-six possible cipher alphabets) the plaintext *zebra*, for instance, would be enciphered as *VZLYD*. However, there is nothing new at this juncture that is any different from, say, the Caesar cipher with the cipher alphabet having the letter c below the Z. Alberti had an idea, however (which is why he wanted the inner circle to be able to rotate). This idea would revolutionize the forward movement of cryptological development. After a random number of plaintext words had been enciphered, usually three or four, Alberti would move the inner disk to a new setting. Hence, he would now be using a *new* cipher alphabet. Suppose that he moved the inner circle so that z sits over K. Then *zebra* would be enciphered as *KADTR*, a new ciphertext for the same plaintext as above since we have a *new* cipher alphabet. In fact, with his cipher disk, Alberti invented the first polyalphabetic cipher in history. However, he did even more.

Alberti had twenty letters, as depicted in Figure 2.1. This excludes the letters h, k, and y, deemed to be unnecessary, and since j, u, and w were not part of his alphabet, this left twenty letters. The inner circle consists of the twenty-four letters of the Latin alphabet, put in the cells at random, including *&*. The disk also included the numbers 1 through 4 in the outer ring of his

original disk. In a book, he used these numbers in two-, three-, and four-digit sets from 11 to 4444 yielding $336 = 4^2 + 4^3 + 4^4$ codegroups. Beside each digit he would write a phrase such as "Send in the troops" for the number 21, say. Then, with the setting in Figure 2.1, the code group 21 is enciphered as *&P*, *enciphered code*. Alberti was the first to discover it, and it is a testimony to his being centuries ahead of his time that enciphered code, when it was rediscovered at the end of the nineteenth century, was simpler than that of Alberti!

Exercises

In Exercises 2.1–2.2, use the Caesar cipher described on page 83 to decrypt the numeric ciphertext.

2.1. $22, 10, 7, \quad 6, 11, 7, \quad 11, 21, \quad 5, 3, 21, 22.$

(*This is a quote by Julius Caesar himself when crossing the River Rubicon, that delineated the frontier between Gaul and Italy proper. The quote indicates the fact that he was virtually declaring war on Rome since his military power was limited to Gaul.* (*See* [88].))

2.2. $3, 16, 1, \quad 21, 22, 11, 9, 15, 3, \quad 25, 11, 14, 14, \quad 6, 17, \quad 22, 17,$
$4, 7, 3, 22, \quad 3, \quad 6, 17, 9, 15, 3.$

(*This is taken from* Supers and Supermen (1920) *by Philip Guedalla* (1889–1944)*, a British writer.*)

2.3. Polybius (see Biography 2.6 on page 90) invented a substitution cipher by enciphering letters into pairs of numbers as follows.

The Polybius Square

Table 2.3

	1	2	3	4	5
1	*a*	*b*	*c*	*d*	*e*
2	*f*	*g*	*h*	*ij*	*k*
3	*l*	*m*	*n*	*o*	*p*
4	*q*	*r*	*s*	*t*	*u*
5	*v*	*w*	*x*	*y*	*z*

Label a five-by-five square with the numbers 1 through 5 for the rows and columns, and string the English alphabet through the rows, considering "ij" as a single letter, as given in Table 2.3.

Then, look at the intersection of any row and column (with row number listed first and column number listed second) as the representation of the letter in question. For instance, k is 25 and q is 41. Hence, the letters are plaintext and the numbers are ciphertext. This device is called the *Polybius checkerboard* or *Polybius square*.

Use the Polybius square to decrypt the numeric ciphertext given by

4434 3111123442 2443 4434 35421154.

(This is the motto of the Benedictine Order.)

Using the Polybius square defined in Exercise 2.3, decipher the numeric ciphertexts given in Exercises 2.4–2.8.

2.4. 331513154343244454 323444231542 3421 243351153344243433

(This is a quote from *Love in a Wood*, Act III, scene iii, (1671) by the English dramatist Williiam Wycherley (ca. 1640–1716).)

2.5. 2134421315 52244423344544 32243314 2111313143 1254 244443 345233 521524222344 (This is a quote from *Odes* Book 3 (23 B.C.) by Horace (Quniton Horatius Flaccus) (65–8 B.C.), a Roman poet.)

2.6. 33344423243322 4533141542 442315 434533 2443 11131324141533441131

(This is a quote from *Emilia Galotti*, Act iv, (1772) by Gotthold Ephraim Lessing (1729–1781), German critic and dramatist.)

Biography 2.6 Polybius*, who lived approximately from 200 to 118 B.C., was a Greek historian and statesman. Polybius' intended use of his square was to send messages great distances by means of torches and hilltops. The sender would hold a torch in each hand, then raise the torch in the right hand the number of times to signal the row, and the torch in the left hand the number of times to signal the column. There is no evidence that these were actually used in this fashion or any other in ancient Greece. However, there are many variations of his cipher that have been constructed. The reader may even concoct one by pairing different letters than "ij" and stringing the alphabet in a different way from the straightforward one given in Table 2.3. One such interpretation of Polybius' cipher involved turning the digits into sounds. A known application in the twentieth century was the one developed by Russian prisoners who used* knocks *to convey speech. For instance, using Table 2.3, a prisoner might knock on a wall twice, followed by three knocks for the letter "h" then proceed in this fashion to send a complete message. Hence, this came to be known as the* knock cipher*.*
Polybius' substitution cipher has found great acceptance among cryptographers up to modern times, who have used it as the basis for numerous ciphers. We will mention some as we encounter them later in our cryptographic voyage.

2.7. 44243215 2443 3234331554

(This quote is from *Money*, Act 3, (1840), by Edward George Bulwer-Lytton (Lord Lytton) (1803–1873), A British literary patron and writer.)

2.8. 32112515 2311434415 433134523154

(This quote is attributed to *Augustus* in *The Lives of the Twelve Caesars* by Suetonius (C. Suetonius Tranquillus) (ca. 70–140 A.D.), a Roman historian (see [88]).

2.2 Classic Ciphers

In Section 2.1 we learned about certain ciphers which comprise a type of cryptosystem that we now define.

Definition 2.4 **Block Ciphers**

A Block Cipher *is a cryptosystem that separates the plaintext message into strings, called* blocks*, of fixed length $k \in \mathbb{N}$, called the* blocklength*, and enciphers one block at a time.*

Classically, block ciphers are divided into two types, substitution and transposition ciphers, about which we learned on page 81. We now look at some special cases of these Block Ciphers, starting with one familiar to us, an example of the simplest kind of encryption.

◆ **Shift Ciphers**

The Caesar cipher is a special case of a *Shift Cipher*, defined as follows. Let $b, n \in \mathbb{N}$ and for each nonnegative $j < n$, define the enciphering transformation by

$$E_e(m_j) = c_j \equiv m_j + b \,(\text{mod } n),$$

for $m_j \in \mathcal{M}$ and $c_j \in \mathcal{C}$, or simply $E_e(m_j) = c_j = m_j + b \in \mathbb{Z}/n\mathbb{Z} = \mathcal{C}$. The deciphering transformation is given by

$$D_d(c_j) = m_j \equiv c_j - b \,(\text{mod } n),$$

or simply $D_d(c_j) = m_j = c_j - b \in \mathbb{Z}/n\mathbb{Z} = \mathcal{M}$. The Shift Cipher is a Symmetric-key Cipher with $d = -e$, since e is addition of b modulo n and $-e = d$ is subtraction of b modulo n, the additive inverse of e. This is an example of a Block Cipher where the blocklengths are $k = 1$. The Caesar cipher is the special case obtained by taking $b = 3$ and $n = 26$. Also, for fans of the Stanley Kubrick film *2001: A Space Odyssey*, take $b = 1$ and $n = 26$, wherein the message *hal* is enciphered as IBM. Shift Ciphers are relatively easy to cryptanalyze since there are only $|\mathcal{A}|$ keys to exhaustively search, where $\mathcal{A}$ is the alphabet of definition. For instance, with the Caesar cipher, $|\mathcal{A}| = 26$.

In turn, the Shift Cipher is a special case of a more general cryptosystem, for which we now set the stage. Let $a, b, n \in \mathbb{N}$ and for $m \in \mathbb{Z}$ define

$$E_e(m) = am + b \in \mathbb{Z}/n\mathbb{Z},$$

where the key e is the ordered pair (a, b). Notice that for $a = 1$ we are back to the Shift Cipher where the key is b. Such a transformation is called an *Affine function*. In order to guarantee that the deciphering transformation exists, we need to know that the inverse of the affine function exists. By Exercise 2.20 on page 105, this means that $f^{-1}(c) \equiv a^{-1}(c - b) \,(\text{mod } n)$ must exist. By the preamble to Definition 1.11 on page 24, this can happen only if $\gcd(a, n) = 1$.

Also, by Definition 1.12 on page 37, there are $\phi(n)$ natural numbers less than n and relatively prime to it. Hence, since b can be any of the choices of natural numbers less than n, we have shown that there are exactly $n\phi(n)$ possible Affine Ciphers, the product of the possible choices for a with the number for b, since this is the total number of possible keys. We have motivated the following.

◆ Affine Ciphers

Let $\mathcal{M} = \mathcal{C} = \mathbb{Z}/n\mathbb{Z}$, $n \in \mathbb{N}$, $\mathcal{K} = \{(a, b) : a, b \in \mathbb{Z}/n\mathbb{Z} \text{ and } \gcd(a, n) = 1\}$, and for $e, d \in \mathcal{K}$, and $m, c \in \mathbb{Z}/n\mathbb{Z}$, set $E_e(m) \equiv am + b \pmod{n}$, and $D_d(c) \equiv a^{-1}(c - b) \pmod{n}$.

Thus, as with the Shift Cipher of which the Affine Cipher is a generalization, $e = (a, b)$ since e is multiplication by a followed by addition of b modulo n, and $d = (a^{-1}, -b)$ is subtraction of b followed by multiplication with a^{-1}. In the case of the Shift Cipher, the inverse is additive and in the case of the Affine Cipher, the inverse is multiplicative. Of course, these coincide precisely when $a = 1$. In either case, knowing e or d allows us to easily determine the other, so they are symmetric-key cryptosystems. They are also Block Ciphers with the trivial blocklengths of $k = 1$.

Example 2.1 Let $n = 26$, and let $\mathcal{M} = \mathcal{C} = \mathbb{Z}/26\mathbb{Z}$. Define an Affine Cipher as follows.

$$E_e(m) = 7m + 5 = c \in \mathbb{Z}/26\mathbb{Z},$$

and since $7^{-1} \equiv 15 \pmod{26}$,

$$D_d(c) = 15(c - 5) = 15c - 23 \in \mathbb{Z}/26\mathbb{Z} = \mathcal{M}.$$

Let the following table give the numerical equivalents in the alphabet of definition for each element in $\mathcal{M} = \mathcal{C}$.

Using the above, we wish to encipher the following message and provide a cryptogram:

cryptanalyst

To do this, we first translate each letter into the numerical equivalent in the alphabet of definition, via Table 2.2 on page 83 as follows.

2 17 24 15 19 0 13 0 11 24 18 19.

Then we apply $E_e(m)$ to each of these numerical equivalents m to get the following.

19 20 17 6 8 5 18 5 4 17 1 8.

Finally, we use Table 2.2 to translate back into the English alphabet to get the cryptogram:

TURGIFSFERBI

which we send.

Both shift and affine ciphers are examples of *simple* or monoalphabetic substitution ciphers, about which we learned on page 87, where we also learned

about polyalphabetic ciphers. An example of a polyalphabetic substitution cipher is the following due to Vigenère (see Biography 2.7) on page 94, although he was responsible for a more complex version than this simple interpretation. He employed the idea that others had invented of using the plaintext as its own key. However, he added something new, a *priming key*, which is a single letter (known only to the sender and the legitimate receiver) that is used to decipher the first plaintext letter, which would, in turn, be used to decipher the second plaintext letter, and so on. The following is an example of an *autokey cipher*, which is a cryptosystem wherein the plaintext itself (in whole or in part) serves as the key (usually after employing an initial priming key).

THE VIGENÈRE TABLEAU

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

◆ The Vigenère Autokey Polyalphabetic Cipher

(a) Put the plaintext letters in a row.

(b) Place the priming key letter below the first plaintext letter. Then put the first plaintext letter below the second, the second below the third, and so on to the penultimate below the last.

(c) Replace each letter of the plaintext with the letter at the intersection of the row labelled by the plaintext letter and column labelled by the key letter.

Example 2.2 *Let us first choose a priming key, say* z*, and assume that the plaintext is* leave at midnight*.*

l	e	a	v	e	a	t	m	i	d	n	i	g	h	t
z	l	e	a	v	e	a	t	m	i	d	n	i	g	h
K	P	E	V	Z	E	T	F	U	L	R	V	O	N	A

For instance, the row labelled l *intersects with the column labelled* z *at the ciphertext letter* K*, and so on. To decipher, the receiver knows the priming key* z*, so this letter is placed above the ciphertext letter* K *and looks in the row labelled* z *to find the letter* K*, then the label of the column in which* K *sits is the plaintext, namely* l*, and so on, as follows.*

z	l	e	a	v	e	a	t	m	i	d	n	i	g	h
K	P	E	V	Z	E	T	F	U	L	R	V	O	N	A
l	e	a	v	e	a	t	m	i	d	n	i	g	h	t

◆ Cryptanalysis of the Vigenère cipher

The Vigenère cipher was considered, up to the middle nineteenth century, to be unbreakable, and it achieved the title of the *chiffre indéchiffrable*. However, in 1863, F.W. Kasiski found a method for cryptanalyzing it (see Biography 2.8 on page 96).

Biography 2.7 Blaise de Vigenère (1523–1596) *had his first contact with cryptography at age twenty-six when he went to Rome on a two-year diplomatic mission. He familiarized himself with the works of his predecessors, such as Alberti (see Biography 2.5 on page 87). His own work, published in 1585, containing his contributions to cryptography, is called* Traicté des Chiffres. *Vigenère discussed steganographic techniques and a variety of cryptographic ideas. Among them was the idea for an autokey polyalphabetic substitution cipher.*

The principal idea behind Kasiski's attack is the observation that repeated portions of plaintext enciphered with the same part of a key must result in identical ciphertext patterns. Therefore, assuming there is no coincidence, one would expect that the same plaintext portions corresponding to repeated ciphertext were enciphered with the same position in the key. Therefore, the number of symbols between the start of repeated ciphertext patterns should be a multiple of the *keylength* (the number of characters in the key). For example, if the repeated ciphertext is *UWY*, called a *trigram*, and if the number of letters between the *W* and the occurrence of *U* in the next trigram *UWY* is, say, 12, and this is not an accident, then 15 is a multiple of the keylength. Since

it is possible that some of the repeated ciphertext segments are coincidental, a method of analyzing them, called a *Kasiski examination*, is to compute the greatest common divisor (gcd) of the collection of all the distances between the repeated sections. Then choosing the largest factor occurring most often among these *gcd*s is (probably) the keylength. Once a probable keylength ℓ, say, is obtained, a frequency analysis can be performed on a breakdown of the ciphertext into ℓ classes (with an individual class containing every ℓ-th character) to determine the suspected key. The following is an illustration of Kasiski's method for finding the keylength. See also the discussion of the "index of coincidence" on page 101, which is the second part of the cryptanalysis, namely determining the most likely distance between each character in the keyword.

Example 2.3 *Suppose that* "trials" *is the keyphrase and* "secret sessions are terminated" *is the plaintext. Then consider the following Vigenère enciphering.*

t	r	i	a	l	s	t	r	i	a	l	s	t	r
s	e	c	r	e	t	s	e	s	s	i	o	n	s
L	**V**	K	*R*	*P*	*L*	**L**	**V**	A	S	T	G	G	J

i	a	l	s	t	r	i	a	l	s	t	r	i
a	r	e	t	e	r	m	i	n	a	t	e	d
I	*R*	*P*	*L*	X	I	U	I	Y	S	M	V	L

Notice that **LV** *is a block that occurs twice, at the beginning and middle of the first table, and the distance between the occurrence of the first* **L** *and the second* **L** *is* 6. *Also, the trigram RPL occurs in first table and again* 12 *units away in the second table. Hence, since* $\gcd(12, 6) = 6$, *this is the probable keylength by the Kasiski examination, which is indeed correct.*

In 1883, Kerckhoffs (see Biography 2.9 on the following page) published *La Cryptographie militaire*, which may be seen as taking the torch passed by Kasiski. The telegraph had made possible the introduction to cryptology of a new device, the *field cipher*, a rapid means for the military to send secure, secret messages in a theater of war. Kerckhoffs also instituted several tenets for field ciphers.

◆ Kerckhoffs' Rules for Field Ciphers

1. The cryptosystem should be practically unbreakable (breakable in theory, perhaps, but not in practice).

2. A compromised cryptosystem should not inconvenience the correspondents. (This is the one from which his principle, displayed on page 96, seems to be derived since it says that the enemy may know the cryptosystem, but one should still be able to send messages since the enemy cannot cryptanalyze with this knowledge and *without* the key.)

3. The key should be easy to both remember and change at will.

4. Cryptosystems must be amenable to being sent by telegraph.

5. The mechanisms of the cryptosystem must be easily portable and entirely operable by a single entity.

6. The cryptosystem must be easy to use without reference to any manuals or the need for deep mental effort.

These six precepts are clearly aimed at complete perfection that even modern-day ciphers would struggle to achieve (where we can replace *telegraph* by *computer* in condition 4). Also, the second condition basically says (and this is implicit in his Kerckhoffs' principle displayed below) that secrecy lies in the *keys* and not in the cipher itself. Later, when we delve into modern ciphers, we will see that this is taken very seriously today. "Key Management," as it has come to be known, is vital since a cryptanalyst who can break a key is better off than one who knows only the cryptosystem itself.

Biography 2.8 Frederich W. Kasiski (1805–1881) *was born on November 29, 1805, in Western Prussia. He enlisted in East Prussia's thirty-third infantry at the age of seventeen, and retired in 1852 as a major. Although interested in cryptography during his military career, he did not publish any of his ideas until after his retirement. In 1863, he published* Die GeheimschRiften und die Dechiffrir - Kunst*, a general solution to cryptanalyzing polyalphabetic cryptosystems with repeating keywords, including the famed Vigenère cipher, a long-sought-after breakthrough.*

Kerckhoffs' Principle

In assessing the security of a cryptosystem, one should always assume the enemy knows the method being used.

Biography 2.9 Jean-Guillaume-Hubert-Victor-François-Alexandre-Auguste Kerckhoffs von Nieuwenhof *was born on January 19, 1835, of Flemish descent, in Holland. His education involved almost two years of study in England plus degrees obtained at the university in Liège. After some teaching positions and some travelling, Kerckhoffs married and settled down in a town outside Paris. He taught languages there for a number of years. By 1876, he had earned his Ph.D. and by 1881 became a professor of German in Paris. While there, he wrote the aforementioned book, which many consider to be the most succinct text on cryptography ever written. In his book, Kerckhoffs elucidated several basic tenets. In modern times, one of these has come to be known as Kerckhoffs' Principle, displayed above, and has been incorporated into modern cryptographic methodology.*

While we are on the topic of field ciphers, the discussion would not be complete without a description of the famous German field cipher that went into service on March 5, 1918. It is described as follows.

◆ The German ADFGVX Field Cipher

The Germans used a table similar to Table 2.4 below where the twenty-six letters of the alphabet plus the ten digits (with 10 represented by ϕ) populate the six-by-six square, where the coordinates of each letter and digit are uniquely determined by the six letters. For instance, the coordinate of T is **XF**.

Table 2.4

	A	**D**	**F**	**G**	**V**	**X**
A	*B*	3	*M*	*R*	*L*	*I*
D	*A*	6	*F*	ϕ	8	2
F	*C*	7	*S*	*E*	*U*	*H*
G	*Z*	9	*D*	*X*	*K*	*V*
V	1	*Q*	*Y*	*W*	5	*P*
X	*N*	*J*	*T*	4	*G*	*O*

Thus, for instance, *field cipher* would be enciphered as:

DF AX FG AV GF FA AX VX FX FG AG

However, this is only the *transitional* ciphertext, which was then placed in another rectangle to be transposed into the *final* ciphertext using a numerical key as follows. We think of the letters of the key *RIFLE* as having numerical equivalents according to the alphabetic order of the letters, namely E corresponds to 1 since it is the letter in RIFLE that appears first in the alphabet, then F corresponds to 2, and so on. Then place the above transitional ciphertext by rows into a matrix as follows in Table 2.5.

Table 2.5

R	**I**	**F**	**L**	**E**
5	3	2	4	1
D	*F*	*A*	*X*	*F*
G	*A*	*V*	*G*	*F*
F	*A*	*A*	*X*	*V*
X	*F*	*X*	*F*	*G*
A	*G*			

Now the final ciphertext is obtained by "peeling off" the *columns* in the above rectangle according to the order of the numbers as follows.

FFVGAVAXFAAFGXGXFDGFXA

To decipher, we reverse the process, which is easy as long as you know the keyword *rifle*.

Example 2.4 *Suppose that we want to decipher the following, assuming that it was encrypted using the above cipher.*

XFVAXAFFGXFFGXFXFXXGDXDAFGVA

Since there are twenty-eight letters in the ciphertext, we know that the columns under **L** *and* **E** *will have only five letters while the others will have six. Thus, we place the first five letters* $XFVAX$ *under* **E**, *the next six* $AFFGXF$ *under* **F**, *the next six* $FGXFXF$ *under* **I**, *the next five* $XXGDX$ *under* **L**, *and the final six* $DAFGVA$ *under* **R** *the as in Table 2.6.*

Table 2.6

R	**I**	**F**	**L**	**E**
5	3	2	4	1
D	*F*	*A*	*X*	*X*
A	*G*	*F*	*X*	*F*
F	*X*	*F*	*G*	*V*
G	*F*	*G*	*D*	*A*
V	*X*	*X*	*X*	*X*
A	*F*	*F*		

Then we unravel by taking them out by rows, as follows.

DFAXXAGFXFFXFGVGFGDAVXXXXAFF

Now, we look up each digraph in Table 2.5 to get the plaintext:

find the weapons

The reason the name ADFGX was chosen for the cipher is that their Morse code symbols are . -, - . ., . . - ., - - ., - . . -, respectively, which are sufficiently dissimilar so as to avoid confusion. For the benefit of the reader, we provide the following complete table of Morse code symbols.

Morse Code Symbols

Table 2.7

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* - - . .	*0* - - - - -	*1* . - - - -	*2* . . - - -	*3* . . . - -
4 . . . . -	*5*	*6* -	*7* - - . . .	*8* - - - - .	*9* - - - - .

Fullstop . - . - . -	*Comma* - - . . - -	*Query* . . - - . .

On page 81, we saw that a simple substitution cipher encrypts single plaintext symbols as single ciphertext symbols. When groups of one *or more* symbols are replaced by other groups of ciphertext symbols, then we call this cryptosystem a *Polygram Substitution Cipher*. For instance, there are *Digraph ciphers*,

where two letters are used as a single symbol such as with the above ADFGVX cipher. There is another famous digraph cipher as follows.

◆ **The Playfair Cipher**

Consider the following Digraph cipher, where the letters W and X are considered as a single entity.

Table 2.8

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

Pairs of letters are enciphered according to the following rules.

(a) If two letters are in the same row, then their ciphertext equivalents are immediately to their right. For instance, VC in plaintext is RV in ciphertext. (This means that if one is at the right or bottom edge of the table, then one "wraps around" as indicated in the example.)

(b) If two letters are in the same column, then their cipher equivalents are the letters immediately below them. For example, ZF in plaintext is UR in ciphertext, and XB in plaintext is GW in ciphertext.

(c) If two letters are on the corners of a diagonal of a rectangle formed by them, then their cipher equivalents are the letters in the opposite corners and same row as the plaintext letter. For instance, UL in plaintext becomes GF in ciphertext and SZ in plaintext is FD in ciphertext.

(d) If the same letter occurs as a pair in plaintext, then we agree by convention to put a Z between them and encipher.

(e) If a single letter remains at the end of the plaintext, then a Z is added to it to complete the digraph.

Example 2.5 *Using the Playfair cipher presented above, we decipher the ciphertext:* **AMKNUIUPYFUIEJYO**.

Since **AM** *sits on a diagonal with corresponding letters* **do**, *then that digraph is the plaintext. Similarly employing all the rules, we get* **donotztrustzthem** *and once we remove the extraneous letters* **z**, *occuring twice, since* **tt** *occurs twice, we get* **do not trust them** *as the plaintext.*

We now look at another classical polygram substitution cipher that requires knowledge of some elementary matrix theory. The reader unfamiliar with such concepts, or requiring a brief review of the ideas and notation, should consult pages 308–311 in Appendix A.

◆ The Hill Cipher

Let $\mathcal{K} = \{e \in \mathcal{M}_{r\times r}(\mathbb{Z}/n\mathbb{Z}) : e \text{ is invertible}\}$, for fixed $r, n \in \mathbb{N}$, and set $\mathcal{M} = \mathcal{C} = (\mathbb{Z}/n\mathbb{Z})^r$. Then for $m \in \mathcal{M}$, $e \in \mathcal{K}$, $E_e(m) = me$, and $D_d(c) = ce^{-1}$, where $c \in \mathcal{C}$. (Note that e is invertible if and only if $\gcd(\det(e), n) = 1$. See Theorem A.6 on page 311.) This cryptosystem is known as the Hill Cipher, created by L.S. Hill (see Biography 2.11 on page 102).

Example 2.6 Let $r = 2$ and $n = 26$ with alphabet of definition $\mathcal{A} = \mathbb{Z}/26\mathbb{Z}$, where Table 2.2 on page 83 gives the numerical equivalents of plaintext letters. Thus, $\mathcal{M} = \mathcal{C} = (\mathbb{Z}/26\mathbb{Z})^2$, and $\mathcal{K}$ consists of all invertible two-by-two matrices with entries from $\mathbb{Z}/26\mathbb{Z}$, so if $e \in \mathcal{K}$, then $\gcd(\det(e), 26) = 1$. Let us take

$$e = \begin{pmatrix} 3 & 6 \\ 1 & 5 \end{pmatrix}$$

for which $\det(e) = 9$. Suppose that we want to encipher *movie*. First we get the numerical equivalents from Table 2.2: 12, 14, 21, 8, 4. Thus, we may set $m_1 = (12, 14)$, $m_2 = (21, 8)$, and $m_3 = (4, 25)$, where z, with numerical equivalent of 25, is used to complete the last pair. Now use the enciphering transformation defined in the Hill cipher.

$$E_e(m_1) = (12, 14)\begin{pmatrix} 3 & 6 \\ 1 & 5 \end{pmatrix} = (24, 12),$$

$$E_e(m_2) = (21, 8)\begin{pmatrix} 3 & 6 \\ 1 & 5 \end{pmatrix} = (19, 10),$$

and

$$E_e(m_3) = (4, 25)\begin{pmatrix} 3 & 6 \\ 1 & 5 \end{pmatrix} = (11, 19).$$

Now we use Table 2.2 to get the ciphertext letter equivalents and send $XMTKLT$ as the cryptogram.

Now we show how decryption works. Once the cryptogram is received, we must calculate the inverse of e, which is

$$e^{-1} = \begin{pmatrix} 15 & 8 \\ 23 & 9 \end{pmatrix}.$$

Now apply the deciphering transformation to the numerical equivalents of the ciphertext as follows. Given $c_1 = (12, 2)$, $c_2 = (19, 10)$, $c_3 = (11, 19)$, we have

$$D_d(c_1) = D_{e^{-1}}(24, 12) = (24, 12)\begin{pmatrix} 15 & 8 \\ 23 & 9 \end{pmatrix} = (12, 14),$$

$$D_d(c_2) = D_{e^{-1}}(19, 10) = (19, 10)\begin{pmatrix} 15 & 8 \\ 23 & 9 \end{pmatrix} = (21, 8),$$

and $D_d(c_3) = D_{e^{-1}}(11, 19) = (11, 19)\begin{pmatrix} 15 & 8 \\ 23 & 9 \end{pmatrix} = (4, 25).$

The letter equivalents now give us back the original plaintext message *movie* after discarding the letter z at the end.

◆ The One-Time Pad

In the description of the simple version of the Vigenère cipher given on page 93, there is only a single letter for the key. Imagine a cipher where the key has as many symbols as the plaintext itself, and that the key is truly randomly generated and never used more than once. Moreover, both the key and the plaintext are written in binary, namely the alphabet of definition is $\mathcal{A} = \{0, 1\}$. Ciphertext is produced by adding the plaintext to the ciphertext modulo 2. This is the description of the *one-time pad* developed by Vernam and Mauborgne in 1918 (see Biography 2.10). Since the key is as long as the plaintext, and the key selected is truly random and used only once, then the ciphertext is completely random as well. Thus, the one-time pad is unbreakable, meaning that it is impossible to crack by any cryptanalytic methods. Later, when we have developed more mathematics, we will return to this important cipher, which was shown in 1949 by Shannon, to be *proved* to be unbreakable. We will return to this topic later as well when we have the machinery in place to discuss it in detail. Furthermore, the Vernam cipher is not a block cipher, rather it is an example of a stream cipher about which we will learn in Section 2.3.

Biography 2.10 Gilbert S. Vernam, (1890–1960) *a cryptologist working for the American Telegraph and Telephone Company (AT&T), came to the realization that if the Vigenère cipher were used with a truly random key, with keylength the size of the plaintext, called a* running key*, then the Kasiski attack would fail. At this time, AT&T was working closely with the armed forces, so the company reported this to the Army. It came to the attention of Major Mauborgne, head of the Signal Corps' research and engineering division. (When Mauborgne was still just a first lieutenant in 1914, he had published the first solution of the Playfair cipher, see page 99.) He played with Vernam's idea and saw that if the key were reused, then a cryptanalyst could piece together information and recover the key. Hence, he added the second component to the Vernam idea. The key must be used once, and only once, then destroyed. Now, the idea was complete. Use the Vigenère cipher with a truly random running key that is used* exactly *once, then destroyed. The system is called the* one-time pad *and sometimes, perhaps inappropriately in view of Mauborgne's contribution, the* Vernam cipher.

◆ The Index of Coincidence

Another individual, W.F. Friedman (see Biography 2.12 on page 104), was on the fringes of the discovery of the one-time pad. Perhaps his greatest contribution was his discovery of the *index of coincidence* since it provided a much-needed intimate link between cryptology and mathematics that would become paramount as the twentieth century ended and the new millenium began. His index of coincidence is defined, for a ciphertext $\mathcal{C}$, to be the probability that two letters selected at random from $\mathcal{C}$ are identical.

Below we show how to mathematically demonstrate that the index of coincidence for a monoalphabetic cipher is about 0.065 and the index of coincidence for a polyalphabetic cipher is somewhere between 0.0385 and 0.065. For very long keywords, the index of coincidence for polyalphabetic ciphers will be closer to 0.0385. Hence, by a simple analysis of intercepted ciphertext, a cryptanalyst can relatively easily determine the type of cryptosystem being used. This was quite a breakthrough. Moreover, his idea contained a mechanism for determining the probable keylength, as had Kasiski. Here is how it works.

Biography 2.11 Lester S. Hill *devised this cryptosystem in 1929. His only published papers in the area of cryptography appeared in 1929 and 1931. Thereafter, he kept working on cryptographic ideas but turned all of his work over to the Navy in which he had served as a lieutenant in World War I. He taught mathematics at Hunter College in New York from 1927 until his retirement in 1960. He died in Lawrence Hospital in Bronxville, New York, after suffering through a lengthy illness. Hill's rigorous mathematical approach may be said to be one of the factors which has helped foster today's solid grounding of cryptography in mathematics.*

First we need a table of letter frequencies for the English alphabet. This well-known, standard table is presented below as Table 2.9.

Now suppose that n stands for the number of letters in a ciphertext, $\mathcal{C}$, and n_j stands for the number of letters in the j-th position of the English alphabet. In other words, n_1 is the number of occurrences of the letter a in $\mathcal{C}$, n_2 is the number of occurrences of the letter b in $\mathcal{C}$, and so on. Without getting into the reasons for it, the Index of Coincidence, $\mathcal{IC}$, is given as approximately the following:

$$\mathcal{IC} \approx \left(\frac{n_1}{n}\right)^2 + \left(\frac{n_2}{n}\right)^2 + \cdots + \left(\frac{n_{26}}{n}\right)^2.$$

So if we want to compute $\mathcal{IC}$ for the English language from Table 2.9, and since each of the numbers in the table is a percentage, then we divide each by 100 and get

$$\mathcal{IC} \approx (0.8167)^2 + (0.01492)^2 + \cdots + (0.00074)^2 = 0.065,$$

which explains the aforementioned Index of Coincidence for monoalphabetic ciphers, since the frequency is invariant. (Note that the symbol $\approx$ means "approximately equal to," which is good enough since we are dealing with a statistical analysis wherein approximations are sufficient for our investigations.)

Relative Letter Frequencies for English
Table 2.9

a	b	c	d	e	f	g	h	i
8.167	1.492	2.782	4.253	12.702	2.228	2.015	6.094	6.966
j	k	l	m	n	o	p	q	r
0.153	0.772	4.025	2.406	6.749	7.507	1.929	0.095	5.987
s	t	u	v	w	x	y	z	
6.327	9.056	2.758	0.978	2.360	0.150	1.974	0.074	

Now for any language, such as English, with a twenty-six-letter alphabet, in which each letter has the *same* frequency, we get

$$\mathcal{IC} = 26(1/26)^2 = 0.038,$$

which is approximately half of the above Index of Coincidence for English. Hence, the Index of Coincidence helps us in determining if the ciphertext comes from a monoalphabetic or polyalphabetic cipher in the following manner. The closer the $\mathcal{IC}$ is to 0.065, the more likely it is that the message came from a monoalphabetic cipher. If the $\mathcal{IC}$ is much less than 0.065, the cipher is most likely polyalphabetic since frequencies are evened out by polyalphabetic cryptosystems. Hence, the closer the $\mathcal{IC}$ is to 0.038, the greater the chance is that the cipher is polyalphabetic. This was a major contribution by Friedman since he tied a mathematical tool, statistical analysis, to the study of cryptography.

Exercises

In Exercises 2.9–2.12, use the Affine cipher in Example 2.1 on page 92 to decipher the given numeric ciphertext.

2.9. $5, 4, 4, 13, 15, 9, 7, 8, 25, 18, 8, 2, 7, 3, 7, 1, 8, 7, 20, 18, 14, 20, 25, 18, 8$

(*This is a title of a novel, published in 1929 by the German writer* Erich Maria Remarque (1897–1970).)

2.10. $25, 18, 4, 17, 5, 14, 20, 7, 7, 1, 25, 15, 4, 18, 7, 22, 7, 20, 21, 20, 25, 3, 1, 25, 4, 0$

(*This is a quote from* Titan (1800–1803), Volume 2, 140, cycle*, by the German writer* John Paul Richter (1793–1825).)

2.11. $7, 22, 7, 20, 17, 1, 9, 18, 9, 1, 8, 2, 7, 20, 7, 1, 15, 4, 8, 25, 14, 5, 19, 25, 4, 4, 5, 12,$
$25, 20, 5, 8, 9, 25, 18$

(*This is a quote from* Blue Hotel (1899)*, written by the war correspondent and American writer* Steven Crane.)

2.12. $8, 2, 7, 3, 2, 9, 8, 7, 11, 5, 18, 1, 12, 15, 20, 0, 25, 18$

(*This is the title of a poem from* 1899 *written by the British poet* Rudyard Kipling (1865–1936)*, who won the Nobel Prize in* 1907.)

In Exercises 2.13–2.16, use the key given in Example 2.2 on page 94 to decrypt the folowing ciphertexts via the Vigenère cipher.

2.13. **KZJZMASKSVQRTBWRWFRV**

(*This is a quote from* Ars amatoria, Book II, line 233 *by* Ovid (43 B.C.–18 A.D.)*, a Roman poet.*)

2.14. **LMNCGFDDGWWTVNZURGLAHDGWW**

(*This is a quote from* The Imitation of Christ *by* Thomas A. Kempis (1380–1471)*, a German Augustinian canon and writer.*)

Biography 2.12 William Frederick Friedman (1891–1969) *was born in Kishinez, Russia, on September 14, 1891. In 1892, his father fled the antisemitic regulations in Czarist Russia, and his family joined him in Pittsburgh the following year. William obtained his bachelor's degree from Cornell, then joined the Riverside Laboratories* (*which today would be considered to be a "think tank"*) *outside Chicago, in 1915. There Friedman met Elizabeth Smith (1892–1980), whom he married in 1917. The Friedmans soon turned their attention to cryptology. William was training cryptologists at Riverbank, and for course material he wrote eight publications* (*which are collectively known as the* Riverbank Publications). *Today they are highly regarded as containing the basic essentials of cryptological material. Friedman himself, when looking back over his career, thought his greatest cryptological contribution was his conceiving of the* Index of Coincidence, *which appeared in his monograph no. 22 of the numerous ones that he published. Soon after his marriage to Elizabeth, William became the director of the Department of Codes and Ciphers, among his other duties, at Riverbank. After the outbreak of World War I, Riverbank offered its services to the government, and since no such federal agency existed at the time, Riverbank became the* de facto *cryptographic center for the American government. One of the first accomplishments the Friedmans achieved was the following. The Germans had been encouraging Hindu radicals to work toward independence from Britain in the hopes of diverting attention and strength from the war effort. Some of these radicals, who lived in the U.S.A., were sending messages about arms shipments. It turns out they were trying to buy arms in the U.S.A. and ship them from the West Coast. The Friedman's deduced that the codebook used by these radicals was a German-English dictionary published in 1880. This aided William in his testimony given at the trial of 135 Hindu radicals in San Francisco.*

The Friedmans quit Riverbank toward the end of 1920. In 1921, Friedman joined the American Black Chamber, where he eventually headed the Research and Development Division and stayed there until its dissolution in 1929. One man may be said to be chiefly responsible for the creation and (*possibly*) *the dissolution of the American Black Chamber.*

After World War II, Friedman continued in government signals intelligence until 1949 when he became head of the code division of the new Armed Forces Security Agency, *which evolved into the* National Security Agency (NSA). *At NSA he became the chief cryptologist. By the late 1960's his health faded. He died in 1969 in Washington, D.C., and was buried at Arlington National Cemetery. For all his accomplishments and pioneering efforts, he has been dubbed* America's greatest cryptologist.

2.15. **RLTKOPTNJYJDBGKLNA**

(*This is a quote from* Satires *by* Juvenal (circa 60–140 A.D.), *a Roman rhetorician and satirical poet.*)

2.16. **NDEDFKNHVBRGASGUR**

(*This is a quote from* The Greek Anthology *and attributed to* Menander (circa 342–291 B.C.)*, a Greek comic dramatist.*)

2.17. Assume that the following bitstring is a randomly chosen key for the one-time pad described on page 101:

$$k = (1100101000111111000101111010101110100011111110010 0).$$

Also, assume that the following was enciphered using k:

$$c = (1010101100100101000111110011101111110010010110000 0).$$

Find the plaintext string.

2.18. Assume that the following bitstring is a randomly chosen key for the one-time pad:

$$k = (1100101111010000000110010111010011111111000101101001010101011101 1).$$

Also, assume that the following was enciphered using k:

$$c = (1001001110111001001011110111010101011110000100000001011000110000).$$

Find the plaintext string.

2.19. Interpret the plaintext solution to Exercise 2.17 as a bitstring that is the concatenation of bitstrings of length 5 each. Convert the binary to decimal form and use Table 2.2 on page 83 to find the English text equivalent. Do the same for Exercise 2.18.

(*The plaintext from the solutions of Exercises 2.17–2.18 comprise a quote from* Stray Birds *by* Rabindranath Tagore (1861–1941)*, a Bengali philosopher and winner of the 1913 Nobel Prize.*)

2.20. Show that the inverse, if it exists, of an affine function as defined on page 91 is given by

$$f^{-1}(c) = a^{-1}(c - b) \pmod{n}.$$

Using the ADFGVX cipher described on page 97, and assuming the key is **VICTORY***, decipher the cryptograms in Exercises 2.21–2.24.*

2.21. **AFXXFGFAVAAXAGVFDFVGXGDA**
VGFAVFFDVFGDAFFXGFDAFFFF

(*This is a quote fram* Gracian's Manuals (1653) *by* Baltasar Gracián (1601–1658)*, a Spanish Jesuit writer.*)

2.22. **XAFVXXFDXAGAXAXGXAFFFD FFXFFFAFXGAXAGFXXAVG**

(*This is a quote from* Pensées (1842) *by* Joseph Joubert (1754–1824)*, a French essayist.*)

2.23. **XAXAFFDGDVFXXVDVFAAAXGXAGVXVVAAFGXXX**

(*This comprises the dying words of the Roman emperor* Julian the Apostate (332–363 A.D.).)

2.24. **FXVAFGDXFAXVDFFFAAXVAFDAFGFXXA AXFXFAFGGXAFGDGXFFFXAFAXFXFXFA**

(*This is a quote from* Laws of Repentance *by* Maimonides (Moses ben Maimon) (1135–1204)*, a Jewish philospoher and Rabbinic scholar.*)

Using the Playfair cipher described on page 99, decipher the cryptograms in Exercises 2.25–2.28.

2.25. **VMNDONNKHYMFTAFMKEHMVMNDONNK**

(*This is a quote from* Dictionnaire Philosophique (1765) *by* Voltaire (François-Marie Arouet) (1694–1778)*, a French writer and philosopher.*)

2.26. **KDELKTFCWTJE**

(*This is an early fourteenth-century proverb.*)

2.27. **HQAODJWBZKAI**

(*This is a late fourteenth-century proverb.*)

2.28. **VIKTUOVYDJEAZPGTUA**

(*This is a late fourteenth-century proverb.*)

Using the key, e, given in Example 2.6 on page 100, decipher the ciphertexts in Exercises 2.29–2.32, then convert to English plaintext via Table 2.2 on page 83.)

2.29. 23, 2, 13, 24, 23, 7, 9, 23, 16, 22, 12, 19, 11, 9, 4, 8, 0, 20, 10, 2, 0, 25, 21, 3, 12, 22, 12, 8

(*This is a quote from* Ballade of Soporific Absorption (1931) *by* J.C. Squire (1884–1958)*, an English man of letters.*)

2.30. 12, 19, 14, 8, 17, 13, 8, 6, 18, 19, 3, 18, 14, 17, 19, 19, 21, 14, 25, 19, 4, 5

(*This is the title of a poem written by* James Thomson (1834–82)*, a Scottish poet.*)

2.31. $24, 22, 23, 7, 9, 23, 17, 7, 12, 6, 11, 9, 14, 16, 17, 13, 19, 6, 10, 2$

(*This and its conclusion in Exercise 2.32 comprise a quote from* The Gentle Art of Making Enemies (1890) *by* James McNeil Whistler (1834–1903)*, an American-born painter.*)

2.32. $24, 22, 3, 11, 23, 1, 15, 2, 19, 4, 8, 6, 7, 11$

2.33. It can be shown that a variant of the Vigenère cipher given on page 93 is the *Beaufort cipher*, which is defined as follows.

Fix $r, n \in \mathbb{N}$. Both the encryption and decryption functions are given by

$$x \mapsto (e_1 - x_1, e_2 - x_2, \ldots, e_r - x_r),$$

for

$$e = (e_1, \ldots, e_r) \in \mathcal{K} \text{ and } x = (x_1, \ldots, x_r) \in (\mathbb{Z}/n\mathbb{Z})^r.$$

In other words, the encryption and decryption functions are the same, namely they are their own inverses.

Assuming that $r = 3$, $n = 26$, and the following was enciphered using the Beaufort cipher, with the key KEY, find the plaintext using Table 2.2.

$$21, 4, 6, 18, 22, 10, 23, 22, 6, 6, 0, 16, 18, 2, 24, 17, 2, 17, 2, 17, 18$$

(*This is a quote from* Julius Caesar, Act 3, Scene 2 (1599) *by* William Shakespeare (1564–1616)*, perhaps the greatest English dramatist.*)

(*To see that the Beaufort cipher is diectly related to the Vigenère cipher consider the following. For simplicity, we assume that we are working modulo* 26*. Let* $\mathbf{P}$ *be the plaintext,* $\mathbf{C}$ *be the ciphertext, and* $\mathbf{K}$ *be the key. Then with Vigenère, enciphering is done via*

$$\mathbf{P} + \mathbf{K} \equiv \mathbf{C} \,(\text{mod } 26)$$

and deciphering is done via

$$\mathbf{C} - \mathbf{K} \equiv \mathbf{P} \,(\text{mod } 26).$$

With Beaufort, enciphering is done by

$$\mathbf{K} - \mathbf{P} \equiv \mathbf{C} \,(\text{mod } 26)$$

and deciphering occurs by

$$\mathbf{K} - \mathbf{C} \equiv \mathbf{P} \,(\text{mod } 26).)$$

2.34. Compute the index of coincidence, using the formula displayed on page 102, for the following ciphertext.

VEGSWKXWGGAQSCYRCRCRS

Biography 2.13 *The Beaufort cipher was invented by* Admiral Sir Francis Beaufort*, Royal Navy, who was also the creator of the Beaufort scale, which is an instrument that meteorologists use to indicate wind velocities on a scale from 0 to 12, where 0 is calm and 12 is a hurricane.*
The self-decrypting Beaufort cipher was used in a rotor-based cipher machine called the Hagelin M-209, invented in the early 1940's by Boris Caesar Wilhelm Hagelin*. Hagelin was born on July 2, 1892, in the Russian Caucasus. In 1922, Emanuel Nobel, nephew of the famed Alfred Nobel, put Hagelin to work in the firm* Aktiebolaget Cryptograph *or Cryptograph Incorporated. This was a company owned by Avid Gerhard Damn, who invented cipher machines of his own. Hagelin simplified and improved one of Damn's machines, much to the liking of the Swedish army, who placed a large order with the Damn firm. After Damn's death in 1927, Hagelin ran the firm. Later he developed the M-209, which became so successful that in the early 1940's more than 140,000 were manufactured. The royalties from this alone made Hagelin the first to become a millionaire from cryptography.*

2.3 Stream Ciphers

In Section 2.1 we learned about symmetric-key ciphers, and in Section 2.2 we learned about the first kind of such ciphers, block ciphers. This section is devoted to the second kind, stream ciphers. In order to place ourselves in the position of being able to define such cryptosystems, we need the following.

Definition 2.5 **Keystreams, Seeds, and Generators**

If $\mathcal{K}$ is the keyspace for a set of enciphering transformations, then a sequence $k_1k_2\cdots \in \mathcal{K}$ is called a keystream*. A keystream is either randomly chosen or generated by an algorithm, called a* keystream generator*, which generates the keystream from an initial small input keystream called a* seed*. Keystream generators that eventually repeat their output are called* periodic*.*

◆ Randomness

When using a computer, the notion of a randomly generated sequence can only be approximated. In practice, we use a computer program that generates a sequence of digits in a fashion that appears to be random, called a *pseudorandom number generator* (PRNG). Here we say "appears to be random" since computers are *finite state* devices, so any random-number generator on a computer must be periodic, which means it is predictable, so it cannot be truly random. The most that one can expect, therefore, from a computer is *pseudorandomness*, meaning that the numbers pass at least one statistical test for randomness. Of course, these pseudorandom number generators are periodic, but if the periods are large enough, then they can be used for cryptographic applications. Actually designing them is not so easy as it sounds, and there is a vast literature concerning how to obtain a secure means of generating a sequence of digits that has the statistical properties of a truly random sequence, at least in appearance. Even more than that, these keystreams must be *cryptographically secure*. This means they must satisfy the additional property that, for a given output bit, the next output bit must be computationally infeasible (see Footnote 2.1 on page 86) to predict, even given knowledge of all previous bits, knowledge of the algorithm being used, and knowledge of the hardware. These issues, however, are not our concern here. We will assume that we have such a cryptographically secure pseudorandom number generator (CSPRNG) for our keystream, or a truly randomly chosen keystream, and proceed with learning cryptographic techniques.

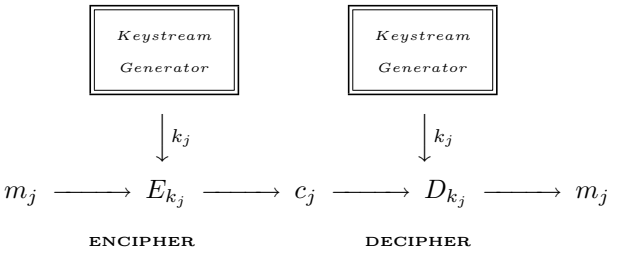
Definition 2.6 **Stream Ciphers**

Let $\mathcal{K}$ be a keyspace for a cryptosystem and let $k_1k_2\cdots \in \mathcal{K}$ be a keystream. This cryptosystem is called a stream cipher *if encryption upon plaintext strings $m_1m_2\cdots$ is achieved by repeated application of the enciphering transformation on plaintext message units, $E_{k_j}(m_j) = c_j$, and if d_j is the inverse of k_j, then deciphering occurs as $D_{d_j}(c_j) = m_j$ for $j \geq 1$. If there exists an $\ell \in \mathbb{N}$ such*

that $k_{j+\ell} = k_j$ *for all* $j \in \mathbb{N}$*, then we say that the stream cipher is* periodic *with* period ℓ*.*

The following is the simplest flow chart for a stream cipher.

Diagram 2.1 A Stream Cipher



Generally speaking, stream ciphers are faster than block ciphers from the perspective of hardware. The reason is that stream ciphers encrypt individual plaintext message units, usually, but not always, one binary digit at a time. In practice, the stream ciphers used are most often those that do indeed encipher one bit at a time. However, in view of our general Definition 2.6 on the page before, we may view a block cipher as a special case of a stream cipher having constant keystream $k_j = k$ for all $j \geq 1$. At least historically, the distinction between block ciphers and stream ciphers is not clear cut. Modern distinctions deem that stream ciphers can encrypt a single bit of plaintext at a time, whereas block ciphers take a number of bits (typically 64 in modern cryptosystems such as the DES cipher, which we will study in Section 3.1) and encipher them as a single block. (See page 123 for some mechanisms that allow us to "stream" a block cipher.)

Typically, stream ciphers are classified as follows.

Definition 2.7 **Synchronous and Self-Synchronizing Ciphers**

A stream cipher is said to be synchronous *if the keystream is generated without use of either the plaintext or of the ciphertext, called keystream generation* independent *of the plaintext and ciphertext. A stream cipher is called* self-synchronizing (*or* asynchronous) *if the keystream is generated as a function of the key and a fixed number of previous ciphertext units. If the stream cipher utilizes plaintext in the keystream generation, then it is called* nonsynchronous*.*

The following two flow charts illustrate a general synchronous and a general asynchronous cipher.

Diagram 2.2 A Synchronous Stream Cipher

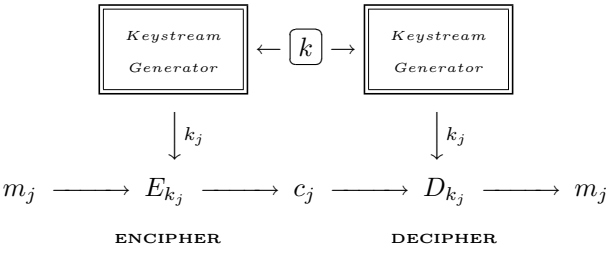
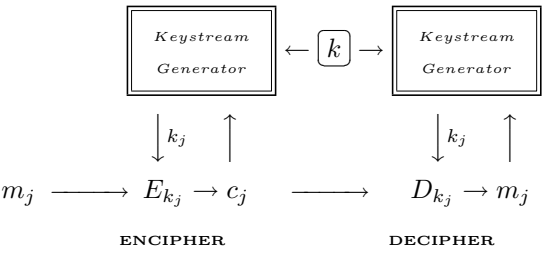


Diagram 2.3 A Self-Synchronizing Stream Cipher



The distinctions between block and stream ciphers are more readily seen in practice than in theory. Stream ciphers encrypting one bit at a time are not suitable for software implementation since bit manipulation is time consuming. Where stream ciphers win out is in the arena of error propagation. Obviously, with a block cipher, a single error will corrupt at least a block's worth of data, whereas implementation of a synchronous stream cipher can guarantee that a single bit error will result in only a single bit of corrupted plaintext. Thus, synchronous stream ciphers would be useful where lack of error propagation is critical. However, use of self-synchronizing stream ciphers can result in error propagation. If the keystream is acting on the nth ciphertext digit and an error occurs, then the deciphering of up to n subsequent ciphertext digits may be incorrect.

The following illustration is an example of a synchronous stream cipher.

Example 2.7 Fix $r, n \in \mathbb{N}$, and let $\mathcal{M} = \mathcal{C} = (\mathbb{Z}/n\mathbb{Z})^s$, the elements of which are s-tuples from $\mathbb{Z}/n\mathbb{Z}$, and $\mathcal{K} = \mathbb{Z}^r$, where $s \geq r$. For $e = (e_1, e_2, \ldots, e_r) \in \mathcal{K}$, let

$$E_{e_j}(m_j) = m_j + e_{j \,(\mathrm{mod}\ r)} \,(\mathrm{mod}\ n), \text{ for all } j = 1, 2, \ldots, s,$$

and for $c = (c_1, c_2, \ldots, c_s)\mathcal{C}$, let

$$D_{d_j}(c_j) = c_j - e_{j \,(\mathrm{mod}\ r)} \,(\mathrm{mod}\ n), \text{ for all } j = 1, 2, \ldots, s.$$

From the above, if we take the cipher with key e of length r, then it may be considered to be a periodic synchronous stream cipher with period r, which is why the subscript on the key is modulo r. The key

$$e = (e_1, e_2, \ldots, e_r)$$

provides the first r elements of the keystream $k_j = e_j$ for $1 \leq j \leq r$, after which the keystream repeats itself.

We studied the Vernam cipher on page 101, which has a mathematically more rigorous definition as follows.

◆ The Vernam Cipher

The Vernam cipher is a stream cipher with alphabet of definition $\mathcal{A} = \{0, 1\}$ that enciphers in the following fashion. Given a bitstring

$$m_1 m_2 \cdots m_n \in \mathcal{M},$$

and a keystream

$$k_1 k_2 \cdots k_n \in \mathcal{K},$$

the enciphering transformation is given by

$$E_{k_j}(m_j) = m_j + k_j = c_j \in \mathcal{C},$$

and the deciphering transformation is given by

$$D_{k_j}(c_j) = c_j + k_j = m_j,$$

where $+$ is addition modulo 2. The keystream is randomly chosen and never used again. For this reason, the Vernam cipher is also called the *one-time pad*.

Current interest in stream ciphers is probably due to the palatable theoretical properties of the one-time pad. A one-time pad can be shown to be theoretically unbreakable. (It was not until 1949 with Shannon's development [86] of the concept of *perfect secrecy* that the one-time pad was proved to be unbreakable, something assumed to be true for many years prior to the proof.) The reason the one-time pad is unbreakable is that since the key is used only once then discarded, a cryptanalyst with access to the ciphertext $c_1 c_2 \cdots c_n$ can only guess at the plaintext $m_1 m_2 \cdots m_n$, since both are equally likely. Conversely, it has been shown that to have a theoretically unbreakable system means that the keylength must be at least that of the length of the plaintext. This vastly reduces the practicality of the system. The reason, of course, is that since the secret key (which can be used only once) is as long as the message, then there are serious key-management problems. Nevertheless, it is part of the folklore that Soviet spies used one-time pads to send messages, and that they were also used in German diplomatic systems starting in the late 1920's. Even the much-ballyhooed *hot-line* between Washington and Moscow, inspired by the Cuban

missile crisis of the 1960's, used what they called the *one-time tape*, which was a physical manifestation of the Vernam cipher. At the American end, this took the form of the ETCRRM II or *Electronic Teleprinter Cryptographic Regenerative Repeater Mixer II*. The manner in which the one-time tape worked was that there existed two magnetic tapes, one at the enciphering source and one at the deciphering end, both having the same keystream on them. To encipher, one performs addition modulo 2 with the plaintext and the bits on the tape. To decipher, the receiver performs addition modulo 2 with the ciphertext and the bits on the (identical) tape at the other end. Thus, they had instant deciphering and perfect secrecy if they used truly random keystreams, each used only once, and the tapes were burned after each use. The same keystream cannot be used twice since the one-time pad would then be open to a known-plaintext attack, given that the key k can be computed by addition modulo 2 of the plaintext with the ciphertext, as seen above. Today, one-time pads are in use for military and diplomatic purposes when unconditional security is of the utmost importance.

On page 93, we introduced the autokey Vigenère cipher, which has a generalization that provides us with a nonsynchronous stream cipher.

Example 2.8 Let $n = |\mathcal{A}|$ where $\mathcal{A}$ is the alphabet of definition. We call

$$k_1 k_2 \cdots k_r \text{ for } 1 \leq r \leq n$$

a *priming key*. Then given a plaintext message unit

$$m = (m_1, m_2, \ldots, m_s) \text{ where } s > r,$$

we generate a keystream as follows.

$$k = k_1 k_2 \cdots k_r m_1 m_2 \cdots m_{s-r}.$$

Then we encipher via:

$$E_{k_j}(m_j) = m_j + k_k \pmod{n} = c_j, \text{ for } j = 1, 2, \ldots, r, \text{ and}$$

$$E_{k_j}(m_j) = m_j + m_{j-r} \pmod{n} = c_j \text{for } j > r,$$

and we decipher via:

$$D_{k_j}(c_j) = c_j - k_j \pmod{n} = m_j, \text{ for } j = 1, 2, \ldots, r, \text{ and}$$

$$D_{k_j}(c_j) = c_j - m_{j-r} \pmod{n} = m_j \text{ for } j > r.$$

This cryptosystem is nonsynchronous since the plaintext serves as the key, from the $(r+1)^{st}$ position onwards, with the simplest case being $r = 1$.

We have excluded $r = s$ in Example 2.8, since, in that case, we have a synchronous cipher that becomes the one-time pad if we assume the keystream is truly randomly generated and never used more than once.

Exercises

Using the autokey Vignère cipher described in Example 2.8 on the preceding page, decrypt the ciphertexts in Exercises 2.35–2.36. Use Table 2.2 on page 83 to get the numerical equivalents.

2.35. The priming key is $k = k_1k_2k_3 = 2, 7, 3$, $n = 26$, and the cryptogram is

VOHXTTMDTAFJLVJYBXZLX.

(*This and its conclusion in Exercise 2.36 comprise a quote from a speech given at Harvard on September 6, 1943, by* Winston Churchill (1874–1965), *British Conservative statesman and Prime Minister from* 1940 *to* 1945, *and* 1951 *to* 1955.)

2.36. The priming key $k = k_1k_2 = 3, 1$ and $n = 26$ and the cryptogram is

DSEKLXLQTUGMJSXHMXTMZL.

Exercises 2.37–2.38 are devoted to the following notion of a generator (*the general notion of which we will study in Section 2.4 along with linear feedback shift registers*). *Let* $a, b \in \mathbb{N}$ *with* $n \geq 2$ *and* $a, b \leq n - 1$. *Suppose that an integer* s_0 *is given with* $0 \leq s_0 \leq n - 1$, *called a* seed. *Define*

$$s_j \equiv as_{j-1} + b \,(\text{mod } n)$$

for $1 \leq j \leq \ell$ *where* $\ell \in \mathbb{N}$ *is the least value such that* $s_{\ell+1} = s_j$ *for some* $j \in \mathbb{N}$ *such that* $j \leq \ell$. *Then*

$$f(s_0) = (s_1, s_2, \ldots, s_\ell)$$

is called a linear congruential pseudorandom number generator — *or simply* linear congruential generator *where* a *is called the* multiplier, ℓ *is called the* period, *and* b *is called the* increment. *It can be shown* (see [46]) *that the maximum period* $\ell = n$ *is achieved if and only if* $\gcd(b, n) = 1$, $a \equiv 1 \,(\text{mod } p)$ *for all primes* $p \mid n$, *and* $a \equiv 1 \,(\text{mod } 4)$ *if* $4 \mid n$.

2.37. Find the maximum period of f, the above-described linear congruential generator, if $b = 0$.

2.38. Find the linear congruential generator f when $a = 23$, $s_0 = 1$, $b = 0$, and $n = 1806$.

2.4 LFSRs

The notion of a shift register is fundamental in cryptography since they provide a toolbox for generating pseudorandom numbers (see the discussion of randomness on page 109).

◆ Linear Feedback Shift Registers

A *linear feedback shift register* (LFSR) is described as follows.

First, there is a *shift register* of length $\ell \in \mathbb{N}$, consisting of a row of ℓ registers (memory cells), from left to right, labelled $R_{\ell-1}, R_{\ell-2}, \ldots, R_1, R_0$, each capable of holding one bit. Let k_0 be the binary value in the right-most register, then k_1, the value in the second register, and so on to the value $k_{\ell-1}$ as the value in the left-most register. Thus, the *initial state* would look like this.

$k_{\ell-1}$	$k_{\ell-2}$		k_1	k_0
$R_{\ell-1}$	$R_{\ell-2}$	$\cdots$	R_1	R_0

An electronic clock controls movement between the registers as follows. The first pulse of the clock causes the left-most entry, $k_{\ell-1}$, in register $R_{\ell-1}$, to get moved to register $R_{\ell-2}$, while the entry, $k_{\ell-2}$, in $R_{\ell-2}$ gets moved to occupy the register $R_{\ell-3}$, and so on until k_1 gets moved to R_0, and k_0 is *tapped* to the *output sequence*. This leaves the register $R_{\ell-1}$ blank. (For instance, $(1000) \to (\_100)$.) To fill this register, we require that it be a linear function of the values k_j for $j = 0, 1, \ldots, \ell - 1$. To achieve this we need a *tap sequence*, which is an ℓ-tuple of bits:

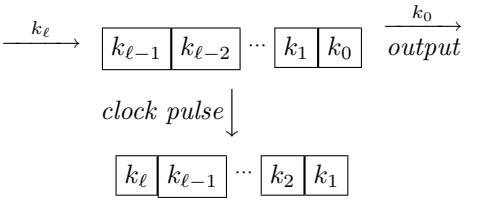
$$(c_{\ell-1} c_{\ell-2} \ldots c_1 c_0),$$

with $\mathbf{c_0 = 1}$. We form

$$k_\ell = \sum_{j=0}^{\ell-1} c_j k_j,$$

called the *linear feedback*, and place the entry k_ℓ in register $R_{\ell-1}$. The effect of this initial clock pulse is illustrated as follows.

Diagram 2.4 One Clock Pulse



A *state* s_m, of the LFSR is the bitstring describing the contents of the entire set of registers R_j after $m+1$ clock pulses where $m \geq 0$. The initial state, called the *seed*, which cannot be the zero vector, is given by the bitstring

$$s_0 = (k_{\ell-1}k_{\ell-2}\dots k_0),$$

and the state after one clock pulse is given by

$$s_1 = (k_\ell k_{\ell-1}\dots k_1).$$

In general, a state is given by

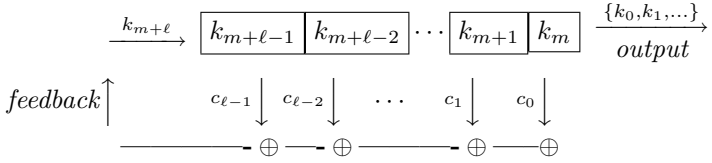
$$s_m = (k_{m+\ell-1}k_{m+\ell-2}\dots k_{m+1}k_m),$$

where $m \geq 0$, and the linear feedback is given by

$$k_{m+\ell} = \sum_{j=0}^{\ell-1} c_j k_{m+j}. \tag{2.1}$$

Equation (2.1) is also called a *binary recurrence relation of length* ℓ.
A generic LFSR that illustrates the above is given in Diagram 2.5.

Diagram 2.5 A Linear Feedback Shift Register



A very simple illustrating instance of the LFSR is given in the following.

Example 2.9 *Suppose that we have an LFSR with* $\ell = 4$*, tap sequence* $(c_3c_2c_1c_0) = (0101)$*, and initial state* $s_0 = (k_3k_2k_1k_0) = (1101)$*. Then we calculate the following.*

m	s_m	k_{m+3}	k_{m+2}	k_{m+1}	k_m
0	s_0	1	1	0	1
1	s_1	0	1	1	0
2	s_2	1	0	1	1
3	s_3	1	1	0	1

For instance, when $m = 2$*,*

$$k_{m+\ell} = k_6 = \sum_{j=0}^{3} c_j k_{j+2} = 1\cdot 1 + 0\cdot 1 + 1\cdot 0 + 0\cdot 0 = 1 = \sum_{j=0}^{\ell-1} c_j k_{j+m}.$$

The output sequence is $\{k_0, k_1, k_2\} = \{101\}$*. Notice that the state* s_3 *equals the state* s_0 *in the above table. This implies that we repeat the states again, which motivates the following.*

◆ Periodicity

How many *distinct* states, $s_m = (k_{m+\ell-1}k_{m+\ell-2}\ldots k_{m+1}k_m)$, is it possible to have? The total number of states (bitstrings) of length ℓ is 2^ℓ since there is a choice of either a 0 or a 1 in each register. This includes the zero bitstring. However, we do not want to include the zero bitstring since we cannot begin with it so it can never appear as a state, *given our requirement that* $\mathbf{c_0} = \mathbf{1}$ (see Exercise 2.39 on page 121). Thus, there are no more than $2^\ell - 1$ possible distinct states for an LFSR. Hence, it must be the case that after at most $2^\ell - 1$ clock pulses, there must be the repetition of a state. Suppose that P is the smallest such value where $s_0 = s_P$. Then we call P the *period length* of the LFSR. (See Exercise 2.41.) For instance, in Example 2.9 on the preceding page, $P = 3$, so the LFSR, depicted therein, has period length 3. In general, it follows that $s_{m+P} = s_m$ for *all* states s_m where $m = 0, 1, 2, \ldots$.

Example 2.9 is an instance where the maximum possible number of states, $2^\ell - 1 = 15$ in that case, is not reached. Those that do reach that maximum are members of a distinguished family.

◆ Maximum Length LFSRs — Pseudo-random Sequences

A *maximum-length LFSR* is one of period $2^\ell - 1$ where ℓ is the number of registers. Another name for them is *pn-sequences*, which was coined by S.W. Golomb because these sequences satisfy certain nice statistical properties, called *Golomb's Randomness Postulates* (see [38, pp. 43–48]). Fortunately, there is an abundance of such sequences. Peterson and Weldon have shown in [69] that for every $\ell \in \mathbb{N}$, there exists an LFSR of length ℓ that has period length $2^\ell - 1$.

Example 2.10 Let $\ell = 4$, tap sequence $(c_3c_2c_1c_0) = (1001)$, with initial state

$$s_0 = (0001) = (k_3k_2k_1k_0).$$

Then the output of the LFSR is given by the following.

m	k_{m+3}	k_{m+2}	k_{m+1}	k_m	n	k_{m+3}	k_{m+2}	k_{m+1}	k_m
0	0	0	0	1	8	1	0	1	0
1	1	0	0	0	9	1	1	0	1
2	1	1	0	0	10	0	1	1	0
3	1	1	1	0	11	0	0	1	1
4	1	1	1	1	12	1	0	0	1
5	0	1	1	1	13	0	1	0	0
6	1	0	1	1	14	0	0	1	0
7	0	1	0	1	15	0	0	0	1

The period length is $P = 15$ since $s_P = s_{15} = s_0 = (0001)$. Since $2^\ell - 1 = 15$, this is a maximum-length LFSR. The pn-sequence output by this LFSR is $\{k_0, k_1, \ldots, k_{13}, k_{14}\} = \{100011110101100\}$.

In Example 2.10, the output sequence has $2^{\ell-1} = 8$ ones and $2^{\ell-1} - 1 = 7$ zeros. In fact, it can be shown that this is always the case, namely a maximum-length LFSR having ℓ registers, has $2^{\ell-1}$ ones and $2^{\ell-1} - 1$ zeros (see Exercise 2.45 on page 121).

◆ Tap Polynomials

If L_ℓ is an LFSR with length ℓ and tap sequence $c_{\ell-1}c_{\ell-2}\dots c_1c_0$, where $c_0 = 1$, then

$$t(x) = x^\ell + c_{\ell-1}x^{\ell-1} + c_{\ell-2}x^{\ell-2} + \cdots + c_1x + c_0 \in (\mathbb{Z}/2\mathbb{Z})[x],$$

is called the *tap polynomial* for L_ℓ.

The tap polynomial is said to be *primitive* provided that $t(x)$ has no proper nontrivial factors and $t(x)$ does not divide $x^d + 1$ for any $d < 2^\ell - 1$. (Another equivalent means of defining $t(x)$ as a primitive polynomial is to say that its roots generate the group of nonzero elements of the associated extension field obtained from the polynomial $t(x)$ of degree ℓ.) With this notion in mind, it can be shown that L_ℓ is a maximum-length LFSR if and only if its tap polynomial $t(x)$ is primitive. For instance, in Example 2.10, the tap polynomial is x^4+x^3+1, which is irreducible (has no nontrivial factors) and does not divide $x^d + 1$ for any $d < 15$.

◆ LFSRs Via Matrices

There is a very palatable, simple, easy-to-understand matrix method of describing the above. Consider the following *tap matrix* derived from the tap sequence and *state matrix* derived from the states.

$$C = \begin{pmatrix} c_1 & c_2 & c_3 & \cdots & c_{\ell-1} & c_\ell \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \quad \text{and} \quad S_m = \begin{pmatrix} k_{m+\ell-1} \\ k_{m+\ell-2} \\ \vdots \\ k_{m+1} \\ k_m \end{pmatrix},$$

so,

$$CS_m = S_{m+1} \text{ for } m = 0, 1, \dots, P-1.$$

For instance, take the case in Example 2.9.

$$C = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \text{and} \quad S_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix},$$

so

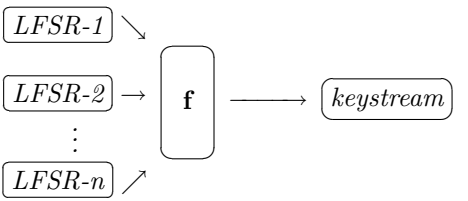
$$CS_2 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} = S_3 = S_P.$$

◆ Security of LFSRs

All a cryptanalyst needs is a few bits of consecutive plaintext and corresponding ciphertext, then by adding modulo 2, eventually the bits of the key are determined. Bitstrings output by a single LFSR are not secure since sequential bits are linear, so it only takes 2ℓ output bits of the LFSR to determine it, even if the feedback scheme is unknown to the cryptanalyst. Indeed, LFSRs are susceptible to what is called a *known-plaintext attack* (see Section 2.6).

The above being said, LFSRs are highly functional as building bricks for more secure systems. One mechanism is to use several of them in parallel, meaning that $n \in \mathbb{N}$ LFSRs are input to a function f, called the *combining function*, which outputs a keystream. Such a system is called a *nonlinear combination generator*.

Diagram 2.6 Nonlinear Combination Generator



The LFSR-based keystream should display reasonably good statistical properties and have a large period length. Also, they should satisfy certain complexity requirements involving the following.

◆ Linear Complexity and LFSRs

An LFSR is said to *generate* a binary sequence if there exists an initial state for which the sequence is the output of the LFSR. The *linear complexity* of a binary sequence $k = \{k_j\}$ is defined to be the length of the shortest LFSR that generates k. If there is no such LFSR that generates k, the linear complexity is said to be *infinite*. If k is the zero sequence, then the linear complexity is defined to be 0.

For instance, if $t(x)$ is an irreducible polynomial over $\mathbb{Z}/2\mathbb{Z}$ with degree ℓ and is the tap polynomial of an LFSR, then that LFSR has linear complexity ℓ for each of its $2^\ell - 1$ nonzero initial states.

A necessary (although possibly not sufficient) property for the security of an LFSR is that it should have large linear complexity. There is an efficient test for determining the linear complexity of a finite binary sequence, called the *Berlekamp-Massey Algorithm*, which shows that if $k = \{k_j\}_{j=1}^n$ is a binary sequence having linear complexity L, then there is a unique LFSR of length L that generates k if and only if $L \leq n/2$. (See [11, Theorem 16.22, p. 303].) It is also known that 2ℓ consecutive bits determine the output of an LFSR with ℓ states (see [11, Result 16.11, p. 298]).

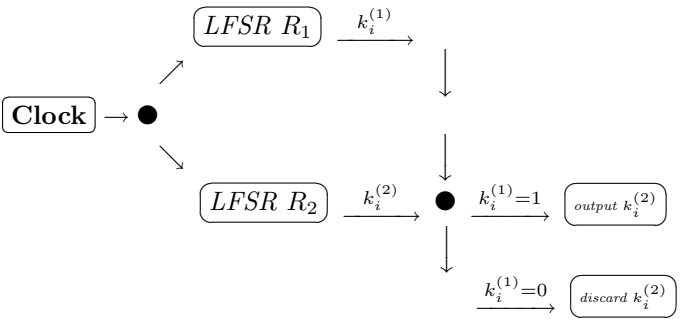
There exists a method of nonlinear combination generation called *clock-controlled generation* that attempts to foil attacks based upon the regular action

of LFSRs such as that described above. Clock control means that nonlinearity is introduced into LFSR-based keystream generators by having the output of one LFSR control the clocking (or stepping) of the LFSR. There is such a keystream generator that is secure if properly set up.

◆ The Shrinking Generator

The shrinking generator, proposed by Coppersmith, Krawczyk, and Mansour in 1993 (see [19]) is described as follows. Let R_1 and R_2 be two LFSRs. If $k_0^{(j)}, k_1^{(j)}, \ldots$ is the output bitstring of R_j for $j = 1, 2$, then the keystream output by the shrinking generator is $s_i = k_{i(1)}^{(2)}$, where $i(1)$ is the position of the i^{th} 1 in $k_0^{(1)}, k_1^{(1)}, \ldots$ for $i \geq 0$. In other words, if a 1 occurs in the i^{th} output position of R_1, then $k_i^{(2)}$ becomes the next bit in the output keystream. Otherwise, it is discarded. Hence, the output keystream is a "shrunken" version of the output of R_2. This is illustrated as follows.

Diagram 2.7



If $\gcd(L_1, L_2) = 1$, then the keystream has period $(2^{L_2} - 1)2^{L_1 - 1}$, and the linear complexity $L(k)$ of the output keystream k satisfies,

$$L_2 \cdot 2^{L_1 - 2} < L(k) < L_2 \cdot 2^{L_1 - 1}.$$

The most efficient attack on the shrinking generator takes $O(2^{L_1} \cdot L_2^2)$ steps, but this attack requires $2^{L_1} \cdot L_2$ consecutive bits from the output sequence. Thus, when R_1 and R_2 are chosen as maximum-length LFSRs with $\gcd(L_1, L_2) = 1$, then for $\max(L_1, L_2) \geq 64$, the shrinking generator appears to be secure against presently known attacks.

Although there exist numerous LFSRs and LFSR-based stream ciphers, for the purposes of this text, we have covered a sufficient amount. For further information, the reader is referred to [58].

◆ Summary

LFSRs are amenable to hardware implementations, the costs of which are low, and LFSRs are extremely fast. The choice of the bits to be tapped can

ensure a statistically random appearance of output bits. Thus, an LFSR can be used as a PRNG, but not a CSPRNG (see the discussion of randomness on page 109 and the discussion of security on page 119). Thus, LFSRs become a trade-off between speed and security. If one is not concerned with security, but rather speed, such as in cable television transmission, then LFSRs are a good bet. For systems of communications requiring high security, they are not. However, they can be built into *non*-linear PRNGs. For examples of this and a deeper insight into LFSRs, see [38] and [89, pp. 119–126] as well as [91] for a cryptanalytic perspective.

Exercises

2.39. Prove that an LFSR, as described in this section, can never have a zero state.

2.40. Suppose you are given the bitstring $B = \{k_0, k_1, \ldots, k_{\ell-1}\}$ of length ℓ, not equal to the zero bitstring. Prove that there is an LFSR whose output sequence contains B.

2.41. Show that periodicity of an LFSR fails to hold (in general) if we allow $c_0 = 0$. (*Hint: Show that Exercise 2.39 fails to hold in general if* $c_0 = 0$.)

2.42. Provide and example where periodicity for an LFSR *does hold* when $c_0 = 0$. (This merely shows that periodicity is not a general fact when $c_0 = 0$, but there are still instances where it can hold. This is counter to claims made in many texts [including the first edition of this one] that it is necessary *and* sufficient for periodicity of an LFSR that $c_0 = 1$. The condition is sufficient but is not necessary.)

2.43. Let $(k_1 k_2 \ldots k_{2n-1})$ be a bitstring that satisfies a linear recurrence of length less than $n \in \mathbb{N}$ (see Equation 2.1 on page 116). Prove that $\det(M) \equiv 0 \pmod 2$, where

$$M = \begin{pmatrix} k_1 & k_2 & k_3 & \cdots & k_{n-1} & k_n \\ k_2 & k_3 & k_4 & \cdots & k_n & k_{n+1} \\ k_3 & k_4 & k_5 & \cdots & k_{n+1} & k_{n+2} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ k_n & k_{n+1} & k_{n+2} & \cdots & k_{2n-2} & k_{2n-1} \end{pmatrix}.$$

2.44. Let an LFSR with five registers be given by the initial state $s_{-1} = (11101)$ and tap sequence $c = (01001)$. Calculate the output seqeunce to verify it is a maximum-length LFSR.

2.45. Show that a maximum-length LFSR with ℓ registers generates $2^{\ell-1}$ ones and $2^{\ell-1} - 1$ zeros.

2.5 Modes of Operation

Another cryptographic basic is how block ciphers (such as DES, which we will study in Section 3.1) may be applied to a variety of situations, called *modes of operation*.

Symmetric-key block ciphers have five *modes of operation* recommended by *National Institute of Standards and Technology* (NIST) — see the NIST homepage: *http://www.nist.gov/*. These modes (initially intended for DES) are meant to address every conceivable application for cryptology to which block ciphers can be applied.

The following is taken from [64].

◆ Block Cipher Modes — Overview

1. **Electronic Code Book (ECB)**: Each 64-bit block of plaintext is enciphered with the same key, albeit independently. This mode is typically used to send small amounts of data such as a symmetric key.

2. **Cipher Block Chaining (CBC)**: The input is the addition, modulo 2, of the previous 64 bits of ciphertext with the succeeding 64 bits of plaintext. Normally, this mode is used as a general-purpose block-transport mechanism but also may be employed for authentication purposes.

3. **Cipher Feedback (CFB)**: This mode employs a chaining mechanism similar to CBC. It uses prior ciphertext as input and outputs pseudo-random strings that are added, modulo 2, with plaintext to produce the next quantity of ciphertext. This mode is employed as a stream-cipher-oriented means for general-purpose messaging since it processes $n \in \mathbb{N}$ bits at a time.

4. **Output Feedback (OFB)**: This is comparable with CFB mode with the exception that its input is the prior block cipher's output. This mode is usually employed for stream-cipher-oriented communications, especially those requiring message authentication.

5. **Counter (CTR)**: The ciphertext is formed via a modulo 2 addition of a plaintext block with an enciphered counter, which is updated for each succeeding block. This mode is remarkably easy to use and is typically utilized for high-speed transmission. In fact, this is the least-known of the modes but is rapidly gaining ground with working cryptographers in the field as an excellent means of using block ciphers in a variety of situations.

◆ Block Cipher Modes — Details

In what follows E_k is the enciphering function for the block cipher E using the key k, whereas $D_k = E_k^{-1}$ denotes the decryption function.

◆ Electronic Code Book (ECB)

We begin with the simplest of the modes. In ECB mode, we input a sequence m_j for $j \geq 1$, of 64-bit plaintext blocks, each of which is enciphered with the same key, producing a string of ciphertext blocks c_j. In other words,

$$\text{enciphering is } E_k(m_j) = c_j \text{ and deciphering is } E_k^{-1}(c_j) = m_j.$$

The problem with this is that two identical plaintext blocks get sent to identical ciphertext blocks, which can be exploited by a cryptanalyst. Some experts feel that this weakness is sufficient to render it insecure for any use, whereas others feel that it is ideal for sending small amounts of data such as the sending of a DES key. It certainly should not be used for sending large amounts of data in any case. The aforementioned weakness of ECB is overcome in the next mode.

First we need to describe another type of attack called the *man-in-the-middle attack*. To describe this attack, we introduce a cryptographic cast of characters. First there are *Alice* and *Bob* who wish communicate with each other. Second, there is *Eve*, who is our *passive* eavesdropper. Third, there is *Mallory*, *the malicious active attacker*. The principal idea in the man-in-the-middle attack is that Mallory assumes a position between Alice and Bob. Mallory can stop all or parts of messages being sent between them and substitute his own data. In this way, he impersonates Alice and/or Bob, who believe they are communicating with each other while they are really talking to Mallory.

◆ Cipher Block Chaining (CBC)

In CBC mode, we first let IV be an initialization vector (meaning a 64-bit input bitstring), set $c_0 = IV$, and let k be the 64-bit input key. Given a sequence m_j of 64-bit plaintext blocks, for $j \geq 1$, we recursively define

$$\text{encryption by } c_j = E_k(c_{j-1} \oplus m_j), \text{ and decryption by } m_j = E_k^{-1}(c_j) \oplus c_{j-1}.$$

Thus, the weakness of ECB mode is eliminated by the modulo 2 addition of plaintext blocks with previous ciphertext blocks, thereby randomizing the plaintext with the previous ciphertext. Essentially, this means that we have "chained together" the sequence of enciphering plaintext blocks. This obscures the relationship between the plaintext and ciphertext, substantially reducing the data for a cryptanalyst to use effectively.

Next is the not-so-obvious problem of how to choose IV. Most texts recognize the problems with leaking information about IV and therefore suggest keeping it as secure as the key, since a cryptanalyst can derive information from it by posing as a sender using the man-in-the-middle attack. However, few cite the best solution to this problem. We should *not* have a *fixed* IV or even a randomized IV since there remains the problem (the one for which it is deemed necessary to keep IV a secret), namely, either method requires that the recipient of the message has to know this IV. In the case of a fixed IV we return to the ECB problem in encryption of the first block of each message. With the randomized IV, we require a secure randomizer at hand, for each message,

which adds more effort in the use of the cipher, since as we will discover later that obtaining secure randomizers is a difficult task. There is a better method, which essentially uses the idea behind the one-time-pad (see page 101).

First, a *nonce* is a unique number used *exactly once* in a given protocol. (This is derived from **n**umber used **once**.) As with the one-time-pad, a nonce should never be used *more* than once. In this fashion, we eliminate the need to keep the nonce secret. A *nonce-generated IV* is one where the IV is enciphered with the block cipher in CBC mode as follows.

1. Using a counter that starts at 0, assign a number to the message and use this number to generate a (unique) nonce.
2. Encipher the nonce with the block cipher, such as DES, to generate the IV.
3. Encipher the message in CBC mode using the IV.
4. Instead of sending $c_0 = IV$ as above, add the message number appended to the front of the ciphertext.
5. To ensure that there is a safeguard built in to guarantee the nonce is never accepted more than once by a recipient, the receiver will not accept messages with an assigned number less than or equal to the previously assigned message numbers.

If there were a popularity contest among the modes, CBC would probably win as the most utilized of them all. It certainly is an excellent all-purpose application for sending block data. However, others are gaining ground.

◆ Cipher Feedback (CFB)

In CFB mode, again we input IV, m_j as above, and set $c_0 = IV$. Then we produce subkeys by enciphering the previous ciphertext block. In other words, for $j \geq 1$,

$$E_k(c_{j-1}) = k_j, \text{ then produce ciphertext: } c_j = m_j \oplus k_j.$$

CFB encryption is similar to CBC encryption in that the chaining mechanism causes ciphertext block c_j to depend on m_k for $k \leq j$. Moreover, the same issues with the IV remain.

◆ Output Feedback (OFB)

In OFB mode, we input IV, k, m_j for $j \geq 1$ as above, and set $k_0 = IV$. Then subkeys are computed by repeatedly encrypting the initialization vector, in a mechanism described by the following.

OFB Feedback Mechanism

$$k_j = E_k(k_{j-1})$$

Then m_j is enciphered via

$$c_j = m_j \oplus k_j \text{ for } j \geq 1.$$

In ECB and OFB modes, changing one input block m_j causes *exactly one* ciphertext block c_j to be changed. This is valuable in such applications as the encryption of satellite transmissions. In CBC and CFB modes, a change to input block m_j changes $c_j, c_{j+1}, \ldots$. This turns out to be useful in applications involving message authentication. In other words, these latter two modes can be used to produce a *message authentication code* (MAC). What this means is that the MAC can be used as an *electronic signature* that will convince the receiving party of the authenticity of the message.

In OFB mode, the block cipher is used to generate a pseudorandom stream of keys. This is an example of a *keystream* (see Definition 2.5 on page 109). The IV has to be random, so it can either be chosen randomly or generated as a nonce as in CBC mode. Moreover, only the enciphering function is needed since enciphering is exactly the same method as deciphering. Also, since the keystream is generated in the above fashion, then there is no padding[2.2] required. In other words, one needs only send a ciphertext as long as the plaintext (and not have to pad to fill in the blocklength).

A major weakness of OFB mode is that if the same IV is ever used for two different messages, then a cryptanalyst (for example, Eve) can add ciphertext modulo 2 to recover plaintext. To see why, assume that c_i and c_j were enciphered using the same keystream, k_i. Then

$$c_i \oplus c_j = m_i \oplus k_i \oplus m_j \oplus k_i = m_i \oplus m_j,$$

and now Eve has a means of computing the difference between two plaintexts. This is a disaster if Eve knows one of the plaintexts already because then she readily gets the other. Moreover, even if she does not know either one, there are means of recovering both from information about the differences between them.

◆ Counter (CTR)

Counter mode (CTR) has been around since 1980 or so but was not standardized until December 2001 by NIST. Thus, it has not appeared in most textbooks as a mode of operation. However, it has recently been gaining in popularity and many consider it to be the best mode. As with OFB, it is a stream cipher, the methodology for which we now describe.

A nonce n is concatenated with the counter i and enciphered to form a single block of key for $i = 1, 2, \ldots$, $k_i = E_k(n, i)$; and ciphertext is obtained via $c_i = m_i \oplus k_i$ for given plaintext blocks m_i. Therefore, the counter and nonce must fit into a single block (for instance, a 128-bit block in most modern-day

[2.2] *Padding* means appending a randomly generated bitstring of suitable length to the plaintext prior to encryption, a practice also called *salting*, since we change the "taste" of the message, so the result is called a *salt*. Moreover, the random bitstring must be independently generated for each separate encryption.

ciphers would not present a problem). For reasons discussed in above modes, the nonce must be used exactly once for each plaintext block encrypted.

To decipher, the same set of nonce/counter concatenated values are used as follows to recover plaintext: For each $i = 1, 2, \ldots$, execute $k_i = E_k(n, i)$, then $m_i = c_i \oplus k_i$.

CTR does not suffer the problems cited for other modes because all the k_i are distinct since they are encipherings of a concatenation of nonce and counter, used only once. Then all plaintext m_i get enciphered via k_i to distinct ciphertext values, so two keyblocks (formed by the ciphertext values) are never the same.

CTR is an all-purpose block-oriented method that is highly useful for high speed transmissions, the reason being that the keystream can be paralleled to any desired level. The structure of CTR, moreover, ensures that its use is as secure as that of the underlying block structure.

CTR, as with OFB, does not require padding, whereas CBC does. CTR may, in fact, be considered to be a simplification of OFB, which solves one of the problems inherent in the latter. The counter replaces the feedback mechanism in OFB, discussed earlier, and this provides a formidable feature of CTR.

CTR Random Access Property
A ciphertext block c_j need not be deciphered in order to decipher c_{j+1}.

With the chaining modes such as CBC, one must decipher c_j in order to decipher c_{j+1}.

CTR, due to its high speed configurations, is used in network security applications, such as *IPSec*, or *IP security* (see [64, Section 8.3]). Another palatable feature of CTR is its simple structure in that, unlike ECB and CBC, CTR requires only the implementation of the enciphering scheme, not the deciphering algorithm. For instance, if the underlying block cipher were AES (see Section 3.2), this matters a lot since the encryption and decryption transformations differ so greatly. This simplifies matters since key scheduling for deciphering is not needed in the CTR implementation. Perhaps, from a security viewpoint, the greatest selling feature of CTR is that it is *provably* secure. For all these reasons, it appears that CTR is on its way to dominance as the mode of choice.

Exercises

2.46. Verify that the deciphering method for CBC mode described on page 123 actually recovers the plaintext.

2.47. Show that for the CFB mode described on page 124 the following decryption method will recover the plaintext:

$$m_j = c_j \oplus E_k(C_{j-1}).$$

2.48. Verify that the CTR random access property, highlighted on page 126, is indeed valid.

2.6 Attacks

Basically, an attack on a cryptosystem is any method that starts with some information about the plaintext and the corresponding ciphertext, enciphered under a key, which is yet unknown to the cryptanalyst. Determining the key and thus the plaintext in its entirety is the end goal. There are two major classes of attacks. One is *passive*, where the cryptanalyst monitors the channels of communications, thereby only *threatening* confidentiality of data. The other is *active*, where the cryptanalyst attempts to add, delete, or somehow alter the message, threatening not only confidentiality but also integrity and authentication of data.

◆ Passive Attacks

Passive attacks may be further subdivided into six classes.

(a) *Chosen-plaintext.* With this attack the cryptanalyst chooses plaintext, is then given corresponding ciphertext, and analyzes the data to determine the enciphering key to obtain plaintext from previously unseen ciphertext. An example of this type of attack is the DC attack against DES described on page 145.

(b) *Chosen-ciphertext.* In this form of attack the cryptanalyst chooses the ciphertext and is given the corresponding plaintext. Of course, given the required data, by their very nature such attacks are difficult to mount. For example, one way to mount a chosen-ciphertext attack is to gain access to the equipment used to encipher, as was the case, for instance, with the Americans gaining access to the Japanese cipher machines prior to World War II. Then the cryptanalyst can use such equipment to deduce plaintext from other intercepted ciphertext, since the equipment does not give the cryptanalyst the decryption key.

(c) *Known-plaintext.* This attack is more practical than chosen-plaintext since the cryptanalyst has some amount of both plaintext and corresponding ciphertext. Even a small amount of information in this type of attack may suffice to find the key. This type of attack was useful against the German Enigma machine in World War II, for instance.

(d) *Ciphertext-only.* Even more practical than the latter is this attack, wherein the cryptanalyst has only the ciphertext as data to deduce the key and subsequent plaintext. Any cryptosystem that is vulnerable to this type of attack is completely insecure.

(e) *Adaptive chosen-plaintext.* This is a chosen-plaintext attack where the choice of plaintext depends upon the previously received plaintexts.

(f) *Adaptive chosen-ciphertext.* This attack is a chosen-ciphertext attack with chosen ciphertext depending upon previously received ciphertexts.

There are certain attacks that we will encounter in the text that are described by name as follows.

◆ Brute-Force Attacks

In this type of attack, also called an *exhaustive serach of the keyspace*, all possible keys are tried to determine which one is being used by communicating parties. For a well-designed cryptosystem, this type of attack is too time consuming to undertake.

To understand next two types of attack, we must first define a couple of items. A *hash function* is a computationally efficient function that maps bitstrings of arbitrary length to bitstrings of fixed length, called *hash values*. A *one-way* hash function $f : \mathcal{M} \mapsto \mathcal{C}$ is a hash function that satisfies the property that $f(m)$ is "easy" to compute for all $m \in \mathcal{M}$, but for randomly chosen c in the image of f, finding an $m \in \mathcal{M}$ such that $c = f(m)$ is computationally infeasible, namely we can easily compute f, but it is computationally infeasible to compute f^{-1}. One-way hash functions are called *cryptographic hash functions* since these functions prevent unauthorized retrieval of the original bitstring. The process of using a hash function on a message is called *hashing the message*.

◆ Dictionary Attacks

Such an attack occurs when an adversary takes a list of probable passwords, hashes all the entries on the list, and compares this list with the list of actual enciphered passwords in an effort to find a match.

We need further data on hash functions to describe the next attack.

A hash function f is said to have a *collision* if $f(x_1) = f(x_2)$ for values $x_1 \neq x_2$. One can see that if the set of possible messages upon which f acts is much larger than the set hash values, called *message digests*, then there should be many examples where collisions occur. This fact is used in the next type of attack.

◆ The Birthday Attack

If we are given a hash function $f : \mathcal{M} \mapsto \mathcal{C}$ with $|\mathcal{C}| = n$ and $|\mathcal{M}| > n$, then there is at least one collision. Now we show how to find such collsions by describing a phenomenon that gives the birthday attack its name.

The following is taken from [64].

The Birthday Paradox: Suppose there are $n > 1$ balls in a container numbered from 1 to n inclusive. Also, let us assume that $m > 1$ balls are drawn one at a time, listed, and replaced each time (where $m < n$). What is the probability that one of the balls is drawn at least twice?

Let $P_j(n, m)$ be the probability that one ball is drawn at least j times. Then, we are seeking

$$P_2(n, m) = 1 - P_1(n, m).$$

To find $P_1(n, m)$, note that the probability that the second drawn ball is different from the first is $1 - 1/n$, the probability that the third ball is different from the first two is $1 - 2/n$, and so the probability that the first three balls are all

different is $(1 - 1/n)(1 - 2/n)$. Continuing in this fashion, we see that the probability that all of the m balls drawn are different is

$$P_1(n, m) = \prod_{j=1}^{m-1} \left(1 - \frac{j}{n}\right) = \frac{1}{n^{m-1}} \prod_{j=1}^{m-1} (n - j) = \frac{(n-1)(n-2)\cdots(n-m+1)}{n^{m-1}}.$$

Thus,

$$P_2(n, m) = 1 - \frac{(n-1)(n-2)\cdots(n-m+1)}{n^{m-1}}.$$

In particular, suppose we want to prove that in any room of twenty-three people, the probability that at least two of them have the same birthday is greater than 50%. From the above this is a fact since

$$P_2(365, 23) \approx 0.5072972343.$$

This phenomenon is called the *birthday paradox*.

The birthday paradox is a special case of the *occupancy problem*, which is given as follows. Suppose that a container has n balls numbered 1 through n inclusive. Again assume that m balls are drawn one at a time, listed, and replaced each time. Then the probability that exactly ℓ of the m balls are different for $1 \leq \ell \leq m$ is given by

$$\frac{1}{\ell!} \sum_{j=0}^{\ell} (-1)^{\ell - j} \binom{\ell}{j} j^m P_1(n, m),$$

where $\binom{\ell}{j}$ is the binomial coefficient. Now we return to the birthday attack that began this discussion. We initially asked how we can find a collision. From the above, the probability that there do not exist any collisions is

$$\prod_{j=1}^{m-1} (1 - j/n),$$

so

$$1 - x \approx e^{-x}$$

for small x values (such as ours). Hence, the probability of no collisions is

$$\prod_{j=1}^{m-1} (1 - \frac{j}{n}) \approx \prod_{j=1}^{m-1} e^{-j/n} = e^{-m(m-1)/(2n)}.$$

Therefore, the probability of at least one collision occurring is

$$p_c \approx 1 - e^{-m(m-1)/(2n)}. \tag{2.2}$$

Since

$$e^{-m(m-1)/(2n)} \approx 1 - p_c,$$

then

$$-m(m-1)/(2n) \approx \ln(1-p_c).$$

Hence,

$$m^2 - m \approx -2n\ln(1-p_c),$$

and so

$$m^2 \approx -2n\ln(1-p_c) \approx 2n\ln(1/(1-p_c)),$$

since we can safely ignore the smaller factor of $-m$ in an approximation, so

$$m \approx \sqrt{2n\ln(1/(1-p_c))}.$$

If $p_c = 1/2$, then $m \approx 1.17\sqrt{n}$. Clearly then, by hashing over little more than $\sqrt{n}$ random elements of $\mathcal{S}$, we have a greater than 50% chance of finding a collision. This is the birthday attack.

The birthday attack places a lower bound on the number of bits a hash function should have in order to be secure. The reason is that the birthday attack can find a collision in $O(2^{k/2})$ hashings on an k-bit function. Thus, if $k = 64$, then it is not secure against the birthday attack since only 2^{32} hashings are required.

The following illustration of the birthday attack was first presented by Yuval in 1979 (see [98]).

● **Alice Cheats Bob Using the Birthday Attack**

The hash function has 64 bits. Alice wants Bob to sign a contract that he thinks will benefit him, and later she wants to "prove" that he signed a contract that actually robs him of his life savings.

(1) Alice prepares two contracts, one that is "good" for Bob, C_G, and one, C_B, that will sign away his savings.

(2) Alice makes very minor changes in each of C_G and C_B. Then she hashes 2^{32} modified versions of C_G and 2^{32} modified versions of C_B.

(3) She compares the two sets of hash values until she finds a collision $h(C_G) = h(C_B)$ and recovers the corresponding preimages.

(4) Alice has Bob sign C_G via the hash of its value.

(5) Later Alice substitutes C_B for C_G whose hash value is the same as that signed by Bob, who has now lost all his money.

From the discussion preceding the Yuval attack above, we see that a birthday attack requires an effort of only the order of 2^{32}. Thus, simple hash functions based on a 64-bit message digest are insecure, so from a cryptographic perspective they are not worth discussing.

Chapter 3

DES and AES

3.1 S-DES and DES

In order to properly describe the DES cryptosystem, we need more mathematically oriented definitions of transposition and substitution ciphers than we have considered thus far.

Block ciphers, defined on page 91, are classically broken into two types, the first of which is described as follows. Note that, in the following, if $\mathcal{S}$ is a finite set, then a *permutation* on $\mathcal{S}$ is a bijection $\sigma : \mathcal{S} \mapsto \mathcal{S}$.

Definition 3.1 **Transposition/Permutation Ciphers**

A simple transposition cipher*, also known as a* simple permutation cipher*, is a symmetric-key block cryptosystem having blocklength $r \in \mathbb{N}$, with keyspace $\mathcal{K}$ being the set of permutations on $\{1, 2, \ldots, r\}$. The enciphering transformation is defined, for each $m = (m_1, m_2 \ldots, m_r) \in \mathcal{M}$, and given $e \in \mathcal{K}$, by*

$$E_e(m) = (m_{e(1)}, m_{e(2)}, \ldots, m_{e(r)}),$$

and for each $c = (c_1, c_2, \ldots, c_r) \in \mathcal{C}$,

$$D_d(c) = D_{e^{-1}}(c) = (c_{d(1)}, c_{d(2)}, \ldots, c_{d(r)}).$$

The cryptosystems in Definition 3.1 have keyspace of cardinality $|\mathcal{K}| = r!$. Permutation encryption involves grouping plaintext into blocks of r symbols and applying to each block the permutation e on the numbers $1, 2, \ldots, r$. In other words, the places where the plaintext symbols sit are permuted. This gives mathematical rigour to the discussion of transpositions given on page 81. Note that the enciphering key e implicitly defines r, since it is a permutation on r symbols. In the following we are going to simplify the notation for permutations displayed on page 81. In Example 3.1 on the next page, the permutation $e = (1, 2, 3, 6, 5, 4)$ means that e sends: $1 \mapsto 2$, $2 \mapsto 3$, $3 \mapsto 6$, $4 \mapsto 1$, $5 \mapsto 4$, and

$6 \mapsto 5$. (Note that the permutation on page 81 is written, in terms of this new succinct notation, as $(5, 10, 6, 7, 8, 9)$, where the omission of $1, 2, 3, 4, 11, 12, 13$ mean that those values remain fixed.)

Example 3.1 Let $r = 6$, $\mathcal{M} = \mathcal{C} = \mathbb{Z}/26\mathbb{Z}$, with the English letter equivalents given by Table 2.2 on page 83. Then if $e = (1, 2, 3, 6, 5, 4)$ is applied to *strong*, we get $TRGSON$ since $m_1 = s$, $m_2 = t$, $m_3 = r$, $m_4 = o$, $m_5 = n$, and $m_6 = g$, so

$$e(m) = (m_{e(1)}, m_{e(2)}, m_{e(3)}, m_{e(4)}, m_{e(5)}, m_{e(6)}) = (m_2, m_3, m_6, m_1, m_4, m_5).$$

Since the inverse transformation is $d = e^{-1} = (4, 1, 2, 5, 6, 3)$, then another way to visualize encryption is to write $4, 1, 2, 5, 6, 3$ in the first row, and the plaintext letter equivalents in the second row, then read the letters off in *numerical order*. For instance,

$$\begin{pmatrix} 4 & 1 & 2 & 5 & 6 & 3 \\ s & t & r & o & n & g \end{pmatrix}.$$

Thus, the first ciphertext in numerical order is T, the second is R, and so on.

An easy means for finding the inverse of a given key e such as in Example 3.1 is given as follows. The key in that example can be written, as we showed on page 81, in the form

$$e = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 3 & 6 & 1 & 4 & 5 \end{pmatrix},$$

since $1 \mapsto 2$, $2 \mapsto 3$, and so on. To find the inverse, just read off in numeric order (determined by the second row) the terms in the first row. For instance, the term in the first row sitting above the 1 is 4, so 4 is the first term in e^{-1}. The term in the first row sitting above the 2 is 1, so 1 is the second term in e^{-1}, and so on. Permutation ciphers are subject to cryptanalysis by frequency analysis since they preserve the frequency distribution of each character.

Now we provide a more rigorous definition than that given on page 81 for the substitution cipher and on page 87 for monoalphabetic and polyalphabetic ciphers.

Definition 3.2 **Substitution Ciphers**

Let $\mathcal{A}$ be an alphabet of definition consisting of n symbols, and let $\mathcal{M}$ be the set of all blocks of length r over $\mathcal{A}$. The keyspace $\mathcal{K}$ will consist of all ordered r-tuples $e = (\sigma_1, \sigma_2, \ldots, \sigma_r)$ of permutations σ_j on $\mathcal{A}$. For each $e \in \mathcal{K}$, and $m = (m_1, m_2, \ldots, m_r) \in \mathcal{M}$ let

$$E(m) = (\sigma_1(m_1), \sigma_2(m_2), \ldots, \sigma_r(m_r)) = (c_1, c_2, \ldots, c_r) = c \in \mathcal{C},$$

and for $d = (d_1, d_2, \ldots, d_r) = (\sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_r^{-1}) = \sigma^{-1}$,

$$D_d(c) = (d_1(c_1), d_2(c_2), \ldots, d_r(c_r)) = (\sigma_1^{-1}(c_1), \sigma_2^{-1}(c_2), \ldots, \sigma_r^{-1}(c_r)) = m.$$

This type of cryptosystem is called a substitution cipher. *If all keys are the same, namely,* $\sigma_1 = \sigma_2 = \cdots = \sigma_r$, *then this cryptosystem is called a* simple substitution cipher *or* monoalphabebetic substitution cipher. *If the keys differ, then it is called a* polyalphabetic substitution cipher.

Simple substitution ciphers suffer from the inherent weakness that a so-called *frequency analysis* can be done on the ciphertext. In other words, a block cipher does not change the number of times that a letter appears in the plaintext. For instance, suppose that the letter E is the most frequently occurring letter in the English language, and we encipher the letter as F. Then the letter that occurs most often in the ciphertext will be F. Thus, by looking at a relatively small amount of ciphertext, a cryptanalyst can recover the key. Polyalphabetic ciphers do not suffer from this weakness since frequencies of symbols are not preserved in the ciphertext. However, polyalphabetic ciphers are not that much more difficult to cryptanalyze since, once the block size r is determined, the ciphertext symbols can be separated into r groups and a frequency analysis can be performed on each group.

The best-known symmetric-key block cipher, (now replaced by the Advanced Encryption Standard that we will study in Section 3.2), is the *Data Encryption Standard* (DES), which was the first commercially available algorithm (namely for use with unclassified computer data) put into use in the 1970's. A complete description of DES is given in the U.S. Federal Information Processing Standards Publication number 46 (FIPS-46), Springfield, Virginia, April 1977.

We begin with an informal overview to describe the mechanisms behind S-DES, which is a simplified version of DES, presented for pedagogical purposes. Although DES reached the end of its cryptographic usefulness by the beginning of the twenty-first century, it is valuable to look at its design and implementation in order to understand how it stood the test for roughly a quarter century before the cryptanalytic onslaught brought on by new mathematical and computational power caused it to fall from grace. The following is taken from [64].

◆ Overview of S-DES

As with any cipher, the encryption function takes the plaintext m and the key k as input. For S-DES, m has bitlength 8 and k has bitlength 10.

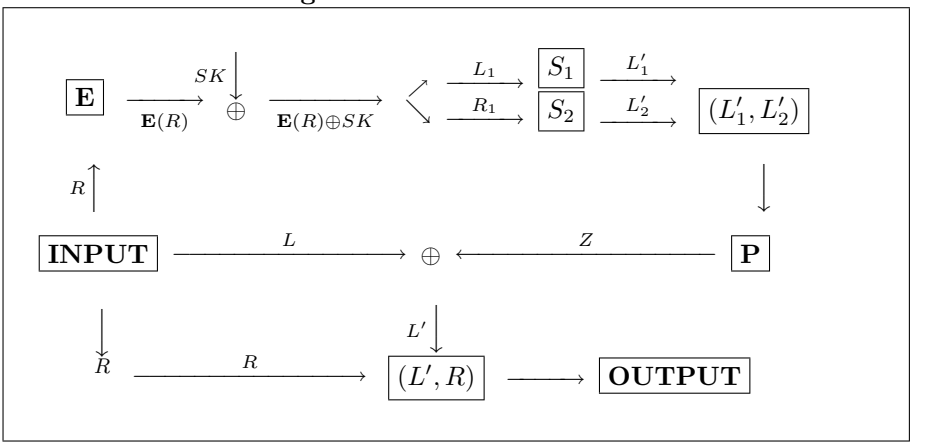
First, m is put through what is called an initial permutation **IP**, followed by two rounds of the same function (described below), which uses both permutation and substitution in its execution, the first round followed by a swap of the left and right 4 bits of the output. The 8-bit output of round two is put through the inverse permutation $\mathbf{IP^{-1}}$ to form the 8-bit ciphertext.

Each round of S-DES is described as follows. The 8-bit input is split into left and right 4-bit blocks, L and R. Then there is an expansion of R to 8 bits via an expansion permutation **E**, to get $\mathbf{E}(R)$. The 8-bit result, $\mathbf{E}(R)$, is added modulo 2, denoted by $\oplus$, to an 8-bit subkey SK, generated from k in a separate S-DES key generation stage. The resulting 8-bit output $E(R) \oplus SK$ is separated into left and right 4-bit strings, L_1 and R_1, which are fed into two separate substitution boxes, S_1 and S_2, respectively, called *S-boxes* (described

in detail below), which are publicly known lookup tables that take 4-bit inputs and output 2-bit strings, L_1' and L_2'. The resulting 4-bit string, (L_1', L_2'), is put through a permutation **P** to produce a 4-bit output Z. Last, Z is added modulo 2 with L to form L', and (L', R) is the output of the round.

All of the above is illustrated in Diagram 3.1, which is a single round of the S-DES cipher. Then, before giving a detailed description of S-DES that will extrapolate the above to a full explanation of all the detailed features of the (simplified) cipher, we look at the motivations behind the design of DES itself.

Diagram 3.1 An S-DES Round



◆ DES Design Principles

In a 1994 publication, [18], Coppersmith described the criteria used in the design of DES. The focus is principally upon the design of the S-Boxes and the permutation function that processes their outputs.

One important aspect of block ciphers, especially DES, that requires elucidation is the notion of *linearity*. A *linear cipher* is one for which each output bit is a linear combination of the input bits. An example of such a cipher is the Hill cipher discussed on page 100. The Hill cipher is easily broken with a known-plaintext attack (see page 127). The reason is that since a key matrix e acts upon a plaintext matrix m to produce a ciphertext matrix c via $c = me$, then this can be analyzed in such a fashion that ultimately an inverse matrix m^{-1} can be found, so that

$$e = m^{-1}c$$

and the key is recovered.

The only *non*-linear aspect of DES are the S-Boxes. Hence, an inherent design of DES stipulates that no output bit of an S-Box can be a linear function of the input bits. If they were, then the entire cryptosystem would be linear and could be broken with a known-plaintext attack.

Now we list the principles that were revealed by Coppersmith in his article, which concentrated upon the S-Boxes and their output. Thus, ensuring non-linearity was the key to ensuring that the cryptosystem could not easily be broken.

1. Linearity in the S-Box construction must be avoided. In other words, no bit output by an S-Box is allowed to be anywhere near a linear function of the input bits.
2. Each row of an S-Box should include all possible output bit combinations.
3. If two inputs to an S-Box differ in precisely one bit, or by exactly two middle bits, then the outputs must differ in a minimum of two bits.
4. If two inputs to an S-Box differ in their first two bits but have identical last two bits, the two outputs must be distinct.
5. There are other criteria such as 2–4, that were designed to thwart differential cryptanalysis and pertain primarily to the permutations that take the outputs of the S-Boxes. Since these criteria are very technical, we do not go into the details for the sake of efficiency. The reader may consult Coppersmith's paper [18] directly for the specifics, if necessary.

Now, we are ready for a detailed description of S-DES. First, recall our discussion and notation for permutations given at the outset of this section. The enciphering and deciphering in S-DES requires several basic components. We begin with two of them that are permutations.

◆ Initial Permutation

Let $m = (m_1m_2m_3m_4m_5m_6m_7m_8)$ be the byte of plaintext input. Then the initial permutation **IP** acts according to the following transposition of places where the plaintext sits, namely, **IP** retains all the plaintext bits but merely permutes them according to the rule given below.

IP								
j	1	2	3	4	5	6	7	8
IP(j)	2	6	3	1	4	8	5	7

Therefore, the action of **IP** on x is given in the following.

IP								
j	1	2	3	4	5	6	7	8
$m_{\mathbf{IP}(j)}$	m_2	m_6	m_3	m_1	m_4	m_8	m_5	m_7

Hence, $\mathbf{IP}(m) = (m_2m_6m_3m_1m_4m_8m_5m_7)$. For instance, if $m = (10010111)$, then $\mathbf{IP}(m) = (01011101)$.

The next component is also a permutation used at various stages of S-DES.

◆ Expansion Permutation

This permutation, denoted by **EP**, takes a *bitstring* (binary number) of length 4 (its *bitlength*) and expands it into a byte according to the following.

EP								
j	1	2	3	4	5	6	7	8
$\mathbf{EP}(j)$	4	1	2	3	2	3	4	1

For instance, if $x = (x_1x_2x_3x_4)$ is the input, then the following table gives us the action of **EP** on it.

EP								
j	1	2	3	4	5	6	7	8
$x_{\mathbf{EP}(j)}$	x_4	x_1	x_2	x_3	x_2	x_3	x_4	x_1

Hence, $\mathbf{EP}(x) = (x_4x_1x_2x_3x_2x_3x_4x_1)$. For example, if $x = (1001)$, then $\mathbf{EP}(x) = (11000011)$.

A very important aspect of S-DES is the key schedule. In other words, we need to understand how the keys are used and generated in the cipher.

◆ S-DES Key Generation

S-DES uses a 10-bit secret (shared) symmetric key

$$k = (e_1e_2e_3e_4e_5e_6e_7e_8e_9e_{10}),$$

say, and employs k to generate two 8-bit (sub)keys for deployment at various stages of the encryption and decryption process. Here is how that is accomplished.

First, a permutation $\mathbf{P}_{10}$ is applied to k according to the following.

$\mathbf{P_{10}}$										
j	1	2	3	4	5	6	7	8	9	10
$\mathbf{P_{10}}(j)$	3	5	2	7	4	10	1	9	8	6

Thus, $\mathbf{P_{10}}(k) = (e_3e_5e_2e_7e_4e_{10}e_1e_9e_8e_6)$.

Secondly, there is a circular left shift of one place, denoted by **LS1**, on each of the left five bits and the right five bits, as follows. $\mathbf{LS1}(e_3e_5e_2e_7e_4) = (e_5e_2e_7e_4e_3)$, and $\mathbf{LS1}(e_{10}e_1e_9e_8e_6) = (e_1e_9e_8e_6e_{10})$. Hence, under this shifting process, $(e_3e_5e_2e_7e_4e_{10}e_1e_9e_8e_6)$ becomes

$$(e_5e_2e_7e_4e_3e_1e_9e_8e_6e_{10}). \tag{3.1}$$

Then we apply yet another permutation called $\mathbf{P_8}$, which selects 8 of the 10 bits and permutes them as follows.

$\mathbf{P_8}$								
j	1	2	3	4	5	6	7	8
$\mathbf{P_8}(j)$	6	3	7	4	8	5	10	9

Applying $\mathbf{P_8}$ to (3.1) yields

$$\mathbf{P_8}(e_5e_2e_7e_4e_3e_1e_9e_8e_6e_{10}) = (e_1e_7e_9e_4e_8e_3e_{10}e_6) = k_1,$$

where k_1 is now our first subkey for use later.

Now, we return to (3.1) and perform a left shift of two places, denoted by **LS2**, on both the left and right 5-bit pieces to get

$$\mathbf{LS2}(e_5e_2e_7e_4e_3) = (e_7e_4e_3e_5e_2), \text{ and } \mathbf{LS2}(e_1e_9e_8e_6e_{10}) = (e_8e_6e_{10}e_1e_9),$$

yielding $(e_7e_4e_3e_5e_2e_8e_6e_{10}e_1e_9)$ to which we apply $\mathbf{P_8}$ to get

$$\mathbf{P_8}(e_7e_4e_3e_5e_2e_8e_6e_{10}e_1e_9) = (e_8e_3e_6e_5e_{10}e_2e_9e_1) = k_2,$$

where k_2 is our second subkey for use in the S-DES cipher.

The next essential component of the S-DES cryptosystem is an important method of substitution, and an innovation of Feistel (see Biography 3.1 on the next page) in his development of the original DES.

◆ S-Boxes

An S-Box or *substitution box* for S-DES is a four-by-four matrix with entries from $\mathbb{Z}/4\mathbb{Z}$ (put into binary) with rows and columns labelled from 0 to 3 (put into binary) that takes a 4-bit input and outputs a 2-bit string as follows.

If $(x_1x_2x_3x_4)$ is the input, then the output is given by one of the two S-Boxes used in S-DES, defined as follows.

$\mathbf{S_0}$		x_2	0	0	1	1
		x_3	0	1	0	1
x_1	x_4					
0	0		01	00	11	10
0	1		11	10	01	00
1	0		00	10	01	11
1	1		00	01	11	10

and

$\mathbf{S_1}$		x_2	0	0	1	1
		x_3	0	1	0	1
x_1	x_4					
0	0		00	01	10	11
0	1		10	00	01	11
1	0		11	00	01	10
1	1		10	01	00	11

Thus, for example if $x = (x_1x_2x_3x_4) = (1101)$ is our input bitstring of length 4, then if we wish to employ the first S-Box, we get $\mathbf{S_0}(1101) = (11)$, since $(x_1x_4) = (11)$ represents the fourth row, and $(x_2x_3) = (10)$ represents the third column, the entry at the intersection of which is 11. Similarly, if we want to use the S-Box $\mathbf{S_1}$, then $\mathbf{S_1}(1101) = (00)$.

Biography 3.1 Horst Feistel *may be considered to be one of the early pioneers in the drive to secure privacy for the public at large using cryptography. Born in Germany in 1914, he emigrated to the U.S.A. in 1934 but would not obtain a U.S. citizenship for another decade. In fact, in 1941, with Germany having declared war on America, he was placed on a (sort of) house arrest, where his movements were restricted to the Boston area where he lived. Yet, surprisingly, on January 31, 1944, the house arrest was lifted, he was granted U.S. citizenship, and the very next day he was given security clearance that allowed him to work at the Air Force Cambridge Research Center (AFCRC). (There is speculation that something may have been going on behind the scenes between Feistel and the U.S. government. See Levy's excellent book* Crypto [55] *for an account of some of these possible scenarios as well as with other related cryptographic activities.) There he set up a cryptography research group that developed some outstanding cryptographic algorithms. In particular, they developed the* MARK XII*, which is widely used in American aircraft. It is known that the NSA had an ambivalent attitude toward Feistel's group. On the one hand, they exerted pressure to steer his work, while at the same time they considered his group to be a threat. Consequently his group was dissolved in the late 1950's. Then Feistel moved to MIT's Lincoln Laboratory, followed by a move to MITRE Corporation, a spinoff of the MIT lab. When he tried to form a cryptography group there, again NSA exerted pressure on MITRE, so his efforts failed, and his group did not materialize.*

The mathematician A.A. Albert, a friend of Feistel, advised him to go to IBM, since they were hiring the brightest scientists to do their own innovative work, a kind of think tank. Feistel began work at their Watson Laboratory in Yorktown Heights, New York. There he created a cryptosystem used in the IBM2984 banking system, known today as the Alternative Encryption Technique*, but then it was called* Lucifer*. Years later, Feistel said that if it had not been for the Watergate scandal that rocked Washington, the NSA would probably have shut down the Lucifer project, as they had so many of his earlier efforts. In fact, in the early 1970's, patent secrecy orders were placed on some of Feistel's inventions by the U.S. government. Lucifer, which used a 128-bit key, was the predecessor of DES. However, the NSA did not want such a strong cipher in public hands, so by the time DES was released the keylength had been cut to 56 bits, less than half of that used by Lucifer.*

Perhaps the most complicated part of S-DES is the function that does the combining of permutation and substitution.

◆ The S-DES Round Function

First, we need to describe a mapping F that takes bitstrings of length 4 using a subkey SK and outputs bitstrings of length 4.

Let $x = (x_1x_2x_3x_4)$ be the input. Then F first uses the expansion **EP** to produce $\mathbf{EP}(x)$, as described on page 136. Then this 8-bit result is added to

the subkey SK, modulo 2. Thus, this result is denoted by

$$\mathbf{EP}(x) \oplus SK = (y_1y_2y_3y_4y_5y_6y_7y_8) = y.$$

For the sake of convenience, we will denote the left four bits of a given byte, such as y, by

$$\mathbf{L}(y) = (y_1y_2y_3y_4) \text{ and the right four bits by } \mathbf{R}(y) = (y_5y_6y_7y_8).$$

The next action of F is to feed $\mathbf{L}(y)$ into $\mathbf{S_0}$ to produce $\mathbf{S_0}(\mathbf{L}(y)) = (z_1z_2)$ and feed $\mathbf{R}(y)$ into $\mathbf{S_1}$ to get $\mathbf{S_1}(\mathbf{R}(y)) = (z_3z_4)$. Thus, under this action y gets sent to $(\mathbf{S_0}(\mathbf{L}(y))\mathbf{S_1}(\mathbf{R}(y))) = (z_1z_2z_3z_4) = z$. Next, we apply the following permutation to z.

$\mathbf{P_4}$				
j	1	2	3	4
$\mathbf{P_4}(j)$	2	4	3	1

Therefore, we get

$$\mathbf{P_4}(z) = (z_2z_4z_3z_1) = Z,$$

which is the final outcome for F, namely,

$$F(x, SK) = Z.$$

Now, the definition of the round function, denoted by f_{SK}, which takes an 8-bit plaintext t and a subkey SK, is given as follows.

$$f_{SK}(t) = (\mathbf{L}(t) \oplus F(\mathbf{R}(t), SK), \mathbf{R}(t)).$$

Thus, the round function alters only $\mathbf{L}(t)$, the left four bits of t, leaving $\mathbf{R}(t)$ unaltered. However, there is a reason for f_{SK} being called a round function, since there are two rounds. The next mechanism, the penultimate one, is a means of swapping left and right bits.

◆ The Switch/Swap Function

The switch function, denoted by **SW**, merely exchanges the left and right four bits of an input m. Hence, if $m = (\mathbf{L}(m), \mathbf{R}(m))$ is an 8-bit input, then

$$\mathbf{SW}(m) = (\mathbf{R}(m), \mathbf{L}(m)).$$

The last aspect of S-DES is the inverse of the initial permutation.

◆ The Inverse of IP

The inverse of **IP**, naturally denoted by $\mathbf{IP}^{-1}$, is given by the following.

$\mathbf{IP}^{-1}$								
j	1	2	3	4	5	6	7	8
$\mathbf{IP}^{-1}(j)$	4	1	3	5	7	2	8	6

(Recall the easy method for finding the inverse of a permutation given on page 132.)

Now, we are in a position to describe the totality of the S-DES cipher.

◆ The S-DES Cryptosystem

Given a 10-bit key k and an 8-bit plaintext m, to encipher, we execute the following.

◆ S-DES Encryption

1. Apply **IP** to m.
2. Apply f_{k_1} to the output from step 1. (This is round 1.)
3. Apply **SW** to the output of step 2.
4. Apply f_{k_2} to the output of step 3. (This is round 2.)
5. Apply $\mathbf{IP}^{-1}$ to the output of step 4.

Hence, the plaintext 8-bit message unit m gets sent to the 8-bit ciphertext message unit c, the output of step 5, under this sequence of steps of the S-DES cipher. To decrypt, we perform the following.

◆ S-DES Decryption

1. Apply **IP** to c.
2. Apply f_{k_2} to the output from step 1. (This is round 1.)
3. Apply **SW** to the output of step 2.
4. Apply f_{k_1} to the output of step 3. (This is round 2.)
5. Apply $\mathbf{IP}^{-1}$ to the output of step 4.

The following is derived from Ed Schaefer, the creator of S-DES, [82].

Example 3.2 *Suppose we are given plaintext bitstring* $m = (10100101)$ *and key bitstring* $k = (0010010111)$. *First we generate our subkeys as follows.*

1. $\mathbf{P_{10}}(k) = 1000010111$.
2. $\mathbf{LS1}(10000) = (00001)$ *and* $\mathbf{LS1}(10111) = (01111)$.

3. $\mathbf{P_8}(0000101111) = (00101111) = k_1$.

4. $\mathbf{LS2}(00001) = (00100)$ *and* $\mathbf{LS2}(01111) = (11101)$ (*applying* **LS2** *to the output of step 2*).

5. $\mathbf{P_8}(0010011101) = (11101010) = k_2$(*applying* $\mathbf{P_8}$ *to the output of step 4*).

Now we encrypt as follows. First we calculate $\mathbf{IP}(m) = (01110100)$. *Then we need to calculate the round function for the first round* $f_{k_1}(01110100) = (\mathbf{L}(01110100) \oplus F(\mathbf{R}(01110100), k_1), \mathbf{R}(01110100))$. *We do this as follows.*

1. $\mathbf{EP}(0100) = (00101000)$.

2. $\mathbf{EP}(0100) \oplus k_1 = (00101000) \oplus (00101111) = (00000111)$.

3. $\mathbf{S_0}(0000) = (01)$ *and* $\mathbf{S_1}(0111) = (11)$.

4. $\mathbf{P_4}(0111) = (1110) = F(\mathbf{R}(01110100), k_1)$.

5. $\mathbf{L}(01110100) \oplus F(\mathbf{R}(01110100), k_1) = (0111) \oplus (1110) = (1001)$.

6. $f_{k_1}(01110100) = (10010100)$.

Now we apply the switch function, $\mathbf{SW}(10010100) = (01001001)$. *The reader may now verify the second round, namely,*

$$f_{k_2}(01001001) = (\mathbf{L}(01001001) \oplus F(\mathbf{R}(01001001), k_2), \mathbf{R}(01001001)) = (01101001).$$

Last, we apply the inverse of the initial permutation, $\mathbf{IP}^{-1}(01101001) = (00110110)$, *which is the ciphertext.*

To decrypt, we reverse the process. First feed c *into* **IP** *to get*

$$\mathbf{IP}(c) = (01101001),$$

then apply f_{k_2} *to get* (*with the reader filling in the details*),

$$f_{k_2}(0110 \oplus F(1001, k_2), 1001) = (01001001).$$

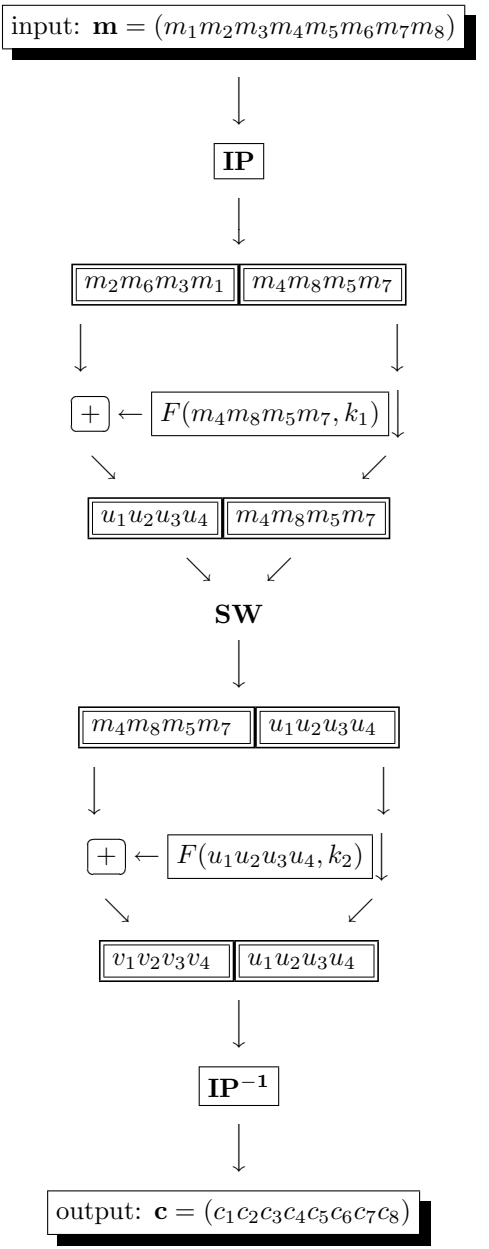
Then $\mathbf{SW}(01001001) = (10010100)$. *Next,*

$$f_{k_1}(1001 \oplus F(0100, k_1), 0100) = (01110100),$$

then the final application yields the original plaintext, $\mathbf{IP}^{-1}(01110100) = (10100101) = m$.

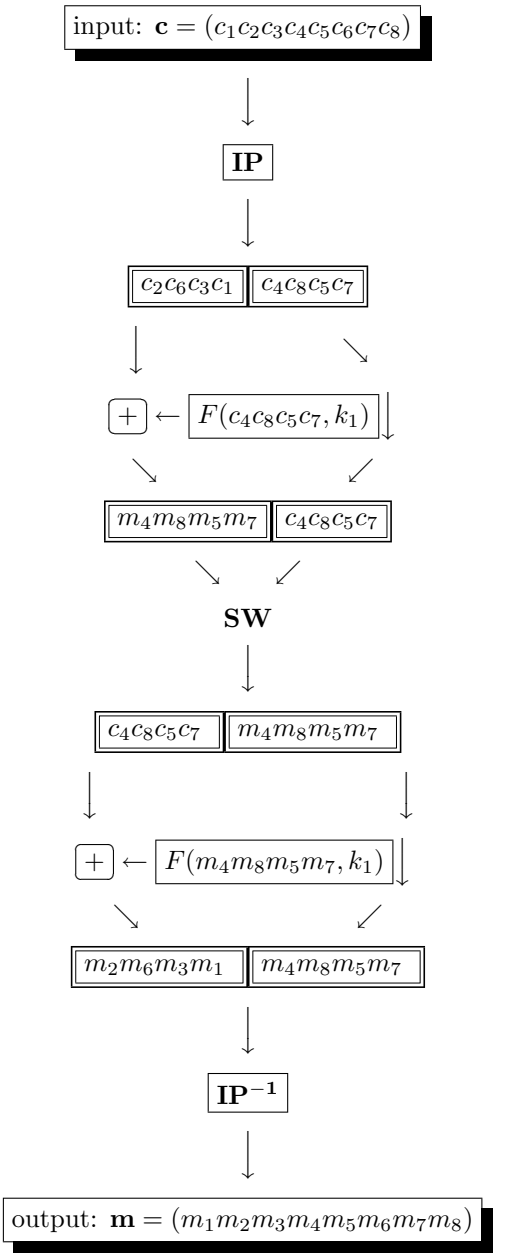
Diagrams 3.2 and 3.3 give a succinct presentation of S-DES.

Diagram 3.2 The S-DES Encryption Flowchart



The action between **IP** and **SW** is round 1, namely, the execution of f_{k_1}, and the action between **SW** and $\mathbf{IP}^{-1}$ is round 2, the action of f_{k_2}.

Diagram 3.3 The S-DES Decryption Flowchart



◆ Analysis of S-DES and Comparison with DES

Schaefer relabelled S-DES as *baby DES* since it is a much simpler block cipher than the full-blown DES. S-DES will encipher one block at a time, and there are 2^8 possible plaintext blocks since we are dealing with 8-bit plaintext bitstrings. In terms of composition of functions, all of the above discussion of S-DES can be encapsulated in the following.

$$(\mathbf{IP}^{-1} \circ f_{k_2} \circ \mathbf{SW} \circ f_{k_1} \circ \mathbf{IP})(m) = \mathbf{IP}^{-1}(f_{k_2}(\mathbf{SW}(f_{k_1}(\mathbf{IP}(m))))) = c.$$

Full DES takes 64-bit plaintext blocks, a 56-bit key, from which sixteen 48-bit subkeys are generated, and sixteen round functions, which we will label f_{k_j} for $j = 1, 2, \ldots, 16$. Hence, we may specify (full) DES now as a single composition of functions.

$$(\mathbf{IP}^{-1} \circ f_{k_{16}} \circ \mathbf{SW} \circ f_{k_{15}} \circ \mathbf{SW} \circ \cdots \circ f_{k_1} \circ \mathbf{IP})(m) = c.$$

Moreover, in DES, we have eight S-Boxes $\mathbf{S}_j$ for $j = 1, 2, \ldots, 8$, each having four rows and sixteen columns, where

$$\mathbf{S_j}(m_1 m_2 m_3 m_4 m_5 m_6)$$

picks out the entry in row $(m_1 m_6)$ and column $(m_2 m_3 m_4 m_5)$, which represents sixteen possible entries, in binary, for each such row. Also, $\mathbf{P_4}$ in S-DES, is replaced by $\mathbf{P_{32}}$ in DES, which is half the bitlength of the input in either case.

One of the weaknesses of DES that makes it unsuitable for use and ranks it as below standard for the modern day are its *weak keys*, which are keys k such that

$$E_k(E_k(m)) = m \text{ for all } m \in \mathcal{M}.$$

DES has four of these as follows, where an exponent means the repetition of that bitstring the number of times the exponent dictates.

$$k \in \{(0^{28}, 0^{28}), (1^{28}, 1^{28}), (0^{28}, 1^{28}), (1^{28}, 0^{28}) \in \mathbb{Z}^{28} \times \mathbb{Z}^{28}\}.$$

With these keys, encryption is the same function as decryption, so these keys must be avoided. There are also *semiweak keys*, which are key pairs (k_1, k_2) such that

$$E_{k_1}(E_{k_2}(m)) = m \text{ for all } \in \mathcal{M}.$$

There are six of them. They are listed as follows:

$$((01)^{14}(01)^{14}, (10)^{14}(10)^{14}),$$

$$((01)^{14}(10)^{14}, (10)^{14}(01)^{14}),$$

$$((01)^{14}(0)^{28}, (10)^{14}(0)^{28}),$$

$$((01)^{14}(1)^{28}, (10)^{14}(1)^{28}),$$

$$((0)^{28}(01)^{14}, (0)^{28}(10)^{14}),$$

and

$$((1)^{28}(01)^{14}, (1)^{28}(10)^{14}).$$

Each of these 56-bit key pairs will encipher plaintext to identical ciphertext. In other words, one key in the pair can decipher messages enciphered with the other key in the pair. Hence, these key pairs generate only two different subkeys, each of which is used eight times in the DES algorithm. They have to be avoided.

Another weakness of DES is the *complementation property*, described as follows. Let $c(k)$ denote the bitwise complementation of an input key k in DES. In other words, replace all 0's with 1's and all 1's with 0's. DES satisfies the following, which the reader may verify by trying this complementation on Diagram 3.1 on page 134.

DES Complementation Property

$$E_{c(k)}(c(m)) = c(E_k(m)).$$

In plain words, if one enciphers the complement of the plaintext with the complement of the key (the left side of the equation), then one gets the complement of the original ciphertext (the right side of the equation).

This says that complementation of the plaintext yields complementation in the ciphertext, and this means that a chosen-plaintext attack against DES has to test only half of the keyspace of 2^{56} keys, namely, 2^{55} of them.

Differential Cryptanalysis

In Section 2.6 on page 127, we learned about chosen-plaintext attacks. One of the best-known chosen-plaintext attacks against iterated block ciphers is *differential cryptanalysis* (DC). The original idea was developed by Murphy [67] in 1990 as an attack on another block cipher. It was improved and perfected by Biham and Shamir [5]–[6] in 1993, who used it to attack DES. DC involves the comparisons of pairs of plaintext with pairs of ciphertext, the task being to concentrate on ciphertext pairs whose plaintext pairs have certain "differences." Some of these differences have a high probability of reappearing in the ciphertext pairs. Those that do are called "characteristics," which DC uses to assign probabilities to the possible keys, with an end goal being the location of the most probable key.

As noted on page 133, DES has been replaced. It reached the end of its usefullness at the end of the twentieth century since it no longer had the ability to deliver as a secure cryptosystem. Since S-DES is a weaker version, it is intended only for pedagogical purposes to display the principles behind the construction of DES itself. In Section 3.2, we will look at its successor, AES. For now we need to look more deeply into the design principles underlying

DES since they are important from several perspectives for an understanding of symmetric-key block ciphers.

◆ Feistel ciphers

A *Feistel cipher* is a block cipher that inputs a plaintext pair (L_0, R_0), where both halves L_0 and R_0 have bitlength $b \in \mathbb{N}$ and outputs a ciphertext pair (R_r, L_r), where R_r and L_r have bitlength $b \in \mathbb{N}$ for each $r \in \mathbb{N}$, according to an iterative process, making it what is called an *iterated block cipher*. A key k is input and subkeys k_j for $j = 1, 2, \ldots, r$ are generated from it via a specified key schedule. Generally, $k_j \neq k_i$ for $j \neq i$, and $k \neq k_j$ for any j.

A function F, called a *round function* (iterated over r rounds, all of which have the same construction, described below), acts on plaintext pairs:

$$(R_{j-1}, k_j) \text{ for } j = 1, 2, \ldots, r$$

in a prescribed fashion, in concert with a switching function. The ciphertext output is

$$(L_j, R_j), \text{ where } L_j = R_{j-1}$$

and

$$R_j = L_{j-1} \oplus f(R_{j-1}, k_j). \tag{3.2}$$

In other words, if

$$(L_0, R_0) = (R_{-1}, R_0)$$

is the initial input plaintext, then for rounds $j = 1, 2, \ldots, r + 1$,

$$(L_{j-1}, R_{j-1}) = (R_{j-2}, R_{j-1})$$

is input and

$$(L_j, R_j) = (R_{j-1}, L_{j-1} \oplus f(R_{j-1}, k_j)) \tag{3.3}$$

is output.

The methodology prescribed for each round is that a substitution is executed on the left-hand data, from the previous round, via the action in the right-hand side of (3.2), to yield

$$(L_{j-1} \oplus f(R_{j-1}, k_j), R_{j-1}). \tag{3.4}$$

This is followed by a permutation yielding (3.3), which essentially results from a swap of the two halves of the data in (3.4). This process turns out to be a configuration of a methodology called the *substitution-permutation network* (SPN) put forth by Shannon, [86], about whom we will say more below.

The above Feistel encryption is essentially the same algorithm as the deciphering scheme. To decipher, one inputs the ciphertext with the use of the subkeys in the reverse order. Hence, we have a nice feature for implementation in that essentially the same algorithm is used for both encryption and decryption.

We now look at some design features of Feistel ciphers. We outline only the barest of statements about each principle, which we will expand in the section immediately following this list.

◆ Feistel Design Principles

1. **Block Size**: A large blocklength is chosen for increased security, with a 64-bit blocklength having been common, but blocklengths of 128 bits or more, are becoming standard due to modern demands stemming from increased cryptanalytic developments.

2. **Keylength**: When first developed, a 64-bit keylength was used, but, as we have seen, this has not survived the cryptanalytic onslaught. Now typically 128-bit keylengths are becoming standard.

3. **Rounds and Round Functions**: More rounds mean more security, with typically sixteen rounds being most common. A round function with increased complexity adds to the security.

4. **Subkeys**: Generation of subkeys from an input key during the operation of the algorithm aids in thwarting cryptanalysis.

S-DES and DES are examples of Feistel ciphers (with the only deviation from the above being that DES and S-DES begin and end with permutations). S-DES has a round function given above with $r = 2$ and subkey generation described in the above key schedule. DES is a Feistel cipher with $r = 16$.

Now we are in a position to explain the intimate details of just how the substitutions and permutations are used in Feistel ciphers in general, and DES in particular.

◆ Confusion and Diffusion

DES is basically a block cipher combining fundamental cryptographic techniques, *confusion* and *diffusion*. *Confusion* obscures the relationship between the plaintext and the ciphertext, which thwarts a cryptanalyst's attempts to study the ciphertext by looking for redundancies and statistical patterns. The best way to cause confusion is through the use of a complex substitution algorithm. (Note that a simple linear substitution such as some we have studied earlier would add negligible confusion. It is *necessary* to have a deeply complex substitution algorithm in order to cause confusion.)

Diffusion dissipates the redundancy of the plaintext by spreading it over the ciphertext, which frustrates a cryptanalyst's attempts to search for redundancies in the plaintext through observations of the ciphertext. The simplest manner in which we can cause diffusion in a binary block cipher is through repeatedly performing a permutation on the data followed by the application of a function to that permutation. This results in bits from different positions in the plaintext contributing to the same position in the ciphertext. Since DES involves an initial permutation followed by sixteen rounds of substitution, then a final permutation, DES essentially employs a sequence of confusion and diffusion techniques.

In 1949, Shannon published [86] in which the terms "confusion" and "diffusion" were introduced. His idea was to thwart frequency analysis by cryptanalysts.

The plaintext block size in DES of a 64-bit key input (reduced to 56 bits in the algorithm, since eight of the bits are parity check bits that are discarded) proved to be insecure for modern purposes. The new AES, which we will study in Section 3.2, has a 128-bit keylength, which is common in much of modern-day cryptosystems. (Many of us will see at the bottom of our browsers, when logging into a secure website such as online banking, something akin to

"*connection secure — RC4: 128-bit encrypted.*"

This is referring to Rivest's secure 128-bit *RC4* cipher, a "stream" cipher — see Section 2.3.) The greater the number of rounds in a Feistel cipher, the greater the security. Today, sixteen rounds is typical. Of course, the greater the complexity of the round function, the greater the difficulty for a cryptanalyst à la Shannon [86]. In fact, Shannon laid down commandments in the 1940's for secure symmetric-key cryptosystems. He first echoed Kerckhoffs' Principle (see page 96). Furthermore, he stipulated that any secure cipher must include *both* confusion and diffusion techniques, as does DES, for instance.

Let us now review classical ciphers in this light. Monolaphabetic substitution ciphers fail Shannon's criterion on both counts since no confusion or diffusion exists, given that all plaintext symbols are sent to the same ciphertext symbols and there is no transposition. With polyalphabetic substitution ciphers such as Vigenère, there is the use of confusion, since plaintext letters do not go to the same ciphertext letters, but they fail at diffusion since there is no transposition. Transposition ciphers use diffusion by definition but confusion is not necessarily employed, certainly not often effectively if it is. Now, we return to to DES.

◆ Double DES

One may strengthen DES by multiple encryptions (which means the application of the encryption algorithm several times in the same fashion as we would compose functions numerous times). For instance, there is *double DES* wherein we have two keys k_1 and k_2 so that encryption is given by

$$E_{k_2} \circ E_{k_1}(m) = E_{k_2}(E_{k_1}(m)) = c \text{ for any } m \in \mathcal{M},$$

and decrypt via

$$m = D_{k_1}(D_{k_2}(c)) = D_{k_1} \circ D_{k_2}(c).$$

On the surface, it would seem that the ostensible keylength in the double DES scheme involves $2 \times 56 = 112$ bits, which would be a significant increase in security over DES. However, reality has a way of interfering with expectations. Double DES has only a 56-bit keylength security level (which makes it only negligibly better in use than the original DES, which has 55-bit keylength security due to the complementation property described on page 145). This weakness of double DES was proved by Merkle and Hellman [60] in 1981. They show

that the security is reduced from 112 bits to 56 bits by making use of the *meet-in-the-middle attack*, which we now describe in the interest of completeness. Moreover, this form of attack is closely related to another attack (called the "birthday attack" — see page 128).

The meet-in-the-middle attack was introduced in 1977 by Diffie and Hellman [25]. It is based upon the following simple observation. Since

$$E_{k_2}(E_{k_1}(m)) = c, \text{ then } D_{k_2}(c) = E_{k_1}(m),$$

given that $D_{k_2} \circ E_{k_2}$ is the identity function, by definition. The way the attack works is that we are given a known plaintext/ciphertext pair (m_1, c_1), and we set up a table, which we will call T_1, of (sorted) values consisting of all 2^{56} possible values of $E_{k_1}(m)$. Now we start calculating another table consisting of all possible values of $D_{k_2}(c)$, one at a time, checking each one against the values in table T_1. If there is a match, say (K_1, K_2), then we take another known plaintext/ciphertext pair (m_2, c_2) and check for the equality:

$$E_{K_1}(m_2) = D_{K_2}(c_2).$$

If so, we accept this key pair as the legitimate keys.

To see why this works, consider the following. Suppose that we have an N-element set of values and we want to find a match of two of them. (Compare the following with the birthday attack described on pages 128 –130.) Say we split the values into two sets of n_1 and n_2 values. There are $n_1 n_2$ pairs of elements and each pair has a chance of $1/N$ in matching up. Hence, the match will likely occur when $(n_1 n_2)/N$ is close to 1. Thus, if we choose

$$n_1 \approx n_2 \approx \sqrt{N},$$

we achieve maximum efficiency in this search. Now, go back to the specific situation with double DES. Since $N = 2^{112}$ and $\sqrt{N} = 2^{56}$, we see why the effective keylength security of double DES is 2^{56}. This level of multiple encryption is therefore insufficient. We need more.

At the end of the twentieth century when DES had reached the end of its reign, and before the AES came into effect, NIST proposed an interim standard as follows; see [28].

◆ Triple DES

Let E_e and D_d denote the DES enciphering and deciphering transformations, respectively, and let k denote a DES key. We employ three keys k_j for $j = 1, 2, 3$. Then enciphering of plaintext is achieved via

$$E_{k_3}(D_{k_2}(E_{k_1}(m))) = c,$$

and deciphering occurs via

$$D_{k_1}(E_{k_2}(D_{k_3}(c))) = m.$$

Multiple encryptions strengthen the cipher so long as we do not have $k_1 = k_2$ or $k_2 = k_3$, because then

$$\text{either } D_{k_2} \circ E_{k_1} \text{ or } E_{k_2} \circ D_{k_3} \text{ is the identity function}$$

so we are back at square one with single DES. It is allowed that $k_1 = k_3$, or that all are distinct.

◆ DES Security-Related Issues

It turns out that multiple encryption of DES would be rendered useless if it were the case that for any given keys k_1 and k_2 there existed a key k_3 such that

$$E_{k_3}(m) = E_{k_2}(E_{k_1}(m))$$

for all plaintext inputs m. (This property, if it held, would be tantamount to DES permutations being closed under composition, and this would happen if DES satisfied the property that the set of permutations is closed as a group under composition.) Then multiple encryptions would be reduced to single encryptions and again we would be back to square one. However, in 1992, Campbell and Weiner saved the day by proving, in [13], that DES is not a group. In fact, they showed that a lower bound on the size of the group generated by composing the set of permutations is 10^{2499}. Thus, since we are safe on these issues, then with the proper choice of three keys triple DES has the effective keylength of 168 bits, making it a reasonable alternative, and triple DES is resistant to the meet-in-the-middle attack. That said, triple DES still inherits the disadvantages of DES, such as weak keys, semiweak keys, and the complementation property mentioned earlier. (It should be pointed out, in anticipation of the later study, that part of the ANSI X59.52 Triple DES Modes of Operation Standard, involving the CBC mode described on the next page, was cryptanalyzed in 2002 (see [4]). As a result, ANSI removed this mode from the proposed standard.)

There are other strengthenings of DES possible. Rivest developed a provably strong improvement to DES, called *DESX*. It simply does the following. Choose three keys k_1, k_2, k_3, and encipher by executing

$$k_1 \oplus E_{k_2}(k_3 \oplus m).$$

In other words, we add a 64-bit key k_3 modulo 2 to the input plaintext m before encryption, then we encipher the result with key k_2, and last add the 64-bit key k_1, modulo 2, to the ciphertext. In 1996, both Kilian and Rogaway [44] and Rogaway [80] demonstrated the improved security of DESX over DES. The security of DESX against the DC attack (see the displayed description on page 145) is roughly equivalent to that of DES.

An attack developed more recently than DC is one by Matsui [57] in 1994, called *linear cryptanalysis* (LC). This is one of the most prominent known-plaintext attacks against block ciphers (see page 127). (Also, see [41] for a nice tutorial treatment of both LC and DC.) LC uses linear approximations to describe the behaviour of the block cipher under attack. Matsui successfully

used LC against DES to obtain a key with 2^{43} known plaintexts (see [56]). In general, block ciphers with larger S-Boxes are less susceptible to DC and LC attacks.

Exercises

Using the key given for the permutation cipher in Example 3.1 on page 132, decrypt the ciphertexts in each of Exercises 3.1–3.4.

3.1. **ANYVIT**

3.2. **URRMDE**

3.3. **RASGNT**

3.4. **RISDNK**

3.5. Apply the initial permutation **IP** described on page 135 to the input

$$m = (10101011).$$

3.6. Apply the expansion permutation described on page 136 to the input

$$x = (1010).$$

3.7. Apply the S-Boxes $\mathbf{S_0}$ and $\mathbf{S_1}$ to the input (1110).

3.8. Given:

$$SK = (01010110) \text{ and } t = (11111011),$$

compute $f_{SK}(t)$, the S-DES round function described on page 138.
Hint: The end result is

$$f_{SK}(t) = (11011011).$$

3.9. Prove the DES complementation property highlighted on page 145.
Hint: Complementation does not affect modulo 2 *addition, namely,*

$$c(x) \oplus c(y) = x \oplus y.$$

3.2 AES

◆ Successor for DES

On January 2, 1997, NIST announced the initiation of an effort to develop the *Advanced Encryption Standard* (AES) as an unclassified, publicly disclosed encryption algorithm for protecting sensitive data. On August 9, 1999, NIST announced five finalists for the AES (in round two of their competition): *MARS*, *RC6*, *Rijndael*, *Serpent*, and *Twofish*. On October 2, 2000, NIST announced that *Rijndael* was selected as the proposed AES. (See the Rijndael fan club home page: *http://www.rijndael.com/*, where it is stated: "This page is dedicated to the fans of the Rijndael block cipher, whose selection, in an upset of Karelinean proportions, as the National Institute of Standards and Technology's proposed Advanced Encryption Standard has brought down the Feistel cipher dynasty.")

◆ The Advanced Encryption Standard (AES) — Rijndael

The name "Rijndael," pronounced as any of "Rhine Dahl," "Rain Doll," or "Reign Dahl," was derived from the names of Rijndael's Belgian designers, Vincent **Rij**men and Joan **Dae**men. The Rijndael cipher is based upon the 128-bit block cipher, called *Square*, which Rijmen and Daemen originally designed with a concentration on resistance against linear cryptanalysis. Later, Lars Knudsen engaged in more cryptanalysis of the square cipher. A paper by these three authors, describing the details of square, was presented at the workshop for *Fast Software Encryption* in the spring of 1997 in Haifa, Israel. In that spring of 1997, Daemen and Rijmen began working on a variant of the square cipher that would allow for key and block lengths of 128, 192, and 256 bits. They called their new cipher design "Rijndael" and submitted it to NIST by the June 1998 deadline. The rest is history. (Rijndael has been called *son of square* and alternatively square has been called *mother of Rijndael* by their creators. See *http://www.esat.kuleuven.ac.be/~rijmen/square/index.html* for a description of and implementation details for *Square*.)

In order to give even a brief description of Rijndael, we need to describe the essential components of it.

◆ The State

The *State*, is the intermediate cipher resulting from application of the round function. The State can be depicted as a $4 \times \mathbf{Nb}$ matrix, with bytes as entries, where **Nb** is the block length divided by 32. For instance, if the input block has 256 bits, then $\mathbf{Nb} = 8 = 256/32$, and the State would appear as a matrix $(a_{i,j}) \in \mathcal{M}_{4\times 8}((\mathbb{Z}/2\mathbb{Z})^8)$ of bytes. In this case, the State has 32 bytes. For an input block of 192 bits, the State would have 24 bytes as a $4\times\mathbf{Nb} = 4\times 6$ matrix, and for a block of length 128, it would have 16 bytes as a $4\times\mathbf{Nb} = 4\times 4$ matrix. Thus, we have variable State size.

Note that the input block (or *plaintext* if the mode of operation is ECB — see page 123) is put into the State (matrix) by column: $a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, a_{1,1} \ldots$ and at the end of the execution of the cipher the bytes are taken from the State in the same order.

◆ The Cipher Key

As with the State, the *cipher key* is portrayed as a $4 \times \mathbf{Nk}$ matrix of bytes, where **Nk** is the key length divided by 32. For instance, if the key length is 128 bits, then the cipher key is $(k_{i,j}) \in \mathcal{M}_{4\times 4}((\mathbb{Z}/2\mathbb{Z})^8)$. Hence, we have variable key size 16, 24, or 32 bytes, depending on key length 128, 192, or 256 bits.

◆ Key Schedule and Round Keys

The *round keys* can be derived from the cipher key by means of the following *key schedule*. There are two parts.

(1) The total number of round key bits equals $B(\mathbf{Nr}+1)$, where B is the block length and **Nr** is the number of rounds defined for each case in Table 3.1 below. For instance, if the block length is 128 bits and $\mathbf{Nr} = \mathbf{12}$, then 1664 round key bits are required.

(2) The cipher key is expanded into the *expanded key* in the following fashion. The expanded key is a linear array of 4-byte *words* (i.e. columns of the key matrix), where the first **Nk** words contain the cipher key. All other words are defined recursively in terms of previously defined words. (For a detailed account of how this is done, see the original description of Rijndael at *http://www.esat.kuleuven.ac.be/~rijmen/rijndael/*.)

Then round keys are extracted from the expanded key as follows. The first round key consists of the first **Nb** words, the second round key consists of the following **Nb** words, and so on.

◆ Round Function

First, we note that the *number of rounds*, denoted by **Nr**, is defined via the following table.

Table 3.1

Nr	**Nb** = 4	**Nb** = 6	**Nb** = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

In this table, we are including the final round, which we will describe below, which slightly differs from the other rounds in that step (3) below is eliminated.

The round function consists of four steps, each with its own name and its own particular function.

(1) Bytesub (BSB): In this step, bytes are mapped by an invertible S-Box, and there is only one single S-Box for the complete cipher. Thus, for instance, the State (position) matrix

$$(a_{i,j}) = (8i + j - 9) \text{ (for } 1 \le i \le 32,\ 1 \le j \le 8)$$

would be mapped, elementwise, by the S-Box to State matrix $(b_{i,j})$ via

$$a_{i,j} \longrightarrow \boxed{\text{S-Box}} \longrightarrow b_{i,j}.$$

This guarantees a high degree of nonlinearity by operating on each of the State bytes $a_{i,j}$ independently. To view the S-Box explicitly, together with a description of how it was constructed, see Appendix C on pages 335–336.

(2) Shift Row (SR): In this step, depending upon the value of **Nb**, row j for $j = 2, 3, 4$ of the State matrix are shifted x_j units to the right, where x_j is defined by Table 3.2 below.

Table 3.2

Nb	x_2	x_3	x_4
4	1	2	3
6	1	2	3
8	1	3	4

For instance, if **Nb = 4**, then

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \xrightarrow[\boxed{\textbf{SR}}]{} \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,3} & a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,0} \end{pmatrix}.$$

The **SR** step introduces high diffusion over multiple rounds and interacts with the next step.

(3) Mix Column (MC): In this step, the columns in the State matrix are treated as polynomials $a(x)$ over

$$\mathbb{F}_{2^8} \cong \mathbb{F}_2[x]/(m(x)),$$

where

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

is the irreducible Rijndael polynomial (see Appendix C on pages 335–336 and pages 311–316 in Appendix A). Then $a(x)$ is multiplied modulo $M(x) = x^4 + 1$ with a fixed invertible polynomial $c(x) = 3x^3 + x^2 + x + 2$, denoted by $c(x) \otimes a(x)$. Here multiplying modulo $x^4 + 1$ means that $x^i \pmod{x^4 + 1} = x^{i \pmod 4}$. It can be shown that if

$$a_j(x) = a_{3,j}x^3 + a_{2,j}x^2 + a_{1,j}x + a_{0,j}$$

represents column j in the State matrix, then $c(x) \otimes a(x)$ can be represented by the matrix product:

$$CA_j = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = B,$$

where the matrix A_j is column j of the State matrix and C is the *circulant* matrix representing $c(x)$. Hence, each column A_j of the State matrix is multiplied in this fashion by C. For instance, if $a(x) = x^3 + 1$, then

$$c(x) \otimes a(x) = 5x^3 + 4x^2 + 2x + 3,$$

which is given by the matrix product:

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 4 \\ 5 \end{pmatrix} = B.$$

This step linearly combines bytes in the columns and creates high intra-column diffusion. This technique is based upon the theory of error correcting codes, such as in cyclic redundancy checks (for example see [64, p. 549]).

(4) Round Key Addition (RKA): In this step, a round key is added modulo 2 to the State. For example,

$$(a_{i,j}) \oplus (k_{i,j}) = (b_{i,j}),$$

where $\oplus$ is addition modulo 2, $(a_{i,j})$ is the State matrix, $(k_{i,j})$ is the round key matrix, and $(b_{i,j})$ is the resulting State matrix. Thus, this step makes the round function key dependent.

There is significant parallelism in the Round Function. All four steps of a given round operate in a parallel manner on bytes, rows, or columns of the State.

◆ Stepwise Description of the Rijndael Cipher

Step 1 (**Initial Addition Round**) There is an initial RKA step.

Step 2 (**Rounds**) There are $\mathbf{Nr} - 1$ rounds executed.

Step 3 (**Final Round**) A final round is executed, where the **MC** step is omitted.

Hence, the detailed sequence of steps for Rijndael is an initial round key addition, then $\mathbf{Nr} - 1$ rounds of **BSB**, **SR**, **MC**, **RKA** each, followed by a final

round consisting of **BSB**, **SR**, **RKA**. Unlike DES, Rijndael does not require a "swapping step" in its rounds since the **MC** step causes every byte in a column to alter every other byte in the column.

Deciphering Rijndael is executed by reversing the steps using inverses and a modified key schedule.

◆ Security of Rijndael

The design of Rijndael practically eliminates the possibility of weak or semi-weak keys, which exist for DES (see see page 144). Moreover, the design of the key schedule virtually eliminates the possibility of equivalent keys. Although the mechanisms of differential and linear cryptanalysis can be adjusted to present attacks on Rijndael, it appears that Rijndael's design is sufficient to withstand these cryptanalytic onslaughts, since its S-Box is nearly perfect for resistance to differential cryptanalysis and the $\mathbb{F}_{2^8}$ equivalent of linear cryptanalysis.

A chosen plaintext attack, called the *Square attack*, which is a dedicated attack on the Square Cipher, can be used as well, since Rijndael inherited many features from Square. However, for seven or more rounds in Rijndael, no such attack, faster than exhaustive key search, has been found. Other attacks such as Biham's *related-key attack* or the *interpolation attacks* introduced by Jakobsen and Knudsen have little chance of success against Rijndael due to the diffusion and non-linearity of Rijndael's Key Schedule and the complicated construction of the S-Box.

◆ Concluding Comments

Unlike the Feistel structure of the round function, such as in DES, where some of the bits of the intermediate State or simply put into a different position unchanged, the Rijndael Round function is composed of three different invertible transformations, called *layers*, through which every bit of the State is treated in a similar fashion, called *uniformity*. The BSB step in each round is a *nonlinear mixing layer* (confusion). SR is a *linear mixing layer* (inter-column diffusion), and **MC** is also a linear mixing layer (inter-byte diffusion within columns). Then there is the *Key addition layer*. These layers ensure that Rijndael Round does *not* have a Fiestel structure. The layers are predominantly based upon the application of the *Wide Trail Strategy*, which is a devised system for providing resistance against Linear and Differential Cryptanalysis, discussed in Daemen's Doctoral Dissertation of March 1995.

Rijndael is well tailored to modern processors (Pentium, RISC, and parallel processors). It is also ideally suited for ATM, HDTV, Voice, and Satellite. Uses for Rijndael include MAC by employing it in a CBC-MAC algorithm. It is also possible to use it as a *Synchronous Stream Cipher*, a *Pseudorandom Number Generator*, or a *Self-Synchronizing Stream Cipher* (the latter, by using it in CFB mode), which we discussed in Sections 2.3 and 2.5.

Chapter 4

Public-Key Cryptography

4.1 The Ideas Behind PKC

The 1970s also saw a revolutionary change in the manner in which cryptographic keys were handled. Cryptography was about to go public. In order to begin to describe the ideas behind this approach, we need some terminology. In what follows, when we speak of an *entity* we will mean any person or thing, such as a computer terminal, which sends, receives, or manipulates information. Also, when we speak of a *channel* we will mean any means of communicating information from one entity to another. A *secure* channel is one that is not physically accessible to an adversary, whereas an *unsecured* channel is one from which entities, other than those for whom the information was intended, can delete, insert, read, or reorder data. Now let us have look at how this idea developed (some of which is taken from [64]).

In a paper [24] published in 1976, Whit Diffie and Martin Hellman (see Biographies 4.1 on the next page and 4.2 on page 159) conceived of a method for two entities who have never met in advance or exchanged keys to establish a shared secret key by exchanging messages over an open (unsecured) channel. Up to the time of this idea, all cryptosystems, including DES, were looking for mechanisms to securely distribute secret keys. As we know, this is necessary because, once a symmetric enciphering key is known, an entity can easily deduce the deciphering key from it. Now, with the introduction of the Diffie-Hellman idea, which has come to be known as the *Diffie-Hellman Key-Exchange*, entities could exchange keys in the open and ensure privacy. It seems contrary to the very notion of secrecy. However, that is the brilliance of the scheme: use two essentially *different* keys, one for enciphering that can be made public, and one for deciphering that can be kept private, a *key pair*. Moreover, the deciphering key would be constructed such that it would be be computationally infeasible to determine it from the public enciphering key. No longer would the key be

symmetric (the deciphering key easily determined from the enciphering key and vice versa). Now there would be an *asymmetric* key pair, the advent of *public-key cryptography* (PKC). How could this possibly work?

Before giving an introduction to the Diffie-Hellman idea, let us look at an analogy, a standard one, for PKC, which will provide an easy-to-understand scenario to give the reader an understanding of how a public key can work. We invoke our cryptographic cast of characters introduced on page 123 for ease of elucidation. Suppose that Bob has a *public* wall safe with a *private* combination known only to him. Moreover, the safe is left open and made available to passers-by. Then, anyone, including Alice, can put messages in the safe and lock it. However, only Bob can retrieve the message, because, even Alice, who left the message in the safe, has no way of retrieving it.

In order to give a general overview of the basic Diffie-Hellman idea, we need the notion of a *one-way function*, which we may view, at this juncture, as a method of enciphering that cannot be reversed. For instance, if you write a message on a piece of paper, then burn it, that is an example of a one-way function since retrieving the message is impossible. One says, in mathematical terms, that this is a function whose values are easy (computationally feasible) to compute, but calculating that inverse is *computationally infeasible* (see Footnote 2.1 on page 86). However, if you burn the paper, how does the intended recipient read the message? You need additional information built into your one-way function so that the intended recipient can recover the message. This additional information is called a *trapdoor*. Mathematically speaking, a *trapdoor* in a one-way function is additional information that makes the finding of the inverse a feasible task, but without the trapdoor information the task is computationally infeasible. For now, think of a trapdoor as information that allows you to invert the function (decrypt the message), but if you do not know it, you cannot invert the function. It is easy enough, as our paper-burning example indicated, to find one-way functions, but getting those with trapdoors requires a bit more effort. So now let us see how the Diffie-Hellman idea works.

Biography 4.1 Martin E. Hellman *was born on October 2, 1945. He obtained all his academic degrees in electrical engineering: his bachelor's degree from New York University in 1966; his master's degree in 1967; and his Ph.D. in 1969, the latter two from Stanford. He was employed at IBM and at MIT, but returned to Stanford in 1971. He remained there until 1996, when he received his Professor Emeritus status. We already learned above that he was one of the pioneers of PKC. He has been involved in computer privacy issues going back to the debate over the DES keylength in 1975 (see Levy's book [55] for some background to this fascinating story). He has not only demonstrated his scholarship with numerous publications, but also has excelled in teaching. He was recognized with four teaching awards; three of these were from minority-student organizations. He is now retired from research and teaching. He and Dorothie, his wife of some thirty-five years, live on campus at Stanford.*

Alice and Bob have never met, but want to establish a secret means of communicating with one another. Bob and Alice both have unique public keys, which we may envision as long strings of bits, published in some public database of keys that anyone can look up. Both Alice and Bob also have private keys that they keep secure and known only to themselves, namely, *only* Bob knows his private key and *only* Alice knows her private key. (We use the convention that the term *private* key is reserved for use in association with public-key cryptography, also called *asymmetric-key* cryptography, whereas the term *secret* key is reserved for symmetric-key cryptography. The cryptographic community has adopted this convention since it takes two or more entities to share a secret (such as the *symmetric* secret key), whereas it is truly *private* when only *one* entity knows about it, such as with the asymmetric private key.)

Biography 4.2 Bailey Whitfield Diffie *was born on June 5, 1944. His advanced education began when he entered MIT in 1961 from which he graduated in 1965, later accepting a job at Mitre Corporation. There he worked under the tutelage of Ronald Silver. Silver taught Diffie a great deal and inspired him to look further into cryptographic issues. In 1969, Diffie left Mitre and joined John McCarthy's Artificial Intelligence Lab at Stanford. By 1975, as described on page 158, the collaboration with Hellman, with input from Merkle, created the breakthrough. For his involvement, along with Hellman and Merkle, in the discovery of the notion of PKC, he was awarded a Doctorate in Technical Sciences,* Honoris Causa*, by the Swiss Federal Institute of Technology in 1992. His current position is Chief Security Officer at Sun Microsystems, in Palo Alto, California, where he has been since 1991. He has numerous awards from the Association of Computing Machinery (ACM), IEEE, NIST, NSA, and the Franklin Institute. The reader wanting more details of his involvement in public policy concerning cryptography, and his opposition to limitations on the use of cryptography by individuals and corporations, should consult Levy's book* [55]*, wherein Diffie is a central character.*

Alice takes a message and uses Bob's public key via a one-way function to encipher the message in a manner that *only* Bob's private key can decipher. So when Alice sends the cryptogram, the only person in the world who can decipher it is Bob, with his private key. Now suppose that another of our cast of characters, eavesdropping adversary *Eve*, intercepts the message. Without Bob's private key, she has only trial and error at her disposal to try to cryptanalyze it, probably taking millions of years, so her interception is useless. Thus, since Bob is the only person who has *both* elements of the key pair, he can decipher the message instantly. The message might contain the symmetric-key k, for example, and a reference to the symmetric-key algorithm, such as DES, for example. Similarly, Bob uses Alice's public key and a one-way function to encrypt a response, which would say that he agrees to use DES with symmetric-key k for their correspondence, and sends this to Alice, who uses her private key to decrypt, and she is the *only* one who can do so.

In the Diffie-Hellman scheme, k is the shared secret key independently gener-

ated by both Alice and Bob. The key exchange is complete since Alice and Bob are in agreement on k. Hence, over an unsecured channel, they have established a secure means of communicating.

The observant reader may wonder why they do not just use this key pair for all of their correspondence rather than using it to set up a key exchange for use with a symmetric-key cryptosystem. The reason has to do with efficiency, as we will see in detail later. Public-key methods are extremely slow compared with symmetric-key methods. In later discussions, we will see how both the public-key and symmetric-key cryptosystems come to be used, in concert, to provide the best of both worlds combining the efficiency of symmetric-key ciphers with the increased security of public-key ciphers, called *hybrid cryptosystems*.

In what follows, we will use the term *protocol* to mean, in general human terms, behaviour such as that *prearranged etiquette* understood at a formal dinner party. On the other hand, a *cryptographic protocol* means an algorithm, involving two or more entities, using cryptography to achieve a security goal, which might involve issues of authentication, privacy, and secrecy, all of which we will discuss in detail later in the text.

The Diffie-Hellman paper [24] was the "door-opener" to *public-key cryptography* in that it was the landmark, since it had the first cryptographic protocol with public-key properties including the idea of a trapdoor one-way function; a partial solution to the public-key cryptosystem; and digital signatures which we will study later.

In summary, the Diffie-Hellman key exchange allowed two entities to set up a shared secret symmetric key, but they did not provide any method for enciphering messages or any way to extend to digital signatures, which are digital data strings that associate a given message with its sender. As Diffie and Hellman put it at the outset of their paper, "We propose new techniques for developing public key cryptosystems, but the problem is still largely open." This would take a couple more years.

RSA and PKC

In 1978, a paper [79] was published by R. Rivest, A. Shamir, and L. Adleman (see Biographies 4.3 on the facing page, 4.4 on page 162, and 4.5 on page 164). In this paper they describe a public-key cryptosystem, including key generation and a public-key cipher, whose security rests upon the presumed difficulty of factoring integers into their prime factors. This cryptosystem, which has come to be known by the acronym from the authors' names, the *RSA cryptosystem*, has stood the test of time to this day, where it is used in cryptographic applications from banking and in e-mail security to e-commerce on the Internet. We will be discussing all these applications as we progress through the text, and we will provide the details of the RSA algorithm later. The astonishing aspect of the RSA cipher is that it rests upon mathematical developments from the eighteenth century, merely updated to our modern-day information-based computer world. In the RSA paper [79], Alice and Bob make their first appearance as sender and recipient of messages. These characters were quickly adopted by the

cryptographic community and were expanded to include a family of characters, such as Eve, and a host of others whom we will meet as our horizons broaden in our travels.

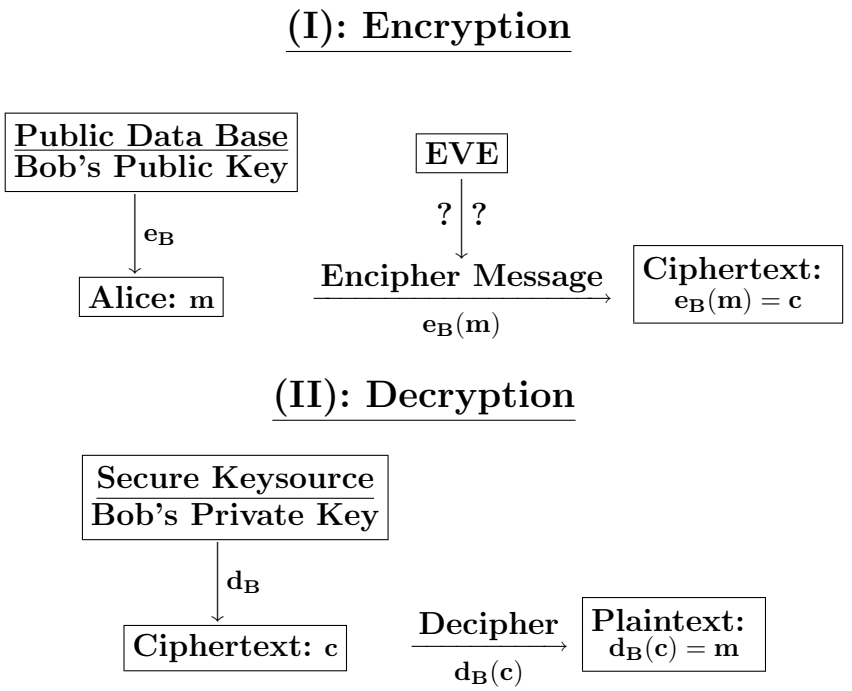
As Diagram 4.1 on the next page illustrates, if Alice wants to send a message to Bob, she looks up his public key e_B in a public database and encrypts her message m with it to get $e_B(m) = c$, as ciphertext. Should Eve be listening on the unsecure channel, over which Alice and Bob are communicating, the data of the original message would still be unknown. This is because Eve does not have access to Bob's securely protected provate key d_B, which is required to decipher the cryptogram. Of course, for this to work,

$$d_B(e_B(m)) = m$$

must hold for all messages m, and it must be impossible (or computationally infeasible) for anyone to decipher m from e_B without knowledge of d_B, to which only Bob has access. (Think of d_B as Bob's trapdoor information [his unique key] for unlocking the encrypting [one-way] function e_B, to recover m. Using the analogy described on page 158, $e_B(m)$ is his wall safe, which Alice locked with the message m inside, and to which only he has the combination [key].) Hence, unlike a symmetric key cryptosytem, an asymmetric key cryptosystem or PKC has two distinct keys for each person, a public one, such as Bob's e_B, which everyone can access, and a truly private one, such as Bob's d_B, which he and only he knows and keeps secure. Hence, we make the distinction between asymmetric-key encryption or PKC and secret-key encryption or SKC where both the enciphering and deciphering keys must be kept secret.

Biography 4.3 Ronald L. Rivest *received a B.A. in mathematics from Yale University in 1969 and a Ph.D. in computer science from Stanford University in 1974. He is a co-inventor of the RSA public-key cryptosystem and founder of RSA Data Security (now called RSA Security after having been bought by Security Dynamics). Among his numerous, outstanding honours and positions are Fellow of the American Academy of Arts and Science, Fellow of the Association for Computing Machinery, member of the National Academy of Engineering, Director of the the Financial Cryptographic Association, Director of the International Association for Cryptologic Research, Fellow of the World Technology Network, member of MIT's Laboratory for Computer Science; member of MIT's laboratory's Theory of Computing Group, a leader of the MIT Cryptography and Information Security Group, and currently the Andrew and Erna Viterbi Professor of Electrical Engineering and Computer Science at MIT. He, together with Adleman and Shamir, was awarded the 2000 IEEE Koji Kobayashi Computers and Communications Award, as well as the Secure Computing Lifetime Achievement Award. Moreover, he founded Peppercoin™, a company that provides a digital payment service for merchants, which ostensibly allows merchants to process small digital transactions for only pennies. He is widely respected as an expert in cryptographic design and cryptanalysis.*

Diagram 4.1 A Generic Public-Key Cryptosystem



Biography 4.4 Adi Shamir *is an Israeli cryptographer who is currently the Borman Professor in the Applied Mathematics Department of the Weizman Institute of Science in Israel. He obtained his Ph.D. from Stanford in 1977 after which he did postdoctoral work at Warwick University in England. Shamir's name is attached to a wide variety of cryptographic schemes, many of which we will study in this text, including the Fiat-Shamir identification protocol, RSA, DC* (*see the boxed description on page 145*)*, and his polynomial secret-sharing scheme, to mention only a few. On April 14, 2003, the ACM formally announced that the A.M. Turing Award* (*essentially the "Nobel Prize of computer science"*) *would go to Adleman, Shamir, and Rivest for their developmental work on PKC.*

◆ Public-Key Cryptosystems (PKC)

A cryptosystem consisting of a set of enciphering transformations $\{E_e\}$ and a set of deciphering transformations $\{D_d\}$ is called a *public-key cryptosystem* or an *asymmetric cryptosystem* if for each key pair (e, d) the enciphering key e, called the *public key*, is made publicly available, whereas the deciphering key d, called the *private key* (see page 159), is kept secret. The cryptosystem must satisfy the property that it is computationally infeasible to compute d from e.

In order to motivate the study of PKCs, we provide the following before we begin to describe the various types of PKCs and their uses.

◆ PKCs and SKCs — A Comparison

1. **Security**: With a PKC only the private key needs to be kept a secret, concealed by one entity, and public keys may be distributed freely. With an SKC there must be a shared secret key known by at least two entities. No PKC has been proven secure, yet except for the one-time-pad this is also true for SKCs.

2. **Longevity**: With PKCs, key pairs may be used without change in most cases over long periods of time, years in some situations. With SKCs, there may have to be a change of keys for each session.

3. **Key Management**: If a multiuser large network is being used (without a key server) then fewer private keys will be required with a PKC than with an SKC. For instance, if $n \in \mathbb{N}$ entities are communicating, using DES, then the number of keys required to allow any two entities to communicate is $n(n-1)/2$. Also, every user on the system has to store $n-1$ keys. This is called *key predistribution*. With a public-key cryptosystem, only n keys are required for any two entities to communicate since only one (public) key for each entity has to be stored. Hence, SKC, by itself, on the Internet is completely unworkable. Internet e-commerce cannot be supported by SKCs alone.

4. **Key Exchange**: In a PKC, no (private) key exchange between communicating entities is necessary. (Note that this tells us that the Diffie-Hellman key-exchange protocol, discussed on page 167, is not a public-key cryptosystem, although it contained the basic original ideas for it.) With an SKC, it is difficult and risky to exchange a secret key. In fact, one of the principal uses of PKC is for the exchange of a secret symmetric key.

5. **Digital Signatures and General Authentication**: Another of the principal roles played by PKC is that of providing digital signatures since they offer virtually the only means for securely doing so. On the other hand, the principal use of SKCs is bulk data enciphering.

6. **Efficiency**: PKCs are slower than SKCs. For instance, the RSA cryptosystem is roughly a thousand times slower than DES.

7. **Key Sizes**: The key sizes for a PKC are significantly larger than those required for an SKC. For instance, the private key in the RSA cryptosystem should be 1024 bits, whereas with an SKC generally 128 bits will suffice. Usually, private keys are ten times larger than secret keys.

8. **Nonrepudiation**: This means that the sender of a message cannot deny having sent it. With PKCs we can ensure nonrepudiation with digital signatures, whereas with SKCs we need Trent as a trusted third party.

We may summarize one salient point derived from the above: PKC is *not* meant to *replace* SKC but rather to *supplement* it for the goal of achieving maximum security and efficiency. This is done as follows. The general motivation behind modern cryptographic usage, especially on the Internet for e-commerce, is to employ PKC to obtain symmetric keys, which are then used in an SKC. Such cryptosystems are called *hybrid cryptosystems* or *digital envelopes*, which have the advantages of both types of cryptosystems. Here is how they work in practice.

Alice and Bob have access to an SKC, which we will call S. Also, Bob has a public-private key pair (e, d). In order to send a message m to Bob, Alice first generates a symmetric key, called a *session key* or *data encryption key*, k to be used only once. (The property of producing a new session key each time a pair of users wants to communicate is called *key freshness*.) Alice enciphers m using k and S obtaining ciphertext $E_k(m) = c$. Using Bob's public key e, Alice encrypts k to get $E_e(k) = k'$. Both of these encryptions are fast since S is efficient in the first enciphering, and the session key is small in the second enciphering. Then Alice sends c and k' to Bob, who deciphers k with his private key d, via $D_d(k') = k$. Then Bob easily deduces the symmetric deciphering key k^{-1}, which he uses to decipher

$$D_{k^{-1}}(c) = D_{k^{-1}}(E_k(m)) = m.$$

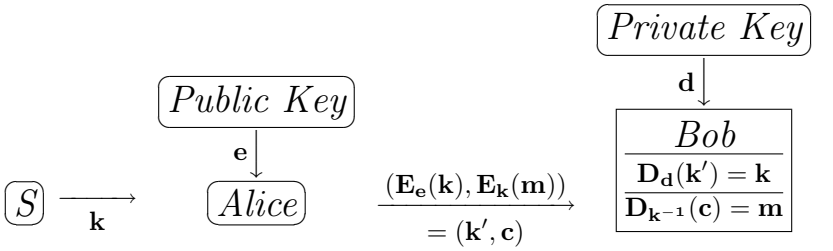
Hence, the PKC is used only for the sending of the session key, which provides a digital envelope that is both secure and efficient, a very nice and elegant resolution of the above problems. The next section begins with an illustration of a typical digital envelope and sets the stage for a discussion of various PKCs.

Biography 4.5 Leonard Adleman *was born on December 31, 1945, in San Francisco, California. He received his B.Sc. in mathematics from the University of California at Berkeley in 1972 and his Ph.D. there in 1976. His doctoral thesis was done under the guidance of Manuel Blum and was titled* Number Theoretic Aspects of Computational Complexity. *He is married with three children and is currently Henry Salvatori Professor of Computer Science and Professor of Molecular Biology at the University of Southern California Los Angeles, California, where he has been since 1980. His professional interests are algorithms, computational complexity, computer viruses, cryptography, DNA computing, immunology, molecular biology, number theory, and quantum computing. His most recent activity is the building of a DNA computer, that has the potential for a vastly faster computation for the future. He noticed that a protein, called* polymerase, *which produces complementary strands of DNA, resembles the operation of a Turing machine. Adleman reached the conclusion that DNA formation essentially functions in a fashion similar to a computer, so he is interested in constructing a viable DNA computer.*

4.2 Digital Envelopes and PKCs

Much of what follows is taken from [64] and adapted to the needs of this text.

Diagram 4.2 Digital Envelope — Hybrid Cryptosystem



We are ready to look at the various PKCs. The ideas behind the most famous PKC, namely RSA, about which we will learn the details in the next section, are based upon the simple mathematical idea of exponentiation and related matters. The first of the related matters is a notion that we need to set up the first exponentiation cipher. The security of many cryptosystems depends upon the difficulty of solving certain problems such as the following.

If we are dealing with real numbers, then finding e from α^e is called the logarithm function. In $\mathbb{F}_p^*$ (or more generally in any finite group) this is called the *discrete logarithm problem* (DLP). Thus, we present the problem formally as follows.

Discrete Log Problem (DLP):

Given a prime p, a generator m of $\mathbb{F}_p^*$, and an element $c \in \mathbb{F}_p^*$, find the unique integer e with $0 \leq e \leq p-2$ such that

$$c \equiv m^e \pmod{p}. \tag{4.1}$$

The DLP is often called simply *discrete log*. Here $e \equiv \log_m(c) \pmod{p-1}$. If p is "properly chosen," this is a very difficult problem to solve. One of the ways that p has to be properly chosen is to insist upon $p-1$ having at least one large prime factor. This is due to the Silver-Pohlig-Hellman Algorithm, which allows for efficient calculation of discrete logs when $p-1$ has only small prime factors. Due to the technical nature of the algorithm, we have placed a description of it in Appendix E.

It can be shown that the complexity of finding e in (4.1) when p has n digits is roughly the same as factoring an n-digit number (for instance, see [68]). Therefore, computing discrete logs is virtually of the same degree of difficulty

as factoring, and since there are no known tractable factoring algorithms, we assume that the *integer factoring problem* (IFP) is intrinsically difficult (see [64, Appendix C]). Hence, cryptosystems based upon the discrete log problem are assumed to be secure. Yet, there is no verification of this abstractly in the sense that no nontrivial lower bounds have been found for the complexity of integer factorization. (See the discussion of complexity in Section 1.8.)

A symmetric-key cipher whose security depends upon the discrete log problem is our next topic. It involves the name of a contributor whom we met earlier — see Biography 4.1 on page 158.

Now it is time for us to go back to symmetric-key cryptography (SKC) and learn about an exponentiation cipher that will help us set the stage for PKC in general and RSA in particular.

◆ The Pohlig-Hellman[4.1] Symmetric-Key Exponentiation Cipher

(a) A secret prime p is chosen and a secret enciphering key $e \in \mathbb{N}$ with $e \leq p-2$.

(b) A secret deciphering key d is computed via $ed \equiv 1 \pmod{p-1}$.

(c) Encryption of plaintext message units m is: $c \equiv m^e \pmod{p}$.

(d) Decryption is achieved via $m \equiv c^d \pmod{p}$.

Example 4.1 *Let* $p = 181$*, and set* $e = 97$*, with plaintext* $1, 4, 19, 17, 0, 24, 0, 11$*. Then we encipher each by exponentiating as follows, where all congruences are modulo* 181*.*

$$1^{97} \equiv 1; \quad 4^{97} \equiv 94; \quad 19^{97} \equiv 19; \quad 17^{97} \equiv 92;$$

$$0^{97} \equiv 0; \quad 24^{97} \equiv 158; \quad 0^{97} \equiv 0; \quad 11^{97} \equiv 168.$$

Then we send off the ciphertext. To decipher, we need the inverse of e *modulo* $180 = p-1$*, and this is achieved by using the Euclidean algorithm (see Theorem 1.2 on page 3) to solve*

$$97d + 180x = 1,$$

which has a solution $d = 13$ *for* $x = -7$*, and this is the least positive such value of* d*. So we may decipher via* $94^{13} \equiv 4$ *and so on to retrieve the plaintext.*

▼ Analysis

Since knowledge of e and p would allow a cryptanalyst to obtain d, then both p and e must be kept secret. The security of this cipher is based on the difficulty of solving the DLP; namely, an adversary without knowledge of e or d would have to compute $e \equiv \log_m(c) \pmod{p-1}$.

The Pohlig-Hellman cipher is an example of the use of *fixed-exponent exponentiation* where the base may vary but the exponent is fixed. The next algorithm, which we mentioned informally discussed on pages 158–160, is an

example of the use of *fixed-base exponentiation* where the exponent may vary but the base is fixed. This algorithm is a prime motivator for PKC, and its security depends upon the DLP.

◆ The Diffie-Hellman Key-Exchange Protocol

Suppose that Alice and Bob have not yet met nor exchanged keys, but they want to establish a shared secret key k by exchanging messages over an unsecured channel. First Alice and Bob agree on a large prime p and a generator α of $\mathbb{F}_p^*$ $(2 \leq \alpha \leq p-2)$. These need not be kept secret, so Alice and Bob can agree over an unsecured channel. Then the protocol proceeds as follows.

(1) Alice chooses a random (large) $x \in \mathbb{N}$ and computes the least positive residue X of α^x modulo p and then sends X to Bob (and keeps x secret).

(2) Bob chooses a random (large) $y \in \mathbb{N}$ and computes the least positive residue Y of α^y modulo p and then sends Y to Alice (and keeps y secret).

(3) Alice computes the least positive residue of Y^x modulo p, and Bob computes the least positive residue of X^y modulo p. Since

$$Y^x \equiv \alpha^{yx} \equiv \alpha^{xy} \equiv X^y \equiv k \,(\mathrm{mod}\ p),$$

they have a shared secret key k.

Example 4.2 *Suppose that the parameters are* $p = 2663$, $\alpha = 2$, $x = 1085$, *and* $y = 1701$. *Then* $X \equiv 2^{1085} \equiv 252 \,(\mathrm{mod}\ p)$, $Y \equiv 2^{1701} \equiv 1524 \,(\mathrm{mod}\ p)$, $Y^x \equiv 1524^{1085} \equiv 2103 \,(\mathrm{mod}\ p)$, *and* $X^y \equiv 252^{1701} \equiv 2103 \,(\mathrm{mod}\ p)$. *Hence,* $k = 2103$ *is the shared secret key.*

▼ Analysis

In the Diffie-Hellman protocol, k is the shared secret key independently generated by both Alice and Bob. The key exchange is complete, since Alice and Bob are in agreement on k. The Diffie-Hellman protocol differs from the Pohlig-Hellman cipher in that the latter requires that both p and e be kept secret since d could be deduced from them, whereas in the former p and α may be made public due to the intractability of the DLP. However, there is a subtler problem here that we need to discuss, not only in reference to the above but also for later use.

A cryptanalyst, Eve, listening to the channel would know p , α, X and Y but neither x nor y. Thus, Eve faces what is called the

Diffie-Hellman Problem (DHP):

find $\alpha^{xy} \,(\mathrm{mod}\ p)$ given $\alpha, \alpha^x \,(\mathrm{mod}\ p)$ and,

$\alpha^y \,(\mathrm{mod}\ p)$ (but not x or y).

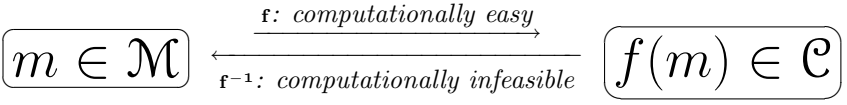
If Eve can solve the DLP, then she can clearly solve the DHP. Whether the converse is true or not is unknown. In other words, it is not known if it is possible for a cryptanalyst to solve the DHP without solving the DLP. Nevertheless, the consensus is that the two problems are equivalent. Thus, for practical purposes, one may assume that the Diffie-Hellman Key-Exchange protocol is secure as long as the DLP is intractable.

There are two more ingredients for the RSA recipe that we need. The first we encountered briefly on page 158, and we now formalize the notion.

One-Way Functions

A one-to-one function f from a set $\mathcal{M}$ to a set $\mathcal{C}$ is called *one-way* if $f(m)$ is "easy" to compute for all $m \in \mathcal{M}$, but for a randomly selected c in the image of f, finding an $m \in \mathcal{M}$ such that $c = f(m)$ is computationally infeasible. In other words, we can easily compute f, but it is computationally infeasible to compute f^{-1}.

Diagram 4.3 One-Way Function



One-way functions have a plethora of cryptographic uses. For instance, one use of one-way functions, about which we will see more later, is *password security*. Suppose that Alice has a password p and f is a one-way function. Then p can be stored as $f(p)$ on a computer. When Alice logs in to her account, the computer takes p, calculates $f(p)$, and checks that it matches the stored value. A cryptanalyst who gets hold of the password file will have only the $f(p)$ value for each user, and obtaining p is computationally infeasible.

The above being said, there is no rigorous mathematical proof that one-way functions actually exist. Yet, as we saw on page 158, we have working definitions, pragmatic ones, that serve us well. Moreover, we now have "candidate" one-way functions such as the DLP and the IFP, discussed earlier. The reader interested in a deeper analysis of this issue may consult books on complexity theory that go well beyond the basics covered in Section 1.8 (see [34] for instance).

The reader may wonder at this point how it is that we could devise a cryptosystem using one-way functions. The recipient of a message enciphered with a one-way function would ostensibly be no better off than a cryptanalyst at finding the plaintext since computing the inverse is computationally infeasible. This is correct, so the recipient needs more information, the idea for which is contained in what follows.

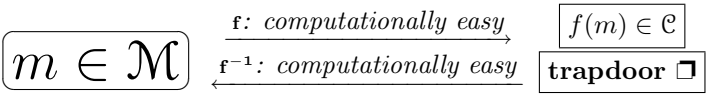
◆ Trapdoor One-Way Functions

A *trapdoor one-way function* or *public-key enciphering function* is a one-way function,

$$f : \mathcal{M} \mapsto \mathcal{C},$$

satisfying the additional property that there exists information, called *trapdoor information*, or simply *trapdoor*, that makes it feasible to find $m \in \mathcal{M}$ for a given $c \in \text{img}(f)$ such that $f(m) = c$, but without the trapdoor this task becomes infeasible.

Diagram 4.4 Trapdoor One-Way Function



The essential idea behind the Diffie-Hellman key exchange is the use of trapdoor one-way functions. The Diffie-Hellman protocol, discussed earlier in this section, allows for entities who have never met or exchanged information to establish a shared secret key by exchanging messages over an unsecured channel. Since exponentiation modulo p is polynomial time, then enciphering is easy. However, finding the inverse, solving the DLP, is computationally infeasible without the trapdoor, namely, one of the secret pair (x, y). This can be made to work in a more general context, using the IFP, that is most germane to our discussion of RSA in the next section and will provide a nice motivator.

Example 4.3 Let $f(x) \equiv x^e \pmod{n}$ where $n = pq$ with $p \neq q$ primes, and suppose that

$$de \equiv 1 \pmod{(p-1)(q-1)}.$$

Then applying f is computationally easy, but finding

$$f^{-1}(x^e) \equiv f^{-1}(f(x)) \equiv x^{ed} \equiv x \pmod{n} \tag{4.2}$$

is computationally infeasible without the trapdoor d.

▼ Analysis

First, we note that $(p-1)(q-1)$ in Example 4.3 is just the Euler function (see page 37) applied to pq, since

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

and the application of f^{-1} in Equation (4.2) is just Euler's generalization of Fermat's Little Theorem (see Theorem 1.18 on page 40). This little bit of elementary number theory is really all that is behind the RSA cipher, so understanding this is sufficient to understand the entire cryptosystem.

In order to show that the finding of the trapdoor d in Example 4.3 is based upon the IFP, in this case factoring n, we need to show that computing d is "as hard as" factoring n. Determining what *as hard as* means will involve some discussion of probabilities.

If we can factor n, obviously we can compute $(p-1)(q-1)$. Then we can use the Euclidean algorithm to find d from e (in computationally feasible time), since we need merely solve

$$ed + x(p-1)(q-1) = 1$$

for x and d. A simple example is $e = 7$, $p = 101$, and $q = 167$. Then solving

$$7d + 100 \cdot 166 \cdot x = 1$$

is easily achieved, with $x = -2$, and $d = 4743$. In fact, it can be shown that being able to compute d can be converted (with an arbitrarily high probability) into an algorithm for factoring n (see [63, p. 65] for instance). In other words, knowledge of d can be converted into an algorithm for factoring n (with an arbitrarily small probability of failure to do so). Thus, to say that finding d is as hard as factoring the modulus is not a proven fact, rather a conjecture based on some (rather solid) evidence. We will formalize this conjecture in the next section.

Note, as well, that if we have $\phi(n)$ and n, then we can factor n. The reason is that we can find p and q by successively computing

$$p + q = n - (p-1)(q-1) + 1 \text{ and } p - q = \sqrt{(p+q)^2 - 4n}, \tag{4.3}$$

so we get

$$p = \frac{1}{2}\left[(p+q) + (p-q)\right] \text{ and } q = \frac{1}{2}\left[(p+q) - (p-q)\right].$$

Hence, finding d or finding $\phi(n)$ means we can factor n.

Exercises

4.1. Prove that the DLP presented on page 165 is independent of the generator m of $\mathbb{F}_p^*$. (This means you must demonstrate that any algorithm that computes logs to base m can be used to compute logs to any other base m' that is a generator of $\mathbb{F}_p^*$.)

4.2. The *generalized discrete log problem* is described as follows. Given a finite cyclic group G of order $n \in \mathbb{N}$, a generator α of G, and an element $\beta \in G$, find that unique nonnegative integer $x \leq n-1$ such that $\alpha^x = \beta$. Given the fact that such a group G is isomorphic to $\mathbb{Z}/n\mathbb{Z}$ (see page 316), one would expect that an efficient algorithm for computing discrete logs in one group would imply an efficient algorithm for the other group. Explain why this is *not* the case.

Exercises 4.3–4.6 refer to the Pohlig-Hellman exponentiation cipher presented on page 166. In each exercise, use the data to decipher the given cryptogram and produce the plaintext via Table 2.2 on page 83.

4.3. $p = 167$, $e = 3$, and $c = (125, 162, 0, 154, 9, 11, 26, 12, 9, 64, 35, 0, 26)$.

4.4. $p = 263$, $e = 73$, and

$$c = (246, 18, 156, 0, 256, 127, 18, 156, 96, 256, 235, 0, 132, 68).$$

4.5. $p = 397$, $e = 59$, and

$$c = (160, 313, 2, 223, 11, 0, 1, 286, 369, 160, 2, 160).$$

4.6. $p = 1013$, $e = 5$, and

$$c = (685, 0, 323, 934, 997, 352, 535, 11, 352, 323, 323, 32, 0, 644, 32, 11).$$

Exercises 4.7–4.10 refer to the Diffie-Hellman key exchange protocol introduced on page 167. In each exercise, use the data to find the shared secret key k.

4.7. $p = 877$, $\alpha = 2$, $x = 25$, and $y = 3$.

4.8. $p = 907$, $\alpha = 2$, $x = 32$, and $y = 153$.

4.9. $p = 1193$, $\alpha = 3$, $x = 69$, and $y = 96$.

4.10. $p = 1471$, $\alpha = 6$, $x = 51$, and $y = 22$.

4.11. Explain why selecting $\alpha = p - 1$ would be a very negative choice.

4.12. Verify the statement made on page 168 to the effect that if Eve can solve the DLP, then she can solve the DHP.

4.13. Suppose that p is a prime such that $q = (p-1)/2$ is also prime, called a *safe prime*. Furthermore, assume that α and β are primtive roots modulo p with $\alpha^a \equiv \beta \pmod{p}$, where a is kept secret. Define a hash function

$$h : \mathbb{Z}/q^2\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}$$

as follows for a given message $m = r_0 + r_1 q$ where $0 \le r_0, r_1 \le q - 1$,

$$h(m) \equiv \alpha^{r_0} \beta^{r_1} \pmod{p}.$$

Prove that h is strongly collision reistant. In other words, prove that if $h(m) = h(m_1)$ for different messages m and m_1, then we can determine a.

4.3 RSA

Although the Diffie-Hellman Key-Exchange protocol, discussed on page 167, was the genesis of a profound investigation into the notion of PKC, their scheme did not provide a complete solution to the establishment of a complete PKC. They provided only a mechanism for the exchange of keys and, by the authors' own admission, left open the problem of establishing a working secure PKC (see page 158). The first to do this, as we know, have their names attached to the acronym that did provide such a solution.

◆ The RSA Public-Key Cryptosystem

We break the algorithm into two parts with the underlying assumption that Alice wants to send a message to Bob.

(I) RSA Key Generation

1. Bob generates two large, random primes $p \neq q$ of roughly the same size and computes both $n = pq$ and
$$\phi(n) = (p-1)(q-1).$$
The integer n is called his (*RSA*) *modulus*.

2. He selects a random $e \in \mathbb{N}$ such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$. The integer e is called his (*RSA*) *enciphering exponent*. Then using the extended Euclidean algorithm (see Theorem 1.7 on page 12), he computes the unique $d \in \mathbb{N}$ with $1 < d < \phi(n)$ such that
$$ed \equiv 1 \pmod{\phi(n)}.$$

3. Bob publishes (n, e) in some public database and keeps d, p, q, and $\phi(n)$ private. Thus, Bob's (*RSA*) *public-key* is (n, e) and his (*RSA*) *private key* is d. The integer d is called his (*RSA*) *deciphering exponent*.

(II) RSA Public-Key Cipher

enciphering stage:

In order to simplify this stage, we assume that the plaintext message $m \in \mathcal{M}$ is in numerical form with $m < n$. Also, $\mathcal{M} = \mathcal{C} = \mathbb{Z}/n\mathbb{Z}$, and we assume that $\gcd(m, n) = 1$.

1. Alice obtains Bob's public key (n, e) from the database.

2. She enciphers m by computing $c \equiv m^e \pmod{n}$ using the repeated squaring method given on page 31 and sends $c \in \mathcal{C}$ to Bob.

deciphering stage:

Once Bob receives c, he uses d to compute $m \equiv c^d \pmod{n}$.

Remark 4.1 *To see that Bob's decryption actually recovers m, we observe the following. Since*

$$ed \equiv 1 \pmod{\phi(n)},$$

there exists a $g \in \mathbb{Z}$ such that

$$ed = 1 + g\phi(n).$$

If $p \nmid m$, then by Fermat's Little Theorem, $m^{p-1} \equiv 1 \pmod{p}$. Hence,

$$m^{ed} = m^{1+g(p-1)(q-1)} \equiv m(m^{g(q-1)})^{p-1} \equiv m \pmod{p}. \tag{4.4}$$

If $p|m$, then (4.4) holds again since $m \equiv 0 \pmod{p}$. Hence, $m^{ed} \equiv m \pmod{p}$ for any m. Similarly, $m^{ed} \equiv m \pmod{q}$. Since $p \neq q$, $m^{ed} \equiv m \pmod{n}$. Thus,

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

Example 4.4 Suppose that Bob chooses $(p, q) = (3677, 4013)$. Then $n = 14755801$ and $\phi(n) = 14748112$. If Bob selects $e = 5$, then solving $1 = 5d + \phi(n)x$ (for $x = -2$), he gets $d = 5899245$, his private key. Also, $(14755801, 5)$ is his public key. Alice obtains Bob's public key and wishes to send the message $m = 279257$. She enciphers using Bob's public key to get

$$c \equiv m^5 \equiv 1028750 \pmod{n},$$

which she sends to Bob. He uses his private key d to decipher via

$$c^d \equiv 10287150^{5899245} \equiv 279257 \equiv m \pmod{n}.$$

▼ Block Size

We cannot properly encipher the plaintext message unit if it is a numerical value $m \geq n$. (The reader may try an example, say, $m = 72892588$, in Example 4.4, and see that information is lost [under modular reduction] and the system fails.) When $m \geq n$, we must subdivide the plaintext numerical equivalents into blocks of equal size, a process called *message blocking*. If we are dealing with numerical equivalents of the plaintext in base N integers for some fixed $N > 1$, then message blocking is accomplished by choosing that unique integer ℓ such that $N^\ell < n < N^{\ell+1}$ (see Exercise 4.22 on page 180). Then we write the message as blocks of ℓ-digit, base N integers (with zeros packed to the right in the last block if necessary), and encipher each separately. Since $N^\ell < n$, each block of plaintext corresponds to an element of $\mathbb{Z}/n\mathbb{Z}$. Therefore, since $n < N^{\ell+1}$, then each ciphertext message unit can be uniquely written as an $(\ell + 1)$-digit, base N integer in $\mathcal{C} = \mathbb{Z}/n\mathbb{Z} = \mathcal{M}$.

▼ Modulus Size

For the modern day and the near future, an RSA modulus of 1024 to 4096 bits would be considered secure, but this is dependent upon how long one is

willing to wait for the software to generate a secure keypair as well. Certain RSA moduli of n digits that are a product of two primes of approximately the same size are denoted by RSA-n, called an *RSA challenge number*. These are published on the Internet, and the reader may request the list from *challenge-rsa-list@rsa.com*. These are numbers for which rewards are offered to factor them. We will return to some concrete examples of these numbers shortly.

▼ Modulus Parameters

It is worthy of discussion to look at the possible choices for the various parameters in the RSA Algorithm (or simply RSA henceforth for convenience). For instance, in the choice of the primes p and q, one should not choose them too close together. Suppose that $p > q$ and p is "close" to q, in the sense that $(p+q)/2$ is only slightly bigger than $\sqrt{n} = \sqrt{pq}$. Given that

$$\left(\frac{p+q}{2}\right)^2 - n = \left(\frac{p-q}{2}\right)^2,$$

then we have a solution $(x, y) \in \mathbb{N} \times \mathbb{N}$ to $x^2 - n = y^2$. To factor n we need only test those integer values $x > \sqrt{n}$ until $x^2 - n = y^2$ for some $y \in \mathbb{N}$, since $n = (x-y)(x+y)$. Thus, choosing p and q too close together will make this task much easier. Take, for instance, our modulus in Example 4.4 on the preceding page, $n = 14755801$ where $\lfloor \sqrt{n} \rfloor = 3841$. Testing for $x = 3842, 3843, 3844$ does not yield a square, but for $x = 3845$ we get $3845^2 - 14755801 = 168^2$, so we factor n as $n = 14755801 = (3845 - 168)(3845 + 168) = 3677 \cdot 4013$. (Note that $(p+q)/2 = 3845 > 3841 = n$, so the values of p and q are too close to one another in the preceding sense.) Of course, our example is too small to be realistic, for reasons discussed above, but even with much larger primes the idea works, so caution must be exercised. Certain precautions should also be taken in the choice of p and q to ensure that $p-1$ and $q-1$ do not have any large common factor. If they have such a factor, then it becomes easier to obtain the deciphering exponent d since the inverse of the enciphering exponent e modulo $\operatorname{lcm}(p-1, q-1)$ suffices for d. Take an example of the extreme case where $p = 23$, $q = 67$, and $e = 5$. Here $22 = (p-1) \mid (q-1) = 66$, so we need only compute the inverse of e modulo $q-1$, which is $d = 53$. Another caution is that $\phi(n)$ should have a large prime factor, or more precisely not be a product of only small prime factors. The reasons for this have been discussed when we talked about the Diffie-Hellman algorithm where we provided a solution to this problem, namely choose $(p-1)/2$ (and $(q-1)/2$) to be prime, which solves the aforementioned lcm problem as well. As seen in Exercise 4.13 on page 171, such primes are called safe primes. It is suspected, but there is no proof, that there are infinitely many safe primes. Last, on these cautionary issues, although one may select an enciphering exponent e in the range $1 < e < \phi(n)$, some of these choices are very bad. For instance, if $e = \phi(n)/2 + 1$, then by Euler's Theorem 1.18,

$$m^{\phi(n)/2+1} = (m^{\phi(p)})^{\phi(q)/2} m \equiv m \pmod{p},$$

and

$$m^{\phi(n)/2+1} = (m^{\phi(q)})^{\phi(p)/2} m \equiv m \pmod{q},$$

so

$$m^{\phi(n)/2+1} \equiv m \pmod{n},$$

namely $m^e \equiv m \pmod{n}$ for all $m \in \mathcal{M}$, clearly not a desirable outcome.

▼ The Euler Function and Alternatives

If one needs to execute RSA key generation *often*, say for numerous communications between banks, then we must ensure that finding such an e over a large number of trials is fast. In other words, we need to know that calculating $1-\phi(n)/n$ over N trials can be made arbitrarily small for large N. For instance, this will happen if

$$(1 - \phi(n)/n)^N < (1/2)^N.$$

In particular, if $n = pq$ is an RSA modulus, then

$$\phi(n)/n = (1 - 1/p)(1 - 1/q) > (4/5)^2 = .64 > 1/2.$$

It can be shown by more sophisticated techniques that for arbitrary n,

$$\phi(n)/n > (1 + o(1))e^{-\gamma}/\ln\ln n,$$

where

$$\gamma = \lim_{n\to\infty} (1 + 1/2 + \cdots + 1/n - \ln n)$$

is *Euler's Constant*.

Instead of $\phi(n)$, one may use the Carmichael Function $\lambda(n)$, which is defined in the solution to Exercise 2.37 on page 369. If p and q are chosen such that $\gcd(p-1, q-1)$ is small, then $\phi(n)$ and $\lambda(n)$ are about the same size.

On pages 169 and 170, we discussed an instance that is tantamount to the encryption and decryption of the RSA cipher. Therein, we talked about a notion that we can now name.

The RSA Conjecture
Cryptanalyzing RSA must be as difficult as factoring.

Although there is no proof of this conjecture, the aforementioned discussion in the previous section tells us that the evidence is strong and the general consensus is that the conjecture is valid. A good reason for believing this is that the only known method for finding d given e is the extended Euclidean algorithm applied to e and $\phi(n)$. Yet, to compute $\phi(n)$, we need to know p and q, namely, we need to know how to factor n.

Given the above statement, it is worth a few more words on the extended Euclidean algorithm. This algorithm calculates the $\gcd(e, \phi(n))$, and when $\gcd(e, \phi(n)) = 1$, it calculates the $e^{-1} \pmod{\phi(n)}$. This is accomplished relatively quickly.

There are numerous cryptosystems that are called *equivalent to the difficulty of factoring*. For instance, there are RSA-like cryptosystems whose difficulty to

break is as hard as factoring the modulus. It can be shown that any cryptosystem for which there is a constructive proof of equivalence to the difficulty of factoring is vulnerable to a chosen-ciphertext attack (see Section 2.6 on page 127). We have already seen that factoring an RSA modulus allows the breaking of the cryptosystem, but the converse is not known. In other words, it is not known if there are other methods of breaking RSA.

Although it is not easy to prove, it can be shown that computing the deciphering exponent d in RSA is computationally equivalent to factoring the modulus, namely they have the same complexity. (See [63, p. 65].) To state this in another fashion: knowing how to factor the modulus allows us to compute d, and knowing how to compute d can be converted into an algorithm for factoring the modulus.

▼ Authentication

We conclude this section with a discussion of mechanisms for authentication, meaning that both the origin of data and the entity who sent it must somehow be verified. To see why this is necessary in public-key cryptography, one may witness one type of interception by a cryptanalyst as described in the following example.

Example 4.5 Suppose that Mallory wants to decipher

$$c \equiv m^e \pmod{n}$$

to recover plaintext m enciphered using RSA and sent by Alice to Bob. Furthermore, suppose that Mallory can intercept and disguise c by selecting a random $x \in (\mathbb{Z}/n\mathbb{Z})^*$ and computing

$$\overline{c} \equiv cx^e \pmod{n}.$$

Not knowing this, Bob computes

$$\overline{m} \equiv \overline{c}^d \pmod{n}$$

and sends it to Alice. Now m can be recovered if Mallory intercepts $\overline{m}$ as follows. Since

$$\overline{m} \equiv \overline{c}^d \equiv c^d(x^e)^d \equiv mx \pmod{n},$$

then Mallory merely computes $m \equiv \overline{m}x^{-1} \pmod{n}$. (*This is an example of an adaptive chosen-ciphertext attack on RSA* (*see page 127*). *Such attacks on RSA can be thwarted by ensuring that the plaintext messages have a certain structure, which is unlikely to be maintained if disguised by Mallory. Then if Bob receives a ciphertext that decrypts to a plaintext without this structure, c is rejected by Bob as being fraudulent. For numerous other attacks on RSA, see* [58].)

The problem, in general, is described and illustrated as follows. Suppose that Alice and Bob are communicating via some public-key cryptosystem, and Mallory is listening to the channel. Mallory can impersonate Bob by sending

Alice a public key e' (which Alice will assume to be the public key $e \neq e'$ of B). Then Alice sends

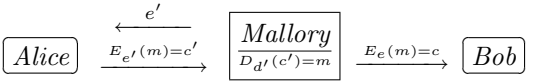
$$E_{e'} = m^{e'} = c'$$

to Bob. Mallory intercepts the enciphered message and deciphers using private key d' via

$$(c')^{d'} = m^{e'd'} = m.$$

Then Mallory enciphers m with public key e and sends $c = m^e$ to Bob, impersonating Alice, and neither Alice nor Bob is the wiser. This process is illustrated in the following.

Diagram 4.5 Impersonation Attack on Public-Key Cryptosystems



Before formalizing the notion of a digital signature, we should compare and contrast the features of a digital signature vs. a conventional (handwritten) signature. For instance, once a conventional signature is put upon a document, then the signature becomes a physical part of that document. With a digital signature algorithm, there must be some means incorporated for "binding" the signature to the message. Also, there is the issue of verification. A conventional signature, such as that on a credit card, is verified by comparing the signature on the credit card with that on the sales slip. This is, of course, very insecure since a criminal can forge a signature. Digital signatures, however, are secure since they can be verified with a (publicly known) verification algorithm, which must be part of the signature algorithm. Since people cannot disavow signatures, the verification algorithm must take care of this in some fashion as well. Finally, copies of documents with handwritten signatures can usually be identified as distinct from the original, whereas "copies" of digital signatures are identical to each other. Hence, care must be exercised in order to prevent unauthorized reuse, for instance by binding the date to the signature.

We have already discussed some methods for authentication, called MACs (see page 125). We now look at some more in greater detail starting with a formalization of the notions surrounding authentication.

Definition 4.1 **Digital Signatures**

Let $\mathcal{M}$ be a message space and $\mathcal{K}$ be a key space.

(1) *A* digital signature *is a digital data string that associates a given $m \in \mathcal{M}$ with its sender. In other words, $\mathcal{M}$ is the set of elements to which a signer can affix a digital signature.*

(2) *Let* $\mathcal{S}$ *be a set of elements, called the* signature space (*usually bitstrings*), *of fixed length used to bind the signer to the message. A* redundancy function *is an injective function*

$$R : \mathcal{M} \mapsto \mathcal{M}_{\mathcal{S}},$$

where $\mathcal{M}_{\mathcal{S}}$ *is a set of elements called the* signing space.

For each $k \in \mathcal{K}$, *there is* digital signature transformation, *or* signature transformation, *which is an injective mapping* $sig_k : \mathcal{M}_{\mathcal{S}} \mapsto \mathcal{S}$.

Note that selection of the redundancy function is critical in signature schemes. For example, such a selection should not be made independently of sig_k *since this could compromise the security of the signature scheme. The first international standard for digital signatures published in 1991 by the International Standards Organization is called ISO/IEC 9796* (*see* [58, pp. 442–444]) *This provides secure redundancy function, which can be used with the RSA Signature Scheme described below.*

(3) *A method for producing a digital signature,* $\text{sig}_k \circ R : \mathcal{M} \mapsto \mathcal{S}$, *is called a* digital signature generation algorithm, *or* signature generation algorithm.

(4) *A* digital verification algorithm *or* verification algorithm *is a method for verifying that a digital signature is authentic.*

(5) *A* digital signature scheme, *or* signature scheme *is composed of two parts: a signature generation algorithm and a signature verification algorithm.*

(6) *A signature scheme with* message recovery *is a signature scheme for which the message being sent is not required as input to the validation algorithm. In this case, the original message is recovered from the signature itself. A signature scheme* with appendix *is a signature scheme where the message* is *required as input for the verification algorithm.*

We now describe a signature scheme related to RSA. Note that any public-key system with $\mathcal{M} = \mathcal{C}$ can be used for signatures. The sender can use the private key to encipher the message m, thereby obtaining the signature s and the receiver can decipher the signature s using the sender's public key to compare it with the message m.

◆ The RSA Signature Scheme

Let $\mathcal{M} = \mathcal{C} = \mathcal{S} = \mathcal{M}_{\mathcal{S}} = \mathbb{Z}/n\mathbb{Z}$ where $n = pq$ with $p \neq q$ randomly chosen primes. A publicly known redundancy function $R : \mathcal{M} \mapsto \mathcal{M}_s$ is chosen.

The first step in the process is that each Alice and Bob generate public and private keys using the RSA key generation algorithm given on page 172. Then the following is executed.

RSA Signature Generation and Verification
Alice signs $m \in \mathcal{M}$ and Bob verifies Alice's signature as follows.

signature generation stage:
Alice performs the following steps.

(1) Compute $R(m) = m'$.

(2) Compute
$$s = \mathrm{sig}_d(m) \equiv (m')^d \pmod{n},$$
and send s to Bob.

verification stage:
Bob performs the following steps.

(1) Obtain Alice's public-key (n, e).

(2) Compute $m' \equiv s^e \pmod{n}$.

(3) Verify that $m' \in \mathrm{img}(R)$, and reject if it is not.

(4) Recover $m = R^{-1}(m')$.

Thus, we see that the RSA Signature Scheme is a signature scheme with message recovery. We now illustrate, where as usual we must use unrealistically small values, and a trivial redundancy function, for pedagogical purposes.

Example 4.6 Let $p = 1783$ and $q = 2099$ be the primes chosen by Alice, and let $R(m) = m$ for all $m \in \mathcal{M}$. Thus, $n = 3742517 = pq$ and $\phi(n) = 3738636$. Assuming that Alice chooses $e = 9667$, then using the extended Euclidean algorithm on
$$1 = 9667 \cdot d + 3738636 \cdot x,$$
Alice computes $d = 281935$ (where $x = -729$). Therefore, Alice's public key is
$$(e, n) = (9667, 3742517)$$
and the private key is $d = 281935$. Suppose that the message to be signed is $m = 1011$. Then Alice computes
$$R(m) = m' = m = 1011,$$
and
$$\mathrm{sig}_d(m') = (m')^d = 1011^{281935} \equiv 739457 \equiv s \pmod{3742517}$$
and sends s to Bob, which completes the signature generation stage. Now Bob completes the verification stage by computing
$$s^e \equiv 739457^{9667} \equiv 1011 \equiv m \pmod{3742517}.$$
The signature and message are accepted since $m \in \mathrm{img}(R)$.

Exercises

Exercises 4.14–4.17 pertain to the RSA public-key cryptosystem described on page 172. Find the plaintext numerical value of m from the parameters given. You will first have to determine the private key d from the given data via the methodology illustrated in Example 4.4 on page 173. If repeated squaring is not employed (see page 31), then a computer utilizing a mathematical software package will be required for these calculations.

4.14. $(p, q) = (167, 547)$, $n = 91349$, $e = 5$, and $c \equiv 88291 \pmod{n}$.

4.15. $(p, q) = (211, 691)$, $n = 145801$, $e = 11$, and $c \equiv 121919 \pmod{n}$.

4.16. $(p, q) = (367, 911)$, $n = 334337$, $e = 17$, and $c \equiv 226756 \pmod{n}$.

4.17. $(p, q) = (1187, 1481)$, $n = 1757947$, $e = 13$, and $c \equiv 1757118 \pmod{n}$.

Exercises 4.18–4.21 refer to the RSA signature scheme developed on page 178. Use the given parameters in each case to first compute the encryption key e using the Euclidean algorithm on $\phi(n)$ and d. Then compute $c^e \pmod{n}$. If

$$m \equiv c^e \pmod{n},$$

then accept the signature as valid, since $\mathrm{ver}_k(m, c) = 1$. *Otherwise, reject the signature, since* $\mathrm{ver}_k(m, c) = 0$.

4.18. $n = 466727$, $\phi(n) = 465336$, $d = 296123$, $m = 10101$, and $c = 369510$.

4.19. $n = 971743$, $\phi(n) = 969760$, $d = 74597$, $m = 2134$, and $c = 689844$.

4.20. $n = 1081357$, $\phi(n) = 1079260$, $d = 571373$, $m = 7381$, and $c = 725226$.

4.21. $n = 141373$, $\phi(n) = 139968$, $d = 29467$, $m = 8872$, and $c = 32961$.

4.22. Show that the choice of ℓ in the discussion of message blocking on page 173 is maximal for unique decryption when the plaintext numerical equivalents m are bigger than n. In other words, if $k > \ell$ is chosen as the blocklength when $m > n$, then decryption will not be unique in the RSA cipher.

In Exercises 4.23–4.29, use Equation (4.3) on page 170 to find the primes p and q.

4.23. $pq = 514541$ and $\phi(pq) = 513084$.

4.24. $pq = 1009427$ and $\phi(pq) = 1007400$.

4.25. $pq = 1737251$ and $\phi(pq) = 1734600$.

4.26. $pq = 3660479$ and $\phi(pq) = 3656640$.

4.27. $pq = 2579047$ and $\phi(pq) = 2575600$.

4.28. $pq = 7863043$ and $\phi(pq) = 7855120$.

4.29. $pq = 3398909$ and $\phi(pq) = 3394560$.

4.4 ElGamal

The title for this section is the name of a major contributor to several cryptographic schemes. Much of the following is taken from [64].

Biography 4.6 Taher ElGamal *was born in Cairo, Egypt, on August 18, 1955. He obtained his bachelor's degree in electrical engineering from Cairo University in 1977. He obtained both his master's degree and his Ph.D. from Stanford University in 1981 and 1984, respectively. His doctorate was done under the supervision of Martin Hellman* (*see Biography 4.1 on page 158*). *While at Stanford, he helped pioneer digital signatures and PKC. He founded Security Inc. in 1988, which later became the Kroll-O'Gara Information Security Group, where he became president of its Information Security Group. From 1991 to 1993, ElGamal was the Director of Engineering at RSA Security Inc., where he produced the RSA cryptographic toolkits and the initial VeriSign certificate issuance products. From 1993 to 1995, he was Vice President of Advanced Technologies at OKI Electric. From 1995 to 1998, he held the position of Chief Scientist of Netscape Communications where he was a pioneer in Internet security technology. Other accomplishments include development of Internet credit card payment schemes. He also serves on various boards of directors and has been a member of the technical staff at Hewlett-Packard Laboratories since 1984. ElGamal is a respected leader in the worldwide information security industry.*

The following cryptographic scheme bases its security upon the DLP (see (4.1), page 165). The cryptosystem was first published in [26] in 1985.

The following is performed assuming that Alice wants to send a message m to Bob, and $m \in \{0, 1, \ldots, p-1\}$ (equivalent to the actual plaintext).

(I) ElGamal Key Generation

1. Bob chooses a large random prime p and a primitive root α modulo p.
2. Bob then chooses a random integer a with $2 \leq a < p-1$ and computes $\alpha^a \pmod{p}$.
3. Bob's public key is (p, α, α^a) and his private (session) key is a.

(II) ElGamal Public-Key Cipher

Enciphering stage:

1. Alice obtains Bob's public key (p, α, α^a).
2. She chooses a random natural number $b < p-1$.
3. She computes $\alpha^b \pmod{p}$ and $m\alpha^{ab} \pmod{p}$.

4. Alice then sends the ciphertext $c = (\alpha^b, m\alpha^{ab})$ to Bob.

Deciphering stage:

1. Bob uses his private key to compute $(\alpha^b)^{-a} \equiv (\alpha^b)^{p-1-a} \pmod{p}$.
2. Then he deciphers m by computing $(\alpha^b)^{-a} m\alpha^{ab} \pmod{p}$.

Example 4.7 Suppose that Alice wants to send the message $m = 2132$ to Bob using the ElGamal cipher. Bob chooses $p = 3359$, $\alpha = 11$, and $a = 5$, his private key. He computes $\alpha^a \equiv 11^5 \equiv 3178 \pmod{p}$. Bob's public key is therefore $(p, \alpha, \alpha^a) = (3359, 11, 3178)$, which Alice downloads from some public database. She chooses $b = 69$ and computes both

$$\alpha^b \equiv 11^{69} \equiv 193 \pmod{p}$$

and

$$m\alpha^{ab} \equiv 2132 \cdot 3178^{69} \equiv 2719 \pmod{p}.$$

The ciphertext is $c = (193, 2719)$, which Alice sends to Bob. He uses his private key to compute

$$(\alpha^b)^{p-1-a} \equiv 193^{3353} \equiv 2243 \pmod{p},$$

and

$$(\alpha^b)^{-a} m\alpha^{ab} \equiv 2243 \cdot 2719 \equiv 2132 \pmod{p},$$

thereby recovering m.

▼ Analysis

Key Generation Options: Although it is preferable in step 1 of key generation, one need not choose a primitive root, provided one chooses an element $\alpha \in (\mathbb{Z}/p\mathbb{Z})^*$ whose order is close to the size of p. In other words, the smallest $r \in \mathbb{N}$ such that $\alpha^r \equiv 1 \pmod{p}$ must be nearly as large as p. Such α are called *near-primitive roots*. In the case of a primitive root, $r = p - 1$.

Security Issues: The random number b generated by Alice in step 2 of the enciphering stage must be kept secret since one can recover

$$m = m\alpha^{ab}(\alpha^a)^{-b}$$

from knowledge of it, given that $m\alpha^{ab}$ and α^a are made public. Furthermore, b should never be used twice. Suppose that Alice uses b for two different messages m_1 and m_2, and Eve knows m_1. Then this is how Eve can obtain m_2. The two ciphertexts are $c_1 = (\alpha^b, m_1\alpha^{ab})$ and $c_2 = (\alpha^b, m_2\alpha^{ab})$. Then she calculates that

$$m_2\alpha^{ab} m_1^{-1} m_1 \alpha^{-ab} = m_2.$$

We conclude the discussion of security issues with a detailed argument to show that indeed the security of the ElGamal cipher is based on the DLP.

To see this, we demonstrate first that the ElGamal cipher is equivalent to the Diffie-Hellman Key-Exchange protocol. Assume Eve can solve the DHP (see page 167), and she desires to get m from $c = (\alpha^b, m\alpha^{ab})$. Since she can solve the DHP, she can determine $\beta \equiv \alpha^{ab} \pmod{p}$ from α^a and α^b. Therefore, she can reconstruct the message $m \equiv \beta^{-1} m\alpha^{ab} \pmod{p}$, In other words, if Eve can break the Diffie-Hellman cipher, she can break the ElGamal.

Now assume that Eve can cryptanalyze the ElGamal cipher above. Then she can obtain any message m from knowledge of $p, \alpha, \alpha^a, \alpha^b$, and $m\alpha^{ab}$. If Eve wants to get α^{ab} from $p, \alpha, \alpha^a, \alpha^b$, she computes

$$(m\alpha^{ab})m^{-1} \equiv \alpha^{ab} \pmod{p}.$$

In other words, we have shown that cryptanalyzing ElGamal is tantamount to cryptanalyzing Diffie-Hellman. In fact, the ElGamal cipher may be viewed as a Diffie-Hellman key exchange on $k = \alpha^{ab}$, which is used to encrypt m in step 3 of the enciphering stage. Thus, we have demonstrated that although Diffie-Hellman is not itself a public-key cryptosystem, it is the basis for the ElGamal public-key cryptosystem. Furthermore, ElGamal's cipher has difficulty equivalent to the Diffie-Hellman key exchange. Moreover, as noted on page 167, if Eve can solve the DLP, she can solve the DHP. The converse is not known, but the consensus is that it is true. Hence, we assume that the security of the ElGamal cipher is based on the DLP. Last, as with RSA, a modulus of 1024 to 2048 bits is recommended for long-term security.

Deciphering Verification: The reason Bob's deciphering stage works is because

$$(\alpha^b)^{-a} m\alpha^{ab} \equiv m\alpha^{ab-ab} \equiv m \pmod{p}.$$

In 1985, ElGamal developed a signature scheme (see [26]–[27]). It turns out that these publications were perhaps a little hasty since he had not applied for patent rights, thereby forfeiting his rights to those patents. Variations of ElGamal's scheme did get patented by others such as Schnorr (see [55, pp. 180–181]). Also, RSA is a signature scheme with message recovery, whereas ElGamal's is a signature scheme with appendix.

◆ ElGamal Signature Scheme

The goal is for Alice to sign and send a message to Bob for verification. The message should be hashed before signing, but for the sake of simplicity we will not do this and leave the issue for a discussion in the analysis after the description of the signature scheme.

Key Generation Stage: First Alice engages in ElGamal key generation as described on page 181 for Bob. Thus, Alice's public key is (p, α, y), α being a primitive root modulo a large random prime p (with intractable DLP in $\mathbb{F}_p$) and her private key is a, where $y \equiv \alpha^a \pmod{p}$. The message to be signed is $m \in \mathbb{F}_p^*$.

Signing Stage: Alice performs each of the following:

1. Select a random $r \in (\mathbb{Z}/(p-1)\mathbb{Z})^*$.
2. Compute $\beta \equiv \alpha^r \pmod{p}$ and $\gamma \equiv (m - a\beta)r^{-1} \pmod{p-1}$.
3. For $k = (p, \alpha, a, y)$ the signed message $\text{sig}_k(m, r) = (\beta, \gamma)$ is sent, along with m, to Bob.

Verification Stage: Bob does each of the following:

1. Using Alice's public key (p, α, y) verify that $\beta \in \mathbb{F}_p^*$ and reject if not.
2. Compute $\delta \equiv y^\beta \beta^\gamma \pmod{p}$ and $\sigma \equiv \alpha^m \pmod{p}$.
3. $\text{ver}_k(m, (\beta, \gamma)) = 1$ if and only if $\sigma \equiv \delta \pmod{p}$. Otherwise reject.

Example 4.8 Let $p = 3469$, with primitive root $\alpha = 2$. Alice selects $a = 153$ as her private key and computes $\alpha^a \equiv 2^{153} \equiv 2501 \equiv y \pmod{3469}$. Thus, her public key is $(p, \alpha, y) = (3469, 2, 2501)$. If $m = 1121$, and she chooses $r = 251$, then she computes

$$\beta \equiv 2^{251} \equiv 2142 \pmod{3469}.$$

Then she computes

$$\gamma \equiv (m - a\beta)r^{-1} \equiv (1121 - 153 \cdot 2142) \cdot 251^{-1} \equiv 1849 \pmod{3468},$$

and sends $\text{sig}_k(1121, 251) = (\beta, \gamma) = (2142, 1849)$ to Bob. First Bob verifies that $\beta \in (\mathbb{Z}/p\mathbb{Z})^*$, then computes

$$\delta \equiv y^\beta \beta^\gamma \equiv 2501^{2142} 2142^{1849} \equiv 1487 \equiv 2^{1121} \equiv \alpha^m \pmod{3469},$$

so Bob accepts the signature as valid.

▼ Analysis

Suppose that Mallory tries to forge Alice's signature on m by choosing a random $r_1 \in (\mathbb{Z}/(p-1)\mathbb{Z})^*$ and computing $\beta' \equiv \alpha^{r_1} \pmod{p}$. Mallory is now in the position of having to compute

$$\gamma' \equiv (m - a\beta')r_1^{-1} \pmod{p-1}.$$

However, if the DLP in $\mathbb{F}_p$ is intractable, then this computation is infeasible so only a guess at the value of γ' is possible with a probability of success being $1/p$. For large p, this is insignificant.

As noted prior to the description of the ElGamal scheme, we purposely did not hash the message. However, one *must* hash the message or else Mallory can forge a signature on a random message. Here is how he does it.

Suppose that Mallory selects $r_1, r_2 \in (\mathbb{Z}/(p-1)\mathbb{Z})^*$. He then computes

$$\beta_1 \equiv \alpha^{r_1} y^{r_2} \pmod{p} \quad \text{and} \quad \gamma_1 \equiv -\beta_1 r_2^{-1} \pmod{p-1}.$$

Now we show that (β_1, γ_1) is a valid signature for the message

$$m_1 \equiv \gamma_1 r_1 \pmod{p-1}.$$

We have, $y^{\beta_1}\beta_1^{\gamma_1} \equiv \alpha^{a\beta_1}\alpha^{(r_1+ar_2)\gamma_1} \equiv \alpha^{a\beta_1}\alpha^{(r_1+ar_2)(-\beta_1 r_2^{-1})} \equiv$

$$\alpha^{a\beta_1}\alpha^{-\beta_1 r_1 r_2^{-1}-\beta_1 a} \equiv \alpha^{-\beta_1 r_1 r_2^{-1}} \equiv \alpha^{\gamma_1 r_1} \equiv \alpha^{m_1} \pmod{p}.$$

In any case, a hash function h must be applied to the original message, and the hash is signed. Thus, Mallory would have to find a message m' such that $h(m') = m$, which he has a very low probability of doing if h is strongly collision resistant. However, if step 1 of the verification stage is not enforced, then Mallory can forge certain signatures of his own choosing if he has a previous legitimate message signed by Alice, as demonstrated in the following.

Suppose that a previous legitimate signature by Alice for a message m is (β, γ). Furthermore, suppose that Mallory is lucky and $m^{-1} \pmod{p-1}$ exists, and Mallory chooses a message m_1 to forge. Mallory computes both congruences

$$t \equiv m_1 m^{-1} \pmod{p-1}$$

and

$$\gamma_1 \equiv t\gamma \pmod{p-1}.$$

By the Chinese Remainder Theorem, he can also compute a solution $x = \beta_1$ to the congruences:

$$x \equiv \beta t \pmod{p-1} \text{ and } x \equiv \beta \pmod{p}.$$

Thus,

$$y^{\beta_1}\beta_1^{\gamma_1} \equiv \alpha^{a\beta_1}\beta^{t\gamma} \equiv \alpha^{a\beta t}\alpha^{rt\gamma} \equiv \alpha^{t(a\beta+r\gamma)} \equiv \alpha^{tm} \equiv \alpha^{m_1 m m^{-1}} \equiv \alpha^{m_1} \pmod{p}.$$

Hence, (β_1, γ_1) is accepted as a valid signature by the verification stage for m_1, if step 1 in that stage is ignored. The essential nature of step 1 in the verification stage was first observed in [7].

The value r chosen by Alice in the signing stage has to be kept secret, or there is a total break of the system since Mallory can get a from knowledge of r. Since β, γ, and m are known, then knowledge of r means that he may compute

$$a \equiv (m - r\gamma)\beta^{-1} \pmod{p-1}.$$

Also, if Alice is careless and uses r for the signing of two different messages, then Mallory can get r and break the system as above. Here is how he gets r.

If $\text{sig}_k(m_1, r) = (\beta, \gamma_1)$ and $\text{sig}_k(m_2, r) = (\beta, \gamma_2)$, then

$$y^{\beta}\beta^{\gamma_1} \equiv \alpha^{a\beta+r\gamma_1} \equiv \alpha^{m_1} \pmod{p},$$

and

$$y^{\beta}\beta^{\gamma_2} \equiv \alpha^{a\beta+r\gamma_2} \equiv \alpha^{m_2} \pmod{p}.$$

Therefore,

$$\alpha^{m_2-m_1} \equiv \beta^{\gamma_2-\gamma_1} \equiv \alpha^{r(\gamma_2-\gamma_1)} \pmod{p}.$$

Hence,

$$m_2 - m_1 \equiv r(\gamma_2 - \gamma_1) \pmod{p-1}.$$

If $\gcd(p-1, \gamma_2-\gamma_1) = g$, then

$$\frac{m_2-m_1}{g} \equiv \frac{r(\gamma_2-\gamma_1)}{g} \pmod{(p-1)/g}.$$

Thus,

$$r \equiv \left(\frac{m_2-m_1}{g}\right)\left(\frac{\gamma_2-\gamma_1}{g}\right)^{-1} \pmod{(p-1)/g},$$

since $\gcd((\gamma_2-\gamma_1)/g, (p-1)/g) = 1$, and once we have r we can get a as above.

Exercises

In Exercises 4.30–4.33, use the ElGamal public-key cryptosystem described on page 181 to recover the plaintext m from the ciphertext c via the parameters given by the prime p and Bob's private key a in each instance.

4.30. $p = 1213$, $a = 15$, $c = (\alpha^b, m\alpha^{ab}) = (661, 193)$.

4.31. $p = 1973$, $a = 71$, $c = (\alpha^b, m\alpha^{ab}) = (596, 146)$.

4.32. $p = 2833$, $a = 17$, $c = (\alpha^b, m\alpha^{ab}) = (522, 982)$.

4.33. $p = 3359$, $a = 19$, $c = (\alpha^b, m\alpha^{ab}) = (1093, 2530)$.

Exercises 4.34–4.37 pertain to the ElGamal signature scheme delineated on pages 183 and 184. For the given parameters, determine if Bob should accept the signature as valid.

4.34. for $p = 641$, $\alpha = 3$, $y = 88$, $\beta = 480$, and $\gamma = 532$, so Alice sends, $m = 121$ and $\text{sig}_k(m, r) = (\beta, \gamma) = (480, 532)$.

4.35. for $p = 3023$, $\alpha = 5$, $y = 2391$, $\beta = 335$, and $\gamma = 2367$, so Alice sends, $m = 203$ and $\text{sig}_k(m, r) = (\beta, \gamma) = (335, 2367)$.

4.36. for $p = 5023$, $\alpha = 3$, $y = 3796$, $\beta = 2294$, and $\gamma = 3740$, so Alice sends, $m = 444$ and $\text{sig}_k(m, r) = (\beta, \gamma) = (2294, 3740)$.

4.37. for $p = 7481$, $\alpha = 6$, $y = 5979$, $\beta = 1723$, and $\gamma = 7045$, so Alice sends, $m = 487$ and $\text{sig}_k(m, r) = (\beta, \gamma) = (1723, 7045)$.

4.5 DSA — The DSS

We close this chapter with a description and discussion of the first Digital Signature Standard recognized by any government. In the 1990's, NIST proposed the *Digital Signature Standard* (DSS). Although this evolved into a new standard in the twenty-first century, for simplicity we present the original standard here (see [33] for the current DSS, which uses key sizes of 1024 bits or *more*).

◆ Digital Signature Algorithm (DSA — the DSS)

Setup Stage:

1. Alice selects a prime q with 160 bits. Then she selects a prime p with bitlength a multiple of 64 between 512 and 1024, satisfying the property that q divides $p-1$.
2. She chooses an $\alpha \in \mathbb{F}_p^*$ of order q modulo p. This can be done, for instance, by selecting a primitive root a modulo p and setting $\alpha \equiv a^{(p-1)/q} \pmod{p}$.
3. A cryptographic hash function $h : \mathbb{F}_q^* \mapsto \mathcal{B}_{160}$ (bitstrings of length 160) is selected. She chooses a private key $e \in \mathbb{N}$ such that $e < q$ and computes $\beta \equiv \alpha^e \pmod{p}$.
4. She publishes (p, q, α, β) and keeps private her key e.

Signing Stage: Alice performs the following in order to sign a message $m \in \mathbb{F}_q^*$. In what follows, we will assume that any powers of α or β have been reduced modulo p before being used in any congruence modulo q:

1. Select a random $r \in \mathbb{N}$ such that $r \leq q-1$.
2. Compute $\gamma \equiv \alpha^r \pmod{q}$.
3. Compute $\sigma \equiv r^{-1}(h(m) + e\gamma) \pmod{q}$.
4. Alice sends m and $\text{sig}_k(m, r) = (\gamma, \sigma)$ to Bob.

Verification Stage: Bob executes the following steps:

1. Obtain Alice's public data (p, q, α, β).
2. Compute $\delta_1 \equiv \sigma^{-1}h(m) \pmod{q}$ and $\delta_2 \equiv \sigma^{-1}\gamma \pmod{q}$.
3. Compute $\delta \equiv \alpha^{\delta_1}\beta^{\delta_2} \pmod{q}$.
4. $\text{ver}_k(m, (\gamma, \sigma)) = 1$ if and only if $\delta \equiv \gamma \pmod{q}$, in which case Bob accepts, and rejects otherwise.

▼ Analysis

First we show why, in step 4 of the verification stage, the criterion actually verifies Alice's signature. It does so since, first of all,

$$\delta_1 + e\delta_2 \equiv \sigma^{-1}h(m) + e\sigma^{-1}\gamma \equiv \sigma^{-1}(h(m) + e\gamma) \equiv r \,(\mathrm{mod}\ q),$$

then

$$\gamma \equiv (\alpha^r \,(\mathrm{mod}\ p)) \equiv (\alpha^{\sigma^{-1}h(m)+e\sigma^{-1}\gamma} \,(\mathrm{mod}\ p)) \equiv (\alpha^{\delta_1}\beta^{\delta_2} \,(\mathrm{mod}\ p)) \equiv \delta \,(\mathrm{mod}\ q).$$

Of course, the key e must be kept private or the scheme can be broken, since anyone in possession of e can sign any data and thereby impersonate Alice. Moreover, if r is used more than once, e can be recovered by a cryptanalyst (easily verified, given our many previous related discussions on such matters).

In order to see why the DSA depends upon the DLP for its security, we look at step 2 of the setup stage. Since the Silver-Pohlig-Hellman attack (see Appendix E) is useless against large prime factors of $p-1$, then this is sufficient to thwart such attacks, and computing r from knowledge of the public γ is deemed to be computationally infeasible. This is the DLP. Moreover, the reader may wonder why we did not just choose a primitive root a modulo p rather than

$$\alpha \equiv a^{(p-1)/q} \,(\mathrm{mod}\ p).$$

The reason is that it is a generally held opinion that many pieces of information about divisors of $p-1$ can collectively add up to something useful, so DSA avoids this potential problem by keeping all congruences as modulo q data in the signing and verification stages.

An advantage of DSA is that in a precomputation stage the exponentiation of α can be done offline and need not be part of the signature generation. Another positive feature is that DSA has relatively short signatures of 320 bits, so the signing can be done efficiently. Some disadvantages of DSA include the fact that it cannot be used for key exchange. Moreover, the modulus at a mere 512 bits can be a drawback for security, so the prime p should actually be chosen such that $2^{1023} < p < 2^{1024}$ for long-term security. There is another potential problem that one would not imagine and is difficult to detect, namely, the building of a *subliminal channel* into DSA. This is a method of signing an innocuous message with subliminal bits hidden in it. This could be as little as one bit per message or as much as two bytes per message. For the reader interested in how this is done in detail see [91, pp. 300–301].

DSA evolved into the new *Digital Signature Standard* in FIPS 186-1 announced by NIST on December 15, 1998, and this included the RSA digital signature scheme. On February 15, 2000, NIST announced the approval of FIPS 186-2, and this included the upgraded Digital Signature Standard (DSS), the RSA digital signature standard, and the *Elliptic Curve Digital Signature Algorithm* (ECDSA).

The governmental plans for DSA are akin to that of the role played by DES. They include applications such as cash transactions, data exchange, data storage, electronic mail, and software distribution, to mention a few.

Chapter 5

Primality Testing

5.1 True Primality Tests

On page 7, we informally agreed upon a definition of primality testing. However, that definition is only one of the types of primality testing, which we now formalize. The following is distinct from the probabilistic kinds that we will study in Section 5.2.

Definition 5.1 **Primality Proofs**

A Primality Proving Algorithm, *also known as a* True Primality Test, *is a deterministic algorithm* (*see Biography 1.4 on page 8*) *that, given an input* n*, verifies the hypothesis of a theorem whose conclusion is that* n is prime. *A* Primality Proof *is the computational verification of such a theorem. In this case, we call* n *a* provable prime — *a prime that is verified by a Primality Proving Algorithm.*

The classical example of a True Primality Test is the following. First, recall that a Mersenne number is one of the form

$$M_n = 2^n - 1.$$

◆ **Lucas-Lehmer True Primality Test For Mersenne Numbers**

The algorithm consists of the following steps performed on an input Mersenne number $M_n = 2^n - 1$ with $n \geq 3$.

(1) Set $s_1 = 4$ and compute $s_j \equiv s_{j-1}^2 - 2 \pmod{M_n}$ for $j = 2, \ldots, n-1$.

(2) If $s_{n-1} \equiv 0 \pmod{M_n}$, then conclude that M_n is prime. Otherwise, conclude that M_n is composite.

Example 5.1 Input $M_{13} = 8191$. Then we compute $\overline{s_j}$, the least nonnegative residue of s_j modulo M_{13}, as follows: $\overline{s_2} = 14$, $\overline{s_3} = 194$, $\overline{s_4} = 4870$, $\overline{s_5} = 3953$, $\overline{s_6} = 5970$, $\overline{s_7} = 1857$, $\overline{s_8} = 36$, $\overline{s_9} = 1294$, $\overline{s_{10}} = 3470$, $\overline{s_{11}} = 128$, and $\overline{s_{12}} = 0$. Thus, M_{13} is prime by the Lucas-Lehmer Test. (Exercises 5.2– 5.3 on page 197 pertain to the Lucas-Lehmer Test and Mersenne numbers.)

We now look at a primality test on n for which a partial factorization of $n-1$ needs to be known in order to determine if it is prime — see Biography 5.1 on the facing page.

Theorem 5.1 **Pocklington's Theorem**

Let $n = ab + 1 \in \mathbb{N}$ with $a, b \in \mathbb{N}$, $b > 1$ and suppose that for every prime divisor q of $b > 1$ there exists an integer m such that $m^{n-1} \equiv 1 \pmod{n}$ and $\gcd(m^{(n-1)/q} - 1, n) = 1$. Then $p \equiv 1 \pmod{b}$ for every prime $p \mid n$. Furthermore, if $b > \sqrt{n} - 1$, then n is prime.

Proof. Let $p \mid n$ be prime and set $c = m^{(n-1)/q^e}$ where q is a prime and $e \in \mathbb{N}$ with $q^e || b$ (see Footnote 1.6 on page 49). Therefore, since

$$\gcd(m^{(n-1)/q} - 1, n) = 1,$$

then $c^{q^e} \equiv 1 \pmod{p}$, but $c^{q^{e-1}} \not\equiv 1 \pmod{p}$. Thus, $\mathrm{ord}_p(c) = q^e$, so q^e divides $(p-1)$ by Proposition 1.5 on page 44. Since q was arbitrarily chosen, then $p \equiv 1 \pmod{b}$. For the last assertion of the theorem, assume that $b > \sqrt{n} - 1$ and that n is composite. Let p be the smallest prime dividing n. Then $p \le \sqrt{n}$, so $\sqrt{n} \ge p > b \ge \sqrt{n}$, a contradiction. Hence, n is prime. □

Example 5.2 Suppose that we wish to test $n = 54419$ for primality using Pocklington's Theorem knowing that $n - 1 = 2 \cdot 27209$, where 27209 is prime, and if $b = 27209 = q$, with $a = m = 2$, then $m^{n-1} = 2^{n-1} \equiv 1 \pmod{n}$ but $\gcd(m^{(n-1)/q} - 1, n) = \gcd(3, 54419) = 1$, so n is prime. (See Exercises 5.4–5.5.)

Another figure involved in the development of primality testing was Proth — see Biography 5.2 on page 192.

Theorem 5.2 **Proth's Theorem**

Let $k, t \in \mathbb{N}$ with t odd and $2^k > t$. Then $n = 2^k t + 1$ is prime if and only if $c^{(n-1)/2} \equiv -1 \pmod{n}$, where c is a quadratic nonresidue modulo n.

Proof. If n is prime, then by Euler's Criterion given in Theorem 1.21 on page 53

$$c^{(n-1)/2} \equiv -1 \pmod{n}.$$

Conversely, we may invoke Theorem 5.1 with $a = t$, $b = 2^k$ and $m = c$ to conclude that if

$$c^{(n-1)/2} \equiv -1 \pmod{n},$$

then n is prime. □

Example 5.3 Suppose that we wish to use Proth's Primality Test on $n = 7681$. Since $n = 7681 = 2^9 \cdot 15 + 1$ and $2^9 = 512 > t = 15$, then we have satisfied the hypothesis of Theorem 5.2. Since 13 is a quadratic nonresidue modulo n, and $13^{3840} = 13^{(n-1)/2} \equiv -1 \pmod{n}$, then n is prime. See Exercises 5.6– 5.7 on page 197.

The next result involves Fermat numbers, which we introduced in Exercise 1.90 on page 41.

Theorem 5.3 **Pepin's Primality Test**

For $n \in \mathbb{N}$, $\mathfrak{F}_n = 2^{2^n} + 1$ is prime if and only if $5^{(\mathfrak{F}_n - 1)/2} \equiv -1 \pmod{\mathfrak{F}_n}$.

Proof. Using Proth's Theorem with $t = 1$, the result will follow if we can show that 5 is a quadratic nonresidue modulo $\mathfrak{F}_n$ for any $n \geq 2$. By a simple induction argument one may verify that $2^{2^n} \equiv 1 \pmod{5}$ for all $n \geq 2$. Thus, $\mathfrak{F}_n \equiv 2 \pmod{5}$ for any $n \geq 2$. Hence, using the Quadratic Reciprocity Law, Theorem 1.23 on page 61, we have the Legendre Symbol equality,

$$\left(\frac{5}{\mathfrak{F}_n}\right) = \left(\frac{\mathfrak{F}_n}{5}\right) = \left(\frac{2}{5}\right) = -1,$$

where the last equality comes from Proposition 1.8 on page 63. □

Biography 5.1 Henry Cabourn Pocklington (1870–1952) *worked mainly in physics, the discoveries in which got him elected as a Fellow of the Royal Society. His professional career was spent as a physics teacher at Leeds Central Higher Grade School in England up to his retirement in 1926. Nevertheless, his six papers in number theory were practical and innovative. See* [81] *for more detail. As the proof of Theorem 5.2 shows, Pocklington's result is more general. In fact, it turns out that it was Pocklington who generalized Proth's result, but he did so without being aware of Proth's work. Thus, one may say that Pocklington was an enlightened amateur who was not aware of the history of number theory that was, after all, merely a hobby for him.*

Example 5.4 $\mathfrak{F}_4 = 65537$ is prime since

$$5^{(\mathfrak{F}_4 - 1)/2} = 5^{32768} \equiv -1 \pmod{\mathfrak{F}_4}.$$

Remark 5.1 *Pepin's Test was generalized by Hurwitz* [42] *in 1896 and by Carmichael* [14] *in 1913 as follows. If $r \in \mathbb{N}$ and $\Phi_r(x)$ is the r^{th} cyclotomic polynomial, then r is prime if and only if there exists an $s \in \mathbb{N}$ such that $\Phi_{r-1}(s) \equiv 0 \pmod{r}$. Pepin's Test is the special case where $r = \mathfrak{F}_n$ and*

$$\Phi_{r-1}(x) \equiv x^{(\mathfrak{F}_n - 1)/2} + 1 \equiv 0 \pmod{\mathfrak{F}_n}.$$

See [62, Exercise 2.23, p. 87].

We now engage in a discussion of what unifies the algorithms presented in this section, and what contrasts them. The Lucas-Lehmer Primality Test for

Mersenne numbers, Pepin's Primality Test, and Proth's Test are all polynomial-time algorithms, which are True Primality Tests. However, they lack generality since they provide only provable primes for special numbers, namely Mersenne numbers, Fermat numbers, and numbers of the form $2^k t + 1$ where $2^k > t$, respectively. On the other hand, Pocklington's Theorem relies on a knowledge of at least a partial factorization of $n - 1$. This is also a True Primality Test, since it produces provable primes. However, since the test requires knowledge of factorization (and as we have seen there are no known fast factorization algorithms), then this algorithm does not run in polynomial time . In fact, there is no such test (requiring a knowledge of factorization) that runs in polynomial time.

Biography 5.2 François Proth (1852–1879) *was a self-taught farmer who lived in the village of Vaux devant Damloup near Verdun, France. The theorem that we prove here is one of four results, which he produced, that can be used for primality testing. Proth probably had proofs of his results, but he did not produce them. For more information see* [36] *and* [74].

What underlies the above discussion is essentially a result, encountered on page 36, namely Fermat's Little Theorem 1.16, which says that if p is prime then $a^{p-1} \equiv 1 \pmod{p}$ for all $a \in \mathbb{Z}$ with $\gcd(a, p) = 1$. The converse of this theorem, if it were true, would provide a simple and fast method for obtaining provable primes. However, the converse fails in general and counterexamples abound. In fact, it has recently been shown that it fails in the worst possible way, infinitely often. What we mean by the "worst possible way" is that there exist composite integers n such that

$$a^{n-1} \equiv 1 \pmod{n} \text{ for } \textit{any} \text{ integers } a \text{ relatively prime to } n, \tag{5.1}$$

such as $n = 561$. Composite integers n satisfying (5.1) are called *Carmichael numbers*, which we introduced in Exercise 1.103 on page 43. In 1994, it was proved that there exist infinitely many Carmichael numbers, (see [2]). Nevertheless, one can use the converse of Fermat's Little Theorem to prove primality of n if one can find an element of order $n-1$ in $(\mathbb{Z}/n\mathbb{Z})^*$, namely find a primitive root modulo n (see Exercises 1.110–1.111 on page 50). Credit for the following result is essentially due to Lehmer, although seeds of it may be found in the work of Proth and Lucas. (See Biographies 1.18–1.19.)

Theorem 5.4 **Proofs Via the Converse of Fermat's Little Theorem**

If $n \in \mathbb{N}$ with $n \geq 3$, then n is prime if and only if there is an $m \in \mathbb{N}$ such that

$$m^{n-1} \equiv 1 \pmod{n},$$

but

$$m^{(n-1)/q} \not\equiv 1 \pmod{n}$$

for any prime $q \mid (n-1)$.

Proof. If n is prime, then the result follows from Exercise 1.110 on page 50 and Fermat's Little Theorem 1.16 on page 36. Conversely, let $n = ab+1$. Then one of a or b is bigger than $\sqrt{n}-1$, since otherwise,

$$n = ab + 1 \leq (\sqrt{n} - 1)^2 + 1 = n - 2\sqrt{n} + 2,$$

which implies that $\sqrt{n} \leq 1$, so $n = 1$, a contradiction. Hence, we may assume without loss of generality that $b > \sqrt{n} - 1$, and the result now follows from Pocklington's Theorem 5.1 on page 190. □

The following is a variation of Theorem 5.4 given by Brillhart and Selfridge in [10]. (See Exercises 5.11–5.12.)

Corollary 5.1 *$n \in \mathbb{N}$ with $n \geq 3$ is prime if and only if, for each prime q dividing $n-1$, there exists an integer m_q such that*

$$m_q^{n-1} \equiv 1 \,(\text{mod } n)$$

and

$$m_q^{(n-1)/q} \not\equiv 1 \,(\text{mod } n).$$

Proof. If n is prime, then by Theorem 5.4 we have the result with $m = m_q$ for all $q|(n-1)$. Conversely, if such m_q exist, then by the Chinese Remainder Theorem we may find a solution $x = m$ to the system of congruences

$$x \equiv m_q \,(\text{mod } q^e)$$

for all primes q such that

$$q^e || (n-1).$$

Thus, the result now follows from Theorem 5.4. □

Of course, the major pitfall with Theorem 5.4 is that it requires a knowledge of the factorization of $n-1$. However, as we have seen with the algorithms in this section, it works well with special numbers such as the Fermat numbers to which Pepin's Test applies. Also, one can get a test with knowledge of only a partial factorization as in Pocklington's Theorem. In the next section, we will see how probabilistic methods are used in an effort to approach primality testing using the converse of Fermat's Little Theorem as enunciated in Theorem 5.4.

The preceding discussion shows us that the True Primality Tests, that are based upon Theorem 5.4, are broken down into two distinct categories — those that sacrifice speed and those that sacrifice generality. In the next section, which deals with "Probabilistic Primality Tests," we will see that both speed and generality are maintained, but *correctness*, obtaining provable primes, is sacrificed. Thus, in each type of primality test, *exactly* one of the following properties is sacrificed: speed, correctness, or generality.

We conclude this section with a discussion of a relatively recent result that provides an unconditional algorithm for primality testing.

◆ Primes is in P

The following is an unconditional deterministic polynomial-time algorithm for primality testing presented in [1] by M. Agrawal, N. Kayal, and N. Saxena. For notation in what follows, see Definition 1.12 on page 37 and Definition 1.14 on page 44, as well as results on polynomial rings especially as they pertain to finite fields on pages 311–316.

In what follows, $\mathbb{Z}_n$ for a given integer $n > 1$ denotes $\mathbb{Z}/n\mathbb{Z}$, and if $h(X) \in \mathbb{Z}_n[X]$, then the notation,

$$f(X) \equiv g(X) \,(\text{mod } h(X), n)),$$

is used to represent the equation $f(X) = g(X)$ in the quotient ring $\mathbb{Z}_n[X]/(h(X))$. In particular, for suitably chosen r and a, values, we will be looking at equations of the following type:

$$(X + a)^n \equiv X^n + a \,(\text{mod } X^r - 1, n). \tag{5.2}$$

Algorithm 5.1 —
Unconditional Deterministic Polynomial-Time Primality Test

Input an integer $n > 1$, and execute the following steps.

1. If $n = a^b$ for some $a \in \mathbb{N}$ and $b > 1$, then terminate with output "n is composite."
2. Find the smallest $r \in \mathbb{N}$ such that $\text{ord}_r(n) > 4 \log_2^2 n$.
3. If $1 < \gcd(a, n) < n$ for some $a \leq r$, then output "n is composite."
4. If $n \leq r$, then output "n is prime."
5. Set $a = 1$ and execute the following:
   (i) Compute $Y(a) \equiv (X + a)^n - X^n - a \,(\text{mod } X^r - 1, n)$.
   (ii) If $Y(a) \not\equiv 0 \,(\text{mod } X^r - 1, n)$, output "$n$ is composite." Otherwise, go to step (iii).
   (iii) If $Y(a) \equiv 0 \,(\text{mod } X^r - 1, n)$, set $a = a + 1$. If
   $$a < \lfloor 2\sqrt{\phi(r)} \cdot \log_2(n) \rfloor,$$
   go to step (i). Otherwise, go to step 6.
6. Output "n is prime."

▼ Analysis

The reason the authors of [1] considered equations of type (5.2) was that they were able to prove the following.

Polynomial Primality Criterion

If $a \in \mathbb{Z}$, $n \in \mathbb{N}$ with $n > 1$, and $\gcd(a, n) = 1$, then n is prime if and only if

$$(X + a)^n \equiv X^n + a \pmod{n}. \tag{5.3}$$

The satisfaction of polynomial congruence (5.3) is a simple test, but the time taken to test the congruence is too expensive. To save time, the authors looked at the congruence modulo a polynomial, whence congruence (5.2). However, by looking at such congruences, they introduced the possibility that composite numbers might satisfy (5.2), which indeed they do. Yet, the authors were able to (nearly) restore the characterization given in the above polynomial primality criterion by showing that for a suitably chosen r, if (5.2) is satisfied for several values of a, then n must be a prime power. Since the number of a values and the suitably chosen r value are bounded by a polynomial in $\log_2(n)$, they achieved a deterministic polynomial time algorithm for primality testing.

The authors of [1] were able to to establish the following facts about their algorithm. The reader will need the concepts of ceiling and floor functions (see Definition 1.3 on page 2 and Exercise 1.12 on page 4).

Facts Concerning Algorithm 5.1

1. The algorithm outputs "n is prime" if and only if n is prime. (Hence, it outputs "n is composite" if and only if n is composite.)
2. There exists an $r \leq \lceil 16 \log_2^5(n) \rceil$ such that $\operatorname{ord}_r(n) > 4 \log_2^2(n)$.
3. The asymptotic time complexity of the algorithm is $O(\log_2^{10.5+\varepsilon}(n))$ for any $\varepsilon > 0$.
4. It is conjectured that the time complexity of the algorithm can be improved to the best-case scenario where $r = O(\log_2^2(n))$, which would mean that the complexity of the algorithm would be

$$O(\log_2^{6+\varepsilon}(n)) \text{ for any } \varepsilon > 0.$$

Two conjectures support the authors' conjecture in part 4 above. They are given as follows.

Artin's Conjecture

If $n \in \mathbb{N}$ is not a perfect square, then the number of primes $q \leq m$ for which $\operatorname{ord}_q(n) = q - 1$ is asymptotically $A(n) \cdot m / \ln(m)$, where $A(n)$ is *Artin's constant* given by

$$A(n) = \prod_{j=1}^{\infty} \left(1 - \frac{1}{p_k(p_k - 1)}\right) = 0.3739558136\ldots,$$

with p_k being the kth prime.

If Artin's conjecture becomes effective for

$$m = O(\log_2^2(n)),$$

then it follows that there is an

$$r = O(\log_2^2(n))$$

with the desired properties.

The other conjecture that supports their contention is given as follows.

Sophie Germain's Prime Density Conjecture

The number of primes $q \leq m$ such that $2q+1$ is also a prime (called a *Sophie Germain prime*) is asymptotically

$$2C_2 m/\ln^2(m),$$

where C_2 is the *twin prime constant* given by

$$C_2 = \prod_{p \geq 3} \frac{p(p-2)}{(p-1)^2} \approx 0.6601611816\ldots.$$

If the Sophie Germain conjecture holds, then

$$r = O(\log_2^{2+\varepsilon}(n)) \text{ for any } \varepsilon > 0 \text{ such that } \mathrm{ord}_r(n) \geq 4\log_2^2(n).$$

Hence, the algorithm, with this r value, yields a time complexity of:

$$O(\log_2^{6+\varepsilon}(n)) \text{ for any } \varepsilon > 0.$$

The authors of [1] leave one more conjecture, the affirmative solution of which would improve the complexity of algorithm 5.1 to $O(\log_2^{3+\varepsilon}(n))$ for any $\varepsilon > 0$.

Conjecture 5.1 *If r is a prime not dividing $n > 1$ and if*

$$(X-1)^n \equiv X^n - 1 \,(\mathrm{mod}\ X^r - 1, n),$$

then either n is prime or $n^2 \equiv 1 \,(\mathrm{mod}\ r)$.

The result given in Algorithm 5.1 is a major breakthrough, and the simplicity of the approach is even more noteworthy given the attempts at finding such an algorithm through much more difficult techniques such as those discussed in the previous section. The algorithm uses essentially only elementary properties of polynomial rings over finite fields and a generalization of Fermat's Little Theorem in that context — quite impressive indeed.

Exercises

☆ 5.1. Let R be a commutative ring with identity containing $\mathbb{Z}/m\mathbb{Z}$ as a subring for a given fixed $m \in \mathbb{N}$ and assume that there is an $\alpha \in R$ such that $\alpha^s = 1$ but for all primes $p \mid s$, $\alpha^{s/p} - 1$ is a unit in R, namely it is invertible in R. Prove that if there exists a $k \in \mathbb{N}$ such that $f(x) = \prod_{j=0}^{k-1}(x - \alpha^{m^j}) \in (\mathbb{Z}/m\mathbb{Z})[x]$, then for any $r \mid s$, there exists a $j \geq 0$ such that $r \equiv m^j \pmod{s}$.

☆ 5.2. Let $M_n = 2^n - 1$ with $n \geq 3$ odd. Prove that M_n is prime if and only if $M_n \mid s_{n-1}$, where s_{n-1} is defined in the Lucas-Lehmer Algorithm given on page 189.

5.3. Prove that all Mersenne numbers M_p are relatively prime for distinct primes p.

5.4. Use Pocklington's Theorem to test $n = 104759$ for primality, knowing that the prime $q = 52379$ divides $(n - 1)$.

5.5. Test $n = 105817$ for primality using Pocklington's Theorem if you know that the prime $q = 4409$ divides $n - 1$.

5.6. Use Proth's Theorem to test $n = 13313$ for primality.

5.7. Test $n = 40961$ for primality using Proth's Theorem.

5.8. Use Theorem 5.4 on page 192 to verify that $n = 16487$ is prime.

5.9. Use Theorem 5.4 on page 192 to verify that $n = 16547$ is prime.

5.10. Given $n \in \mathbb{N}$, set $(n-1)! = q(n)n(n-1)/2 + r(n)$, where $q(n), r(n) \in \mathbb{N}$ with $0 \leq r(n) < n(n-1)/2$. In other words, $r(n)$ is the remainder after dividing $(n-1)!$ by $n(n-1)/2$. Prove that

$$\{r(n) + 1 : r(n) > 0\} = \{p : p > 2 \text{ is prime}\}.$$

(*Hint: Use Wilson's Primality Test, which says that* $n \in \mathbb{N}$ *is prime if and only if* $(n-1)! \equiv -1 \pmod{n}$.) This result was first proved by J. de Barinaga in 1912 (see [23, p. 428]).

5.11. Use Corollary 5.1 on page 193 to prove that $n = 8273$ is prime by finding and testing m_q for each prime q dividing $n - 1$.

5.12. Perform the same task as in Exercise 5.11 for $n = 9907$.

5.13. Let $n \in \mathbb{N}$ and assume that $a^{n-1} \equiv 1 \pmod{n}$ for all $a \in (\mathbb{Z}/n\mathbb{Z})^*$ with $\gcd(a, n) = 1$. Prove that n is squarefree.

5.2 Probabilistic Primality Tests

The primality tests in this section will be based upon *randomized algorithms*, namely those that make random decisions at certain points in the execution, so that the execution paths may differ each time the algorithm is invoked with the same input. This is in contrast to the primality tests in the previous section that were essentially based upon deterministic algorithms.

Employing probabilistic tests for primality, it is a basic tool to use Fermat's Little Theorem 1.16 on page 36 to rule out composites. For instance, if $n = 377$, then $2^{376} \equiv 94 \not\equiv 1 \pmod{377}$, so by Fermat's theorem, $n = 377$ is not prime. Indeed, $377 = 13 \cdot 29$. Notice that we did not have to factor n to determine this. In general, it is easier to prove compositeness than to actually factor the number. The following makes use of these facts. This is the most widely used of the probabilistic tests, so we focus solely upon it here as a highlight of such tests. The following presentation is adapted from [63]. This test is most often called the *Miller-Rabin Test* in the literature — see Biographies 5.4 on page 200 and 5.5 on page 201. However, John Selfridge was using the test in 1974 before Miller first published the result, so we credit Selfridge here with this recognition — see Biography 5.3 on page 200.

◆ The Miller-Selfridge-Rabin (MSR) Primality Test

Let $n - 1 = 2^t m$ where $m \in \mathbb{N}$ is odd and $t \in \mathbb{N}$. The value n is the input to be tested by executing the following steps, where all modular exponentiations are done using the repeated squaring method described on page 31.

(1) Choose a random integer a with $2 \leq a \leq n - 2$.

(2) Compute
$$x_0 \equiv a^m \pmod{n}.$$
If
$$x_0 \equiv \pm 1 \pmod{n},$$
then terminate the algorithm with

"n is probably prime."

If $x_0 \not\equiv \pm 1 \pmod{n}$ and $t = 1$, terminate the algorithm with

"n is definitely composite."

Otherwise, set $j = 1$ and go to step (3).

(3) Compute
$$x_j \equiv a^{2^j m} \pmod{n}.$$
If $x_j \equiv 1 \pmod{n}$, then terminate the algorithm with

"n is definitely composite."

If $x_j \equiv -1 \pmod{n}$, terminate the algorithm with

"n is probably prime."

Otherwise set $j = j + 1$ and go to step (4).

(4) If $j = t - 1$, then go to step (5). Otherwise, go to step (3).

(5) Compute

$$x_{t-1} \equiv a^{2^{t-1}m} \pmod{n}.$$

If $x_{t-1} \not\equiv -1 \pmod{n}$, then terminate the algorithm with

"n is definitely composite."

If $x_{t-1} \equiv -1 \pmod{n}$, then terminate the algorithm with

"n is probably prime."

Example 5.5 Consider $n = 1729$. Since $n-1 = 2^6 \cdot 27$, then $t = 6$ and $m = 27$. Select $a = 2$. Then

$$x_0 \equiv 2^{27} \equiv 645 \pmod{n},$$

so we set $j = 1$ and compute

$$x_1 \equiv 2^{2 \cdot 27} \equiv 1065 \pmod{n},$$

so we set $j = 2$ and compute

$$x_2 \equiv 2^{4 \cdot 27} \equiv 1 \pmod{n}.$$

Thus, by step (3) of the MSR test we may conclude that n is definitely composite. This value of $n = 1729 = 7 \cdot 13 \cdot 19$ is an example of a Carmichael number introduced in Exercise 1.103 on page 43.

Remark 5.2 *If n is composite but declared to be "probably prime" with base a by the Miller-Selfridge-Rabin test, then*

n is said to be a strong pseudoprime to base a.

Thus, the above test is often called the strong pseudoprime test *in the literature. Strong pseudoprimes to base a are much sparser than composite n for which $a^{n-1} \equiv 1 \pmod{n}$, called* pseudoprimes *to base a. An instance of the latter that is not an example of the former is given in Example 5.5, since $n = 1729$ is a pseudoprime to base 2 or, indeed to any base, since it is a Carmichael number but as the example demonstrates is not a strong pseudoprime to base*

2. *Carmichael numbers are also called* absolute pseudoprimes, *since they are pseudoprimes to any base* (*including those bases* a *for which* $\gcd(a, n) > 1$).

Although strong pseudoprimes are known to be sparser than pseudoprimes, it is also known that for any set $\mathcal{S}$ *of bases there exist infinitely many integers that are strong pseudoprimes for all elements in* $\mathcal{S}$. *It is also known that the smallest strong pseudoprime for* $\mathcal{S} = \{2, 3, 5, 7\}$ *is*

$$n = 3215031751.$$

This fact was first published in [72].

▼ Analysis

Let us look a little closer at the Miller-Selfridge-Rabin test to see why it it is possible to declare that "n is definitely composite" in step (3). If $x \equiv 1 \pmod{n}$ in step (3), then for some j with $1 \leq j < t - 1$:

$$a^{2^j m} \equiv 1 \pmod{n}, \text{ but } a^{2^{j-1} m} \not\equiv \pm 1 \pmod{n}.$$

Thus, it can be shown that $\gcd(a^{2^{j-1} m} - 1, n)$ is a nontrivial factor of n. Hence, if the Miller-Selfridge-Rabin test declares in step (3) that "n is definitely composite," then indeed it is. Another way of saying this is that if n is prime, then Miller-Selfridge-Rabin will declare it to be so. However, if n is composite, then it can be shown that the test fails to recognize n as composite with probability at most $(1/4)$.

Biography 5.3 John Selfridge *was born in Ketchican, Alaska, on February 17, 1927. He received his doctorate from U.C.L.A. in August of 1958, and became a professor at Pennsylvania State University six years later. He is a pioneer in computational number theory. The term "strong pseudoprime" was introduced by Selfridge in the mid-1970's, but he did not publish this reference. However, it did appear in a paper by Williams* [94] *in 1978.*

This is why the most we can say is that "n is probably prime." However, if we perform the test r times for r large enough, this probability $(1/4)^r$ can be brought arbitrarily close to zero. Moreover, at least in practice, using the test with a single choice of a base a is usually sufficient.

Biography 5.4 Gary Miller *obtained his Ph.D. in computer science from U.C. Berkeley in 1974. He is currently a professor in computer science at Carnegie-Mellon University. His expertise lies in computer algorithms.*

Also, in step (5), notice that we have not mentioned the possibility that

$$a^{2^{t-1} m} \equiv 1 \pmod{n}$$

specifically. However, if this did occur, then that means that in step (3), we would have determined that

$$a^{2^{t-2}m} \not\equiv \pm 1 \,(\text{mod } n),$$

from which it follows that n cannot be prime. Furthermore, by the above method, we can factor n since $\gcd(a^{2^{t-2}m} - 1, n)$ is a nontrivial factor. This final step (4) is required since, if we get to $j = t - 1$, with $x \not\equiv \pm 1 \,(\text{mod } n)$ for any $j < t - 1$, then simply invoking step (3) again would dismiss those values of $x \not\equiv \pm 1 \,(\text{mod } n)$, and this would not allow us to claim that n is composite in those cases. Hence, it allows for more values of n to be deemed composite, with certainty, than if we merely performed step (3) as with previous values of j.

Biography 5.5 Michael Rabin (1931–) *was born in Breslau, Germany (now Wroclaw, Poland), in 1931. In 1956, he obtained his Ph.D. from Princeton University where he later taught. In 1958, he moved to the Hebrew University in Jerusalem. He is known for his seminal work in establishing a rigorous mathematical foundation for finite automata theory. For such achievements, he was co-recipient of the 1976 Turing award, along with Dana S. Scott. He now divides his time between positions at Harvard and the Hebrew University in Jerusalem.*

▼ Relationship to RSA and Factoring

On page 174 we discussed the ease of factoring an RSA modulus $n = pq$ when the primes p and q are close together, in the sense defined there. What is implicit in that discussion is the following.

If we have an $n \in \mathbb{N}$ such that

$$x^2 \equiv y^2 \,(\text{mod } n) \text{ with } x \not\equiv \pm y \,(\text{mod } n) \text{ for some } x, y \in \mathbb{Z}, \tag{5.4}$$

then n is necessarily composite since $\gcd(x - y, n)$ provides a nontrivial factor of n. This idea was known to Fermat who, in 1643, developed a method of factoring based upon the following observation.

If $n = rs$ is an odd natural number with $1 < r < \sqrt{n}$, then

$$n = a^2 - b^2 \text{ where } a = (r + s)/2 \text{ and } b = (s - r)/2.$$

Thus, in order to find a factor of n, we need only look at values $x = y^2 - n$ for

$$y = \lfloor \sqrt{n} \rfloor + 1, \lfloor \sqrt{n} \rfloor + 2, \ldots, (n - 1)/2$$

until a perfect square is found. This is called *Fermat's difference of squares method*. (In Chapter 6 we will look at factoring methods in detail.)

Fermat's method is contained within the MSR test as a fundamental principle. To illustrate this fact, consider Example 5.5 on page 199 with

$$x_2 \equiv x_1^2 \equiv 1 \,(\text{mod } 1729).$$

Here $\gcd(x_1 - 1, 1729) = 133 = 7 \cdot 19$ is a nontrivial factor of n and $1729 = 7 \cdot 13 \cdot 19 = 133 \cdot 13$.

▼ How Pseudoprimes Pass MSR

We have mentioned that strong pseudoprimes are necessarily less likely to occur than pseudoprimes. We have illustrated the MSR test on pseudoprimes above, and now we compare and contrast with an example of a strong pseudoprime and explanation of the mechanism by which it escapes detection via MSR.

Consider $n = 1373653$ and $a = 2$. Since $n - 1 = 2^2 \cdot 343413 = 2^t \cdot m$, then

$$x_0 \equiv 2^m \equiv 890592 \,(\text{mod } n) \text{ and } x_1 = x_{t-1} \equiv 2^{2m} \equiv -1 \,(\text{mod } n),$$

then by step (3) of MSR, we declare that n is probably prime. However, the prime decomposition is $n = 829 \cdot 1657$. Hence, n is a strong pseudoprime. Now, we look at how this occurs in more detail.

From the above, we have that $x_0 \equiv -1 \,(\text{mod } q)$ for each of the prime divisors q of n, and similarly $x_1 \equiv 1 \,(\text{mod } q)$ for each such q. In other words, the first time each of the $x_i \equiv 1 \,(\text{mod } q)$ for each prime q dividing n is at $i = 1$. It is rare to have the sequences $x_i \,(\text{mod } q)$ reach 1 at the same time for each prime dividing n. As an instance, we look to Example 5.5, which failed to pass the MSR even though it is an absolute pseudoprime. In that case,

$$x_0 \equiv 1 \,(\text{mod } 7), x_0 \equiv 8 \,(\text{mod } 13), x_0 \equiv 1 \,(\text{mod } 19);$$

$$x_1 \equiv 1 \,(\text{mod } 7), x_1 \equiv -1 \,(\text{mod } 13), x_1 \equiv 12 \,(\text{mod } 19);$$

$$x_2 \equiv 1 \,(\text{mod } 7), x_2 \equiv 1 \,(\text{mod } 13), x_2 \equiv 1 \,(\text{mod } 19).$$

Notice: the first time $x_i \equiv 1 \,(\text{mod } 7)$ is for $i = 0$, the first time $x_i \equiv 1 \,(\text{mod } 13)$ is for $i = 2$, and the first time $x_i \equiv 1 \,(\text{mod } 19)$ is for $i = 0$. Hence, they do not all reach 1 at the same time. The scarcity of this phenomenon points to the effectiveness of the MSR test.

▼ Concluding Comments

The Miller-Selfridge-Rabin test is an example of a *Monte Carlo* algorithm, meaning a probabilistic algorithm that achieves a correct answer more than 50% of the time. More specifically, Miller-Selfridge-Rabin is a Monte Carlo algorithm for compositeness, since it provides a proof that a given input is composite but provides only some probabilistic evidence of primality. Furthermore, Miller-Selfridge-Rabin is a *yes-biased* Monte Carlo algorithm, meaning that a "yes" answer is always correct but a "no" answer may be incorrect. In this case, the answer is to the decision problem — (see page 71): "Is n composite?" A yes-biased Monte Carlo algorithm is said to have *error probability* $\alpha \in \mathbb{R}^+$ with $0 \leq \alpha < 1$, provided that for any occurrence in which the answer is "yes" the algorithm will give the incorrect answer "no" with probability at most α, where the probability is computed over all possible random choices made by the algorithm for a given input. Therefore, the Miller-Selfridge-Rabin algorithm is

a yes-biased Monte Carlo algorithm for the decision problem "Is n composite?" with error probability $\alpha = (1/4)^r$.

There are many related algorithms that we have not discussed here, such as the Solovay-Strassen test, because the Miller-Selfridge-Rabin test is computationally less expensive, easier to implement, and at least as correct. For information on such tests, the reader may consult [63, pp. 84–86], for instance.

Exercises

5.14. Let $n = 1 + p^t m$, p an odd prime, $t, m \in \mathbb{N}$, and suppose that there exists an $a \in \mathbb{Z}$ such that

$$a^{n-1} \equiv 1 \,(\text{mod } n).$$

Prove that if $q \mid n$ is prime, then either

$$a^{(n-1)/p} \equiv 1 \,(\text{mod } q)$$

or

$$q \equiv 1 \,(\text{mod } p^t).$$

5.15. Use the MSR algorithm to test $n = 7331$ for primality with $a = 2$.

5.16. With input parameter $a = 3$, test $n = 9377$ for primality using the MSR algorithm.

5.17. Use the MSR test for $n = 2152302898747$ with $a = 5$.

(*Hint: Use the repeated squaring method, displayed on page 31, since even with mathematical software packages a single exponentiation will likely cause overflow. A similar comment holds for Exercise 5.18.*)

5.18. Employ the MSR test for $n = 3474749660383$ with $a = 7$.

5.19. Prove that if $n \in \mathbb{N}$ such that $2^n \equiv 1 \,(\text{mod } n)$, then $n = 1$.

5.20. Given $D, n, k \in \mathbb{N}$ such that $\gcd(n, D) = 1$, with n odd having canonical prime factorization

$$n = \prod_{j=1}^{k} p_j^{a_j},$$

define

$$\psi_D(n) = \frac{1}{2^{k-1}} \prod_{j=1}^{k} p_j^{a_j - 1} \left(p_j - \left(\frac{D}{p_j} \right) \right),$$

where the symbol on the right is the Legendre Symbol. Prove that n is prime if and only if

$$\psi_D(n) = n - (D/n).$$

5.21. With reference to Exercise 5.20, prove that if n is odd and

$$(n - (D/n)) \mid \psi_D(n),$$

then n is prime.

5.3 Recognizing Primes

In Exercise 4.13 on page 171, we introduced *safe primes*, namely those primes p, those for which $(p-1)/2$ is also prime. Safe primes are also important in selecting an RSA modulus $n = pq$ because if p and q are safe primes, then RSA is not vulnerable to $p-1$ and $p+1$ factoring methods, which we will discuss in Chapter 6. In general, having safe primes in the modulus makes it more difficult to factor. However, finding such primes is also more difficult. We present the following algorithm for so doing, which is taken from [63]. First we need the following concept. If $B \in \mathbb{N}$, then a number $n \in \mathbb{N}$ is said to be a *B-smooth number* if all primes dividing n are no larger than B, and B is called a *smoothness bound*.

◆ Algorithm for Generating (Probable) Safe Primes

Let b be the input bitlength of the required prime. Execute the following.

(1) Select a $(b-1)$-bit odd random $n \in \mathbb{N}$ and a smoothness bound B (determined experimentally).

(2) Trial divide n by primes $p \leq B$. If n is divisible by any such p, go to step (1). Otherwise, go to step (3).

(3) Use the Miller-Selfridge-Rabin test on page 198 to test n for primality. If it declares that "n is probably prime," then go to step (4). Otherwise, go to step (1).

(4) Compute $2n+1 = q$ and use the Miller-Selfridge-Rabin test on q. If it declares q to be a probable prime, terminate the algorithm with q as a "probable safe prime". Otherwise go to step (1).

There are primes that have even more constraints to ensure security of the RSA modulus. They are given as follows, which is taken from [64].

Definition 5.2 **Strong Primes**

A prime p is called a strong prime *if each of the following hold.*

(1) $p-1$ *has a large prime factor* q.

(2) $p+1$ *has a large prime factor* r.

(3) $q-1$ *has a large prime factor* s.

The following algorithm was introduced in [39].

◆ Gordon's Algorithm for Generating (Probable) Strong Primes

(1) Generate two large (probable) primes $r \neq s$ of roughly equal bitlength using the MRS test described on pages 198–199.

(2) Select the first prime in the sequence $\{2js+1\}_{j\in\mathbb{N}}$, and let

$$q = 2js + 1$$

be that prime.

(3) Compute

$$p_0 \equiv r^{q-1} - q^{r-1} \pmod{rq}.$$

(4) Find the first prime in the sequence $\{p_0 + 2iqr\}_{i\in\mathbb{N}}$, and let

$$p = p_0 + 2iqr$$

be that prime, which is a strong prime.

Although it is possible to generate primes that are both safe and strong, the algorithms are not as efficient as Gordon's algorithm. Furthermore, choosing random primes large enough will generally thwart direct factoring attacks. The following, also taken from [63], provides a mechanism for generating large random primes.

◆ Large (Probable) Prime Generation

We let b be the input bitlength of the desired prime and let B be the input smoothness bound (empirically determined). Execute the following steps.

(1) Randomly generate an odd b-bit integer n.

(2) Use trial division to test for divisibility of n by all odd primes no bigger than B. If n is so divisible, go to step (1). Otherwise go to step (3).

(3) Use the MSR test n for primality. If it is declared to be a probable prime, then output n as such. Otherwise, go to step (1).

◆ Large (Provable) Prime Generation

Begin with a prime p_1, and execute the following steps until you have a prime of the desired size. Initialize the variable counter $j = 1$.

(1) Randomly generate a small odd integer m and form $n = 2mp_j + 1$.

(2) If $2^{n-1} \not\equiv 1 \pmod{n}$, then go to step (1). Otherwise, go to step (3).

(3) Using the primality test on page 192, with prime bases $2 \le a \le 23$, if for any such a,

$$a^{(n-1)/p} \not\equiv 1 \pmod{n}$$

for any prime p dividing $n-1$, then n is prime. If n is large enough, terminate the algorithm with output n as the provable prime. Otherwise, set $n = p_{j+1}$, $j = j+1$, and go to step (1). If the test fails go to step (1).

Note that since we have a known factorization of $n-1$ in the above algorithm, and a small value of m to check, then the test is simple and efficient.

Exercises

5.22. Prove that if n is a Carmichael number, introduced in Exercise 1.103 on page 43, then n is squarefree.

5.23. Given $n \in \mathbb{N}$, prove that the following are equivalent.

(a) $a^n \equiv a \pmod{n}$ for all $a \in \mathbb{Z}$.
(b) $a^{n-1} \equiv 1 \pmod{n}$ for all $a \in \mathbb{Z}$ such that $\gcd(a, n) = 1$.
(c) n is squarefree and $(p-1) \mid (n-1)$ for all primes p dividing n.

(*These equivalent conditions are known as* Korselt's criterion *discovered in 1899.*)

5.24. Prove that the prime p output by Gordon's algorithm is indeed a strong prime.

5.25. Suppose that we obtain two strong primes p and q using Gordon's algorithm on page 204. Also, we set

$$n = pq, \qquad n' = (p-1)(q-1)/4,$$

and choose a random integer e such that

$$\gcd(e, n') = 1, \quad (p-1) \nmid (e-1), \quad (q-1) \nmid (e-1).$$

Show how to set up an RSA cryptosystem with public enciphering key e and RSA modulus n.

5.26. An odd integer $n > 2$ is called an *Euler probable prime to base* a if

$$a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n} \text{ and } \gcd(a, n) = 1,$$

where the symbol on the right of the congruence is the Jacobi symbol (see page 62). A composite probable prime to base a is called an *Euler pseuoprime to base* a.

Solve each of the following

(a) Prove that if n is an Euler probable prime, then n is a probable prime to base a.
(b) Prove that if $n \equiv 3 \pmod{4}$ is an Euler pseudoprime to base a, then n is a strong pseudoprime to base a.
(c) Prove that if n is an Euler pseudoprime to base a and $\left(\frac{a}{n}\right) = -1$, then n is a strong pseudoprime to base a.

5.27. Prove that if n is a strong pseudoprime to base a, then n is a strong pseudoprime to base a^{2j+1} for any $j \in \mathbb{N}$.

Chapter 6

Factoring

6.1 Classical Factorization Methods

Given the importance of factoring in the security of RSA and other cryptosystems, it is worth our having a closer look at the issue to which we devote this chapter. We first look at the following basic building block.

◆ The Integer Factoring Problem — (IFP)

Given $n \in \mathbb{N}$, find primes p_j for $j = 1, 2, \ldots, r \in \mathbb{N}$ with $p_1 < p_2 < \cdots < p_n$ and $e_j \in \mathbb{N}$ for $j = 1, 2, \ldots, r$, such that

$$n = \prod_{j=1}^{r} p_j^{e_j}.$$

A simpler problem than the IFP is the notion of *splitting* of $n \in \mathbb{N}$, which means the finding of factors $r, s \in \mathbb{N}$ such that $1 < r \leq s$ such that $n = rs$. Of course, with an RSA modulus, splitting and the IFP are the same thing. Yet, in order to solve the IFP for any integer, one merely splits n, then splits n/r and s if they are both composite, and so on until we have a complete factorization.

Now we discuss some older methods that still have relevance for the methods of today.

Trial Division The oldest method of splitting n is *trial division*, by which we mean dividing n by all primes up to $\sqrt{n}$. For $n < 10^8$, or within that neighbourhood, this is not an unreasonable method in our computer-savvy world. However, for larger integers, we need more elaborate methods.

Fermat Factoring On page 201, we discussed Fermat's method for factoring, which we called his *difference of squares method*. Although the order of magnitude (see page 67) of Fermat factoring can be shown to be $O(n^{1/2})$, Lehman has shown how to reduce the complexity to $O(n^{1/3})$ when combined

with trial division. This is all contained in [49], complete with a computer program. There is also a method, from D.H. Lehmer, for speeding up the Fermat method when all factors are of the form $2k\ell + 1$ (see [10]).

Euler's Factoring Method This method applies only to integers of the form

$$n = x^2 + ay^2 = z^2 + aw^2,$$

where $x \neq z$ and $y \neq w$. In other words, n can be written in two distinct ways in this special form for a given nonzero value of $a \in \mathbb{Z}$. Then

$$(xw)^2 \equiv (n - ay^2)w^2 \equiv -ay^2w^2 \equiv (z^2 - n)y^2 \equiv (zy)^2 \pmod{n},$$

from which we may have a factor of n, namely, provided that $xw \not\equiv \pm zy \pmod{n}$. In this case, the (nontrivial) factors of n are given by $\gcd(xw \pm yz, n)$.

The Euler method essentially is predicated on the congruence (5.4) on page 201, but unlike the Fermat method, not all integers have even one representation in the form $n = x^2 + ay^2$.

Legendre's Factoring Method This method is a precursor to what we know today as *continued fraction methods for factorization* (see pages 321–324 and Section 6.2). Legendre reasoned in the following fashion. Instead of looking at congruences of the form (5.4), he looked at those of the form

$$x^2 \equiv \pm py^2 \pmod{n} \text{ for primes } p, \tag{6.1}$$

since a solution to (6.1) implies that $\pm p$ is a quadratic residue of all prime factors of n. For instance, if the residue is 2, then all prime factors of n are congruent to $\pm 1 \pmod 8$ (see part (5) of Theorem 1.24 on page 64). Therefore, he would have halved the search for factors of n. Legendre applied this method for various values of p, thereby essentially constructing a quadratic sieve by getting many residues modulo n. (A *sieve* may be regarded as any process whereby we find numbers via searching up to a prescribed bound and eliminate candidates as we proceed until only the desired solution set remains. A [general] *quadratic* sieve is one in which about half of the possible numbers being sieved are removed from consideration, a technique used for hundreds of years as a scheme for eliminating impossible cases from consideration.) This allowed him to eliminate potential prime divisors that sit in various linear sequences, as with the residue 2 example above. He realized that if he could achieve enough of these, he could eliminate primes up to $\sqrt{n}$, thereby effectively developing a test for primality.

The linchpin of Legendre's method is the continued fraction expansion of $\sqrt{n}$ since he was simply finding *small* residues modulo n. Legendre was essentially building a sieve on the prime factors of n, which did not let him predict, for a given prime p, a different residue to yield a square. This meant that if he found a solution to

$$x^2 \equiv py^2 \pmod{n},$$

he could not predict a solution,

$$w^2 \equiv pz^2 \pmod{n},$$

distinct from the former. If he had been able to do this, he would have been able to combine them as

$$(xw)^2 \equiv (pzy)^2 \,(\mathrm{mod}\ n)$$

and have a factor of n *provided that* $xw \not\equiv \pm pzy \,(\mathrm{mod}\ n)$ since we are back to congruence (5.4).

Gauss invented a method that differed from Legendre's scheme only in the approach to finding small quadratic residues of n; but his approach makes it much more complicated (see [35, Articles 333 and 334, pp. 403–406]).

In the 1920's, one individual expanded the idea, described above, of attempting to match the primes to create a square. We now look at his important influence.

Kraitchik's Factoring Method Maurice Kraitchik (see Biography 6.1 on the following page) determined that it would suffice to find a *multiple* of n as a difference of squares in attempting to factor it. For this purpose, he chose a polynomial of the form, $kn = ax^2 \pm by^2$, for some integer k, which allowed him to gain control over finding two distinct residues at a given prime to form a square, which Legendre could not do. In other words, Kraitchik used quadratic polynomials to get the residues, then multiplied them to get squares (not a square times a small number). Kraitchik developed this method over a period of more than three decades, a method later exploited by D.H. Lehmer and R.E. Powers (see [52]). They employed Kraitchik's technique but obtained their residues as Legendre had done. Later this was exploited in the development of an algorithm that systematically extracted the best of the above ideas, which we will present in the next section.

◆ The MSR Test and Factoring

When we discussed the MSR probabilistic primality test on pages 198–201, we saw that we got factors of n whenever n is a pseudoprime to base a but not a strong pseudoprime to base a, namely when $a^{n-1} \equiv 1 \,(\mathrm{mod}\ n)$. However, it is rare that the latter occurs. Suppose, on the other hand, that for a given modulus $n \in \mathbb{N}$ there exists an exponent $e \in \mathbb{N}$ such that $x^e \equiv 1 \,(\mathrm{mod}\ n)$ for all $x \in \mathbb{N}$ with $\gcd(x, n) = 1$, where e called a *universal exponent*. Then it may be possible to factor n as follows.

◆ Universal Exponent Factorization Method

Let e be a universal exponent for $n \in \mathbb{N}$ and set $e = 2^b m$ where $b \geq 0$ and m is odd. Execute the following steps.

(1) Choose a random base a such that $1 < a < n - 1$. If $\gcd(a, n) > 1$, then we have a factor of n, and we may terminate the algorithm. Otherwise go to step (2).

(2) Compute $x_0 \equiv a^m \,(\mathrm{mod}\ n)$. If $x_0 \equiv 1 \,(\mathrm{mod}\ n)$, then go to step (1). Otherwise, compute $x_j \equiv x_{j-1}^2 \,(\mathrm{mod}\ n)$ for all $j = 1, \ldots, b$. If

$$x_j \equiv -1 \,(\mathrm{mod}\ n),$$

then go to step (1). If $x_j \equiv 1 \pmod{n}$, but $x_{j-1} \not\equiv \pm 1 \pmod{n}$, then $\gcd(x_{j-1} - 1, n)$ is a nontrivial factor of n so we may terminate the algorithm.

If one compares the above with the Miller-Selfridge-Rabin probabilistic primality test, striking similarities will be seen. However, the primality test is not guaranteed to have a value such that $x_j \equiv 1 \pmod{n}$ as we have in the universal exponent method (due to the existence of the exponent e).

There is also some relevance of the above to the RSA cipher. When $n = pq$ is an RSA modulus, a universal exponent is sometimes taken to be $\text{lcm}(p-1, q-1)$ instead of $\phi(n) = (p-1)(q-1)$. Yet these two values will be roughly the same since $\gcd(p-1, q-1)$ has an expectation of being small when p and q are chosen arbitrarily. Furthermore, given the RSA encryption exponent e and decryption exponent d, since $de-1$ is a multiple of $\phi(n)$, then $de-1$ is a universal exponent, and the above method can be used to factor n. This is the major value of our universal exponent test since actually finding e is difficult in practice.

Example 6.1 Given $n = 15841$, since $e = 2^5 \cdot 405$, we choose $a = 2$ for which $2^{405} \equiv 1 \pmod{n}$. Thus, we go to step (1) and choose another base, $a = 3$. We compute $x_0 \equiv 3^{405} \equiv 2820 \pmod{n}$, then $x_1 \equiv 2820^2 \equiv 218 \pmod{n}$, and $x_2 \equiv 218^2 \equiv 1 \pmod{n}$. Since we know that $x_1 \not\equiv \pm 1 \pmod{n}$, then $\gcd(217, 15841) = 217$ is a factor of n. Indeed $n = 217 \cdot 73$.

Biography 6.1 Maurice Borisovich Kraitchik (1882–1957) *obtained his Ph.D. from the University of Brussels in 1923. He worked as an engineer in Brussels and later as a Director at the Mathematical Sciences section of the Mathematical Institute for Advanced Studies there. From 1941–1946, he was Associate Professor at the New School for Social Research in New York. In 1946, he returned to Belgium, where he died on August 19, 1957. His work over thirty-five years on factoring methods stands tall today because he devised and used a variety of practical techniques that are found today in computer methods such as the Quadratic Sieve* (*see Section 6.4 on page 217*). *He is also the author of the popular book* Mathematical Recreations [47] .

Exercises

6.1. Use Euler's method presented on page 208 to factor $10817 = 25^2 + 13 \cdot 28^2$.

6.2. Use Euler's method to factor $14417 = 65^2 + 13 \cdot 28^2$.

6.3. Use Legendre's method, presented on page 208, to factor $-95^2 + 17 * 24^2 = 767$.

6.4. Use Legendre's method to factor $-65^2 + 17 \cdot 24^2 = 5567$

6.5. Use the universal exponent method on $n = 87611$ with $e = 43212$.

6.6. Use the universal exponent method on $n = 104957$ employing $e = 51918$.

6.2 The Continued Fraction Algorithm

First we need to define a *factor base*, which is a set of "small" primes that remain the primes under consideration for the algorithm at hand.

◆ The Continued Fraction Algorithm

Suppose that we wish to factor $n \in \mathbb{N}$ and a smoothness bound B has been selected — see page 204. Then execute the following steps:

(1) Choose a factor base of primes $\mathcal{F} = \{p_1, p_2, \ldots, p_k\}$ for some $k \in \mathbb{N}$ determined by B and a large upper index value J.

Note that from knowledge about the distribution of smooth integers close to $\sqrt{n}$, the optimal k is known to be one that is chosen to be

$$k \approx \sqrt{\exp(\sqrt{\log(n)\log\log(n)})}. \tag{6.2}$$

(2) Set $Q_0 = 1$, $P_0 = 0$, $A_{-2} = 0$, $A_{-1} = 1$, $A_0 = \lfloor\sqrt{n}\rfloor = q_0 = P_1$. For each natural number $j \leq J$, recursively compute Q_j using the following formulas:

$$Q_j = \frac{n - P_j^2}{Q_{j-1}},$$

$$q_j = \left\lfloor \frac{P_j + \lfloor\sqrt{n}\rfloor}{Q_j} \right\rfloor,$$

$$A_j = q_j A_{j-1} + A_{j-2},$$

$$P_{j+1} = q_j Q_j - P_j,$$

and trial divide Q_j by the primes in $\mathcal{F}$ to determine if Q_j is p_k-smooth. If it is, use its factorization

$$Q_j = \prod_{i=1}^{k} p_i^{a_{i,j}}$$

to form the binary $k+1$-tuple,

$$\mathfrak{v}_j = (v_{0,j}, v_{1,j}, v_{2,j}, \ldots, v_{k,j}),$$

where $v_{0,j}$ is, respectively, 0 or 1 according as j is even or odd, and for $1 \leq i \leq k$, $v_{i,j}$ is, respectively, 0 or 1 according to whether $a_{i,j}$ is even or odd. If Q_j is not p_k-smooth, discard it and return to calculate Q_{j+1}.

(3) For each set $\mathcal{S}$ of indices j for the vectors $\mathfrak{v}_j$ constructed in (2), for which it is discovered that

$$\sum_{j \in \mathcal{S}} v_{i,j} \equiv 0 \,(\mathrm{mod}\ 2), \quad 0 \leq i \leq k,$$

we have $x^2 \equiv y^2 \pmod{n}$, where

$$x = \left[\prod_{j \in \mathcal{S}} (-1)^j Q_j\right]^{1/2} \quad \text{and } y \equiv \prod_{j \in \mathcal{S}} A_{j-1} \pmod{n}.$$

If $x \not\equiv \pm y \pmod{n}$, then $\gcd(x \pm y, n)$ gives a nontrivial factor of n.

By Corollary A.4 on page 324,

$$A_{j-1}^2 - nB_{j-1}^2 = (-1)^j Q_j,$$

which is the essence of the algorithm. Thus, we have that

$$nB_{j-1}^2 \equiv A_{j-1}^2 \pmod{p},$$

for any prime $p \mid Q_j$, so n is a quadratic residue modulo p. Hence, we only put primes p in the factor base for which n is a quadratic residue modulo p. The following gives a small illustration of the continued fraction algorithm, called CFRAC by some users.

Example 6.2 Let $n = 5969$. A convenient choice for a factor base will be

$$\mathcal{F} = \{2, 5, 23, 43, , 71, 103\}.$$

Since $\lfloor \sqrt{n} \rfloor = 77$, then we compute the following table (where $J = 8$).

j	P_j	q_j	A_{j-1}	$(-1)^j Q_j$	$\mathfrak{v}_j$
0	0	77	1	1	$(0,0,0,0,0,0,0)$
1	77	3	77	-40	$(1,1,1,0,0,0,0)$
2	43	1	232	103	$(0,0,0,0,0,0,1)$
3	60	5	309	-23	$(1,0,0,1,0,0,0)$
4	55	1	1777	128	$(0,1,0,0,0,0,0)$
5	73	30	2086	-5	$(1,0,1,0,0,0,0)$
6	77	19	64357	8	$(0,1,0,0,0,0,0)$
7	75	3	1224869	-43	$(1,0,0,0,1,0,0)$
8	54	1	3738964	71	$(0,0,0,0,0,1,0)$

We have a set $\mathcal{S}$ such that

$$\sum_{j \in \mathcal{S}} v_{i,j} \equiv 0 \pmod{2} \text{ for each } i = 0, 1, \ldots, 6.$$

This set consists of the vectors for $j = 4, 6$, namely

$$\mathcal{S} = \{\mathfrak{v}_4, \mathfrak{v}_6\} = \{(0,1,0,0,0,0,0), (0,1,0,0,0,0,0)\}$$

for which we have $Q_4 = 2^7$, $Q_6 = 2^3$, $A_4 = 1777$, and $A_6 = 64357$. We compute

$$\prod_{j \in \mathcal{S}} A_{j-1} \equiv 2318 \pmod{5969},$$

and since

$$y^2 = \prod_{j \in \mathcal{S}} A_{j-1}^2 \equiv 2318^2 \equiv x^2 = \prod_{j \in \mathcal{S}} Q_j = (32)^2 \pmod{n},$$

then we check $\gcd(x \pm y, n)$. We compute that both

$$\gcd(x - y, n) = \gcd(2318 - 32, 5969) = 127$$

and

$$\gcd(x + y, n) = \gcd(2318 + 32, 5969) = 47.$$

Thus, we have factored $n = 47 \cdot 127$.

The CFRAC algorithm was developed by Brillhart and Morrison in the early 1970's (see [66]). It is widely acclaimed to be the very first efficient *general* factorization algorithm put into use. It is subexponential time (see page 69), which essentially means that if the running time to factor n is n^a, then a slowly decreases as $n \mapsto \infty$.

Exercises

Factor each of the integers in Exercises 6.7–6.16 using the continued fraction algorithm.

6.7. $n = 3090847$.

6.8. $n = 4446833$.

6.9. $n = 3774403$.

6.10. $n = 13556087$.

6.11. $n = 35923031$.

6.12. $n = 67168841$.

6.13. $n = 63382447$.

6.14. $n = 71685749$.

6.15. $n = 82979779$.

6.16. $n = 81491677$.

6.3 Pollard's Algorithms

In 1974, Pollard published a factorization scheme (see [70]) that utilizes Euler's generalization of Fermat's Little Theorem (see Theorem 1.18 on page 40). He reasoned that if $(p-1) \mid n$ where p is prime, then $p \mid (t^n - 1)$ provided that $p \nmid t$, which follows from Euler's theorem, so p may be found by employing Euclid's algorithm (see Theorem 1.2 on page 3).

◆ Pollard's $p-1$ Algorithm

Suppose that we wish to factor $n \in \mathbb{N}$, and that a smoothness bound B has been selected (see page 204). Then we execute the following.

(1) Choose a base $a \in \mathbb{N}$ where $2 \leq a < n$ and compute $g = \gcd(a, n)$. If $g > 1$, then we have a factor of n. Otherwise, go to step (2).

(2) For all primes $p \leq B$, compute $m = \left\lfloor \frac{\ln(n)}{\ln(p)} \right\rfloor$ and replace a by $a^{p^m} \pmod{n}$ using the repeated squaring method given on page 31. (Note that this iterative procedure ultimately gives $a^{\prod_{p \leq B} p^m}$ modulo n for the base a chosen in (1).)

(3) Compute $g = \gcd(a-1, n)$. If $g > 1$, then we have a factor of n, and the algorithm is successful. Otherwise, the algorithm fails.

▼ Analysis

Let $\ell = \prod_{j=1}^{t} p_j^{a_j}$, where $p_j^{a_j}$ runs over all prime powers such that $p_j \leq B$. Since $p_j^{a_j} \leq n$, then $a_j \ln(p_j) \leq \ln(n)$, so $a_j \leq \left\lfloor \frac{\ln(n)}{\ln(p_j)} \right\rfloor$. Hence,

$$\ell \leq \prod_{j=1}^{t} p_j^{\lfloor \ln(n)/\ln(p_j) \rfloor}.$$

Now, if $p \mid n$ is a prime such that $p-1$ is B-smooth, then $(p-1) \mid \ell$. Therefore, for any $a \in \mathbb{N}$ with $p \nmid a$, $a^\ell \equiv 1 \pmod{p}$, by Fermat's Little Theorem. Thus, if $g = \gcd(a^\ell - 1, n)$, then $p \mid g$. If $g = n$, then the algorithm fails. Otherwise, it succeeds.

Example 6.3 Let $n = 44717$, and choose a smoothness bound $B = 13$, then select $a = 2$. We know that a is relatively prime to n so we proceed to step (2). The table shows the outcome of the calculations for step (2).

p	2	3	5	7	11	13
m	15	9	6	5	4	4
a	2363	11932	38704	35988	10962	4172

Then we go to step (3) and check $\gcd(a-1, n) = \gcd(4171, 44717) = 97$. Thus, we have factored $n = 97 \cdot 461$. Observe that $p = 97$ is B-smooth since $p - 1 = 2^5 \cdot 3$, but $q = 461$ is not since $q - 1 = 2^2 \cdot 5 \cdot 23$.

▼ Concluding Comments

The running time for Pollard's $p-1$ algorithm is $O(B\ln(n)/\ln(B))$ modular multiplications, assuming that $n \in \mathbb{N}$ and there exists a prime $p \mid n$ such that $p-1$ is B-smooth. This is of course the drawback to this algorithm, namely, that it requires n to have a prime factor p such that $p-1$ has only "small" prime factors. A generalization of the $p-1$ method was given by Lenstra using elliptic curves, which we will study later in the text. In the Elliptic curve algorithm, which we study in Section 6.5, we will see that success in factoring depends upon an integer "close" to p having only small prime factors, which is less demanding than the $p-1$ algorithm and therefore more likely to occur.

Pollard also developed another method for factoring in 1975, called the *Monte Carlo factoring method*, also known as the *Pollard rho method*.

◆ Pollard's Rho Method

Given $n \in \mathbb{N}$ composite, and p an (as yet unknown) prime divisor of it, perform the following steps.

(1) Choose an integral polynomial f with $\deg(f) \geq 2$—usually $f(x) = x^2 + 1$ is chosen for simplicity.

(2) Choose a randomly generated integer $x = x_0$, the *seed*, and compute $x_1 = f(x_0)$, $x_2 = f(x_1) \ldots, x_{j+1} = f(x_j)$ for $j = 0, 1, \ldots B$, where the bound B is determined by step (3).

(3) Sieve through all differences $x_i - x_j$ modulo n until it is determined that

$$x_B \not\equiv x_j \pmod{n}$$

but $x_B \equiv x_j \pmod{p}$ for some natural number $B > j \geq 1$. Then

$$\gcd(x_B - x_j, n)$$

is a nontrivial divisor of n.

Example 6.4 If $n = 37351$, and $x_0 = 2$ is the seed with $f(x) = x^2 + 1$, then $x_1 = f(x_0) = 5$, $x_2 = f(x_1) = 26$, $x_3 = f(x_2) = 677$, $x_4 = f(x_3) = 3146$, $x_5 = f(x_4) = \overline{36653}$, and $x_6 = f(x_5) = \overline{1642}$, where the bar notation denotes the fact that we have reduced the values to the least residue system modulo n. We find that all $\gcd(x_i - x_j, n) = 1$ for $i \neq j$ until $\gcd(x_6 - x_0, n) = \gcd(1640, 37351) = 41$. In fact, $37351 = 41 \cdot 911$.

Now we illustrate the reason behind the name *Pollard rho method*. We take $n = 29$ as the modulus and $x_0 = 2$ as the seed, then we proceed through the Pollard rho method to achieve the Diagram 6.1 on the next page.

Diagram 6.1 Pollard's Rho Method Illustrated

We take $n = 29$ as the modulus and $x_0 = 2$ as the seed, then we proceed through the Pollard rho method to achieve the following diagram.

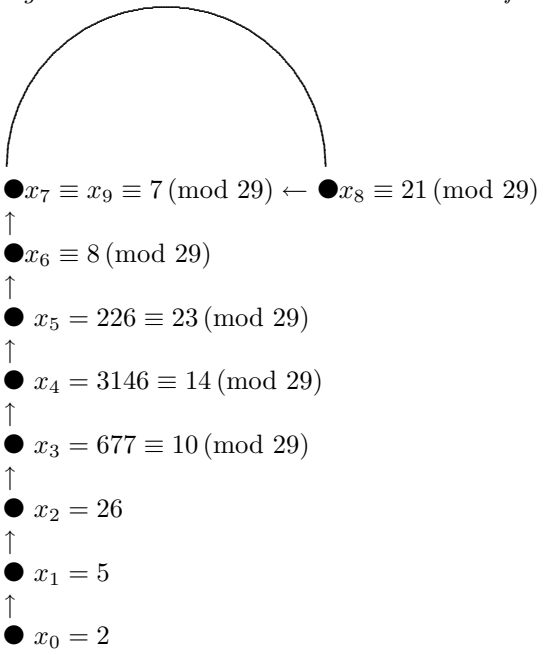


Diagram 6.1 shows us that when we reach x_9, then we are in the period that takes us back and forth between the residue system of 7 and that of 21 modulo 29. This is the significance of the left pointing arrow from the position of x_8 back to the position of x_7, which is the same as the reside system of x_9. This completes the circuit. The shape of the symbol is reminiscent of the Greek symbol ρ, *rho*, pronounced *row*.

Pollard's two methods above may be invoked when trial division fails to be useful. However, if the methods of Pollard fail to be useful, which they will for *large* prime factors, say, with the number of digits in the high teens, then we need more powerful machinery. In the next section we look at one of those.

Exercises

Factor each of the integers in Exercises 6.17–6.20 using both of Pollard's methods.

6.17. $n = 1324237$.

6.18. $n = 3162571$.

6.19. $n = 5951129$.

6.20. $n = 9480431$.

6.4 The Quadratic Sieve

On page 209, we discussed the early pioneering efforts invovled in Kraitchik's factoring method. In the early 1980's, Carl Pomerance was able to fine tune the parameters in Kraitchik's sieve method (see [71]).

◆ The Quadratic Sieve (QS) Algorithm

(1) Choose a *factor base* $\mathcal{F} = \{p_1, p_2, \ldots, p_k\}$, where the p_j are primes for $j = 1, 2, \ldots, k \in \mathbb{N}$.

(2) For each nonnegative integer j, let $t = \pm j$. Compute

$$y_t = (\lfloor \sqrt{n} \rfloor + t)^2 - n$$

until $k + 2$ such values are found that are p_k-smooth. For each such t,

$$y_t = \pm \prod_{i=1}^{k} p_i^{a_{i,t}}, \tag{6.3}$$

and we form the binary $k + 1$-tuple,

$$\mathfrak{v}_t = (v_{0,t}, v_{1,t}, v_{2,t}, \ldots, v_{k,t}),$$

where $v_{i,t}$ is the least nonnegative residue of $a_{i,t}$ modulo 2 for $1 \leq i \leq k$, $v_{0,t} = 0$ if $y_t > 0$, and $v_{0,t} = 1$ if $y_t < 0$.

(3) Obtain a subset $\mathcal{S}$ of the values of t found in step (2) such that for each $i = 0, 1, 2, \ldots, k$,

$$\sum_{t \in \mathcal{S}} v_{i,t} \equiv 0 \,(\text{mod } 2). \tag{6.4}$$

In this case,

$$x^2 = \prod_{t \in \mathcal{S}} x_t^2 \equiv \prod_{t \in \mathcal{S}} y_t = y^2 \,(\text{mod } n),$$

where $x_t = \lfloor \sqrt{n} \rfloor + t$, so $\gcd(x \pm y, n)$ provides a nontrivial factor of n if $x \not\equiv \pm y \,(\text{mod } n)$.

In step (2), we have that $y_t \equiv x_t^2 \,(\text{mod } n)$. Thus, if a prime $p \mid y_t = x_t^2 - n$, we have $x_t^2 \equiv n \,(\text{mod } p)$. Thus, we must exclude from the factor base any primes p for which there is no solution $x \in \mathbb{Z}$ to the congruence $x^2 \equiv n \,(\text{mod } p)$. In other words, we exclude from the factor base any primes p for which n is *not* a quadratic residue modulo p.

Example 6.5 Let $n = 60377$. From Equation (6.2) on page 211 , $k = 13$, so we choose the first thirteen primes for which n is a quadratic residue. They comprise our factor base $\mathcal{F} = \{2, 7, 11, 23, 29, 31, 37, 41, 53, 59, 61, 67, 71\}$. In the table on page 218, we see, by inspection, that a subset $\mathcal{S}$ of the values of t such

that $\sum_{t\in\mathcal{S}} v_{i,t} \equiv 0 \pmod 2$ for each $i = 0, 1, 2, \ldots, 13$ is $\mathcal{S} = \{-1, -3, -6, -22\}$. (Note that $\lfloor\sqrt{n}\rfloor = 245$ in this case.) Thus,

$$\prod_{t\in\mathcal{S}} x_t^2 = 244^2 \cdot 242^2 \cdot 239^2 \cdot 223^2 \equiv 50885^2 \equiv x^2 \pmod{60377},$$

and

$$\prod_{t\in\mathcal{S}} y_t = 2^6 \cdot 7^2 \cdot 11^4 \cdot 29^2 \cdot 37^2 \equiv 25408^2 \equiv y^2 \pmod{60377}.$$

By computing both of the values,

$$\gcd(x - y, n) = \gcd(50885 - 25408, 60377) = 349$$

and

$$\gcd(x + y, n) = \gcd((50885 + 25408, 60377) = 173,$$

we get that $n = 60377 = 173 \cdot 349$.

t	x_t	y_t	$\mathfrak{v}_t$
-1	244	-29^2	$(1,0,0,0,0,0,0,0,0,0,0,0,0,0)$
-3	242	$-7^2 \cdot 37$	$(1,0,0,0,0,0,0,1,0,0,0,0,0,0)$
3	248	$7^2 \cdot 23$	$(0,0,0,0,1,0,0,0,0,0,0,0,0,0)$
-4	241	$-2^3 \cdot 7 \cdot 41$	$(1,1,1,0,0,0,0,0,1,0,0,0,0,0)$
4	249	$2^3 \cdot 7 \cdot 29$	$(0,1,1,0,0,1,0,0,0,0,0,0,0,0)$
-6	239	$-2^3 \cdot 11 \cdot 37$	$(1,1,0,1,0,0,0,1,0,0,0,0,0,0)$
6	251	$2^6 \cdot 41$	$(0,0,0,0,0,0,0,0,1,0,0,0,0,0)$
7	252	$53 \cdot 59$	$(0,0,0,0,0,0,0,0,0,1,1,0,0,0)$
-10	235	$-2^5 \cdot 7 \cdot 23$	$(1,1,1,0,1,0,0,0,0,0,0,0,0,0)$
11	256	$7 \cdot 11 \cdot 67$	$(0,0,1,1,0,0,0,0,0,0,0,0,1,0)$
-16	229	$-2^8 \cdot 31$	$(1,0,0,0,0,0,1,0,0,0,0,0,0,0)$
16	261	$2^6 \cdot 11^2$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0)$
-20	225	$-2^3 \cdot 23 \cdot 53$	$(1,1,0,0,1,0,0,0,0,1,0,0,0,0)$
-22	223	$-2^3 \cdot 11^3$	$(1,1,0,1,0,0,0,0,0,0,0,0,0,0)$
22	267	$2^5 \cdot 11 \cdot 31$	$(0,1,0,1,0,0,1,0,0,0,0,0,0,0)$

▼ Analysis

Some elementary linear algebra underlies the solution to a factorization problem using the QS as depicted in Example 6.5. By ensuring that there are $k+2$ vectors $\mathfrak{v}_t$ in a $k+1$-dimensional vector space $\mathbb{F}_2^{k+1}$, we guarantee that there is a linear dependence relation among the $\mathfrak{v}_t$. In other words, we ensure the existence of the set $\mathcal{S}$ in step (3) of the algorithm such that congruence (6.4) holds. There is no guarantee that $x \not\equiv \pm y \pmod n$, but there are usually several dependency relations among the $\mathfrak{v}_t$, so there is a high probability that at least one of them will yield an (x, y) pair such that $x \not\equiv \pm y \pmod n$. The problem, of course, is that for "large" smoothness bounds B, we need a lot of congruences before we may be able to get these dependency relations.

▼ Concluding Comments

The first successful implementation of the QS in which a serious number was factored occurred in 1983 when J. Gerver [37] factored a 47-digit number. Then, in 1984, the authors of [21] factored a 71-digit number.

The QS has been employed using an approach called *factoring by electronic mail*. This is a term used by Lenstra and Manasse in [53] to mean the distribution of the Quadratic Sieve operations to hundreds of physically separated computers all over the world, and in 1988 they used this approach to factor a 106-digit number. Indeed, it is this *parallel computing* that picks up the time.

In 1994, the authors of [3] factored the RSA-129 number (see page 174) by using the electronic mail factoring technique with over 1600 computers and more than 600 researchers around the globe. The unit of time measurement for factoring is called a *mips year*, which is defined as being tantamount to the computational power of a computer rated at one *million instructions per second* (mips) and used for one year, which is equivalent to approximately $3 \cdot 10^{13}$ instructions. For instance, factoring the RSA-129 challenge number required 5000 mips years, and in 1989 the aforementioned factorization of the 106-digit number needed 140 mips years.

This chapter has presented some algorithms that perform more efficiently on certain numbers of a special form such as the $p-1$ Method, as well as more general purpose algorithms such as the QS technique. An overall strategy should go down many avenues. First, one should try to get small prime factors, which may be accomplished using trial division up to some reasonable bound, then use Fermat's Difference of Squares Method, after which one could employ other algorithms for small prime factors such as Pollard's $p-1$ Method or his rho method. When all else fails to get a complete factorization, the big guns may be brought to bear such as the Quadratic Sieve. The Elliptic Curve Method, which we will study in Section 6.5, can also be used in advance of the latter two sieves for finding small prime factors.

Although there is no known polynomial-time algorithm for integer factoring, what we have seen in the above presentations is that it is highly unlikely that there is one. Complexity theory in the modern day does not help us since it gives mostly upper bounds and we need lower bounds to determine the amount of time a cryptanalyst would need to break a cryptosystem.

Exercises

Factor each of the integers in Exercises 6.21–6.24 using the QS method.

6.21. $n = 3191491$.

6.22. $n = 12358397$.

6.23. $n = 42723991$.

6.24. $n = 74299271$.

6.5 The Elliptic Curve Method (ECM)

Some basic knowledge of elliptic curves is required before we present the factoring method. We begin with the basic definition.

Definition 6.1 Elliptic Curves
Let F be a field with characteristics not equal to 2 or 3. If $a, b \in F$ are given such that $4a^3 + 27b^2 \neq 0$ in F, then an elliptic curve E defined over F *is given by an equation $y^2 = x^3 + ax + b \in F[x]$. The set of all solutions $(x, y) \in F$ to the equation:*

$$y^2 = x^3 + ax + b, \tag{6.5}$$

together with a point $\mathfrak{o}$, called the point at infinity*, is denoted by $E(F)$, called the set of F-*rational points on E. *The value $\Delta(E) = -16(4a^3 + 27b^2)$ is called the* discriminant of the elliptic curve E.

▼ Elliptic Curve Facts

We assume that $E(\mathbb{Q})$ is an elliptic curve over $\mathbb{Q}$ given by $y^2 = x^3 + ax + b$ where $a, b \in \mathbb{Z}$, and $\mathfrak{o}$ denotes the point at infinity.

(1) (**Addition of points**): For any two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on E, with $P, Q \neq \mathfrak{o}$ and $P \neq -Q$, define

$$P + Q = (x_3, y_3) = (m^2 - x_1 - x_2, m(x_1 - x_3) - y_1), \tag{6.6}$$

where

$$m = \begin{cases} m_1/m_2 = (y_2 - y_1)/(x_2 - x_1) & \text{if } P \neq Q, \\ m_1/m_2 = (3x_1^2 + a)/(2y_1) & \text{if } P = Q, \end{cases} \tag{6.7}$$

and

$$\text{if } P = \mathfrak{o}, \text{ for instance, then } P + Q = Q \text{ for all points } Q \text{ on } E,$$

and

$$\text{if } P = -Q, \text{ then } P + Q = \mathfrak{o}.$$

(2) (**Reduction modulo** n): Let $n > 1$ be given and fixed with $\gcd(n, 6) = 1$ and $\gcd(4a^3+27b^2, n) = 1$. Then we refer to E reduced modulo n when the coefficients a, b are reduced modulo n, and each point P on E is reduced modulo n in the following fashion. If $P = (r_1/r_2, s_1/s_2)$ where

$$\gcd(r_1, r_2) = \gcd(s_1, s_2) = \gcd(r_2 s_2, n) = 1,$$

then

$$P = (t_1, t_2), \text{ where } t_1 \equiv r_1 r_2^{-1} \pmod{n} \text{ and } t_2 \equiv s_1 s_2^{-1} \pmod{n},$$

with r_2^{-1} and s_2^{-1} being the multiplicative inverses of r_2 and s_2 modulo n, respectively. We denote the reduced curve by $E(\mathbb{Z}/n\mathbb{Z})$, and if n is a prime, then this is a group.

(3) (**Modular group law**): Suppose that P_1, P_2 are points on $E(\mathbb{Q})$ where $P_1 + P_2 \neq \mathfrak{o}$ and the denominators of P_1, P_2 are prime to n. Then $P_1 + P_2$ has coordinates having denominators prime to n if and only if there does not exist a prime $p \mid n$ such that $P_1 + P_2 = \mathfrak{o} \pmod{p}$ on the elliptic curve $E(\mathbb{Z}/p\mathbb{Z})$.

In the following algorithm, $n \in \mathbb{N}$ is assumed to be composite, prime to 6, and not a perfect power, and $r \in \mathbb{N}$ is a parameter. The goal is to split n.

(1) (**Select and Elliptic Curve**): Choose a random pair (E, P) where $E = E(\mathbb{Z}/n\mathbb{Z})$ is an elliptic curve:

$$y^2 = x^3 + ax + b \text{ and } P \text{ is a point on } E.$$

Check that $g = \gcd(n, 4a^3 + 27b^2) = 1$. If not, then we have split n if $1 < g < n$, and we may terminate the algorithm. Otherwise, we select another (E, P) pair.

(2) (**Choosing bounds**): Select $M \in \mathbb{N}$ and bounds $A, B \in \mathbb{N}$ such that the canonical prime factorization for M is $M = \prod_{j=1}^{\ell} p_j^{a_{p_j}}$ for small primes $p_1 < p_2 < \cdots < p_\ell \leq B$ where $a_{p_j} = \lfloor \ln(A)/\ln(p_j) \rfloor$ is the largest exponent such that $p_j^{a_j} \leq A$. Set $j = k = 1$.

(3) (**Calculating multiple points**): Using (6.6) and (6.7) from page 220, compute $p_j P$.

(4) (**Computing the gcd**):

(a) If $p_j P \not\equiv \mathfrak{o} \pmod{n}$, then set $P = p_j P$, and reset k to $k + 1$.

(i) If $k \leq a_{p_j}$, then go to step (3).

(ii) If $k > a_{p_j}$, then reset j to $j + 1$, and reset k to $k = 1$. If $j \leq \ell$, then go to step (3). Otherwise go to step (5).

(b) If $p_j P \equiv \mathfrak{o} \pmod{n}$, then compute $\gcd(m_2, n)$ for m_2 in (6.7). If $n > g$, terminate the algorithm, since we have split n. If $g = n$, go to step (5).

(5) (**Selecting a new pair**): Set $r = r - 1$. If $r > 0$, go to step (1). Otherwise, terminate with "failure".

Example 6.6 *Let* $n = 923$ *and select* $(E, P) = (y^2 = x^3 + 2x + 9, (0, 3))$. *Then* $\gcd(4 \cdot 2^3 + 27 \cdot 9^2, 923) = 1$, *so we choose* $B = 4$, *based upon* (6.8), *and let* $A = 3, M = 6 = 2 \cdot 3 = p_1 \cdot p_2$. *Now, using* (6.6)–(6.7), *with* $p_1 = 2$, *we calculate*

$$p_1 P = 2(0, 3) \equiv (9^{-1}, -82 \cdot 27^{-1}) \equiv (718, 373) \not\equiv \mathfrak{o} \pmod{n}.$$

Thus we set $P = (718, 373)$ *and compute*

$$p_2 P = 3P \equiv 2P + P \equiv (505, 124) + (718, 373) \equiv \mathfrak{o} \pmod{n}.$$

Thus, we have that a denominator in (6.7) *is not prime to* n. *In fact, the calculation of* m *for* $4P + 2P$ *yields* $m = (124 - 373)/(505 - 718) = 83/71$, *and* $\gcd(923, 71) = 71$. *Indeed,* $n = 13 \cdot 71$, *and we have split* n.

What Example 6.6 illustrates is that the failure of the existence of a modular inverse for some m in the calculations may lead to a factor of n. Another way of saying this is that the group law for multiplication actually fails in $\mathbb{Z}/n\mathbb{Z}$ since n is not prime and this allows us to get the factor. Indeed, it is somewhat inaccurate in the ECM algorithm to say that $p_j P \equiv \mathfrak{o} \pmod{n}$, when in fact it is $p_j P \equiv \mathfrak{o} \pmod{p}$ where p is the factor for which we were searching. However, this is legitimate since we were, in a sense, assuming n to be prime and doing the calculations as if it were so, in the *hope* that the calculations would "break down" with an undefined denominator for some value of m in (6.7).

A significant advantage of the ECM is that its running time is highly reliant on the factor, $p \mid n$, found. Hence, one of the most useful means of employing the ECM is for finding "small" prime factors in a number n, which is too large to find *all* its factors. The reasons behind this are as follows. Assuming that p is the smallest prime dividing n, the expected running time of the ECM is known (under certain plausible assumptions) to be

$$O(\exp(\sqrt{(2 + o(1)) \ln p (\ln \ln p)}) \cdot \ln^2 n).$$

This may be used in practice to select a smoothness bound B in step (2) of the algorithm as

$$B = \exp(\sqrt{\ln p (\ln \ln p)/2}). \tag{6.8}$$

Since we do not know p in advance, we may nevertheless select (for p) the value $\lfloor \sqrt{n} \rfloor$. In this case, it is estimated that one out of every B iterations will be successful in splitting n.

The worst-case scenario for the ECM is when n is an RSA modulus, in which case we have that the expected running time is

$$O\left(\exp(\sqrt{(2 + o(1)) \ln n (\ln \ln n)}\right) = O\left(n^{\sqrt{(2+o(1))(\ln \ln n)/\ln n}}\right).$$

This being said, it is not surprising that ECM is most successful at splitting *non*-RSA moduli, *usually* finding prime factors of fewer than forty decimal digits in large composite numbers.

Exercises

Factor each of the integers in Exercises 6.25–6.29 using the ECM.

6.25. $n = 561707$.

6.26. $n = 3329663$.

6.27. $n = 20235773$.

6.28. $n = 54231487$.

6.29. $n = 72425447$.

Chapter 7

Electronic Mail and Internet Security

7.1 History of the Internet and the WWW

We begin by looking at the origins of the *Internet*, which is the system architecture underpinning the globally interconnected network of computers, the set of all the computer networks connected, via routers, all over the globe. The principal information retrieval scheme for the Internet is the *World-Wide Web* (WWW). The following is a discussion of where the Internet and WWW began, how they developed, and how the infrastructure evolved.

Computer networks go back as far as the late 1950's in the form of special-purpose systems. For instance, there was the inception of the airline registration system called SABRE. By the early 1960's, time-sharing systems, which allowed multiple employees to access the computer virtually simultaneously, were in use in many cutting-edge corporations. Such computers came to be known as *hosts* and with a vision toward a host-to-host network. By 1969, the first implementation of a host-to-host, general-purpose network, ARPANET, was put into service for the U.S. Department of Defence's *Advanced Research Projects Agency* (ARPA). ARPANET supported host-to-host, time-sharing connections in the United States, mainly at government-supported research sites such as universities. (However, the military people wanted separate communications, so they created *Milnet*, which remained connected and accessible by ARPANET users.) ARPANET also contained one of the first e-mail protocols called *simple mail transfer protocol* (SMTP). The other current e-mail standard is *Post Office Protocol* (POP). When e-mail arrives at an SMTP server, it is forwarded to a POP server where it is stored until accessed by the user who logs on, with username and password, to the POP server, which then retrieves the mail and sends it. POP3 is the latest version, which can be used with or without SMTP.

From SMTP evolved the *file transfer protocol* (FTP) needed for use with much larger data packages than those usually found in an e-mail transmission. In order to process these bigger blocks of data, ARPANET used a new segmenting mechanism called *packet switching*, which broke down large blocks of data into manageable packets for independent dispatch, and later reconstruction at the target site. This new notion for processing of packets via segmentation and reconstruction was one of the earliest means of communication *without* a *dedicated channel*, meaning a channel reserved exclusively for one type of communication. The term is often used to mean a *leased or private line*. On the other hand, a *dedicated server* is a particular computer in a network reserved specifically for the purpose of fulfilling the needs of the network. Today, however, most servers are not dedicated since the computer may be employed to be a server in addition to performing other duties. The antithesis of dedicated is *general purpose*.

Although packet networks were created in the private sector in the 1970's, such as Telnet in the United States, these were not host-to-host connections. Instead, they were *virtual* circuits over packet networks. In the 1970's and 1980's, the host-to-host networks remained in government control.

ARPA was replaced by DARPA, the *Defence Advanced Research Project Agency*, which may be seen as having played a seminal role in the establishing of a mini-version of the Internet via its researchers employing a network for their communications. Essentially, DARPA employed a combination of ground- and satellite-based packet networks, which allowed a combination of ground-based radio system transportable access to computing facilities, coupled with a satellite-based connection between the United States and Europe. However, there was no interconnection among the ground-based net, the satellite-based net, and other networks. In other words, the modern-day Internet had not yet come into existence.

By the mid-1970's, the notion of data packets evolved into a scheme called *Transmission Control Protocol* (TCP), allowing interconnected networks all over the world to transmit and receive data. TCP contained a world-wide addressing scheme, called the *Internet Protocol* (IP), permitting routers to deliver data packets to their target sites. By the mid-1980's, the TCP/IP scheme was effectively adopted worldwide.

The *National Science Foundation* (NSF) in the United States played a significant role in establishing TCP/IP as a universal standard. In the mid-1980's, they funded the first five supercomputing centres, and the development of NSFNET, a network to connect these centres. However, private enterprise was not allowed to use NSFNET for their transactions, so in the late 1980's, a commercial distribution of networks was developed in the private sector called the Commercial Internet Exchange (CIX). However, by 1993, federal legislation allowed NSF to open NSFNET to commerce. Consequently, in 1995, NSF dropped its support of NSFNET, since they saw the willingness of the private sector to support a communications network on their own, which marked the end of government control of the Internet, as with cryptography discussed earlier.

The IETF, the *Internet Engineering Task Force*, has developed and main-

tained standards. The IETF is indirectly overseen by the *Internet Society*, a nonprofit organization that acts as a conscience and guide for the Internet. The Internet Society supports the *Internet Architecture Board* (IAB), which oversees the technical development of the Internet. In particular, IAB supervises IETF. (See *http://ietf.org/*.) By the late 1990's, the number of *Internet Service Providers* (ISPs) had mushroomed, and we now have tens of millions of ISP subscribers, with no end in sight.

In 1988, the *Corporation for National Research Initiatives* provided the first commercial Internet connection linking e-mail, called *MCI* mail, after which other e-mail providers entered the fray, and Internet traffic has never been the same. In September 1993, the National Centre for Supercomputing Applications at the University of Illinois introduced *Mosaic*, which was the first of a new breed of computer programs called a *browser*, which made it easier to access, obtain, and display Internet files. Embedded in Mosaic was a collection of protocols, developed at *Centre Européen de Recherche Nucléaire* (CERN), for an Internet application called the *World-Wide Web*. From Mosaic Communications Corporation evolved Netscape Communications Corporation, established in April 1994, to develop Mosaic for commercial use. Mosaic was released officially in December 1994, after which it swiftly became the predominant browser. Later, Microsoft Corporation developed *Internet Explorer*, which was derived from the Mosaic idea. In fact, Mosaic was the first program to produce a multimedia graphical user interface (GUI). A GUI refers to the use of pictures, as well as text, to display the output of a program. This may be presented in the form of icons or buttons, for instance, which a user can control via a mouse-controlled pointer. Although the concept of a GUI was conceived at Xerox's PARC laboratory in the late 1970's, it was Apple with its Macintosh operating system that first employed it in a computer for general use. The term *multimedia* refers to the interaction between computer and user including graphics, text, video, speech, and often hypertext.

In the 1980's, CERN saw a clear and increasing need for researchers, students, and visiting scientists to quickly become conversant with the latest developments in physics and information processing. CERN's project included the use of their hardware and software to implement some elementary browsers for individual users, at their workstations, who incorporated their ideas into the framework.

In March 1989, the WWW was initiated as an information retrieval system based upon the client-server model, which we will study in Section 7.5. To operate the scheme, the researchers at CERN created a protocol named *HyperText Transfer Protocol* (HTTP), a measure initiated to standardize server-client communications. The WWW browser was officially released in January 1992, and the acceptance of the WWW was accelerated by the aforementioned creation of Mosaic, and the number of users of the WWW quickly became astronomical.

The WWW allows users to access the universe of data all connected to each other via *hypertext* (also called *hypermedia links* or simply *hyperlinks*), which are electronic interconnections that tie together blocks of data permitting easy access by users. The mechanism for this to work is that hypertext is essentially

an aspect of a computer program permitting a user to choose a word or phrase and obtain more data on it — a definition or related commentary within the text, for example. Mosaic introduced this notion to the WWW to allow users to employ the *point-and-click* option they had on their personal computers for some time. For instance, point at the text "hypertext" at a WWW site, and one might be taken to a document with comments on "hyperlinks." This provides users with instant access, cross-referencing to a large array of linked relevant data pertaining to their target idea. It allows users to access small pieces of data at any given time, digest it, and move on to more data through more links.

A hypertext document and its associated hyperlinks are written in *HyperText Markup Language* (HTML), which comes with an assigned URL. Users can add to the documents on the WWW by creating their own *homepages* written in HTML, which is a simple, easy-to-learn language. The users merely dictate the structure and content they want on their sites, and the detailed presentation and extraction of information is left to the users' browsers.

7.2 Pretty Good Privacy (PGP)

The e-mail encryption program, which is the topic of this section, was invented by Phil Zimmermann (see Biography 7.1 on the following page).

In the early 1980's, with some friends, Zimmermann created a company called Metamorphic Systems. He received a phone call at the company one day, perhaps one that changed the direction of his thinking for good, from a man named Charlie Merritt, who had accomplished what Zimmermann failed to do years ago: implement the RSA PKC on a microcomputer. NSA had effectively shut down Merritt's company by threatening action if they did not stop exporting their software program outside the United States. Since this was the heart of their enterprise, they had to find another way, calling companies such as Metamorphic Systems to see if their software might be incorporated in the company's hardware for export. The idea excited Zimmermann, and it inspired him to begin writing his own program for e-mail encryption using PKC.

It took a while for the ideas to develop and the relationship to evolve, but by November 1986, Merritt and Zimmermann had a project for using RSA. Nevertheless, RSA Data Security Inc. had patents on the protocols they wanted to use. Attempts were made to strike a deal with the patent holders, but nothing substantive came out of those discussions. Zimmermann, undeterred, continued to work on his ideas to produce a cipher without the explicit use of RSA protocols.

In 1990, he had developed this communications program, which he called *Pretty Good Privacy* (PGP), a name derived from a fictitious entity on a radio show, *Ralph's Pretty Good Grocery*. By 1991, Zimmermann became concerned that some impending legislation by the government might make it illegal for him to launch PGP 1.0, so he turned to the Internet. He uploaded copies of PGP 1.0 to the Internet for anyone to use, that is, *freeware*. His intention was not to profit but to make encryption available to the masses for privacy considerations. Almost overnight the program became a hit, and Zimmermann was delighted, but version 1.0 had its failings. He plugged the holes and killed the bugs in 1.0 to produce a vastly superior version 2.0. One particularly important improvement was the addition of certificates. (Think of a *certificate* as a quantity of information signed by a trusted authority.) However, Zimmermann's program had no access to such trusted authorities, which are part of what is known as *public key infrastructure* (PKI) (see [64, Section 6.2] for a detailed description of PKI), so he had to come up with a new idea. That idea was to make the *users* of PGP, themselves, the *trusted authorities*. To do this, he had the idea of signed keys, as a symbol of "trust," for the communicating parties, something he developed into what he called a *web of trust*, which we will describe below.

In September 1992, Zimmermann posted PGP 2.0 on the Internet as freeware, and as the light of 1992 faded into memory, Zimmermann was becoming a very famous man indeed. However, fame sometimes engenders costs. In 1993, he was put under criminal investigation, since the government charged that PGP was available to criminals, and they were also concerned about export regula-

tions. The exportation of strong cryptography programs, they maintained, was deemed to be equivalent to illegally exporting munitions! Fortunately, perhaps because the government finally realized the futility of this war with the Internet as the battleground, they officially dropped the investigation on January 11, 1996.

Zimmermann launched a new company called *Pretty Good Privacy Inc.* to market the software to commercial enterprises, but due to his lack of business acumen it was going nowhere fast, so he turned over the reigns to some business types. However, the company eventually went to the brink of bankruptcy before it was sold to *Network Associates Inc.* (NAI), an established computer firm, where Zimmermann remained as its figurative head as well as special adviser and consultant.

It is worth ending this anecdote with an ironic note about Zimmermann and the commercial version of PGP. During a party held by NAI at a conference in 2000, Zimmermann staged a demonstration of launching a commercial version of his product over a computer to a market abroad, an act for which he was, years earlier, put under criminal investigation. The new millennium has arrived, and privacy is no longer in the hands of private enterprise or governments.

PGP has enjoyed remarkable success and is now widely used over the globe as a mechanism for secure e-mail transmission and file storage.

Biography 7.1 Phil Zimmermann *was born in 1954 and raised in Florida. His interest in codes began at an early age, an interest which continued through his youth, so that by the time he entered Florida Atlantic University in 1972, he turned to computers as a tool for the cryptographic skills that he, independently, had honed over the years. The above discussion of the evolution of PGP shows his use of those skills.*
Zimmermann has received numerous awards for his achievements. Among them are the Chrysler Award for Innovation in Design in 1995 — see: http: //www.chrysler.com/design/design_influences/design_awards/1995/*; the 1995 Pioneer award from the Electronic Frontier Foundation; the Norbert Wiener Award from Computer Professionals for Social Responsibility, for promoting the responsible use of technology, in 1996; a Lifetime Achievement Award from Secure Computing magazine in 1998; the Louis Brandeis Award from Privacy International in 1999; and in 2001, he was inducted into the* CRN Industry Hall of Fame *— see:*
http://www.crn.com/sections/special/hof/industryHOF_Main.asp.

◆ Web of Trust

One of the trust models employed in PKI is called *user-centric trust* in which each user makes the decision as to which certificates to accept or reject. For instance, a user, such as Alice, exchanges certificates that are public keys of those other users with whom she wants to communicate. She protects her certificate from alteration by signing it with her private key. Upon receipt of

Bob's certificate, say, Alice acts as a trusted authority by assigning it one of the following levels:

(1) *Complete trust*, meaning that she trusts Bob and anyone whose certificate is signed with Bob's key.

(2) *Partial trust*, meaning that Alice does not completely trust Bob, so certificates signed by Bob must also be signed by other users (whom she does trust) before she accepts it.

(3) *No trust*, meaning that Alice does not trust Bob and will not trust any certificate signed by Bob.

(4) In some implementations there is a fourth level of *uncertain*, but this essentially amounts to no trust.

In this way she builds a *web of trust* with other users, but this model is not acceptable for such applications as e-commerce.

◆ Pretty Good Privacy (PGP)

The following is adapted from the more general description given in [64]. PGP embodies four protocols for the secure transmission of e-mail messages.

▼ PGP Protocols

1. Authentication and compression.
2. Confidentiality.
3. E-mail compatibility.
4. Segmentation.

Now we look at each of these in detail. We assume that Alice is communicating with Bob.

▼ Authentication (Digital Signature) and Compression

Protocol Steps

1. Alice creates a message, m, to be used for the purpose of authenticating herself to Bob.
2. SHA-1 (see Appendix F) is used on m to create a 160-bit message digest, $h(m)$.
3. Alice enciphers $h(m)$ with her private RSA key d_A. She sends

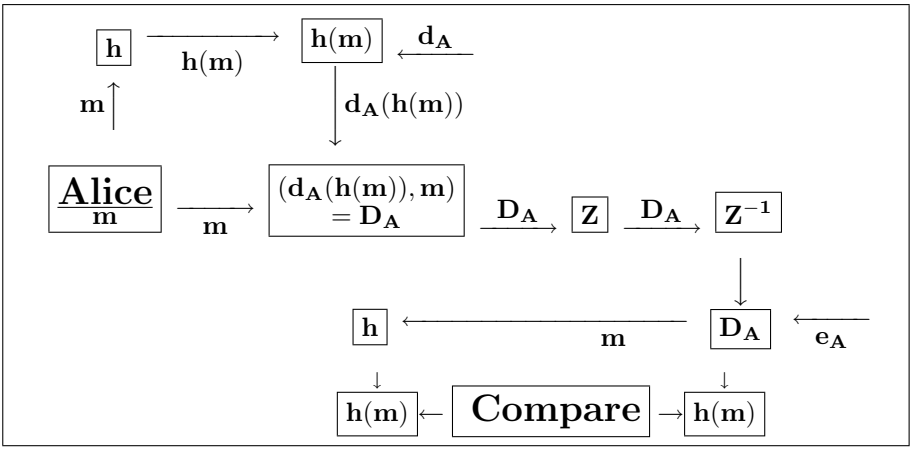
$$D_A = (d_A(h(m)), m)$$

to Bob. On the network, D_A passes through a ZIP compression operation, denoted by Z. (For details on ZIP see [64, Section 8.1].) Note however, that many different types of compression may be used here. We are being specific by using ZIP for ease of elucidation.

4. After decompression, denoted by Z^{-1}, Bob uses Alice's public RSA key e_A to decipher and recover $h(m)$.

5. Bob applies h to the value of m sent by Alice and compares the result to the value of $h(m)$ he deciphered in step 4.

Diagram 7.1 PGP Authentication



▼ Confidentiality

Several mechanisms using SKC may be used for ensuring PGP confidentiality. Among them is Triple DES (3DES) (see page 149), and this is the one we assume will be used in what follows, denoted by E herein. (For other options, see [64, Section 8.1].) Moreover, we will assume that 64-bit CFB mode is also used in what follows (see page 124). We will use RSA as our PKC, but ElGamal is also an option (see Section 4.4), as well as Diffie-Hellman/DSS (see page 167 and Section 4.5).

Protocol Steps

Alice wants to send an enciphered message m to Bob.

1. Alice generates a 128-bit nonce k to be used as a one-time-only (session) key for this message and uses it (after compression of m using Z) via 3DES to get $E_k(Z(m))$.

2. Alice enciphers k with Bob's public RSA key e_B to get $e_B(k)$ and sends

$$(e_B(k), E_k(Z(m)))$$

to Bob.

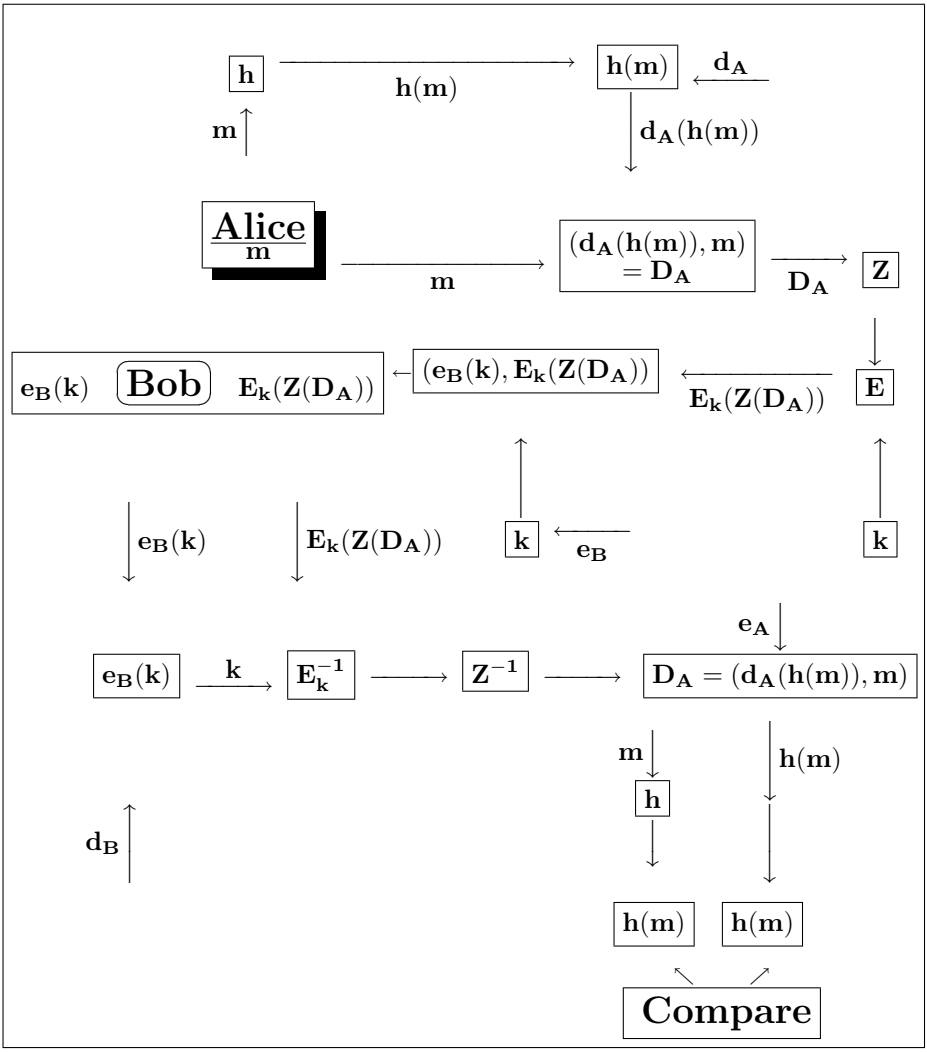
3. Bob deciphers k with his private RSA key d_B and recovers m with k (after decompression with Z^{-1}).

▼ Authentication and Confidentiality

This is illustrated in Diagram 7.2, with the amalgamation of the previous two protocols as follows.

Steps 1–3 of the authentication protocol are executed, followed by steps 1 and 2 of the confidentiality protocol (acting on $Z(D_A)$ rather than $Z(m)$). Then Bob recovers k with his private RSA key d_B and uses k to recover the compressed version of D_A via E. Then steps 4 and 5 of the authentication protocol are executed.

Diagram 7.2 PGP Authentication and Confidentiality



▼ Compression Analysis

For the purposes of efficient e-mail transmission and file storage, PGP has a built-in default mechanism that compresses m after signing but before enciphering. The order of signing vs. compression deserves some elucidation.

If Alice were to compress m, forming $Z(m)$, then sign it to form, $d_A(Z(m))$, it would be necessary to either store $Z(m)$ for the purposes of later verification by Bob, or once Bob obtains $Z(m)$ via e_A, then it would be necessary to form $Z(m)$ from m for comparison. Both of the latter options entail additional workload over merely storing $(d_A(m), m)$. Furthermore, Z is a randomized operation in the sense that the same input may produce different compressed outputs at different times, say, $Z(m) = x$ at time t_1, and $Z(m) = y$ at time t_2, with $x \neq y$. However, any version can decompress to get the correct version of compression by any other version; in other words,

$$Z^{-1}(x) = Z^{-1}(y) = m.$$

Yet, forming, say, $d_A(m)$ at time t_1 would restrict the PGP scheme to the version of Z applied at time t_1, since we would have to verify Alice's application of that version of the compression at time t_1, which is an unacceptable shackle to put on the security mechanism. Last, speaking of security, enciphering is applied after compression for increased cryptographic security since $Z(m)$ has less redundancy than does m, so cryptanalytic attacks are made much tougher on Mallory.

Before discussing the next aspect of PGP protocols, we need the following well-known method of representation of data.

▼ ASCII

ASCII is the acronym for *American Standard Code for Information Interchange*. Each symbol is represented as a 7-bit word and allows for 128 possible symbols to be so represented. Typically, a bit is appended to the 7-bit word as either a parity-check bit or an error-check bit to see if an error occurred in transmission. The mechanism for ASCII conversion is radix-64 transformation, wherein binary blocks of three bytes each are converted into four ASCII symbols, each of which is appended with an error check in the form of a *cyclic redundancy check*; see Appendix G. Note that the term "radix" is a synonym for "base."

▼ E-Mail Compatibility

Typically, PGP sends a stream of bytes of data. However, there are certain e-mail networks allowing only ASCII data to be transmitted. PGP satisfies this requirement by transforming the stream of bytes into a stream of printable ASCII characters, using an encoding technique called *radix* 64, which we describe in Appendix G. This inflates the message by 33%, but the aforementioned compression stage offsets this message expansion. In fact, a standard

analysis of the PGP mechanism shows that, even with the message expansion, the net compression is approximately one-third.

Once $h(m)$ is formed, the concatenation, $(m, h(m))$, is signed by Alice to get $d_A(m, h(m)) = D_A$, which is compressed via Z. Then she enciphers via k using E to get, $E_k(Z(D_A))$, then $e_B(k)$ is appended to get $(e_B(k), E_k(Z(D_A))$, which is converted to ASCII. Upon receipt, Bob reconverts to binary; he recovers k via d_B, which he uses to get $Z(D_A)$. This is passed through Z^{-1}, after which he uses e_A to get $(m, h(m))$. He applies h to m and compares his $h(m)$ to the version sent by Alice.

▼ Segmentation

Anyone who has tried to send a very large e-mail attachment knows that certain e-mail sites will "bounce back" the message stating that it exceeds the maximum message length allowable (which typically over the Internet is $5 \cdot 10^4$ bytes). Hence, segmentation, the splitting up of the message into smaller pieces or segments, is necessary. PGP meets this requirement by segmenting a message into manageable, acceptable blocks for easy transmission. Segmentation is done after all of the above processing is completed.

Now that we have described the basics of the fundamental protocols underlying PGP, we look in detail at the various aspects of the message transfer and reassembly, starting with the components of the message itself.

▼ Message Components

There are three basic components in a message m to be sent by Alice.

1. **Session Key**: This component has two facets. First there is Bob's identifier I_{e_B} for his public RSA key e_B, defined by $I_{e_B} \equiv e_B \pmod{2^{64}}$, namely, the least significant 64 bits of e_B. The identifier I_{e_B} is the most efficient means to transfer the key verifier to Bob that does not involve the use of too much space or too much workload to do the verifying. (Note that this identifier is essentially a probabilistic identifier in the sense that it is possible for two different public keys to have the same least significant 64 bits, but the probability is very low given the bitlength involved.)

   The second facet of the session key component is the session key k, itself.

2. **Signature**: This component has four facets. There is the timestamp t_A, which corresponds to the creation time of Alice's signature. Then there is the identifier I_{e_A} for Alice's public key, via $e_A \equiv I_{e_A} \pmod{2^{64}}$ (see the description of this device, presented for Bob's key, in part 1 above). Third, there is the message digest, $h(t_A, m)$, which is formed (with t_A appended to thwart replay attacks). Last, there are the two leading bytes L_1 and L_2, of $h(t_A, m)$, which allows Bob to ensure that the correct public key, e_A, was used to decipher the message for authentication. He does this by comparing the plaintext copy of these bytes with the first two bytes of the deciphered message digest. (Note that in the previous discussion

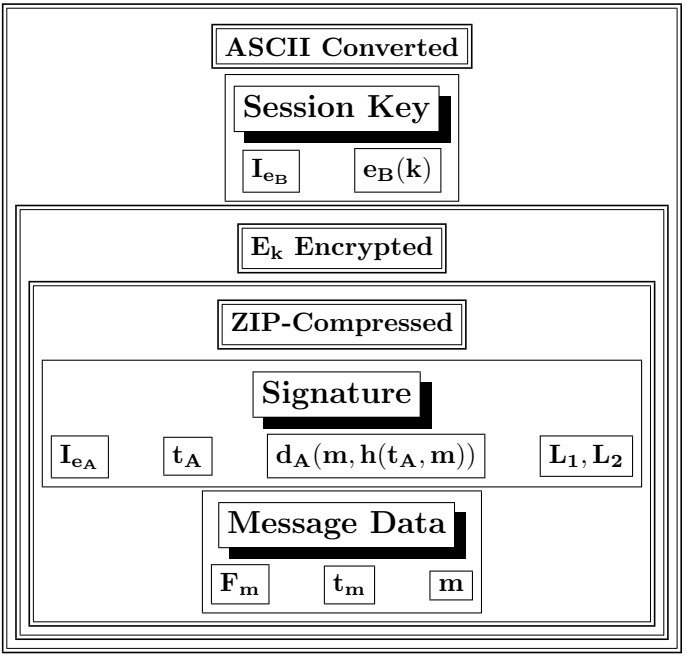
and diagrams we did not mention, explicitly, the timestamp in order to simplify the presentation. Thus, we are assuming, tacitly, that it is present and handled in the aforementioned fashion.)

3. **Message**: This is the component consisting of the message data, m, itself, accompanied by a timestamp, t_m, specifying the creation time of m, as well as a filename F_m.

Both the message and signature components are ZIP compressed, then enciphered with the session key. The session component together with the compressed components are then converted to ASCII.

In Diagram 7.3, we are assuming that the (otherwise optional) operations of: ensuring confidentiality by forming $e_B(k)$, ensuring authentication by forming $d_A(m, h(t_A, m))$, ZIP compression of the signature and message components is carried out, and ASCII conversion is executed on all components. Each of the symbols in the diagram are defined in the discussion preceding the diagram. Each double box contains a set of operations to be carried out, and the nesting of the boxes dictates the order of the operations from inner to outer.

Diagram 7.3 PGP Message Components



The next topic is a fundamental feature of PGP and is a mechanism for an individual user to communicate with entities it knows, securely, and efficiently.

From the above, it can be seen that the key identities, I_{e_A} and I_{e_B}, for Alice and Bob, respectively, provide authentication and confidentiality. Bob's public and private keys are stored securely at his computer along with public keys of others, such as Alice, with whom he communicates. PGP uses data structures to store them, called *public key rings* and *private key rings*. We now describe each of these in turn and delineate the schemes by which private keys are securely maintained.

▼ Key Rings

Essentially, key rings are flat files containing a sequential list of keys. How the key is stored is up to the implementer. The private key ring is stored only on Alice's computer, which stores the RSA key pairs owned by her, and is accessible only to Alice. In the private key ring, each entry for an entity has the following fields (but typically she will only have one entry, namely, her own public/private key pair).

▼ Private Key Ring Individual Field Entry

1. **Timestamp**: t_A, the creation time of (e_A, d_A).
2. **Key ID**: $I_{e_A} \pmod{2^{64}}$.
3. **Public Key**: e_A.
4. **Private Key**: d_A (enciphered using CAST-128, 3DES, or IDEA). The actual key d_A is not stored on Alice's computer, only the encrypted version. Here is the actual mechanism by which Alice accesses the private key, when needed, in order to achieve maximum security.

   Private Key Storage and Access Steps

   (i) Alice chooses a passphrase that she will use for enciphering private keys. (It is paramount that she keep this secure, never write it down, or disclose it to anyone.)

   (ii) When the PGP program generates a new RSA key pair, such as (e_A, d_A), it will prompt Alice for her passphrase, P, and using SHA-1, a 160-bit hash $h(P)$ is formed, and the passphrase is discarded.

   (iii) The program enciphers d_A, using an SKC, E (which is one of 3DES, IDEA, CAST-128 or AES), with $h(P)$ as the key, namely, to form $E_{h(P)}(d_A)$, and discards $h(P)$. Then $E_{h(P)}(d_A)$ is stored on Alice's private key ring.

   (iv) Whenever Alice wants to access d_A, she must provide the passphrase. The PGP program provides her with $E_{h(P)}(d_A)$, generates $h(P)$, and deciphers d_A using E with $h(P)$, namely, via

$$E^{-1}_{h(P)}(E_{h(P)}(d_A)) = d_A.$$

5. **User ID**: ID_A, which could be, for instance: *Alice@PGPprivateRing.com*.

▼ Public Key Ring Individual Entry

This ring is used to store the public keys of other users, such as Bob, with whom Alice communicates. The following are the fields in Bob's entry. Items 4, 6, and 8 are under a framework, called a *trust-flag-byte*, the contents of which are described individually in each field entry, and refer to the web-of-trust model described on pages 228–229.

1. **Timestamp**: t_B, which is the creation time of the entry.
2. **Key ID**: $I_{e_B} \pmod{2^{64}}$.
3. **Public Key**: e_B.
4. **Owner Trust**: trust-flag-byte, which is the trust, assigned by Alice, that indicates the degree to which e_B can be trusted to sign other public-key certificates. When a new public key is to be added to the public-key ring, the PGP program prompts Alice to assign a level of trust to the key owner, Bob in this case. When the level of trust is *complete trust*, then the public key is also put on Alice's private-key ring. In the case where Bob's key appears on Alice's private-key ring, there is a *buckstop bit*, which is set to 1 in that instance.
5. **User ID**: ID_B, which is Bob's identifier, such as *Bob@PGPpublicRing.ca*.
6. **Key Legitimacy**: trust-flag-byte, which is the level of trust that the PGP program (which computes this field), imparts to the binding of Bob's user ID to e_B. The means by which this is determined by the PGP program is on a weighted basis, whereby the PGP program bases the weighting upon the signature trust fields present in item 8. There is also a *warnonly bit*, which is set to 1 if Alice only wants to be warned that e_B is only used for enciphering but is not fully validated.
7. **Signature**: When a new public key, Bob's in this case, is added, one or more signatures could be appended to it, and more may be added later.
8. **Signature Trust**: trust-flag-byte, which is the degree of trust that Alice assigns Bob to certify public keys, so is essentially a cached version of field 4 (owner trust), in the following sense. Upon addition of a signature, the PGP program looks through the public-key ring to determine if Bob's signature is among the public-key owners therein. If so, the trust value given in field 4 is assigned, and if not, an *unknown* value is assigned to this field. This field is periodically updated by the PGP program, which scans the public-key ring for all signatures owned by Bob and updates this field to be the same as the owner trust field.

Now that we have the notion of PGP rings, we can give a more detailed and informed description of PGP message generation, processing, and reception.

▼ PGP Message Processing Protocol Via Key Rings

This protocol description, and accompanying diagrams on pages 238–239, depict the PGP message generation and processing upon reception using key rings. Since we fully described the mechanism for ASCII conversion and ZIP compression above, we eliminate those stages for the sake of simplicity. Moreover, we are assuming that both signing and encryption are required.

Protocol Steps

We assume, as above, that Alice is sending a message to Bob.

1. The PGP program obtains Alice's encrypted private RSA key d_A from her private-key ring using ID_A (for instance, *Alice@PGPprivateRing.com*) as an index for so doing.

2. The PGP program requests Alice's keyphrase in order to provide her with this enciphered version, which she provides and d_A is obtained as in part (iv) of private-key storage and access on page 235.

3. Alice generates the message m, and the digital signature $d_A(h(m))$ is formed as in the authentication protocol described on page 230. However, the public-key identifier, I_{e_A}, her public-key identifier from the signature component of the message (see part 2 on page 233), must be appended to the signature since Bob must know which public key is intended for use given that Alice could have many private keys.

4. The PGP program uses a random-number generator to create a session key k, as above, and forms $E_k(m)$.

5. The PGP program gets e_B, Bob's public key from Alice's public-key ring using ID_B (for example, *BOB@PGPpublicRing.ca*) as an index.

6. Then the PGP program forms $e_B(m)$, and $(e_B(k), E_k(m))$ is sent to Bob.

7. Upon reception, the PGP program obtains Bob's encrypted private key, d_B, from his own private key ring using I_{e_B}, from the session key component of the message (see part 1 on page 233), as an index.

8. The PGP program requests Bob's passphrase, which he delivers, and decrypts to get the session key, k, which is used to recover the message $(d_A(h(m)), m)$.

9. The PGP program gets e_A from Bob's public key ring, using I_{e_A} from the signature component of the message (see part 2 on page 233), as an index. This is used to recover the $h(m)$ sent by Alice.

10. The PGP program computes $h(m)$ from Alice's sent message m and compares it with the $h(m)$ sent by Alice for authentication.

Diagram 7.4 PGP Message Generation and Encryption Via Rings

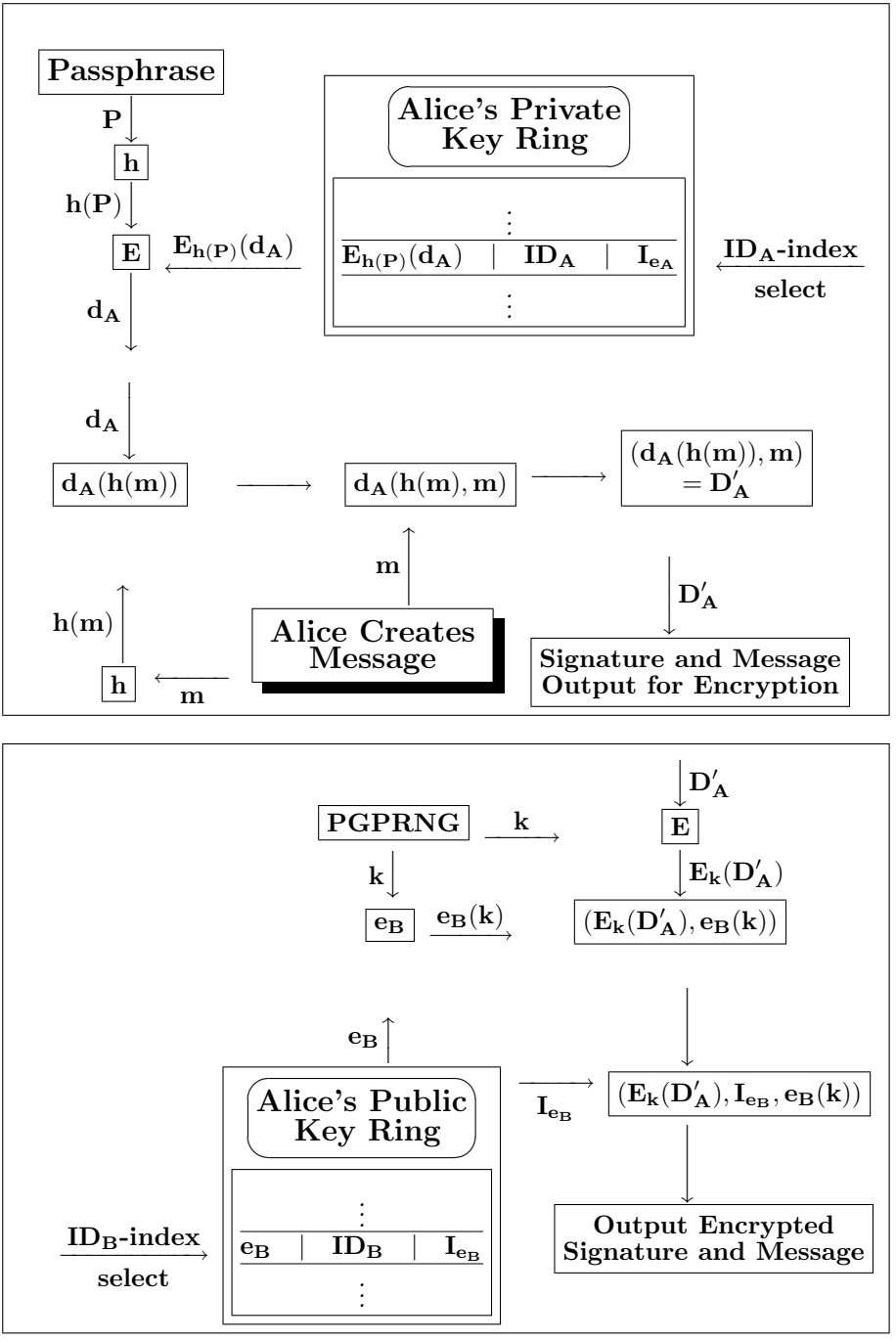
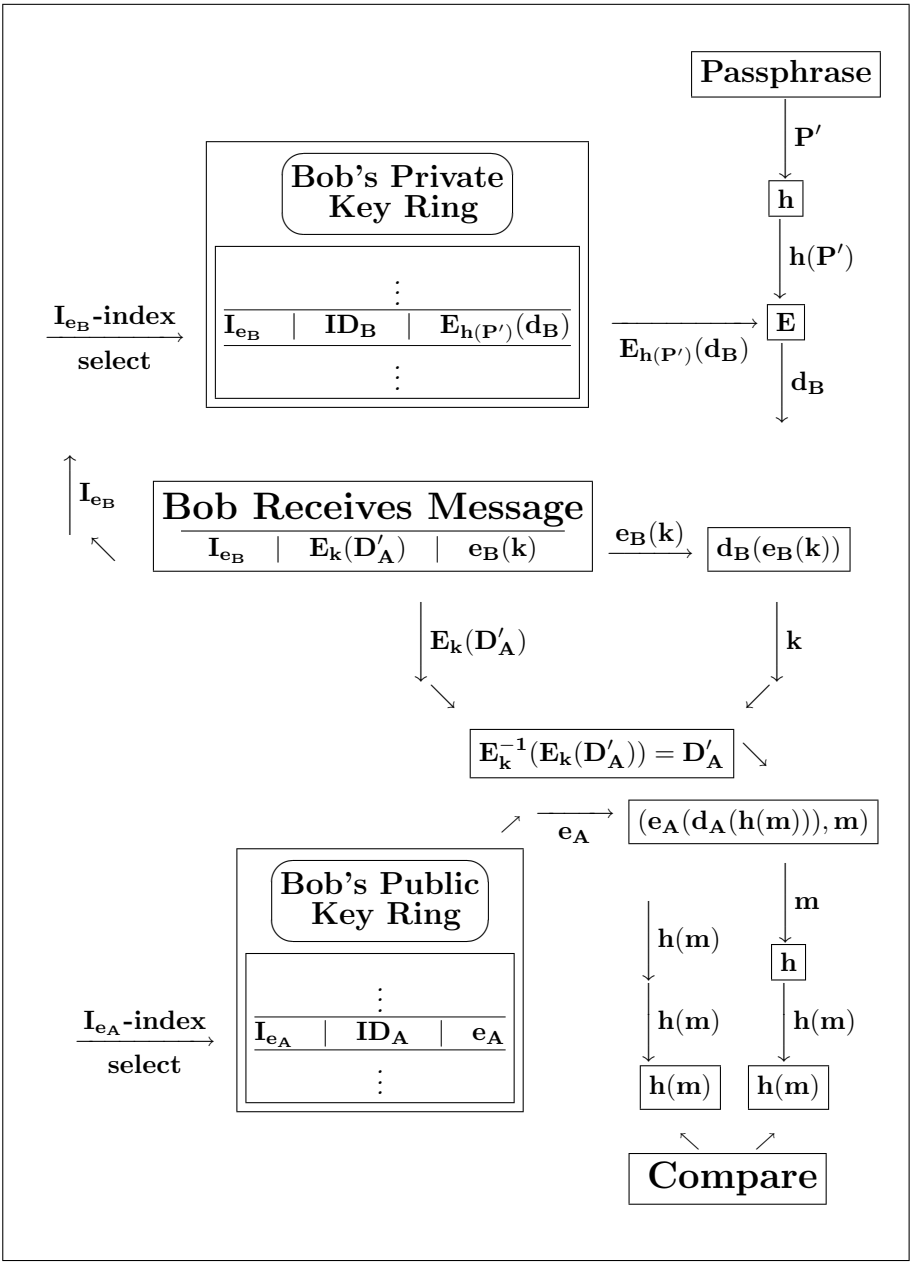


Diagram 7.5 PGP Message Reception, Decryption, and Authentication



▼ Analysis and Summary

PGP utilizes a package of algorithms, in a general-purpose application, which is operating system and machine independent, embodying only a few simple operations. It is freeware for individuals and of moderate cost to commercial enterprises who enjoy vendor support. Moreover, the scheme is independent of government control. RFC2440 (see [77]) contains the cryptography in OpenPGP. Note that documents called RFC's, *Requests For Comments*, are the official working notes of the Internet research and development community. See

http://www.rfc-editor.org/rfcxx00.html.

The trust model used by PGP does not include a PKI specification, but its web-of-trust approach (see pages 228–229) does provide a convenient trust-use mechanism for the purpose of linking trust with public keys, as depicted by our discussion of public and private-key rings on pages 235–239. This is a particularly clever and innovative means of dealing with one of the principle weaknesses of PKC, namely, the protection of public keys from being compromised.

It should be noted that although we used specific protocols in the above description, PGP is parameterized so that many different protocols may be used as specified in RFC2440. For instance, to create a PGP key today one might typically employ AES and SHA-256.

In conclusion, PGP embodies an interwoven collection of protocols (including public-key management) in an efficient yet secure manner to ensure authentication and confidentiality of e-mail services, as well as file storage.

▼ Zimmermann's Reason's Why PGP Should Be Used

The following, attributed to Phil Zimmermann, is an appropriate closure to this section.

"If privacy is outlawed, only outlaws will have privacy. Intelligence agencies have access to good cryptographic technology. So do the big arms and drug traffickers. So do defence contractors, oil companies, and other corporate giants, but ordinary people and grassroots political organizations mostly have not had access to affordable military grade public-key technology. Until now.

PGP empowers people to take privacy into their own hands. There's a growing social need for it. That's why I wrote it."

See the following hyperlink for the complete original English text:

http://www.pgpi.org/doc/whypgp/en/.

Exercises

7.1. The description of PGP herein assumed the use of triple DES. Explain why single DES would not be suitable for use with PGP.

7.2. Why would PGP use CFB mode over the more commonly used CBC mode?

7.3 Protocol Layers and SSL

To become acquainted with the notion of a "protocol layer," we must understand its (formal) inception, which began with the following organization.

▼ ISO and OSI Models

The *International Organization for Standardization* (ISO), a nongovernmental body created in 1947 to promote the development of standardization and related activities and embodying members from 148 countries, is a world federation of national standards organizations. ISO is *not* an acronym, rather its roots are from the Greek *isos* meaning *equal*, which will be recognized as the prefix *iso-*, such as in *isometric*. It happened that *equal* devolved to *standard*, and the ISO name was adopted. Additionally, this provides the feature of not requiring translation for various languages, as would an acronym.

ISO develops precise criteria for such applications as the development of a framework of international standards in computer networks, for instance. (A *network* is a hardware and software communications system.) In 1978, ISO developed a model of network protocols, called a *protocol stack*, which is a layered set of protocols working together to render a set of network functions.

The ISO model divides the architecture among seven layers, where we understand a *layer* to be the environment of two or more communications devices in which a particular network protocol operates. The ISO model is called the *Open Systems Interconnection Reference Model* (OSI-RM). OSI is the umbrella name for a set of nonproprietary protocols and specifications, which includes the OSI-RM, having the following seven layers, from the bottom to the top. Much of the following has been adapted from [64].

◆ OSI-RM Seven Layer Protocol Stack

1. **Physical Layer**: This bottom layer deals with electrical and mechanical connections to the network.

2. **Data Link Layer**: This layer splits data into *frames*, which are *data packets* containing the header and trailer information required by the physical layer. The data link layer executes error checking and retransmits correct frames for any corrupted frames it receives, thereby providing an error-free connection to the next layer up to which it sends the frames.

3. **The Network Layer**: This is the communications subnet layer, which decides the routing of packets received from the data link layer to be used by the next layer up. Most commonly, IP is used (see page 224).

4. **The Transport Layer**: This middle layer is essentially the communications system component of a given protocol. For instance, the TCP protocol, discussed on page 224, is one such communications system. Although TCP itself is not cryptographically secure, mechanisms can be used to make it

so. For instance, in 1996 the IETF (see page 224) formed a committee, the *Transport Layer Security* (TLS) working group. Their mandate was to develop a standard for *Secure Sockets Layer* (SSL), a protocol that originated at Netscape in 1994 and which we will describe in detail in this section.

In January 1999, the TLS working group published the TLS protocol. However, TLS is essentially a version of SSL, so we will not describe it here but rather wait to get the full description of SSL, which is not, in itself, a single protocol but rather two layers of protocols using TCP to provide a secure connection with WWW browsers — see Section 7.1.

To summarize, and expand the role of this layer, essentially the transport layer decides how to utilize the network layer to render a virtual error-free connection between hosts. Thus, it both initiates and terminates connections between hosts.

5. **Session Layer**: This layer uses the transport layer to establish a connection between hosts for certain processes, so it essentially handles the security side and the creation of the session itself.

6. **Presentation Layer**: This layer executes such functions as text compression and format conversions, which is the mechanism for ironing out differences between two hosts. If there are incompatible processes in the next layer up, the presentation layer allows the process to communicate via the session layer.

7. **Application Layer**: This top layer essentially handles the user's needs. For instance, it deals with such issues as allowing a user to access a remote resource through a network without having to know if the resource is remote or local, a feature called *network transparency*. It will style itself after the user's particular desires such as e-mail message formatting.

   The application layer also deals with resource allocation and problem partitioning. The presentation layer provides the top layer with familiar local representation of data, which is independent of the format used on the network.

▼ Analysis

Network Connections embody a set of independent protocols, each in a different layer. The only variable layer is the applications layer. The function of each other layer is to employ the layer one step below it and provide a service to the layer one step above it. Each of the network's components on a given host uses protocols applicable to its layer to communicate with its analogous component in another host. Such layered protocols are sometimes known as *peer-to-peer protocols*.

A fundamental strong point in the use of layered protocols is that the mechanism for delivering information from one layer to another is specified clearly

as part of the protocol's definition. Moreover, changes within a protocol layer are prevented from affecting the other layers, which vastly simplifies the task of designing and maintaining network communication systems.

◆ SSL Protocol — Simplified

SSL is an Internet protocol that provides authenticity and secrecy for session-based communication. It provides a secure channel on the client/server model (see Section 7.5) using a secret sharing scheme. The security model of SSL is that it encrypts the channel by enciphering the bits that go through that channel. As mentioned earlier, SSL began with Netscape, who originated it, and in 1996 they handed over the specifications of SSL to IETF, who worked to standardize the SSL version 3 model, which had been released in 1995. In 1999, the TLS working group released TLS version 1, which has now become the IETF standards-track variant of the SSL version 3 protocol (see [22]). The cryptographic power of SSL/TLS is that it operates at the transport level so HTTP runs on top of SSL, called HTTPS.

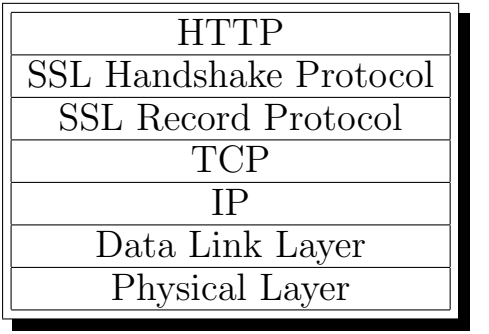
To understand the layers of SSL, we must introduce the names of the two main subprotocols to be discussed in detail below:

(1) the handshake protocol

which operates above the

(2) record protocol.

This is illustrated below.



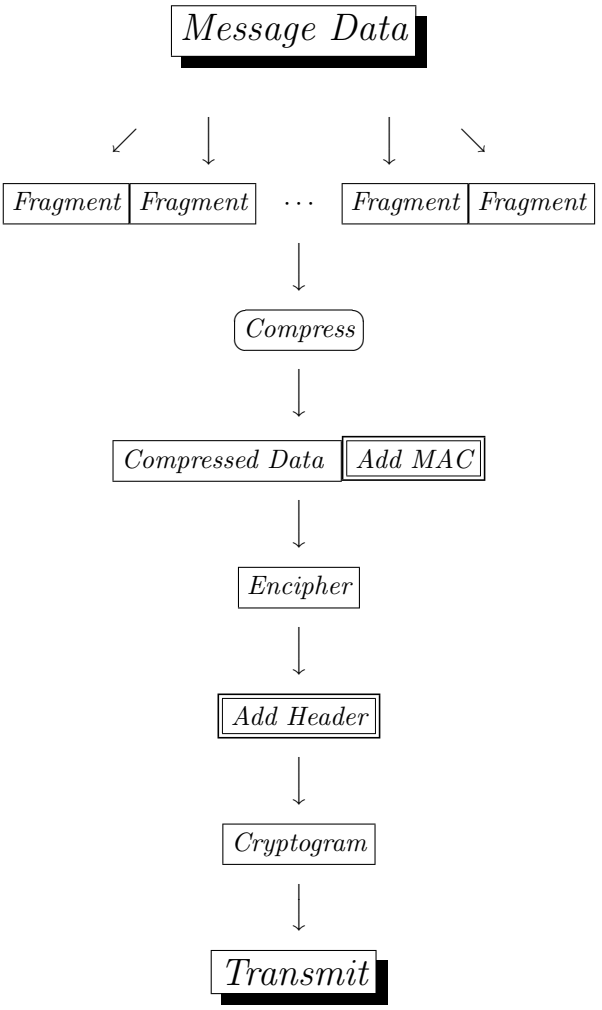
We begin by describing the lower level of SSL.

◆ SSL Record Protocol

This protocol defines the format used to transmit data and is used by the handshake protocol to exchange messages between client and server.

First the message to be transmitted is *fragmented*, which means it breaks the message down into manageable blocks. Then it compresses the data (but this is an optional exercise in SSL). It then applies a MAC (see page 125), enciphers the data, adds a header, and transmits the cryptogram as a TCP unit. This is illustrated in Diagram 7.6 on the next page.

Diagram 7.6 SSL Record Protocol Actions



Upon receipt of the transmitted data, it is deciphered, authenticated, decompressed, reassembled, and delivered to users at higher levels.

▼ Analysis

The message data is typically fragmented into blocks of 2^{14} bytes, after which data compression is optional. The encryption is done with an SKC cipher, which can be any of the *suite of ciphers* (or sets of ciphers) supported by SSL, listed below in order of cryptographic strength. These are the cipher suites for SSL implementations that use the RSA key-exchange algorithm.

1. Triple DES (see page 149), using 168-bit encryption and SHA-1 message authentication (see Appendix F).
2. The only stream cipher, RC4 (see Appendix H), using 128-bit encryption and MD5 message authentication (see Appendix F). When the RC4 is used, the MAC is first computed, then the MAC and compressed data are enciphered.
3. RC2 (a block cipher developed by Rivest for RSA Data Security), employing 128-bit encryption and MD5 message authentication.
4. DES (see Section 3.1) with 56-bit encryption and SHA-1 message authentication.

SSL supports the above variety of cipher suites since clients and servers may support different ciphers depending upon numerous factors.

The following protocol shows how the server and client authenticate one another, send certificates, and establish session keys. (See Section 7.5 for a general description of the client-server model.)

◆ The SSL Handshake Protocol

Below there are actions that are mandatory, situation dependent, or optional. We will call those that are either situation dependent or optional merely *optional* for simplification of presentation in Diagram 7.7 on page 248.

I Contact and Establish Capabilities:

1. The client sends the server a *client-hello* message, which contains the following fields:
   - (i) The client's SSL version number (usually the highest SSL version supported by the client).
   - (ii) Cipher suite (usually listed in decreasing order of preference), each element (cipher suite) of which includes both a key-exchange algorithm and the details of the cipher proposed. The following is the *SSL key-exchange suite of algorithms*:
     - (a) **RSA**: The RSA public key of the recipient is used to encipher the secret key, but in order to validate the process, a public-key certificate for the recipient must be accessible.

(b) **Authenticated Diffie-Hellman**: In this type of key exchange, it is mandated that the server certificate contains the Diffie-Hellman public-key parameters, authenticated (signed) by Trent as a CA. If the client is required to send a certificate (see step 3 of stage II, below), then the public-key parameters as so included (see step 2 of stage III). Hence, the Diffie-Hellman-generated secret key is fixed in this case.

(c) **Anonymous Diffie-Hellman**: Essentially the Diffie-Hellman key exchange as given on page 167 is used with no authentication, meaning that the SSL handshake protocol supports a totally anonymous operation in which neither the client nor the server is authenticated. As we saw with the Diffie-Hellman protocol in particular, and with PKC in general (see Diagram 4.5 on page 177), impersonation is possible since the entities are not authenticated, leaving the scheme open to the man-in-the-middle attacks.

(d) **Fortezza**: The *Key-Exchange Algorithm* (KEA) is the key exchange algorithm used with Fortezza. KEA was declassified by the U.S. Department of Defence on June 23, 1998. KEA requires a 1024-bit prime modulus, generated via the DSS specifications in [32]. Moreover, KEA is based on a Diffie-Hellman protocol that uses SKIPJACK for the purpose of reduction of final values to an 80-bit key (see [31]).

(iii) Some randomly generated data consisting of a 32-bit timestamp and 28 bits generated by a CSPRNG (see page 109), both of which are treated as nonces (see page 124) to prevent *replay attacks*, also called *playback attacks*, which are attacks employing data obtained from a previous execution of the protocol for the purpose of deception.

(iv) List of compression methods supported by the client.

(v) A variable-length session ID.

2. The server sends a *server-hello*, which consists of the same parameters as the client-hello. For instance, the server selects a cipher suite from the list proposed by the client, and the server chooses a compression method from the client-proposed list. However, the random field is generated by the server independent of the client-generated random field.

II Key Exchange and Server Authentication:

1. The server sends an identification certificate to the client (required for all key exchanges except anonymous Diffie-Hellman). (If RSA is used, we assume that the server's public key was sent with the certificate.)

2. The server sends a server-key-exchange message (*not* required only if either the server has sent a certificate with authenticated Diffie-Hellman parameters in step 1, or if RSA key exchange is used). If exercised, this contains the server's public keying material.

3. If the server is *not* using anonymous Diffie-Hellman, it may send a request for the client's certificate. Contained in the client certificate request is a certificate type that dictates the PKC to be employed. For instance, if either RSA or DSS is used with authenticated Diffie-Hellman, then authentication (only) is accomplished via an RSA or DSS signature on the certificate.

4. The server sends a *server-hello-done* message.

III Key Exchange and Client Authentication:

1. After receiving the server-hello-done message, the client verifies the server's certificate if sent and other server-hello parameters. If all is valid, the client responds.

2. If requested, the client sends a certificate. If authenticated Diffie-Hellman is being used, then the client's public-key parameters are included.

3. The client-key-exchange message must now be sent. The key-exchange mode dictates the content as follows:

   (i) If RSA is used, then the client generates a 48-byte *premaster secret*, which is encrypted with the server's public key (sent with certificate in Stage I).

   (ii) If anonymous Diffie-Hellman is employed, then the client's public Diffie-Hellman parameters are sent.

   (iii) If authenticated Diffie-Hellman is used, then the parameters were already sent in step 1 of stage II, so this is a null action.

   (iv) If Fortezza is used, then the client's Fortezza parameters are sent.

3. If a certificate has been requested, the client signs a piece of datum that is unique to the handshake and known by both client and server, along with the encrypted premaster secret.

IV Finish Protocol:
To simplify the final stage, we assume that RSA is being used.

1. If the server verifies the client's identity, then the server uses its private key to decipher the premaster secret. Then the server performs a sequence of steps to create the *master secret* from the premaster secret, a one-time 48-byte generated for this session. These same steps are followed by the client to recover the master secret.

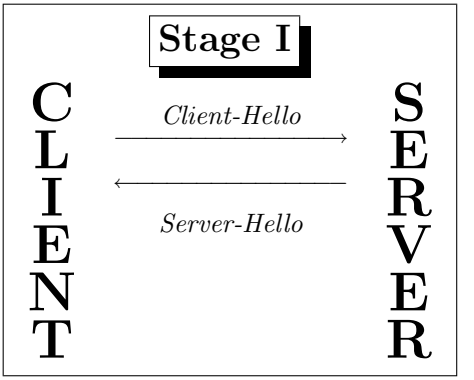
2. Both the client and the server use the master secret to generate session keys, which are symmetric keys used to encipher and decipher information exchanged over the course of this SSL session, and to verify its *integrity*, meaning the detection of changes that might have occurred in the time period from transmission to reception.

3. The client sends a *client-finished message* saying that all future messages will be encrypted with the session key, the first message encrypted with the secret session-key independently generated by the client and server.

4. The server sends a similar encrypted *server-finished message*, which assures the client it is communicating with the server since the client sent the premaster secret encrypted with the server's public RSA key, which only the server could have deciphered to calculate the session key.

5. The handshake is now completed and the client and server may exchange application layer information with a secure connection. (Caution must be exercised in certain generic implementations of SSL. See [48], for instance.)

Diagram 7.7 SSL Handshake Protocol Actions



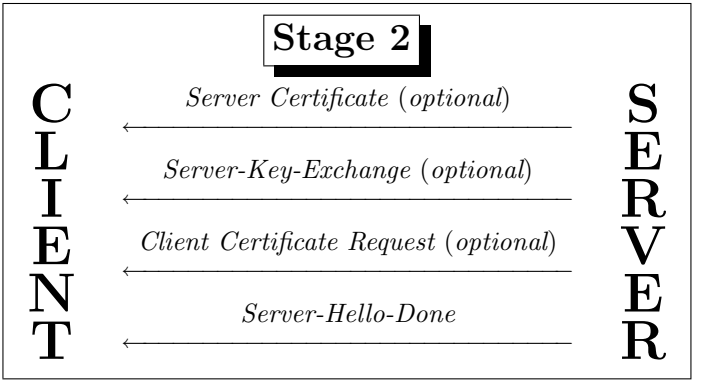
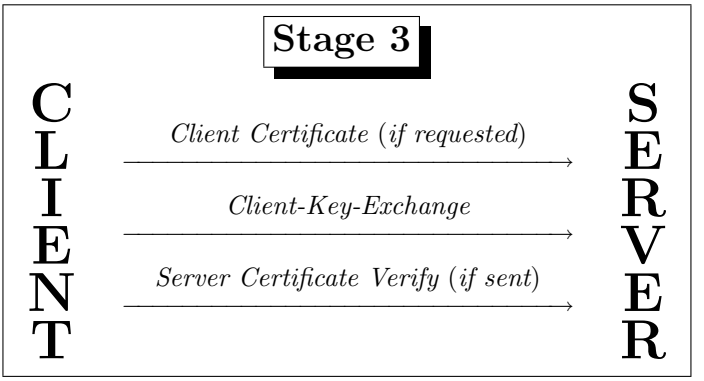
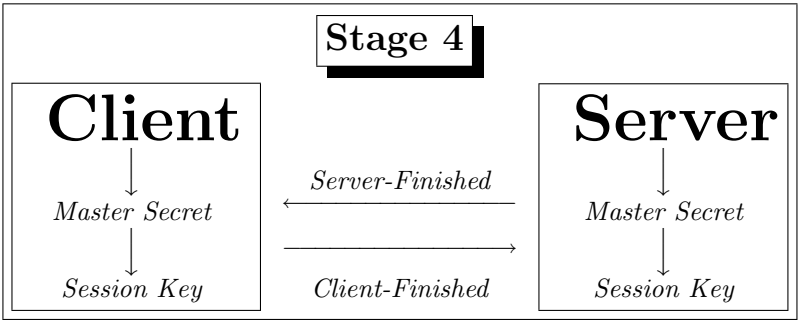


Diagram 7.7 SSL Handshake Protocol Actions (continued)





▼ Analysis

SSL server authentication allows a user to confirm a server's identity, which is quite important if, for instance, the user is sending a credit card number over a network and needs to check the receiving server's identity. SSL client authentication allows a server to confirm a user's identity. This is very important if, for example, a bank is sending confidential financial information to a customer and needs to check the recipient's identity. An enciphered SSL session is protected by a tamper-detection mechanism, which automatically checks to see if information has been altered in transit, a secure hybrid cryptosystem, with the handshake allowing independent creation of symmetric keys for fast enciphering, deciphering, and tamper detection during the session.

If authentication of client and/or server is chosen, management of certificates is required, which makes the use of SSL somewhat unwieldy given the necessity for a proper management of those certificates. Yet, the certificates render a scalable key-management scheme, which is a powerful mechanism. Of course, a totally anonymous SSL mode provides no authentication and opens the scheme up to the man-in-the-middle attack, as noted earlier. However, when users want to take advantage of SSL on their website without being associated with their host, then anonymous SSL is the way to go. Hence, the anonymous SSL server has its place, and there are numerous vendors available to sell such packages.

7.4 Internetworking and Security — Firewalls

IPs, or Internet Protocols, provide services for connecting hosts over various disparate networks, as we have seen. To accomplish this, however, each IP must be embedded, not only at each host site and its associated network, but also in routers. This presents challenges for these routers since they connect such dissimilar systems. Here are some of the differences routers face. (Much of what is in this section and Section 7.5 is adapted from [64].)

◆ Network Dissimilarities

Address Labels: The various schemes for networks to allocate a target address to data in an Internet mechanism may range from 48-bit assignments to encoded decimal representations. Therefore, some kind of universal standardization is needed together with a central archive for record keeping.

Fragmentation: On page 243, we already met the concept of message fragmentation. Fragmentation is required because of network disparities in maximum packet sizes permitted.

Interfaces: A router must be designed to execute its duties irrespective of the disparate hardware and software interfaces among networks.

Network Dependability: A router must be independent of the differences in network reliability, which may range from unreliable to end-to-end dependability.

◆ Firewalls

All the above being said, the primary concern is with local security, so we need firewalls. The term "firewall" is taken from the firefighting profession, wherein a firewall is a barrier constructed to prevent the spread of fire. In the computer world, it means keeping the flames of disaster, ubiquitous on the Internet, away from your local network and preventing entities from inside the local network from opening a "door" that will let those flames in. A firewall may be defined as a combination of hardware and software, located at the interface between two networks, that enforces an access control security policy between them. For instance, these security gateways may screen IP addresses, or ports requested on incoming connections, to decide what traffic is permitted into the local network. A *gateway* is an access point on a network that plays the role of an entrance to another network. More generally, a *node* on the Internet is a connection point, typically with the capacity to read, process, and forward data to other nodes. Thus, a node may be a computer or other device. For a user at home, an ISP (see page 225) is a gateway. For a business enterprise, a gateway node may play the role of both a proxy server (see page 254) and firewall.

Origins of Firewalls: Development of firewall architecture has been contemporaneous with the evolution of the Internet. Not surprisingly, initial funding for firewall research was the domain of the U.S. Department of Defence.

The origins of the first commercial firewall architecture may be traced to the mid-to-late 1980's with Cisco Systems, who introduced (static) *packet filters*. In the late 1980's and early 1990's the next generation of firewalls called *circuit level firewalls* came out of research at AT&T Bell Labs. Then the third generation of firewalls came to attention in the early 1990's, out of work from Bell Labs and others, with *application layer firewalls*. A fourth generation, called *dynamic packet filtering firewalls*, sometimes called *stateful inspection*, was epitomized by *Firewall-1*, the first user-friendly firewall architecture, released by Check Point Technologies in 1994. This essentially replaced static packet filtering as a standard. Today there is the fifth generation of firewall, called *Kernel Proxy Architecture*, the first commercial incarnation being Cisco System's *Centri Firewall*, released in 1997. All of the aforementioned types will be discussed below.

▼ **Firewall Design Principals**: If the security goal of a local network that has its own local security policy is to explicitly deny all transmission that fail those criteria, then the following firewall design goals should be sought: (1) all data traffic into and out of the local network must physically be directed through the firewall, and (2) the firewall must be impenetrable.

The local security policy will dictate the level of monitoring and what traffic will be permitted or denied access. Typically a local network will want a balance between protection of that local system from threats and access to the Internet.

What A Firewall Can Do

◗ First of all, in general terms, firewalls guard against unauthorized access from outside the protected local network but allow access from within the local network to the outside. A more intricate firewall scheme will ensure that certain entities within the local network are prevented from accessing certain sensitive documents inside, as well as prevent users from within the local network from sending confidential, sensitive, or vulnerable data outside the firewall.

◗ A firewall provides a single *choke point* where security, audit, tracking (of logins, Internet usage, etc.), and other management functions may be concentrated into a single system. Security alarms can also be set.

◗ Firewalls may also serve the function of *Network Address Translator* (NAT), by which it can alter data in packets to change the network address, which means one set of IP addresses is used for local network traffic and another is used for external traffic. The firewall would have a *NAT box* installed to make all the requisite IP address translations. In this fashion, the firewall hides all local network IP addresses. Moreover, behind the firewall in the local network, the use of a distinct set of IP addresses means there is no conflicting intersection with IP addresses from outside.

What Firewalls Cannot Do

◗ A firewall cannot thwart attacks that go around it. There might be a dial-out server behind the firewall, for instance, that circumvents it by dialling directly to an ISP.

◗ If there are hackers within the local network, the firewall will not detect them. Possibly, an employee of a corporation operates in concert with Mallory outside the corporation to steal vital data by giving him needed passwords. No firewall can prevent this.

◗ A firewall is not an antivirus program. Thus, infected files or programs may get through. A firewall is not the place for virus-control software, since there are simply too many ways for viruses to be sent. It would be virtually impossible for a firewall to filter every piece of datum for a possible virus. Furthermore, even if it could be implemented, it would still guard against viruses only from the Internet. There are viruses that come in CDs, via modems, as well as via the Internet. A better mechanism is to have antivirus software installed in every individual computer in the local network.

◗ A firewall is only as secure as the operating system (OS) in which it sits. If there are weaknesses in the OS, a firewall cannot protect against them.

▼ Basic Kinds of Network Firewalls

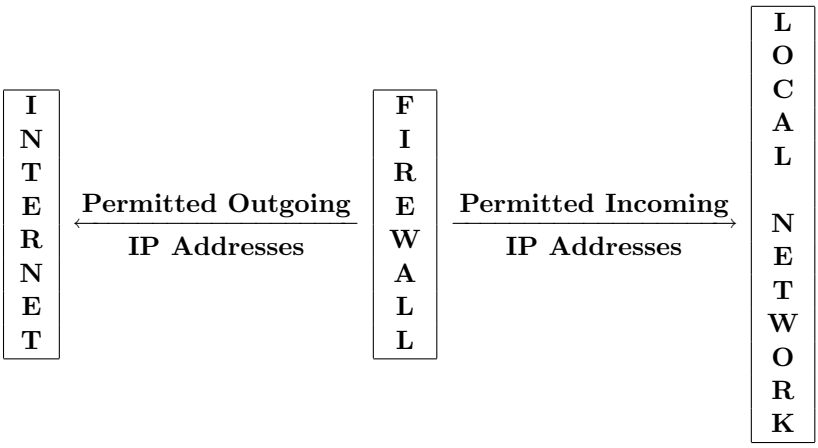
1. **Packet Filters — Screening Routers**: A simple firewall configuration is called a *packet filter*, which records the permitted origins and target IP addresses, as well as port number. (Port numbers are integers ranging from 0 to 65,000, which allow data to be sent directly to a specific device that is "tuned in" to the designated port on a target computer. Port numbers less than 1024 are for use and assignment only by a systems administrator. Typically, a port on a computer is specified by the IP address [of the computer on which the port is active], followed by a colon, and the number of the port, such as 123.214.2.7:60.) If a packet has an address that is not on its list, it is discarded. Given its simplicity, this type of firewall is both efficient and transparent to users, as well as being inexpensive to implement. However, this very simplicity makes it vulnerable to such attacks as *network layer address spoofing*.

   Spoofing: In general (not necessarily computer-related) terms, spoofing means assuming another entity's identity. In a computer context, IP spoofing, faking the origin of a message, was an idea tossed around the cryptographic community in the 1980's. It first appeared in reality when there was a problem discovered with the TCP protocol, called *sequence prediction*. Unfortunately, IP spoofing is a problem intrinsic to the TCP/IP model. Yet there are measures to be taken as we will see below. First, we look at some spoofing attacks.

   In the case of Mallory, say, trying to breach a firewall, he might use a (source) IP address of a local network host in the hope of his packet being delivered by a system that "trusts" the IP addresses of internal hosts. Some examples of IP spoofing are man-in-the-middle attacks (see page 123). For instance, there is the *routing redirect attack*, where data are redirected from the original host to Mallory's host. There is also the *source routing attack* where Mallory redirects individual packets. IP

spoofing is used almost always in denial-of-service attacks, wherein Mallory might spoof a source IP address to thwart tracing his steps, and thus stopping the attack is made that much more difficult. (Typically a *denial-of-service* attack [DOS] is one that impedes the normal functioning of communications sites, which can invovle anything from disrution of the entire network to suppressing all messages to a particular target site, or the opposite, namely overloading the network with messages.) These are but a few of many attacks involving spoofing.

Diagram 7.8 Simple Firewall: Packet Filter



2. **Stateful Inspection Packet Filters — Dynamic Filtering**: Since the aforementioned packet filter firewall bases its decisions on whether the IP address or port number correspond to those listed in the packet filter's configuration, the filtering process is *static*. However, there is a methodology wherein it is possible to incorporate the notion of the *state* of a connection into a packet filter. This is accomplished by using a *state table* and some data in the TCP headers to record those packets previously given access within a connection. In other words, stateful inspection keeps track of an IP packet over a period of time; that is, it "remembers" the interaction between the local network and the Internet. This makes it possible to thwart unauthorized incoming traffic. This implementation of a packet filter is called *stateful inspection packet filtering*. Packets leaving the local network that require a particular kind of incoming packet are recorded. Any packets coming into the local network are allowed only if they embody an appropriate response. Whereas static packet filtering essentially checks only headers, dynamic filtering of packets looks at the packet in context, namely, all the way to the application layer. With dynamic filtering, a network administrator is allowed to define the guidelines to satisfy the requirements of the local network.

3. **Application-Level Gateways — Proxy Servers**: First we get some terminology in order. A *server* may be viewed as a program, or computer, that provides services to other programs, or computers. A *proxy server* is a server that acts as a go-between for a user in a business enterprise, say, and the Internet so that enterprise can ensure security and control, as well as possibly caching. A *cache* is a memory location that stores data for quick access. For example, if a user requests a WWW page and the proxy server has a cache with that page already in it, downloaded previously for another user, then that page can be forwarded immediately to the next user on request. This saves a great deal of time over the server having to actually request the WWW page from where it really sits on the Internet.

   Proxy servers are also called *application proxies*, since they require two ingredients, a *proxy server* and a *proxy client*. Suppose that a user, Alice in the local network wants to connect to a service on the Internet. Her request, together with her authentication ID, is first sent to the proxy server at the gateway/firewall using a TCP/IP application such as HTTP or FTP. The proxy server, acting in the role of the Internet server, assesses the request, and based on the local network security policy allows or denies Alice's wish. If approved the proxy server sends the data, as TCP pieces, to the proxy client, which contacts the actual Internet server. Then connections are established between the Internet server and the proxy client, which relays them to the proxy server for transfer to Alice. Hence Alice's outbound connections are always made to the proxy server, and the Internet's connections are always made with the proxy client. There is never a direct connection between Alice and the Internet server.

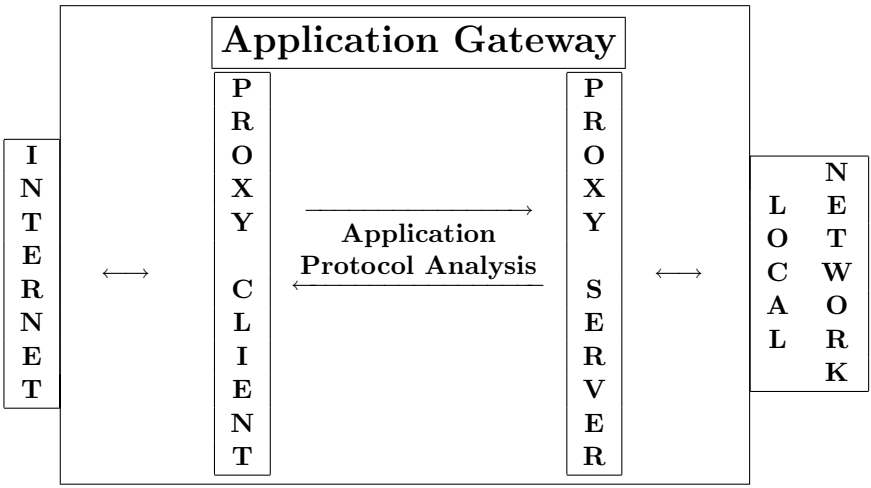
   Application gateways execute intricate record keeping and audit of traffic passing through them, as well as the traditional access restrictions required of any firewall. These firewalls may be used as NATs (see page 251). The reason is that the data exit the firewall after having been processed by an application, which usually conceals the source address of the data. Thus, the complexity of this type of firewall slows performance and reduces transparency. On the other hand, they are more secure than packet filters and render thorough audit records. Moreover, since they do not operate at the TCP/IP level, rather at the applications level, they need to screen only a small number of permissible applications.

   There are several more advantages to the use of application gateways. They recognize and administer high-level protocols such as HTTP and FTP. At the same time, application gateways present the semblance that they are connecting directly with external servers. They can also be employed *within* the local network to route services to other servers therein. Lastly, these gateways can be used for caching (as described above), and may be employed for user authentication.

   There are some disadvantages to application gateways such as the fact that the local network cannot run a network server on the firewall server. Also, if a new protocol has to pass through the gateway, a new proxy has to

be implemented, which causes inefficiencies. Moreover the complication of the process further reduces efficiency since modifications to configurations often have to be made.

Diagram 7.9 Application-Level Gateway/Firewall



4. **Circuit-Level Gateway**: These firewalls are very fast but have limited security checks. They are a type of proxy server where a virtual "circuit" is established between the local network and the proxy server, which receives requests, via the circuit, from Alice in the local network, and after changing the IP address delivers data to the Internet host. Any user outside the local network sees only the IP address of the proxy server, and when it receives a response, it is relayed back through the circuit to Alice.

   The security checks are restricted to the firewall's checking of permissions for Alice to send her message to the Internet, based on local security policy and whether the target Internet host has permission to receive Alice's data. If a connection is established, no further checks are done. Hence, circuit-level gateways are best used when Alice is a trusted local network user.

   These gateways transmit TCP connections, such as TELNET, wherein once the connection is established the firewall forwards data unrestricted. This makes circuit-level gateways more secure than static packet filters but less so than application gateways, since there is no applications-level checking. The circuit-level firewall security is essentially the decision pertaining to which connections will be permitted. Whereas the applications-level gateway operates (necessarily) at the applications level, the circuit-level

gateway functions at the session level, which explains the means by which the proxy sets a virtual circuit between Alice and the Internet host on a session-by-session basis.

Disadvantages to the circuit-level gateways are that they are restricted to TCP protocol access, they have limited ability to audit events, and they cannot interpret the application protocol being employed.

Now we turn to a circuit-level gateway implementation, which is considered to be an Internet standard firewall. First, we need to expand our understanding of several notions. On page 223, we (informally) defined the term (computer) host to mean those computers that provide services to other computers and to users on a network (such as the Internet). There is more to it. A host has associated with it a *host number*, and coupled with its *network number* forms its unique IP address (see page 224). The host number is that part of the IP address that determines which computer on the subnetwork is being addressed. (A *subnetwork* is a set of computer systems under the control of a single administrative domain that uses a specific network-access protocol. Forming subnets, *subnetting*, allows a network supervisor to segment the host part of an IP address into more than one *subnet*, which is interconnected but an independent portion of a network.) The Network number is that part of the IP address that designates the specific network to which the host belongs.

The term *IP reachability* is often used synonymously with *Internetworking*, which means any technology and associated mechanisms allowing communications across disparate computer networks. The following firewall has a basic function, which is to provide hosts on either side of it to communicate *without* direct IP reachability.

◆ The Socks Firewall/Proxy

SOCKSv5 is an IETF standard (see [75]) known as the *Authenticated Firewall Traversal* (AFT). SOCKS (derived from *SOCK*et*S*) is a networking proxy protocol allowing hosts on one side of the SOCKS server to access hosts on the other side of the SOCKS server without direct IP reachability. When used as a firewall, SOCKS redirects requests for connections from both sides of the SOCKS server; therefore it acts as a proxy server. The SOCKS protocol makes connection requests, establishes proxy circuits, relays data, and authenticates clients. This is accomplished as follows.

First an application client, Bob, sends the SOCKSv5 server a request for connection. If the request succeeds, Bob sends a list of authentication schemes that he can support. Then the SOCKSv5 server selects one, or if none of the methods intersects nontrivially with the network administrator's security policy, no connection is made with Bob. If a method is available, the SOCKSv5 server sends the choice to Bob, after which authentication is set up between Bob and the server.

Once authenticated, Bob sends his request to the SOCKSv5 server, and that request must contain the IP address of the application server with which Bob wishes to connect. Then the SOCKSv5 server evaluates Bob's request and either rejects it or accepts it. If it is accepted, then using the address sent by Bob the server connects to the specified application server and establishes a circuit between Bob and it, notifying Bob in the process. Once established, the circuit conveys data between Bob and the external server with the SOCKSv5 server screening each fragment of the data and relaying them between the two.

There is an earlier version of the protocol, SOCKSv4, but it had some issues that were not fully considered or were omitted altogether such as authentication, which SOCKv5 addresses completely. SOCKSv5 is used as a firewall, proxy server in VPNs, as well as a single communications protocol that authenticates users and establishes a communications channel. SOCKSv5 uses the same channel for both authentication and communication establishment, which has a higher degree of integrity guarantees built into the process. Moreover, it does so without direct IP reachability.

At this juncture, we need to define another concept. A *UDP* is a *User Datagram Protocol*, which is a communications protocol providing service for network communications that use IP. In fact, UDP is an alternative to TCP. UDP actually transfers what is called a "datagram" from one computer to another. A datagram is an independent data unit not requiring preprocessing in order to be transported from origin to target site on the network. Datagram is a term that has been replaced by the word *packet*, and either term is simply meant to refer to any message unit that the IP handles and the Internet transfers from one site to another. UDP differs from TCP in that it does not keep track of the order in which packets arrive at the target site. Thus, since UDP does not process the sequence of packets, time is saved, so UDP is used over TCP when there is only a small amount of data to process. Both TCP and UDP are transport layer mechanisms.

SOCKS may be configured to work with virtually any application, and it can set up not only TCP connections, but also UDP connections via a proxy. UDP capacity is another improvement of SOCKSv5 over its former version. This is a valuable addition since UDP provides a couple of services not available with TCP. One is an (optional) capacity, called a *checksum*, meaning a value related to the contents of a packet, sent with the packet, or stored to detect if the data have been altered during transmission. The other UDP feature (over TCP) is that it provides port numbers to help differentiate user requests. SOCKS uses *sockets* to record and track a given connection. (Think of a socket as one endpoint of an interprocess communication link between two entities on a network, and each entity establishes their own socket.)

5. **Kernel Proxy Firewall**: The fifth and latest generation of firewall is the

most intelligent. It has the capacity to do stateful inspection of network packets at *every* protocol layer of the network stack. It does so via the existence of a proxy within the kernel (core of the firewall) and relays packets on a session-by-session basis using a custom TCP/IP stack. In this fashion, each packet is screened at every layer from the physical to the application and back. Yet, despite this complexity, the filtering can be done efficiently. It accomplishes this via the kernel embodying the full set of available proxies. The kernel stands ready to proxy *any* protocol layer and execute full security checks.

The proxy server examines each incoming packet against a network security policy. If the packet passes this security point, it is checked against existing sessions. If the packet belongs to one, it is relayed to the proxy stack for that session. Each such proxy stack is dynamically built for each session. If the packet does not belong to an existing session, a new proxy stack is created and the packet is relayed to that stack for analysis.

Each of the dynamically created stacks analyze the network packet for those protocols determined by the specific session. Each packet may be discarded at a given layer if it does not meet security standards, or it may be modified at the pertinent protocol proxy. Furthermore, each proxy layer records state information for a given session.

If there are particular requested services, the proxy establishes an application layer extension. This renders the specific services, such as caching, without sacrificing efficiency. If no such additional services are needed, the packet does not go to the applications level.

There is also a *native network stack*, which stands alone without changes and has its own separate security policy allocated to it. Packets may be passed to the native stack after passing security checks/modifications, or the packets may be delivered to other computers, if so destined.

The new firewall architecture marries the need for some of the best possible security with exceptional performance. It still suffers from the failing of all firewalls as outlined on page 251, of course, but is a fantastic stride forward for network security.

There are hybrid systems employing combinations of the above firewalls using what is called a *bastion host*, which is a host that a local network designates as the *only* computer allowed to be accessed directly from the Internet and is used to shield the local network from security breaches. Usually, bastion hosts are stages for either application-level or circuit-level gateways. An example would be what is called a *screened subnet firewall* wherein a packet filter firewall is positioned on either side of the bastion host, thereby creating an isolated subnetwork. Another example is one configured to have both the packet filter and application gateway firewalls positioned on either side of the bastion host. Numerous such configurations are possible. The end goal is maximum security with minimum time.

7.5 Client–Server Model and Cookies

We have informally discussed client-server model applications throughout the text such as SSL in Section 7.3. Now we look at the general nature of such models, largely from the perspective of "cookies," which we will define and study in detail below. The so-called client-server model is one of the central features of Internetworking. It is time to settle on a general definition of these terms.

◆ Client-Server Model

A *client*, when considered as part of software, is a computer program (although we may use Alice and/or Bob in these roles) that relies on a server to perform some operations. (In the client-server model, the term "program" may be replaced by the term "computer" on which the program runs, sometimes called a "host computer," but this computer is typically employed for more tasks than just the client-server architecture.) Think of a client as a "requester of services." A *server* in this context is a computer program that provides access (for the client) to WWW formats and protocols (or to where HTML documents are stored). Think of a server as a "provider of services." The *client-server model* is a relationship between two programs in which one program, the client, makes a request of the other program, the server, which fulfils the request.

Client-Server Origin and Role: The client-server model was introduced in the 1980's as message-based modular software intended for use over a network. The motivation was to improve functionality, versatility, interoperability, and scalability over a single mainframe computer with time sharing. It is possible to configure the client-server architecture so that it operates on a single computer; in other words, the same machine serves the role of both client and server. However, the intention for, and full value of, the client-server model is realized over a network with physically separated client and server machines. This is because the client-server model was introduced largely to address the limitations of file-sharing architecture where the server downloaded files from the shared location to the desktop environment. This type of architecture was strained by a large number of online users and large volumes of data. The client-server architecture, in contrast, was a means by which the file server was replaced by the database server. By employing a *database management system* (DBMS), user enquiries could be answered directly, thereby reducing network traffic via an enquiry and response rather than total file transfer. The term *Intranet* means the employment of Internet technology for a given organization to implement client-server applications. To do this, a corporation, for example, would merely have to change its code on an HTTP server, as opposed to updating the client code on numerous desktop computers in its organization.

Among the simplest forms of servers are the file servers, whereas among the more advanced servers are the database servers. As mentioned earlier, use of a file server to transfer data over a network slows the process considerably. In the client-server model, the client sends a request to a server, which processes the

request on its own power to find the requested data rather than transferring all the information back to the client to find its own data.

Client-Server and HTTP: On page 225, we were introduced to HTTP and shown its application in protocol layers studied in Section 7.3. Here is how HTTP fits into the client-server model. When Alice opens her WWW *browser* (a client software program used for locating and viewing different types of Internet resources such as data on a WWW site), she indirectly makes use of HTTP. Each WWW server contains an HTTP *daemon* denoted by *HTTPD*, which is a continuously running program (by itself under the operating system) whose sole purpose is to (wait for and) handle requests that a given computer system receives periodically. The etymology of daemon is from the Greek meaning an attendant supernatural being, on a hierarchy between gods and humans (pronounced *dee-muhn*).

Now we need to look at another Internet notion. A *URL* is the acronym for *Uniform Resource Locator*, which is the global address associated with given data. The first part of the URL specifies which protocol to use, and the second part indicates the domain name. For example, *http://www.math.ucalgary.ca/~ramollin/* indicates that this is a WWW page and the HTTP protocol should be used. The second part is the domain name where my homepage is located.

Alice's browser is an HTTP client that makes requests to a server by, say, opening a WWW file via the typing in of a *Uniform Resource Locator* (URL). By so doing, her browser formulates an HTTP request and sends it to the IP address indicated by the URL. The HTTP daemon at the server site receives the request and sends back a response in the form of requested files. Unfortunately, HTTP is what is known as a *stateless protocol*, which means that each time Alice visits a WWW site (or even when she just clicks to another location from that site), the server sees this as her first visit. In other words, the server forgets all that has transpired after each request unless there is a means to somehow "stamp" Alice so that the server will remember the details of her last visit. The following is a mechanism for accomplishing this task.

◆ Cookies

The origin of the term "cookie" is uncertain, although its inventor, Netscape, claims it was a name chosen at random. Some claim that it was derived from a similar Unix operating system transaction called a "token." On MAC computers, the cookies are kept in a list called "magic cookie," whereas on IBM CPUs they are in a file called "cookies.txt."

What is a cookie and how does it fit into the client-server model? In simplest terms, a cookie is data (for future use) that are stored by a server on the client side of a client-server model. For instance, a cookie might record Alice's preferences when visiting, QQQ.com. The cookie is a means by which the server can store its own data about Alice on Alice's own computer.

Analogy: An analogy is a voucher Alice gets when she brings her shoes to a cobbler, Corbett, for example. If she returns for her shoes without that voucher, Corbett will not be able to locate her shoes. To him, she could be

a new customer. Alice's voucher is necessary for Corbett to maintain record keeping, and it establishes a formal relationship (which we will call a *state*) between him and Alice.

Cookies and HTTP: In Internet terms, a server, when returning an HTTP object to Alice, includes a cookie that has a description of the range of URLs for which that cookie is valid. Any future HTTP requests made by Alice that fall in that range will include the current value of the cookie from Alice sent back to the server. This means that she can shop online and store information about the currently selected items, and it frees Alice from retyping her user ID for each visit. The sites at which she shops can store preferences on her computer and have Alice supply those preferences every time she visits that site. For instance, the QQQ.com server provides the cookie to Alice's browser, which stores it in its memory as a text file. Each time her browser sends a request to QQQ.com (when she types in its URL for example), the cookie is sent back to the server.

Types of Cookies: There are different types of cookies. For instance, a *session cookie* (or *transient cookie*) is one that is erased when Alice closes her browser, because the session cookie is stored in temporary memory and discarded after the browser is closed. These transient cookies do not obtain information from Alice's computer. Rather, they store data in a session ID format, which does not explicitly identify Alice. Another type of cookie is the *persistent cookie* (also called *permanent* or *stored* cookie), which is a cookie set with an expiration date and is stored on Alice's hard drive until it expires (or else Alice, herself, deletes it). (A *hard disk*, also called a *disk drive*, is part of a unit that stores [and provides efficient access to] large blocks of data on one or more electromagnetically charged surfaces.) Persistent cookies gather information about Alice, including her WWW surfing behaviour or her preferences at QQQ.com. The QQQ.com server may use this information to present Alice with a customized welcome page with, say, "Hello Alice" the next time she visits.

Alice's browser automatically updates her cookies every time she revisits a site, since once the browser is closed, the cookies are resaved to disk.

Effect of Cookies: In the final analysis, a cookie is simply a piece of text, not a program, and only Alice's browser can store cookies on her hard drive, if it is a persistent cookie. The data are stored in a special file called a cookie list and is done without the knowledge or consent of Alice. However, it cannot be used for, say, a virus, so it is harmless in that regard. Moreover, the number of cookies allowed for storage on Alice's hard drive is also restricted. Most browsers conform to RFC 2109 (see [76] and page 240), which puts a limitation of 300 cookies that may be stored on a given hard drive (with a 4096 byte-per-cookie maximum). This involves a limit of twenty cookies per WWW site, so if fifteen sites maximize the cookies on Alice's hard drive, then the next time a cookie is to be set, Alice's browser will discard her least-used cookie to free space for the new cookie.

When Alice returns to QQQ.com, her browser will automatically and, again without her knowledge or consent, transmit the cookie containing her personal data to QQQ.com's server.

Cookie Ingredients: Cookies transport between server and client as an HTTP header, and the formal specifics of this header as defined in RFC 2109. There are six parameters that can be assigned to a cookie. The first two are mandatory and are set by pairing them together. The others (set optionally), configured manually or automatically, typically are separated by semicolons.

1. **Name**: This is any alphanumeric value (excluding semicolons, commas, and white space) used to identify the cookie.

2. **Value**: This cookie value may be any scalar.

3. **Expiration Date**: This determines the valid lifetime of the cookie and, if not explicitly set, defaults to the end of the session as long as Alice's browser is open.

4. **Path**: This sets the subset of URL paths on a domain for which the cookie is valid. If a path is not specified, the default is the path of the document that created the cookie.

5. **Domain**: This is the textual equivalent of a numerical IP address. When searching a cookie list, a comparison is made between the *tail* of the valid host domain name (such as QQQ.com) and the tail of the cookies on the list. For instance, it might be shopping.QQQ.com, which indeed satisfies the *tail matching* for the domain QQQ.com. Because of this tailmatching, no domain is allowed to set a cookie with fewer than two dots, in order to distinguish among tails such as those containing *.com*, *.ca*, *.gov*, and so on. Thus, for instance, QQQ.com would not be an allowed cookie on the list. Moreover, the server setting the cookie must be a member of that domain. For instance, WWW.QQQ.com cannot set a cookie for the domain WWW.RRR.com, since the security breaches would be severe.

6. **Secure Label**: If this label is set to TRUE, then the cookie may be sent only over a secure channel, typically HTTPS (see page 243). The default is FALSE, since most WWW sites do not need secure connections.

Basically cookies are pieces of textual data generated by a WWW server for storage on a client's computer for future access. Cookies are embedded in HTML information that flows between the client browser and the server. Most often both the storage of, and access to, cookies goes unnoticed by the client. However, any client, concerned about privacy issues can set his or her computer to notify of any attempt to set a cookie and will ask permission. Of course, this may become a headache since there will be a lot of "alerts." The crucial issue is for the client to be "aware" of the issues, which this section addresses. Cookies cannot damage your computer or give out private data on you without your giving it out at a WWW site in the first place. The bottom line is that cookies were meant as a mechanism to make it easier for you to access your favourite WWW sites by storing information, so you do not have to login each time you visit, a process that was impossible before the advent of cookies due to the stateless nature of HTTP.

Chapter 8

Leading-Edge Applications

8.1 Login and Network Security

This chapter is adapted from [64] to suit this text. The following describes mechanisms for securely logging in to a computer.

▼ Login Security

To *login* (also called *signing in*) to a computer, we must provide a passphrase, which may be as simple as a single word, called a *password*, or a sequence of words used to identify us uniquely for secure access to the system. The encrypted passphrase will be accompanied by our plaintext *username ID*. A user ID, authenticated by its associated passphrase, determines the privileges allotted to the user, which may vary from personal e-mail access to *superuser* status, where actions may be executed that are protected by the operating system.

If we are trying to login from home, or a hotel when on a trip, to gain access to a computer at work, for instance, this is called *remote login*. In this case, passwords may travel over unsecured channels, making them susceptible to eavesdropping by Eve or interception by Mallory. Mechanisms exist for dealing with these situations. E-mail security via PGP and S/MIME were described in Chapter 7.2. Secure session-based communication via SSL was explored in Section 7.3. Now we delve further into password protection.

On page 168, we described the use of one-way functions in the role of password security. Also, we have already been introduced to the concept of a "salted" message (see Footnote 2.2 on page 125).

▼ Why Use Salt?

The purposes behind salting a passphrase are threefold.

1. Eliminating the visibility of duplicate passphrases on a user's file.
2. Increasing the bitlength of the passphrase to thwart password guessing.

3. Helping thwart attacks such as the dictionary attack (see see page 128).

▼ Proactive Password Selection

Since the average person is notoriously lazy about choosing proper passwords, instead selecting easy-to-remember words and neglecting security, there needs to be a mechanism for ensuring that user-chosen passwords are acceptable. This is where a proactive password checker comes into play. This built-in checker will determine if a user-selected password is acceptable and reject it if not, prompting the user to try again. System enforcement may contain some of the following criteria.

Passphrase Selection Criteria

Parts 1–4 below refer to the criteria for a proactive checker itself, whereas the remainder are more for a given user to consider when choosing a passphrase.

1. All passphrases must have at least ten symbols.
2. There must be at least three of: lower case letters; upper case letters; numeric; and characters such as !,#,),&,*,⁕, and so on.
3. No symbols should be repeated.
4. No actual words should be used.
5. No personal data such as birthdays or telephone numbers should be used.
6. Memorize the passphrase. Never write it down and do not store it on your computer as a file.

If properly implemented, the above criteria ensure that a brute-force attack is made less likely to succeed. There exist methods for creating effective and efficient passphrase checkers that do not require lots of space and time as would, say, a list of stored "unacceptable" phrases.

Attacks on Passwords

The following list encompasses some current password attacks.

1. *Password sniffing*: This is an attack in which Eve listens to data traffic that includes secret passwords in order to capture and use them at a later time.
2. The *birthday attack*: This attack is described on page 128.
3. *Spoofing*: This type of attack is described on page 252.
4. The *dictionary attack*: This attack is described on page 128.
5. *Password-cracking software* is available over the Internet.

6. *Social engineering attacks*: This refers to a group of attacks that exploit human weakness or gullibility. These techniques consist of employing nondigital mechanisms to gain digital data from the victim. One of the most common is for the criminal to masquerade as a bank official to get the victim's PIN usually based on a claim that it is required to fix something with the account. These attacks, therefore, require that the victim's trust is obtained so that the victim will disclose information to the criminal.

7. *Packet sniffers*: These are programs that monitor, capture, or analyze network traffic, or databases. For instance, a database might be scrutinized by Mallory to detect passwords. If he is successful at gaining access to a system-level password, Mallory can create a new account that can be used at will as a *back door* to get into the network and its resources, including the altering of core system files, such as the password for the system administrator account, the list of server services and permissions, and the login information for other machines, containing critically confidential information. This could create chaos since the daily workings of the network are up for grabs, and Mallory's network packet sniffer can be modified to include his information or change system information in a network packet, forcing network connections to behave erratically, at best.

Item 7 above also has legitimate uses as follows. A *snoop server* is a server that uses a packet sniffer to capture network traffic for analysis. For example, an employer might want to use a snoop server to monitor the WWW sites visited by the employees. Snoop servers typically operate in *promiscuous mode*, which is a networking mode allowing a network device (a unit of removable hardware) to access all packets, irrespective of their target addresses. In this manner, a snoop server, for instance, can seize any data packet, copy, and store it to a file for later analysis and reporting. For example, the Sun operating system, *Solaris*, has a feature called the *snoop command*, permitting administrators to capture packets with an attendant packet description or summary. However, this also permits intruders (running the Solaris OS) to scrutinize the traffic over the network.

In general, a promiscuous mode is used for legitimate monitoring of network activity. This might involve the performance of diagnostic testing to try to resolve such problems as bottlenecks in the flow of traffic or general troubleshooting to identify a variety of performance problems. Modern sniffers can be configured to automatically alert administrators when a performance problem is triggered by some preset standard that they set as a local bound.

For the following discussion, we should expand our understanding of the notion of a hard drive, the basic definition of which was given on page 261. We extend it here to get a better idea of how it functions. A hard disk is essentially a collection of stacked disks, each storing data electromagnetically recorded in concentric circles called *tracks*. Two *heads*, one located on each side of a disk, read or write the information on these tracks as the disk spins. The spin speed is

anywhere from 4500 to 7200 rpms. Think of the comparison with a phonograph record and its player having a phonograph arm ("head") to "read" the music. Now we look at packet sniffers more closely.

A packet sniffer can be configured to store copies of packets in memory or hard drive, which might be done via temporary storage in a buffer for later analysis. Employers might want to monitor any number of employee activities such as who visits the employee's site; what an employee downloads, including streaming audio and video; contents of incoming and outgoing e-mail messages; which sites the employee visits; and the contents of what the employee views at a given site. The amount of traffic scanned by a given packet sniffer will depend upon the location of the computer in the network. If it is located in a relatively secluded area of the network, then the sniffer will be able to scan only a tiny portion of traffic over the network, but if it is the principal domain server, for instance, the packet sniffer will be able to scan virtually all of the traffic.

The above being said, Mallory still likes packet sniffers, since if successful, he can use them to seize passwords from data packets traversing the network and wreak havoc as described above. One method of thwarting Mallory is to encipher the headers of packets using SSL in browser-based traffic (see Section 7.3).

Before going into a deeper discussion of ethernet and promiscuous mode, we look at the organization that developed the standards.

IEEE

IEEE, pronounced *I-Triple E*, is the *Institute of Electrical and Electronics Engineers* Incorporated. The AIEE, *American Institute of Electrical Engineers*, which was founded in 1884, merged with the IRE, Institute of Radio Engineers, in 1963 to form IEEE. The primary function of IEEE, for our interest, is the development of standards for communications security, the most famous of which are the IEEE 802 standards for (LAN)s *Local Area Networks* and WANs, *Wide Area Networks*. A LAN is a collection of computers and their attendant mechanisms sharing a common communications channel or wireless linkage and (usually) a shared server. The common server has applications and data storage, which may be accessed by the LAN users who may vary in number from a couple to several thousand. A WAN differs from a LAN in that it is a geographically more dispersed network, which usually includes shared user networks. In size between a LAN and a WAN is a *MAN* or *Metropolitan Area Network*, typically meaning the interconnection of networks in a city into a single large network. A MAN, of course, provides a more efficient connection to a WAN. For more information on IEEE and its standards, visit *http://www.ieee.org/portal/index.jsp*.

Ethernet and Promiscuous Mode

Ethernet (as specified in IEEE 802.3) is the most commonly employed *Local Area Network* (LAN). Ethernet evolved from a framework called *Alohanet*, named for the *Palo Alto Research Center Aloha Network*, which was developed into Ethernet by XEROX then further expanded later by DEC, Intel, and XEROX. There exist Ethernet configurations that provide transmission speeds up

to 10 billion bits per second, called *Ten-Gigabit Ethernet*, which is specified in IEEE 802.3a. The future of all interconnections of LANs, WANs, and MANs is generally predicted to be via the Ten-Gigabit Ethernet.

Now that we know the basics of Ethernet, we describe the use of packet sniffers in this context. Ethernet was designed to filter out all data traffic not belonging to it. When a packet sniffer is installed in Ethernet hardware, that filter is turned off and the hardware goes into promiscuous mode. Thus, if Alice and Bob are communicating over an Ethernet channel with a packet sniffer attached, Mallory can read all the traffic between them. Packet sniffers on an Ethernet consist of the following parts.

Packet Sniffer Components

1. **Hardware**: In promiscuous mode, every packet is received and read by a *network adapter* (sometimes called a *network interface*) which is a physical device such as a card (and its software driver); which connects a host computer to network traffic, allowing the host to send and receive packets.

2. **Capture Driver**: This type of driver captures the network traffic and stores it to a buffer, for instance. A *driver*, in general, is a program that controls a particular device, such as a printer or disk drive. Either the driver will come with the operating system or will have to be loaded when the device is added. Think of a driver as a translator between the device and the programs using the device.

   A *device driver* is a program that controls a specific device such as a printer. Thus, we may (informally) think of a capture driver as a program that controls the capture of information packets for the packet sniffer.

3. **Buffer**: The captured data from the network are stored in a buffer until they can be analyzed.

4. **Protocol Analyser**: This aspect of the packet sniffer strips off any encoding and analyzes the data (see Appendix G).

The antithesis of promiscuous mode is *nonpromiscuous mode* wherein packets are scanned and passed on if those data packets are not theirs. Only the target site device receives and reads the data in this mode.

Now we return to the issue of login security. We have addressed the issue of password selection and checking, remote logins, and attacks that may obtain passwords. We turn to a modern secure method for password storage.

◆ Security Tokens

A *security token* is a special device (a physical object usually ranging in size from that of a housekey to that of a credit card) that a user carries for the purpose of authorized access to a network. For example, the device may be embedded in a *key fob*, which has the physical appearance of a key but has built-in authentication mechanisms consisting of the following:

1. The user's PIN, authenticating, say Alice, as the fob's owner.

2. A login ID, which is displayed after Alice correctly enters her PIN, allowing her to login to the network.

Token Applications

1. A token may be embedded in a *smart card*, which has the physical appearance of a credit card but has the above authorization mechanisms embedded. (We will study smart cards in detail in Section 8.3.) The login ID is not static and may actually change every few minutes for security reasons, so if a security token is lost, and Mallory finds it, he cannot access the network without Alice's PIN. Furthermore, an additional security measure against the possibility that Mallory might launch a brute-force attack to recover Alice's PIN is that the device would be disabled after a small number of attempts to enter the PIN, say, three or four. Hence, security tokens provide one of the foremost, modern, practical methods for the storing of secret keys.

2. Since employees of, say, a corporation need to insert their security token into their computers for network access, the corporate administrators must guard against human laziness. For instance, a user, such as Alice, might decide to leave her office to get a coffee and not remove the token from her computer, which is a security risk. To guard against this, the employers may require that the token is needed for access to her office, the coffee machine, the filing cabinets, the department office, the rest room, and so on. In this fashion, the token cannot be left unattended in any reasonable scenario. This makes such a system foolproof but not idiot-proof. (An adage is that genius knows its limitations, but stupidity is unbounded.)

We will learn about other security options such as biometrics in Section 8.4. For now, we turn to a remote login protocol that is considered to be the industry standard.

◆ The Secure Shell Remote Login Protocol (SSH)

We describe the latest version, SSH2, which corrects failings of the original, including susceptibility to certain attacks. Although the protocols in SSH2 described below may have many differing formats, we do not delve into that detail. Instead, suitable references will be provided for the interested reader. We concentrate upon the description of the main protocols and focus upon SSH2 as a development that is on an approach to becoming the new standard for remote login.

Before going into further explanation, we need to discuss the system upon which SSH is based. *Unix* (pronounced *you-niks*) originated in 1969 at AT&T Bell Labs to provide an interactive time-sharing system. In 1974, Unix attained

the status of the first operating system to be written in the C programming language. Being a nonproprietary operating system, it evolved as freeware and eventually became the first standard operating system that could be openly developed by virtually anybody. We may rightfully view both the client-server model and Unix as vital developments in the evolution of the Internet, with a focus toward computing networks and away from independent computers.

What is SSH?

Secure Shell or SSH (sometimes called *Secure Socket Shell* — not be be confused with SSL — see Section 7.3) is essentially a Unix-based command interface using PKC-oriented, secure remote login protocols. It allows a user to execute commands on a remote computer, as well as securely move files from one host to another. It provides strong authentication and secure communication over an insecure channel. SSH was designed to replace insecure applications such as Telnet and FTP (see page 224).

Basically, How Does SSH work?

The SSH mode of operation is quite simple on the surface. The host computer first authenticates itself to the client, establishing a unilateral server-to-client secure channel. Then a user, Alice, say, on a client computer employing unilateral public-key and/or password-based protocols authenticates herself to the server. Once the link is secure, not only can files between hosts be transported, but also other TCP/IP connections may be forwarded over that secure link. All algorithms used to ensure security are negotiated, so if some algorithm is cryptanalyzed, it is a simple matter to eliminate it and switch to another in the cipher suite.

The following is a detailed description of SSH2, starting with an overview of SSH in general.

◆ SSH Protocol Architecture

We will assume that Alice is the user on the client computer and she wishes to establish secure communications with the (remote) host computer.

▼ Overview of SSH Protocols

1. **Transport Layer Protocol**: This protocol provides strong host authentication, confidentiality via strong encryption, and integrity protection from the server to the client computer. This layer also thwarts the man-in-the-middle attack. Moreover, it optionally supports compression. Although there are other possible data streams over which this transport layer may run, we assume that it does so over the canonical one, TCP/IP. The other layers of the SSH protocol run on top of the secure tunnel provided by the transport layer — see pages 241–242.

2. **User Authentication Protocol**: This protocol runs over the transport layer protocol for the purpose of authenticating Alice to the server. The

DSA cipher is used for authentication (see page 187). Once this protocol is completed, there is a mutually authenticated secure channel between Alice and the host.

3. **Connection Protocol**: This protocol runs over the encrypted tunnel estabilised above. It multiplexes (where *multiplexing* means the use of a transmission channel to carry two or more signals at the same time) that tunnel into numerous logical channels that may be used for a rich variety of application-support services, including remote program execution, signal propagation, and connection forwarding.

▼ SSH Protocols in Detail

SSH Transport Layer: The purpose of this layer is to ensure secure communication between Alice, as the client user, and the remote server, as the host. Once Alice contacts the server, key data must be exchanged in order to construct the tunnel. With SSH2, it is mandated that DSS be used (see page 187). The host sends its public key, called the *host key* e_S, as identification. In order for Alice to be certain that she is communicating with the correct server, she must have prior knowledge of e_S, for which two trust models are available.

Key Exchange Protocol

It is mandated in [95]–[96] that the Diffie-Hellman key-exchange protocol (see page 167) be used to arrive at key agreement as follows.

We assume that p is a large safe prime; α is a primitive root modulo p; h is a hash cryptographic hash function; and that identification data have been exchanged in advance such as both Alice's and the server's ID, I_A and I_S, as well as Alice's and the server's protocol versions V_A and V_S, respectively.

1. Alice generates a random number r with $1 < r < p - 1$, then she calculates $c_A \equiv \alpha^r \pmod{p}$, which she sends to the server.

2. The server generates a random number s with $1 < s < p - 1$ and computes each of the following:

   (a) $c_S \equiv \alpha^s \pmod{p}$.
   (b) $K \equiv c_A^s \pmod{p}$.
   (c) $H_S = h(V_A, V_S, I_A, I_S, e_S, c_A, c_S, K)$.
   (d) $D_S(H_S)$, the server's digital signature.

   Then the server sends $D_S(H_S)$ to Alice.

3. Alice certifies e_S as described in the above discussion preceding the key exchange protocol. Once done, she computes

$$K \equiv c_S^r \pmod{p}$$

   and

$$H_S = h(V_A, V_S, I_A, I_S, e_S, c_A, c_S, K).$$

She may then verify the server's signature $D_S(H_S)$. If this is valid, then she accepts the key K as the shared secret session key, which may now be used for encrypting communication between Alice and the server.

Upon completing construction of the secure tunnel via the transport mode described above, it is Alice's turn to authenticate herself to the server.

Authentication

First, the server informs Alice of the various authentication mechanisms supported. She may choose any of these methods. For instance, the server might send Alice a challenge that she signs with her private PKC key, allowing the server to use her public PKC key to authenticate her.

Once the authentication of Alice has occurred, the server will typically log her into the remote computer and provide her with a shell. Thereafter all communications with her remote shell will be automatically encrypted. It should be noted, however, that the SSH shell forbids login to an insecure FTP server, for instance. The remote host is required to posses SSH-enabled software. There is a mechanism called SFTP, which is an FTP replacement that runs over an SSH tunnel.

There is a mechanism for avoiding the use of SFTP altogether but which supports the SSH SFTP protocol. It is called OpenSSH, which is a version of SSH available over the Internet, supported by the *Open BSD Project*; see *http://www.openbsd.org/*. It contains the SSH program, which replaces *rlogin* (remote login) and *telnet*. Rlogin is a UNIX command allowing a user to login to other UNIX hosts on a network and interact as if physically present at the remote host. Rlogin is similar to the better-known telnet command. However, both are insecure. The OpenSSH suite replaces not only these two UNIX utilities, but also others such as *ssh-add*, *ssh-keygen*, and so on, as well as the sftp-server. Sftp is an interactive file transfer program that operates over an encrypted SSH tunnel, capable of using many features of SSH. In summary, we simply use SFTP under the SSH shell supported by OpenSSH.

Given the secure tunnel provided by the transport layer, the authentication methods do not require the level of security that would be required without the channel. Once the transport tunnel and the user authentication with key exchange are completed, Alice and the server can create a new channel. This is accomplished as follows.

Connection

When the above protocols are completed, Alice and the server may negotiate the characteristics of each new channel to allow multiplexing the single connection between Alice and the remote host. Each channel is assigned a different number for both ends, Alice and the host, according to [50] and [97]. When Alice wants to open a new channel, she transmits this channel number along with her request. The host stores this information for the purpose of orienting communications to that specific channel, which allows differing sessions to be unaffected and prevents the main SSH connection from being disrupted. This

is required since SSH sends different channels over a common secure tunnel. There is a mechanism for these channels, called *flow control*, which ensures the transmission of data in an ordered fashion; for example, the data will not be sent to Alice, say, until she has already been alerted to the fact that a channel is open for the message transfer.

▼ Analysis

SSH protects against any attempts by intervening hosts to intercept plaintext passwords or general manipulation of data. However, certain generic implementations of SSH are insecure (see [48]). SSH2 supports PGP keys as well as the SOCKS firewall (see Section 7.4). Last, we look at how SSH differs from SSL (see Section 7.3). With SSL, authentication is optional, whereas it is mandatory in SSH. A totally anonymous SSL, discussed on page 249, is susceptible to the man-in-the-middle attack, whereas the SSH protocol has built-in mechanisms to thwart such attacks via automatically maintaining, checking, and updating public host keys. Also, it is more unwieldy to use the certificate management necessary via a PKI in SSL, whereas the SSH keys are a relatively simple matter to handle. Moreover, SSH has a wide range of client-authentication options, whereas with SSL only PKC is an option. Last, SSH has many more features implicit in its multiplexing via the connection protocol than does SSL at any level.

Exercises

8.1. Suppose we have a large codebook that consists of short words with corresponding six-digit numbers. Devise a means of choosing passphrases from this list based on the roll of dice.

8.2. Assume that you have a password p and each time you logon to a WWW site the host uses a one-way function f to calculate $f(p)$ and compares this with a stored value. Devise a means using only this function that will prompt you to change your password after, say, 100 logins.

8.3. The SSH protocol presented on pages 268–272, has a significant additional feature called *port-forwarding*, which means that either Alice or the server can bind a socket to a collection of specified ports. In practice, what this means is that when Alice, say, connects to one of these ports, the call is relayed to the other end of this particular SSH call, from which another call is made to some other predetermined port. Effectively, this is an SSH built-in tunnelling mechanism (see page 270 for details). Discuss the pros and cons of such tunnels.

(*Hint: Consider setting up such tunnels to avoid firewalls.*)

8.2 Viruses and Other Infections

We begin with a description of the most common computer infection known to the general public.

◆ Viruses

A *virus* is a hidden, and typically malicious, program that "infects" your computer by copying itself into and becoming part of another program called the *host program*, without which the virus cannot run. The effect varies from the merely annoying to the completely destructive, such as deletion of files, erasing of programs, or even erasing of an entire hard drive. On the other hand, they may just flash the message "infected" without end, for instance. We will learn about the various types of viruses and how they work in this section, and later we will discuss other types of infections that do not need such a host program to infect a computer. Moreover, although most viruses are written with a computer in mind, such as those that will attack only a PC but not a Macintosh, there are platform-independent viruses (see macro viruses below).

The most common vehicle for infection today is the Internet, and many viruses arrive by e-mail. Downloading files from the Internet or opening an e-mail file may trigger a virus. However, even the exchange of infected disks is a mechanism for spreading infection.

The term "virus" from the biological realm is used here since the computer virus acts in a similar manner to an infectious disease. A biological virus is a string of nucleic acid (DNA or RNA) that may infect a living cell by assuming control of it and instructing it to replicate the virus many times over. Similarly, computer viruses attach themselves, replicate themselves, and spread in a manner akin to biological viruses. They may take control of the computer's OS, for instance, and whenever a new piece of software is encountered, they copy themselves to that new program, thereby infecting it. With the Internet, where you may access resources running on other computers, there is a rich culture for the spread of this kind of infection.

Once a program is infected with a virus program, it becomes the host. The virus program runs secretly when the host program is run, since it stays hidden in the legitimate program, remaining dormant until the infected program is run (or, as we will see below, until an infected data file is accessed). A virus may be embedded in an executable program; then, once run, the virus code is executed first, then the original program code. The following are the aspects of a computer that a virus attacks.

Virus Targets

Viruses may infect any of the following:

1. **Executable Program Files**: An *executable program* is a set of instructions that can be input to the memory of a computer and executed. In other words, it is a program that may be run as a self-contained procedure, which consists of a main program and, possibly, one or more subprograms.

Usually, the name of such a program is all that is required to run it, merely the typing in of the program name and requesting that the computer run it.

2. **File Directories**: A computer's file-directory system keeps track of the location of data files, and without them the computer will not function.

3. **Macros**: First we define a *macro* as a collection of instructions stored in an executable form, usually written to automate a few steps. Macros may be application specific, such as a word-processing macro that executes certain steps within that program, or general purpose, such as a keyboard macro that types in a user's login name when a specific short sequence of keys is pressed on the keyboard.

   Today, a virus program can be written so that, for instance, it may attach itself to a macro and is launched whenever the macro is run. When we discuss "macro viruses" later, we will see that Microsoft Word (MS-Word) documents are virtually always the target since they contain programs, the macro language, that are automatically executed when one of these "data" files is opened.

4. **System Sector**: First we define a *sector* as one of the areas (or "pie slices") into which the disk is segmented. This division of the disk into pie slices is the method of organizing it for access of data to the read-write heads of the disk drive. Moreover, the disk is further divided into concentric circles, so that a given area can be located via the intersection of a given sector and the concentric track passing through it. There are further subdivisions of the tracks into what are called *clusters*, which are the storage units (usually 256 or 512 byte lengths, which are minimal in terms of allowing the unit to be addressable).

   The system sector refers to special areas on the computer's hard drive containing programs that are executed when the computer is booted. These are not files but rather small segments of the hard disk that the hardware reads as a single unit. The system sector is required for the normal functioning of the computer, even though they are invisible to normal programs. Sometimes this is called the *boot sector*. (Note that to *boot* a computer, also called *booting up*, is the action of loading an OS into the computer's main memory, *random access memory* [RAM], meaning the memory space that is basically used to store dynamic data — that data that change during execution of a program. On a large computer or a mainframe, booting is sometimes called *initial program load* [IPL]. To *reboot* is to reload or, in the case of larger machines, to *re-IPL*.)

How Viruses Work: When an infected program is run, the first action is to invoke the virus program and run it, since this is the first instruction line of the controlling program. The second instruction is for the virus program to check to see if the program it is about to infect has already been infected or not.

The mechanism by which this is accomplished is a message called a *v-marker* or *virus marker*, which the virus program places in the legitimate program. If the virus program encounters a v-marker, it does not replicate there since it knows that the program is already infected. Then it seeks uninfected executable files (those without v-markers) and infects them. If a virus begins by infecting a program, then each time that program is run, it seeks out uninfected files. Often the virus is embedded in a game or utility.

Once a virus program determines that there are no more files to infect, it may begin to damage the computer and its data. The virus program may corrupt program or data files so that they work either erratically or not at all. They might destroy all the files on the computer, alter the system files needed to reboot, or perform any other of a number of damaging actions.

Now we look at the evolution of a given virus from its initial infection to its end goal attacks.

Stages of a Virus

1. **Infection Stage**: The virus infects some area of the computer as discussed earlier. Some viruses then remain dormant until a "trigger" sets it in motion while others go to stage 2 immediately.

2. **Replication Stage**: In this stage, the virus reproduces itself onto other programs using the initial infected program to do so. Then each new infected program will undergo the same replication stage.

3. **Activation Stage**: The virus is triggered to perform its end goal. The trigger may be any number of events from the time of day, the date, or any other event such as the number of times the program is executed.

4. **Execution Stage**: The virus performs its end goal, which may range from erasure of the computer's hard drive to the merely annoying, including simply slowing down the performance of the computer.

Types of Viruses

1. **Boot- (System-) Sector Virus**: First we define the *master boot record* (MBR) also called the *partition sector*, as the first sector of a computer's hard disk, which indicates the location of the OS and the methodology for finding it. This is necessary for the booting of the OS into the computer's RAM. The MBR is also called the *master partition table* since it contains a table that houses data on each of the hard disk's partitions. The MBR also contains a program whose function is to read the boot sector record of that partition that contains the OS to be booted into RAM (see page 274).

   These kinds of viruses infect the MBR. When a computer is rebooted, the virus spreads its infection.

2. **File Virus**: File-infecting viruses attach themselves to executable program files. Once the program is loaded, the infected program is executed and seeks out uninfected executable files.

3. **Memory-Resident Virus**: This kind of virus stays in memory after it executes and after its host program is terminated, whereas a *nonmemory-resident* virus activates only when an infected program executes.

4. **Polymorphic Virus**: This is a particularly nasty virus that mutates every time it infects a new program. Therefore, detection of this type of infection is difficult since it leaves no unique trail ("signature") to follow.

5. **Stealth Virus**: This kind of virus is specifically designed to disguise its existence from virus-scanning software. For instance, if a stealth virus has infected the MBR, then its function might be to interrupt a virus-scanning software's request to examine the MBR and then transmitting a (false) copy of the original *uninfected* MBR.

Examples

An example of a virus that is a combination of some of the above is the following.

Multipartite Virus: These viruses infect in one format type, then transform into another. For instance, one might begin as a boot-system virus and then move to become an attack on executable files.

An example of a memory-resident virus is the following modern-day virus that takes advantage of features found in data-processing software.

Macro Virus: This type of virus is one of the most recent and, unlike the others, is platform independent. In other words, it will infect those using a Macintosh computer as well as those using Microsoft Windows, for instance. The reason is that these viruses are programs written to attach themselves to macros used in modern-day data-processing systems, such as MS-Word, MS-Excel, and AmiPro. These macro languages fit the three conditions that make them ripe for macro infection, namely, they (1) assign specific macro programs to specific files; (2) copy macro programs from one file to another; and (3) pass control to some macro program without the user's explicit permission, that is, they are automatic. The aforementioned word-processing systems were designed to be automatic, and as such, if an infected document is opened, the viral macro will replicate itself into the computer's startup files. From then on, the machine is infected and the macro virus will reside on the computer until eradicated. Any document on the machine that uses the infected application can then become infected. If the machine is on a network, the infection will likely spread to other machines on the network. If a disk with the infection is shared, then the virus will spread to the recipient's machine. Today, macros are deemed to make up two-thirds of all computer viruses, according to experts.

The typical agent for spreading macro viruses is via e-mail. The most notorious macro virus was *Melissa*, launched in 1999. Melissa was distributed by e-mail and applied to MS-Word documents. Moreover, those recipients who opened the documents found that the first fifty people in their address books also received the virus. This was so effective that on Friday, March 26, 1999, Microsoft Corporation was forced to disable incoming e-mail. Melissa operated by incorporating a message that told the recipient that an important (secret) message was contained in the attachment. Once opened, the infected file was read to the global macro file. Then the virus employed the *visual basic language* (a graphical programming language introduced by Microsoft in 1990 and used for developing GUI Windows applications) to read the first fifty names in the address book and send them all the virus. Macro viruses are memory resident since they are active not only when the infected documents are opening or closing, but also for the entire time the system is running.

Melissa suggests that e-mail is becoming the medium of choice for attackers, and this is indeed the case.

E-Mail Virus: Malicious software employing e-mail is becoming more common with each passing day. Melissa was just the beginning. More powerful versions of e-mail viruses have emerged wherein the virus is spread to all the e-mail addresses within the address book of the infected host. Thus, the rapid deployment of e-mail viruses is now a major threat.

On Thursday, May 4, 2000, a new e-mail virus called the *"I Love You"* virus, also called the *love bug*, spread itself around the world in a matter of hours. Its name is derived from the fact that it contained a message to check the attached "love letter," which was a file in Visual Basic containing the virus. If the e-mail was deleted without opening the attachment, then the computer was safe. However, if opened, the computer was infected and the virus was distributed via e-mail employing MS-Outlook's address book. This was an advance in the degree of malevolence over Melissa since the latter sent only to the first fifty addresses, whereas the former sent to everyone in the address book. The love bug was much more destructive than Melissa since it copied itself into two vital system directories and added triggers in the Windows registry. This meant that every time an infected computer rebooted, the love bug was executing. It infected data files by overwriting them using Visual Basic and deleting the original file. Typically, files associated with WWW development and multimedia files were extinguished, such as those of type MP3 (music) and JPG (images). An example, to illustrate the magnitude of the losses, was reported by the Norwegian photo agency Scanpix, which lost over 6000 of its photos and was able to recover fewer than twenty-five percent of them. The love bug affected versions of the Windows and NT operating systems only, so Macintosh and Unix platforms were safe. Yet this was enough to cause billions of dollars in damages around the globe.

In October 2002, the *Bugbear* virus infected Windows platforms through a hole in the security system in MS-Outlook, MS-Outlook Express, and Internet Explorer. Once a machine was infected, the virus copied all passwords and

credit card numbers typed by a user, then it sent the information to numerous e-mail addresses. It was estimated that in the first week it sent roughly 320,000 e-mail messages. In 2003, the virus appeared in a more virulent strain called *Bugbear.B*, which took only one day to cause the damage the previous strain had caused in three days. The reason was that a flaw in MS-Outlook allowed the program to automatically open e-mail attachments. The perpetrator of the Bugbear strains has not been apprehended.

Virus Detection and Prevention

The following steps may be taken to protect and defend yourself from infection by computer viruses.

1. **Check before Use**: Before using any floppy disk or downloaded files, always run a virus-scanner program on them. There are numerous reputable vendors who have relatively inexpensive (or in some cases free) virus-scanning software available. Moreover, updates will be provided as a service by the vendor. As we have seen, the race to beat the attacker is based on knowing what is out there. You should also use the software to do a virus scan after each reboot of your computer.

2. **Create Emergency Disk**: For the worst-case scenario where you get infected and you cannot reboot your machine, the only saviour may be an emergency disk that you have set in advance to use for that scenario. Ensure that the disk is write protected at the time it is created.

3. **Disable**: Do not allow the enabling of such automatic features as the opening of e-mail attachments, downloading of files, or the like. Disable these features.

4. **Documents (MS)**: Do not open any MS-Word document unless you are certain it is not infected. Remember not to view these as "data files," since they may be infected with a macro virus.

5. **E-Mail**: Be cautious in the extreme about e-mail that you receive, even if you know and trust the sender very well, since anyone may be an unwitting victim. The above-described scenarios should be enough to convince anyone of that. If there is an attachment, especially if it is an executable file, you must verify that it is virus free. Delete it if there is doubt or, if you believe it to be valid and from a valid source, contact that source *before* opening it. Ask them what is in the file, whether they know if it is virus free from having scanned it, say, and why it has been sent to you. Then, and only then, should you attempt to open such a file.

6. **Infection Detected**: If your virus scanner detects an infection, locate the virus, identify it, and use the software to remove all traces of it. The virus must be removed from all systems in order to restore your computer to health. Remember, it is *detection*, *identification*, and *removal* in the case of a viral infection. If it is not possible to either identify or remove the

virus, then the infected program should be discarded and a new, clean backup copy should be reloaded.

7. **Software for Blocking**: Some more sophisticated software exists for the purpose of actually blocking behaviour that is deemed to be malicious. Again, reputable software vendors have numerous such devices available. For instance, there are *Internet filters*, which will screen out any e-mail related to pornography, violence, or other such offensive material as well as potentially malicious e-mail. There are *spam blockers* to prevent all sorts of irritating e-mail from getting through to you, not just the infected kind. There are *e-mail virus blockers*, which should take care of effectively protecting your computer by identifying and blocking potentially dangerous attachments.

Advanced Protection

There exist modern methods that excel in their ability to protect from and eliminate attacks. We look at two of the most common and most effective.

Generic Decryption: Polymorphic viruses may require more sophisticated software. The most modern such device is called a *generic decryption engine* (GDE). Basically a GDE tricks a polymorphic virus into decrypting and revealing itself. If a scanner with GDE is installed, then it makes three assumptions: (1) the body of the polymorphic virus has enciphering to thwart detection; (2) the virus must decrypt before it can execute; and (3) once a polymorphic virus does execute, it must immediately assume control of the computer to decipher the body of the virus, after which the control of the machine is taken over by the completely decrypted virus. The GDE loads each new program file into a self-contained virtual computer that is generated from RAM. It is inside this virtual computer that the program files run as though on a real computer. Therefore, a polymorphic virus can do no damage since it is running in the virtual computer, which is isolated from the real computer. The virtual computer allows the virus to decrypt after which the virus body is exposed to the GDE scanner, which can identify the strain via a signature. If there is no virus to expose, the GDE stops execution and drops the program, proceeding to the next file. Think of the GDE as a rat and think of the files loaded to it as injections given to the rat to detect the presence of a virus. If there is no adverse behaviour in the rat, there is no virus in the injected substance, whereas if there is, then the rat is observed for symptoms that will identify the virus.

A GDE scanner has five basic components: (1) a processes emulator, (2) a memory emulator, (3) a system emulator, (4) a virus signature scanner, and (5) a decision mechanism. The process emulator is an imitation of a CPU, which reads the instructions in an executable file. This includes software versions of all registers and other CPU hardware, so the actual processor is unaffected. The memory emulator imitates the memory of the computer, where the emulated memory is employed instead of real memory. The system emulator actually imitates the OS and hardware of a computer. This should also include a virtual

drive that is capable of being read, formatted, and so on. The virus signature scanner is a module that scans the program code of the loaded file for known virus signatures. This module interrupts the GDE process to return it to the scanner for it to look at the code for signatures. The decision as to when to interrupt is given by the decision-making mechanism, which may be the most vital part of the GDE since we want to ensure speed. Thus, proper decision making must be made so that the optimum use of the GDE is ensured. The GDE innovation seriously reduces the time taken to analyze polymorphic viruses, from weeks to minutes.

The second type of antivirus device is a comprehensive virus protection mechanism developed at IBM in the late 1990's. For more data on the original research papers from IBM and related development, go to the following site: *http://www.research.ibm.com/antivirus/*. In 1999, Symantec entered into a licensing agreement with IBM to market the idea as antivirus software for business and personal computing, officially released as a commercial product in October 2000.

Digital Immune System (DIS): The idea is, as the title suggests, to mimic the human immune system in a computer so that a virus is automatically captured as it enters a system to be analyzed, removed, and ensure that the system is updated with detection and protection mechanisms (if it is a new virus). Essentially this builds on the emulation idea described above. The central goal of the DIS is to drastically reduce the delay time between discovery of a virus and when a remedy is transmitted to all vulnerable systems. What we describe here is essentially the version designed by IBM and Symantec.

DIS Closed-Loop Process

We first describe this process, then illustrate the "closed-loop."

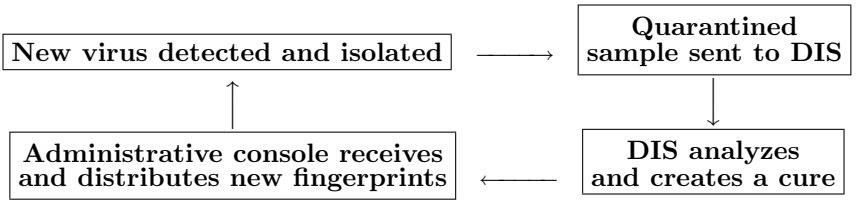
1. **Detection**: A virus is detected at some source point such as a gateway, server, or client machine.

2. **Quarantine**: A sample of the virus is sent to the Digital Immune System central quarantine where it is isolated and scanned with the latest virus definitions. If it turns out to be a known virus, then the cure can be sent immediately back to the source of infection and no further action is required. Otherwise, central quarantine strips all sensitive data such as MS-Word documents (to ensure confidentiality), and the sample is sent to Symantec Security Response. This transmission is accomplished over HTTP on port 80, using SSL, which ensures confidentiality and authentication (see Section 7.3).

3. **Automated Processing**: The DIS automatically analyzes the sample and creates a cure, which is sent back to the administrative console at the source.

4. **Administrative Console**: The new fingerprint is distributed by the administrative console throughout the source network to be added as an update to the current virus definitions.

Diagram 8.1 DIS Closed-Loop Virus Methodology



Analysis of DIS: The DIS, arguably, represents the pinnacle of antivirus software currently available. The DIS approach is stronger than other antivirus techniques since it is automated and scalable and does not require human intervention for decoding viruses and creating signatures. The number of false positives is kept low and supplies end-to-end automation of submission, analysis, and transmission of new fingerprints for virus definition updates. There is relatively little maintenance needed with the DIS system, and costs are minimal given the alternatives. If the administrative console is allowed to streamline the control of the system at the given organizational source, then the maximum benefit will be received, since administrators have control of the level of automation.

There are other kinds of malicious programs requiring a host program and are not considered to be viruses due to the manner in which they operate. We now look at their morphology.

◆ Logic Bombs

The *logic bomb*, also known as *slag code*, is a much older device than the virus. Like a bomb, it requires a trigger to set it off ("explode") until which time it remains dormant in a host program. The results are particularly ugly, as would be the effects of a real bomb in a populated area. It may make the entire hard drive unreadable, or it may be more insidious and merely change a byte here and there, avoiding detection until it does irreversible damage. The trigger may be any of a number of vehicles from an elapsed amount of time; a particular date and time (December 31, 1999, at 24:00 hours, for instance); or perhaps the removal of an employee from the payroll file, indicating that he was fired. If he were really clever, the bomb would go off a few months after his termination. In this case, the logic bomb would trigger a piece of malicious code to *slag* (destroy) essential files in the company's system. This use of logic bombs clearly demonstrates the need for audit trails, as well as clearly delineated breakdown of individual duties at any organization.

A logic bomb may be considered to be a delayed-action virus in terms of effect. They can be eliminated before they explode by using virus-scanning software. If the scanning software is put on auto-protect mode, including e-mail screening, then the probability of catching a logic bomb in time is increased.

◆ Trojan Horse

The name *Trojan horse* comes from the story of Troy. It is piece of malicious code that is inserted into a seemingly benign program, but it differs from a virus in that it does not replicate itself. For instance, you might download a movie or some music from the Internet and find that it contained a Trojan horse that erases your hard disk. Other popular alternatives for downloads that contain Trojan horses are FTP archives (see page 224). Another is peer-to-peer exchanges over an IRC channel — *IRC* stands for *Internet Relay Chat*, which was originally designed for people to "chat" in real time. IRC users trade movies, music, games, and software — peer-to-peer sharing. You have to be careful since the more you download or exchange, the greater the risk of getting a Trojan horse as part of the deal, since Trojan horses are very common among IRC traders. Do not download from people or sites unless you are 100% certain of them. Never use auto-download features, since you must check every file first. Moreover, check it out *before* you download it since if you download an executable file that has a Trojan horse and run it to check it out, then you are already infected. As with the other types of infection discussed above, use a virus scanner, but do not *rely* on it. The fact of the matter is that even when up to date, it may miss something, especially if the infection is very new.

If you do get infected, then the best eradication is a backup of the entire hard disk, and reinstall the OS and all applications from their original disks. This might become necessary since a typical Trojan horse attack is to destroy the *file allocation table* (FAT) on your hard disk. A FAT is the table that maintains a map of the clusters on the hard disk (see page 274). Without a FAT or with a damaged FAT, your computer will not operate properly.

An interesting example of the use of a Trojan horse comes from the OpenSSH source (see page 271). It turns out that in 2002, only the second day after the latest version of OpenSSH was released and ready for download on the Internet, the developers made the somewhat startling discovery that the original package had been exchanged for one with a Trojan horse embedded in it. The checksum (see page 257) was found to have been altered. When installed, the Trojan horse attempted to communicate with another Internet computer to await commands. Fortunately, they caught it early.

Now we look at malicious code that has similarities to a virus but some differing characteristics that make it a favourite for a network attack.

◆ Worms

A *worm* is a (malicious or nonmalicious) code that replicates itself and is self-propagating. Thus, a worm is independent and designed to thrive in network environments without human intervention. Unlike a virus, it needs no host program. Rather, the computers themselves provide the hosts. The programs

running on individual computer hosts are called *segments* of the complete worm. The OS in a given system is not needed to manage the worms since they seek out resources for themselves, finding remote machines and spawning a remote process on that machine. Thus, a worm program is a program that spans machine boundaries as part of a distributed computation. Some worms have a main segment that coordinates the activities of the other segments, and such a worm is sometimes called an *octopus*. Worms that are contained within a single computer are sometimes given the name *host worms*, and those that have many segments on more than one machine are deemed to be *network worms*. A host worm uses the network connections for the sole purpose of copying itself to other machines, whereas the network worm uses the network connections for communication between each of its segments. Those host worms that delete themselves after launching a copy on another host, guaranteeing there is only one version of the worm running on the network at any given time, are sometimes called *rabbits*. It is the network worm that is most common and which will be our focus.

In the 1970's before the Internet was a fact, the first two worms were sent through ARPANET (see page 223), the predecessor of the Internet, as programs called *Creeper* and *Reaper*. First there was Creeper, which used idle processor CPU time in ARPANET to replicate itself on one system and move onto the next. Then Reaper was created to follow the path of Creeper through the network, deleting the segments of Creeper as it went. However, these did no damage to the computers they "infected" since they were designed to explore the possibility of making use of idle CPU time. Such nonmalicious worms are called *existential worms*, since their only function is to stay alive and propagate. In 1973, F. Shoch and J.A. Hupp of the Xerox Palo Alo Research Center developed an existential worm program to move through an Ethernet network. Later, in 1982, these two individuals wrote a paper [87], which contained the first formal definition of the term "worm." Shoch lifted the term from a 1975 science fiction novel, called *Shockwave Rider*, in which the author, John Brunner, conceived of the concept of a worm that takes over a network, and as one of *Shockwave Rider*'s characters puts it: "... now it's so goddamn comprehensive that it cannot be killed. Not short of demolishing the net!" (see [12, p. 247]).

As Shoch and Huff found, even the creation of an existential worm opens problems with its control. In the initial stages of development of their worms, they once left one running on a system overnight only to return the next morning to find it had crashed several hosts, and their attempts to reboot resulted in the worm's crashing the system. Therefore, they had to build a code in the worm that would shut it down when a signal was received through the network. These were problems when the creation of the worms was that based on *benign* intent. When written as malicious code, the consequences proved to be disastrous.

There is the case of the infamous *Morris worm*. This was the first true Internet worm. In 1988, a Cornell University graduate student, and son of the chief scientist at NSA's National Security Center, Robert Tappan Morris Jr., wrote a worm program (designated for UNIX systems). Supposedly his intention was that it be an existential network worm. He got it wrong. His program had

serious shortcomings in terms of containing the worm. On November 22, 1988, after he released the worm, it propagated itself so many times that it effectively crashed several thousand host machines. It is estimated that as much as ten million dollars (U.S.) was lost in terms of productivity, and this was despite the fact that the worm left no permanent damage once eradicated. Morris was sentenced to three years probation and ordered to pay a fine of ten thousand dollars (U.S.).

Although there have been worms in the past century, the more recent ones in this millennium have been the most devastating in terms of cost. In July 2001, two variants of the *Code Red* worm were released. It exploited a security weakness in MS-Internet Information Server (MS-IIS). Code Red launched a three-phase attack: scanning, flooding, and sleeping. In the scanning stage, it sought vulnerable machines and ran malicious code on them. In the flooding stage, false IP packets were sent to "flood" machines with useless messaging. At the height of its activity, Code Red infected a couple thousand computers each minute, ultimately contaminating in excess of one-third of a million machines and costing 1.2 billion dollars (U.S.). The final sleep stage was intended to last forever. The culprits who wrote Code Red have not yet been apprehended.

In August 2003, the *Blaster* worm, also called *Lovesan*, caused mayhem with various Windows servers. Blaster searched for unprotected machines and sent itself to those computers. Once it located a vulnerable machine, it sought out the file *mblast.exe*, retrieved it, then scanned other systems similarly. Blaster was written to launch a DOS attack (see page 253) on Microsoft's updated WWW site. Microsoft found a means of thwarting the attack on their site, but Blaster still infected around a half million computers. Microsoft offered a quarter of a million dollars (U.S.) for information that would lead to the arrest of Blaster's creators. However, to date, there have been no arrests. Microsoft has a five million dollar reward fund for the apprehension of the various malicious code authors not yet caught.

On Friday, April 30, 2004, a worm called *Sasser* began spreading over the Internet. It exploited a vulnerability of MS-Windows Local Security Authority Subsystem Service (LSASS). Sasser scanned for vulnerable machines, created a remote connection with them, installed an FTP server, and downloaded itself to the new host. From there it sought out the vulnerable LSASS components on other machines. Sasser caused the LSASS component of Windows to crash. On May 7, 2004, German authorities arrested Sven Jaschan, an eighteen-year-old student who created a total of five separate versions of Sasser. Jaschan is also responsible for twenty-eight variants of the *Netsky* worm. Key evidence leading to Jaschan's apprehension was given by a peer group familiar with his activities. They had approached Microsoft officials in Germany asking about the reward. Once informed that they would indeed get it, they turned him in, after which Microsoft paid the quarter million dollar (U.S.) reward to them. This arrest caused Microsoft officials to have confidence that their reward fund would have a positive effect on the eventual arrest of the perpetrators of the Blaster and Code Red worms.

In early 2006, the *Kama Sutra worm* came into existence. It infected files

within Windows OS and was ready to be unleashed on February 3, 2006, and the third of each month thereafter. This infection promised sexy pictures with e-mail subject lines such as "Kama Sutra pics," thus relying on a computer user's desire to see nasty pictures and to get him or her to take action by double clicking on an attachment. However, since the payload was delayed until the third of the month, the user was unaware of the infection. The end result was the overwriting of Window's office documents such as Excel spread sheets and PDFs (portable document format).

Although users of Apple computer's MAC OS X are usually spared from the above type of attack, a worm called *OSX/Leap-A* was the first such virus-like infection to be aimed specifically at the MAC platform in early 2006. The mechanism for spreading was via Apple's *iChat* instant messaging program, which happens to be compatible with America Online's AIM instant messaging program. The worm infected the so-called *buddy list*. However, the infection was not automatic since the worm program would first ask the user to accept the file. In any case, this represented an example of how malicious code was continuing to spread to other platforms.

Antiworm Countermeasures: In the computer word, one must be aware of both internal and external potential attackers, especially if you are an employer. Disgruntled employees can be a greater threat than any external source. We have talked at length about measures against internal threats such as the use of firewalls (see Section 7.4), monitoring, and access control. Relying solely on firewalls is insufficient. Each server must be protected as a separate entity. We have already discussed the technological devices such as blocking software, including antivirus mechanisms, and access-control software (see page 279). There should also be human intervention such as risk analysis and in-depth security policies. Using the human and technological devices in concert can be the most effective of security-management mechanisms.

8.3 Smart Cards

The term "smart card" has entered our discussions briefly thus far (see page 268, for instance). Now it is time to delve into the details. First we must learn about certain details concerning technical aspects of computers as they relate to smart cards.

◆ Processors and Microcontrollers

A *microprocessor* is any integrated circuit (IC) containing the CPU of a small computer. A *CPU* is the *Central Processing Unit*, which controls the operation of a computer, including the execution of arithmetic and logical operations as well as other instructions. In a smart card or microcomputer, the entire CPU is on a single chip. In general a computer *processor* is the logic circuitry that responds to and deals with the instructions that run the computer. However, in the modern day, the term "processor" has been replaced by "CPU."

A microcontroller is a computer on a chip and is created via the integration of the fundamental components of a microprocessor: RAM, ROM, and digital I/O (input/output) ports into the same chip die. Other features might include: serial I/O, a timer module; analogue to digital converters (ADC); and even serial peripheral drivers. Examples are Motorola's M68HC08 family of 8-bit microcontrollers and Microchip's PIC17 Family with 16-bit program word.

◆ Memory

Nonvolatile memory means any kind of solid-state memory that does not lose its contents when the computer is turned off. In the case of a memory card, when it is removed from the card reader, the power is cut off, yet the card stores the data. On the other hand, *volatile* memory loses its contents when the computer is turned off. Nearly all RAM is volatile except, of course, battery-powered RAM. Included under the heading of nonvolatile memory are not only EEPROM, but also all other forms of ROM such as programmable read-only memory (PROM); erasable programmable read-only memory (EPROM); and flash memory, sometimes called *flash RAM*. The latter type of memory can be erased and reprogrammed. The term "flash" is derived from the fact that in a microchip a section of memory cells is erased in one solitary act, *in a flash*. Flash memory is employed in PC cards, digital cell phones, printers, and digital cameras, for example.

SRAM is *static* RAM as opposed to the more common *dynamic* RAM or *DRAM*. The term "static" is employed to differentiate it from the conventional form of RAM in that it does not need to be refreshed as does DRAM. Therefore, SRAM is faster and more reliable than DRAM. However, it is more expensive in terms of financial cost, storage space, and power consumption. Thus, DRAM is necessarily volatile memory.

What is a Smart Card? Smart cards are made of plastic and are of credit-card size, having an embedded microprocessor chip with internal memory or merely a memory chip with nonprogramming logic.

Types of Smart Cards: Classifications for smart cards are described in the following.

1. **Standard Memory Cards**: These are cards that merely store data. They do not possess data-processing capabilities. Typically, these cards have a *magnetic strip* (so are often called *magnetic strip cards*). These cards store private data, usually employed as credit or debit cards, which require physical contact with a device to read the data on the magnetic strip.

2. **Intelligent Memory Cards**: These cards have a built-in wired logic circuit to access the memory (usually 1 K to 16 K bits) of the card. Sometimes these cards can be configured to restrict access via a password or system key. These cards are often called *protected memory cards*.

3. **Stored-Value Cards**: Sometimes these are called *memory cards with register*. These are cards that have security features hardwired into the chip at the point of manufacture. Examples of such cards are prepaid phone cards, wherein a terminal inside the pay phone will write a declining balance into the card's memory. The card is discarded when the balance is zero; or if the card has a rechargeable capacity, it can be reset.

4. **Processor Cards**: These cards, perhaps the most deserving of the name *smart card*, contain memory and a processor and have data-processing capabilities. This is an integrated circuit (IC) card with ISO/IEC 7816 interface. (We learned about the ISO on page 241. The IEC is the *International Electrotechnical Commission*, a Switzerland-based organization that sets standards for electronic devices. A committee, JTC1, is joint between ISO and IEC, and its mandate is information technology standardization.) If an 8-bit microprocessor had the task of RSA cryptographic calculations, for instance, it could take several minutes. Thus, a cryptographic coprocessor is typically added to the architecture, thereby reducing cryptographic calculations to a few hundred microseconds.

Types 1–3 are often grouped under the single heading of *memory cards* and type 4 under the heading of *microprocessor chip cards*. Memory cards are, naturally, the least expensive and most common. They contain what is called *Electronically Erasable Programmable Read-Only Memory* (EEPROM) nonvolatile memory. For security, the data may be locked in by a PIN of up to eight digits written to a special file on the card.

Chips: There are three kinds of smart card chips as follows.

1. **Memory Chips**: Naturally, the most basic and least expensive are those chips that merely store data and have no processing capabilities. Once created, memory chips cannot be reprogrammed, since they can hold only static data such as personal information that do not require dynamic enciphering capacity. To change the capacity of such a memory card, it would need to be replaced entirely.

2. **Applications-Specific Integrated Circuits (ASIC)**: The ASIC chips are hardwired to keep data and execute a specific processing job. Of course, this processing capacity makes the ASIC chip stronger than the memory chip. Yet, the ASIC chip cannot be reprogrammed, as is the case with the memory chip. However, the ASIC chip does allow for some static encryption, but this is suitable only for low-level security applications.

3. **Microprocessor Chips**: These chips are the most powerful and versatile of the three types. They cannot only do what both the memory and ASIC chips can do, but also they are capable of dynamic encryption, and they can be reprogrammed or updated, unlike the previous two. Processor cards have microprocessor chips that typically come in 8-, 16-, or 32-bit formats. Their data storage may range from 300 to 32,000 bytes.

   Microprocessor-based smart cards have the benefits of (1) a high level of security, having the capacity to execute PKC or SKC protocols, including DES, RSA, and ECC; (2) multiple applications on the same card; and (3) ease of updating existing applications or the addition of new ones.

   Microprocessor cards have numerous applications: the access medium for identification, for electronic signatures, for access to restricted areas, to protect data storage, and for e-commerce.

Card Operating Systems: The microprocessor in a smart card is controlled by a *Card Operating System* (COS), which is a piece of firmware stored in the ROM of the microcontroller IC embedded in the card. The COS has the following fundamental tasks.

1. Both establish and control communication between the card and any card-reading device.
2. File management.
3. Memory management.
4. Management of applications including loading and operating.
5. Protect data access.
6. Instruction processing and execution control.
7. Execute and manage cryptographic protocols when communicating with a card-reading device.

Smart Cards and PKI: The structure for smart cards employing PKI is described in RFC 2459 (see [78]). Smart cards may be embedded with functions that generate public and private PKC keys inside the cards, meaning that the private key is not sent to any site outside the card. In other words, the smart card need not export the private key in order to use a given application.

Suppose that Alice interfaces her smart card with her computer for the purpose of using some application, which requires Alice's signature on a document to authenticate her. In order to get the card to communicate with the application, a hash of Alice's document, e-mail for instance, is sent to the card. The card signs the document with her private key (all this taking place inside the card), and the signed document is sent to the application. Hence, her private key is never exposed to the outside, in particular to her computer. Smart cards may employ SSH (see page 268) to authenticate to an application remotely, for instance.

Contact vs. Contactless: The communication between a smart card and a card reader or detection device might be direct, namely, physical contact, or *contactless* using radio frequency. Thus, smart cards are further divided into contact and contactless (sometimes called *proximity*) cards. Contactless cards are embedded with not only a chip, but also an antenna for the purpose of sending a signal to the reading device. Typically, a few centimeters of distance will allow the mechanism to receive the signal and authenticate the card owner for access to that device. Contact cards are usually employed for access to secure areas in a business enterprise, for instance, whereas contactless cards are typically used for mass-transit access or for door locks.

Contactless cards use wireless self-powered induction technology, as defined in the standard, ISO/IEC 14443. The latest use for such a card in mass transit is the *Oyster Card* issued in London, England, in January 2004. The card is rechargeable, secure since, if lost or stolen, it may be cancelled and reissued; and it is valid London-wide including the "Tube," Tramlink, DLR (Docklands Light Railway), and National Rail services across the entire London bus network.

Contactless cards have the benefits of speed of transaction time; convenience; low maintenance (compared with contact cards); and consumer appeal where key fobs, rings, or other devices may be used in place of a plastic card. Many upscale residential areas are looking at replacing locks with contactless smart cards in North America. The fact remains that contacts are the most frequent breakdown points in the electromagnetic system as a result of dirt, and wear on the mechanism. Contactless cards solve these problems and improve performance in the balance, so user acceptance will surely increase.

Last, there are cards that combine certain features, called *combi-cards* or *multifunction* cards. This might involve a combination of password and biometric such as a fingerprint. Also, there is the possibility of combining both contact and contactless features in one card.

Physical Properties: The actual body of the card is plastic, which may be *polyvinyl chloride* (PVC) or *acrylonitrile butadiene styrene* (ABS). The card itself may contain a signature strip, a printed signature, or a cardholder photograph. Of course, the plastic body will be embossed with the proprietary graphics such as with Visa or MasterCard. The size of the card is specified by ISO/IEC 7816-1, namely, $85.6 \times 54 \times 0.76$ mm. This standard includes definitions of resistance to static electricity, electromagnetic radiation and mechanical stress, as well as the location of the card's magnetic strip and embossing area.

The dimension and location of the contacts is specified in ISO/IEC 7816-2. This includes the *module*, which is the smallest part of the card that is capable of accommodating a chip and its contacts. The mechanism for securing the module in place on the card is via encasing it in a resin amalgam, which, for security reasons, should be designed so it cannot be removed without destroying the circuitry (see page 293).

There are also cards, called *mini-cards*, which are in size between that of a regular smart card and its module. These are often used for mass-transit applications, where the size of the cards mimics the size of the magnetic-strip tickets they replace.

In the following, Diagrams 8.2 and 8.3 give the placement of the electrical contacts in a smart card chip, numbered C1–C8, and describe the function of each.

Diagram 8.2 Smart Card Chip — Electrical Contacts

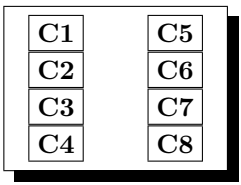


Diagram 8.3 Functions of Electrical Contacts

Position	Abbreviation	Function
C1	VCC	Power Supply Voltage
C2	RST	Reset Microprocessor
C3	CLK	Clock Frequency
C4	RFU	Reserved for Future Use
C5	GND	Ground
C6	VPP	Programming or Write Voltage
C7	I/O	Serial Input/Output Line
C8	RFU	Reserved for Future Use

All the data transmitted to and from a smart card are through the C7 contact point. Once a smart card is inserted into a card reader, for instance, a client-server relationship ensues. The physical transmission is defined in ISO/IEC 7816-3, so any reader must conform to that standard.

Card Origins: The French are responsible for the term "smart card," in development since the 1970's when the French invested a large amount of money into this R&D technology. They originally called these cards *carte a memoire* or *memory card* in the 1970's. The French government's marketing arm, *Intelimatique*, coined the term *smart card* in 1980. In fact, Roy Bright of Intelimatique

(see [9]) was the one who coined the word "smart card" (which is sometimes written as a single word *smartcard*). In 1970, the concept of the smart card was filed in a patent by Kunitaka Arimura of Japan. The patent was restricted to Japan and to the technical aspects of the smart card idea, namely, to integrate data storage and arithmetic logic on a single silicon chip. Shortly after his patent was filed, the first smart cards were issued in Japan.

The first patent for an IC card (which we consider to have the properties of a smart card in today's world) was filed by the French journalist Roland Moreno in 1974 (see [65]). Moreno's patent was the first to be broad-based not only in France, but also in major industrial countries around the globe. By 1977, the first commercial developers of an IC card product were three manufacturers, Bull CP8, SGS Thompson, and Schlumberger. Also in that year, the French banking system had a smart card payment scheme in place, and by 1978 the first prototype card was produced. In 1979, Motorola introduced the first secure individual chip microcontroller. It was a prototype made in Toulouse, France, for Bull CP8, having programmable 1-K memory and microprocessor 6805.

Credit cards contain data including either signature or picture for identification of the person authorized to use it for account access or services. The use of credit cards, on a local scale, actually goes back to the 1920's in the United States, when some oil companies and hotel chains started issuing them to customers for purchases at their enterprises. On a global scale, the first credit card for use at a large multiplicity of businesses was Diners Club Inc., in 1950. Their card employed PVC plastic, which replaced earlier paper-based cards. They were the first to institute charging an annual fee billed to their cardholders. By 1958, American Express entered the stage with its card. The first bank to issue a card was the Bank of America in 1959 with its *BankAmericard* distributed initially in California only, adding other states starting in 1966. In 1976, it was renamed Visa, and later MasterCard followed suit. In 1981, MasterCard (formerly called Master Charge) introduced the first gold-card program, and in 1983 it was the first to employ a laser hologram as an antifraud mechanism. Internationally, BankAmericard was known by other names before Visa came into being. In Canada, a number of banks, in concert, issued Chargex cards. In the U.K., the BarclayCard was issued by Barclay's Bank. Both of the latter used the blue-white-blue motif familiar to BankAmericard holders. The blue and gold motif on the Visa cards was selected to represent the blue sky and gold-coloured hills of California, where BankAmericard originated.

The 1980's saw much field testing of smart cards. The world's first significant IC card test was conducted in France with their testing of serial memory phone cards in 1982. In 1983, the first nationwide smart card scheme was put in place by the French for their public telephone payment system (see [9]). In 1984, the French adopted the Bull CP8 card as their standard for the first version of their bank debit cards Carte Bleue. By 1986, the French also were the first to introduce a smart card scheme in the form of a health card. In 1987, the ISO introduced the first card standards in the form of ISO/IEC 7816-X. The 7816 series of standards today define everything from the physical shape of the card to the format the commands may take when communicating with the card. This

includes not only the functionality of the card, but also the very position and shape of the electrical connectors and the protocols defining the power voltages to be applied to them (see Diagrams 8.2 and 8.3).

By the early 1990's the French were involved in field testing of combi-cards. Also in the early 1990's, Germany was involved in memory card distribution on a mass scale. In 1994, they started the distribution of some 80 million serial memory chip citizen health cards. Now, every German citizen has a health smart card. By the mid-1990's, mobile phone use was conducted and paid via smart cards by some three million users. By the late 1990's, the major players in the credit card industry were looking at standards for interoperability. In 1996, MasterCard and Visa began developing two types: JavaCard, sponsored by Visa, and *Multi-application Operating System* (MULTOS), sponsored by MasterCard.

Attacks on Smart Cards: There are numerous attacks against smart cards that need to be reviewed so we may better understand the threats and not fall victim to them.

Side-channel attacks are those wherein a cryptanalyst, Mallory, say, has an additional channel of information about the system he is trying to break. Timing analysis of message encryption falls into this category. The reason that side-channel attacks are so effective against smart cards is that Mallory may have full control of the card. Countermeasures for side-channel attacks come from a combination of software implementations and actual hardware.

Countermeasures against timing attacks include the following: (1) avoiding delays (make all operations take the same amount of time), (2) equalization of multiplication and squaring (the time taken to execute multiplication and exponentiation should be set to be very similar), (3) power consumption balancing (operations should be made to appear constant from outside the card, which can be accomplished with dummy gates and the like to even out the power consumption to some constant value), (4) add random noise (enough to stop an attack), and (5) physical shielding.

Magnetic strip cards, having no computing power at all, are subject to what is known as a *skimming attack*. In this case, an illegal card reader can be used to copy the data in the card (once it is swiped through the illegal device) for the purpose of counterfeiting cards and incurring illegal charges. Some criminals have even resorted to planting these devices in legal ATMs for the purpose of gathering these data. Once the data have been captured, the card owner might be presented with a screen that says there has been a malfunction. In some cases, the criminals engineer the card reader so that it does not interfere with the ATM's function. In this case, the customer will get his or her cash, when making a withdrawal, say, but his or her data are still captured for later use by the criminal element. The ATMs most susceptible to this kind of attack are not usually the ones at banks themselves but rather at convenience stores, bars, hotel lobbies, and the like. Moreover, they are typically the kind of ATM where the card is swiped rather than inserted into the machine directly. Also, skimming may be accomplished by dishonest businesses when your card is taken out of your sight for payment, say at a restaurant, and run through a skimmer.

To thwart skimming attacks, do not use ATMs where something appears to be out of place. Keep all PINS safe and never give them to anyone. Do not let strangers "assist" you at an ATM. If your card is not returned after usage in an ATM, immediately contact the institution that issued the card. Treat your cards as if they were cash and do not let them out of your sight.

Returning to IC cards, there are *tampering attacks*, which may be broken down into four subsets: (1) *microprobing*, where the chip itself is accessed and manipulated and there is direct tampering with the IC; (2) *software attacks*, the exploitation of weaknesses in cryptographic protocols or their implementation via the I/O interface; (3) *eavesdropping*, the monitoring of any electronic radiation produced by the microprocessor's executions; and (4) *fault generation*, creating malfunctions in a microprocessor for the purpose of establishing access.

Attacks (2)–(4) are *noninvasive* attacks. On the other hand, microprobing is an *invasive attack* that requires a significant amount of laboratory time, expensive equipment, and expertise. In order to extract the chip, the plastic card is destroyed. Once the chip is removed, it may be mapped and analyzed, and information is obtained. One countermeasure for such attacks (already available with some microprocessors) is the embedding of a sensor mesh above the actual chip, so that any tampering would trigger an erasure of nonvolatile memory.

With noninvasive attacks, smart cards are especially vulnerable since their microprocessors are exposed without the safeguards built into larger devices, such as electromagnetic shielding. A microprocessor is basically a collection of a relatively small number of *flipflops* (registers, latches, and SRAM cells), which establish its current state, together with a logic design that calculates that state based on a clock cycle and other states. A *register* is a specialized, high-speed storage region of the CPU. No data are capable of being processed before being put into registers. A CPU's power is defined in terms of the number and capacity of registers it possesses. For example, an 8-bit CPU has registers that maintain 8-bit words each, so each command sent to such a CPU is capable of handling 8 bits of information. A *latch* is a digital logic circuit for storing bits. The components of a latch are the data input to it, a clock input, and its output. The term "latch" comes from the function of the clock activity; for example, when active, the clock input triggers the data input to be "latched" (stored) and transferred to output when the clock input becomes inactive. The value of the clock output is then set and maintained until the clock input is again activated. This analogue effect is one of the vulnerabilities that can be exploited via fault-generation attacks in smart cards, namely, by causing one or more flipflops to take on the incorrect state (see [8]).

Countermeasures to thwart noninvasive attacks include inserting a random-number generator at the clock-cycle level and embedding a tamper sensor that will disable the entire microprocessor upon detection of unauthorized activity.

8.4 Biometrics

◆ Overview

The science and technology of quantifying and analyzing biological or behavioural data is what we call *biometrics*. The characteristics to be measured are DNA; ear geometry; eye retina (the nerve endings inside the eyeball that capture and send light to the brain) and irises (the coloured part visible at the front of the eye), facial geometry, fingerprints, hand geometry, and voice frequency. The data to be analyzed are stored in a database for comparison with existing records.

Typically, software is used to identify specific *match points*, which are then processed into a value that may be compared with biometric data that are scanned when the owner of a smart card, say, tries to gain access. Biometrics may be used to provide authentication for access to a bank account, to pay for products or services from a business, to pay for telephone charges, and so on. Biometrics can be employed in addition to, or in place of, say, a PIN.

Sensors are used to record the biometric information. Cameras are used for facial, eye, hand, and ear geometry; microphones for voice; chemical laboratories for DNA; and any number of sensors for fingerprints including pressure sensitive, thermal, optic, and capacitive devices.

◆ Biometrics and Smart Cards

Various government agencies use biometrics in their smart card technology. The U.S. Department of Defense Common Access Card has a photograph together with a fingerprint embedded in its functionality. Spain has a social security card including biometrics in its smart card application. The Netherlands has a system called *Privum* for automated border crossing. Their smart card has a photograph and iris biometrics. Brunei employs a national ID smart card having a photograph together with fingerprint biometrics. The United Kingdom has the *Asylum Seekers Card*, which is a smart card with a photograph and fingerprint biometrics. It is not long before more countries are added to the list in an effort to secure their borders.

The bottom line for smart cards supported by biometrics is that it raises security levels to very high standards. The reason is that such cards possess the following.

The Three Fundamental Aspects of Authentication

1. Something the user *has* (the smart card, itself)
2. Something the user *knows* (a PIN or password)
3. Something the user *is* (the biometrics)

◆ Accuracy and Robustness of Biometrics

Biometric Traits

Biometric traits develop in one of three ways:

1. *Genotypic* (through genetics)
2. *Phenotypic* (through early embryo development)
3. *Behavioural* (through training)

Robust biometrics are those that are not subject to significant changes. Certain biometric traits may vary over time due to aging, growth, injury and later regeneration, wear and tear, and so on. The *least* changeable biometrics are DNA and iris pattern followed by retina, fingerprints, and hand geometry. In terms of *accuracy* (minimal error rates plus clarity and consistency), iris and retina measurements rank *ahead* of DNA, although all three are difficult quantifications to obtain and are costly to process. The reason that DNA trails the other two eye biometrics is that DNA cannot distinguish between monozygotic twins, but the eye biometrics can do so (and better than the other biometrics). Fingerprints rank roughly fourth on the accuracy scale but are relatively easy to obtain and inexpensive to process in comparison with the other three. An iris match against a database can be made 300 times faster than a match to a fingerprint in the same database. Hence, despite the cost differential, the speed and high accuracy of eye biometrics make it vastly superior to the fingerprinting biometric. Once costs descend, this must surely be the medium of choice, if for no other reason than the key factor in selection of an appropriate biometric is its accuracy. At the bottom of the list are face geometry, followed by finger geometry and voice patterning.

◆ Verification vs. Identification

We discussed the use of smart cards and biometrics for *verification* of individuals above, where verification means the following.

Verification

The individual's identity is entered into the system, via a smart card, say, then a biometric feature is scanned. If that scanned trait matches the one previously stored in the card, then verification is successful. This kind of "verification" is also often called "authentication" of the individual.

The notion of verification must be separated from the issue of *identification*, given as follows.

Identification

An individual's recorded biometric feature is compared with *all* the corresponding biometrics in the database. If there is a match, then the individual is identified, and the user's ID may be processed later for verification.

Identification is very useful in fighting crime. For instance, if an individual's fingerprint or DNA, say, is lifted from a crime scene, and a match is made to it after searching a database, this provides crime fighters with evidence to prosecute.

In order for biometrics to be effective, there must be an *enrolment process*, where an individual consents to having a biometric image captured, such as a fingerprint or eye scan, from which the characteristics are extracted. This allows the creation of the user's biometric template, which is stored centrally, in a database, or locally, on a smart card, say. Think of verification as a one-to-one comparison, which confirms that the credential belongs to the individual who is presenting it. The authenticating device need only have access to the individual's enrolled biometric template, which may be stored locally or in a database. Identification, on the other hand, is a one-to-many comparison. It verifies that the given entity exists within a given population and is not enrolled with another ID. Moreover, it will verify that the individual is not on a list of prohibited entities. In this case, the database must contain a set of all entities applying for the access, say, to enter a country, and their biometric templates.

As shown in Diagrams 8.4 and 8.5, the acceptance or rejection will be based on some threshold value derived from the security policy of the system being accessed.

Diagram 8.4 Verification

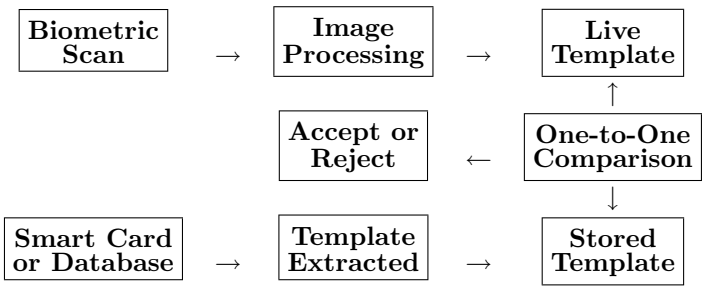
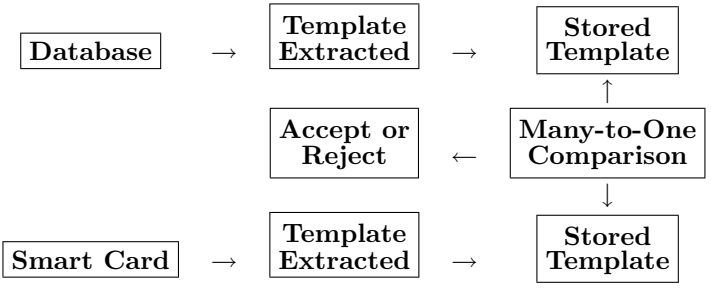


Diagram 8.5 Identification



Exercises

8.4. Suppose that a smart card (see Section 8.3) uses the RSA cipher with public encryption exponent $e = 3$. Next assume that m is Alice's credit card number and she buys merchandise from three shops whose public moduli are n_1, n_2, and n_3, respectively. Thus, each shop computes

$$m^3 \pmod{n_j} \text{ for } j = 1, 2, 3,$$

respectively. If Mallory has been observing these transactions, how can he recover m?

8.5. Cite some problems that might occur with the use of voice as a biometric identifier.

8.6. Compare fingerprint and iris scanning as biometric identifiers from the perspective of which is more accurate and least open to replication.

Appendix A: Fundamental Facts

In this appendix, we set down some fundamental facts, beginning with the fundamental notion of a set. Proofs may be found in standard introductory texts on the subject matter.

◆ Well-Definedness

A set of objects is *well defined* provided that it is always possible to determine whether or not a particular element belongs to the set. The classical example of a collection that is *not* well defined is described as follows. Suppose that there is a library with many books, and each of these books may be placed into one of two categories, those that list themselves in their own index and those that do not. The chief librarian decides to set up a Master Directory, which will keep track of those books that do not list themselves. Now, the question arises: Does the Master Directory list itself? If it does not, then it *should* since it only lists those that do not list themselves. If it does, then it *should not* for the same reason—a paradox! This is called the *Russell Paradox* or *Russell Antinomy*. The problem illustrated by the Russell Paradox is with *self-referential* collections of objects. We see that Russell's collection is not *well defined*, so it is not a set. Russell's example may be symbolized as $\mathcal{S} = \{x : x \notin \mathcal{S}\}$. The term "unset" is often used to describe such a situation.

◆ Sets

Definition A.1 Sets

A set *is a* well-defined *collection of* distinct *objects. The terms* set*,* collection*, and* aggregate *are synonymous. The* objects *in the set are called* elements *or* members*. We write $a \in \mathcal{S}$ to denote membership of an element a in a set $\mathcal{S}$, and if a is not in $\mathcal{S}$, then we write $a \notin \mathcal{S}$.*

This definition *avoids* the problems of the contradictions that arise in such discussions as the Russell Antinomy.

Set notation is given by putting elements between two braces. For instance, an important set is the set of *natural numbers*:

$$\mathbb{N} = \{1, 2, 3, 4, \ldots\}.$$

In general, we may specify a set by properties. For instance,

$$\{x \in \mathbb{N} : x > 3\}$$

specifies those natural numbers that satisfy the property of being bigger than 3, which is the same as $\{x \in \mathbb{N} : x \neq 1, 2, 3\}$.

Definition A.2 **Subsets and Equality**

A set $\mathcal{T}$ *is called a* subset *of a set* $\mathcal{S}$*, denoted by* $\mathcal{T} \subseteq \mathcal{S}$ *if every element of* $\mathcal{T}$ *is in* $\mathcal{S}$*. On the other hand, if there is an element* $t \in \mathcal{T}$ *such that* $t \notin \mathcal{S}$*, then we write* $\mathcal{T} \not\subseteq \mathcal{S}$ *and say that* $\mathcal{T}$ *is* not *a subset of* $\mathcal{S}$*. We say that two sets* $\mathcal{S}$ *and* $\mathcal{T}$ *are* equal*, denoted by* $\mathcal{T} = \mathcal{S}$ *provided that* $t \in \mathcal{T}$ *if and only if* $t \in \mathcal{S}$*, namely both* $\mathcal{T} \subseteq \mathcal{S}$*, and* $\mathcal{S} \subseteq \mathcal{T}$*. If* $\mathcal{T} \subseteq \mathcal{S}$*, but* $\mathcal{T} \neq \mathcal{S}$*, then we write* $\mathcal{T} \subset \mathcal{S}$ *and call* $\mathcal{T}$ *a* proper *subset of* $\mathcal{S}$*. All sets contain the* empty set*, denoted by* $\varnothing$*, or* $\{\}$*, consisting of no elements. The set of* all *subsets of a given set* $\mathcal{S}$ *is called its* power set*.*

Definition A.3 **Complement, Intersection, and Union**

The intersection *of two sets* $\mathcal{S}$ *and* $\mathcal{T}$ *is the set of all elements common to both, denoted by* $\mathcal{S} \cap \mathcal{T}$*, namely*

$$\mathcal{S} \cap \mathcal{T} = \{a : a \in \mathcal{S} \text{ and } a \in \mathcal{T}\}.$$

The union *of the two sets consists of all elements that are in* $\mathcal{S}$ *or in* $\mathcal{T}$ *(possibly both), denoted by* $\mathcal{S} \cup \mathcal{T}$*, namely*

$$\mathcal{S} \cup \mathcal{T} = \{a : a \in \mathcal{S} \text{ or } a \in \mathcal{T}\}.$$

If $\mathcal{T} \subseteq \mathcal{S}$*, then the complement of* $\mathcal{T}$ *in* $\mathcal{S}$*, denoted by* $\mathcal{S} \setminus \mathcal{T}$ *is the set of all those elements of* $\mathcal{S}$ *that are not in* $\mathcal{T}$*, namely*

$$\mathcal{S} \setminus \mathcal{T} = \{s : s \in \mathcal{S} \text{ and } s \notin \mathcal{T}\}.$$

Two sets $\mathcal{S}$ *and* $\mathcal{T}$ *are called* disjoint *if* $\mathcal{S} \cap \mathcal{T} = \varnothing$*.*

For instance, if $\mathcal{S} = \mathbb{N}$, and $\mathcal{T} = \{1, 2, 3\}$, then $\mathcal{S} \cap \mathcal{T} = \mathcal{T} = \{1, 2, 3\}$, and $\mathcal{S} \cup \mathcal{T} = \mathbb{N}$. Also, $\mathcal{S} \setminus \mathcal{T} = \{x \in \mathbb{N} : x > 3\}$.

Definition A.4 **Set Partitions**

Let $\mathcal{S}$ *be a set , and let* $\mathfrak{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots\}$ *be a set of nonempty subsets of* $\mathcal{S}$*. Then* $\mathfrak{S}$ *is called a* partition *of* $\mathcal{S}$ *provided both of the following are satisfied.*

(a) $\mathcal{S}_j \cap \mathcal{S}_k = \varnothing$ for all $j \neq k$.

(b) $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \cdots \cup \mathcal{S}_j \cdots$, namely $s \in \mathcal{S}$ if and only if $s \in \mathcal{S}_j$ for some j.

For an example of partitioning, see the notion of *congruence* on page 20.

Definition A.5 **Binary Relations and Operations**

Let s_1, s_2 *be elements of a set* $\mathcal{S}$*. Then we call* (s_1, s_2) *an* ordered pair*, where* s_1 *is called the* first component *and* s_2 *is called the* second component*. If* $\mathcal{T}$ *is*

another set, then the Cartesian product *of* $\mathcal{S}$ *with* $\mathcal{T}$*, denoted by* $\mathcal{S} \times \mathcal{T}$*, is given by the set of ordered pairs:*

$$\mathcal{S} \times \mathcal{T} = \{(s,t) : s \in \mathcal{S}, t \in \mathcal{T}\}.$$

A relation R *on* $\mathcal{S} \times \mathcal{T}$ *is a subset of* $\mathcal{S} \times \mathcal{T}$ *where* $(s,t) \in R$ *is denoted by* sRt*. A relation on* $\mathcal{S} \times \mathcal{S}$ *is called a binary relation. A relation* R *on* $(\mathcal{S} \times \mathcal{S}) \times \mathcal{S}$ *is called a binary operation on* $\mathcal{S}$ *if* R *associates with each* $(s_1, s_2) \in \mathcal{S} \times \mathcal{S}$*, a unique element* $s_3 \in \mathcal{S}$*. In other words, if* $(s_1, s_2)Rs_3$ *and* $(s_1, s_2)Rs_4$*, then* $s_3 = s_4$*.*

For example, a relation on $\mathcal{S} \times \mathcal{T} = \{1,2,3\} \times \{1,2\}$ is $\{(1,1),(1,2)\}$. Notice that there does not exist a unique second element for 1 in this relation. We cannot discuss a binary operation here since $\mathcal{S} \neq \mathcal{T}$. The next section provides us with an important notion of a binary operation.

◆ Functions

Definition A.6 *A* function f (*also called a* mapping *or* map) *from a set* $\mathcal{S}$ *to a set* $\mathcal{T}$ *is a relation on* $\mathcal{S} \times \mathcal{T}$*, denoted by* $f : \mathcal{S} \to \mathcal{T}$*, which assigns each* $s \in \mathcal{S}$ *a unique* $t \in \mathcal{T}$*, called the* image of s under f*, denoted by* $f(s) = t$*. The set* $\mathcal{S}$ *is called the* domain *of* f *and* $\mathcal{T}$ *is called the* range *of* f*. If* $\mathcal{S}_1 \subseteq \mathcal{S}$*, then the* image of $\mathcal{S}_1$ under f*, denoted by* $f(\mathcal{S}_1)$*, is the set* $\{t \in \mathcal{T} : t = f(s)$ *for some* $s \in \mathcal{S}_1\}$*. If* $\mathcal{S} = \mathcal{S}_1$*, then* $f(\mathcal{S})$ *is called the* image of f*, denoted by* $\text{img}(\mathcal{S})$*. If* $\mathcal{T}_1 \subseteq \mathcal{T}$*, the* inverse image of $\mathcal{T}_1$ under f*, denoted by* $f^{-1}(\mathcal{T}_1)$*, is the set* $\{s \in \mathcal{S} : f(s) \in \mathcal{T}_1\}$*.*

A function $f : \mathcal{S} \to \mathcal{T}$ *is called* injective (*also called* one-to-one) *if and only if for each* $s_1, s_2 \in \mathcal{S}$*,* $f(s_1) = f(s_2)$ *implies that* $s_1 = s_2$*. A function* f *is* surjective (*also called* onto) *if* $f(\mathcal{S}) = \mathcal{T}$*, namely if for each* $t \in \mathcal{T}$*,* $t = f(s)$ *for some* $s \in \mathcal{S}$*. A function* f *is called* bijective (*or a* bijection) *if it is both injective and surjective. Two sets are said to be in a* one-to-one correspondence *if there exists a bijection between them.*

Each of the following may be verified for a given function $f : \mathcal{S} \to \mathcal{T}$.

A.1. If $\mathcal{S}_1 \subseteq \mathcal{S}$, then $\mathcal{S}_1 \subseteq f^{-1}(f(\mathcal{S}_1))$.

A.2. If $\mathcal{T}_1 \subseteq \mathcal{T}$, then $f(f^{-1}(\mathcal{T}_1)) \subseteq \mathcal{T}_1$.

A.3. The identity map, $1_{\mathcal{S}} : \mathcal{S} \to \mathcal{S}$, given by $1_{\mathcal{S}}(s) = s$ for all $s \in \mathcal{S}$, is a bijection.

A.4. f is injective if and only if there exists a function $g : \mathcal{T} \to \mathcal{S}$ such that $gf = 1_{\mathcal{S}}$, and g is called a *left inverse of* f.

A.5. f is surjective if and only if there exists a function $h : \mathcal{T} \to \mathcal{S}$ such that $fh = 1_{\mathcal{T}}$, and h is called a *right inverse for* f.

A.6. If f has both a left inverse g and a right inverse h, then $g = h$ is a unique map called the *two-sided inverse of* f.

A.7. f is bijective if and only if f has a two-sided inverse.

Notice that in Definition A.5 a binary operation on $\mathcal{S}$ is just a function on $\mathcal{S} \times \mathcal{S}$. The number of elements in a set is of central importance.

Definition A.7 **Cardinality**

If $\mathcal{S}$ and $\mathcal{T}$ are sets, and there exists a one-to-one mapping from $\mathcal{S}$ to $\mathcal{T}$, then $\mathcal{S}$ and $\mathcal{T}$ are said to have the same cardinality*. A set $\mathcal{S}$ is finite if either it is empty or there is an $n \in \mathbb{N}$ and a bijection $f : \{1, 2, \ldots, n\} \mapsto \mathcal{S}$. The number of elements in a finite set $\mathcal{S}$ is sometimes called its* cardinality*, or* order*, denoted by $|\mathcal{S}|$. A set is said to be* countably infinite *if there is a bijection between the set and $\mathbb{N}$. If there is no such bijection and the set is infinite, then the set is said to be* uncountably infinite*.*

Example A.1 If $n \in \mathbb{N}$ is arbitrary and $n_0 \in \mathbb{N}$ is arbitrary but fixed, then the map $f : \mathbb{N} \mapsto n_0\mathbb{N}$ via $f(n) = n_0 n$ is bijective, so the multiples of $n_0 \in \mathbb{N}$ can be identified with $\mathbb{N}$. For instance, the case where $n_0 = 2$ shows that the even natural numbers may be identified with the natural numbers themselves.

Definition A.8 **Indexing Sets and Set Operations**

Let I be a set, which may be finite or infinite (possibly uncountably infinite), and let $\mathcal{U}$ be a universal set*, which means a set that has the property of containing all sets under consideration. We define*

$$\cup_{j \in I} \mathcal{S}_j = \{s \in \mathcal{U} : s \in \mathcal{S}_j \text{ for some } j \in I\},$$

and

$$\cap_{j \in I} \mathcal{S}_j = \{s \in \mathcal{U} : s \in \mathcal{S}_j \text{ for all } j \in I\}.$$

Here, I is called the indexing set*, $\cup_{j \in I} \mathcal{S}_j$ is called a* generalized set-theoretic union*, and $\cap_{j \in I} \mathcal{S}_j$ is called a* generalized set-theoretic intersection*.*

Example A.2 The reader may verify both of the following properties about generalized unions and intersections. In what follows, $\mathcal{T}, \mathcal{S}_j \subseteq \mathcal{U}$.

(a) $\mathcal{T} \cup (\cap_{j \in I} \mathcal{S}_j) = \cap_{j \in I} (\mathcal{T} \cup \mathcal{S}_j)$.

(b) $\mathcal{T} \cap (\cup_{j \in I} \mathcal{S}_j) = \cup_{j \in I} (\mathcal{T} \cap \mathcal{S}_j)$.

◆ Arithmetic

The natural numbers $\{1, 2, 3, 4, \ldots\}$ are denoted by $\mathbb{N}$ and the *integers* $\{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$ are denoted by $\mathbb{Z}$. See [61, pp. 1–6] for a brief history of the development of the integers.

For this we need a larger set. The following are called the *rational numbers*.

$$\mathbb{Q} = \{a/b : a, b \in \mathbb{Z}, \text{ and } b \neq 0\}.$$

Rational numbers have *periodic* decimal expansions. In other words, they have patterns that repeat *ad infinitum*. For instance, $1/2 = 0.5000\ldots$ and $1/3 = 0.333\ldots$. However, there are numbers whose decimal expansions have *no* repeated pattern, such as

$$\sqrt{2} = 1.41421356237\ldots,$$

so it is not a quotient of integers. These numbers, having decimal expansions that are not periodic, are called irrational numbers, denoted by $\mathfrak{I}$. It is possible that a sequence of rational numbers may *converge* to an irrational one. For instance, define

$$q_0 = 2, \text{ and } q_{j+1} = 1 + \frac{1}{q_j} \text{ for } j \geq 0.$$

Then

$$\lim_{j \mapsto \infty} q_j = \frac{1+\sqrt{5}}{2},$$

called the *Golden Ratio*, denoted by $\mathfrak{g}$ which we will study in this appendix. The reader familiar with Fibonacci Numbers (see page 8) will have recognized that for $j \geq 0$,

$$q_{j+1} = 1 + \frac{1}{q_j} = \frac{q_j + 1}{q_j} = \frac{F_{j+3}}{F_{j+2}},$$

so

$$\lim_{j \mapsto \infty} \frac{F_{j+3}}{F_{j+2}} = \mathfrak{g}.$$

The *real numbers* consist of the set-theoretic union:

$$\mathbb{R} = \mathbb{Q} \cup \mathfrak{I}.$$

To complete the hierarchy of numbers (at least for our purposes), the *complex numbers* employ $\sqrt{-1}$, as follows:

$$\mathbb{C} = \{a + b\sqrt{-1} : a, b \in \mathbb{R}\}.$$

We now provide the *Fundamental Laws of Arithmetic* as a fingertip reference for the convenience of the reader.

The Laws of Arithmetic:

◆ **The Laws of Closure**. If $a, b \in \mathbb{R}$, then $a + b \in \mathbb{R}$ and $ab \in \mathbb{R}$.

◆ **The Commutative Laws**. If $a, b \in \mathbb{R}$, then $a + b = b + a$, and $ab = ba$.

◆ **The Associative Laws**. If $a, b, c \in \mathbb{R}$, then $(a + b) + c = a + (b + c)$, and $(ab)c = a(bc)$.

◆ **The Distributive Law**. If $a, b, c \in \mathbb{R}$, then $a(b + c) = ab + ac$.

◆ The Cancellation Law Let $a, b, c \in \mathbb{R}$. If $a + c = b + c$, then $a = b$ for any $c \in \mathbb{R}$. Also, if $ac = bc$, then $a = b$ for any $c \in \mathbb{R}$, with $c \neq 0$.

Note that as a result of the distributive law, we may view $-a$ for any $a \in \mathbb{R}$ as $(-1) \cdot a$, or -1 *times* a.

We now look at inverses under multiplication.

◆ The Multiplicative Inverse

If $z \in \mathbb{R}$ with $z \neq 0$, then the *multiplicative inverse* of z is that number $1/z = z^{-1}$ (since $z \cdot \frac{1}{z} = 1$, the multiplicative identity). In fact, division may be considered the inverse of multiplication.

Now we look at square roots and the relationship with exponentiation.

If $a < 0$, then $\sqrt{a} \notin \mathbb{R}$. For instance, $\sqrt{-1} \notin \mathbb{R}$ and $\sqrt{-5} \notin \mathbb{R}$. Consider, $\sqrt{25} = 5 \in \mathbb{R}$. A common error is to say that $\sqrt{25} = \pm 5$, but this is **false**. The error usually arises from the confusion of the solutions to $x^2 = 25$ with the solutions to $\sqrt{5^2} = x$. Solutions to $x^2 = 25$ are certainly $x = \pm 5$, but the **only** solution to $\sqrt{5^2} = x$ is $x = 5$, the unique *positive* integer such that $x^2 = 25$. A valid way of avoiding confusion with $\sqrt{x^2}$ is the following development.

We may define *exponentiation* by observing that for any $x \in \mathbb{R}$, $n \in \mathbb{N}$,

$$x^n = x \cdot x \cdots x,$$

multiplied n times. Note that by convention $x^0 = 1$ for any nonzero real number x (and 0^0 is undefined). In what follows, the notation $\mathbb{R}^+$ means all of the *positive* real numbers. For rational exponents, we have the following.

Definition A.9 **Rational Exponents**

Let $n \in \mathbb{N}$. If n is even and $a \in \mathbb{R}^+$, then $\sqrt[n]{a} = b$ means that unique value of $b \in \mathbb{R}^+$ such that $b^n = a$. If n is even and $a \in \mathbb{R}$ with a negative, then $\sqrt[n]{a}$ is undefined. If n is odd, then $\sqrt[n]{a} = b$ is that unique value of $b \in \mathbb{R}$ such that $b^n = a$. In each case, a is called the base *for the exponent.*

Based on Definition A.9, the symbol $a^{\frac{m}{n}}$ for $a \in \mathbb{R}^+$ and $m, n \in \mathbb{N}$ is given by

$$a^{\frac{m}{n}} = \left(a^{\frac{1}{n}}\right)^m.$$

Definition A.10 **Absolute Value**

If $x \in \mathbb{R}$, then

$$|x| = \begin{cases} x & \text{if } x \geq 0, \\ -x & \text{if } x < 0, \end{cases}$$

called the absolute value *of x.*

With Definition A.10 in mind, we see that if $x > 0$, then

$$\sqrt{x^2} = (x^2)^{1/2} = (x)^{2 \cdot 1/2} = x^1 = x = |x|,$$

and if $x < 0$, then

$$\sqrt{x^2} = \sqrt{(-x)^2} = (-x)^{2 \cdot 1/2} = (-x)^1 = -x = |x|.$$

Hence,

$$\sqrt{x^2} = |x|.$$

Also,

$$a^{-\frac{m}{n}} = \frac{1}{a^{\frac{m}{n}}}.$$

In general, we have the following laws.

Theorem A.1 **Laws for Exponents**

Let $a, b \in \mathbb{R}^+$, and $n, m \in \mathbb{N}$.

(a) $a^n b^n = (ab)^n$.

(b) $a^m a^n = a^{m+n}$.

(c) $(a^m)^n = a^{mn}$.

(d) $(a^m)^{\frac{1}{n}} = \sqrt[n]{a^m} = a^{\frac{m}{n}} = (a^{\frac{1}{n}})^m$.

Proof. See [61, Proposition 1.4.1, p. 46]. □

Corollary A.1 *Let $a, n \in \mathbb{N}$. Then $\sqrt[n]{a} \in \mathbb{Q}$ if and only if $\sqrt[n]{a} \in \mathbb{Z}$.*

Note that we cannot have a *negative* base in Theorem A.1. The reason for this assertion is given in the following discussion. If we were to allow $-5 = \sqrt{25}$, then by Theorem A.1,

$$-5 = \sqrt{25} = 25^{1/2} = (5^2)^{1/2} = 5^{2 \cdot 1/2} = 5^1 = 5,$$

which is a contradiction. From another perspective, suppose that we allowed for negative bases in Theorem A.1. Then

$$5 = \sqrt{25} = \sqrt{(-5)^2} = ((-5)^2)^{1/2} = (-5)^{2 \cdot 1/2} = (-5)^1 = -5,$$

again a contradiction. Hence, only positive bases are allowed for the laws in Theorem A.1 to hold.

Since we have the operations of addition and multiplication, it would be useful to have a notation that would simplify calculations.

◆ **The Sigma Notation**

We can write $n = 1 + 1 + \cdots + 1$ for the sum of n copies of 1. We use the Greek letter upper case *sigma* to denote *summation*. For instance, $\sum_{i=1}^{n} 1 = n$ would be a simpler way of stating the above. Also, instead of writing the sum of the first one hundred natural numbers as $1 + 2 + \cdots + 100$, we may write it

as $\sum_{i=1}^{100} i$. In general, if we have numbers $a_m, a_{m+1}, \cdots, a_n$ $(m \leq n)$, we may write their sum as

$$\sum_{i=m}^{n} a_i = a_m + a_{m+1} + \cdots a_n,$$

and by convention

$$\sum_{i=m}^{n} a_i = 0 \text{ if } m > n.$$

The letter i is the *index of summation* (and any letter may be used here), n is *the upper limit of summation*, m is *the lower limit of summation*, and a_i is a *summand*. In the previous example, $\sum_{i=1}^{n} 1$, there is no i in the summand since we are adding the *same* number n times. The upper limit of summation tells us how many times that is (when $i = 1$). Similarly, we can write, $\sum_{j=1}^{4} 3 = 3 + 3 + 3 + 3 = 12$. This is the simplest application of the sigma notation. Another example is $\sum_{i=1}^{10} i = 55$.

Theorem A.2 **Properties of the Summation (Sigma) Notation**

Let $h, k, m, n \in \mathbb{Z}$ with $m \leq n$ and $h \leq k$. If R is a ring, then:

(a) If $a_i, c \in R$, then $\sum_{i=m}^{n} ca_i = c \sum_{i=m}^{n} a_i$.

(b) If $a_i, b_i \in R$, then $\sum_{i=m}^{n} (a_i + b_i) = \sum_{i=m}^{n} a_i + \sum_{i=m}^{n} b_i$.

(c) If $a_i, b_j \in R$, then

$$\sum_{i=m}^{n} \sum_{j=h}^{k} a_i b_j = \left(\sum_{i=m}^{n} a_i \right) \left(\sum_{j=h}^{k} b_j \right) = \sum_{j=h}^{k} \sum_{i=m}^{n} a_i b_j = \left(\sum_{j=h}^{k} b_j \right) \left(\sum_{i=m}^{n} a_i \right).$$

A close cousin of the summation symbol is the product symbol defined as follows.

◆ The Product Symbol

The multiplicative analogue of the summation notation is the *product symbol* denoted by Π, upper case Greek *pi*. Given $a_m, a_{m+1}, \ldots, a_n \in R$, where R is a given ring and $m \leq n$, their product is denoted by:

$$\prod_{i=m}^{n} a_i = a_m a_{m+1} \cdots a_n,$$

and by convention $\prod_{i=m}^{n} a_i = 1$ if $m > n$.

The letter i is the *product index*, m is the *lower product limit* n is the *upper product limit*, and a_i is a *multiplicand* or *factor*.

For example, if $x \in \mathbb{R}^+$, then

$$\prod_{j=0}^{n} x^j = x^{\sum_{j=0}^{n} j} = x^{n(n+1)/2},$$

(see Theorem 1.4 on page 10).

Above, we defined the product notation. For instance, $\prod_{i=1}^{7} i = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 = 5040$. This is an illustration of the following concept.

Definition A.11 **Factorial Notation!**

If $n \in \mathbb{N}$, then $n!$ (read "enn factorial") *is the product of the first n natural numbers. In other words,*

$$n! = \prod_{i=1}^{n} i.$$

We agree, by convention, that $0! = 1$. In other words, multiplication of no factors yields the identity.

The factorial notation gives us the number of distinct ways of arranging n objects. For instance, if you have ten distinct books on your bookshelf, then you can arrange them in 10!=3,628,800 distinct ways.

◆ The Pigeonhole Principle

Certain counting arguments rely upon a simple idea as follows. If n sets contain $n+1$ distinct elements in total, then at least one set must contain two or more elements. This is *the Pigeonhole Principle*, from the application of $n+1$ pigeons flying into n holes. This principle is equivalent to the *Dirichlet Box Principle*, which says if more than $m \in \mathbb{N}$ objects are placed in m boxes, then at least one of the boxes contains at least two elements.

Now that we have the factorial notation under our belts, we may introduce another symbol, based on it, which is valuable in number theory.

Definition A.12 **Binomial Coefficients**

If $k, n \in \mathbb{Z}$ with $0 \leq k \leq n$, then the symbol $\binom{n}{k}$ (read "n choose k") is given by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$

the binomial coefficient.

The binomial coefficient is used in the theory of probability as the number of different combinations of n objects taken k at a time. For instance, the number of ways of choosing two objects from a set of five objects, *without regard for order*, is $\binom{5}{2} = 5!/(2!3!) = 10$ distinct ways.

Proposition A.1 **Properties of the Binomial Coefficient**
If $n, k \in \mathbb{Z}$ and $0 \le k \le n$, then

(a) $\binom{n}{n-k} = \binom{n}{k}$. **(Symmetry Property)**

(b) $\binom{n+1}{k+1} = \binom{n}{k+1} + \binom{n}{k}$. **(Pascal's Identity)**

(c) $\sum_{i=0}^{n} (-1)^i \binom{n}{i} = 0$. **(Null Summation Property)**

(d) $\sum_{i=0}^{n} \binom{n}{i} = 2^n$. **(Full Summation Property)**

Proof. See [61, Proposition 1.2.1, pp. 18–19]. □

An important fundamental result involving binomial coefficients that we will need in the text is the following.

Theorem A.3 **The Binomial Theorem**
Let $x, y \in \mathbb{R}$, and $n \in \mathbb{N}$. Then

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^{n-i} y^i.$$

Proof. See [61, Theorem 1.2.3, p. 19]. □

Note that the full and null summation properties in Proposition A.1 are just special cases of the binomial theorem (with $x = y = 1$ and $x = 1 = -y$, respectively.)

In the above, we have used the symbols $>$ (greater than) and $<$ (less than). We now formalize this notion of ordering as follows.

Definition A.13 **Ordering**
If $a, b \in \mathbb{R}$, then we write $a < b$ if $a - b$ is negative and say that a is strictly less than *b. Equivalently, $b > a$ means that b* is strictly bigger than *a. (Thus, to say that $b - a$ is positive is equivalent to saying that $b - a > 0$.) We also write $a \le b$ to mean that $a - b$* is not positive, *namely $a - b = 0$ or $a - b < 0$. Equivalently, $b \ge a$ means that $b - a$* is nonnegative, *namely $b - a = 0$ or $b - a > 0$.*

Now we state the principle governing order.

◆ **The Law of Order** If $a, b \in \mathbb{R}$, then *exactly one* of the following must hold: $a < b$, $a = b$, or $a > b$.

A basic rule, which follows from the Law of Order, is the following.

◆ **The Transitive Law** Let $a, b, c \in \mathbb{R}$. If $a < b$ and $b < c$, then $a < c$.

What now follows easily from this is the connection between order and the operations of addition and multiplication, namely if $a < b$, then $a + c < b + c$ for any $c \in \mathbb{R}$, and $ac < bc$ for any $c \in \mathbb{R}^+$. However, if $c < 0$, then $ac > bc$.

In the text, we will be in need of some elementary facts concerning matrix theory. We now list these facts, without proof, for the convenience of the reader. The proofs, background, and details may be found in any text on elementary linear algebra.

✦ Basic Matrix Theory

If $m, n \in \mathbb{N}$, then an $m \times n$ matrix (read "m by n matrix") is a rectangular array of entries with m rows and n columns. We will assume that the entries come from a commutative ring with identity R (see page 23). If A is such a matrix, and $a_{i,j}$ denotes the entry in the i^{th} row and j^{th} column, then

$$A = (a_{i,j}) = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots a_{2,n} \\ \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots a_{m,n} \end{pmatrix}.$$

Two $m \times n$ matrices $A = (a_{i,j})$, and $B = (b_{i,j})$ are equal if and only if $a_{i,j} = b_{i,j}$ for all i and j. The matrix $(a_{j,i})$ is called the *transpose* of A, denoted by

$$A^t = (a_{j,i}).$$

Addition of two $m \times n$ matrices A and B is done in the natural way.

$$A + B = (a_{i,j}) + (b_{i,j}) = (a_{i,j} + b_{i,j}),$$

and if $r \in R$, then $rA = r(a_{i,j}) = (ra_{i,j})$, called *scalar multiplication*, which is used most often in practice for $R = \mathbb{R}$.

Under the above definition of addition and scalar multiplication, the set of all $m \times n$ matrices with entries from R, a commutative ring with identity, form a set, denoted by $\mathcal{M}_{m\times n}(R)$. When $m = n$, this set is in fact a ring given by the following.

If $A = (a_{i,j})$ is an $m \times n$ matrix and $B = (b_{j,k})$ is an $n \times r$ matrix, then the *product* of A and B is defined as the $m \times r$ matrix:

$$AB = (a_{i,j})(b_{j,k}) = (c_{i,k}),$$

where

$$c_{i,k} = \sum_{\ell=1}^{n} a_{i,\ell} b_{\ell,k}.$$

Multiplication, if defined, is associative, and distributive over addition. If $m = n$, then $\mathcal{M}_{n\times n}(R)$ is a ring, with identity given by the $n \times n$ matrix:

$$I_n = \begin{pmatrix} 1_R & 0 & \cdots & 0 \\ 0 & 1_R & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1_R \end{pmatrix},$$

called *the $n \times n$ identity matrix*, where 1_R is the identity of R.

Another important aspect of matrices that we will need throughout the text is motivated by the following. We maintain the assumption that R is a commutative ring with identity. Let $(a, b), (c, d) \in \mathcal{M}_{1\times 2}(R)$. If we set up these row vectors into a single 2×2 matrix

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

then $ad - bc$ is called the *determinant* of A, denoted by $\det(A)$. More generally, we may define the determinant of any $n \times n$ matrix in $\mathcal{M}_{n\times n}(R)$ for any $n \in \mathbb{N}$. The determinant of any $r \in \mathcal{M}_{1\times 1}(R)$ is just $\det(r) = r$. Thus, we have the definitions for $n = 1, 2$, and we may now give the general definition inductively. The definition of the determinant of a 3×3 matrix

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

is defined in terms of the above definition of the determinant of a 2×2 matrix, namely $\det(A)$ is given by

$$a_{1,1} \det \begin{pmatrix} a_{2,2} & a_{2,3} \\ a_{3,2} & a_{3,3} \end{pmatrix} - a_{1,2} \det \begin{pmatrix} a_{2,1} & a_{2,3} \\ a_{3,1} & a_{3,3} \end{pmatrix} + a_{1,3} \det \begin{pmatrix} a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix}.$$

Therefore, we may inductively define the determinant of any $n \times n$ matrix in this fashion. Assume that we have defined the determinant of an $n \times n$ matrix. Then we define the determinant of an $(n + 1) \times (n + 1)$ matrix $A = (a_{i,j})$ as follows. First, we let $A_{i,j}$ denote the $n \times n$ matrix obtained from A by deleting the i^{th} row and j^{th} column. Then we define the *minor* of $A_{i,j}$ at position (i, j) to be $\det(A_{i,j})$. The *cofactor* of $A_{i,j}$ is defined to be

$$\operatorname{cof}(A_{i,j}) = (-1)^{i+j} \det(A_{i,j}).$$

We may now define the determinant of A by

$$\det(A) = a_{i,1}\operatorname{cof}(A_{i,1}) + a_{i,2}\operatorname{cof}(A_{i,2}) + \cdots + a_{i,n+1}\operatorname{cof}(A_{i,n+1}). \tag{A.1}$$

This is called the *expansion of a determinant by cofactors* along the i^{th} row of A. Similarly, we may expand along a column of A.

$$\det(A) = a_{1,j}\operatorname{cof}(A_{1,j}) + a_{2,j}\operatorname{cof}(A_{2,j}) + \cdots + a_{n+1,j}\operatorname{cof}(A_{n+1,j}),$$

called the *cofactor expansion along the j^{th} column of A*. Both expansions can be shown to be identical. Hence, a determinant may be viewed as a function that assigns a real number to an $n \times n$ matrix, and the above gives a method for finding that number. Other useful properties of determinants that we will have occasion to use in the text are given in the following.

Theorem A.4 **Properties of Determinants**

Let R be a commutative ring with identity and let $A = (a_{i,j})$, $B = (b_{i,j}) \in \mathcal{M}_{n\times n}(R)$. Then each of the following hold.

(a) $\det(A) = \det(a_{i,j}) = \det(a_{j,i}) = \det(A^t)$.

(b) $\det(AB) = \det(A)\det(B)$.

(c) *If matrix A is achieved from matrix B by interchanging two rows (or two columns), then* $\det(A) = -\det(B)$.

(d) *If $\mathcal{S}_n$ is the symmetric group on n symbols, then*

$$det(A) = \sum_{\sigma\in\mathcal{S}_n} (\operatorname{sgn}(\sigma))a_{1,\sigma(1)}a_{2,\sigma(2)}\cdots a_{n,\sigma(n)},$$

where $\operatorname{sgn}(\sigma)$*, is* 1 *or* -1 *according as σ is even or odd.*

If $A \in \mathcal{M}_{n\times n}(R)$, then A is said to be *invertible*, or *nonsingular* if there is a unique matrix denoted by

$$A^{-1} \in M_{n\times n}(R)$$

such that

$$AA^{-1} = I_n = A^{-1}A.$$

Here are some properties of invertible matrices.

Theorem A.5 **Properties of Invertible Matrices**

Let R be a commutative ring with identity, $n \in \mathbb{N}$, and A invertible in $\mathcal{M}_{n\times n}(R)$. Then each of the following holds.

(a) $(A^{-1})^{-1} = A$.

(b) $(A^t)^{-1} = (A^{-1})^t$*, where "t" denotes the transpose.*

(c) $(AB)^{-1} = B^{-1}A^{-1}$

In order to provide a formula for the inverse of a given matrix, we need the following concept.

Definition A.14 **Adjoint**

Let R be a commutative ring with identity. If $A = (a_{i,j}) \in \mathcal{M}_{n\times n}(R)$, then the matrix

$$A^a = (b_{i,j})$$

given by

$$b_{i,j} = (-1)^{i+j}\det(A_{j,i}) = \operatorname{cof}(A_{j,i}) = \left[(-1)^{i+j}\det(A_{i,j})\right]^t$$

is called the adjoint of A.

Some properties of adjoints related to inverses, including a formula for the inverse, are as follows. In what follows, a *unit* or *invertible element* u in a commutative ring with identity R means an element for which there exists a multiplicative inverse. In other words, an element $u \in R$ is a unit if there exists an element $u^{-1} \in \mathbb{R}$ such that $uu^{-1} = 1_R$.

Theorem A.6 **Properties of Adjoints**

If R is a commutative ring with identity and $A \in \mathcal{M}_{n\times n}(R)$, then each of the following holds.

(a) $AA^a = \det(A)I_n = A^aA$.

(b) *A is invertible in $\mathcal{M}_{n\times n}(R)$ if and only if* $\det(A)$ *is a unit in R, in which case* $A^{-1} = A^a/\det(A)$.

Example A.3 If $n = 2$, then the inverse of a nonsingular matrix

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

is given by

$$A^{-1} = \begin{pmatrix} \frac{d}{\det(A)} & \frac{-b}{\det(A)} \\ \frac{-c}{\det(A)} & \frac{a}{\det(A)} \end{pmatrix}.$$

✦ Polynomials and Polynomial Rings

If R is a ring, then a *polynomial* $f(x)$ in an *indeterminant* x with *coefficients* in R is an infinite formal sum

$$f(x) = \sum_{j=0}^{\infty} a_j x^j = a_0 + a_1 x + \cdots + a_n x^n + \cdots,$$

where the *coefficients* a_j are in R for $j \geq 0$ and $a_j = 0$ for all but a finite number of those values of j. The set of all such polynomials is denoted by $R[x]$. If $a_n \neq 0$, and $a_j = 0$ for $j > n$, then a_n is called the *leading coefficient* of $f(x)$. If the leading coefficient $a_n = 1_R$, in the case where R is a commutative ring with identity 1_R, then $f(x)$ is said to be *monic*.

We may add two polynomials from $R[x]$, $f(x) = \sum_{j=0}^{\infty} a_j x^j$ and $g(x) = \sum_{j=0}^{\infty} b_j x^j$, by

$$f(x) + g(x) = \sum_{j=0}^{\infty} (a_j + b_j)x^j \in R[x],$$

and multiply them by

$$f(x)g(x) = \sum_{j=0}^{\infty} c_j x^j,$$

where

$$c_j = \sum_{i=0}^{j} a_i b_{j-i}.$$

Also, $f(x) = g(x)$ if and only if $a_j = b_j$ for all $j = 0, 1, \ldots$. Under the above operations $R[x]$ is a ring, called the *polynomial ring over R in the indeterminant x*. Furthermore, if R is commutative, then so is $R[x]$, and if R has identity 1_R, then 1_R is the identity for $R[x]$. Notice that with these conventions, we may write $f(x) = \sum_{j=0}^{n} a_j x^j$, for some $n \in \mathbb{N}$, where a_n is the leading coefficient since we have tacitly agreed to "ignore" zero terms.

If $\alpha \in R$, we write $f(\alpha)$ to represent the element $\sum_{j=0}^{n} a_j \alpha^j \in R$, called the *substitution* of α for x. When $f(\alpha) = 0$, then α is called a *root* of $f(x)$. The substitution gives rise to a mapping $\overline{f} : R \mapsto R$ given by $\overline{f} : \alpha \mapsto f(\alpha)$, which is determined by $f(x)$. Thus, $\overline{f}$ is called a *polynomial function* over R.

✦ Characteristic of a Ring

The characteristic of a ring R is the smallest $n \in \mathbb{N}$ (if there is one) such that $n \cdot r = 0$ for all $r \in R$. If there is no such n, then R is said to have characteristic 0. Any field containing $\mathbb{Q}$ has characteristic zero, while any field containing the finite field $\mathbb{F}_p$ for a prime p has characteristic p (see the discussion following Definition A.17 below).

Definition A.15 **Degrees of Polynomials**

If $f(x) \in R[x]$, with $f(x) = \sum_{j=0}^{d} a_j x^j$, and $a_d \neq 0$, then $d \geq 0$ is called the degree of $f(x)$ over R, *denoted by* $\deg_R(f)$. *If no such d exists, we write $\deg_R(f) = -\infty$, in which case $f(x)$ is the zero polynomial in $R[x]$ (for instance, see Example A.5 below). If F is a field of characteristic zero, then*

$$\deg_{\mathbb{Q}}(f) = \deg_F(f)$$

for any $f(x) \in \mathbb{Q}[x]$. If F has characteristic p, and $f(x) \in \mathbb{F}_p[x]$, then

$$\deg_{\mathbb{F}_p}(f) = \deg_F(f).$$

In either case, we write $\deg(f)$ *for* $\deg_F(f)$, *without loss of generality, and call this* the degree of $f(x)$.

With respect to roots of polynomials, the following is important.

Definition A.16 **Discriminant of Polynomials**

Let $f(x) = a\prod_{j=1}^{n}(x - \alpha_j) \in F[x]$, $\deg(f) = n > 1$, $a \in F$ a field in $\mathbb{C}$, where $\alpha_j \in \mathbb{C}$ are all the roots of $f(x) = 0$ for $j = 1, 2, \ldots, n$. Then the discriminant of f *is given by*

$$\operatorname{disc}(f) = a^{2n-2} \prod_{1 \leq i < j \leq n} (\alpha_j - \alpha_i)^2.$$

From Definition A.16, we see that f has a multiple root in $\mathbb{C}$ (namely for some $i \neq j$ we have $\alpha_i = \alpha_j$, also called a *repeated root*) if and only if $\text{disc}(f) = 0$.

Example A.4 If $f(x) = ax^2 + bx + c$ where $a, b, c \in \mathbb{Z}$, then $\text{disc}(f) = b^2 - 4ac$ and if $f(x) = x^3 - c$, then $\text{disc}(f) = -27c^2$.

Definition A.17 **Division of Polynomials**

We say that a polynomial $g(x) \in R[x]$ divides $f(x) \in R[x]$, *if there exists an* $h(x) \in R[x]$ *such that* $f(x) = g(x)h(x)$. *We also say that* $g(x)$ *is a* factor *of* $f(x)$.

Definition A.18 **Irreducible Polynomials over Rings**

A polynomial $f(x) \in R[x]$ *is called* irreducible (*over* R) *if* $f(x)$ *is not a unit in* R *and any factorization* $f(x) = g(x)h(x)$, *with* $g(x), h(x) \in R[x]$ *satisfies the property that one of* $g(x)$ *or* $h(x)$ *is in* R, *called a* constant polynomial. *In other words,* $f(x)$ *cannot be the product of two nonconstant polynomials. If* $f(x)$ *is not irreducible, then it is said to be* reducible.

Note that it is possible that a reducible polynomial $f(x)$ could be a product of two polynomials of the *same degree* as that of f. For instance, $f(x) = (1 - x) = (2x + 1)(3x + 1)$ in $R = \mathbb{Z}/6\mathbb{Z}$.

In general, it is important to make the distinction between degrees of a polynomial over various rings, since the base ring under consideration may alter the makeup of the polynomial.

For the following example, recall that a *finite field* is a field with a finite number of elements $n \in \mathbb{N}$, denoted by $\mathbb{F}_n$. In general, if K is a finite field, then $K = \mathbb{F}_{p^m}$ for some prime p and $m \in \mathbb{N}$, also called *Galois fields*. The field $\mathbb{F}_p$ is called the *prime subfield* of K. In general, a prime subfield is a field having no proper subfields, so $\mathbb{Q}$ is the prime subfield of any field of characteristic 0 and $\mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p$ is the prime field of any field $K = \mathbb{F}_{p^m}$. In the following result, the term *cyclic* in reference to a multiplicative abelian group G means that a group generated by some $g \in G$ *coincides* with G. Note that any group of prime order is cyclic and the product of two cyclic groups of relatively prime order is also a cyclic group. Also, if $\mathcal{S}$ is a nonempty subset of a group G, then the intersection of all subgroups of G containing $\mathcal{S}$ is called the subgroup *generated* by $\mathcal{S}$..

Theorem A.7 **Multiplicative Subgroups of Fields**

If F *is any field and* F^* *is a finite subgroup of the multiplicative subgroup of nonzero elements of* F, *then* F^* *is cyclic. In particular, if* $F = \mathbb{F}_{p^n}$ *is a finite field, then* F^* *is a finite cyclic group.*

Example A.5 The polynomial $f(x) = 2x^2 + 2x + 2$ is of degree two over $\mathbb{Q}$. However, over $\mathbb{F}_2$, $\deg_{\mathbb{F}_2}(f) = -\infty$, since f is the zero polynomial in $\mathbb{F}_2[x]$.

Some facts concerning irreducible polynomials will be needed in the text as follows.

Theorem A.8 **Irreducible Polynomials Over Finite Fields**

The product of all monic irreducible polynomials over a finite field $\mathbb{F}_q$ whose degrees divide a given $n \in \mathbb{N}$ is equal to $x^{q^n} - x$.

Based upon Theorem A.8, the following may be used as an algorithm for testing polynomials for irreducibility over prime fields and thereby generate irreducible polynomials.

Corollary A.2 *The following are equivalent.*

(a) f is irreducible over $\mathbb{F}_p$, where p is prime, and $\deg_{\mathbb{F}_p}(f) = n$.

(b) $\gcd(f(x), x^{p^i} - x) = 1$ for all natural numbers $i \leq \lfloor n/2 \rfloor$.

The following is also a general result concerning irreducible polynomials over any field.

Theorem A.9 **Irreducible Polynomials Over Arbitrary Fields**

Let F be a field and $f(x) \in F[x]$. Denote by $(f(x))$ the principal ideal in $F[x]$ generated by $f(x)$ (see Definition A.22 on page 317). Then the following are equivalent.

(a) f is irreducible over F.

(b) $F[x]/(f(x))$ is a field.

Another useful result is the following.

Theorem A.10 **Polynomials, Traces, and Norms**

Suppose that $f(x) \in R[x]$ is a monic, irreducible polynomial (over R where R is an integral domain), $\deg(f) = d \in \mathbb{N}$, and α_j for $j = 1, 2, \ldots, d$ are all of the roots of $f(x)$ in $\mathbb{C}$. Then

$$f(x) = x^d - Tx^{d-1} + \cdots \pm N,$$

where

$$T = \sum_{j=1}^{d} \alpha_j \text{ and } N = \prod_{j=1}^{d} \alpha_j,$$

where T is called the trace *and N is called the* norm *(of any of the roots of $f(x)$).*

Now that we have the notion of irreducibility for polynomials, we may state a unique factorization result for polynomials over fields.

Theorem A.11 **Unique Factorization for Polynomials**

If F is a field, then every nonconstant polynomial $f(x) \in F[x]$ can be factored in $F[x]$ into a product of irreducible polynomials $p(x)$, each of which is unique up to order and units (nonzero constant polynomials) in F.

The Euclidean Algorithm applies to polynomials in a way that allows us to talk about common divisors of polynomials in a fashion similar to that for integers.

Definition A.19 **The GCD of Polynomials**

If $f_i(x) \in F[x]$ for $i = 1, 2$, where F is a field, then the greatest common divisor of $f_1(x)$ and$f_2(x)$ *is the* unique *monic polynomial $g(x) \in F[x]$ satisfying both:*

(a) *For $i = 1, 2$, $g(x)|f_i(x)$.*

(b) *If there is a $g_1(x) \in F[x]$ such that $g_1(x)|f_i(x)$ for $i = 1, 2$, then $g_1(x)|g(x)$.*

If $g(x) = 1$, we say that $f_1(x)$ and $f_2(x)$ are relatively prime*, or* coprime *denoted by*

$$\gcd(f_1(x), f_2(x)) = 1.$$

There is also a Euclidean result for polynomials over a field.

Theorem A.12 **Euclidean Algorithm for Polynomials**

If $f(x), g(x) \in F[x]$, where F is a field, and $g(x) \neq 0$, there exist unique $q(x), r(x) \in F[x]$ such that

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

where $\deg(r) < \deg(g)$. *(Note that if $r(x) = 0$, the zero polynomial, then* $\deg(r) = -\infty$.)

Finally, if $f(x)$ and $g(x)$ are relatively prime, there exist $s(x), t(x) \in F[x]$ such that

$$1 = s(x)f(x) + t(x)g(x).$$

Theorem A.13 **Lagrange's Theorem**

Suppose that p is a prime and $f(x) \in \mathbb{Z}[x]$ is polynomial of degree $d \geq 1$ modulo p. Then

$$f(x) \equiv 0 \,(\mathrm{mod}\ p)$$

has at most d incongruent solutions.

Proof. See [61, Theorem 2.51, p. 104]. □

We will need the following important polynomial in the main text.

Definition A.20 **Cyclotomic Polynomials**

If $n \in \mathbb{N}$, then the n^{th} cyclotomic polynomial is given by

$$\Phi_n(x) = \prod_{\substack{\gcd(n,j)=1 \\ 1 \le j < n}} (x - \zeta_n^j).$$

Also, the degree of $\Phi_n(x)$ is $\phi(n)$, the Euler Totient (see Definition 1.12).

Note that despite the form of the cyclotomic polynomial given in Definition A.20, it can be shown that $\Phi_n(x) \in \mathbb{Z}[x]$. The reader may think of the term *cyclotomic* as "circle dividing," since the n^{th} roots of unity divide the unit circle into n equal arcs. Also, the ζ_n^j are sometimes called *De Moivre Numbers* (see [92, p. 388]).

Biography 8.1 Abraham De Moivre (1667–1754) *was a French-born Huguenot who left for England when Louis XIV revoked the Edict of Nantes in 1685. He was one of the pioneers of the theory of probability in the early eighteenth century. He became acquainted with Newton and Halley when he went to England. However, as a Frenchman, he was unable to secure a university position there and remained mostly self-supporting through fees for tutorial services. Yet he produced a considerable amount of research, perhaps the most famous of which is his* Doctrine of Chances *first published in 1718. This and subsequent editions had more than fifty problems on probability. Perhaps the most famous theorem with De Moivre's name attached to it is the one that says: For a, b coordinates in the complex plane, r the radius and ϕ the angle that the radius vector makes with the real axis, $(a + bi)^n = r^n(\cos(n\phi) + i\sin(n\phi))$.*

The following section is of importance for us in the main text as a tool for the description of numerous cryptographic devices (see page 23).

✦ **Action on Rings**

Definition A.21 **Morphisms of Rings**

If R and S are two rings and $f : R \to S$ is a function such that $f(ab) = f(a)f(b)$, and $f(a + b) = f(a) + f(b)$ for all $a, b \in R$, then f is called a ring homomorphism. *If, in addition, $f : R \to S$ is an injection as a map of sets, then f is called a* ring monomorphism. *If a ring homomorphism f is a surjection as*

a map of sets, then f is called a ring epimorphism. *If a ring homomorphism f is a bijection as a map of sets, then f is called a* ring isomorphism, *and R is said to be* isomorphic *to S, denoted by $R \cong S$. Lastly,* $\ker(f) = \{s \in S : f(s) = 0\}$ *is called the* kernel of f. *Also, f is injective if and only if* $\ker(f) = \{0\}$.

There is a fundamental result that we will need in the text. In order to describe it, we need the following notion.

Definition A.22 **Ideal, Cosets, and Quotient Rings**

An ideal I *in a commutative ring R with identity is a subring of R satisfying the additional property that $rI \subseteq I$ for all $r \in R$. If I is an ideal in R then a* coset *of I in R is a set of the form $r + I = \{r + \alpha : \alpha \in I\}$ where $r \in R$. The set*

$$R/I = \{r + I : r \in R\}$$

becomes a ring under multiplication and addition of cosets given by

$$(r + I)(s + I) = rs + I, \text{ and } (r + I) + (s + I) = (r + s) + I,$$

for any $r, s \in R$ (and this can be shown to be independent of the representatives r and s). R/I is called the quotient ring of R by I, *or the* factor ring of R by I, *or the* residue class ring modulo I. *The cosets are called the* residue classes modulo I. *A mapping*

$$f : R \mapsto R/I,$$

which takes elements of R to their coset representatives in R/I, is called the natural map *of R to R/I, and it is easily seen to be an epimorphism. The cardinality of R/I is denoted by $|R : I|$.*

Example A.6 Consider the ring of integers modulo $n \in \mathbb{N}$, $\mathbb{Z}/n\mathbb{Z}$ introduced in Definition 1.10. Then $n\mathbb{Z}$ is an ideal in $\mathbb{Z}$, and the quotient ring is the residue class ring modulo n.

Remark A.1 *Since rings are also groups, then the above concept of cosets and quotients specializes to groups. In particular, we have the following. Note that an index of a subgroup H in a group G can be defined similarly to the above situation for rings as follows. The* index *of H in G, denoted by $|G : H|$, is the cardinality of the set of distinct right (respectively left) cosets of H in G. Our principal interest is when this cardinality is finite (so this allows us to access the definition of cardinality given earlier). Then* Lagrange's Theorem for groups *says that*

$$|G| = |G : H| \cdot |H|,$$

so if G is a finite group, then $|H| \mid |G|$. In particular, a finite abelian group G has subgroups of all orders dividing $|G|$.

Now we are in a position to state the important result for rings. The reader unfamiliar with the notation "img" of a function should consult Definition A.6 on page 300 for the description.

Theorem A.14 **Fundamental Isomorphism Theorem for Rings**

If R and S are commutative rings with identity, and

$$\phi : R \to S$$

is a homomorphism of rings, then

$$\frac{R}{\ker(\phi)} \cong \text{img}(\phi).$$

Example A.7 If $\mathbb{F}_q$ is a finite field where $q = p^n$ (p prime) and $f(x) \in \mathbb{F}_p[x]$ is an irreducible polynomial of degree n (see page 311), then

$$\mathbb{F}_q \cong \frac{\mathbb{F}_p[x]}{(f(x))}.$$

The situation in Example A.7 is related to the following definition and theorem.

Definition A.23 **Maximal and Proper Ideals**

Let R be a commutative ring with identity. An ideal $I \neq R$ is called maximal *if whenever $I \subseteq J$, where J is an ideal in R, then $I = J$ or $I = R$. (An ideal $I \neq R$ is called a* proper *ideal.)*

Theorem A.15 **Rings Modulo Maximal Ideals**

If R is a commutative ring with identity, then M is a maximal ideal in R if and only if R/M is a field.

Example A.8 If F is a field and $r \in F$ is a fixed nonzero element, then

$$I = \{f(x) \in F[x] : f(r) = 0\}$$

is a maximal ideal and

$$F \cong F[x]/I.$$

Another aspect of rings that we will need in the text is the following. If $\mathcal{S} = \{R_j : j = 1, 2, \ldots, n\}$ is a set of rings, then let R be the set of n-tuples $(r_1, r_2, \ldots, r_n)$ with $r_j \in R_j$ for $j = 1, 2, \ldots n$, with the *zero element* of R being the n-tuple, $(0, 0, \ldots, 0)$. Define addition in R by

$$(r_1, r_2, \ldots, r_n) + (r'_1, r'_2, \ldots, r'_n) = (r_1 + r'_1, r_2 + r'_2, \ldots, r_n + r'_n),$$

for all $r_j, r'_j \in R_j$ with $j = 1, 2, \ldots, n$, and multiplication by

$$(r_1, r_2, \ldots, r_n)(r'_1, r'_2, \ldots, r'_n) = (r_1 r'_1, r_2 r'_2, \ldots, r_n r'_n).$$

This defines a structure on R called the *direct sum* of the rings R_j, $j = 1, 2, \ldots, n$, denoted by

$$\oplus_{j=1}^{n} R_j = R_1 \oplus \cdots \oplus R_n, \tag{A.2}$$

which is easily seen to be a ring. Similarly, when the R_j are groups, then this is a direct sum of groups, which is again a group.

In the text, we will have occasion to refer to such items as vector spaces, so we remind the reader of the definition. The reader is referred to pages 22–25, where we discussed the axioms for algebraic objects such as groups, rings, and fields. In particular, for the sake of completeness, note that any set satisfying all of the axioms of Theorem 1.10 on page 22, except (g), is called a *division ring*.

✦ Vector Spaces

A *vector space* consists of an additive abelian group V and a field F together with an operation called *scalar multiplication* of each element of V by each element of F on the left, such that for each $r, s \in F$ and each $\alpha, \beta \in V$ the following conditions are satisfied:

A.1. $r\alpha \in V$.

A.2. $r(s\alpha) = (rs)\alpha$.

A.3. $(r + s)\alpha = (r\alpha) + (s\alpha)$.

A.4. $r(\alpha + \beta) = (r\alpha) + (r\beta)$.

A.5. $1_F\alpha = \alpha$.

The set of elements of V are called *vectors* and the elements of F are called *scalars*. The generally accepted abuse of language is to say that V is a *vector space over* F. If V_1 is a subset of a vector space V that is a vector space in its own right, then V_1 is called a *subspace of* V.

Example A.9 For a given prime p, $m, n \in \mathbb{N}$, the finite field $\mathbb{F}_{p^n}$ is an n-dimensional vector space over $\mathbb{F}_{p^m}$ with p^{mn} elements.

Definition A.24 **Bases, Dependence, and Finite Generation**

If $\mathbb{S}$ is a subset of a vector space V, then the intersection of all subspaces of V containing $\mathbb{S}$ is called the subspace generated by $\mathbb{S}$, *or* spanned by $\mathbb{S}$. *If there is a finite set $\mathbb{S}$, and $\mathbb{S}$ generates V, then V is said to be* finitely generated. *If $\mathbb{S} = \varnothing$, then $\mathbb{S}$ generates the zero vector space. If $\mathbb{S} = \{m\}$, a singleton set, then the subspace generated by $\mathbb{S}$ is said to be the* cyclic subspace generated by m.

A subset $\mathbb{S}$ of a vector space V is said to be linearly independent *provided that for distinct $s_1, s_2, \ldots, s_n \in \mathbb{S}$, and $r_j \in V$ for $j = 1, 2, \ldots, n$,*

$$\sum_{j=1}^{n} r_j s_j = 0 \text{ implies that } r_j = 0 \text{ for } j = 1, 2, \ldots, n.$$

If $\mathbb{S}$ is not linearly independent, then it is called linearly dependent. *A linearly independent subset of a vector space that spans V is called a* basis *for V.*

In the text, we will have need of the following notion, especially as it pertains to the infinite binary case.

✦ **Sequences**

Definition A.25 *A sequence is a function whose domain is $\mathbb{N}$, with images denoted by a_n, called the n^{th}* term of the sequence. *The entire sequence is denoted by $\{a_n\}_{n=1}^{\infty}$, or simply $\{a_n\}$, called an infinite sequence or simply a* sequence. *If $\{a_n\}$ is a sequence, and $L \in \mathbb{R}$ such that*

$$\lim_{n \to \infty} a_n = L,$$

then the sequence is said to converge (*namely when the limit exists*) *whereas sequences that have no such limit are said to* diverge. *If the terms of the sequence are nondecreasing, $a_n \leq a_{n+1}$ for all $n \in \mathbb{N}$, or nonincreasing, $a_n \geq a_{n+1}$ for all $n \in \mathbb{N}$, then $\{a_n\}$ is said to be* monotonic. *A sequence $\{a_n\}$ is called* bounded above *if there exists an $M \in \mathbb{R}$ such that $a_n \leq M$ for all $n \in \mathbb{N}$. The value M is called an* upper bound *for the sequence. A sequence $\{a_n\}$ is called* bounded below *if there is a $B \in \mathbb{R}$ such that $B \leq a_n$ for all $n \in \mathbb{N}$, and B is called a* lower bound *for the sequence. A sequence $\{a_n\}$ is called* bounded *if it bounded above and bounded below.*

Some fundamental facts concerning sequences are contained in the following.

Theorem A.16 **Properties of Sequences** *Let $\{a_n\}$ and $\{b_n\}$ be sequences. Then*

(a) *If $\{a_n\}$ is bounded and monotonic, then it converges.*

(b) *If* $\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n = L \in \mathbb{R}$, *and* $\{c_n\}$ *is a sequence such that there exists a natural number* N *with* $a_n \le c_n \le b_n$ *for all* $n > N$, *then* $\lim_{n\to\infty} c_n = L$.

(c) *If* $\lim_{n\to\infty} |a_n| = 0$, *then* $\lim_{n\to\infty} a_n = 0$.

Note that part (c) of Theorem A.16 is a corollary to part (b).

◆ Continued Fractions

Proofs for the results in this section can be found in [61, pp. 221–272].

Definition A.26 *If* $q_j \in \mathbb{R}$ *where* $j \in \mathbb{Z}$ *is nonnegative and* $q_j \in \mathbb{R}^+$ *for* $j > 0$, *then an expression of the form*

$$\alpha = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \ddots + \cfrac{1}{q_k + \cfrac{1}{q_{k+1} + \ddots}}}}$$

is called a continued fraction. *If* $q_k \in \mathbb{Z}$ *for all* $k \ge 0$, *then it is called a* simple continued fraction, *denoted by* $\langle q_0; q_1, \ldots, q_k, q_{k+1}, \ldots\rangle$. *If there exists a nonnegative integer* n *such that* $q_k = 0$ *for all* $k \ge n$, *then the continued fraction is called* finite. *If no such* n *exists, then it is called* infinite.

Note that the classical definition of a *simple* continued fraction is a continued fraction that arises from the reciprocals as in the Euclidean Algorithm, so the "numerators" are all 1 and the denominators all integers. This is to distinguish from more general continued fractions in which the numerators and denominators can be functions of a complex variable, for instance. Simple continued fractions are also called *regular* continued fractions in the literature.

Definition A.27 **Convergents**

Let $n \in \mathbb{N}$ *and let* α *have continued fraction expansion* $\langle q_0; q_1, \ldots, q_n, \ldots\rangle$ *for* $q_j \in \mathbb{R}^+$ *when* $j > 0$. *Then*

$$C_k = \langle q_0; q_1, \ldots, q_k\rangle$$

is the k^{th} convergent *of* α *for any nonnegative integer* k.

Theorem A.17 **Finite Simple Continued Fractions are Rational**

Let $\alpha \in \mathbb{R}$. Then $\alpha \in \mathbb{Q}$ if and only if α can be written as a finite simple continued fraction.

Theorem A.18 **Representation of Convergents**

Let $\alpha = \langle q_0; q_1, \ldots, q_n, \ldots \rangle$ for $n \in \mathbb{N}$ be a continued fraction expansion. Define two sequences for $k \in \mathbb{Z}$ nonnegative:

$$A_{-2} = 0, A_{-1} = 1, A_k = q_k A_{k-1} + A_{k-2},$$

and

$$B_{-2} = 1, B_{-1} = 0, B_k = q_k B_{k-1} + B_{k-2}.$$

Then

$$C_k = A_k/B_k = \frac{q_k A_{k-1} + A_{k-2}}{q_k B_{k-1} + B_{k-2}}$$

is the k^{th} convergent of α for any nonnegative integer $k \leq n$.

Theorem A.19 **Irrationals Are Infinite Simple Continued Fractions**

Let $\alpha \in \mathbb{R}$. Then α is irrational if and only if α has a unique infinite simple continued fraction expansion *$\alpha = \alpha_0 = \langle q_0; q_1, \ldots \rangle = \lim_{k\to\infty} C_k$, where $q_{k-1} = \lfloor \alpha_{k-1} \rfloor$ with $\alpha_k = 1/(\alpha_{k-1} - q_{k-1})$ and $C_k = A_k/B_k$ for $k \in \mathbb{N}$.*

Note that in what follows, a *surd* is the square root of an integer that is not a perfect square. The term *surd* is actually an archaic term for *square root*. The term *quadratic surd* also refers to the objects introduced in Definition A.30 on the next page.

Theorem A.20 **Convergents of Surds**

Suppose that $D > 0$ is not a perfect square, $n \in \mathbb{Z}$, and $|n| < \sqrt{D}$. If (x, y) is a positive solution *of $x^2 - Dy^2 = n$, namely $x, y \in \mathbb{N}$, then x/y is a convergent in the simple continued fraction expansion of $\sqrt{D}$.*

Definition A.28 **Periodic Continued Fractions**

An infinite simple continued fraction $\alpha = \langle q_0; q_1, q_2, \ldots \rangle$ is called periodic *if there exists an integer $k \geq 0$ and $\ell \in \mathbb{N}$ such that $q_n = q_{n+\ell}$ for all integers $n \geq k$. We use the notation*

$$\alpha = \langle q_0; q_1, \ldots, q_{k-1}, \overline{q_k, q_{k+1}, \ldots, q_{\ell+k-1}} \rangle,$$

as a convenient abbreviation. The smallest such natural number $\ell = \ell(\alpha)$ is called the period length *of α, and $q_0, q_1, \ldots, q_{k-1}$ is called the* pre-period *of α. If k is the* least *nonnegative integer such that $q_n = q_{n+\ell}$ for all $n \geq k$, then $q_k, q_{k+1}, \ldots, q_{k+\ell-1}$ is called the* fundamental period *of α. If $k = 0$ is the least such value, then α is said to be* purely periodic, *namely $\alpha = \langle \overline{q_0; q_1, \ldots, q_{\ell-1}} \rangle$.*

In order to introduce the next concept, we need the following notion.

Definition A.29 **Discriminants**

Let $D_0 \neq 1$ be a square-free integer, and set

$$\Delta_0 = \begin{cases} D_0 & \text{if } D_0 \equiv 1 \pmod 4, \\ 4D_0 & \text{otherwise .} \end{cases}$$

Then Δ_0 is called a fundamental discriminant *with associated* fundamental radicand *D_0. Let $f_\Delta \in \mathbb{N}$, and set $\Delta = f_\Delta^2 \Delta_0$. Then*

$$\Delta = \begin{cases} D & \text{if } D \equiv 1 \pmod 4 \text{ and } f_\Delta \text{ is odd,} \\ 4D & \text{otherwise.} \end{cases}$$

is a discriminant *with* conductor *f_Δ, and associated* radicand

$$D = \begin{cases} f_\Delta^2 D_0 & \text{if } D_0 \not\equiv 1 \pmod 4 \text{ or } f_\Delta \text{ is odd ,} \\ (f_\Delta/2)^2 D_0 & \text{otherwise ,} \end{cases}$$

having underlying fundamental discriminant Δ_0 with associated fundamental radicand D_0. (We use the letter f above for conductor since the German word for it is Führer*. The origins of the mathematical term* conductor *are rooted in the German language.)*

Definition A.30 **Quadratic Irrationals**

Suppose that Δ is a discriminant with underlying radicand $D > 1$. A quadratic irrational*, with underlying discriminant Δ, is a number of the form*

$$\alpha = \frac{P + \sqrt{D}}{Q}, \quad (P, Q \in \mathbb{Z})$$

where $Q \neq 0$ and $P^2 \equiv D \pmod Q$.

Theorem A.21 **Lagrange: Quadratic Irrationals Are Periodic**

Let $\alpha \in \mathbb{R}$. Then α has a periodic infinite simple continued fraction expansion if and only if α is a quadratic irrational.

Theorem A.22 **Pure Periodicity Equals Reduction**

Let $\alpha = \langle q_0; q_1, \ldots \rangle$ be an infinite simple continued fraction, with $\ell(\alpha) = \ell \in \mathbb{N}$. Then α is purely periodic if and only if $\alpha > 1$ and $-1 < \alpha' < 0$, where α' is the algebraic conjugate of α. Any quadratic irrational which satisfies these two conditions is called reduced*.*

Corollary A.3 *If $D > 1$ is not a perfect square, then*

$$\sqrt{D} = \langle q_0; \overline{q_1, \ldots, q_{\ell-1}, 2q_0} \rangle,$$

where $q_j = q_{\ell-j}$ for $j = 1, 2, \ldots, \ell - 1$ and $q_0 = \lfloor \sqrt{D} \rfloor$.

Of crucial importance in the text involving the continued factoring algorithm in Section 6.2 is the following material.

Theorem A.23 **Continued Fractions and Recursion**

Let D be a positive integer that is not a perfect square, and let

$$\alpha_0 = (P_0 + \sqrt{D})/Q_0$$

be a quadratic irrational. Recursively define the following for $k \geq 0$:

$$\alpha_k = (P_k + \sqrt{D})/Q_k, \tag{A.3}$$

$$q_k = \lfloor \alpha_k \rfloor, \tag{A.4}$$

$$P_{k+1} = q_k Q_k - P_k, \tag{A.5}$$

and

$$Q_{k+1} = (D - P_{k+1}^2)/Q_k. \tag{A.6}$$

Then $P_k, Q_k \in \mathbb{Z}$ and $Q_k \neq 0$ for $k \geq 0$, and $\alpha_k = \langle q_k; q_{k+1}, \ldots \rangle$.

Theorem A.24 **Continued Fractions and Quadratic Irrationals**

Let $\alpha = (P + \sqrt{D})/Q$ be a quadratic irrational and set

$$G_{k-1} = Q_0 A_{k-1} - P_0 B_{k-1} \quad (k \geq -1),$$

where A_{k-1}, B_{k-1} are given in Theorem A.18 on page 322. Then

$$G_{k-1}^2 - B_{k-1}^2 D = (-1)^k Q_k Q_0 \quad (k \geq 1). \tag{A.7}$$

Corollary A.4 *If $\alpha = \sqrt{D}$, then Equation* (A.7) *becomes*

$$A_{k-1}^2 - B_{k-1}^2 D = (-1)^k Q_k. \tag{A.8}$$

Theorem A.25 **Reduction and Periodicity**

Let α be a reduced quadratic irrational with $\ell(\alpha) = \ell$. Then both of the following hold.

(a) *$P_0 = P_{k\ell}$ and $Q_0 = Q_{k\ell}$ for all $k \geq 0$.*

(b) *If $\beta = \sqrt{D}$ and $\ell(\beta) = \ell$, then $P_1 = P_{k\ell}$ for all $k \geq 1$ and $Q_0 = Q_{k\ell} = 1$ for all $k \geq 0$.*

Appendix B: Computer Arithmetic

The advent of modern-day electronic computers has radically altered the nature of cryptography. Therefore, it is imperative that we learn to speak and manipulate the vernacular of computers. That is the purpose of this appendix.

We are familiar with the Hindu-Arabic numeral system, which is a base 10 (decimal) system. For example, $5146 = 5 \cdot 10^3 + 1 \cdot 10^2 + 4 \cdot 10^1 + 6 \cdot 10^0$. Other civilizations from antiquity used different bases. For instance, the ancient Babylonians used base 60 (sexagesimal) and the ancient Mayans used base 20. Modern-day computers use base 2. The reason for the latter has to do with how computers store data internally. In other words, computers, at their very essence, really only understand two possibilities such as "electrical charge or no electrical charge" or "magnetized clockwise or magnetized counterclockwise." Thus, the language that we need to learn begins with an understanding of how digits are represented in a computer. To begin to touch base with this understanding, we need the following elementary result.

Theorem B.1 **Base Representations of Integers**

If $b > 1$ is an integer, then every $n \in \mathbb{N}$ has a unique representation as

$$n = \sum_{j=0}^{t_n} a_j b^j,$$

where t_n is the smallest nonnegative integer such that $a_j = 0$ for all $j > n$.

Proof. See [61, Theorem 1.5.1, p. 52]. □

If $n \in \mathbb{N}$, then by Theorem B.1, there is a unique representation:

$$(n)_{10} = (a_{t_n} a_{t_n - 1} \dots a_1 a_0)_b,$$

for the *base* b (or radix b) representation of the base 10 integer n, where $j = t_n$ is the largest nonnegative integer such that $a_j \neq 0$. The value $t_n + 1$ is called the *base-b length* of n, and the a_j are called the *base* (radix) b digits of n. For instance, if $b = 2$, then the base 2 length is called the *bitlength* of n.

Important applications of Theorem B.1 are those representations for $n \in \mathbb{N}$ of the form

$$n = \sum_{j=0}^{t_n} a_j 2^j,$$

where $0 \leq a_j \leq 1 \quad (t_n \geq 0)$. The a_j are called *bits*, which is a contraction of *bi*nary digi*ts*, and $(a_{t_n} a_{t_n - 1} \dots a_0)$ is called a *bitstring of length* $t_n + 1$. In other words, n has bitlength $t_n + 1$. For instance, 1057, a base 10 integer, has a binary representation:

$$1 \cdot 2^{10} + 0 \cdot 2^9 + 0 \cdot 2^8 + 0 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 2^0$$

or simply, $(1057)_{10} = (10000100001)_2$.

A *byte*, for future reference, is an 8-bit binary integer. The first recorded (but unpublished) appearance of the binary notation occurred in 1605 in a manuscript by Thomas Harriot. The first *published* appearance of the binary system was in the 1670 manuscript, *Mathesis Biceps I*, by a Cistercian bishop, Juan de Caramuel Lobkowitz, but his publication contained no discussion or examples of binary arithmetic. The first to contain such a discussion was a paper by Leibniz (see Biography B.3 on page 328), *Mémoires de l'Académie Royal des Sciences*, which appeared in 1703. Leibniz attributed mystical import to the fact that all numbers could be expressed in terms of zeros and ones. The binary system remained somewhat of a curiosity thereafter until the advent, in the 1930's, of electromechanical and electromagnetic circuitry. By the mid 1940's, there was support by major figures in the scientific community, such as von Neumann (see Biography B.2 on page 327), and since then binary computers have proliferated.

Biography B.1 Thomas Harriot (1560–1621) *was an astronomer and mathematician. He gained fame as the leading scientist surrounding Sir Walter Raleigh, who sent him as a scientific advisor on an expedition in 1585 to Roanoke Island, off the coast of what is now called North Carolina. In Harriot's work* Artis Analyticae Praxis ad Aequationes Algebraicas Resolvendas, *or* The Analytical Arts Applied to Solving Algebraic Equations, *he advanced the theory of equations. He also introduced the greater than* ($>$) *and the less than* ($<$) *signs. Among his* (*unfortunately*) *unpublished discoveries were the following: sunspots, and the moons of Jupiter* (*independently of his contemporary Galileo*), *and the law of refraction* (*bending of light*) *before Willebrord van Roijen Snell* (1591–1626), *who published the discovery, which is basic to modern geometrical optics. He died on July 2, 1621.*

Computers have an upper bound on the size of integers that can be used for its arithmetic operations. This upper bound is called the *word size*, which can be measured in binary as 2^e on an e-bit binary computer, or as 10^e on an e-digit decimal computer, for instance. These considerations are important when talking about large-scale arithmetic in connection with implementation of cryptosystems (about which we will learn later), for example. *Word length* is defined as the logarithm (taken to the appropriate base) of the word size. When we need to do computer arithmetic with integers bigger than the word size, then we must devote more than one word to each integer. One of the primary functions of this section is to describe how the basic arithmetic operations are formally performed.

Base names other than binary include $b = 3$ or *ternary*, $b = 4$ or *quaternary*, $b = 5$ or *quinary*, and so on. In base $b = 8$ we have what we call *octal* representation. For instance, the decimal integer 100 has octal representation $100 = 1 \cdot 8^2 + 4 \cdot 8^1 + 4 \cdot 8^0$, so $(100)_{10} = (144)_8$. Base 16, or *hexadecimal* (a term that has both Greek and Latin roots) representation uses the numbers 0 through 9 as well as the letters A, B, C, D, E, and F to represent the numbers 10, 11, 12, 13, 14, and 15, respectively. For instance, the decimal integer 195951310 has the base 16 representation $11 \cdot 16^6 + 10 \cdot 16^5 + 13 \cdot 16^4 + 15 \cdot 16^3 + 10 \cdot 16^2 + 12 \cdot 16^1 + 14$,

which translates into $(195951310)_{10} = (BADFACE)_{16}$.

Biography B.2 John von Neumann (1903–1957) *was born on December 3, 1903, in Budapest, Hungary, as Margiattai Neumann János. After receiving his Ph.D. in mathematics from the University of Budapest, his academic positions included: lecturer at the University of Berlin 1926–1929; lecturer at the University of Hamburg 1929–1930; visiting lecturer at Princeton University 1930; and ultimately a permanent member, along with Einstein, at the Institute for Advanced Study at Princeton in 1933. By the late 1930's he had published ground-breaking work on rings of operators, now called Neumann Algebras. During World War II, he was a consultant for the armed forces, where his accomplishments included the drawing up of a report on computer capabilities. In 1946, he published a paper coauthored with A.W. Burks and H.H. Goldstine that detailed virtually the entire field of "automatic computation," including designs for a stored-program computer. This paper strongly influenced the later design of digital computers. In 1955, von Neumann had been appointed to the Atomic Energy Commission, and in 1956 he received the Enrico Fermi award. Among his varied interests and contributions were: computer design, game theory (in which he set the mathematical cornerstone with the minimax theorem published in 1928), group theory, logic and foundations, meteorology, probability theory, and quantum physics. He died on February 8, 1957, in Washington, D.C.*

We have not yet addressed the issue of *negative* number representation. There are numerous ways to do this, among which is the *ones' complement*, which is a term used to describe the following. If $n \in \mathbb{Z}$ and $(|n|)_{10} = (a_{t_n} a_{t_n-1} \dots a_0)_2$, then for any integer $m > t_n + 1$, the *ones' complement* m-bit representation of n is given by:

$$\begin{cases} (\overbrace{00\dots0}^{m-t_n-1\text{copies}}\, a_{t_n} a_{t_n-1} \dots a_0)_2 & \text{if } n > 0, \\ (\underbrace{11\dots1}_{m-t_n-1\text{copies}}\, 1 - a_{t_n} 1 - a_{t_n-1} \dots 1 - a_0)_2 & \text{if } n < 0. \end{cases}$$

The $n \in \mathbb{Z}$ which can be represented as m-bit ones' complement binary integers are those in the range $-2^{m-1} - 1 \le n \le 2^{m-1} - 1$. The basic idea is to convert n to a binary digit and pack $m - t_n - 1$ zeros to the left. Then if $n > 0$, this is the ones' complement, whereas if $n < 0$, then replace all zeros by ones and all ones by zeros. This is illustrated as follows.

Example B.1 *To represent* $(26)_{10}$ *and* $(-26)_{10}$ *as* $m = 7$*-bit ones' complement integers, we calculate that* $(26)_{10} = (11010)_2$*, so* $(0011010)_2$ *is the* 7*-bit ones' complement representation of the decimal digit* 26*, whereas for* -26 *it is* $(1100101)_2$.

Notice that in the ones' complement system, $+0$ is represented by $(00)_2$, as an $m = 2$-bit binary integer, and -0 is represented by $(11)_2$. Since -0 and $+0$ are the same, but represented differently, some care in practice is required.

A method of representing negative numbers that does not have this disadvantage is called the *ten's complement* notation, which is described as follows. Assuming that we are working with $n-$bit numbers, then we always work with 10^n. For instance, if $n = 10$, then a negative number such as -3495657980 is represented as 6504342020 $(= 10^{10} - 3495657980)$. Hence, any number with a leading digit bigger than 4 is assumed to be negative, so no explicit sign is attached. This system clearly has limitations on size, but avoids the problem of the representation of $+0$ and -0.

Now that we know how to represent numbers to various bases in numerous ways, it is time to look at the arithmetic involved in using these bases. We begin with the most basic of arithmetic operations — addition.

Biography B.3 Gottfried Wilhelm von Leibniz (1646–1716) *was born on July 1, 1646, in Leipzig, Saxony (now Germany). By the age of twelve, he had taught himself Latin and Greek in order to be able to read the books of his father, who was a philosophy professor at Leipzig. Leibniz studied law at Leipzig from 1661 to 1666 and ultimately received a doctorate in law from the University of Altdorf in 1667. He pursued a career in law at the courts of Mainz from 1667 to 1672. Then he went to Paris from 1672 to 1676, during which time he studied mathematics and physics under Christian Huygens (1629–1695). In 1676, he left for Hannover, where he remained for the balance of his life. Leibniz began looking for a uniform and useful notation for the calculus in 1673. By November 1676, he discovered the now famous* $dx^n = nx^{n-1}dx$ *for nonzero* $n \in \mathbb{Q}$. *In 1684, he published the details of the differential calculus, the year before Newton published his famed* Principia. *Leibniz's formal approach was to have a vital impact upon the development of the calculus. The bitter dispute between Newton and Leibniz concerning priority over the discovery of the calculus is detailed in* [61, pp. 234–235]. *In 1700, Leibniz founded the Berlin Academy and was its first president. Then he became increasingly reclusive until his death in Hannover on November 14, 1716.*

◆ Addition Using Arbitrary Bases

Let $b, m, n \in \mathbb{N}$ and $b > 1$. By Theorem B.1, we have the unique base b representations

$$m = \sum_{j=0}^{t} a_j b^j \text{ and } n = \sum_{j=0}^{t} c_j b^j, \tag{B.1}$$

where t is the largest integer such that $a_j + c_j \neq 0$. Therefore, by Theorem A.2 on page 305,

$$m + n = \sum_{j=0}^{t} (a_j + c_j) b^j.$$

Also, by the Division Algorithm, Theorem 1.1, $a_0 + c_0 = q_0 b + r_0$, where $q_0, r_0 \in \mathbb{Z}$ with $0 \leq r_0 < b$. Given that $0 \leq a_0, c_0 \leq b - 1$, then $0 \leq a_0 + c_0 \leq 2b - 2$, so

$q_0 \in \{0, 1\}$. In particular, if $q_0 = 1$, then we call q_0 a *carry* to the next position. Continuing in this fashion, we get,

$$a_1 + c_1 + q_0 = q_1 b + r_1 \text{ with } 0 \leq r_1 \leq b - 1$$

for some $q_1 \in \mathbb{Z}$. Since $0 \leq a_1 + b_1 + q_0 \leq 2b - 1$, then $q_1 \in \{0, 1\}$. Again if $q_1 = 1$, then it is called a carry to the next position. By induction, there exist $r_j, q_j \in \mathbb{Z}$ for each natural number j such that

$$a_j + c_j + q_{j-1} = q_j b + r_j \text{ with } 0 \leq r_j \leq b - 1,$$

where $q_j \in \{0, 1\}$. Since t is the largest integer such that $a_j + c_j \neq 0$, then if $q_t = 1$, we set $r_{t+1} = q_t$ and write the base b representation of $m + n$ as $(r_{t+1} r_t \dots r_0)_b$, whereas if $q_t = 0$, then we write it as $(r_t r_{t-1} \dots r_0)_b$.

Example B.2 Suppose that we wish to calculate the addition of the two binary numbers $(10000011)_2$ and $(11010101)_2$. The following table illustrates the above process. The left-pointing arrows over a given column indicate that there is a carry from that position to the next. The binary addition is on the left and the decimal addition is on the right for easy reference.

j	8	$\overleftarrow{7}$	6	5	4	3	$\overleftarrow{2}$	$\overleftarrow{1}$	$\overleftarrow{0}$	binary/decimal	2	1	0
a_j		1	0	0	0	0	0	1	1	$\longleftrightarrow$	1	3	1
c_j		1	1	0	1	0	1	0	1	$\longleftrightarrow$	2	1	3
r_j	1	0	1	0	1	1	0	0	0	$\longleftrightarrow$	3	4	4

Thus, the addition of the two binary numbers, each of bitlength 8, is a binary number $(101011000)_2$ of bitlength 9, whereas the corresponding decimal numbers of bitlength 3 each sum to a decimal number $(344)_{10}$ of bitlength 3. Notice that, in the binary addition, $q_0 = q_1 = q_2 = q_7 = 1$, which accounts for the carries from each of those positions. On the other hand $q_0 = q_1 = q_2 = 0$ in the decimal addition, which accounts for the *lack* of carries in that summation.

We now look at the complementary notion of subtraction of integers to various bases. Of course, properly viewed, subtraction is not new since it is merely the *addition* of a number to a negative number.

◆ Subtraction Using Arbitrary Bases

Consider the representations given in (B.1) under the assumption that $m > n$. Then

$$m - n = \sum_{j=0}^{t} (a_j - c_j) b^j,$$

where t is the largest integer such that $a_j - c_j \neq 0$. We use the Division Algorithm to get

$$a_0 - c_0 = q_0 b + r_0 \text{ where } q_0, r_0 \in \mathbb{Z} \text{ with } 0 \leq r_0 < b.$$

Since $0 \leq a_0, c_0 < b$, then $-b < a_0 - c_0 < b$. Hence, $-1 \leq q_0 \leq 0$. If $q_0 = -1$, then we must borrow from the next position, so q_0 is called a *borrow* in this case. Continuing in this fashion,

$$a_1 - c_1 + q_0 = q_1 b + r_1 \text{ where } q_0, r_0 \in \mathbb{Z} \text{ with } 0 \leq r_1 < b.$$

Since $-b \leq a_1 - c_1 + q_0 < b$, then $q_1 \in \{-1, 0\}$. Using induction, we see that for any nonnegative integer j, we get

$$a_j - c_j + q_{j-1} = q_j b + r_j \text{ where } q_j, r_j \in \mathbb{Z} \text{ with } 0 \leq r_j < b,$$

and $q_j \in \{-1, 0\}$, with $q_{-1} = 0$. Hence,

$$\sum_{j=0}^{t} r_j b^j = \sum_{j=0}^{t} (a_j - c_j + q_{j-1} - q_j b) b^j = \sum_{j=0}^{t} (a_j - c_j) b^j,$$

since

$$\sum_{j=0}^{t} (q_{j-1} - q_j b) b^j = 0.$$

Since $j = t$ is the largest integer such that $a_j - c_j \neq 0$, the base b representation of $m - n$ is

$$(r_t r_{t-1} \ldots r_0)_b.$$

Example B.3 Suppose that we wish to calculate the result of subtracting $(100131)_4$ from $(303020)_4$. Consider the following tabular illustration, where the left-pointing arrow designates a borrow from the next position.

j	5	4	3	$\overleftarrow{2}$	$\overleftarrow{1}$	$\overleftarrow{0}$
a_j	3	0	3	0	2	0
c_j	1	0	0	1	3	1
r_j	2	0	2	2	2	3

Hence,

$$(303020)_4 - (100131)_4 = (202223)_4.$$

The actual development in the table that corresponds to the preceding notation is given as follows.

$$a_0 - c_0 = 0 - 1 = -1 \cdot 4 + 3 = q_0 b + r_0,$$

$$a_1 - c_1 + q_0 = 2 - 3 - 1 = -1 \cdot 4 + 2 = q_1 \cdot b + r_1,$$

$$a_2 - c_2 + q_1 = 0 - 1 - 1 = -1 \cdot 4 + 2 = q_2 \cdot b + r_2,$$

$$a_3 - c_3 + q_2 = 3 - 0 - 1 = 0 \cdot 4 + 2 = q_3 \cdot b + r_3,$$

$$a_4 - c_4 + q_3 = 0 - 0 + 0 = 0 \cdot 4 + 0 = q_4 \cdot b + r_4,$$

and

$$a_5 - c_5 + q_4 = 3 - 1 + 0 = 0 \cdot 4 + 2 = q_5 b + r_5 = q_t b + r_t.$$

Example B.4 The following illustrates the subtraction of the octal integer $(67677)_8$ from $(77501)_8$, where the left-pointing arrow denotes a borrow from the next position.

j	4	$\overleftarrow{3}$	$\overleftarrow{2}$	$\overleftarrow{1}$	$\overleftarrow{0}$
a_j	7	7	5	0	1
c_j	6	7	6	7	7
r_j	0	7	6	0	2

Thus,

$$(77501)_8 - (67677)_8 = (7602)_8.$$

The next step in the learning of computer arithmetic is multiplication, which is, properly viewed, a sequence of additions.

◆ Multiplication Using Arbitrary Bases

Consider the representations given in (B.1). We now determine the value of

$$mn = \left(\sum_{j=0}^{t} a_j b^j\right)\left(\sum_{j=0}^{t} c_j b^j\right),$$

where t is the largest integer such that $a_j c_j \neq 0$. We may simplify our task by observing that

$$mn = \sum_{j=0}^{t} (mc_j) b^j.$$

In other words, our task is simplified to the task of understanding how to find mc_j in base b for each j, then determining how to find $(mc_j)b^j$, and finally adding up the terms. For simplicity of explanation, we let $c_j = c$ for now.

By the Division Algorithm, $ca_0 = q_0 b + s_0$ for some integers s_0, q_0 with $0 \leq s_0 \leq b-1$. Also, since $0 \leq ca_0 \leq (b-1)^2$, then $0 \leq q_0 \leq b-2$. Continuing in this fashion, we get

$$a_1 c + q_0 = q_1 b + s_1 \text{ where } s_1, q_1 \in \mathbb{Z} \text{ with } 0 \leq s_1, q_1 \leq b-1.$$

By induction we get

$$a_j c + q_{j-1} = q_j b + s_j \text{ where } s_j, q_j \in \mathbb{Z} \text{ with } 0 \leq s_j, q_j \leq b-1$$

for $j = 0, 1, \ldots, t-1$ with $q_{-1} = 0$, and $s_t = q_{t-1}$, where $j = t$ is the largest integer such that $mc \neq 0$. Thus, if $q_t = 0$, then mc is

$$(a_t \ldots a_0)_b (c)_b = (s_t \ldots s_0)_b,$$

and if $q_t = 1$, then mc is

$$(a_{t+1} a_t \ldots a_0)_b (c)_b = (s_{t+1} s_t \ldots s_0)_b.$$

Now we can achieve $(mc_i)b^i$ by what is a *shift* (to the left). For instance, $(1301)_4$ multiplied by 4^3 is just a shift three places to the left of the digits of $(1301)_4$ and a fill-up of the original three places with zeros. In other words, 4^3 times $(1301)_4$ is $(1301000)_4$. Now we just add up the results of our efforts, and we have the base b representation of mn. (Note that for $s, t \in \mathbb{N}$ an s-bit integer multiplied by a t-bit integer yields an $(s+t)$-bit integer.)

Example B.5 We wish to multiply the two ternary digits $(222)_3$ and $(121)_3$. The following diagram illustrates the process.

			2	2	2
			1	2	1
			2	2	2
	1	2	2	1	
	2	2	2		
1	2	0	1	0	2

In terms of the notation preceding this example, we have the following. Let

$$m = (222)_3 = (a_2a_1a_0)_3$$

and

$$n = (121)_3 = (c_2c_1c_0)_3.$$

For $c = c_0 = 1$, we clearly have

$$(a_2a_1a_0)_3c_0 = (222)_3 = (s_2s_1s_0)_3.$$

For $c = c_1 = 2$,

$$ca_0 = 4 = 1 \cdot 3 + 1 = q_0b + s_0,$$
$$ca_1 + q_0 = 5 = 1 \cdot 3 + 2 = q_1b + s_1,$$
$$ca_2 + q_1 = 5 = 1 \cdot 3 + 2 = q_2b + s_2,$$

and $s_3 = 1$. Therefore,

$$(a_2a_1a_0)_3c_1 = (1221)_3 = (s_3s_2s_1s_0)_3.$$

For $c = c_2 = 1$, we have

$$(a_2a_1a_0)_3c_2 = (222)_3 = (s_2s_1s_0)_3.$$

Now we perform shifting on each mc_j. We have

$$(222)_3 \cdot 3^0 = (222)_3,$$
$$(1221)_3 \cdot 3^1 = (12210)_3,$$

and

$$(222)_3 \cdot 3^2 = (22200)_3.$$

When we add up these amounts, we get

$$(222)_3 + (12210)_3 + (22200)_3 = (120102)_3,$$

which is what the above diagram told us.

Now we turn our attention to division in various bases. As we will see, the division process amounts to a sequence of subtractions.

◆ Division Using Arbitrary Bases

Suppose that $m, n \in \mathbb{N}$, with $m \leq n$, then by the Division Algorithm,

$$n = mq + r \text{ for some } q \in \mathbb{N}, r \in \mathbb{Z} \text{ with } 0 \leq r \leq m - 1.$$

First we determine the base b representation of q as follows. Suppose that

$$q = (d_t d_{t-1} \dots d_0)_b, \quad t \geq 0.$$

Set

$$s_i = m \sum_{j=0}^{t-i} d_j b^j + r \geq 0$$

for any nonnegative integer $i \leq t$. Since $d_j \leq b - 1$ for all nonnegative integers $j \leq t$, we have

$$s_i \leq m \sum_{j=0}^{t-i} (b-1) b^j + r = m \left(\sum_{j=0}^{t-i} b^{j+1} - \sum_{j=0}^{t-i} b^j \right) + r = m(b^{t-i+1} - 1) + r.$$

Since $r < m$, it follows that $s_i < mb^{t-i+1}$. Also, since

$$s_i = s_{i-1} - m d_{t-i+1} b^{t-i+1},$$

for $1 \leq i \leq t+1$ where $s_{t+1} = r$, we have

$$\frac{s_{i-1}}{mb^{t-i+1}} = d_{t-i+1} + \frac{s_i}{mb^{t-i+1}} \geq d_{t-i+1} = \frac{s_{i-1} - s_i}{mb^{t-i+1}} > \frac{s_{i-1}}{mb^{t-i+1}} - 1.$$

In other words,

$$d_{t-i+1} = \left\lfloor \frac{s_{i-1}}{mb^{t-i+1}} \right\rfloor,$$

for $1 \leq i \leq t+1$.

The above is essentially an algorithm for finding each digit in the base b representation of q; namely, we subtract mb^{t-i+1} from s_{i-1} enough times until the result is negative. Then d_{t-i+1} is one less than the number of subtractions. We also observe that, since

$$d_t = \lfloor s_0/(mb^t) \rfloor = \lfloor n/(mb^t) \rfloor,$$

then this is our starting point.

Example B.6 Suppose we want to divide $n = (101)_2$ by $m = (11)_2$. Set

$$(101)_2 = (11)_2 (d_2 d_1 d_0)_2 + r = mq + r.$$

Then $t = 2$, $b = 2$ and $q = (d_2 d_1 d_0)$. Since

$$mb^t = (11)_2 \cdot 2^2 = (1100)_2,$$

then subtracting this from $s_0 = (101)_2 = n$ yields a negative number, so $d_2 = d_t = 0$. Since

$$mb^{t-1} = (11)_2 \cdot 2 = (110)_2,$$

then subtracting this from

$$s_1 = s_0 - md_t b^t = (101)_2 - (11)_2 \cdot 0 \cdot 2^2 = (101)_2$$

yields a negative number, so $d_1 = d_{t-1} = 0$. Since $mb^{t-2} = (11)_2$, and the subtraction of this from $s_2 = (101)_2$ yields $(10)_2$, whereas subtracting $(11)_2$ from $(10)_2$ yields a negative number, then $d_0 = d_{t-2} = 1$. We have shown that $q = (1)_2$. To get r, we look at

$$r = s_{t+1} = s_3 = s_2 - md_{t-2}b^{t-2} = (101)_2 - (11)_2 \cdot 1 \cdot 1 = (10)_2.$$

Hence,

$$n = (101)_2 = (11)_2(1)_2 + (10)_2 = mq + r.$$

Appendix C: The Rijndael S-Box

The means by which Rijndael's invertible S-Box, explicitly given below, was constructed consists of composing two functions. For each i, j with $1 \leq i \leq 32$, $1 \leq j \leq 8$, the following is executed. The first is to take the multiplicative inverse $\sum_{k=0}^{7} b_k 2^j$ of each nonzero $8i + j - 9$ in $\mathbb{F}_{2^8}$, with 0 getting mapped to 0 (see pages 311–314 in Appendix A). Thus, $a_{i,j} = 8i + j - 9$ gets mapped to $\sum_{k=0}^{7} b_k 2^j$ (where we have suppressed any reference to the i, j in the coefficients b_k for convenience of presentation in what follows). Then the following Affine function is applied:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{pmatrix}.$$

Thus, $s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0$ is the binary equivalent of the decimal digit appearing the the S-Box at position (i, j).

To illustrate how this S-Box was constructed, we first observe that the column matrix, added on the left of the equality, is binary for the decimal digit 99 (or equivalently, the hexadecimal digit 63). So, for instance, $a_{0,0} = 0$ gets mapped to 0, so each $b_k = 0$ for $1 \leq k \leq 7$. Adding the zero transformation to the column matrix yields 99, which is the first entry in the S-Box. A less trivial illustration is the entry $a_{11,3} = 82$ in the position matrix. It has representation as the binary polynomial $x^6 + x^4 + x$ in $\mathbb{F}_{2^8} \cong \mathbb{F}_2[x]/(m(x))$, where $m(x) = x^8 + x^4 + x^3 + x + 1$ is the irreducible Rijndael polynomial (see Example A.7 on page 318 in Appendix A). The multiplicative inverse of 82 in $\mathbb{F}_{2^8}$ is given by $x^2 + 1$, so $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0) = (0, 0, 0, 0, 0, 1, 0, 1)$. Thus:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

and 0 is the decimal entry in position $(11, 3)$ of the S-Box:

99	124	119	123	242	107	111	197
48	1	103	43	254	215	171	118
202	130	201	125	250	89	71	240
173	212	162	175	156	164	114	192
183	253	147	38	54	63	247	204
52	165	229	241	113	216	49	21
4	199	35	195	24	150	5	154
7	18	128	226	235	39	178	117
9	131	44	26	27	110	90	160
82	59	214	179	41	227	47	132
83	209	0	237	32	252	177	91
106	203	190	57	74	76	88	207
208	239	170	251	67	77	51	133
69	249	2	127	80	60	59	168
81	163	64	143	146	157	56	245
188	182	218	33	16	255	243	210
205	12	19	236	95	151	68	23
196	167	126	61	100	93	25	115
96	129	79	220	34	42	144	136
70	238	184	20	222	94	11	219
224	50	58	10	73	6	36	92
194	211	172	98	145	149	228	121
231	200	55	109	141	213	78	169
108	86	244	234	101	122	174	8
186	120	37	46	28	166	180	198
232	221	116	31	75	189	139	138
112	62	181	102	72	3	246	14
97	53	87	185	134	193	29	158
225	248	152	17	105	217	142	148
155	30	135	233	206	85	40	223
140	161	137	13	191	230	66	104
65	153	45	15	176	84	187	22

Appendix D: Knapsack Ciphers

In this appendix, we study the class of problems named in the appendix header and some public-key cryptosystems based upon them. This class of problems is a generalization of the following notion.

Definition D.1 **The Subset Sum Problem**

Given $m, n \in \mathbb{N}$ and a set $\mathcal{S} = \{b_j : b_j \in \mathbb{N}, \text{ for } j = 1, 2, \ldots, n\}$, called a knapsack set*, determine whether or not there exists a subset $\mathcal{S}_0$ of $\mathcal{S}$ such that the sum of the elements in $\mathcal{S}_0$ equals m. In other words, given $\mathcal{S}$ and $m \in \mathbb{N}$, determine whether or not there exist $a_j \in \{0, 1\}$, for $j = 1, 2, \ldots, n$, such that*

$$\sum_{j=1}^{n} a_j b_j = m. \tag{D.1}$$

It is known that the Subset Sum Problem is **NP**-complete (see page 73). The Subset Sum Problem is often mistakenly identified as the following more general problem.

Definition D.2 **The Knapsack Problem**

Given natural numbers m_1, m_2 and sets $\{b_j : b_j \in \mathbb{N}, \text{ for } j = 1, 2, \ldots, n\}$ and $\{c_j : c_j \in \mathbb{N}, \text{ for } j = 1, 2, \ldots, n\}$, determine whether or not there exists a set $\mathcal{S}_0 \subseteq \{1, 2, \ldots, n\}$ such that

$$\sum_{j \in \mathcal{S}_0} b_j \leq m_1 \text{ and } \sum_{j \in \mathcal{S}_0} c_j \geq m_2.$$

(*The Subset Sum Problem is the special case where $b_j = c_j$ for $j = 1, 2, \ldots, n$ and $m_1 = m_2 = m$.*)

The knapsack problem derives its name from the special case of the Subset Sum Problem, which may be restated in nonmathematical terms as follows. Given a collection of items, each of different sizes, is it possible to put some of the items into a knapsack (of a given size) so that the knapsack is full? Computationally, this is equivalent to actually determining the a_js in Equation (D.1), given that such a_js in fact exist. This computational version of the Subset Sum Problem is **NP**-hard (which means that the existence of a polynomial time algorithm for its solution would imply that $\mathbf{P} = \mathbf{NP}$).

Knapsack public-key cryptography is based on the Subset Sum Problem. The basic idea is to choose a case of the Subset Sum Problem that can be easily solved, then cryptographically disguise it to be an instance of the Subset Sum Problem that is hard to solve. In 1978, the first practical incarnation of such a cryptosystem was introduced by Merkle and Hellman in [59]. We present it for

its historical implications in that we may see how the knapsack cryptosystems evolved.

The Merkle-Hellman Cipher is based upon the following easily solved instance of the Subset Sum Problem.

Definition D.3 **Superincreasing Sequences**

A superincreasing sequence *is a sequence* $(b_1, b_2, \ldots, b_n)$ *with* $b_j \in \mathbb{N}$ *for* $j = 1, 2, \ldots, n$ *satisfying the property that*

$$b_i > \sum_{j=1}^{i-1} b_j,$$

for each $i \in \{2, 3, \ldots, n\}$.

Solving the Subset Sum Problem for superincreasing sequences is shown to be easy as follows. To find a subset $\mathcal{S}_0$ of

$$\mathcal{S} = \{b_1, b_2, \ldots, b_n\}$$

which sums to a given $d \in \mathbb{N}$ (assuming that such an $\mathcal{S}_0$ exists) one first sets $x_n = 1$ if $b_n \leq d$, and $x_n = 0$ otherwise. Then one merely looks at each successive b_{n-i} for $i = 1, 2, \ldots, n-1$ and one puts $x_{n-i} = 1$ if

$$b_{n-i} \leq d - \sum_{j=n-i+1}^{n} b_j$$

and put $x_{n-i} = 0$ otherwise. Then

$$d = \sum_{j=1}^{n} x_j b_j. \tag{D.2}$$

◆ The Merkle-Hellman Knapsack Cryptosystem

(I) Merkle-Hellman Key Generation

Given fixed $n \in \mathbb{N}$, entity Bob performs the following steps.

(1) Let $(b_1, b_2, \ldots, b_n)$ be a superincreasing sequence and $s \in \mathbb{N}$ such that $s > \sum_{j=1}^{n} b_j$.

(2) Randomly choose $r \in \mathbb{N}$ such that $r \leq s - 1$ and $\gcd(r, s) = 1$.

(3) Let $\sigma \in S_n$, the symmetric group on n letters.

(4) Compute $k_j \equiv r b_{\sigma(j)} \pmod{s}$ for $j = 1, 2, \ldots, n$.

(5) Bob's public key is $(k_1, \ldots, k_n)$ and the private key is $(\sigma, r, s, (b_1, \ldots, b_n))$.

(II) Merkle-Hellman Knapsack Public-Key Cipher

Alice performs the following steps.

enciphering stage:

(1) Obtain Bob's public key $(k_1, k_2, \ldots, k_n)$.

(2) Let $m = m_1 m_2 \ldots m_n$ be the representation of the message m as a bitstring of length n.

(3) Compute $c = \sum_{j=1}^{n} k_j m_j$ and send c to Bob.

Once Bob receives c, then the following is performed.

deciphering stage:

(1) Compute $d \equiv r^{-1} c \pmod{s}$.

(2) By solving the superincreasing Subset Sum Problem as described prior to Equation (D.2), obtain $x_j \in \{0, 1\}$ such that

$$d = \sum_{j=1}^{n} x_j b_j.$$

(3) Recover $m = x_{\sigma(1)} x_{\sigma(2)} \ldots x_{\sigma(n)}$.

To see why step (3) of the deciphering stage works, we observe that

$$d \equiv r^{-1} c = r^{-1} \sum_{j=1}^{n} k_j m_j = \sum_{j=1}^{n} (r^{-1} k_j) m_j \equiv \sum_{j=1}^{n} b_{\sigma(j)} m_j \pmod{s}.$$

Since $s > d \geq 0$, then

$$d = \sum_{j=1}^{n} b_{\sigma(j)} m_j = \sum_{j=1}^{n} b_j x_j = \sum_{j=1}^{n} b_{\sigma(j)} x_{\sigma(j)},$$

so $x_{\sigma(j)} = m_j$ for each $j = 1, 2, \ldots, n$.

Example D.1 Let $n = 7$, and assume that entity Bob selects superincreasing sequence $(b_1, b_2, b_3, b_4, b_5, b_6, b_7) = (1, 2, 4, 8, 16, 32, 70)$, $s = 200$, $r = 27$, and

$$\sigma : (1, 2, 3, 4, 5, 6, 7) \mapsto (7, 4, 5, 6, 3, 2, 1).$$

Then Bob computes $k_j \equiv 27 b_{\sigma(j)} \pmod{200}$ for $1 \leq j \leq 7$ to get: $(k_1, k_2, k_3, k_4, k_5, k_6, k_7) = (90, 16, 32, 64, 108, 54, 27)$, which is Bob's public key, and the private key is $(\sigma, 27, 200, (1, 2, 4, 8, 16, 32, 70))$. Suppose that the message to be sent is $m = 1010011$. Using Bob's public key, Alice computes

$$c = \sum_{j=1}^{n} k_j m_j = 1 \cdot 90 + 0 \cdot 16 + 1 \cdot 32 + 0 \cdot 64 + 0 \cdot 108 + 1 \cdot 54 + 1 \cdot 27 = 203,$$

which gets sent to Bob, who then computes

$$d \equiv r^{-1}c = 27^{-1} \cdot 203 \equiv 163 \cdot 203 \equiv 89 \,(\mathrm{mod}\ 200).$$

Then Bob solves the superincreasing Subset Sum Problem as follows. Since $b_n = b_7 = 70 < d = 89$, then $x_7 = 1$. Since $b_{n-1} = b_6 = 32 > 89 - 70 = d - b_n$, then $x_6 = 0$; $b_{n-2} = b_5 = 16 < 89 - 70 = 19$ implies $x_5 = 1$; $b_{n-3} = b_4 = 8 > 89 - 70 - 16$ implies $x_4 = 0$; $b_{n-4} = b_3 = 4 > 89 - 70 - 16$ implies $x_3 = 0$; $b_{n-5} = b_2 = 2 < 89 - 70 - 16$ implies $x_2 = 1$; and $b_{n-6} = b_1 = 1$ implies $x_1 = 1$. Thus,

$$d = 89 = 70 + 16 + 2 + 1 = \sum_{j=1}^{n} x_j b_j.$$

Therefore, since $m_j = x_{\sigma(j)}$, then $m_1 = x_{\sigma(1)} = x_7 = 1$; $m_2 = x_{\sigma(2)} = x_4 = 0$; $m_3 = x_{\sigma(3)} = x_5 = 1$; $m_4 = x_{\sigma(4)} = x_6 = 0$; $m_5 = x_{\sigma(5)} = x_3 = 0$; $m_6 = x_{\sigma(6)} = x_2 = 1$; $m_7 = x_{\sigma(7)} = x_1 = 1$, and m is recovered.

In 1982, unfortunately for this historically important and elegant cipher, a polynomial-time algorithm for breaking it was produced by Shamir (see [84]–[85]). Also, in 1982 (see [16]–[17]) a new knapsack cryptosystem was proposed. Until recently, the following was the only known secure knapsack cipher. However, it was broken by S. Vaudenay in 2001 (see [90]).

☞ The Chor-Rivest Knapsack Cryptosystem

(I) Chor-Rivest Key Generation

Entity Bob performs the following steps.

(1) Choose a finite field $\mathbb{F}_q$ of characteristic p, where $q = p^n$, $p > n$, and $p^n - 1$ has only small prime factors.

(2) Choose a random monic irreducible polynomial $r(x)$ with $\deg(r) = n$ over $\mathbb{F}_p$. (A method for generating irreducible polynomials is given, for instance, in Corollary A.2 on page 314.) We may view the elements of $\mathbb{F}_q$ as those from $(\mathbb{Z}/p\mathbb{Z})[x]/(r(x))$ (see Example A.7 on page 318 and the material preceding it).

(3) Let $\ell(x) \in (\mathbb{Z}/p\mathbb{Z})[x]/(r(x)) \cong \mathbb{F}_q$ be randomly chosen as a generator of $\mathbb{F}_q^*$ (see page 313).

(4) For each $\alpha \in \mathbb{Z}/p\mathbb{Z}$, find the discrete logarithm, $a_\alpha = \log_{\ell(x)}(x + \alpha)$ using the Pohlig-Hellman Algorithm in Appendix E.

(5) Choose a random permutation σ on $\{0, 1, \ldots, p-1\}$.

(6) Randomly choose $d \in \mathbb{Z}$ such that $0 \le d \le p^n - 2$.

(7) For any nonnegative integer $j \le p-1$ compute $c_j \equiv (a_{\sigma(j)} + d) \,(\mathrm{mod}\ p^n - 1)$.

(8) Bob has public key $((c_0, c_1, \ldots, c_{p-1}), p, n)$ and private key $(r(x), \ell(x), \sigma, d)$.

(II) Chor-Rivest Knapsack Public-Key Cipher

Alice performs the following steps.

enciphering stage:

(1) Obtain Bob's public key as given above.

(2) Let $\binom{p}{n}$ be the binomial coefficient (see Definition A.12 on page 306) and represent the message m as a bitstring of length $\lfloor \log_2 \binom{p}{n} \rfloor$.

(3) Replace m by the binary p-tuple $V = (V_0, V_1, \ldots, V_{p-1})$ having exactly n values of i such that $V_i = 1$ for $0 \leq i \leq p-1$ determined as follows. For $i = 1, \ldots, p$, execute:

Step i. If $m \geq \binom{p-i}{n}$, then execute the following steps.

(a) Set $V_{i-1} = 1$.

(b) Reset $m = m - \binom{p-i}{n}$.

(c) Reset $n = n - 1$.

(d) If $i < p$, then go to step $i + 1$. If $i = p$, terminate with output V.

If $m < \binom{p-i}{n}$, then execute the following steps.

(a) Set $V_{i-1} = 0$.

(b) If $i < p$, then go to step $i + 1$. If $i = p$, terminate with output V.

(Note that if any value $\binom{m}{n}$ with $0 \leq m < n$ is encountered, then this value is considered to be 0.)

(4) Compute $c \equiv \sum_{j=0}^{p-1} V_j c_j \pmod{p^n - 1}$, and send c to Bob.

Once Bob receives c, then the following is performed.

deciphering stage:

Bob performs the following steps.

(1) Compute $z \equiv (c - nd) \pmod{p^n - 1}$.

(2) Compute $h(z) \equiv \ell(x)^z \pmod{r(x)}$.

(3) Compute $k(x) = h(x) + r(x)$, where $deg_{\mathbb{Z}/p\mathbb{Z}}(k) = n$ (see Definition A.15 on page 312).

(4) Factor $k(x)$ into linear factors over $\mathbb{Z}/p\mathbb{Z}$:

$$k(x) = \prod_{j=1}^{n} (x + r_j) \qquad (r_j \in \mathbb{Z}/p\mathbb{Z}),$$

which may be accomplished by computing $k(x)$ for all $x \in \mathbb{F}_p$.

(5) Since $V_{\sigma^{-1}(r_j)} = 1$ for $j = 1, 2, \ldots, n$, we recover m as:

$$m = \sum_{j=1}^{n} V_{\sigma^{-1}(r_j)} \binom{p-1-\sigma^{-1}(r_j)}{n-\sum_{i=0}^{\sigma^{-1}(r_j)-1} V_i},$$

where we set $\sum_{i=0}^{\sigma^{-1}(r_j)-1} V_i = 0$ if $\sigma^{-1}(r_j) = 0$.

An easy exercise shows that the deciphering stage must indeed recover m. In the following illustration, we keep the parameters small for simplicity.

Example D.2 Suppose that entity Bob chooses $p = 5$, $n = 3$, and randomly selects $r(x) = x^3 + x + 1$, which is irreducible over $\mathbb{F}_5$, and randomly chooses $\ell(x) = 2x^2 + 2$ which generates $\mathbb{F}_{5^3}$ (see Theorem A.7 on page 313). To see the latter, note that

$$\ell(x)^{(5^3-1)/2} = \ell(x)^{124/2} = \ell(x)^{62} \equiv -1 \pmod 5$$

in

$$\mathbb{F}_{5^3} \cong (\mathbb{Z}/5\mathbb{Z})[x]/(r(x)).$$

Then Bob computes discrete logs as follows using the Pohlig-Hellman Algorithm. $a_0 = \log_{\ell(x)}(x) = 30$, $a_1 = \log_{\ell(x)}(x+1) = 28$, $a_2 = \log_{\ell(x)}(x+2) = 106$, $a_3 = \log_{\ell(x)}(x+3) = 50$, and $a_4 = \log_{\ell(x)}(x+4) = 23$. Then Bob randomly selects $\sigma : \{0,1,2,3,4\} \mapsto \{4,0,2,1,3\}$, and $d = 29$, after which Bob computes

$$c_0 \equiv a_{\sigma(0)} + d \equiv a_4 + 29 \equiv 23 + 29 \equiv 52 \pmod{124},$$

$$c_1 \equiv a_{\sigma(1)} + d \equiv a_0 + 29 \equiv 30 + 29 \equiv 59 \pmod{124},$$

$$c_2 \equiv a_{\sigma(2)} + d \equiv a_2 + 29 \equiv 106 + 29 \equiv 11 \pmod{124},$$

$$c_3 \equiv a_{\sigma(3)} + d \equiv a_1 + 29 \equiv 28 + 29 \equiv 57 \pmod{124},$$

and

$$c_4 \equiv a_{\sigma(4)} + d \equiv a_3 + 29 \equiv 50 + 29 \equiv 79 \pmod{124},$$

so Bob's public key is $((c_0, c_1, c_2, c_3, c_4), p, n) = ((52, 59, 11, 57, 79), 5, 3)$ and Bob's private key is $(r(x), \ell(x), \sigma, d) = (x^3 + x + 1, 2x^2 + 2, \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 4 & 0 & 2 & 1 & 3 \end{pmatrix}, 29)$. Now Alice gets Bob's public key and represents the message $m = 5$ as a bitstring $m = 101$ of length $\lfloor \log_2 \binom{5}{3} \rfloor = 3$. Alice then replaces bitstring m with 5-tuple V determined as follows. For $i = 1$, $V_0 = 1$ since $m = 5 > \binom{p-i}{n} = \binom{4}{3} = 4$. For $i = 2$, $V_1 = 0$ since $m = 5 - \binom{4}{3} = 1 < \binom{p-i}{n} = \binom{3}{2} = 3$. For $i = 3$, $V_2 = 1$ since $m = 1 = \binom{p-i}{n} = \binom{2}{2} = 1$. For $i = 4$, $V_3 = 0$ since $m = 0 < \binom{p-i}{n} = \binom{1}{1} = 1$. For $i = 5$, $V_4 = 1$ since $m = 0 = \binom{p-i}{n} = \binom{0}{1}$. Thus, $V = (1, 0, 1, 0, 1)$ and Alice computes

$$c \equiv \sum_{j=0}^{4} V_j c_j = c_0 + c_2 + c_4 = 52 + 11 + 79 \equiv 18 \pmod{124},$$

which Alice sends to Bob, who computes $z \equiv (c - nd) = 18 - 3 \cdot 29 \equiv 55 \pmod{124}$, $h(z) = \ell(x)^z = (2x^2 + 2)^{55} \equiv 4x^2 + 3 \pmod{x^3 + x + 1}$, and $k(x) = h(x) + r(x) = x^3 + 4x^2 + x + 4$, which factors over $\mathbb{F}_5$ as $k(x) = (x+2)(x+3)(x+4)$, so $r_1 = 2$, $r_2 = 3$, and $r_3 = 4$. Since $\sigma^{-1}(r_1) = \sigma^{-1}(2) = 2$, $\sigma^{-1}(r_2) = \sigma^{-1}(3) = 4$, and $\sigma^{-1}(r_3) = \sigma^{-1}(4) = 0$, Bob can recover m as follows.

$$m = \sum_{j=1}^{n} V_{\sigma^{-1}(r_j)} \binom{p - 1 - \sigma^{-1}(r_j)}{n - \sum_{i=0}^{\sigma^{-1}(r_j)-1} V_i} =$$

$$V_2 \binom{2}{2} + V_4 \binom{0}{1} + V_0 \binom{4}{3} = 1 + 0 + 4 = 5 = 101.$$

When the Chor-Rivest Algorithm is properly set up, it was thought to be secure against known attacks. However, as we noted earlier, it has been cryptanalyzed. The major problem with the Chor-Rivest cryptosystem is the huge size of the public key. For instance, in practice the recommended values for p and n are $p = 197$ and $n = 24$, and in this case, the public key has approximately 36000 bits. In 1991, Hendrik Lenstra introduced a modified version of the algorithm in [54], which does not require step (4) in the key generation. In other words, computation of discrete logs is avoided. However, Lenstra's Algorithm, called the *powerline system*, is not a knapsack cryptosystem.

Appendix E: Silver-Pohlig-Hellman Algorithm

◆ Silver-Pohlig-Hellman Algorithm for Computing Discrete Logs

Let α be a generator of $\mathbb{F}_p^*$ and let $\beta \in \mathbb{F}_p^*$, and assume that we have a factorization

$$p - 1 = \prod_{j=1}^{r} p_j^{a_j} \qquad a_j \in \mathbb{N},$$

where the p_j are distinct primes. The technique for computing $e = \log_\alpha \beta$ is to compute e modulo $p_j^{a_j}$ for $j = 1, 2, \ldots, r$, then apply the Chinese Remainder Theorem (see Theorem 1.12 on page 26). Since we operate on each prime power $p_j^{a_j}$, we replace p_j with q for simplicity in what follows, and simply refer to q^a with the understanding that we are operating on each of the r prime powers in this fashion. To compute e modulo q^a we need to determine e in its base q representation:

$$e = \sum_{i=0}^{a-1} b_i q^i \qquad \text{where } 0 \le b_i \le q - 1 \text{ for } 0 \le i \le a - 1.$$

To find these b_i, we proceed as follows. First, set $\beta_0 = \beta = \alpha^e$, and observe that

$$(p-1) \sum_{k=i}^{a-1} b_k q^{k-i-1} \equiv (p-1) b_i / q \,(\text{mod } p-1). \tag{E.1}$$

1. Calculate b_0. By (E.1),

$$\beta_0^{(p-1)/q} \equiv \alpha^{(p-1)b_0/q} \,(\text{mod } p), \tag{E.2}$$

using Fermat's Little Theorem (see Theorem 1.16 on page 36). Thus, we compute $\alpha^{(p-1)k/q} \,(\text{mod } p)$ until (E.2) occurs, in which case k is b_0.

2. Calculate b_i for $i = 1, 2, \ldots, a - 1$. First, recursively define

$$\beta_i = \beta \alpha^{-\sum_{k=0}^{i-1} b_k q^k}.$$

By (E.1),

$$\beta_i^{(p-1)/q^{i+1}} \equiv \alpha^{(p-1)\sum_{k=i}^{a-1} b_k q^{k-i-1}} \equiv \alpha^{(p-1)b_i/q} \,(\text{mod } p), \tag{E.3}$$

so we compute $\alpha^{(p-1)k/q}$ modulo p for nonzero $k \le a-1$ until the left and right sides of (E.3) are congruent modulo p, in which case k is b_i.

A small example is in order. This is, of course, not realistic in terms of the degree of difficulty, but for pedagogical purposes it will suffice, and we will do this often for the same reasons throughout.

Example E.1 Let $p = 37$. Then $\alpha = 2$ generates $\mathbb{F}_{37}^*$. Given $\beta_0 = \beta = 19$, we want to compute $e = \log_2(19)$ in $\mathbb{F}_{37}^*$. We have $p - 1 = 36 = 2^2 \cdot 3^2 = p_1^{a_1} p_2^{a_2}$. All congruences in the balance of this example are assumed to be modulo 37.

For $p_1 = 2$:

k	0	1
$\alpha^{(p-1)k/p_1}$	1	$2^{18} \equiv 36$

i	0	1
β_i	19	$19 \cdot 2^{-1} \equiv 28$
$\beta_i^{(p-1)/p_1^{i+1}}$	$19^{18} \equiv 36$	$28^9 \equiv 36$
b_i	1	1

Thus, the base 2 representation of $\log_2(19)$ modulo 4 is

$$\sum_{i=0}^{a-1} b_i p_1^i = 1 \cdot 2^0 + 1 \cdot 2^1 \equiv 3 \,(\text{mod } 4). \tag{E.4}$$

For $p_2 = 3$:

k	0	1	2
$\alpha^{(p-1)k/p_2}$	1	$2^{12} \equiv 26$	$2^{24} \equiv 10$

i	0	1
β_i	19	$19 \cdot 2^{-2} \equiv 14$
$\beta_i^{(p-1)/p_2^{i+1}}$	$19^{12} \equiv 10$	$14^9 \equiv 10$
b_i	2	2

Thus, the base 3 representation of $\log_2(19)$ modulo 9 is

$$\sum_{i=0}^{a_2-1} b_i p_2^i = 2 \cdot 3^0 + 2 \cdot 3^1 \equiv 8 \,(\text{mod } 9). \tag{E.5}$$

Solving (E.4)–(E.5) by the Chinese Remainder Theorem, we get that $e = \log_2(19) = 35$ in $\mathbb{F}_{37}^*$.

If $n = p - 1$, then given a factorization of n, the running time of the Silver-Pohlig-Hellman discrete log algorithm is

$$O\left(\sum_{j=1}^{r} a_j \left(\ln n + \sqrt{p_j}\right)\right)$$

group multiplications. This implies that the Pohlig-Hellman algorithm is efficient only if the prime divisors of $p - 1$ are small. This is the reason why we talked about a proper choice of p on page 165 for the intractability of the discrete log problem.

Appendix F: SHA-1

On page 229, we discussed the use of the following algorithm taken from [64].

◆ SHA-1

Modern cryptography requires custom-built hash functions to meet current standards for security. In 1995, the *Secure Hash Algorithm* (SHA-1) was developed for the NSA and standardized by NIST (see [29]). SHA-1 employs a 160-bit hash function. In 2002, NIST updated SHA-1 (see [30]) in what they called the *Secure Hash Standard* (SHS) containing specifications for 256-, 384-, and 512-bit message digests, called (respectively) SHA-256, SHA-384, and SHA-512. Naturally, these upgraded hash standards are much slower than SHA-1, yet the increased security level makes them excellent choices for modern cryptosystems. In terms of speed combined with modern-day security requirements the SHA-256 is perhaps the best choice, since the security level is 2^{128}, based on the above-established fact that the birthday attack on a message digest of size 256 bits produces an effort of about 2^{128} iterations of workload. Similarly, the SHA-1 scheme requires on 2^{80} iterations, and some cryptographers feel that this is insufficient for modern standards. Yet, from our perspective, it embodies the fundamentals of the SHA algorithms and so deserves to be studied in detail, since it provides a simple method for describing the underlying mechanisms.

◆ SHA-1

Background Assumptions

The algorithm inputs messages of maximum bitlength 2^{64} and outputs 160-bit message digests. The input is divided into blocks of 512 bits.

▼ Algorithm Steps

1. **Padding**: The input message, denoted by m, is padded so that its bitlength $\ell \equiv 448 \pmod{512}$. If ℓ is already 448 modulo 512, *before* padding, then we still pad, in this case with 512 bits. The padded message is denoted by M.

2. **Appending**: A block of 64 bits is appended to M.

3. **Buffering**: A 160-bit buffer is employed to hold the intermediate and final outputs of the algorithm. We represent the buffer having five 32-bit registers, labelled $ABCDE$. (The buffer is initialized with specific hexadecimal values that we will not cite here for the sake of simplicity.) We will denote the five initialization values by

$$(H_1H_2H_3H_4H_5) \rightarrow (ABCDE).$$

4. **Processing**: A module consisting of four rounds of twenty steps each employs three different primitive logic functions. We will, for the sake of simplicity, not describe their individual specific functions, rather we will call them f_1, f_2, and f_3. Each of these function inputs three 32-bit data strings or *words* and outputs 32-bit words. The notation is as follows.

   We will assume that there is only one 512-bit block. The procedure can be iterated to accommodate as many such blocks as necessary. M is divided into sixteen 32-bit words, denoted by m_j for $j = 0, 1, \ldots, 15$. Then each m_j is put into temporary storage $m_j \rightarrow X_j$. Then we expand the sixteen 32-bit words into eighty 32-bit words as follows.

   First, we need some notation. Let $\oplus$ be addition modulo 2, and let $\mathbf{LS_k}$ be a circular shift left of k places (for instance, see page 137, where we used a slightly different notation for the $k = 2$ case in our description of S-DES). For $j = 16, 17, \ldots, 79$, assign the following storage:

$$\mathbf{LS_1}(X_{j-16} \oplus X_{j-14} \oplus X_{j-8} \oplus X_{j-3}) \rightarrow X_j.$$

5. **Rounds**: We need to employ four constants c_i for $i = 1, 2, 3, 4$. (These have a certain hexadecimal representation that we need not cite here, again for the sake of simplicity.) Then each round operates on (the already-initialized) buffer's so-called *chaining variables* $ABCDE$, of 160 bits segmented into five 32-bit words, by updating the contents of the buffer in each step as follows (where + denotes is addition modulo 2^{32}):

   Round 1: For $j = 0, 1, \ldots, 19$, set,

$$(\mathbf{LS_5}A + f_1(B, C, D) + E + X_j + c_1, A, \mathbf{LS_{30}}(B), C, D) \rightarrow (A, B, C, D, E).$$

   Round 2: For $j = 20, 1, \ldots, 39$, set,

$$(\mathbf{LS_5}A + f_2(B, C, D) + E + X_j + c_2, A, \mathbf{LS_{30}}(B), C, D) \rightarrow (A, B, C, D, E).$$

   Round 3: For $j = 40, 1, \ldots, 59$, set,

$$(\mathbf{LS_5}A + f_3(B, C, D) + E + X_j + c_3, A, \mathbf{LS_{30}}(B), C, D) \rightarrow (A, B, C, D, E).$$

   Round 4: For $j = 60, 1, \ldots, 79$, set,

$$(\mathbf{LS_5}A + f_2(B, C, D) + E + X_j + c_4, A, \mathbf{LS_{30}}(B), C, D) \rightarrow (A, B, C, D, E).$$

6. After completion of the fourth round (or 80th step), we assign

$$(H_1 + A, H_2 + B, H_3 + C, H_4 + D, H_5 + E) \rightarrow (A, B, C, D, E).$$

   which is the output message digest of 160 bits.

One could simplify the rounds as a single set of iterations as follows. For each $i = 1, 2, 3, 4$, set the following storage for each of the values:

$$j = 20(i-1), 20(i-1)+1, \ldots, 20(i-1)+19,$$

$$(\mathbf{LS_5}A + f_i(B,C,D) + E + X_j + c_i, A, \mathbf{LS_{30}}(B), C, D) \longrightarrow (A,B,C,D,E), \tag{E.6}$$

where

$$f_4 = f_2.$$

Diagram F.1 illustrates a single step in a single round, which is actually one iteration of (E.6). Diagram F.2 on the facing page gives the complete processing of a 512-bit block (assuming step 4 above is completed).

Diagram F.1 SHA-1 Single Step

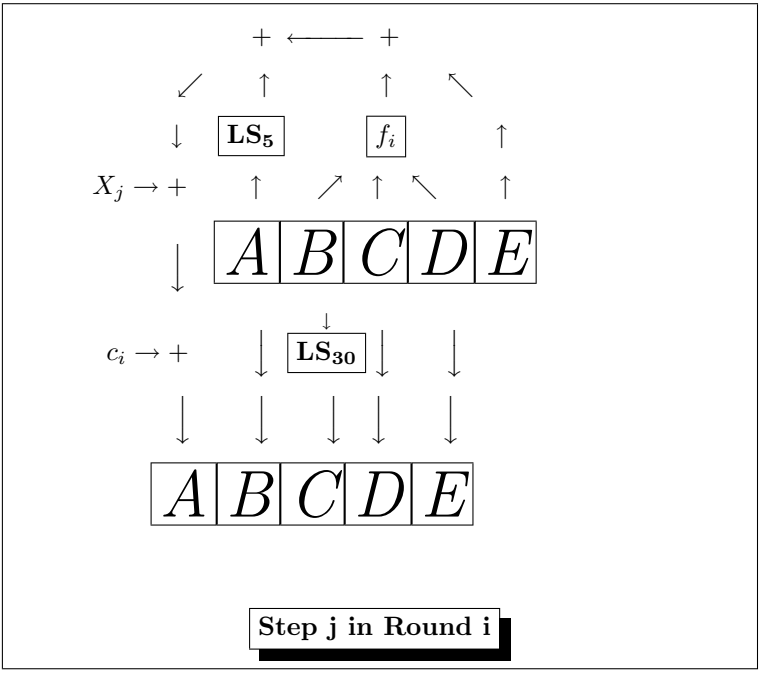
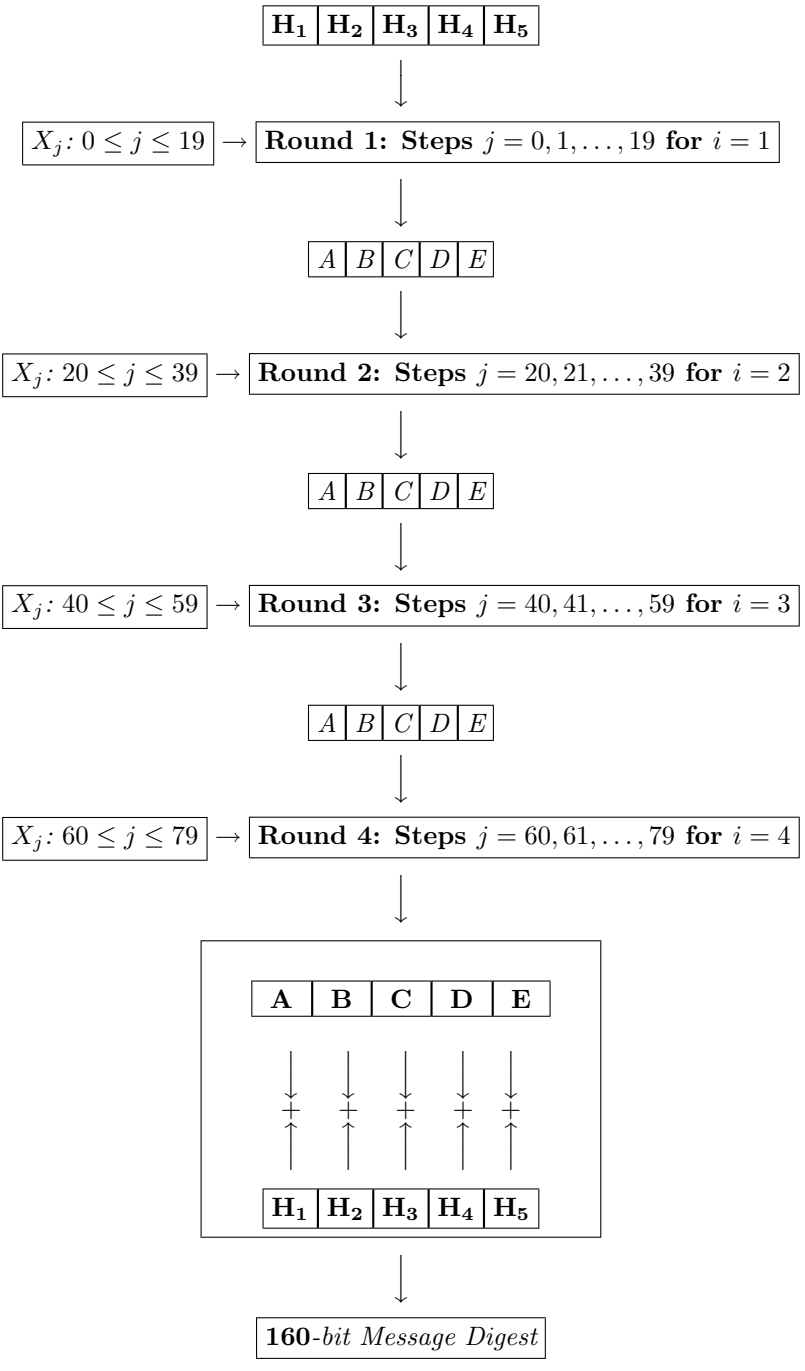


Diagram F.2 SHA-1 Processing of 512-Bit Block



Appendix G: Radix-64 Encoding

On page 232, we described the use of radix-64 encoding techniques. To describe it we need to understand that this type of *encoding* is noncryptographic in the following sense. If $\mathcal{S}$ is a message source, then we may define an injective function $f : \mathcal{S} \mapsto \mathcal{B}$, where $\mathcal{B}$ is the set of all bitstrings of finite length. This is called an *encoding* of messages from $\mathcal{S}$.

We need to understand the meaning of a *cyclic redundancy check* (CRC) as well. The mechanism for computing a CRC is a shift register, which we discussed on page 115, together with addition modulo 2. First, all values in the shift registers are initialized to 0. Then the bits of the message are shifted one at a time until the entire message has been processed into the shift register unit. The receiver uses exactly the same shift register unit to calculate the CRC for the message and to verify its agreement with the CRC transmitted by the sender.

Radix-64 is a data encoding scheme consisting of base-64 encoded data with a 24-bit CRC appended to it, as specified in RFC2440 (see [77]). This is necessary to accommodate restrictions in many email systems that only permit the use of blocks consisting of ASCII text. In essence, the radix-64 conversion, also called *ASCII armor*, may be viewed as a wrapper put on the binary message for transmission over nonbinary email channels.

Radix-64 Conversion
Table G.1

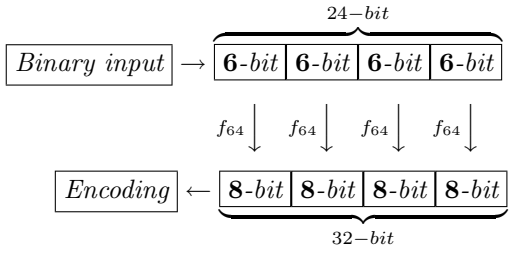
6-bit Input	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Encoding	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*
6-bit Input	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*
Encoding	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*
6-bit Input	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*
Encoding	*W*	*X*	*Y*	*Z*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
6-bit Input	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*	*41*	*42*	*43*
Encoding	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*q*	*r*
6-bit Input	*44*	*45*	*46*	*47*	*48*	*49*	*50*	*51*	*52*	*53*	*54*
Encoding	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*	*0*	*1*	*2*
6-bit Input	*55*	*56*	*57*	*58*	*59*	*60*	*61*	*62*	*63*	*PAD*	
Encoding	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*+*	*/*	*=*	

Table G.1 presents the character set of sixty-five printable characters, one of which, the = sign, is used for padding. However, in order for radix-64 encoded data to travel through mail-handling systems, there are no control characters for such systems to detect when scanned, which results in a text file that is secure against alterations made by e-mail systems. Since one character is used

for padding, there are $2^6 = 64$ characters to be employed for representation, so that each character may be used to represent 6 bits of input data. In fact, this is from where "radix-64" is derived since a 6-bit number has 64 combinations. We represent the 6-bit input data in their decimal value form for convenience in the table, while the character encodings are represented by upper- and lowercase English alphabet letters, together with the integers 0 through 9, and the symbols +, /, and lastly = for padding.

The radix-64 encoding is a mapping denoted by f_{64} acting on 6-bit inputs that are grouped into blocks that are mapped to 32-bit blocks. Each of the four 6-bit input values is mapped to an 8-bit character. In essence, this means that three bytes are mapped to four printable characters. This is illustrated in Diagram G.1.

Diagram G.1 Radix-64 Encoding Illustration



Example G.1 For instance, suppose that the text for encoding consists of the three bytes 01010000, 00100001, and 10000000, which are put into four 6-bit input values: 010100, 000010, 000110, and 000000, whose decimal representations are: 20, 2, 5, and 0. Looking at Table G.1, we get the radix-64 encodings as: UCFA.

The radix-64 conversion also appends a CRC for the purpose of detecting transmission errors. Essentially this is a *checksum*, meaning a value computed to check the validity of a data transmission, usually by detecting transmission errors. In the case of the armor checksum, a 24-bit CRC is converted to four bytes of radix-64 encoding that is prepended by an = sign to the four-byte code. For the actual mechanism by which this is done, the reader may consult [77].

Appendix H: Quantum Cryptography

Quantum cryptography is based on the following principle, which we must understand before proceeding

◆ Heisenberg's Uncertainty Principle

Heisenberg (see Biography H.2 on page 355) proposed a new mathematical foundation for mechanics where physical objects are represented by matrices involving only *observable*, namely *measurable* objects. In 1927, he published his *uncertainty* (or *indeterminacy*) principle, which says that it is logically impossible to measure *simultaneously* every facet of a given object with perfect precision. Then Bohr and Heisenberg developed a philosophy of *complementarity* to account for these new *relative* physical variables — being dependent upon an appropriate measuring process. At its core, complementarity says that a person doing the measuring interacts with the observed object, resulting in its being a function of measurement, so the object is not revealed as itself.

Heisenberg translated his uncertainty principle into a simple statement: "Even in principle, we *cannot* know the present in all detail. For that reason, everything observed is a selection from a plenitude of possibilities and a limitation on what is possible in the future." In particular, this can be illustrated simply by the inescapable fact that one cannot measure both the velocity and the position of a particle at the same time. The very act of measuring one alters or influences the other, whether the other is measured or not. The reason is that if we want to measure the position of a particle, it is necessary to use light with very short wavelength, because in order to provide data on position we need wavelengths comparable with the object from which we seek to gather data. This is where the problem arises since short wavelength light transmits a big velocity boost to the electron when it bounces off it to provide position data. On the other hand, if we want to measure velocity, then we use very long wavelengths, which alters its position. This is the built-in fundamental uncertainty formulated by Heisenberg's Indeterminacy Principle. It is precisely this uncertainty that can be utilized to generate a secret key for a quantum cryptosystem. Now we describe how to do this.

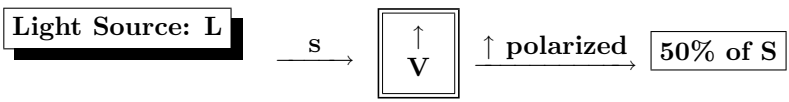
◆ Photons and a Quantum Experiment

We begin by looking at basic properties of light. The particles that constitute light are called *photons*. These photons make up light waves, which are examples of *electromagnetic waves*, meaning that they have an electric field that travels perpendicular to their associated magnetic field. Photons travelling through space vibrate (or oscillate) as they move. This vibration can be horizontal, denoted by $\rightarrow$; vertical, denoted by $\uparrow$; 45°, denoted by $\nearrow$; or 135°, denoted by $\nwarrow$. The angle of the vibration is known as the *polarization* of the photon. This is a simple type of polarization, called *linear*, meaning that as the photon propagates, the electric field stays in the same plane. This linearity assumption

simplifies the situation by allowing only four possible polarizations, rather than the infinitely many possibilities (namely all angles in between).

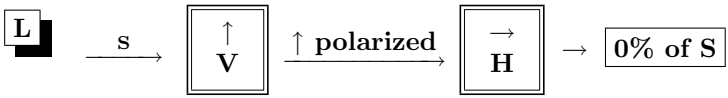
Now we need to understand a little bit about polarization of light. We are going to look at the effects of a Polaroid filter on a light source (see Biography H.1 on the following page). We will assume that the axis of the filter is oriented in one of the aforementioned four ways. Quantum theory dictates that if α is the angle that the plane of the electric field of the photon makes with the axis of a Polaroid filter, then there is a probability of $\cos^2 \alpha$ that the photon will emerge with its polarization reset to that of the filter's axis, and a probability of $1 - \cos^2 \alpha$ that it will be absorbed (to be re-emitted later as heat). For example, if the polarizer axis is vertical, then light emitted with random polarization means that if α is only slightly off vertical the photon has a high probability of passing through. If it is 45°, then it has a 50% chance of getting through, and this decreases to zero at the horizontally polarized photons. Hence, roughly 50% of the randomly emitted photons get through, and as they pass through the vertical filter they all emerge as ↑ polarizations. Call that polarization filter **V**, and the situation is illustrated in Diagram H.1.

Diagram H.1 Polarization with Filter V



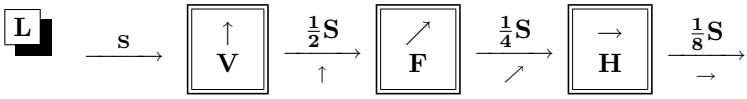
Now suppose that we use a polarizer axis that is horizontal, denoted by **H**. Then no light gets past filter **H** after having passed through filter **V** because all of the photons are polarized ↑, whose angle with **H** is $\alpha = 90°$, and the probability of getting through is $\cos^2 90° = 0$. This process is illustrated in Diagram H.2.

Diagram H.2 Polarization with Filters V and H



Suppose we now place a filter with polarizer axis 45°, denoted by **F** between **V** and **H**. Then the 50% of those photons that get through **V** now have a 50% chance of getting through **F**, and each of those will be polarized as ↗, so now 25% of the original photons got through. Now we approach **H** and each of the ↗ has a 50% chance of getting through **H**. Hence, once through all three filters, 12.5% of the original photons are emitted. Surprisingly, having put another filter between two that allowed no photons through now allows 12.5% through.

Diagram H.3 Polarization with Filters V, F, and H



This is the basic principle upon which Polaroid sunglasses work. One can demonstrate this principle using a pair of Polaroid sunglasses by taking one lens out and placing it in front of the fixed lens. There will be an orientation that is exactly the same for both lenses, so that the fixed lens has no effect on the loose lens. If the loose lens is now rotated ninety degrees, the effect will be complete blackness. This is because the polarization of the lenses are now perpendicular, so that photons that get through the one lens are blocked by the other. By rotating the loose lens forty-five degrees, one now gets an intermediate stage between complete blackness and no effect. This is because half of the photons that pass through the one lens succeed in getting through the other. Placing a third lens in front of the loose lens with axis perpendicular to the fixed lens, we get about half the light from the first two being filtered through, which is Diagram H.3 in action.

Biography H.1 Edwin Herbert Land (1909–1991) *patented a cellophane-like polarizing filter, the first to polarize light, a process that reduces light glare. In 1932, Land co-founded the Land-Wheelwright Laboratories in Boston. By 1937, Land founded the Polaroid Corporation and began to use his filters in Polaroid sunglasses and a variety of other applications. However, Land is best known for his invention and marketing of instant photography, called Polaroid photography. In 1947 he presented the Polaroid Land Camera, which took one minute to produce a finished photograph. After his retirement from Polaroid in 1980, he worked with the nonprofit Rowland Institute of Science supported by the Rowland Foundation that he founded in 1960. Land stands second only to Thomas Edison in the number of patents issued to him, more than 500. He received a number of awards for his contributions to knowledge about polarized light, photography, and colour perception. Land died in Cambridge, Massachusetts, on March 1, 1991.*

◆ Quantum Key Generation

Now we turn back to cryptography and show how the above notion of polarization and its effects can be employed to generate cryptograms.

Our goal is for Alice and Bob to communicate in binary using the polarization effects from our earlier discussion. First, we set up two binary schemes based on those polarizations.

Rectilinear Scheme: This scheme will be denoted by +, wherein a 1 is represented by ↑ and a 0 is represented by →.

Diagonal Scheme: This scheme will be denoted by $\times$, which uses $\nearrow$ for 1 and $\nwarrow$ for 0.

To send a message Alice can randomly switch between these two schemes. For example, she might send a photon string consisting of

$$\rightarrow\rightarrow\uparrow\rightarrow\nearrow\nwarrow\uparrow$$

using the combination of methods: $+ + + + \times \times +$ so the message is:

$$0\ 0\ 1\ 0\ 1\ 0\ 1.$$

Biography H.2 Werner Karl Heisenberg (1901–1976) *was born on December 5, 1901, in Würzburg, Germany. He began studying physics at the University of Munich, obtaining his doctorate in 1923 on turbulence in fluid streams. In the fall of 1924, after a brief stint studying under Max Born at Göttingen, Heisenberg went to study at the Universitits Institut for Teoretisk Fysik in Copenhagen under Bohr, whose work on the atom had inspired him. By mid-1925, he had solved a major physical problem that laid the bedrock for the development of quantum mechanics — the science that explains the discrete energy states and other forms of quantized energy, such as in the light of atomic spectra and in the phenomenon of stability exhibited by macroscopic bits of matter. (By "quantization" it is meant that observable quantities do not vary continuously but rather are formed into discrete nuggets called quanta, which in the context of energy means a discrete quantity of energy proportional in magnitude to the frequency of radiation it represents. This is the feature of quantum mechanics that makes computation, classical or quantum, possible.) From 1927 to 1941, Heisenberg was professor at the University of Leipzig, then for the next four years he was director of the Kaiser Wilhelm Institute for Physics in Berlin. In 1932, he was awarded the Nobel Prize for Physics. During World War II, he worked on the development of a nuclear reactor with Otto Hahn, one of the discoverers of nuclear fission. After the war, he organized and became director of the Max Planck Institute for Physics and Astrophysics at Göttingen. In 1958, he and the institute moved to Munich, where he died in 1976.*

▼ Quantum Key-Generation Protocol

In the following, when we say that Alice and Bob "openly" communicate, we mean that they converse over an unsecured telephone line.

1. Alice openly communicates to Bob a string of $n \in \mathbb{N}$ photons with random polarizations in the two schemes, $+$ and $\times$, with the particular polarized photons denoted by $p_1, p_2, \ldots, p_n$. Each polarized photon p_j is associated with one of the schemes $+$ or $\times$, so we denote s_j to denote that scheme under which p_j is polarized, for $j = 1, 2, \ldots, n$. For instance, if $p_1 = \rightarrow$, then $s_1 = +$.

2. Bob has a polarization detector with two settings.

(a) A + detector that can measure the polarizations ↑ and → with perfect accuracy but misinterprets ↗ or ↖ as one of ↑ or →.

(b) A × detector that can measure ↗ and ↖ with perfect accuracy but misinterprets ↑ and → as one of the ×-polarized ones.

Both settings cannot be used at the same time due to the uncertainty principle (we cannot simultaneously measure both + and × polarizations).

Bob sets the polarization detector at random settings. Sometimes the correct detector (corresponding to Alice's choice) is picked for the incoming photon, and sometimes not. We denote his received photons as $q_1, q_2, \ldots, q_n$ and his corresponding randomly selected schemes as $t_1, t_2, \ldots, t_n$.

3. Alice openly communicates to Bob the s_j for each $j = 1, 2, \ldots, n$, but not p_j. If $s_j = t_j$, then q_j is selected. Otherwise, q_j is discarded. We will label the selected ones as $q_1, q_2, \ldots, q_m$, without loss of generality, for the sake of convenience.

4. Alice openly communicates to Bob her choice of randomly selected $\ell < m$ of the p_j. They compare her ℓ of the p_j with Bob's corresponding q_j. If any of these do not match, Alice and Bob know there must be an eavesdropper (see analysis below) and they abort the run. Otherwise, they go to step 5.

5. Alice and Bob discard the ℓ randomly tested $p_j = q_j$, and maintain the remaining $m - \ell$ of them as the secret key.

Analysis: The bitstring corresponding to their agreed-upon secret key is truly random since Alice's initial photon burst was random and Bob's choice of polarization methods was random. Hence, this agreed-upon bitstring can be used for a one-time pad.

To see why the above key-generation scheme is the equivalent of a one-time pad, suppose that Mallory has also attempted to measure the initial photon burst from Alice. Then Bob and Mallory are in exactly the same situation since both of them will choose the wrong detector roughly half of the time (but not the same half). The uncertainty principle guarantees that Mallory has no means of duplicating Alice's original settings, so even if Mallory is eavesdropping on the telephone conversation, thereby gaining knowledge of the correct polarization settings, this does not help because Mallory will have measured about half of these incorrectly. Hence, this one-time pad is absolutely unbreakable, since Mallory cannot intercept Alice's message without making errors.

Mallory's presence is detected by the very act of measuring. If Alice sends a ↗, for instance, and Mallory uses the + detector, then the incoming ↗ will emerge as one of ↑ or →, since this is the only way that photon can get through Mallory's detector. If Bob measured the transformed photon with the × detector and ↖ emerges, then a correct setting of the detector will result in an incorrect

reading. In this case Mallory has altered the resulting q_j. Of course, it might also occur that Bob's reading results in the correct $\nearrow$ emerging. Therefore, Mallory has a one in four chance of being detected for each photon checked. Since ℓ of the q_j are checked in step 4, then the probability of detecting Mallory is $1 - (3/4)^\ell$. Hence, for arbitrarily large ℓ (and sufficiently large corresponding n), we can make this as close to 1 as we desire.

The above analysis shows that quantum cryptography allows key distribution between two entities (who share no prior keying material) that is provably secure against enemies with unlimited computing power, provided that the entities have access to a conventional channel, aside from the quantum channel.

Solutions to Odd-Numbered Exercises

Section 1.1

1.1 Since $ab = 1$, either both a and b are positive or both are negative. In the former case, $1 = ab \geq a \geq 1$, so $a = 1$ and similarly $b = 1$. If both are negative, then $1 = (-a)(-b) \geq -a \geq 1$, so $-a = 1$, and similarly $-b = 1$, as required.

1.3 Since $a|b$ and $b|a$, there are integers x and y such that $b = ax$ and $a = by$. Therefore, $b = ax = (by)x = b(xy)$. Since $b \neq 0$, we may use the cancellation law (see page 303) to get $xy = 1$. By Exercise 1.1, $x = \pm 1$, so $a = \pm b$.

1.5 If $2m + 1 \in \mathbb{Z}$, $m \in \mathbb{N}$, then $(2m + 1)^2 = 4m^2 + 4m + 1$=4m(4m+1)+1. Since one of m or $m + 1$ is even, then $(2m + 1)^2 = 8n + 1$, where $n = m(m + 1)/2 \in \mathbb{Z}$.

1.7 Since $a|b$ implies $am = b$ for some $m \in \mathbb{Z}$, then $cam = cb$. Therefore, $ca|cb$. Conversely, if $ca|cb$, then $can = cb$ for some $n \in \mathbb{Z}$. Since $c \neq 0$, then $an = b$ by the Cancellation law, which implies $a|b$.

1.9 $x = 5$, $y = -1$.

1.11 (a) 1155 (b) 16156 (c) 40953 (d) 10010.

1.13 (a) If $\lceil x \rceil = x = \lfloor x \rfloor + 1$, then $x \notin \mathbb{Z}$. Conversely, if $x \notin \mathbb{Z}$, then

$$-1 < x - \lfloor x \rfloor - 1 \leq \lceil x \rceil - \lfloor x \rfloor - 1 \leq x - \lfloor x \rfloor < 1.$$

Since the only integer between -1 and 1 is 0, then $\lceil x \rceil - \lfloor x \rfloor - 1 = 0$, namely $\lceil x \rceil = \lfloor x \rfloor + 1$.

(b) By Definition 1.3 on page 2, $x - 1/2 < \lfloor x + 1/2 \rfloor < x + 1/2$, so the result follows.

1.15 By Definition 1.3, $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$. Therefore, $x - 1 < \lfloor x \rfloor \leq x$.

1.17 Let $n = \lfloor x \rfloor$ and $m = \lfloor y \rfloor$, then $x = n + z$ and $y = m + w$ where $0 \leq z, w < 1$. Thus,

$$\lfloor x \rfloor + \lfloor y \rfloor = n + m \leq \lfloor n + z + m + w \rfloor = \lfloor x + y \rfloor,$$

and by Exercise 1.16, this equals

$$n + m + \lfloor z + w \rfloor \leq n + m + 1 = \lfloor x \rfloor + \lfloor y \rfloor + 1.$$

1.19 By Exercise 1.15,

$$\frac{x}{n} - 1 < \frac{x - 1}{n} \leq \left\lfloor \frac{\lfloor x \rfloor}{n} \right\rfloor \leq \frac{\lfloor x \rfloor}{n} \leq \frac{x}{n},$$

and

$$\frac{x}{n} - 1 \leq \left\lfloor \frac{x}{n} \right\rfloor \leq \frac{x}{n},$$

so by the definition of the floor function,

$$\left\lfloor \frac{x}{n} \right\rfloor = \left\lfloor \frac{\lfloor x \rfloor}{n} \right\rfloor.$$

1.21 If c is divisible by every common divisor of a and b, then in particular $g|c$. Hence, $c = \gcd(a, b)$ by the definition of the gcd.

1.23 Let g_1 and g_2 be two greatest common divisors. Given that $g_1|a$ and $g_1|b$, then $g_1|g_2$, since g_2 is a greatest common divisor. Similarly, $g_2|g_1$. By Exercise 1.3, $g_1 = g_2$.

1.25 If $\gcd(a, b) = a$, then clearly $a|b$. Conversely if $a|b$, then $a|g = \gcd(a, b)$. However, $g|b$, so $a|b$ by Exercise 1.4 (b).

1.27 Since

$$g = \gcd(a, b) = \gcd(c(a/c), c(b/c)),$$

then by Exercise 1.24, $g = c\gcd(a/c, b/c)$ from which we get the desired result, $g/c = \gcd(a/c, b/c)$.

1.29 The first equality is clear. For the last one, let $d = \gcd(a, b+am)$. Then $d|a$ and $d|(b + am)$. Therefore, $d|b$, which implies that $d|g$. Since there exist $x, y \in \mathbb{Z}$ such that,

$$d = ax + y(b + am) = a(x + ym) + yb,$$

then $g|d$. By Exercise 1.3, $d = g$.

1.31 If $\ell = b$, then $a|\ell = b$. Conversely, if $a|b$, then $an = b$ for some $n \in \mathbb{Z}$ and $\mathrm{lcm}(a, an) = an = b$.

1.33 Since $\ell \,\Big|\, ab$, then $\ell \leq ab$.

1.35 Since $b|\ell$, then $\ell = bn$ for some $n \in \mathbb{Z}$. Also, since $a|\ell$, and $g = 1$, then $a|n$ by Exercise 1.26. Therefore, by Exercise 1.33, $ab \geq \ell = ab(n/a) = bn \geq ba$. In other words, $n = a$, from which we get that $\ell = ab$.

Section 1.2

1.37 Let R_n be the number of pairs of rabbits at month n. Let M_n denote the number of pairs of mature rabbits, and I_n the number of immature rabbits. Then

$$R_n = M_n + I_n.$$

For any $n \geq 3$, $M_n = R_{n-1}$, and $I_n = M_{n-1}$, since every newborn pair at time n is the product of a mature pair at time $n - 1$. Thus,

$$R_n = R_{n-1} + M_{n-1}.$$

Moreover, $M_{n-1} = R_{n-2}$. Thus, we have

$$R_n = R_{n-1} + R_{n-2}.$$

1.39 $\mathfrak{g}^2 = ((1 + \sqrt{5})/2)^2 = (6 + 2\sqrt{5})/4 = (3 + \sqrt{5})/2 = (1 + \sqrt{5})/2 + 1 = \mathfrak{g} + 1$.

1.41 For all $n \in \mathbb{N}$, when n is divided by 4, the remainder will be $0, 1, 2$, or 3. In other words, $n = 4m$, $n = 4m + 1$, $n = 4m + 2$, or $n = 4m + 3$ for some $m \in \mathbb{N}$. So, if the number n is odd, then the remainder must be either 1 or 3. Thus, if p is an odd prime, then it is of the form $4n+1$ or $4n+3$, and $4n+3 = 4n+4-1 = 4m-1$ where $m = n + 1$.

1.43 (a) Let $\prod_{i=1}^{r} p_i^{t_i} = c$. By Exercise 1.22 on page 5, it suffices to show that any common divisor of a and b must divide c. If $d|a$ and $d|b$, then $d = \prod_{i=1}^{r} p_i^{a_i}$, where $a_i \leq m_i$, and $a_i \leq n_i$. Hence, $a_i \leq \min\{m_i, n_i\} = t_i$, so $d|c$. Hence, $c = \gcd(a, b)$.

(b) If $a|b$, then $\gcd(a, b) = a$, which means that $m_i = t_i \leq n_i$ for all i. Conversely, if $m_i \leq n_i$ for all i, then by part (a), $\gcd(a, b) = \prod_{i=1}^{r} p_i^{m_i} = a$. Therefore, $a|b$.

1.45 By Exercise 1.44, if $g = \gcd(a, b) > 1$, then there must be a prime dividing g and so dividing both a and b, a contradiction that secures the result.

1.47 By Exercise 1.46, $\sigma(2^k) = 2^{k+1} - 1 = 2 \cdot 2^k - 1$.

1.49 (a) $\sigma(69) = 96$, (b) $\sigma(96) = 252$, (c) $\sigma(100) = 217$, (d) $\sigma(64) = 127$, (e) $\sigma(2^k) = 2^{k+1} - 1$, (f) $\sigma(10000) = 24211$.

1.51 We use induction. If $n = 1$, then

$$\frac{1}{\sqrt{5}}\left[\mathfrak{g}^n - \mathfrak{g}'^n\right] = \frac{1}{\sqrt{5}}\left[\frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2}\right] = \frac{\sqrt{5}}{\sqrt{5}} = 1 = F_n.$$

Assume that $F_n = \frac{1}{\sqrt{5}}[\mathfrak{g}^n - \mathfrak{g}'^n]$. By the induction hypothesis, we have

$$F_{n+1} = F_n + F_{n-1} = \frac{1}{\sqrt{5}}\left[\mathfrak{g}^n - \mathfrak{g}'^n\right] + \frac{1}{\sqrt{5}}\left[\mathfrak{g}^{n-1} - \mathfrak{g}'^{n-1}\right],$$

and by factoring out appropriate powers, this is equal to

$$\frac{1}{\sqrt{5}}\left[\mathfrak{g}^{n-1}(1+\mathfrak{g}) - \mathfrak{g}'^{n-1}(1+\mathfrak{g}')\right].$$

By Exercise 1.39, $1 + \mathfrak{g} = \mathfrak{g}^2$. It may be similarly verified that $1 + \mathfrak{g}' = \mathfrak{g}'^2$. Hence,

$$F_{n+1} = \frac{1}{\sqrt{5}}[\mathfrak{g}^{n+1} - \mathfrak{g}'^{n+1}].$$

1.53 If

$$n = pq = x^2 - y^2 = (x - y)(x + y),$$

then either $x - y = q$ and $x + y = p$ or $x - y = 1$ and $x + y = pq$. In the former case, $x = (p+q)/2$, $y = (p-q)/2$, and in the latter case, $x = (pq+1)/2$, $y = (pq-1)/2$.

Section 1.3

1.55 Since a is even, then $a = 2n$ for some n so $a^2 = (2n)^2 = 4n^2 \equiv 0 \pmod 4$.

1.57 $x = 1380$.

1.59 $x = 43$.

1.61 $x = 62817$.

1.63 A counterexample for (a) is obtained by letting $a = 4$ and $n = 9$. In that case, $a \equiv 1 \pmod 3$, but $a^2 \not\equiv 1 \pmod 9$. A proof of (b) is obtained by merely noting if $a^2 \equiv 1 \pmod n$, then any prime $p|n$ must satisfy $p|a^2 - 1 = (a-1)(a+1)$, so $p|(a-1)$ or $p|(a+1)$, namely $a \equiv \pm 1 \pmod p$. A counterexample for (c) is given by the one for (a).

1.65 Let $y \in \mathbb{Z}$ such that $x_0 \equiv y \pmod{n/g}$. Then there exists a $z \in \mathbb{Z}$ such that $x_0 = y + zn/g$, so $ax_0 = ay + azn/g$. However, $ax_0 \equiv b \pmod{n}$, so there exists a $w \in \mathbb{Z}$ such that $ax_0 = b + wn$. Hence, $ay + azn/g = b + wn$, so

$$ay = b + n(w - z(a/g)) \equiv b \pmod{n}.$$

Conversely, if $ay \equiv b \pmod{n}$, then $ay \equiv ax_0 \pmod{n}$, so there exists a $t \in \mathbb{Z}$ such that $ay = ax_0 + nt$. Thus,

$$y = x_0 + nt/a = x_0 + (n/g)(tg/a) \equiv x_0 \pmod{n/g}.$$

1.67 (a) $n = 1, 23$, (b) $n = 1, 2, 3$, (c) $n = 1, 2, 4, 8$, (d) $n = 1, 2, 4$.

1.69 There exists a $k \in \mathbb{Z}$ such that $a = b + nk/m$. From the Division Algorithm (Theorem 1.1 on page 2), we may write k as $k = qm - j$ with $1 \leq j \leq m$. Therefore,

$$\begin{aligned} a &= b + \frac{n}{m}(qm - j) = b - j\frac{n}{m} + nq \\ &= b + n - n - j\frac{n}{m} + nq = b + \frac{n}{m}(m - j) + n(q - 1). \end{aligned}$$

Hence, $a \equiv b + \frac{n}{m}(m - j) \pmod{n}$.

1.71 (a) It suffices to show that no two elements of the set $\{ar_1 + b, ar_2 + b, \ldots, ar_n + b\}$ are congruent modulo n. If $ar_i + b \equiv ar_j + b \pmod{n}$, then $ar_i \equiv ar_j \pmod{n}$. Since $\gcd(a, n) = 1$, and $a(r_i - r_j) = nt$ for some $t \in \mathbb{Z}$, then $r_i - r_j = n(t/a)$. Therefore, one gets $r_i \equiv r_j \pmod{n}$, forcing $i = j$ since r_i and r_j are in a complete residue system modulo n.

(b) Again, it suffices to prove that no two elements of the given set are congruent modulo n. Assume, to the contrary, that for some natural numbers $i, j, k, \ell \leq n$, we have $mr_i + ns_j \equiv mr_k + ns_\ell \pmod{n}$. Then $mr_i \equiv mr_k \pmod{n}$. Since $\gcd(m, n) = 1$, then as in part (a), $r_i \equiv r_j \pmod{n}$, forcing $i = j$, as in part (a).

1.73 By the Chinese Remainder Theorem, we can solve $z \equiv y_j \pmod{x_j}$ for $j = 1, 2$ and the result follows.

1.75 Suppose that $a \equiv b \pmod{n_i}$ where $i = 1, 2, \ldots, k$. Then for all such i, n_i divides $(a - b)$. Let $n_i = \prod_{j=1}^{t} p_j^{a_j^{(i)}}$, for $1 \leq i \leq k$, where the $a_j^{(i)}$ are nonnegative, $t \in \mathbb{N}$ and the p_j are distinct primes. Let

$$T_j = \max_{1 \leq i \leq k} \{a_1^{(i)}, a_2^{(i)}, \ldots, a_t^{(i)}\}.$$

Then

$$\prod_{j=1}^{t} p_j^{T_j} = \operatorname{lcm}(n_1, \ldots, n_k) \;\Big|\; (a - b).$$

Conversely, if $a \equiv b \pmod{\ell}$, then by part (d) of Proposition 1.4 on page 19, $a \equiv b \pmod{n_i}$ for each $i = 1, 2, \ldots, k$.

1.77 (a) $x = 2$, (b) $x = 8$, (c) $x = 8$, (d) $x = 40$, (e) $x = 40$, (f) $x = 41$.

1.79 If a is its own multiplicative inverse modulo p, then $a^2 \equiv 1 \pmod{p}$. Therefore, $p|(a^2 - 1) = (a - 1)(a + 1)$. By Exercise 1.38 on page 15, either $p|(a - 1)$ or $p|(a + 1)$, which yields the result, $a \equiv \pm 1 \pmod{p}$.
Conversely, if $a \equiv \pm 1 \pmod{p}$, then $a^2 \equiv 1 \pmod{p}$, so a is its own inverse modulo p.

1.81 Since $a^{-1}b^{-1}ab = (a^{-1}a)(b^{-1}b)$, then $a^{-1}b^{-1}ab \equiv 1 \pmod{n}$.

1.83 Since $61 = 1 + 2^2 + 2^3 + 2^4 + 2^5$, then $k = 5$, $d_j = 1$ for $j = 0, 2, 3, 4, 5$ and $d_1 = 0$. Also, since $d_0 = 1$, we set $c_0 = 3$, $x_0 = 3$, and $j = 1$. The following are the j^{th} steps for $j = 1, 2, 3, 4, 5$.

(1) $x_1 \equiv x_0^2 \equiv 9 \pmod{101}$. Since $d_1 = 0$, $c_1 = c_0 = 3$.

(2) $x_2 \equiv 9^2 \equiv 81 \pmod{101}$. Since $d_2 = 1$, then

$$c_2 \equiv 3 \cdot 81 = 41 \equiv 41 \pmod{101}.$$

(3) $x_3 \equiv 81^2 \equiv 97 \pmod{101}$. Since $d_3 = 1$,

$$c_3 \equiv 97 \cdot 41 \equiv 38 \pmod{101}.$$

(4) $x_4 \equiv 97^2 \equiv 16 \pmod{101}$. Since $d_4 = 1$, then

$$c_4 \equiv 16 \equiv 38 \equiv 2 \pmod{101}.$$

(5) $x_5 \equiv 16^2 \equiv 54 \pmod{101}$. Since $d_5 = d_k = 1$, then

$$c_5 \equiv 54 \cdot 2 \equiv 7 \pmod{101}.$$

Hence,

$$3^{61} \equiv 7 \pmod{101}.$$

Section 1.4

1.85 Let $n \equiv 3 \pmod{4}$, and suppose that $x^2 \equiv -1 \pmod{n}$, for some integer x. If all primes dividing n are of the form $p \equiv 1 \pmod{4}$, then it follows that $n \equiv 1 \pmod{4}$. Hence, there is a prime $p|n$ such that $p \equiv 3 \pmod{4}$. Therefore, $x^2 \equiv -1 \pmod{p}$. By Fermat's Little Theorem, $x^{p-1} \equiv 1 \pmod{p}$. However,

$$x^{p-1} = (x^2)^{(p-1)/2} \equiv (-1)^{(p-1)/2} \equiv -1 \pmod{p}$$

since $(p-1)/2$ is odd. Therefore, we have that $-1 \equiv 1 \pmod{p}$, forcing $p = 2$, a contradiction.

1.87 Since the exercise calls for *any* $n \in \mathbb{N}$, then to prove it for $n = 1$ is sufficient, and by Fermat's Little Theorem, $q \mid (b^{q-1} - 1)$.

1.89 We use induction. If $n = 1$, then

$$g_{n+2} = g_3 = g_{n+1} + g_n = a + b = aF_1 + bF_2 = aF_n + bF_{n+1}.$$

This is the induction step. Assume the induction hypothesis, namely: $g_{m+1} = aF_{m-1} + bF_m$ for all $m \leq n$. Consider

$$g_{n+2} = g_{n+1} + g_n = aF_{n-1} + bF_n + aF_{n-2} + bF_{n-1},$$

by the induction hypothesis. The latter equals

$$a(F_{n-1} + F_{n-2}) + b(F_n + F_{n-1}) = aF_n + bF_{n+1}.$$

1.91 (a) Since $p|(a^n+1)$, then there exists an $\ell \in \mathbb{N}$ such that $a^n = p\ell - 1$. Also, by the Division Algorithm, there exist $q, r \in \mathbb{N}$ such that $p = 2nq + r$ with $1 \leq r < 2n$. If $r = 1$, then we have the first conclusion of part (a), so we assume that $r > 1$ and deduce the second conclusion. By Fermat's Little Theorem, $p|(a^{p-1} - 1)$, so

$$p \,\Big|\, (a^{2nq+r-1} - 1) = (a^n)^{2q} a^{r-1} - 1 = (p\ell - 1)^{2q} a^{r-1} - 1,$$

and by the Binomial Theorem this equals $(pt + 1)a^{r-1} - 1$ for some $t \in \mathbb{Z}$. We have shown that

$$p \,\Big|\, (a^{r-1} - 1),$$

so we may let $a^{r-1} = 1 + pz$ for some $z \in \mathbb{N}$. Set $g = \gcd(r - 1, n)$. Then $r - 1 = gw$ and $n = gk$ with $\gcd(w, k) = 1$, by Exercise 1.28 on page 5. Also, by Theorem 1.7 on page 12, there exist $u, v \in \mathbb{Z}$ such that $uk - vw = 1$. Thus, by the Binomial Theorem,

$$a^{nu} = (p\ell - 1)^u = (-1)^u + pm,$$

for some $m \in \mathbb{Z}$. Hence,

$$p \,\Big|\, (a^{nu} - (-1)^u) = a^{gku} - (-1)^u = a^{g(1+vw)} - (-1)^u = a^{gvw} a^g - (-1)^u =$$

$$(a^{gw})^v a^g - (-1)^u = (a^{r-1})^v a^g - (-1)^u = (1 + pz)^v a^g - (-1)^u,$$

and by the Binomial Theorem, this equals $(1 + py)a^g - (-1)^u$ for some $y \in \mathbb{Z}$. We have shown that

$$p \,\Big|\, (a^g - (-1)^u).$$

Since $(a^g - (-1)^u)|(a^{gk} - (-1)^{uk})$, and $p|(a^n + 1) = a^{gk} + 1$, then uk must be odd, so both u and k are odd. Therefore, in total, we have shown that

$$p \,\Big|\, (a^g + 1) = a^{n/k} + 1$$

for an odd divisor k of n.

(b) This is done in the same fashion as in the proof of part (a) and is simpler since we do not have to be concerned about parity issues. Since $p|(a^n - 1)$, then $a^n = p\ell + 1$. By the Division Algorithm, there exist $q, r \in \mathbb{Z}$ such that $p = nq + r$ with $1 \leq r < n$. If $r = 1$, then we are done, so we assume that $r > 1$. Since $p|(a^{p-1} - 1)$, then as in the proof of part (a), $p|(a^{r-1} - 1)$. Also, if $g = \gcd(n, r - 1)$, then as in the proof of part (a), $p|(a^g - 1)$. Since g is a divisor of n, we are done.

1.93 By Fermat's LittleTheorem,

$$\sum_{j=1}^{p-1} j^{p-1} \equiv \sum_{j=1}^{p-1} 1 \equiv p - 1 \equiv -1 \,(\mathrm{mod}\ p).$$

1.95 For each natural number $j \leq (p - 1)/2$, we have $p - j \equiv -j \,(\mathrm{mod}\ p)$. Thus,

$$\left[\left(\frac{p-1}{2}\right)!\right]^2 \equiv (-1)^{(p-1)/2}(p-1)! \equiv 1 \,(\mathrm{mod}\ p),$$

using Wilson's Theorem and the fact that $p \equiv 3 \pmod 4$. Hence, by Exercise 1.79 on page 33,

$$\left(\frac{p-1}{2}\right)! \equiv \pm 1 \pmod p.$$

1.97 By Euler's Theorem 1.18 on page 40, $m^{\phi(n)} \equiv 1 \pmod n$, so

$$m \cdot m^{\phi(n)-1} \equiv 1 \pmod n.$$

In other words, $m^{\phi(n)-1}$ is a multiplicative inverse of m modulo n.

1.99 This follows from a recursive use of Claim 1.1 in the proof of Theorem 1.17.

1.101 A simple solution is $n = -4$ since $-4 = 5(-1) + 1$ is a division of five heaps with the monkey getting one coconut and the sailor getting -1 coconuts. Thus, since during the process we have to divide by 5 six times, the smallest positive solution is $n = -4 + 5^6 = 15,621$.

1.103 For composite n, $\gcd(b, n) = 1$ implies that $\gcd(b, p_j) = 1$ for all $j \leq r$. Therefore, $b^{p_j - 1} \equiv 1 \pmod{p_j}$, by Fermat's Little Theorem. By hypothesis, $n - 1 = m_j(p_j - 1)$ for some $m_j \in \mathbb{Z}$. Since

$$b^{n-1} = b^{m_j(p_j-1)} = (b^{p_j-1})^{m_j},$$

then $b^{n-1} \equiv 1 \pmod{p_j}$. Thus,

$$b^{n-1} \equiv 1 \pmod{\prod_{j=1}^{r} p_j},$$

namely $b^{n-1} \equiv 1 \pmod n$.

Conversely, assume that $a^{n-1} \equiv 1 \pmod n$ for each a with $\gcd(a, n) = 1$. In particular, if a is a primitive root modulo p, for any prime divisor p of n, then $(p-1) \,\Big|\, (n-1)$, by Proposition 1.5 on page 44.

1.105 Let $a \in \mathbb{N}$ be arbitrary. Suppose that there exists a $b \in \mathbb{N}$ with $b^b \equiv a \pmod n$. If $p^2 | n$ for some prime p, then for $a = p$, $b^b \equiv p \pmod{p^2}$. However, $p^2 | b^b$ since $b > 1$ from the last congruence, so

$$p \equiv b^b \equiv 0 \pmod{p^2},$$

a contradiction. Hence, n is square-free, so $\gcd(n, \phi(n)) = 1$, since any prime dividing n cannot divide $\phi(n)$ given that $\phi(n) = \prod_{j=1}^{t}(p_j - 1)$ where $n = \prod_{j=1}^{t} p_j$.

Conversely, assume that $\gcd(n, \phi(n)) = 1$. Thus, there exist $r, s \in \mathbb{Z}$ such that $1 = rn + s\phi(n)$. Therefore, for any $a \in \mathbb{Z}$,

$$a - 1 = (a-1)rn + (a-1)s\phi(n) = xn + y\phi(n),$$

where $x = (a-1)r$, and $y = (a-1)s$. Let $b = a - xn$, and $g = \gcd(n, a)$. Then

$$b^{b-1} \equiv (a - xn)^{a-xn-1} \equiv (a - xn)^{y\phi(n)} \equiv 1 \pmod{n/g},$$

by Euler's Theorem 1.18 on page 40. Hence,

$$b^b \equiv b \equiv a \pmod{n/g}.$$

Since $b^b \equiv a \equiv 0 \pmod g$, then

$$b^b \equiv a \pmod n.$$

Section 1.5

1.107 If $g^{p-1} \not\equiv 1 \pmod{p^2}$, then g is a primitive root modulo p^2 since

$$p | \mathrm{ord}_{p^2}(g^{p-1}) | \phi(p^2) = (p-1)p.$$

Therefore, by induction on a, we may conclude that

$$p^{a-1} | \mathrm{ord}_{p^a}(g^{p-1}) | \phi(p^a),$$

for any $a \geq 2$. In other words, g is a primitive root modulo p^a. Now assume that $g^{p-1} \equiv 1 \pmod{p^2}$. Thus, by the Binomial Theorem,

$$(g+p)^{p-1} = \sum_{j=0}^{p-1} g^j p^{p-1-j} \binom{p-1}{j},$$

from which we get,

$$(g+p)^{p-1} \equiv g^{p-1} + p(p-1)g^{p-2} \equiv 1 - pg^{p-2} \pmod{p^2}.$$

Therefore, $(g+p)^{p-1} \not\equiv 1 \pmod{p^2}$, so $g+p$ is a primitive root modulo p^2, and as above, an induction on a shows that $g+p$ is a primitive root modulo p^a for any $a \geq 2$.

1.109 If $a^r \equiv 1 \pmod{m}$ for some $r \leq \phi(m)$, then $a^{r\phi(n)} \equiv 1 \pmod{m}$, so

$$a^{r\phi(n)} = 1 + ms$$

for some $s \in \mathbb{N}$. Also, $1 \equiv (a^r)^{\phi(n)} \pmod{n}$, so there is a $t \in \mathbb{N}$ such that

$$1 + nt = a^{r\phi(n)} = 1 + ms.$$

Since $\gcd(m,n) = 1$, then $m \mid t$ and $n \mid s$. Thus, $a^{r\phi(n)} \equiv 1 \pmod{mn}$, so $r = \phi(m)$ since a is a primitive root modulo mn. This shows that a is a primitive root modulo m. Reversing the roles of m and n shows that a is also a primitive root modulo n.

1.111 If m is not a primitive root modulo p, then since $\mathrm{ord}_p(m) \mid (p-1)$, there is a prime divisor q of $(p-1)/\mathrm{ord}_p(m)$. Thus, $(p-1)/q$ is a multiple of $\mathrm{ord}_p(m)$, so $m^{(p-1)/q} \equiv 1 \pmod{p}$.

1.113 Since $\gcd(-1,m) = 1$, then $1 \equiv (-1)^n \equiv -1 \pmod{m}$, so $m|2$. If $m|2$, the result is clear.

1.115 (a) $2, 3$ are all the incongruent primitive roots modulo 5, (b) $2, 6, 7, 8$ are all the incongruent primitve roots modulo 11, (c) $3, 7$ are all the incongruent primitive roots modulo 10, (d) $2, 3, 10, 13, 14, 15$ are all the incongruent primitive roots modulo 19.

Section 1.6

1.117 (a) 11 (b) 7 (c) 16 (d) 17 (e) 15 (f) 10.

1.119 Let $\operatorname{ind}_a^p(c) = x$, $\operatorname{ind}_a^p(b) = y$, and $\operatorname{ind}_b^p(c) = z$.
Since
$$b^{zy} \equiv (b^z)^y \equiv c^y \equiv (a^x)^y \equiv (a^y)^x \equiv b^x \,(\mathrm{mod}\ p),$$
then by Fermat's Little Theorem, $zy \equiv x \,(\mathrm{mod}\ p-1)$, as required.

1.121 For any primitive root a modulo p^c, congruence (1.3) yields,
$$e \cdot \operatorname{ind}_a^{p^c}(x) \equiv \operatorname{ind}_a^{p^c}(b) \,(\mathrm{mod}\ \phi(p^c)).$$
The result now follows via Exercise 1.64 on page 32.

1.123 We compute the following where $\overline{s_j}$ means the least nonnegative residue of s_j modulo $M_{13} = 8191$: $\overline{s_2} = 14$, $\overline{s_3} = 194$, $\overline{s_4} = 4870$, $\overline{s_5} = 3953$, $\overline{s_6} = 5970$, $\overline{s_7} = 1857$, $\overline{s_8} = 36$, $\overline{s_9} = 1294$, $\overline{s_{10}} = 3470$, $\overline{s_{11}} = 128$, $\overline{s_{12}} = 0$. Thus, M_{13} is prime by the Lucas-Lehmer test.

Section 1.7

1.125 If $1 \le i, j \le (p-1)/2$, then it is easy to show that unless $i = j$, $i^2 \not\equiv j^2 \,(\mathrm{mod}\ p)$. Thus, there are $(p-1)/2$ quadratic residues, namely $1^2, 2^2, \ldots, [(p-1)/2]^2$. Since these are all the squares, then the remainder are nonresidues. This establishes the first Lengendre identity.

For the second Legendre identity, consider the sum over j modulo p so the sum goes over a complete residue system with j as index of summation if and only if it goes over a complete residue system modulo $j + a$ as index of summation. Hence,
$$\sum_{j=0}^{p-1} \left(\frac{(j-a)(j-b)}{p} \right) = \sum_{j \not\equiv 0 \,(\mathrm{mod}\ p)} \left(\frac{j(j+a-b)}{p} \right).$$
If $a \equiv b \,(\mathrm{mod}\ p)$, then the sum becomes
$$\sum_{j=1}^{p-1} \left(\frac{j^2}{p} \right) = p - 1.$$
If $a \not\equiv b \,(\mathrm{mod}\ p)$, then we may consider the sum over all $j \not\equiv 0 \,(\mathrm{mod}\ p)$. In this case, there is a multiplicative inverse j^{-1} of j modulo p. Also, as the sum goes over a reduced residue system modulo p with j as index of summation, then so does the sum with j^{-1} as index of summation. Thus,
$$\sum_{j \not\equiv 0 \,(\mathrm{mod}\ p)} \left(\frac{j(j+a-b)}{p} \right) = \sum_{k \not\equiv -1 \,(\mathrm{mod}\ p)} \left(\frac{1+k}{p} \right) = \sum_{k=0}^{p-2} \left(\frac{1+k}{p} \right) - \left(\frac{1}{p} \right),$$
where the summand on the right is 0 by the first Legendre identity, and $-(\frac{1}{p}) = -1$ as required.

1.127 If $n = b^2$, then $(\frac{m}{b^2}) = (\frac{m}{b})^2 = 1$ for all $m < n$ such that $\gcd(m, n) = 1$. Conversely, if $(\frac{m}{n}) = 1$ for all natural numbers $m < n$ relatively prime to n, and n is not a perfect square, then $n = b^2 t$ where $t > 1$ is squarefree. Let $p \,\big|\, t$ be a prime and let $p^c || n$ where $c \ge 1$ is odd since t is squarefree. Let r be a quadratic nonresidue modulo p. By the Chinese Remainder Theorem, there is a solution $x = a$ to the system of congruences,

$$x \equiv r \pmod{p} \text{ and } x \equiv 1 \pmod{n/p^c}.$$

Therefore,

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p^c}\right)\left(\frac{a}{n/p^c}\right) = \left(\frac{a}{p^c}\right) = \left(\frac{a}{p}\right) = \left(\frac{r}{p}\right) = -1,$$

a contradiction, so $t = 1$.

1.129 Since $\left(\frac{4}{p}\right) = 1$, then

$$\left(\frac{a}{p}\right)\left(\frac{f(x)}{p}\right) = \left(\frac{4a^2x^2 + 4abx + 4ac}{p}\right) = \left(\frac{(2ax+b)^2 - \Delta}{p}\right).$$

As x ranges over $0, 1, \ldots, p-1$, so does $2ax + b$ modulo p. Thus,

$$\sum_{x=0}^{p-1}\left(\frac{a}{p}\right)\left(\frac{f(x)}{p}\right) = \sum_{j=0}^{p-1}\left(\frac{j^2 - \Delta}{p}\right).$$

However, $\Delta \equiv 0 \pmod{p}$, so

$$\sum_{x=0}^{p-1}\left(\frac{a}{p}\right)\left(\frac{f(x)}{p}\right) = \sum_{j=1}^{p-1}\left(\frac{j^2}{p}\right) = p - 1.$$

Therefore, multiplying the above through by $\left(\frac{a}{p}\right)$, we get

$$\sum_{x=0}^{p-1}\left(\frac{f(x)}{p}\right) = (p-1)\left(\frac{a}{p}\right).$$

Section 1.8

1.131 For all $n \in \mathbb{N}$, $n! < n^n$. Therefore, $n! = O(n^n)$.

1.133 $f(b, n) = b^n$.

1.135 This follows from the fact that $\ln(n)/n \to 0$ as $n \to \infty$.

1.137 This follows from the fact that $f << g$ implies that $\ln(f) << \ln(g)$.

1.139 If each integer less than n has at most t bits, then $n!$ has at most $n(t+1)$ bits, and $n(t+1) = O(nt)$. Thus, in the $n - 2 = O(n)$ multiplications involved in computing $n!$, we multiply an integer with at most t bits by an integer with $O(nt)$ bits. This requires $O(nt^2)$ bit operations. We do this $O(n)$ times, so the total number of bit operations required is $O(nt^2)O(n) = O(n^2t^2)$. However, we know that $t = O(\log n)$ (see page 68). Hence, $O(n^2t^2) = O(n^2\log^2(n))$.

1.141 This is an application of Theorem 1.25 on page 70.

Section 2.1

2.1 The die is cast.

2.3 To labor is to pray.

2.5 Force without mind falls by its own weight.

2.7 Time is money.

Section 2.2

2.9 All Quiet on the Western Front.

2.11 Every sin is the result of a collaboration.

2.13 Love is a kind of warfare.

2.15 Stake life upon truth.

2.17 (01100001000110100000100010010000010100011010000100).

2.19 The decimal equivalents of the bitstrings of length five are $12, 4, 13, 0, 17, 4, 2, 17, 20, 4, 11, 1, 20, 19, 12, 0, 13, 8, 18, 10, 8, 13, 3$, which convert via Table 2.2 to:

Men are cruel, but Man is kind.

2.21 Since the ciphertext has forty-eight letters, we know that the column under **Y** will have six letters and there will be seven in each of the others. Thus, the placement is as follows.

V	**I**	**C**	**T**	**O**	**R**	**Y**
6	2	1	5	3	4	7
D	A	A	V	V	G	D
A	V	F	F	F	D	A
F	A	X	F	D	A	F
F	A	X	D	F	V	F
X	X	F	V	V	G	F
G	A	G	F	G	F	F
F	G	F	G	X	A	

Taking these off by rows, we get:

DAAVVGDAVFFFDAFAXFDAFFAXDF
VFXXFVVGFGAGFGFFFGFGXA.

Using Table 2.4, we decipher to get:

Always act as if you were seen.

2.23 Since there are thirty-six letters in the ciphertext, the column under **V** has six letters and all others have five letters. This results in the following.

V	**I**	**C**	**T**	**O**	**R**	**Y**
6	2	1	5	3	4	7
V	F	X	X	F	V	F
X	D	A	G	X	F	G
V	G	X	X	X	A	X
V	D	A	A	V	A	X
A	V	F	G	D	A	X
A						

Taking these off by rows, we get:

VFXXFVFXDAGXFGVGXXXAXVDAAVAXAVFGDAXA.

Using Table 2.4, we decipher to get:

You have won, Galilean.

2.25 Common Sense is not so common. (There are two uses of a z between the two occurences of mm in common that are removed.)

2.27 Love is blind. (A z at the end was removed since it was added as filler.)

2.29 But I'm not so think as you drunk I am.

2.33 Passion, I see, is catching.

Section 2.3

2.35 The numerical equivalents are

$$21, 14, 7, 23, 19, 12, 3, 19, 0, 5, 9, 11, 21, 9, 24, 1, 23, 25, 11, 23.$$

Thus, we calculate as follows, where all congruences are assumed modulo 26. $m_1 = c_1 - k_1 = 21 - 2 = 19$, $m_2 = c_2 - k_2 = 14 - 7 = 7$, $m_3 = c_3 - k_3 = 7 - 3 = 4$, $m_4 = c_4 - m_1 = 23 - 19 = 4$, $m_5 = c_5 - m_2 = 19 - 7 = 12$, $m_6 = c_6 - m_3 = 19 - 4 = 15$, $m_7 = c_7 - m_4 = 12 - 4 = 8$, $m_8 = c_8 - m_5 = 3 - 12 \equiv 17$, $m_9 = c_9 - m_6 = 19 - 15 = 4$, $m_{10} = c_{10} - m_7 = 0 - 8 \equiv 18$, $m_{11} = c_{11} - m_8 = 5 - 17 \equiv 14$, $m_{12} = c_{12} - m_9 = 9 - 4 = 5$, $m_{13} = c_{13} - m_{10} = 11 - 18 \equiv 19$, $m_{14} = c_{14} - m_{11} = 21 - 14 = 7$, $m_{15} = c_{15} - m_{12} = 9 - 5 = 4$, $m_{16} = c_{16} - m_{13} = 24 - 19 = 5$, $m_{17} = c_{17} - m_{14} = 1 - 7 \equiv 20$, $m_{18} = c_{18} - m_{15} = 23 - 4 = 19$, $m_{19} = c_{19} - m_{16} = 25 - 5 = 20$, $m_{20} = c_{20} - m_{17} = 11 - 20 \equiv 17$, $m_{21} = c_{21} - m_{18} = 23 - 19 = 4$.

Then via Table 2.2, we get the letter equivalents:

The empires of the future.

2.37 The answer is $\lambda(n)$, the *Carmichael Function*, defined as follows. If $n = 2^a \prod_{j=1}^{k} p_j^{a_j}$ is the canonical prime factorization of $n \in \mathbb{N}$, namely $2 < p_1 < p_2 < \cdots < p_k$, then

$$\lambda(n) = \begin{cases} \phi(n) & \text{if } n = 2^a, \text{ and } 1 \le a \le 2, \\ 2^{a-2} = \phi(n)/2 & \text{if } n = 2^a, a > 2, \\ \operatorname{lcm}(\lambda(2^a), \phi(p_1^{a_1}), \ldots, \phi(p_k^{a_k})) & \text{if } k \ge 1. \end{cases}$$

(See [61] for more information on this function. The Carmichael Function was first discussed by Cauchy [15] in 1841.)

Section 2.4

2.39 If $s_{-1} = (k_{\ell-1} k_{\ell-2} \ldots k_1 k_0)$, then $s_0 = (\sum_{j=0}^{\ell-1} c_j k_j, k_{\ell-1} \ldots k_2 k_1)$. Thus, using the matrix C defined on page 118, we get that $C s_{-1} = s_0$. Since $\det(C) = c_0 = 1$, then C is nonsingular. Since $s_j = C^{j+1} s_{-1}$ for $j = 1, 2, \ldots, P-1$ where P is the period length of the LFSR, then

$$\det(C^{j+1}) = (\det(C))^{j+1} = 1,$$

so C^{j+1} is nonsingular. Also, s_{-1} is not the zero state. Hence, the LFSR has no zero state.

2.41 If $c_0 = 0$, then the LFSR with the intiial state $s_{-1} = (00 \ldots 01)$ generates only zero states since $s_0 = (\sum_{j=0}^{\ell-1} c_j k_j, 0 \ldots 00)$, where $\sum_{j=0}^{\ell-1} c_j k_j = c_0 = 0$. Hence, there cannot exist a $P \in \mathbb{N}$ such that $s_{-1} = s_{p-1}$.

2.43 If there is a recurrence relation of length less than n, then one row of M is a linear combination of the other rows. Therefore, $\det(M) \equiv 0 \pmod 2$.

2.45 Given s_j for $j = -1, 0, \ldots, P-1$ is the binary representation of an $n \in \mathbb{N}$ where $n \leq 2^\ell - 1$, we need only count the number of odd and the number of even numbers. If the right-most entry of s_j is 0, then n is even and if a 1 occurs there, then n is odd. Hence, there are $2^{\ell-1}$ odd numbers, and $2^{\ell-1}$ even numbers, but we exclude the zero binary digit from the even ones, and the result follows.

Section 2.5

2.47 $c_j \oplus E_k(c_{j-1}) = m_j \oplus k_j \oplus k_j = m_j$.

Section 3.1

3.1 vanity.

3.3 grants.

3.5 $\mathbf{IP}(m) = (00110101)$.

3.7 $\mathbf{S_0}(1110) = (11)$ and $\mathbf{S_1}(1110) = (10)$.

3.9 $\mathbf{EP}(x) \oplus SK = c(\mathbf{EP}(x)) \oplus c((SK))$ since complementation cancels out $\oplus$ addition. Also, $c(\mathbf{EP}(x)) = \mathbf{EP}(c(x))$. Thus, $f_{SK}(c(t)) = f_{SK}(c)$, so

$$c(\mathbf{E}_k(m)) = \mathbf{E}_{c(k)}(c(m)).$$

Section 4.2

4.1 Let m, m' be generators of F_p^*, and let $\beta \in \mathbb{F}_p^*$. Set $x = \log_m(\beta)$, $y = \log_{m'}(\beta)$, and $z = \log_m(m')$. Then

$$m^x = \beta = (m')^y = (m^z)^y = m^{xy},$$

so $x \equiv zy \pmod{p-1}$. Hence,

$$\log_{m'}(\beta) = y \equiv xzs^{-1} \equiv (\log_m(\beta))(\log_m(m'))^{-1} \pmod{p-1}.$$

Hence, any algorithm that computes logs to be m can be used to compute logs to any other base m' that is a generator of $\mathbb{F}_p^*$.

4.3 The plaintext numerical values are given by

$$m = (5, 11, 0, 18, 7, 8, 13, 19, 7, 4, 15, 0, 13),$$

which translates via Table 2.2 using the deciphering key $d = 111$ to

flash in the pan.

4.5 The plaintext numerical values are given by

$$m = (19, 7, 8, 13, 10, 0, 1, 14, 20, 19, 8, 19),$$

which translates via Table 2.2 using the deciphering key $d = 47$ to

think about it.

4.7 $\alpha^x \equiv 2^{25} \equiv 412 \equiv X \pmod{877}$; and $\alpha^y \equiv 2^3 \equiv 8 \equiv Y \pmod{877}$. Also,

$$Y^x \equiv 8^{25} \equiv 794 \pmod{877},$$

$$X^y \equiv 412^3 \equiv 794 \pmod{877},$$

and so $k \equiv \alpha^{xy} \equiv 794 \pmod{877}$.

4.9 $\alpha^x \equiv 3^{69} \equiv 919 \equiv X \pmod{1193}$; and $\alpha^y \equiv 3^{96} \equiv 30 \equiv Y \pmod{1193}$. Also,

$$Y^x \equiv 30^{69} \equiv 489 \pmod{1193},$$

$$X^y \equiv 919^{96} \equiv 489 \pmod{1193},$$

and so $k \equiv \alpha^{xy} \equiv 489 \pmod{1193}$.

4.11 If $\alpha = p - 1$, then

$$X \equiv \alpha^x \equiv (p-1)^x \equiv (-1)^x \equiv \pm 1 \pmod{p},$$

and

$$Y \equiv \alpha^y \equiv (p-1)^y \equiv (-1)^y \equiv \pm 1 \pmod{p}.$$

Thus, $X^y \equiv (\pm 1)^y \equiv k \equiv Y \pmod{p}$, forcing $k = \pm 1$.

4.13 Suppose that $m = r_0 + r_1 q$ and $m_1 = r_0' + r_1' q$ and

$$\alpha^{r_0} \beta^{r_1} \equiv \alpha^{r_0'} \beta^{r_1'} \pmod{p}.$$

Since $\beta \equiv \alpha^a \pmod{p}$, then the above congruence may be written as

$$\alpha^{a(r_1 - r_1') - (r_0' - r_0)} \equiv 1 \pmod{p}.$$

However, α is a primitive root modulo p from which it follows that

$$\alpha^b \equiv 1 \pmod{p} \text{ if and only if } b \equiv 0 \pmod{p-1}.$$

Hence,

$$a(r_1 - r_1') \equiv (r_0' - r_0) \pmod{p-1}.$$

If $g = \gcd(r_1 - r_1', p-1)$, then there are exactly g solutions to the latter congruence (see Section 1.3). Since $(p-1)/2$ is prime, and $0 \le r_1, r_1' \le q-1$, we must have that $-(q-1) \le r_1 - r_1' \le q-1$. So if $r_1 - r_1' \ne 0$, then $q > |r_1 - r_1'|$, so $g = 1, 2$. We have shown that there are only two possible values for a, and by calculating α^a for each of these, exactly one will give β. Hence, a is determined. Note that we cannt have

$$r_1 - r_1' = 0$$

since, if it were, then $r_0 = r_0' \equiv 0 \pmod{p-1}$. Given that

$$-(q-1) \le r_0' - r_0 \le q - 1,$$

it follows that $r_0 = r_0'$, forcing $m = m_1$, a contradiction to the assumed distinctness of the messages.

Section 4.3

4.15 $\phi(n) = 144900$ and $1 = 11d + 144900x$ is solved by $x = -4$ and $d = 52691$, so $c^d \equiv 9876 \equiv m \pmod{n}$.

4.17 $\phi(n) = 1755280$ and $1 = 13d + 1755280x$ is solved by $x = -11$ and $d = 1485237$, so $c^d \equiv 1111111 \equiv m \pmod{n}$.

4.19 Solving $1 = 74597e + 969760x$ yields $e = 13$ and $x = -1$. Since $c^e \equiv 2134 \pmod{n}$, we accept.

4.21 As in the above we get $e = 19$ and since $c^e \equiv 8872 \equiv m \pmod{n}$, we accept.

4.23 $p = 599$ and $q = 859$.

4.25 $p = 1181$ and $q = 1471$.

4.27 $p = 1097$ and $q = 2351$.

4.29 $p = 1021$ and $q = 3329$.

Section 4.4

4.31 $(\alpha^b)^{-a} \equiv (596)^{-71} \equiv 623 \pmod{1973}$, so

$$(\alpha^b)^{-a} m \alpha^{ab} \equiv 623 \cdot 146 \equiv 200 \equiv m \pmod{1973}.$$

4.33 $(\alpha^b)^{-a} \equiv (1093)^{-19} \equiv 3243 \pmod{3359}$, so

$$(\alpha^b)^{-a} m \alpha^{ab} \equiv 3243 \cdot 2530 \equiv 2112 \equiv m \pmod{3359}.$$

4.35 $\delta \equiv 2391^{335} \cdot 335^{2367} \equiv 2212 \pmod{3023}$ and $\sigma \equiv 5^{203} \equiv 2212 \pmod{3023}$, so Bob accepts.

4.37 $\delta \equiv 5979^{1723} \cdot 1723^{7045} \equiv 2031 \pmod{7481}$ and $\sigma \equiv 6^{487} \equiv 2031 \pmod{7481}$, so Bob accepts.

Section 5.1

5.1 Let $r|m$. It suffices to prove the result for the case where r is prime. Let $m = rt$ for some $t \in \mathbb{N}$. Thus, $rt = 0$ in R so r is a zero divisor in R (see page 25). Let $I = \{x \in R : xt = 0\}$. Then I is an ideal in R and $r \in I$ (see Definition A.22 on page 317). Let M be a maximal ideal in R containing r (see Definition A.23), and set $F = R/M$, which is a field (see Theorem A.15). Since $\alpha^{s/p} - 1$ is a unit in R for any prime divisor p of s, then the order of α modulo M must be s. In other words, $\alpha^s - 1 = 0$ in M but $\alpha^j - 1 \neq 0$ in M for any nonnegative $j < s$. (Otherwise, there would be a unit in M forcing $M = R$, contradicting Definition A.23.) Since $f(x) \in \mathbb{Z}/m\mathbb{Z}[x]$ and $r = 0$ in F with $r|m$, then we may assume without loss of generality that $f(x) \in \mathbb{Z}/r\mathbb{Z}[x]$. Thus, $f(\alpha^r) = 0$, so $\alpha^r = \alpha^{m^j}$ for some nonnegative $j < k$. Since the order of α modulo M is s, then $r \equiv m^j \pmod{s}$.

5.3 By Exercise 1.30 on page 5, $\gcd(2^p - 1, 2^q - 1) = 2^{\gcd(p.q)} - 1 = 1$.

5.5 Since $n = a \cdot b + 1 = 24 \cdot 4409 + 1 = 105817$, $m^{n-1} \equiv 2^{105816} \equiv 1 \pmod{n}$, $\gcd(m^{(n-1)/q} - 1, n) = \gcd(2^{24} - 1, 105817) = 1$, and clearly $b = q = 4409 > \sqrt{n}$, then n is prime.

5.7 Since $n = 40961 = 2^{13} \cdot 5 + 1$, $c^{(n-1)/2} \equiv 3^{20480} \equiv -1 \pmod{n}$, and $c = 3$ is a quadratic nonresidue modulo n, then n is prime.

5.9 Since $n = 16547 = 2 \cdot 8273 + 1 = 2q + 1$ where $q = 8273$ is prime and

$$m^{(n-1)} \equiv 2^{16546} \equiv 1 \pmod{n},$$

while $m^{(n-1)/q} \equiv 2^2 \not\equiv 1 \pmod{n}$, then n is prime.

5.11 Since $n-1 = 8272 = 2^4 \cdot 11 \cdot 47$, then select $m_2 = 3$ for which $m_2^{8272} \equiv 1 \pmod{n}$, and

$$m_2^{(n-1)/2} \equiv 3^{4136} \equiv -1 \pmod{n}.$$

Also, for $m_{11} = m_{47} = 2$,

$$m_{11}^{n-1} \equiv 1 \equiv m_{47}^{n-1} \pmod{n},$$

while $m_{11}^{(n-1)/11} \equiv 2^{752} \equiv 3581 \pmod{n}$; and $m_{47}^{(n-1)/47} \equiv 2^{176} \equiv 165 \pmod{n}$. Hence, n is prime.

5.13 Suppose that $p^t || n$, where p is any prime dividing n. Also, let a be a generator of $(\mathbb{Z}/p^t\mathbb{Z})^*$. By the Chinese Remainder Theorem 1.12 on page 26, there exists an element $b \in (\mathbb{Z}/n\mathbb{Z})^*$ satisfying the congruences,

$$b \equiv a \pmod{p^t} \text{ and } b \equiv 1 \pmod{n/p^t}.$$

Thus, by hypothesis,

$$b^{n-1} \equiv a^{n-1} \equiv 1 \pmod{p^t}.$$

Therefore, $\text{ord}_{p^t}(b) = \phi(p^t) = p^{t-1}(p-1) \,\Big|\, (n-1)$ by Proposition 1.5 on page 44. Hence, $t = 1$ as required.

Section 5.2

5.15 Since $n = 7331$ and $n - 1 = 2 \cdot 3665 = 2m$ with $2^m \equiv -1 \pmod{n}$, then we conclude with n being probably prime, and indeed it is.

5.17 Since $n = 2152302898747$ and $n-1 = 2 \cdot 1076151449373$, with $5^m \equiv -1 \pmod{n}$, then we declare n to be a probable prime. However, $n = 6763 \cdot 10627 \cdot 29947$ is the canonical prime factorizarion. Indeed, it is known that this is the smallest strong pseudoprime to all bases $2, 3, 5, 7, 11$.

5.19 If $n > 1$ with $2^n \equiv 1 \pmod{n}$, then there is a smallest prime p dividing n. Thus, by Proposition 1.5 on page 44, $\text{ord}_p(2) \,\Big|\, n$. However,

$$2^{p-1} \equiv 1 \pmod{p},$$

by Fermat's Little Theorem, so by Proposition 1.5 again,

$$\text{ord}_p(2) \,\Big|\, (p-1).$$

Hence, $\text{ord}_p(2) < p$. But since $1 < \text{ord}_p(2) \,\Big|\, n$, this contradicts the minimality of p.

5.21 If $n = p^a$ where $a > 1$, then $(p^a - (D/n)) \,\Big|\, \psi_D(n) = p^a - p^{a-1}(D/p)$. Thus, $p^a - p^{a-1} < p^a - 1 \leq p^a - (D/n) \leq p^a - p^{a-1}(D/p)$, so $(D/p) = -1$. Therefore, $p^a \pm 1$ divides $p^a + p^{a-1}$, which cannot happen since

$$2p^a - 2 > p^a + p^{a-1} \geq p^a \pm 1 \geq p^a - 1.$$

If $k > 1$

$$\psi_D(n) \leq \frac{1}{2^{k-1}} \prod_{j=1}^{k} p_j^{a_j - 1}(p_j + 1) = 2n \prod_{j=1}^{k} \frac{1}{2}\left(1 + \frac{1}{p_j}\right) \leq$$

$$2n \cdot \frac{2}{3} \cdot \frac{3}{5} \cdots \leq \frac{4n}{5} < n - 1,$$

since both $k > 1$ and $n > 5$, contradicting the hypothesis: $(n - (D/n)) \Big| \psi_D(n)$.

Section 5.3

5.23 We have that (a) implies (b), *a fortiori*. If (b) holds, then by Exercise 1.103 on page 43, n is a Carmichael number. Thus, by Exercise 5.22, n is squarefree and by Exercise 1.103, $(p-1) \Big| (n-1)$ for all primes p dividing n. Therefore, (b) implies (c). It remains to show that (c) implies (a). If (c) holds, we need to show $n \Big| (a^n - a)$ for all $a \in \mathbb{Z}$. However, since n is squarefree, it suffices to show that each prime p dividing n also divides $(a^n - a)$. Since $a^{p-1} \equiv 1 \pmod{p}$ for each a relatively prime to p by Fermat's Little Theorem, then since $n - 1 = (p-1)s$ for some $s \in \mathbb{N}$, then $a^{n-1} \equiv a^{s(p-1)} \equiv 1 \pmod{p}$, so $a^n \equiv a \pmod{p}$. Lastly, if $p \Big| a$, then $a^{n-1} \equiv a \equiv 0 \pmod{p}$, so we have completed the proof.

5.25 Use the extended Euclidean algorithm 1.7 on page 12 on e and $2n'$ to find integers d and m' such that $ed + 2n'm' = 1$. Then destroy all records of p, q, n', and m', and keep d as the private key (trapdoor). Thus, $m^e \equiv c \pmod{n}$ is the enciphering function and $c^d \equiv m \pmod{n}$ is the deciphering function where $ed \equiv 1 \pmod{\phi(n)}$.

5.27 Let $n - 1 = 2^t m$ where m is odd. Since n is a strong pseudoprime to base a, then either

$$a^m \equiv \pm 1 \pmod{n}$$

or

$$a^{2^i m} \equiv -1 \pmod{n}$$

for some positive $i < t$. In the former case

$$(a^{2j+1})^m \equiv (\pm 1)^{2j+1} \equiv \pm 1 \pmod{n}$$

and in the latter case

$$(a^{2j+1})^{2^i m} \equiv (-1)^{2j+1} \equiv -1 \pmod{n}.$$

In any case, n is a strong pseudoprime to base a^{2j+1}.

Section 6.1

6.1 $10817 = 29 \cdot 373$.

6.3 $767 = 13 \cdot 59$.

6.5 $87611 = 79 \cdot 1109$.

Section 6.2

6.7 $n = 3090847 = 1481 \cdot 2087$.

6.9 $n = 3774403 = 1123 \cdot 3361$.

6.11 $n = 35923031 = 5039 \cdot 7129$.

6.13 $n = 63382447 = 7757 \cdot 8171$.

6.15 $n = 82979779 = 8999 \cdot 9221$.

Section 6.3

6.17 $n = 1324237 = 1021 \cdot 1297$.

6.19 $n = 5951129 = 2281 \cdot 2609$.

Section 6.4

6.21 $n = 3191491 = 2311 \cdot 1381$.

6.23 $n = 42723991 = 5711 \cdot 7481$.

Section 6.5

6.25 $n = 561707 = 331 \cdot 1697$.

6.27 $n = 20235773 = 3557 \cdot 5689$.

6.29 $n = 72425447 = 7673 \cdot 9439$.

Section 7.1

7.1 It has the same reason as it was replaced for the standard by AES, namely it is insecure in the modern day.

Section 8.1

8.1 Throw two dice three times, for each word, recording the numbers after each throw. Then after $3n$ ($n \in \mathbb{N}$) throws, one has an n-word passphrase. Assuming the dice are fair, this is a random selection.

8.3 This tunneling may be used to advantage to connect through a firewall to upload and download mail securely, as well as browse WWW sites. However, Mallory can easily establish an open connection to an Internet telnet server, for instance, or any other much more malicious intervention.

Section 8.4

8.5 (a) Leaving a voice mail message to another party allows them to spoof an identity.
(b) Having a cold, laryngitis, etc. can alter the voice print.

Bibliography

[1] M. Agrawal, N. Kayal, and N. Saxena, *Primes is in P*, Annals of Math. **160** (2004), 781–793. (*Cited on pages 194–196.*)

[2] W.R. Alford, A. Granville, and C. Pomerance, *There are infinitely many Carmichael numbers*, Ann. Math. **140** (1994), 703–722. (*Cited on page 192.*)

[3] D. Atkins, M. Graff, A.K. Lenstra, and P.C. Leyland, *The magic words are SQUEAMISH OSSIFRAGE* in **Advances in Cryptology** — ASIACRYPT '94, Springer-Verlag, Berlin, LNCS **917**, (1995), 263–277. (*Cited on page 219.*)

[4] E. Biham and L.R. Knudsen, *Cryptanalysis of the ANSI X9.52 CBCM mode*, J. Cryptol. (2002), 47–59. (*Cited on page 150.*)

[5] E. Biham and A. Shamir, *Differential cryptanalysis of the full 16-round DES*, **Advances in Cryptology** — CRYPTO '92, Springer-Verlag (1993), 487–496. (*Cited on page 145.*)

[6] E. Biham and A. Shamir, **Differential Cryptanalysis of the Data Encryption Standard**, Springer-Verlag, New York (1993). (*Cited on page 145.*)

[7] D. Bleichenbacher, *Generating ElGamal signatures without knowing thesecret key*, in **Advances in Cryptology** — EUROCRYPT '96, Springer-Verlag, Berlin, LNCS **1070** (1996), 10–18. (*Cited on page 185.*)

[8] D. Boneh, R.A. DeMillo, and R.J. Lipton, *On the importance of checking cryptographic protocols for faults*, in **Advances in Cryptology**, EUROCRYPT '97, Springer-Verlag, Berlin, LNCS **1233** (1997), 37–51. (*Cited on page 293.*)

[9] R. Bright, **Smart Card Principles, Practice, Applications**, LS Howard Books, Chinchester (1988). (*Cited on page 291.*)

[10] J. Brillhart and J. Selfridge, *Some factorizations of* $2^n \pm 1$ *and related results*, Math. Comp. **21** (1967), 87–96. (*Cited on pages 193, 208.*)

[11] A.A. Bruen and M.A. Forcinito, **Cryptography, Information Theory, and Error-Correction**, Wiley (2005). (*Cited on page 119.*)

[12] J. Brunner, **Shockwave Rider**, Ballentine Books, New York (1975). (*Cited on page 283.*)

[13] K.W. Campbell and M.J. Weiner, *Proof that DES is not a group* in **Advances in Cryptology** — CRYPTO '92 Proc., Springer-Verlag, Berlin, LNCS **740** (1993), 518–526. (*Cited on page 150.*)

[14] R.D. Carmichael, *On the numerical factors of the arithmetic forms* $\alpha^n \pm \beta^n$, Ann. Math. **15** (1913–14), 30–70. (*Cited on page 191.*)

[15] A. Cauchy, *Mémoire sur diverses formules relatives à l'algèbre et à la théorie des nombres* (*suite*), C.R. Acad. Sci. Paris **12** (1841), 813–846. (*Cited on page 369.*)

[16] B. Chor and R.L. Rivest, *A knapsack type public key cryptosystem based on arithmetic in finite fields* in **Advances in Cryptology** — CRYPTO '84, Springer-Verlag, Berlin, LNCS **196** (1985), 54–65. (*Cited on page 340.*)

[17] B. Chor and R.L. Rivest, *A knapsack type public key cryptosystem based on arithmetic in finite fields* in IEEE Trans. Inform. Theory, **34** (1988), 901–909. (*Cited on page 340.*)

[18] D. Coppersmith, *The Data Encryption Standard* (*DES*) *and its strength against attacks*, IBM J. R. and D. **38** (1994), 243–250. (*Cited on pages 134–135.*)

[19] D. Coppersmith, H. Krawczyk, and Y. Mansour, *The shrinking generator* in **Advances in Cryptology** — CRYPTO '93, Springer-Verlag, Berlin, LNCS **773** (1994), 22–39. (*Cited on page 120.*)

[20] R. Crandall, K. Dilcher, and C. Pomerance, *A search for Wiefereich and Wilson primes.*, Math. Comp. **66** (1997), 433–449. (*Cited on page 33.*)

[21] J.A. Davies, D.B. Holdridge, and G.L. Simmons, *Status report on factoring* (*at Sandia National Labs*) in **Advances in Cryptology** — EUROCRYPT '84, Springer-Verlag, Berlin, LNCS **209**, (1985), 183–215. (*Cited on page 219.*)

[22] T. Dierks and C. Allen, *The TLS protocol, version 1.0, Internet Request for Comments* 2246 (January 1999). (*Cited on page 243.*)

[23] L.E. Dickson, **History of the Theory of Numbers**, Vol. **1**, Chelsea, New York, (1992). (*Cited on page 197.*)

[24] W. Diffie and M.E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), 644–654. (*Cited on pages 157, 160.*)

[25] W. Diffie and M.E. Hellman, *Exhaustive cryptanalysis of the NBS data encryption standard*, Computer, June (1977). (*Cited on page 149.*)

[26] T. ElGamal, *A public key cryptosystem and signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **31** (1985), 469–472. (*Cited on pages 181, 183.*)

[27] T. ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms* in **Advances in Cryptology** — CRYPTO '84, Springer-Verlag, Berlin, LNCS **196** (1985), 10–18. (*Cited on page 183.*)

[28] FIPS 46-3, *Data encryption standard* (DES) *defines and specifies the use of DES and triple DES*, November (1999). (*Cited on page 149.*)

[29] FIPS 180-1, *Secure Hash Standard*, April 17, 1995. (*Cited on page 346.*)

[30] FIPS PUB 180-2, *Secure Hash Standard* (SHS), August 26, 2002. (*Cited on page 346.*)

[31] FIPS PUB 185, *Escrowed Encryption Standard* (EES), February 9, 1994. (*Cited on page 246.*)

[32] FIPS 186, *Digital signature standard*, Federal Information Processing Standards Publication 186, U.S. Department of Commerce/N.I.S.T. National Tecnical Information Service, Springfield, VA (1994). (*Cited on page 246.*)

[33] FIPS 186-2, *Digital signature standard*, February (2002). (*Cited on page 187.*)

[34] M.R. Garey and D.S. Johnson, **Computers and Intractability**, Freeman, New York, Twenty-second printing (2000). (*Cited on page 168.*)

[35] C.F. Gauss, **Disquisitiones Arithmeticae** (English edition), Springer-Verlag, Berlin (1985). (*Cited on pages 42, 46, 61, 209.*)

[36] A.Géradin *F. Proth*, Sphinx-Oedipe, **7** (1912), 50–51. (*Cited on page 192.*)

[37] J. Gerver, *Factoring large numbers with a quadratic sieve*, Math. Comp. **41** (1983), 287–294. (*Cited on page 219.*)

[38] S.W. Golomb, **Shift Register Sequences**, Holden-Day, San Francisco (1967). Reprinted by Aegean Park Press (1982). (*Cited on pages 117, 121.*)

[39] J. Gordon, *Strong primes are easy to find*, in **Advances in Cryptology**, EUROCRYPT '84, Springer-Verlag, Berlin, LNCS **209** (1985), 216–223. (*Cited on page 204.*)

[40] R.K. Guy, **Unsolved Problems in Number Theory**, Vol. **1**, Second Edition, Springer-Verlag, Berlin (1994). (*Cited on pages 43, 48.*)

[41] H.M. Heys, *A tutorial on linear and differential cryptanalysis*, Technical Report CORR 2001-17, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada (2001). (*Cited on page 150.*)

[42] A. Hurwitz, *Question 801*, L'Intermédiaire Math. **3** (1896), 214. (*Cited on page 191.*)

[43] D. Kahn, **The Codebreakers**, Macmillan, New York (1967). (*Cited on page 80.*)

[44] J. Kilian and P. Rogaway, *How to protect DES against exhasutive key search*, in **Advances in Cryptology** — CRYPTO '96, Springer-Verlag, Berlin (1996), 252–267. (*Cited on page 150.*)

[45] G. Kipper, **Investigator's Guide to Steganography**, Auerbach, (A CRC Press Company), Boca Raton, London, New York, Washington, D.C., (2004). (*Cited on page 80.*)

[46] D.E. Knuth, **The Art of Computer Programming**, Volume **2**/ **Seminumerical Algorithms**, Third Edition, Addison-Wesley, Reading, Paris (1998). (*Cited on pages 76 and 114.*)

[47] M. Kraitchik, **Mathematical Recreations**, Dover, New York (1953). (*Cited on page 210.*)

[48] H. Krawczyk, *The order of encryption and authentication for protecting communications* (*or: How secure is SSL?)*, in **Advances in Cryptology** — CRYPTO 2001, Springer-Verlag, LNCS **2139** (2001), 310–331. (*Cited on pages 248, 272.*)

[49] R.S. Lehman, *Factoring large integers*, Math.Comp. **28** (1974), 637–646. (*Cited on page 208.*)

[50] S. Lehtinen, *SSH protocol assigned numbers*, INTERNET-DRAFT, draft-ietf-secsh-assignednumbers-05.txt, October (2003). (*Cited on page 271.*)

[51] D.H. Lehmer, **Selected Papers of D.H. Lehmer**, Volumes I–III, D. McCarthy (Ed.), The Charles Babbage Research Centre, St. Pierre, Canada (1981). (*Cited on page 57.*)

[52] D.H. Lehmer and R.E. Powers, *On factoring large numbers*, Bull. Amer. Math. Soc. **37** (1931), 770–776. (*Cited on page 209.*)

[53] A.K. Lenstra and M.S. Manasse, *Factoring by electronic mail* in **Advances in Cryptology** — EUROCRYPT '89, Springer-Verlag, Berlin, LNCS **434**, (1990), 355–371. (*Cited on page 219.*)

[54] H.W. Lenstra Jr., *On the Chor-Rivest knapsack cryptosystem*, Journal of Cryptology **3**, (1991), 149–155. (*Cited on page 343.*)

[55] S. Levy, **Crypto**, Penguin Books, New York (2001). (*Cited on pages 138, 158–159, 183.*)

[56] M. Matsui, *The first experimental cryptanalysis of the Data Encryption Standard* in **Advances in Cryptology** — CRYPTO '94, Springer-Verlag, Berlin, LNCS **839** (1994), 1–11. (*Cited on page 151.*)

[57] M. Matsui, *Linear cryptanalysis method for the DES cipher* in **Advances in Cryptology** — EUROCRYPT '93, Springer-Verlag, Berlin, LNCS **765** (1994), 386–397. (*Cited on page 150.*)

[58] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, **Handbook of Applied Cryptography**, CRC Press, Boca Raton, New York, London, Tokyo (1997). (*Cited on pages 75, 120, 176, 178.*)

[59] R.C. Merkle and M.E. Hellman, *Hiding information and signatures in trapdoor knapsacks*, IEEE Trans. Inform. Theory, **24** (1978), 525–530. (*Cited on page 337.*)

[60] R.C. Merkle and M.E. Hellman, *On the security of multiple encryption*, J. Communicatons of the ACM, **24** (1981), 465–467. (*Cited on page 148.*)

[61] R.A. Mollin, **Fundamental Number Theory with Applications**, CRC Press, Boca Raton, New York, London, Tokyo (1998). (*Cited on pages 18, 77, 301, 304, 307–307, 316, 321, 325, 328, 369.*)

[62] R.A. Mollin, **Algebraic Number Theory**, Chapman and Hall/CRC Press, Boca Raton, New York, London, Tokyo (1999). (*Cited on pages 18, 47, 63, 77, 191.*)

[63] R.A. Mollin, **RSA and Public-Key Cryptography**, Chapman and Hall/CRC Press, Boca Raton, FL (2003). (*Cited on pages 170, 160, 176, 198, 203–205.*)

[64] R.A. Mollin, **Codes — The Guide to Secrecy from Ancient to Modern Times**, Chapman and Hall/CRC (2005). (*Cited on pages 122,126, 128, 133, 155, 157, 166, 181, 204, 227, 229–230, 241, 250, 263, 346, 413.*)

[65] R. Moreno and P. Le Clech, *IPR and smart card patents — France*, (Innovatron) — Smart Card Eurpoe, London, December 12 (1995). (*Cited on page 291.*)

[66] M.A. Morrison and J. Brillhart, *A method of factoring and the factorization of F_7*, Math. Comp. **29** (1975), 183–205. (*Cited on page 213.*)

[67] S. Murphy, *The cryptanalysis of FEAL-4 with 20 chosen plaintexts*, J. Cryptol. **2** (1990), 50–61. (*Cited on page 145.*)

[68] P. van Oorschot, *A comparison of practical public-key cryptosystems based on integer factorization and discrete logarithms*, in **Contemporary Cryptography: The Science of Information Integrity**, G. Simmons, ed., IEEE Press, Piscatoway, NJ (1992), 289–322. (*Cited on page 165.*)

[69] W. Peterson and E.J. Weldon, **Error-Correction Codes**, M.I.T. Press, second edition (1972). (*Cited on page 117.*)

[70] J.M. Pollard, *An algorithm for testing the primality of any integer*, Bull. London Math. Soc. **3** (1971), 337–340. (*Cited on page 214.*)

[71] C. Pomerance, *The quadratic sieve factoring algorithm* in **Advances in Cryptology** — EUROCRYPT '84, Springer-Verlag, Berlin, LNCS **209**, (1985), 169–182. (*Cited on page 217.*)

[72] C. Pomerance, J. Selfridge, and S.S. Wagstaff Jr., *The pseudoprimes to* $2.5 \cdot 10^9$, Math. comp. **35** (1980) 1003–1026. (*Cited on page 200.*)

[73] A.J. van der Poorten, **Notes on Fermat's Last Theorem**, Wiley, New York, Toronto (1996). (*Cited on page 37.*)

[74] F. Proth, *Théorèmes sur les nombres premiers*, Comptes Rendus Acad. des Sciences, Paris, **87** (1878), 926. (*Cited on page 192.*)

[75] RFC 1928, *SOCKS protocol version 5*, March (1996). (*Cited on page 256.*)

[76] RFC 2109, *RFC 2109 — HTTP state management mechanism*, February (1997). (*Cited on page 261.*)

[77] RFC 2440, *OpenPGP Message Format*, November (1998). (*Cited on pages 240, 350–351.*)

[78] RFC 2459, *Internet X.509 public key infrastructure certificate and CRL profile*, January (1999). (*Cited on page 288.*)

[79] R.L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the A.C.M. **21** (1978), 120–126. (*Cited on page 160.*)

[80] P. Rogaway, *The security of DESX*, CryptoBytes **2** (Summer 1996). (*Cited on page 150.*)

[81] L. Rosenhead, *Henry Cabourn Pocklington*, Obituary Notices of the Royal Society, (1952), 555-565. (*Cited on page 191.*)

[82] E. Schaefer, *A simplified data encryption standard algorithm*, Cryptologia, January (1996). (*Cited on page 140.*)

[83] B. Schneier, **Applied Cryptography**, Wiley, New York, Toronto (1994). (*Cited on page 72.*)

[84] A. Shamir, *A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem* in **Advances in Cryptology** — CRYPTO '82 Proc., Plenum Press, New York (1983), 279–288. (*Cited on page 340.*)

[85] A. Shamir, *A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem*, IEEE Trans. Inform. Theory, **30** (1984), 699–704. (*Cited on page 340.*)

[86] C.E. Shannon, *Communication theory of secrecy systems*, Bell System Technical J., **28** (1949), 656–715. (*Cited on pages 112, 146–148.*)

[87] J.F. Shoch and J.A. Hupp, *The 'worms' programs: Early experience with a distributed computation*, J. Comm. ACM **25** (1982), 172–180. (*Cited on page 283.*)

[88] C. Suetonius Tranquillus, **The Lives of the Twelve Caesars**, Corner House, Williamstown, Mass. (1978). (*Cited on pages 82, 89–90.*)

[89] J.C.A. Van Der Lubbe, **Basic Methods of Cryptography**, Cambridge University Press (1998). (*Cited on page 121.*)

[90] S. Vaudenay, *Cryptanalysis of the Chor-Rivest cryptosystem*, J. Cryptol. **14** (2001), 87–100. (*Cited on page 340.*)

[91] S.S. Wagstaff, **Cryptanalysis of NumberTheoretic Ciphers**, Chapman and Hall/CRC (2003). (*Cited on pages 121, 188.*)

[92] E.W. Weisstein, **CRC Concise Encyclopedia of Mathematics**, CRC Press LLC, Boca Raton, London, Tokyo (1999). (*Cited on page 316.*)

[93] H.C. Williams, *On numbers analogous to Carmichael numbers*, Canad. Math. Bull. **20** (1977), 133–143. (*Cited on page 43.*)

[94] H.C. Williams, *Primality testing on a computer*, Ars Combin. **5** (1978), 127–185. (*Cited on page 200.*)

[95] T. Ylonen, *SH protocol architechture*, INTERNET-DRAFT, draft-ietf-secsh-architecture-15.txt, October (2003). (*Cited on page 270.*)

[96] T. Ylonen, *SSH transport layer protocol* INTERNET-DRAFT, draft-ietf-secsh-transport-17.txt, October (2003). (*Cited on page 270.*)

[97] T. Ylonen, *SSH connection protocol*, INTERNET-DRAFT, draft-ietf-secsh-agent-18.txt, October (2003). (*Cited on page 271.*)

[98] G. Yuval, *How to swindle Rabin*, Cryptologia **3** (1979), 187–190. (*Cited on page 130.*)

Figure 8.1: The author at the ruins of Phaistos in Crete, Greece, July, 2006.

Richard Anthony Mollin received his Bachelor's and Master's degrees from the University of Western Ontario in 1971 and 1972, respectively. His Ph.D. was obtained from Queen's University in 1975 in Kingston, Ontario, where he was born. Since then he has held various positions including Montreal's Concordia University, the University of Victoria, the University of Toronto, York University, McMaster University in Hamilton, the University of Lethbridge, and Queen's University in Kingston, where he was one of the first NSERC University Research Fellows. He is currently a full professor in the Mathematics Department of the University of Calgary, where he has been employed since 1982. He has over 180 publications, including 8 books, in algebra, number theory, and computational mathematics. He has been awarded 5 separate Killam awards over the past quarter century, including one in 2005, to complete his eighth book *Codes —The Guide to Secrecy from Ancient to Modern Times*, [64]. He is a member of the Mathematical Association of America, past member of both the Canadian and American Mathematical Societies, a member of various Editorial Boards, invited to lecture at numerous universities, conferences, and society meetings, as well as holding numerous research grants from universities and governmental agencies. Moreover, he is the founder of the Canadian Number Theory Association, and held its first conference in Banff in 1988, immediately preceding his NATO Advanced Study Institute.