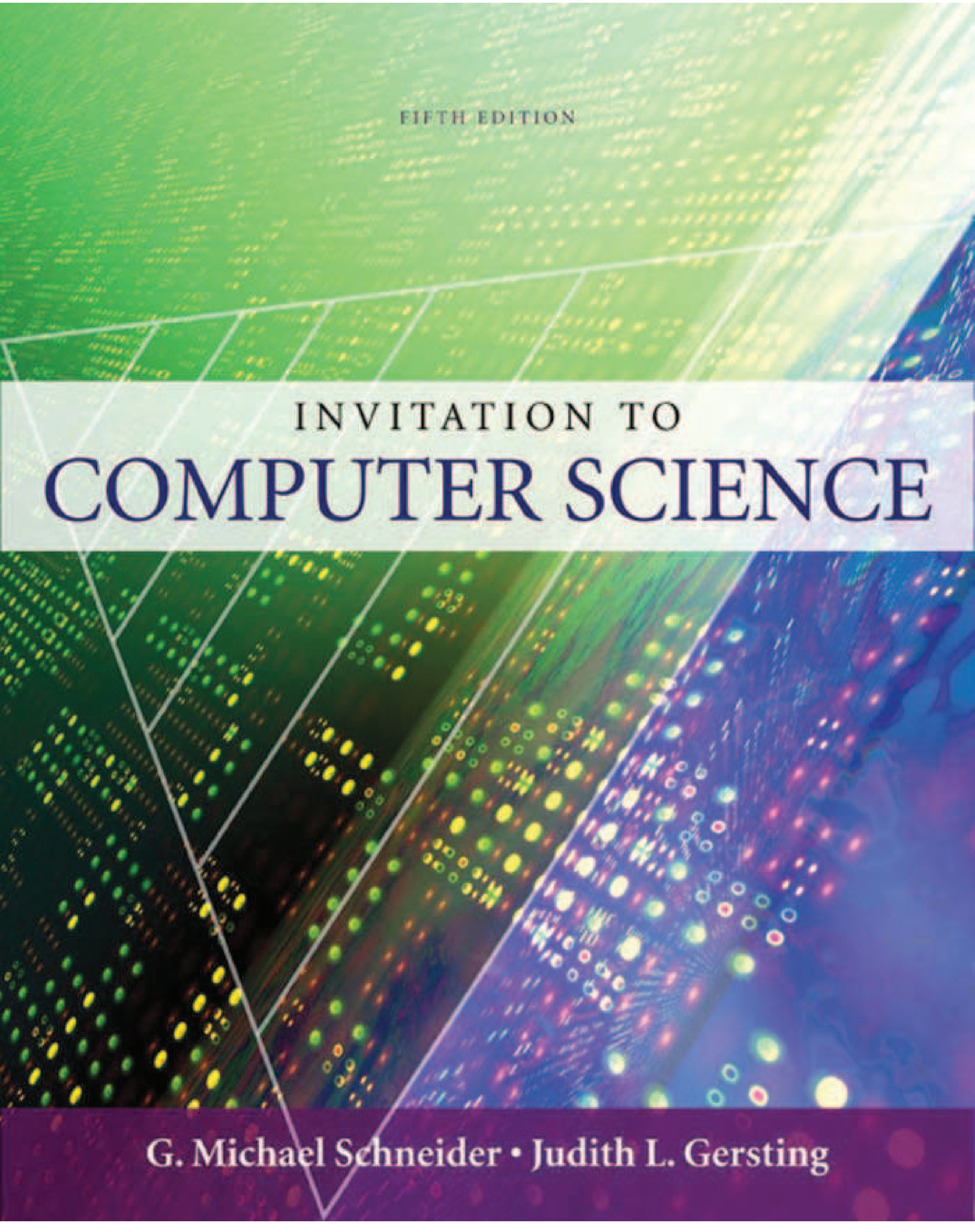


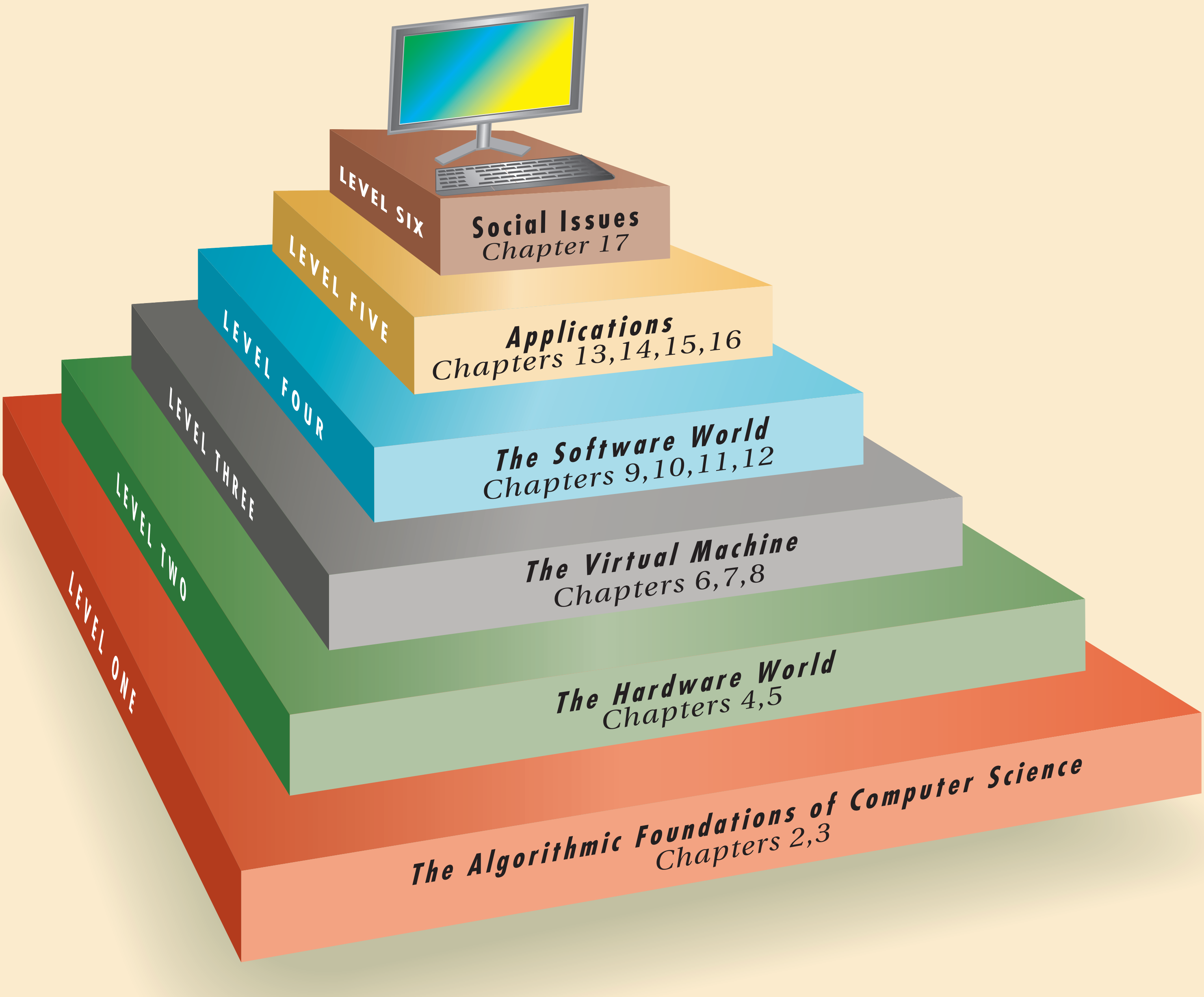
FIFTH EDITION

INVITATION TO COMPUTER SCIENCE

G. Michael Schneider • Judith L. Gersting

5TH EDITION

Invitation to Computer Science


LEVEL SIX
Social Issues
Chapter 17
LEVEL FIVE
Applications
Chapters 13,14,15,16
LEVEL FOUR
The Software World
Chapters 9,10,11,12
LEVEL THREE
The Virtual Machine
Chapters 6,7,8
LEVEL TWO
The Hardware World
Chapters 4,5
LEVEL ONE
The Algorithmic Foundations of Computer Science
Chapters 2,3

5TH EDITION

Invitation to Computer Science

G. Michael Schneider
Macalester College

Judith L. Gersting
University of Hawaii, Hilo

Contributing author:
Keith Miller
University of Illinois, Springfield

COURSE TECHNOLOGY
CENGAGE Learning™

Australia • Brazil • Japan • Korea • Mexico • Singapore • Spain • United Kingdom • United States

COURSE TECHNOLOGY
CENGAGE Learning™

Invitation to Computer Science, Fifth Edition
G. Michael Schneider and Judith L. Gersting

Executive Editor: Marie Lee

Acquisitions Editor: Amy Jollymore

Senior Product Manager: Alyssa Pratt

Development Editor: Deb Kaufmann

Editorial Assistant: Julia Leroux-Lindsey

Marketing Manager: Bryant Chrzan

Content Project Manager: Jennifer K. Feltri

Art Director: Faith Brosnan

Cover Designer: RHDG/Tim Heraldo

Cover Artwork: Fotolia.com (Royalty Free), Image # 375162

Compositor: Integra

ISBN-13: 978-0-324-78859-4
ISBN-10: 0-324-78859-2

Course Technology
20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil, and Japan. Locate your local office at: **international.cengage.com/region**

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

For your lifelong learning solutions, visit **course.cengage.com**
Visit our corporate website at **cengage.com**.

Some of the product names and company names used in this book have been used for identification purposes only and may be trademarks or registered trademarks of their respective manufacturers and sellers.

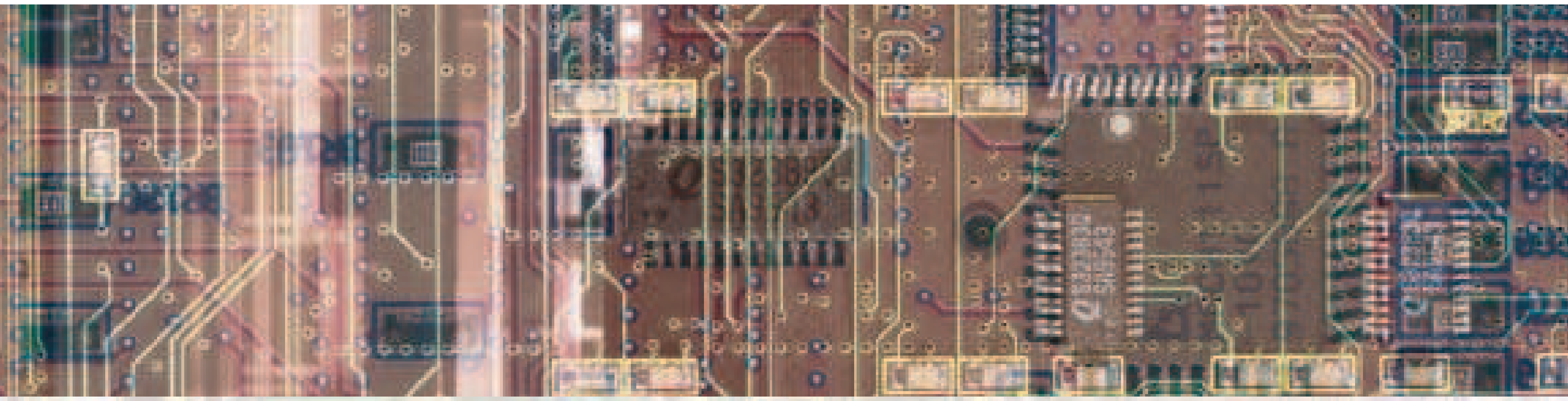
Any fictional data related to persons or companies or URLs used throughout this book is intended for instructional purposes only. At the time this book was printed, any such data was fictional and not belonging to any real persons or companies.

Course Technology, a part of Cengage Learning, reserves the right to revise this publication and make changes from time to time in its content without notice.

The programs in this book are for instructional purposes only.
They have been tested with care, but are not guaranteed for any particular intent beyond educational purposes. The author and the publisher do not offer any warranties or representations, nor do they accept any liabilities with respect to the programs.

Printed in Canada
1 2 3 4 5 6 7 13 12 11 10 09

BRIEF CONTENTS



CHAPTER 8

Information Security

8.1 Introduction

Information security means, of course, keeping information secure—that is, protected from those who should not have access to it. Information security could cover information locked in your filing cabinet, stuffed somewhere in your purse or wallet, or lying around on your desk. But today it usually means electronic information, data stored on your computer's hard disk or data being transmitted across a network.

In the early days of computing, when big mainframes were the only option, physical security was enforced by securing the rooms housing these machines. Only authorized persons had access. Now that there is a machine on virtually every desktop and a laptop in many briefcases, that kind of physical security is harder to obtain, but you can take some obvious steps: Don't leave your laptop lying around; never leave your workstation running when you are not in the room; do not share your password with anyone!

However, the danger of someone nearby swiping your laptop pales in comparison with the risks the Internet and the World Wide Web (discussed in the previous chapter) have brought. As our virtual environment expands, we celebrate the ability to reach out from our desktop, laptop, or handheld device to the rest of the world, receiving information, entertainment, and communications from thousands of sources. But we may be less aware of the potential for thousands of sources around the world to reach into our machines to do harm or steal information.

Security can be breached at many different points in the "virtual machine" we have presented in the last few chapters. Flaws in assembly-language programming can be exploited, operating system protections can be circumvented, and computer networks present all kinds of opportunities for viewing, manipulating, or destroying data. Consequences can range from annoyance to identity theft to major economic losses or threats to national security. Because there are so many ways in which security can be compromised, and the consequences can be so serious, information security is an important topic of both research and practice.

8.2 Threats and Defenses

You are no doubt aware of the possible threats to the security of your personal property, such as your car or the contents of your home or apartment. That's why you probably carry auto and home or renter's insurance. Aside from fire, flood, or other accidents, someone can steal your property, causing you financial harm and emotional distress. Everyone wants a secure environment. Your defenses against theft are to employ locks and possibly an alarm system. The alarm system only comes into play in an active manner if security has already been breached, but the announcement that a property is alarmed can be a deterrent to break-ins. While it's true that an experienced thief can quickly pick a lock, it's easier to break into an unlocked house. Be it thieves or computer hackers, either will attack the most vulnerable spots.

This section discusses the threat of individual computers being accessed by the wrong people, and also the threats to which a computer is exposed through network connections; in addition, it describes various defenses against these threats.

How Hackers became Crackers

Every new technology develops its own set of "undesirables"—those who see a new technology in terms not of potential benefits but of increased opportunities for misuse, just as automobiles brought wonderful benefits but also car thieves and drunk drivers. In computer science our abusive subculture goes by the name **hackers**.

Originally, the word *hacker* did not have a negative connotation. It was a mildly complimentary term for people who knew how to get things done on a computer—those somewhat strange and quirky individuals who seemed to know all the incomprehensible details about how computers worked, in essence, computer enthusiasts. They were the "tinkerers" and "fixers" who could enter some weird sequence of commands that miraculously cured whatever was wrong with your system.

As computers became more and more important to the functioning of society, and as computer networks increased the number of machines that could be accessed by individuals, the term *hacker* began to take on a different meaning. Some hackers turned their talents to figuring out how to override security measures to gain unauthorized access to other computers. Why? Perhaps for fun, or just because they could—the computer equivalent of joyriding in a stolen automobile. The results at first were relatively harmless. But soon these explorations turned to exploitations—ways to attack computer systems and cause destruction. Sometimes the word **cracker** is used to denote those who break into someone else's computer (like "cracking" a safe) as opposed to the more innocent "hacker" of the original use of the word. The general usage, however, is "hacker" for both types of intent. Computer code (scripts) or even downloadable programs for hacking can be found on the Internet. So it's possible to be an amateur hacker with little or no understanding of the technical aspects involved. "Professional" hackers view with disdain those **script kiddies** who resort to such tactics.

Be aware, though, that any type of hacking is illegal and punishable under the law by fines or imprisonment. Just as vandalism is not considered a harmless prank, the misuse of information technology is no longer viewed as the harmless intellectual play of "computer jockeys." It is seen for what it is—a serious crime with serious consequences and very severe penalties. We will examine the legal and ethical issues related to hacking in Chapter 17.

8.2.1 *Authentication and Authorization*

The first line of defense against illicit use of, or threats to, computer resources and sensitive information is a strong authentication and authorization process.

AUTHENTICATION. You want to start up your computer. You want to access your online e-mail account. You want to access your online bank account to transfer money or pay a bill. What's the first thing you have to do in all these cases? Generally it is to log on to your machine or to the appropriate Web page by giving your user ID and password. This authentication process on your computer is managed by your machine's operating system, as we learned in Chapter 6. On the Web page, it is managed by the operating system of the Web server computer. **Authentication** verifies who has the right to gain access to the computer, whether it is your local machine or the Web server. The operating system maintains a file of user IDs and corresponding passwords. When a user attempts to log on to the machine, the operating system reads the user ID and checks that the password matches the password for that user in the password file. Hackers breaking into a computer system look for a file of user IDs and passwords as the "Open, Sesame" for all locked doors.

If the user ID/password list were just in the form of

Tijara	popsicle
Murphy	badboy2
Jaylynn	mom
. . .	

then the entire system would be compromised if this password file were stolen (copied). Instead, the operating system encrypts the password for a given user by encoding the information, using a process (a hash function) that is easy to apply but hard to undo. The **hash function** takes the password the user originally chooses, chops it up, and stirs it around according to a given formula. As a very simple example, suppose the hash function process is the following:

1. Take each letter in the password and replace it with a number representing its place in the alphabet (a → 1, b → 2, etc.). Leave each digit in the password alone. In the above case, "badboy2" would become

   2 1 4 2 15 25 2
2. Add up these digits to get a single integer. In this example

   $2 + 1 + 4 + 2 + 15 + 25 + 2 = 51$
3. Divide the integer from Step 2 by 7 and find the remainder. In this example, dividing 51 by 7 gives a remainder of 2 (51 equals 7×7 with 2 left over).
4. Add 1 to the result from Step 3, then multiply the new number by 9. In this example, the result equals $(2 + 1) \times 9 = 27$.
5. Reverse the digits in the integer from Step 4 and then replace each digit with the corresponding letter of the alphabet. The result for this example is 72, which becomes gb.

The encrypted password file would contain an entry of

Murphy gb

and the original password is discarded.

Now when Tom Murphy attempts to log on, he gives "Murphy" as the user ID and enters a password. The operating system applies the hash function to that password and compares the encrypted result to the encrypted value "gb" stored in the password file for that user ID. If there is a match, Tom is recognized as an authorized user and allowed access to the computer.

You may have forgotten a password to some online account at, let's say, Ninth Street Bank, and asked for help. The people in charge of Ninth Street Bank's Web computer will e-mail you a temporary password that will allow you to log on and then require you to reset your password to something you choose (which will change your entry in the password file). You might find this annoying and wonder why they didn't just send you your original password. As we've just seen, the system at online Ninth Street Bank doesn't actually know your original password, only its encrypted form and—as we are about to see—this isn't enough information to regenerate the original password.

If you managed to steal the encrypted password file, you would not be able to recover the original password, even if you knew the steps of the algorithm. In our simple example, you could reverse Steps 5 and 4, but not Step 3. For example,

chjbup5

also hashes to gb, so clearly the process is not reversible. In fact, algorithms for hashing seem to be well known for each operating system, but, as we have seen here, knowledge of the hashing algorithm does not give you (or the system administrator) certain knowledge of the original password, even if you have the encrypted password file.

But this appears to raise another problem. What if Fred and Alice have two different passwords that hash to the same encrypted value? What if Fred and Alice chose the same original password? In general, this is not a problem—both Fred and Alice can log in using their respective passwords. But if Fred stole the password file and saw that his password and Alice's password hashed to the same value, he would have a better than random chance of guessing Alice's password; he would certainly try his own password with Alice's user ID, and he would be successful if indeed the passwords were the same.

To solve this problem, some operating systems keep a third entry for each user in the password file, namely the exact time at which the user created the password. This timestamp gets appended to the original password, and the result is then run through the encryption algorithm. That way, two identical passwords do not hash to the same value because the probability that they were created at the exact same instant in time is infinitesimally small. When someone attempting to log on gives his or her password, the operating system consults the password file, appends the timestamp for that user ID to the password just entered, encrypts the result, and compares it with the encrypted password entry for that user ID in the password file.

Nonetheless, there are ways in which the operating system's authentication process is vulnerable. Consider Ravi, who has not stolen the password file but nevertheless knows Alice's user ID and wants to hack into Alice's account. Since Ravi knows Alice personally, he might try to guess her password. He might try her nickname, the name of her pet poodle, the title of her favorite band. Of course he could also try "alice", "123456", or—a perennial favorite—"password". Many systems set "password" as the default value, and if Alice hasn't changed (or been required to change) her password, this will get Ravi into Alice's account. Failing at these attempts, Ravi might try a brute-force attack of trying all possible passwords. Suppose there are *n* possible characters that can be used in a password (at least uppercase and lowercase letters and the 10 digits are possibilities). To try all possible passwords of length *k* or less would require

$$n^1 + n^2 + \ . \ . \ . \ + n^k$$

attempts. On the average, Ravi might be successful after about

$$(n^1 + n^2 + \ . \ . \ . \ + n^k)/2$$

attempts. But this will be very time-consuming. In addition, most systems have a lock-out after some number of failed tries, which would foil this approach.

For someone who has stolen the password file, a better way to do a brute-force approach is by using password-cracking software. For a given user ID (which our villain knows because the user ID is not encoded), **password-cracking**

software will first try all words in its built-in dictionary, encrypting each with the well-known hash function and comparing the result with the password file. If this fails, it will then go on to a brute-force attack using all combinations of characters in turn. Such software is amazingly fast and can try a million or more potential passwords per second.

Surprisingly, the easiest way to obtain someone's password is not to steal the password file (hard to do) or to guess that person's password (time-consuming), but to use social engineering. **Social engineering** is the process of using people to get the information you want. If Ravi wants Alice's password, he might just ask her! In a business setting, he might get a chance to snoop around her office and find the yellow sticky note containing her password attached to her monitor or stuck beneath her keyboard. He might violate "password etiquette" and watch over her shoulder while she logs on. Or he could try an indirect approach; he could call Alice's (gullible) secretary and, posing as an IT technician, explain that Alice has called the IT service group about some computer problems she is experiencing and that to fix them, he needs Alice's password. Most companies try to educate their employees about the dangers of social engineering.

Your best defense against someone guessing your password is to be smart about how you choose and use your password.

Password Pointers

Choosing passwords

- Use long passwords (at least 8 characters)
- Use a mixture of uppercase and lowercase letters, digits, and special symbols (if allowed on your system)
- Consider using the first letters of some long phrase that is meaningful to you, mixed with some digits or special symbols
- Avoid personal information such as your name, user ID, or birthdate
- Avoid common dictionary words

Using passwords

- Change your password often (many systems require this)
- Use different passwords for different applications
- Don't tell anyone your password
- Don't write your password down
- Be wary when your browser offers to remember a password for you

Managing passwords may be a bit of a hassle, but security measures are always a balancing act between user convenience and user protection.

While user IDs and passwords are the most common authentication mechanism, there are other options. Some laptops now use **biometric information**, i.e., fingerprint scanning. Some company networks use a **one-time password** scheme that works as follows: The legitimate user enters his or her user ID and partial password. Each user has a small device that then generates the (random) last half of the password, which is good only for a few seconds. The system knows both the first half and last half and checks for a match after the user enters the last half. In this way, the password is quite secure because it is only valid for a very short time.

Don't forget the most basic physical security principles—maintain control of your laptop, be sure no one peers over your shoulder in your office or on the airplane, lock your office door when you leave, and so on. In a secure business or government environment, video surveillance cameras can enhance security, and visitors to a secure server site are checked against an access list and logged in to and out of the room.

PRACTICE PROBLEM

Using the hash function described in this section, verify that the password “chjbup5” also hashes to “gb”.

AUTHORIZATION. **Authorization** governs what an authenticated user is allowed to do. Enforcing authorization rules is also one of the jobs of the operating system, as discussed in Chapter 6. On your own machine, you may have administrative privileges and be able to see everything on the machine. On your banking Web site, you can only see your own account information. The operating system maintains access control lists. Depending on who users are, they have various privileges, such as

- Read access (can read a particular file)
- Write access (can modify a particular file)
- Execute access (can run a particular program file)
- Delete access (can delete a particular file)

As a student in a class, you have read/write/delete access to your own files. You have execute access and read access to programs and files the instructor has placed in a common folder. The instructor probably has access to the files of all students in the class. The **system administrator** or **superuser** has access to everything, and is the person who sets up the authorization privileges for all other users. A careful operating system will check every access every time by every user.

8.2.2 *Threats from the Network*

Once your personal computer, a business computer, or a Web server computer is connected to the Internet, there are many more possibilities for harm. Attacks can come from anonymous sources anywhere in the world via many intermediate nodes that maintain varying levels of security. Recall the complexity of the Internet, the fact that it is not specifically governed by any one entity, and the idea that the whole point is to share information, and it is clear what a difficult task “Internet security” can be.

Most of these threats come in the form of **malware** (malicious software) that can attack an individual computer. The most common attacks to individual computers are by viruses, worms, Trojan horses, and denial of service.

VIRUS. A **virus** is a computer program that, like a biological virus, infects a host computer and then spreads. It embeds itself within another program or file. When that program or file is activated, the virus copies itself and attacks other files on the system. The results may be as simple as taunting pop-up messages, but could also include erratic behavior or drastic slowdown of the computer, corrupted or deleted files, loss of data, or system crashes. The virus is spread from one machine to another by passing on the infected file, perhaps on a flash drive. By far the most common mechanism for spreading a virus, however, is through e-mail attachments. An infected file is attached to an e-mail message and sent out to 100 people, for example. Anyone who downloads and opens the attachment causes the virus to replicate the infected file and perhaps send it out as an e-mail attachment to the first 100 people in that person's personal address book. In this way a virus can potentially spread like wildfire across the Internet.

WORM. A **worm** is very similar to a virus, but it can send copies of itself to other nodes on a computer network without having to be carried by an infected host file. In its most benign form, a worm can clog the Internet so that legitimate traffic is slowed or shut out completely. In addition, the worm might also subvert the host systems it passes through so that, at a later time, those systems can be controlled by the worm's author and used to send spam e-mail, deface Web pages, or perform other mischief.

TROJAN HORSE. A **Trojan horse** (in the software world) is a computer program that does some harmless little job but also, unbeknownst to the user, contains code to perform the same kinds of malicious attacks as viruses and worms—corrupt or delete files, slow down the computer, and the like. It might also upload or download files, capture the user's address book to send out spam e-mail, hide a **keystroke logger** that captures the user's passwords and credit card numbers (and sends them to someone else), or even put the computer under someone else's remote control at some point in the future. A computer can become infected by a Trojan horse when the user downloads infected software from a malicious Web site. In fact even visiting an infected Web site can, behind the scenes, download a Trojan horse (an attack called a **drive-by exploit** or **drive-by download**).

DENIAL OF SERVICE. A denial-of-service (DOS) attack is typically directed at a business or government Web site. The attack automatically directs browsers on many machines to a single URL at roughly the same time, causing so much network traffic to that site that it is effectively shut down to legitimate users. Spam e-mail can accomplish a similar, but less targeted effect, by flooding the Internet with e-mail messages that consume available bandwidth and clog mail servers.

Beware the Trojan Horse

The "Trojan horse" name refers to a wooden horse in Greek mythology which the Greeks left as a gift for the Trojans during the Trojan War. When the Trojans pulled the horse into the city, the Greek soldiers hiding inside sneaked out under cover of nightfall, opened the gates of Troy to the Greek army, and Troy was defeated.

In the same way, Trojan horse software works by hiding destructive code within some seemingly legitimate program where it waits until it can sneak out and spring into action. A Trojan horse known as Coreflood or AF Core Trojan has been around since 2002 but has recently taken a more dangerous turn. Coreflood can infect the computing capability of an entire network. If a system administrator logs on to an infected machine, say for routine maintenance, the Trojan horse infects the system administrator machine and thereafter any machine the system administrator logs on to. The current variation of Coreflood captures banking or brokerage account user IDs, passwords, and balance information. In 2008, a security company was able to find a host computer where the Coreflood perpetrators maintained a database of over 50 GB of data they had stolen from over 378,000 computers during the previous 16 months, including 14,000 machines within a global hotel chain network. Widespread attacks also occurred at financial institutions, hospitals, and even a law enforcement agency. At the time of this writing, the authors of the sophisticated Coreflood attack have not been identified.

Defense Against the Dark Arts

You may feel at this point that you should just unplug your computer from the Internet or disable your wireless capability. The good news is you can protect yourself.

- Be sure your computer has up-to-date **anti-virus software** from a reputable company. Such software can detect worms, viruses, and Trojan horses by distinctive "signatures" those programs carry. It cleans your machine of infected files. Most anti-virus software comes with a feature for automatic updates; this is necessary because the "good guys" have to keep up with the new ideas from the "bad guys."
- Be sure your computer has an up-to-date **firewall** from a reputable company. Firewall software guards the access points to your computer, blocking communications to or from sites you don't permit.
- Be sure your computer has up-to-date **anti-spyware software** from a reputable company. Anti-spyware routinely scans your computer for any "spyware" programs that may have infected your machine—programs that capture information on what Web sites you have visited and what passwords and credit card numbers you have used.
- Be sure to install the latest security patches or updates to your operating system, whatever operating system you use. It's a widely held belief that home computers with Windows operating systems are most vulnerable to attack, but this may only reflect the market share Windows enjoys.
- Don't open e-mail attachments from unknown sources, especially files with .exe or .zip extensions.
- Don't download software except from reputable sources.
- Don't send personal or financial information in response to any e-mail ("My wealthy Nigerian uncle left me a fortune that I am willing to share with you but I need your account number . . .").

Gone Phishin'

Phishing is a practice used to illegally obtain sensitive information such as credit card numbers, account numbers, and passwords. An e-mail is sent claiming to be from a legitimate organization such as a bank, credit card company, or online retailer asking the recipient to click on a link to update or verify his or her account information. Often the message contains a warning that your account will be suspended or cancelled if you fail to provide this information (although this personal information is already on file with the legitimate organization if indeed you do have an account there). The link takes the unwary user to a fake Web site that appears to be legitimate, and captures the information the user enters. Despite the fact that no legitimate bank or other financial organization ever operates this way, many people fall for this scheme and become victims of identity theft, suffering financial losses that in total cost consumers hundreds of millions of dollars.

The Anti-Phishing Working Group (APWG) is an industry and law enforcement association focusing on helping eliminate identity theft resulting from phishing (*www.antiphishing.org/index.html*). It provides discussion forums, maintains statistics, and tests promising technology solutions. The APWG protects the identity of its member industries because such organizations, although guilty of no wrongdoing, are reluctant to admit that their sites were mimicked in phishing attacks. According to data from the APWG, there were over 47,000 phishing attacks in the first half of 2008 launched from over 26,000 unique domain names (a given domain name can host multiple attacks). The average phishing site is left online for less than two days, making it difficult to catch those responsible.

The term "phishing" comes about because perpetrators cast out bait, in the form of e-mail messages, to thousands of potential victims in the hope that one or two will "bite" and fall for this scam. Never follow the link in such a message or reply to the message; instead, delete the message immediately. If you want to check an account's status, open a separate browser window and access your account information as you normally would.

8.3 Encryption

Much of the thrust of information security, as we have talked about it so far, is to devise defenses so that the "bad guys" can't steal our information. If, despite these precautions, files on a computer hard disk or packets passing along a network connection are illegally accessed and fall into the wrong hands, we can still protect their contents through encryption (we make our information meaningless to the bad guys if they do get it). We've already discussed encryption of the password file by the operating system as a security measure, in case the password file is stolen. In this section we'll discuss various other encryption mechanisms, which apply both to data stored and data sent.

8.3.1 *Encryption Overview*

Cryptography is the science of "secret writing." A message (**plaintext**) is encoded (encrypted) before it is sent, for the purpose of keeping its content secret if it is intercepted by the wrong parties. The encrypted message is called **ciphertext**. The ciphertext is decoded (decrypted) back to plaintext when it is received, in order to retrieve the original information. **Encryption** and **decryption** date back thousands of years. The most famous instances of cryptography occur in military history, beginning with Julius Caesar of the Roman Empire, who developed the Caesar cipher. In more modern times, the military importance of cryptography was illustrated by the German Enigma code cracked by the Allies during World War II.

Encryption and decryption are inverse operations because decryption must "undo" the encryption and reproduce the original text. (An exception is hash function encoding, used for password encryption, which is a one-way code and does not involve decryption.) There are many encryption/decryption algorithms, and of course both the sender and receiver must use the same system. A **symmetric encryption algorithm** requires the use of a secret key known to both the sender and receiver. The sender encrypts the plaintext using the key. The receiver, knowing the key, is easily able to reverse the process and decrypt the message. One of the difficulties with a symmetric encryption algorithm is how to securely transmit the secret key so that both the sender and the receiver know what it is; in fact, this approach seems to simply move the security problem to a slightly different level, from transmitting a message to transmitting a key. In an **asymmetric encryption algorithm**, also called a **public key encryption algorithm**, the key for encryption and the key for decryption are quite different, although related. Person A can make an encryption key public, and anyone can encrypt a message using A's public key and send it to A. Only A has the decryption key, however, so only A can decrypt the message. This approach avoids the difficulty of secret key transmission, but it introduces a new problem: The relationship between the decryption key and the encryption key must be sufficiently complex that it is not possible to derive the decryption key from the public encryption key.

8.3.2 *Simple Encryption Algorithms*

CAESAR CIPHER. A **Caesar cipher**, also called a **shift cipher**, involves shifting each character in the message to another character some fixed distance farther along in the alphabet. Specifically, let *s* be some integer between 1 and 25 that represents the amount of shift. Each letter in the message is encoded as the letter that is s units farther along in the alphabet, with the last s letters of the alphabet shifted in a cycle to the first s letters. For example, if $s = 3$, then A is encoded as D, B is encoded as E, X is encoded as A, and Z is encoded as C. The integer s is the secret key. Decoding a message, given knowledge of s, simply means reversing the shift. For example, if $s = 3$, then the code word DUPB is decoded as ARMY.

The Caesar cipher is an example of a **stream cipher**; that is, it encodes one character at a time. This makes it easy to encode just by scanning the plaintext and doing the appropriate substitution at each character. On the other hand, there are only 25 possible keys, so a ciphertext message could be decoded by brute force, that is, by simply trying all possible keys.

In addition, the Caesar cipher is a **substitution cipher**, whereby a single letter of plaintext generates a single letter of ciphertext. We can replace the simple shift mechanism of the Caesar cipher with a more complex substitution mechanism, for example:

$$\begin{tikzcd}
\mathrm{A} \arrow[d, dash] & \mathrm{B} \arrow[d, dash] & \mathrm{C} \arrow[d, dash] & \mathrm{E} \arrow[d, dash] & \mathrm{F} \arrow[d, dash] \ldots \\
\mathrm{Z} & \mathrm{A} & \mathrm{Y} & \mathrm{B} & \mathrm{X} \ldots
\end{tikzcd}$$

(Can you guess the substitution algorithm being used?) However, in any simple substitution cipher, the structure of the plaintext is maintained in the ciphertext—letter frequency, occurrence of double letters, frequently occurring letter combinations, and so forth. With a sufficiently long message, an experienced **cryptanalyst** (code breaker) can use these clues to recover the plaintext.

BLOCK CIPHER. In a **block cipher**, a group or block of plaintext letters gets encoded into a block of ciphertext, but not by substituting one character at a time for each letter. Each plaintext character in the block contributes to more than one ciphertext character, and each ciphertext character is the result of more than one plaintext letter. It is as if each plaintext character in a block gets chopped into little pieces, and these pieces are scattered among the ciphertext characters in the corresponding block. This tends to destroy the structure of the plaintext and make decryption more difficult.

As a simple example, we'll use a block size of 2 and an encoding key that is a 2 × 2 arrangement of numbers called a **matrix**. Here A and B,

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 5 & 1 \\ 2 & 1 \end{bmatrix}$$

are matrices. We can define an operation of matrix multiplication as follows. The product $A \times B$ will also be a 2×2 matrix, where the element in row i, column j of $A \times B$ is obtained by multiplying each element in row i of A by its corresponding element in column j of B and adding the results. So to obtain the element in row 1, column 1 of the result, we multiply the row 1 elements of A by the corresponding column 1 elements of B and add the results:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 9 & \\ & \end{bmatrix}$$

$$1 * 5 + 2 * 2 = 5 + 4 = 9$$

To obtain the element in row 1, column 2 of the result, we multiply the row 1 elements of *A* by the corresponding column 2 elements of *B* and add the results:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 9 & 3 \\ & \end{bmatrix}$$

The completed product A × B is $\begin{bmatrix} 9 & 3 \\ 23 & 7 \end{bmatrix}$.

However, for encryption purposes, we are going to modify this definition. When we add up the terms for each element, whenever we exceed 25, we will start over again, counting from 0. In this scheme, 26 → 0, 27 → 1, 28 → 2, . . . , 52 → 0, and so on.

Not every 2 × 2 matrix can serve as an encryption key; we need an **invertible matrix**. This is a matrix *M* for which there is another matrix *M′* such that

$$M' \times M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

For example, $M = \begin{bmatrix} 3 & 5 \\ 2 & 3 \end{bmatrix}$ is invertible because for $M' = \begin{bmatrix} 23 & 5 \\ 2 & 23 \end{bmatrix}$,

$$M' \times M = \begin{bmatrix} 23 & 5 \\ 2 & 23 \end{bmatrix} \times \begin{bmatrix} 3 & 5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 79 & 130 \\ 52 & 79 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$ (remember that any value > 25 gets wrapped around to 0).[1]

This property is what allows *M′* to reverse the effect of *M*. Also, part of our encryption algorithm is a simple substitution *S* that maps letters into numbers; we'll let *S* be really simple here: $S(A) = 1, S(B) = 2, \ldots, S(Z) = 26$. Obviously *S* is reversible, and we'll call the reverse mapping S': $S'(1) = A$, $S'(2) = B, \ldots, S'(26) = Z$.

To encode our message, we break it up into two-character blocks. Suppose the first two characters form the block (*D E*). We apply the *S* mapping to this block to get (4 5). Now we multiply (4 5) × *M* by treating (4 5) as the row of some matrix (and remember to wrap around if the result exceeds 25):

$$(4 \quad 5) \times \begin{bmatrix} 3 & 5 \\ 2 & 3 \end{bmatrix} = (4 * 3 + 5 * 2 \quad 4 * 5 + 5 * 3)$$

$$= (22 \;\; 35) \rightarrow (22 \;\; 9)$$

Finally, apply the *S′* mapping to get from digits back to characters: *S′* (22 9) = (*V I*). This completes the encoding, and (*V I*) is the ciphertext for the message block (*D E*). Notice that the digit 4 (i.e., the plaintext letter *D*) contributed to both the 22 (*V*) and the 9 (*I*), as did the digit 5 (i.e., the plaintext letter *E*). This **diffusion** (scattering) of the plaintext within the ciphertext is the advantage of a block cipher.

[1] The mathematical function modulo 26 is being applied.

FIGURE 8.1

Steps in Encoding and Decoding for a Block Cipher

Encoding

1. Apply S mapping to plaintext block.
2. Multiply result times M, applying wraparound.
3. Apply S′ to the result.

Decoding

1. Apply S mapping to ciphertext block.
2. Multiply result times M′, applying wraparound.
3. Apply S′ to the result.

For decoding, we reverse the above steps. Starting with the ciphertext ($V\ I$), we first apply S to get (22 9). We then multiply (22 9) by M', the inverse of the encoding key (remembering to wrap around if the result exceeds 25):

$$\begin{aligned}(22 \quad 9) \times \begin{bmatrix} 23 & 5 \\ 2 & 23 \end{bmatrix} &= (22 * 23 + 9 * 2 \qquad 22 * 5 + 9 * 23) \\ &= (524 \;\; 317) \rightarrow (4 \;\; 5)\end{aligned}$$

Finally we apply S' to get back—voilà!—the plaintext ($D\ E$).

Figure 8.1 summarizes the steps. Again, the matrix M is the secret encryption key, from which the decryption key M' can be derived.

PRACTICE PROBLEMS

1. Using a Caesar cipher with $s = 5$, encrypt the message NOW IS THE HOUR.
2. A messenger tells you that the secret key for today for the Caesar cipher is $s = 26$. Should you trust the messenger? Why, or why not?

LABORATORY EXPERIENCE 12

The laboratory software for this laboratory experience uses a block cipher of block size 2. You will be able to encrypt and decrypt messages. The encryption key is again a matrix, but the encryption algorithm is quite different from that of the block cipher discussed in this section. In the illustration shown here, the first block of the plaintext is “th”. This has been encrypted, using the encryption matrix, into “qE”.

Hiding in Plain Sight

Steganography is the practice of hiding the very existence of a message. It's an old idea; an ancient Greek ruler was said to have sent a (not very urgent) message by tattooing the message on the shaved head of one of his slaves and then sending the slave off after his hair had grown back. The message was revealed on the other end by once more shaving the messenger's head.

These days, steganography has again come into favor in the form of hidden text within images posted on the Web. As we learned in Chapter 4, a colored digital image is composed of individual pixels; in the usual RGB format, 8 bits are allocated for each of the red, green, and blue color components. This allows for $2^8 = 256$ variations of intensity for each color. Let's say the red component in a pixel has the following 8-bit value:

11010010

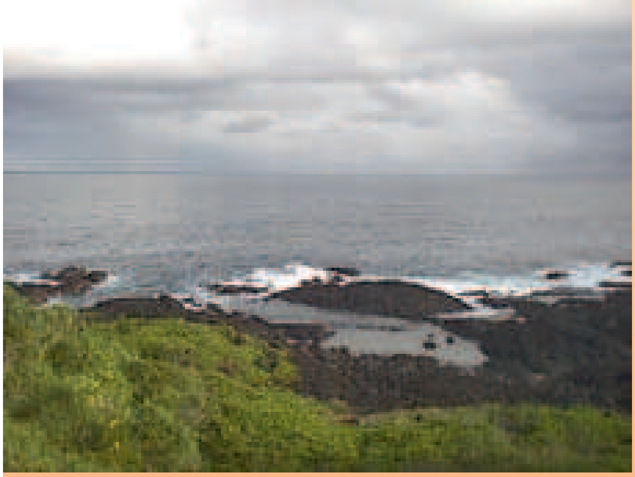
The least-significant bit (the right-most 0) contributes the least to the color intensity. If this bit is changed from 0 to 1, the red component becomes

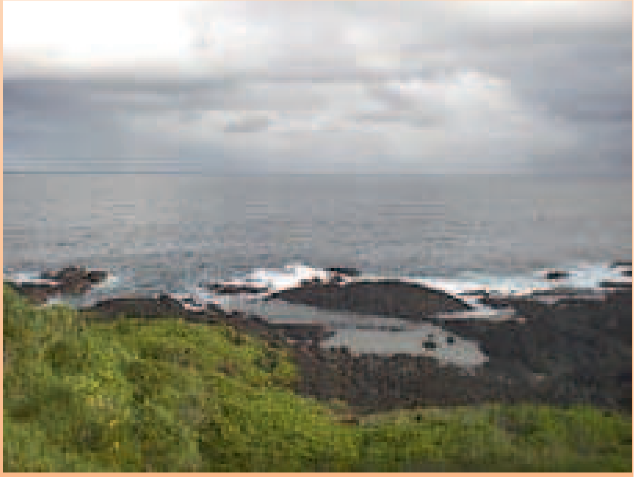
11010011

This tiny change will not be detectable to the human eye viewing the image.

A text file can be hidden in an image file by changing (if needed) the least significant bit of each byte of the image file to match the binary form of the characters in the text. For example, if the first letter of the text file to be hidden is "A", with ASCII code 01000001, then the first 8 bytes of the image file would be modified (if needed) so that the least significant bits are 0, 1, 0, 0, 0, 0, 0, and 1. To the naked eye, the image appears unaltered.

The image on the left below is the original. The image on the right uses steganography to hide 668 KB of text, over 140 double-spaced pages. Can you see any difference?





8.3.3 *DES*

Both of the previous encryption algorithms are too simplistic to provide much real security. **DES (Data Encryption Standard)** is an encryption algorithm developed by IBM in the 1970s for the U.S. National Bureau of Standards (now called the U.S. National Institute of Standards and Technology, or NIST), and is certified as an international standard by the International Organization for Standardization, or ISO (the same organization that certifies the MP3 digital audio format, as discussed in Chapter 4). One might expect this internationally standard algorithm to rest upon some extremely complex and obscure operations, but the DES algorithm actually uses very simple operations—however, it uses them over and over.

DES was designed to protect electronic information, so the plaintext is a binary string of 0s and 1s, just as it is stored in a computer. As we learned in Chapter 4, this means that ordinary text has already undergone an encoding using ASCII or Unicode to convert characters to bit strings. This encoding, however, is not for the purposes of secrecy, and has nothing to do with the cryptographic encoding we are talking about in this chapter.

DES is a block cipher, and the blocks are 64 bits long, meaning that 64 plaintext bits at a time are processed into 64 ciphertext bits. The key is a 64-bit binary key, although only 56 bits are actually used. The algorithm begins by sending the plaintext 64-bit string through an initial **permutation** (rearrangement). The algorithm then cycles through 16 "rounds." Each round *i* performs the following steps:

1. The incoming 64-bit block is split into a left half L_i and a right half R_i. The right half R_i gets passed through unchanged to become the left half of the next round, L_{i+1}.
2. In addition, the 32 bits in the right half get permuted according to a fixed formula and then expanded to 48 bits by duplicating some of the bits. Meanwhile, the 56-bit key is also permuted (the result is passed on as the key to the next round) and then reduced to 48 bits by omitting some of the bits. These two 48-bit strings are matched bit by bit, using an XOR (exclusive OR) gate for each bit. Figure 8.2 shows the standard symbol for an XOR gate, along with its truth table.
3. The resulting 48-bit string undergoes a substitution and reduction to emerge as a 32-bit string. This string is permuted one more time, and the resulting 32-bit string is matched bit by bit, using XOR gates, with the left half L_i of the input. The result is passed to the next round as the new right half R_{i+1}.

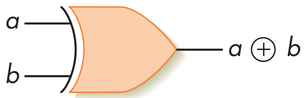
After all 16 rounds are complete, the final left and right halves are recombined into a 64-bit string that is permuted one more time, and the resulting 64-bit string is the ciphertext. Figure 8.3 outlines the steps involved in the DES algorithm.

Two important points about the DES algorithm: The first is that every substitution, reduction, expansion, and permutation is determined by a well-known set of tables. So, given the same plaintext and the same key, everyone using DES ends up with the same ciphertext. The "secret" part is the initial key. The second point is that the same algorithm serves as the decryption algorithm—just start with the ciphertext and apply the sequence of keys in reverse order, that is, the round-16 key first and the original secret key last.

With increased computing power in the hands of those trying to break a code, a 56-bit key does not seem as formidable as when DES was first introduced.

FIGURE 8.2

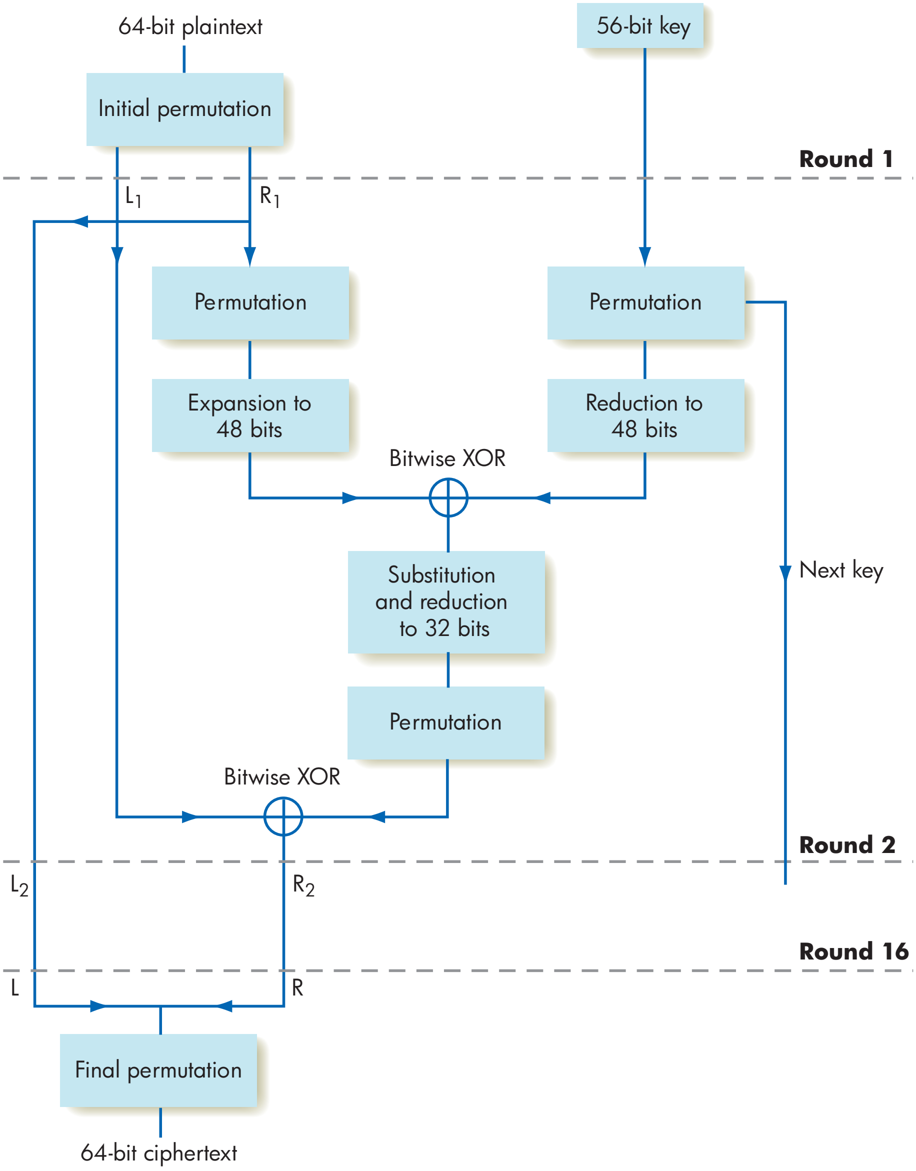
The XOR Gate



a	*b*	*a* ⊕ *b*
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 8.3

The DES Encryption Algorithm



It might even be feasible to try all 2^{56} (72,057,594,037,927,936) possible keys. **Triple DES** improves the security of DES; it requires two 56-bit keys (which can be thought of as a 112-bit key length), and runs the DES algorithm three times: Encode using key 1, decode the result using key 2, encode the result using key 1 again.

Concerns about the eventual breakdown of DES in the face of ever-increasing computing power prompted NIST in 1997 to request proposals for a successor encryption scheme. The result was **AES (Advanced Encryption Standard)**, which was adopted for use by the U.S. government in 2001. AES is based on the Rijndael (pronounced Rin-dahl) algorithm, named for the two Belgian cryptographers who designed it, Vincent Rijmen and Joan Daemen. Like DES, AES uses successive rounds of computations that mix up the data and the key. The key length can be 128, 192, or even 256 bits, and the algorithm appears to be very efficient.

Cracking DES

In 1997, the Electronic Frontier Foundation, a nonprofit civil-liberties organization, began to build the DES Cracker, a PC connected to a large array of custom chips. The entire configuration cost less than $250,000 to build (a very reasonable price at the time for big computing power). This machine was intended to apply brute-force techniques (trying all possible 56-bit keys) to crack ciphertext encoded using DES. In 1998, the machine was used to respond to a challenge in the form of a ciphertext message posed by RSA Laboratories, the research component of RSA Security, a leading electronic security company. The DES Cracker could test 88 billion keys per second, and it found the correct 56-bit key in less than three days.

This result had political and economic overtones because the U.S. government at the time had strict controls on the export of cryptographic software, most of which was limited to encryption algorithms using a 40-bit key or less. This hampered overseas sales of software products with strong encryption. The government also pressured industry within the United States to limit the use of cryptography to DES, claiming that DES codes were highly secure and nearly impossible to crack, a claim clearly refuted by this challenge. The designer of the DES Cracker machine noted that searching for a 40-bit key (the export limit at the time) using the DES Cracker would take 3–12 seconds.

Some people suspected that the government wanted to keep weak encoding in use in order to be able to access information, perhaps infringing on personal privacy. The U.S. export policy was made less restrictive in 1998, although not as a result of the DES Cracker. In Chapter 17, we'll examine the ethical issues raised in a specific instance of government regulation of encryption.

8.3.4 *Public Key Systems*

The encryption algorithms we have discussed so far have all been symmetric encryption algorithms, requiring that both the sender and receiver have knowledge of the key. Our final algorithm is an asymmetric, or public key, encryption algorithm. Remember that the main difficulty with a symmetric algorithm is how to securely transmit the secret key. In a public key system, the encryption key for messages to go to a particular receiver is broadcast to everyone, but the decryption key cannot be derived from it and is known only by the receiver.

The most common public key encryption algorithm is **RSA**, named for its developers, Ron Rivest, Adi Shamir, and Len Adleman at MIT (founders of RSA Security—see the box "Cracking DES"). This algorithm, developed in 1977, is based on results from the field of mathematics known as **number theory**.

A **prime number** is an integer greater than 1 that can only be written as the product of itself and 1. For example, 2, 3, 5, 7, 11, . . . are prime numbers; you can only write 7, for example, as $7 = 1 \times 7$, the product of 1 and 7. The numbers 4, 6, 8, 10, and 12, for example, are not prime because they can be factored in a nontrivial way:

$$4 = 2 \times 2 \qquad 6 = 2 \times 3 \qquad 8 = 2 \times 2 \times 2$$
$$10 = 2 \times 5 \qquad 12 = 2 \times 2 \times 3$$

Any positive integer is either a prime number or a number that can be written in a unique way as a product of prime factors. For example, $12 = 2 \times 2 \times 3$ is the product of three prime factors. The success of RSA encryption depends on the fact that it is extremely difficult to find the prime factors for n if n is a large number. So although information encrypted using RSA is technically not secure, it is secure in practice because of the large amount of computation necessary to find the prime factors of the encoding key.

Here's how RSA works. Two large prime numbers p and q are chosen at – random, and their product $n = p \times q$ is computed. The product $m = (p - 1) \times (q - 1)$ is also computed. Next, a large random number e is chosen in such a way that e and m have no common factors other than 1. This step guarantees the existence of a unique integer d between 0 and m, such that when we compute $e \times d$ using the same sort of wraparound arithmetic we used in the block encoding scheme—that is, whenever we reach m, we start over again counting from 0—the result is 1. There are computationally efficient ways to produce p, q, e, and d.

For example, suppose we pick $p = 3$ and $q = 7$ (a trivially small case). Then,

1. $n = p \times q = 3 \times 7 = 21$
2. $m = (p - 1) \times (q - 1) = 2 \times 6 = 12$
3. Choose $e = 5$ ($e = 5$ and $m = 12$ have no common factors)
4. Then $d = 5$ because $e \times d = 5 \times 5 = 25 = 2 \times 12 + 1$, so when we compute $e \times d$ using wraparound arithmetic with respect to 12, we get 1.

Now the number pair (n, e) becomes the public encryption key, and d is the decryption key. Let's suppose that the plaintext message has been converted into an integer P, using some sort of mapping from characters to numbers. The encryption process is to compute P^e using wraparound arithmetic with respect to n (when you reach n, make that 0). Continuing with our example, suppose $P = 3$. Then the ciphertext is computed as

5. $3^5 = 243 = 11 \times 21 + 12 \rightarrow 12$

(Note that the sender uses both parts of the public key, e and n, to compute the ciphertext.) The receiver decodes the ciphertext C by computing C^d using wraparound arithmetic with respect to n. In our example,

6. $12^5 = 248832 = 11849 \times 21 + 3 \rightarrow 3$

Of course, our example has a major problem in that d is the same as e. Obviously, in a real case, you want e and d to be different. The whole point is that even though n and e are known, the attacker must determine d, which involves finding the prime factors p and q of n. There is no known computationally efficient algorithm for this task.

PRACTICE PROBLEM

1. You receive a message "17" that was sent using the RSA public encryption key (21, 5) of the example in this section. Decode this to find the original numeric message. (You might want to use a spreadsheet to help with this computation.)

8.4 Web Transmission Security

One area where the public is very security-conscious is in making online purchases, which require the purchaser to send out across the network his or her name, address, and—most worrisome of all—credit card number. One method for achieving secure transfer of information on the Web is **SSL (Secure Sockets Layer)**. This is a series of protocols developed by Netscape

Communications (Netscape was an early Web browser) in the mid-1990s. The **TLS (Transport Layer Security)** protocol, first defined in 1999, is based on SSL and is nearly identical to SSL. TLS has a few technical security improvements over SSL, but the major difference is that TSL is nonproprietary and is a standard supported by the Internet Engineering Task Force. The IETF is an open organization of individuals concerned with "the evolution of the Internet architecture and the smooth operation of the Internet."[2]

Both TLS and SSL protocols are in use and are supported by all Web browsers. Technically, TSL/SSL fits between the transport layer, with its TCP protocols, and the application layer, with its HTTP protocols (both discussed in the previous chapter). When you see a closed lock icon at the top or bottom of your Web browser page, or when the URL displayed begins with HTTPS (instead of HTTP), then you can be assured that the communication between your browser and the Web server (the vendor's Web computer) is secure and protected by TLS or SSL. TLS/SSL allows a client (the purchaser's Web browser) and a Web server to agree on the encryption methods to be used, exchange the necessary security keys, and authenticate the identity of each party to the other. (Here we are again with *encryption* and *authentication*, the two pillars of security we've seen before.)

Now that we know a bit more about encryption, we might ask what encryption algorithm TLS/SSL uses; is it DES encryption or the newer, stronger RSA encryption? Surprisingly, it is both. One of the problems with the RSA algorithm is the computational overload for encryption/decryption. What often happens is that RSA is used in the initial stage of communication between client and server. For example, in response to a client request, the server may send the client its public key along with an authentication certificate. This is a certificate issued by a trusted third-party certificate authority; it's like a letter of introduction that attests that the server belongs to the organization the browser thinks it is talking to.

The client, using RSA and the public key of the server, encodes a short message containing the keys for a symmetric encryption algorithm. Because only keys are being encrypted, the message is short and the encryption can be done quickly with little RSA overload. The server receives and decodes this message and responds to the client with a message encoded using the symmetric key. The client and server have now established a secure exchange of keys (one of the issues with a symmetric encryption algorithm) and can complete the transaction using, for example, DES. Figure 8.4 illustrates the major steps in this process, although the technical details may require a few additional transmissions between the client and the server. The exchange of setup information between the client and server, preparatory to exchanging real data, is known as a **handshake**.

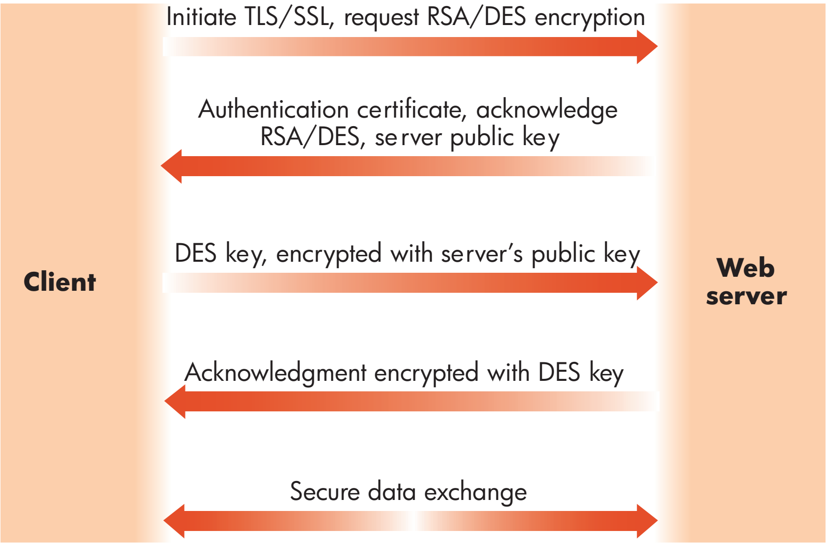
8.5 Conclusion

In this chapter we've looked at components of information security, both on an isolated local computer and a machine exposed to the network. Whether it's an individual's personal data, corporate information, or sensitive government data, all are under threat. Still, a bit of caution and common sense can go a long way. As the well-worn watchword says, "Security: It's Everybody's Business."

[2] http://www.ietf.org/overview.html

FIGURE 8.4

A TLS/SSL Session


Initiate TLS/SSL, request RSA/DES encryption
Authentication certificate, acknowledge RSA/DES, server public key
DES key, encrypted with server's public key
Acknowledgment encrypted with DES key
Secure data exchange
Client
Web server

8.6 Summary of Level 3

We have seen that the hardware described in Chapters 4 and 5 is, by itself, nearly unusable. Working directly with the hardware components of a von Neumann machine—processors, memory, ALU—is impossible for any but the most technically knowledgeable users. To make the system accessible, the system software must create a people-oriented *virtual machine* or *virtual environment* that is easy to understand and use. In addition to ease of use, this virtual environment provides a number of other services, including resource management, security, access control, and efficient resource use. A great deal of work has been done to try to identify and create an optimal virtual environment.

Operating systems—a critical component of the virtual environment—have evolved from early batch systems through multiprogramming and time-sharing to the current network and real-time operating systems. Most modern operating systems allow us to use a large collection of machines, called a computer network, almost as easily as if it were a single logical system. Network technology has expanded our virtual environment to encompass a worldwide grid of interconnected computers. More and more, the computer user on a networked system can deal only with *what* operations need to be done, not with where or how they can be done. The future of computer systems definitely lies in the direction of more abstract and more powerful virtual environments.

As our virtual environment has expanded, our most important asset—our data—is increasingly at risk for theft or destruction by clever outsiders with malicious intent. Constant vigilance is required to maintain information security so that our virtual environment is not only user-friendly but also safe.

Now that we have created a vastly more usable environment in which to work, what do we want to do with it? Well, we probably want to write programs that solve important problems. In the next level of the text, we begin our study of the software world.

EXERCISES

1. Below are three possible logon scenarios. Explain why answer (c) below is preferable in terms of system security.

   a.
   ```
   Welcome to XYZ computing
   Enter user name: jones
   Invalid user name
   Enter user name:
   ```
   b.
   ```
   Welcome to XYZ computing
   Enter user name: smith
   Enter password: password
   Invalid access
   Enter user name:
   ```
   c.
   ```
   Enter user name: smith
   Enter password: password
   Invalid access
   Enter user name: smith
   Enter password: FpQr56
   Welcome to XYZ computing
   ```

2. Using the hash function described in Section 8.2.1, find the encrypted forms of the following passwords:

   a. fido

   b. blank

   c. ti34pper

3. *Merriam-Webster's Collegiate Dictionary*, 11th ed. (Merriam-Webster, Inc., 2003), contains over 225,000 entries. Using a password-cracking tool that can process 1.7 million words per second, how long would it take to test each word in the dictionary as a possible password?

4. A virus attacks a single user's computer and within one hour embeds itself in 50 e-mail attachment files sent out to other users. By the end of the hour, 10% of these have been opened and have infected their host machines. If this process continues, how many machines will be infected at the end of 5 hours? Can you find a formula for the number of machines infected after *n* hours?

5. Using a Caesar cipher with $s = 5$, decode the received message RTAJ TZY FY IFBS.

6. The centurion who was supposed to inform you of *s* was killed en route, but you have received the message MXX SMGX UE PUHUPQP in a Caesar cipher. Find the value of *s* and decode the message.

7. You receive a message that was encoded using a block encoding scheme with the encoding matrix

   $$M = \begin{bmatrix} 3 & 2 \\ 7 & 5 \end{bmatrix}.$$

   a. Verify by computing $M' \times M$ that $M' = \begin{bmatrix} 5 & 24 \\ 19 & 3 \end{bmatrix}$.

   (Remember to wrap around if a value is > 25.)

   b. Decode the ciphertext message MXOSHI.

8. The DES algorithm combines two bit strings by applying the XOR operator on each pair of corresponding bits. Compute the 6-bit string that results from $100111 \oplus 110101$.

9. Using the RSA encryption algorithm, pick $p = 11$ and $q = 7$. Find a set of encryption/decryption keys *e* and *d*.

10. Using the RSA encryption algorithm, let $p = 3$ and $q = 5$. Then $n = 15$ and $m = 8$. Let $e = 11$.

    a. Compute *d*.

    b. Find the code for 3.

    c. Decode your answer to part (b) to retrieve the 3.

11. If a message is encrypted using AES with a key length of 256 bits, the brute-force approach to decryption involves generating each of the 2^{256} possible keys in turn until one is found that decodes the encrypted message. Quantum computing was discussed in Chapter 5. Using a quantum computer, how many qubits are required to represent all 2^{256} possible keys simultaneously?

CHALLENGE WORK

1. Find information about a well-known computer virus or worm. Answer as many of the following questions as you can, and be sure to list your sources.
   a. What is the name of the virus or worm?
   b. When did it first appear?
   c. Who wrote it?
   d. How was it spread?
   e. What systems did it attack?
   f. What were its observable effects?
   g. What are the technical details on how it worked?
   h. What was the "cure"?
   i. Did it spawn "copycat" attacks?
   j. What was its economic impact?
   k. Was the perpetrator found, arrested, convicted?
2. The Vigenère cipher was first proposed in the sixteenth century. At its heart is the Vigenère table (shown below), where each row represents a Caesar cipher of the letters of the alphabet shown in the row of column headers. The shift for row A is $s = 0$, for row B it is $s = 1$, for row C it is $s = 2$, etc. Thus in row C, the column header A is shifted 2 characters to become C, B becomes D, and so forth. The key to the Vigenère cipher is a secret word or phrase known only to the sender and receiver. Each letter in the key is used to encode a letter in the plaintext by finding in the table the row of the key letter and the column of the plaintext letter; their intersection is the ciphertext letter for that plaintext letter. When every letter in the key has been used, the key is repeated.

   For example, suppose the key is

   ```
   SONG
   ```

   and the plaintext message is

   ```
   MEETATNOON
   ```

   Because the key is shorter than the plaintext, it will have to be used several times:

   ```
   SONGSONGSO
   MEETATNOON
   ```

   The first character of the ciphertext is found at the intersection of row S and column M; it is E. The second character of the ciphertext is found at the intersection of row O and column E; it is S. The complete ciphertext is

   ```
   ESRZSHAUGB
   ```

   To decode a received message, you reverse this process. Again matching the key characters to the ciphertext characters,

   ```
   SONGSONGSO
   ESRZSHAUGB
   ```

   find the ciphertext character in the key character's row; the plaintext character is the corresponding column heading. Thus in row S, the E occurs in column M, so M is the corresponding plaintext.

   You receive the following ciphertext message that you know was encoded using the Vigenère cipher with a secret key of PEANUTS:

   ```
   DREVZUQAENQ
   ```

   Decode the ciphertext to find the plaintext.

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

FOR FURTHER READING

Many of the books on information security are quite technical in nature and therefore rather difficult to read, but here is a smattering of more accessible material.

The following books give surveys of information security from the perspective of the business manager or policy maker.

Schou, C., and Shoemaker, D. *Information Assurance for the Enterprise: A Roadmap to Information Security*. New York: McGraw-Hill, 2007.

Whitman, M., and Mattord, H. *Principles of Information Security*, 3rd ed. Boston, MA: Course Technology, 2007.

This book covers various forms of attack, as well as encryption and assessing a system for security.

Easttom, C. *Computer Security Fundamentals*. Englewood Cliffs, NJ: Prentice-Hall, 2006.

The next book is a series of papers and essays on the ethical and policy issues raised by Internet hacking.

Himma, Kenneth. *Internet Security: Hacking, Counterhacking, and Society*. Sudbury, MA: Jones and Bartlett, 2007.