INTRODUCTION TO
NETWORK SECURITY
THEORY AND PRACTICE

JIE WANG • ZACHARY KISSEL

Higher Education Press

WILEY

INTRODUCTION TO NETWORK SECURITY

INTRODUCTION TO NETWORK SECURITY
THEORY AND PRACTICE

Jie Wang

University of Massachusetts Lowell, US

Zachary A. Kissel

Merrimack College, US

WILEY

Contents

Preface

People today are increasingly relying on public computer networks to conduct business and take care of household needs. However, public networks may be insecure because data stored in networked computers or transmitted through networks can be stolen, modified, or fabricated by malicious users. Thus, it is important to know what security measures are available and how to use them. Network security practices are designed to prevent these potential problems. Originating from meeting the needs of providing data confidentiality over public networks, network security has grown into a major academic discipline in both computer science and computer engineering, and also an important sector in the information industry.

The goal of network security is to give people the liberty of enjoying computer networks without the fear of compromising their rights and interests. Network security accomplishes this goal by providing confidentiality, integrity, nonrepudiation, and availability of useful data that are transmitted in open networks or stored in networked computers.

Network security will remain an active research area for several reasons. Firstly, security measures that are effective today may no longer be effective tomorrow because of advancements and breakthroughs in computing theory, algorithms, and computer technologies. Secondly, after the known security problems are solved, other security loopholes that were previously unknown may at some point be discovered and exploited by attackers. Thirdly, when new applications are developed or new technologies are invented, new security problems may also be created with them. Thus, network security is meant to be a long-lasting scuffle between the offenders and the defenders.

Research and development in network security has mainly followed two lines. One line studies computer cryptography and uses it to devise security protocols. The other line examines loopholes and side effects of the existing network protocols, software, and system configurations. It develops firewalls, intrusion detection systems, anti-malicious-software software, and other countermeasures. Interweaving these two lines together provides the basic building blocks for constructing deep layered defense systems against network security attacks.

This book is intended to provide a balanced treatment of network security along these two lines, with adequate materials and sufficient depth for teaching a one-semester introductory course on network security for graduate and upper-level undergraduate students. It is intended to inspire students to think about network security and prepare them for taking advanced network security courses. This book may also be used as a reference for IT professionals.

This book is a revision and extension of an early textbook written by the first author under the title of "Computer Network Security: Theory and Practice," which was co-published in 2008 by the Higher Education Press and Springer. The book is structured into 10 chapters.

Chapter 1 presents an overview of network security. It discusses network security goals, describes common network attacks, characterizes attackers, and defines a basic network security model.

Chapter 2 presents standard symmetric-key encryption algorithms, including DES, AES, and RC4. It discusses their strength and weaknesses. It also describes common block-cipher modes of operations and a recent block-cipher offset-codebook mode of operations. Finally, it presents key generation algorithms.

Chapter 3 presents standard public-key encryption algorithms and key-exchange algorithms, including Diffie–Hellman key exchange, RSA public-key cryptosystem, and elliptic-curve cryptography. It also discusses how to transmit and manage keys.

Chapter 4 presents secure hash functions and message authentication code algorithms for the purpose of authenticating data, including SHA-512, Whirlpool, SHA-3, cryptographic checksums, and the standard hash message authentication codes. It then discusses birthday attacks on secure hash functions and describes the digital signature standard. It presents a dual signature scheme used for electronic transactions and a blind signature scheme used for producing electronic cash. It concludes with a description of the Bitcoin protocol.

Chapter 5 presents several network security protocols commonly used in practice. It first describes a standard public-key infrastructure for managing public-key certificates. It then presents IPsec, a network-layer security protocol; SSL/TLS, a transport-layer security protocol; and several application-layer security protocols, including PGP and S/MIME for sending secure email messages, Kerberos for authenticating users in local area networks, and SSH for protecting remote logins.

Chapter 6 presents common security protocols for wireless local area networks at the data-link layer, including WEP for providing wired-equivalent privacy, WPA and IEEE 802.11i/WPA2 for providing wireless protected access, and IEEE 802.1X for authenticating wireless users. It then presents the Bluetooth security protocol and the ZigBee security protocol for wireless personal-area networks. Finally, it discusses security issues in wireless mesh networks.

Chapter 7 presents the key security issues involved in the burgeoning area of cloud computing, including a discussion of the multitenancy problem and issues of access control. It then presents advanced topics of searchable encryption for cryptographic cloud storage.

Chapter 8 presents firewall technologies and basic structures, including network-layer packet filtering, transport-layer stateful inspections, transport-layer gateways, application-layer proxies, trusted systems and bastion hosts, screened subnets, and network address translations.

Chapter 9 presents intrusion detection technologies, including intrusion detection system architecture and common intrusion detection methods. It also discusses event signatures, statistical analysis, and data mining methods. Finally, it introduces honeypot technologies.

Chapter 10 describes malicious software, such as viruses, worms, and Trojan horses, and introduces countermeasures. It also covers Web security and discusses mechanisms against denial of service attacks.

Since the publication of the first edition, a number of readers have kindly shared with us their personal experiences in dealing with network security attacks. Some of their stories, after minor editing, are included in the text and the exercise problems.

To get the most out of this book, readers are assumed to have taken undergraduate courses on discrete mathematics, algorithms, data communications, and network programming, or

have equivalent preparations. For convenience, Chapter 3 includes a section reviewing basic concepts and results of number theory used in public-key cryptography. While it does not introduce socket programming, the book contains socket API client–server programming exercises. These exercises are designed for computer science and computer engineering students. Readers who do not wish to do them or simply do not have time to write code may skip them. Doing so would not affect much the learning of materials presented in the book.

Exercise problems for each chapter are divided into discussion problems and homework problems. There are six discussion problems in each chapter, designed to hep stimulate readers to think about the materials presented in that chapter at the conceptual level. These problems are intended to be discussed in class, with the instructor being the moderator. The homework problems are designed to have three levels of difficulty: regular, difficult (designated with *), and challenging (designated with **). This book contains a number of hands-on drills, presented as exercise problems. Readers are encouraged to try them all.

This book is intended to provide a concise and balanced treatment of network security with sufficient depth suitable for teaching a one-semester introductory course on network security. It was written on the basis of what the first author learned and experienced during the last 18 years from teaching these courses and on student feedback accumulated over the years. Powerpoint slides of these lectures can be found at `http://www.cs.uml.edu/~wang/NetSec`. Due to space limitations, some interesting topics and materials are not presented in this book. After all, one book can only accomplish one book's mission. We only hope that this book can achieve its objective. Of course, only you, the reader, can be the judge of it. We will be grateful if you will please offer your comments, suggestions, and corrections to us at wang@cs.uml.edu or kisselz@merrimack.edu.

We have benefited a great deal from numerous discussions over the last 20 years with our academic advisors, colleagues, teaching assistants, as well as current and former students. We are grateful to Sarah Agha, Stephen Bachelder, Yiqi Bai, William Baker, Samip Banker, David Bestor, Robert Betts, Ann Brady, Stephen Brinton, Jeff Brown, William Brown, Matthew Byrne, Robert Carbone, Jason Chan, Guanling Chen, Mark Conway, Michael Court, Andrew Cross, Daniel DaSilva, Paul Downing, Matthew Drozdz, Chunyan Du, Paul Duvall, Adam Elbirt, Zheng Fang, Daniel Finch, Jami Foran, Xinwen Fu, Anthony Gendreau, Weibo Gong, Edgar Goroza, Swati Gupta, Peter Hakewessell, Liwu Hao, Steve Homer, Qiang Hou, Marlon House, Bei Huang, Jared Karro, Christopher Kraft, Fanyu Kong, Lingfa Kong, Zaki Jaber, Ming Jia, Kimberly Johnson, Ken Kleiner, Minghui (Mark) Li, You (Stephanie) Li, Joseph Litman, Benyuan Liu, Yan (Jenny) Liu, Wenjing Lou, Jie Lu, Shan (Ivory) Lu, David Martin, Randy Matos, Laura Mattson, Thomas McCollem, Caterina Mullen, Paul Nelson, Dane Netherton, Michael Niedbala, Gerald Normandin, Kelly O'Donnell, Sunday Ogundijo, Xian Pan, Alexander Pennace, Sandeep Sahu, Subramanian Sathappan, John Savage, Kris Schlatter, Patrick Schrader, Susan Schueller, Liqun (Catherine) Shao, Blake Skinner, Chunyao Song, Adnan Suljevic, Hengky Susanto, Anthony Tiebout, David Thompson, Nathaniel Tuck, John Uhaneh, John Waller, Tao Wang, Brian Werner, Brian Willner, Christopher Woodard, Fang Wu, Jianhui Xie, Jie (Jane) Yang, Zhijun Yu, and Ning Zhong for their comments and feedbacks.

During the writing of the first edition, Jared Karro read the entire draft, Stephen Brinton read Chapters 1–5 and 7–8 (cloud security not included), Guanling Chen read Chapter 6, and Wenjing Lou read Chapters 2 and 6. Their comments have helped improve the quality of the

first edition in many ways, and to them we owe our gratitude. We are grateful to Anthony Gendreau and Adnan Suljevic for pointing out typos in the first edition.

We thank the reviewers for interesting suggestions and Ying Liu at the Higher Education Press for initiating this book project and editing the first edition of the book.

Jie Wang

Zachary A. Kissel

About the Author

Dr. Jie Wang is Professor and Chair of Computer Science at the University of Massachusetts Lowell. He is also Director of the University Center for Internet Security and Forensics Education and Research. He received a Ph.D. degree in Computer Science from Boston University in 1990, an M.S. degree in Computer Science from Zhongshan University in 1985, and a B.S. degree in Computational Mathematics from Zhongshan University in 1982. He has over 23 years of teaching and research experience at the university level and has worked as a network security consultant in the financial industry. He represented the University of Massachusetts system in the education task force of the Advanced Cyber Security Center in New England from 2011 to 2013. His research interests include network security, big data modeling and applications, algorithms and computational optimization, computational complexity theory, and wireless sensor networks. His research has been funded continuously by the National Science Foundation since 1991. His research has also been funded by IBM, Intel, and the Natural Science Foundation of China. He has published over 160 journal and conference papers, six books and four edited books. He is active in professional service, including chairing conference program committees and organizing workshops, editing journals and serving as the editor-in-chief of a book series on mathematical modeling.

Dr. Zachary A. Kissel is Assistant Professor of Computer Science at Merrimack College in North Andover, MA. He received a Ph.D. degree in Computer Science from the University of Massachusetts Lowell in 2013, an M.S. degree in Computer Science from Northeastern University in 2007, and a B.S. degree in Computer Science from Merrimack College in 2005. He has network security industry experience working in a security group at Sun Mircrosystems (later Oracle) where he was responsible for maintaining firewalls and cryptographic libraries. His research interests include cryptography and network security. His work has focused mainly on searchable symmetric encryption and access control for data stored on an untrusted cloud.

1

Network Security Overview

If you know your enemies and know yourself, you will win hundred times in hundred battles. If you know yourself but not your enemies, you will suffer a defeat for every victory won. If you do not know yourself or your enemies, you will always lose.

—Sun Tzu, "The Art of War"

The goal of network security is to give people the freedom to enjoy computer networks without the fear of compromising their rights and interests. Network security therefore needs to guard networked computer systems and protect electronic data that is either stored in networked computers or transmitted in the networks. The Internet, which is built on the IP communication protocols, has become the dominant computer network technology. It interconnects millions of computers and edge networks into one immense network system. The Internet is a public network, where individuals or organizations can easily become subscribers of the Internet service by connecting their own computers and networking devices (e.g., routers and sniffers) to the Internet and paying a small subscription fee.

Because IP is a store-forward switching technology, where data is transmitted using routers controlled by other people, user A can read user B's data that goes through user A's network equipment. Likewise, user A's data transmitted in the Internet may also be read by user B. Hence, any individual or any organization may become an attacker, a target, or both. Even if one does not want to attack other people, it is still possible that one's networked computers may be compromised into becoming an attacking tool. Therefore, to achieve the goal of network security, one must first understand the attackers, what could become their targets, and how these targets might be attacked.

1.1   Mission and Definitions

The tasks of network security are to provide *confidentiality*, *integrity*, *nonrepudiation*, and *availability* of useful *data* that are transmitted in public networks or stored in networked computers.

The concept of data has a broad sense in the context of network security. Any object that can be processed or executed by computers is data. Thus, source code, executable code, files in various formats, email messages, digital music, digital graphics, and digital video are each considered data. Data should be read, written, or modified only by legitimate users. That is, unauthorized individuals or organizations are not allowed to have access to data.

Just as CPU, RAM, hard disk, and network bandwidth are resources, data is also a resource. Data is sometimes referred to as *information* or *messages*.

Each piece of data has two possible states, namely, the *transmission state* and the *storage state*. Data in the transmission state is simply data in the process of being delivered to a network destination. Data in the storage state is that which is stored in a local computer or in a storage device. Thus, the meanings of data confidentiality and data integrity have the following two aspects:

1. Provide and maintain the confidentiality and integrity of data that is in the transmission state. In this sense, confidentiality means that data during transmission cannot be read by any unauthorized user, and integrity means that data during transmission cannot be modified or fabricated by any unauthorized user.
2. Provide and maintain the confidentiality and integrity of data that is in the storage state. Within this state, confidentiality means that data stored in a local device cannot be read by any unauthorized user through a network, and integrity means that data stored in a local device cannot be modified or fabricated by any unauthorized user through a network.

Data nonrepudiation means that a person who owns the data has no way to convince other people that he or she does not own it.

Data availability means that attackers cannot block legitimate users from using available resources and services of a networked computer. For example, a computer system infected with a virus should be able to detect and disinfect the virus without much delay, and a server hit by denial of service attacks should still be able to provide services to its users.

Unintentional components in protocol specifications, protocol implementations, or other types of software that are exploitable by attackers are often referred to as *loopholes*, *flaws*, or *defects*. They might be an imperfect minor step in a protocol design, an unforeseen side effect of a certain instruction in a program, or a misconfigured setting in a system.

Defense is the guiding principle of network security, but it is a passive defense because before being attacked, the victim has no idea who the attackers are and from which computers in the jungle of the Internet the attackers will launch their attacks. After a victim is attacked, even if the attacker's identity and computer system are known, the victim still cannot launch a direct assault at the attacker, for such actions may be unlawful. What constitutes legal actions against attackers involves a discussion of relevant laws, which is beyond the scope of this book. Therefore, although offense may be the best defense in military operations, this tactic may not apply to network security. Building a deep layered defense system is instead the best possible defense tactic in network security. Within this type of defense system, multiple layers of defense mechanisms are used to resist possible attacks.

Network security is a major part of information security. In addition to network security, information security deals with many other security issues, including security policies, security auditing, security assessment, trusted operating systems, database security, secure

code, emergency response, computer forensics, software forensics, disaster recovery, and security training.

- Security policies are special rules to protect a computer network system against security attacks. For example, security policies may specify what types of data are to be protected, who should be given the access right of read from or write to the data, and how the data should flow from one place to the next.
- Security auditing is a procedure of checking how well the security policies for a particular computer network system are followed. It may be a manual procedure or an automated procedure run by software tools.
- Security assessment is a procedure of determining the security needs of a particular system, measuring the strength and weakness of the existing security policies, and assessing whether the security policies are reasonable and whether security loopholes exist.
- A trusted operating system is an operating system without any security flaws or loopholes in system designs, computing resource management, software implementations, and configurations.
- Database security is a set of security measures specifically devised for database systems, specifying which data fields are accessible by which level of users.
- Secure software is software that contains no security flaws, loopholes, or side effects.
- Intrusion response is a set of actions that should take place when a computer network system is detected being intruded by intruders.
- Cyber forensics studies how to collect information of user activities from computer systems and network communications, providing evidence to indict cyber criminals. Cyber forensics can be further divided into computer forensics and network forensics.
- Disaster recovery is a set of mechanisms to bring a computer system that goes down because of attacks or natural disasters back to a working status.

This book does not cover these issues, but it may touch certain aspects of them.

1.2   Common Attacks and Defense Mechanisms

Common network security attacks can be characterized into a few basic types. Almost every known network security attack is either one of these basic types or a combination of several basic types.

1.2.1   Eavesdropping

Eavesdropping is an old and effective method for stealing private information. In network communications, the eavesdroppers may intercept data from network traffic using a networking device and a packet sniffer. A packet sniffer, or network sniffer, is a program for monitoring incoming network traffic. When connecting a router to the Internet, for example, one can use a packet sniffer to capture all the IP packets going through that router. `TCPdump` and `Wireshark` (formerly known as `Ethereal`) are network sniffers widely used today, which are available as free downloads (see Exercise 1.5).

Using a packet sniffer as an eavesdropping tool, one can intercept IP packets that go through the router he controls. To capture a particular IP packet, however, the eavesdropper must first determine which communication path the IP packet will travel through. Then, he could either try to get control of a certain router on the path or try to insert a new router of his own on the path. This task is more difficult but is not impossible. For example, the eavesdropper may try to compromise a router on the path and install a packet sniffer in it to intercept the IP packets he is after. The eavesdropper may also use an ARP spoofing technique (see Section 1.2.4) to reroute IP packets to his sniffer without compromising a router.

Eavesdropping wireless communications is easier. In this case, the attacker simply needs to place a receiver with the same radio frequency of the wireless network within the communication range of the network.

There is no way to stop eavesdropping in public networks. To counter eavesdropping, the best defense mechanism is to encrypt data. Computer cryptography is developed for this purpose, where the sender encrypts data into an unintelligible form before he transmits it. Data encryption is a major component of computer cryptography. It uses an encryption key in concert with an encryption algorithm, to break the original data into pieces and mix them up in a certain way to make it unintelligible, so that the eavesdropper cannot obtain any useful information out of it. Thus, even if the eavesdropper is able to intercept the encrypted data, he is still not able to obtain the original data without knowing the decryption key. We often refer the original data as *plaintext* data, or simply plaintext, and encrypted data as *ciphertext* data, or simply ciphertext.

Ciphertext data can be converted back to plaintext data using a decryption key along with a decryption algorithm. The encryption key is a string of characters, which is also referred to as *secret key*. In a symmetric-key encryption algorithm, also referred to as conventional encryption, the encryption key and the decryption key are identical. In a public-key encryption algorithm, also known as asymmetric-key encryption, the encryption key and the decryption key are different.

1.2.2   Cryptanalysis

Cryptanalysis is the art and science of finding useful information from ciphertext data without knowing the decryption keys. For example, in a substitution cipher that substitutes plaintext letters with ciphertext letters, if a ciphertext message reveals a certain statistical structure, then one may be able to decipher it. To obtain a statistical structure of the data, one may calculate the frequency of each character in the ciphertext data and compare it against the known statistical frequency of each character in the language used in the plain text. For example, in the English language, the letter "e" has the highest frequency. Thus, in a substitution cipher, the character that has the highest frequency in the ciphertext data is likely to correspond to the plaintext letter "e" (see e.g., Exercise 1.7). This analysis can be further extended to common phrases. Analyzing statistical structures of ciphertext messages was an effective method to break encryptions before the computer era.

Modern encryption algorithms can produce ciphertext without any trace of statistical structure. Therefore, modern cryptanalysis is focused on analyzing encryption algorithms using mathematical techniques and high-performance computers.

The best method against cryptanalysis is to devise encryption algorithms that reveal no statistical structures in ciphertext messages using sophisticated mathematics and longer

encryption keys. Using sophisticated mathematics makes mathematical analysis difficult. Using longer keys makes brute force attacks impractical. In addition to having stronger encryption algorithms, it is equally important to distribute and manage keys safely and to implement encryption algorithms without exploitable loopholes.

1.2.3 Password Pilfering

Computer users need to prove to the system that they are legitimate users. The most widely used authentication mechanism is in the form of user names and user passwords. User names are public information, but user passwords must be kept secret. Only two parties should have knowledge of the password, namely, the user and the underlying computer program (e.g., an operating system or a specific software application). A password is a sequence of letters, digits, or other characters, which is often selected by the user. Legitimate users enter their user names and passwords to prove their legitimacy to the computer program. An unauthorized user may impersonate a legitimate user to "legitimately" log on to a password-protected system or application, if he can get hold of a legitimate user name and password pair. He can then gain all the "legal" rights to transmit, receive, modify, and fabricate data.

Password protection is often the first defense line, and sometimes, it may be the only defense mechanism available in the system. Thus, we must take measures to ensure that user passwords are well protected against larcenies. For this purpose, we will look at several common methods for pilfering user passwords. These methods include *guessing*, *social engineering*, *dictionary attacks*, *side-channel attacks*, and *password sniffing*. *Phishing* attacks and *pharming* attacks have become the most common form of mass social engineering attacks in recent years.

1.2.3.1 Guessing

Guessing is the simplest method to acquire a password illegitimately. The attacker may get lucky if users use short passwords or if they forget to change the default passwords created for them. Also, users have a tendency to use the same passwords.

According to data compiled yearly by SplashData, a password management company, the top 10 most common passwords used by users, listed in decreasing order of popularity, are as follows:

 1. `123456`
 2. `password`
 3. `12345678`
 4. `qwerty`
 5. `abc123`
 6. `123456789`
 7. `111111`
 8. `1234567`
 9. `iloveyou`
10. `adobe123`

If the user chooses a simple password such as these 10 easy ones, then the guesser would indeed have an easy task.

1.2.3.2   Social Engineering

Social engineering is a method of using social skills to pilfer secret information from the victims. For example, attackers may try to impersonate people with authority or organizations of reputation to trick unvigilant users to reveal their user names and user passwords to the attackers. Impersonation may be carried out either in person or in an electronic form. Phishing and pharming are common electronic forms of social engineering attacks in recent years, targeted at a large number of people.

There are other forms of social engineering attacks. For example, attackers may try to collect recycled papers from the recycle bins in a corporation's office building, hoping to find useful login information. Attackers may also make a Web browser pop up a window asking for user login information.

Physical Impersonation

Physical impersonation means that the attacker pretends to be a different person to delude the victim. For example, the following imaginary conversion between the attacker and a receptionist named Betty demonstrates how a social engineering attack might be carried out in person:

> Attacker: (Speaking with an authoritative voice.) "Hello, Betty, this is Nina Hatcher. I am Marketing Manager of the China branch office."

> Betty: (Thinking that this woman knew my name, my number, and spoke like a manager, she must be whom she said she was.) "Hello, Nina, what can I do for you?"

> Attacker: "Betty, I am attending a meeting in Guangzhou to finalize an important deal with a large corporation in China. To close the deal, I'll need to verify certain technical data produced by your group that I believe is still stored in the computer at your site. This is urgent. I tried to log on to your system today, but for some reason it didn't work. I was able to log on to it yesterday though. Is your computer down? Can you help me out here?"

> Betty: "Well, I don't know what happened. But you may try the following $\cdots$" (Thinking that she is doing the company a favor by telling the marketing manager how to get into the system.)

Phishing

Phishing attacks are mass social engineering attacks that take advantage of people with a tendency to trust authorities. The main forms of phishing attacks are disguised email messages or masqueraded Websites. For example, attackers (also called *phishers*) send disguised email messages to people as if these messages were from banks, credit card companies, or other financial institutions that people may pay attention to. People who receive such messages are told that there was a security breach in their accounts, and so they are required to verify their account information for security purposes. They are then directed to a masqueraded Website to enter their user names and passwords (e.g., see Exercise 1.15). The following example is a real phishing message verbatim (The reader may notice a number of grammatical errors and format problems.):

From: UML NEW EMAIL `<helpdesk@uml.edu>`

To:

Date: Wed, Jul 7, 2010 at 2:28 AM

Subject: Re UNIVERSITY I.T.S UPDATE

Welcome to the university of Massachusetts Lowell New webmail system.

Many of you have given us suggestions about how to make the Umass Lowell webmail better and we have listened. This is our continuing effort to provide you with the best email services and prevent the rate of spam messages received in your inbox folder daily. Consequently all in-active old email accounts will be deleted during the upgrade.

To prevent your account from deletion and or being suspended we recommends all email accounts owner users to upgrade to the new email. Fill in your data in the blank space provided;

```
(Email:_____),  (User I.D_____), (password_____)
(Retype password_____).
```

The University I.T.S

`www.uml.edu`

Checked by AVG - Version: 8.5.437 Virus Database: 271.1.12840 - Release

This was a blunt phishing attack, in which the phisher simply asked the recipients to fill in the blanks with their passwords. Other more sophisticated phishing emails may contain a bogus Website as a trap to capture account information entered by the victims. Here, the email and the Website are the baits. The sniffing mechanisms hiding behind the Web page are the hook. Most phishing emails, no matter how well they are put together, would often contain the lines of "Something happened with your account, and you need to go to this page to fix it, or your account will be deleted". In general, any phishing email would contain a link to a bogus Website, called a *phishing site*. Phishing sites may look like the real ones, with the purpose of luring careless users to enter useful login information only to be captured by the phisher.

Even if you do not plan to enter any information on the bogus Website, clicking the link in the phishing email may already compromise your computer, for modern phishing techniques make it possible to embed exploits in a Web page, and the exploits will be activated when you open the Web page.

Users may look at the following three things to detect abnormalities: (1) the "From" address, which may look odd; (2) the URL links the phishers want them to click on, which may be similar to but definitely different from the real site (e.g., a URL that looks like Citicard is in reality not the Citibank's real site); and (3) the look and feel of the Website if the user fails to identify any abnormality during the first two items, for the bogus Website would not be exactly the same as the real site. For example, the color scheme may look different. If you receive an email from a bank or a credit card company telling you that your have a problem with your account and asking you for your user name and password, then most likely it is a phishing email, for banks or credit card companies would never send emails to their customers asking for their account information.

Sometimes, a phishing email may contain a line similar to this: "To be removed from this list click here." Do not click on this link, for it will notify the attacker that the user did read the email and consequently more annoying emails may come.

Antiphishing extensions of Web browsers are emerging technology for detecting and blocking phishing sites. *Email scanners* may also be used to identify phishing emails. However, blocking phishing and not blocking legitimate emails is challenging, even with appropriate email scanners. Thus, users may also want to develop their own tools to detect compromised email accounts and disable them before they can send out phishing emails.

1.2.3.3   Pharming

Pharming attacks use Web technologies to redirect users from the URLs they want to visit to a URL specified by the attacker, including changing DNS setting or the hosts file on the victim's computer, where DNS stands for domain-name service. Attacks that change DNS settings are also referred to as DNS poisoning. If an DNS-poisoning attack is launched from an insecure home router or wireless access point, it is also referred to as a drive-by pharming. Reported by Symantec in 2008, the first drive-by pharming attack was targeted at a Mexican bank.

Similarly to phishing attacks, pharming may also be used to pilfer user passwords. But pharming attacks do not need to set up baiting messages as phishing attacks normally do and hence may disguise themselves better and trap people in more easily.

To counter pharming attacks, it is important for users to make sure that their DNS software and the hosts files have not been compromised and that the URL they are visiting is the right one before doing anything else.

1.2.3.4   Dictionary Attacks

For security reasons, only encrypted passwords, that is, not in their original form, should be stored in a computer system. This prevents attackers from learning the passwords even if they break into the system. In early versions of UNIX and Linux operating systems, for example, the encrypted user passwords of the system are stored in a file named `passwd` under directory `/etc`. This encryption is not a one-to-one encryption. Namely, the encryption algorithm can calculate the ciphertext string of a given password, but the ciphertext string cannot be uniquely decrypted. Such an encryption is also referred to as an *encrypted hash*. In early versions of UNIX and Linux operating systems, user names and the corresponding encrypted user passwords stored in the `passwd` file were ASCII strings that could be read by users. In later versions of UNIX and Linux operating systems, however, the encrypted user passwords of the system are no longer stored this way. Instead, they are stored in a file named `shadow` under directory `/etc`, which is an access-restricted system file.

In the Windows NT/XP operating system, for another example, the user names and the encrypted user passwords are stored in the system's registry in a file named `SAM`. They can be read using special tools, for example, `pwdump`.

Dictionary attacks take advantage of the way some people use dictionary words, names, and dates as passwords. These attacks find user passwords from their encrypted forms. A typical dictionary attack proceeds as follows:

1. Obtain information of user names and the corresponding encrypted passwords. This was done, for example, in early versions of Unix or Linux by getting a copy of the

`/etc/passwd` file. In Windows XP, it can be done using `pwdump` to read the system registry.

2. Run the encryption routine used by the underlying system on all dictionary words, names, and dates. That is, compute the encrypted hash for each dictionary word, each name, and each date.

3. Compare each output obtained from Step 2 with the encrypted passwords obtained from Step 1. If a match presents, a user password is found. In other words, suppose that w is a word and $w' = crypt(w)$ is the output of the encryption routine $crypt$ on input w. Suppose that u and p_u are a pair of user name and encrypted password of user u. If $w' = p_u$, then w is user u's password or is equivalent to user u's password, for w may not be unique.

Step 2 is computationally intensive, for there are many words, names, and dates. To avoid carrying out this costly computation each time an encrypted hash is given, one would want to precompute Step 2 and store the results (i.e., password-hash pairs) in one table, so that one only needs to do a table lookup to find the corresponding plaintext password from the given encrypted hash. But such a table will be humongous. Constructing a *Rainbow table* helps to reduce the table size and make the computation at Step 2 manageable.

Rainbow Tables

A rainbow table is a table of two columns constructed as follows: let r be a function that maps an encrypted hash of a password to a string in the domain of possible passwords. This function r is referred to as a *reduction* function, for the length of a password is typically shorter than the length of its encrypted hash value. The function r can be defined in a number of ways. For example, suppose that the domain of passwords is a set of all possible eight-character strings. Let h be a cryptographic hash function that, on an eight-character password, generates a 16-character long hash value. Then, we may define r as follows: For any eight-character string w, function r on input $h(w)$ returns the last eight characters of $h(w)$. Function r may also return the first eight characters of $h(w)$ or any combination of eight characters selected from $h(w)$. Note that r is not an inverse function of h.

Let w_{11} be a given password. Apply h and r alternatively to obtain a chain of passwords that are different pairwise:

$$w_{11}, w_{12}, \cdots, w_{1n_1},$$

where n_1 is a number chosen by the user, and

$$w_{1i} = r(h(w_{1,i-1})),$$
$$i = 2, 3, \cdots, n_1.$$
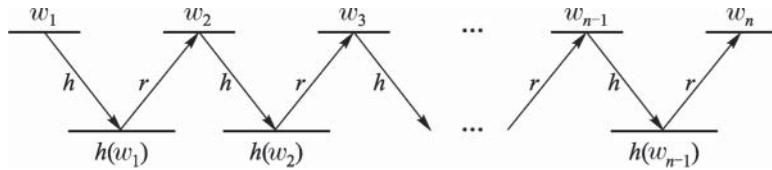
Store

$$(w_{11}, h(w_{1n_1}))$$

in the rainbow table, where w_{11} is in the first column and $h(w_{1n_1})$ is in the second column. Figure 1.1 depicts the construction of a rainbow table.

Now, choose a new password w_{21} (i.e., w_{21} has not been generated in previous chains). Repeat the same procedure for another round to obtain

$$w_{22}, w_{23}, \cdots, w_{2n_2},$$

Figure 1.1   Construction of a rainbow table

where n_2 is a number chosen by the user and $w_{2i} = r(h(w_{2,i-1})$ for $i = 2, 3, \cdots, n_2$, such that the first chain and the second chain are disjoint. That is, for any $1 \leq u \leq n_1$ and $1 \leq v \leq n_2$, we have $w_{1u} \neq w_{2v}$. Store

$$(w_{21}, h(w_{2n_2}))$$

in the rainbow table. Performing this procedure k times will generate k rows in the rainbow table as follows:

Password	Hash value
w_{11}	$h(w_{1n_1})$
w_{21}	$h(w_{2n_2})$
$\cdots$	$\cdots$
w_{k1}	$h(w_{kn_k})$

where w_{j1} is the first password in the jth chain, $h(w_{jn_j})$ is the encrypted hash of the last password in the same chain, and the chains are disjoint pairwise.

Let $f : A \to B$ and $g : B \to A$ be two functions. Let $y \in B$ and $i \geq 0$. Define $(f \circ g)^i(y)$ as follows:

$$(f \circ g)^i(y) = \begin{cases} y, & \text{if } i = 0, \\ f(g((f \circ g)^{i-1}(y))), & \text{if } i \geq 1. \end{cases}$$

Let Q_0 be an encrypted value of a password w. That is, $Q_0 = h(w)$. If

$$h((h \circ r)^i(Q_0)) = h(w_{jn_j})$$

for some $i \geq 0$ and some j with $1 \leq j \leq k$ and $i \leq j$, then w is possible to appear in the jth chain of $w_{j1}, \cdots, w_{jn_j}$. Thus, the following algorithm may help find w.

1. Set $Q_1 \leftarrow Q_0$ and $t \leftarrow 0$. Let $n = \max\{n_1, \cdots, n_k\}$.
2. Check if there is a $1 \leq j \leq k$ such that $Q_1 = h(w_{jn_j})$ and $t \leq n$. If yes, goto Step 3; otherwise, goto Step 4.
3. Apply r and h alternatively on w_{j1} for $0 \leq i \leq j$ times until $w_{jn_i} = (r \circ h)^i(w_{j1})$ is generated such that $h(w_{jn_i}) = Q_0$. If such a w_{jn_i} is found, return $w = w_{jn_i}$; otherwise, goto Step 4.
4. Set $Q_1 \leftarrow h(r(Q_1))$ and $t \leftarrow t + 1$. If $t \leq n$, then goto Step 2. Otherwise, return "password not found." (The rainbow table does not contain the password whose hash value equals Q_0.)

Note that we may use several different reduction functions in the same password chain, which helps avoid collisions that two different chains, starting from different passwords, may end up at the same password or at the same hash value at some point.

Remarks

It is worth noting that dictionary attacks may also be used in a positive way. For example, Windows Office allows users to encrypt Microsoft Word documents, where secret keys used for encryption are generated on the basis of the passwords selected by users. If, after a long while, a user forgets the password of a password-protected document, then the file will no longer be useful, for the user cannot decrypt it. To solve this problem, a company named Elcomsoft developed a password recovery software program using the dictionary attack techniques. This is a positive application of dictionary attacks. On the other hand, we note that if an encrypted office document is stolen, then the thief can also use this program to decrypt the document. There is a positive side and a negative side to every thing. A kitchen knife is intended to chop food, but it can also be used to harm people. Water can carry boats, but it can also topple them.

   We also note that the file `/etc/passwd` in recent versions of UNIX and Linux no longer displays the encrypted user passwords (see Exercise 1.8). This makes it more difficult for the attackers to obtain the list of encrypted passwords for launching a dictionary attack.

1.2.3.5   Password Sniffing

Password sniffers are software programs used to capture remote login information such as user names and user passwords. Common network applications such as Telnet, FTP, SMTP, and POP3 often require users to type in their user names and passwords for authentication, making it possible for a password sniffer to intercept useful login information. For remote logins, however, one may use special programs (e.g., SSH) to encrypt all messages, thus making it more difficult to sniff user passwords.

   SSH and other programs that encrypt login information such as HTTPS, however, are still vulnerable to password sniffing attacks. For example, Cain and Abel, a password recovery tool for the Microsoft Operating Systems, is a network sniffing tool that can capture and crack encrypted passwords using dictionary, brute-force, and cryptanalysis attacks. Cain & Abel can be downloaded free of charge from `http://www.oxid.it/cain.html`.

1.2.3.6   Side-Channel Attacks

Social media sites, such as Facebook, LinkedIn, and Twitter, provide user-friendly platforms for billions of users to interact with each other. Many users also like to post their personal data on social media sites for others to see. However, security measures on social media sites are not as strong as one would like. As a result, it is often easier to obtain user login information from social media sites than from online banking sites. In June 2012, for example, LinkedIn was under a massive attack from Russia, resulting in 6 million user passwords stolen, for the passwords were not encrypted properly.

   In general, attackers can legitimately obtain personal information posted by users from social media sites, including favorite food, pets, siblings, birthdays, and birthplaces, as well as the schools they graduated from, and the places they grew up in. Many of these items are the typical questions the users are asked to verify their identity when logging to their banking accounts. To make things worse, people tend to use the same passwords for multiple accounts, including their banking accounts. Thus, social media has become a side channel for attackers to obtain user passwords of relevant banking accounts.

1.2.3.7 Key-Logging Attacks

A Key logger is software that records key strokes of the user at the point of entry. Eavesdropping keystrokes is a more effective method to capture passwords entered by the user on the keyboard before the passwords are encrypted. Pressing a key on the keyboard will also generate radiation, which may be exploited to learn keystrokes. Attacks such as this are referred to as tempest attacks. We may use anti-key-logging software tools to counter key-logging attacks.

1.2.3.8 Password Protection

The following rules and practices can help protect passwords from pilfering:

1. Use long passwords, with a combination of letters, capital letters, digits, and other characters such as $, #, &, %. Do not use dictionary words, common names, and dates as passwords. This rule makes guessing attacks and dictionary attacks arduous.
2. Do not reveal your passwords to anyone you do not know. Do not submit to anyone who acts as if he has authority. If you have to give out your password to someone you trust, do so face to face. Avoid telling passwords over the phone or using email. This practice helps prevent social engineering breaches.
3. Change passwords periodically and do not reuse old passwords. This rule helps defend users against patient and persistent attackers who may keep on running dictionary attacks on all possible strings formed using the first rule and hope that they may get lucky. Attackers may also keep records of old passwords they have identified.
4. Do not use the same password for different accounts. Thus, even if a user's password for a particular account is compromised, the user's other accounts would still be safe.
5. Do not use remote login software that does not encrypt user passwords and other important personal information. This practice makes password sniffing difficult.
6. Shred all discarded papers using a good paper shredder. This practice makes it difficult for attackers to find useful information from discarded old documents.
7. Avoid entering any information in any popup window, and avoid clicking on links in suspicious emails. Instead, go to the legitimate Website directly using the true URL address, and follow the directions there. This practice helps counter password sniffing and reduce the chance of being caught by phishers.
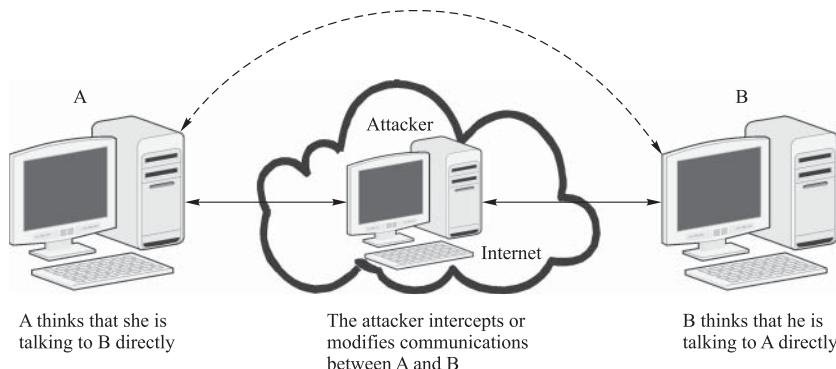
1.2.3.9 Other User-Authentication Methods

Authentication using user passwords is so far the most widely used authentication method.

Traditionally, there are three methods for proving one's identity. The first method uses secret passwords. The second method uses biometrics of unique biological features, for example, fingerprints and retinas. The third method uses authenticating items, for example, passes and certificates of identification. These three methods have been applied and implemented in computer applications.

The first method is implemented in the form of user names and user passwords.

The second method is implemented in the form of connecting biometric devices to a computer, for example, fingerprint readers and retina scanners. These devices are relatively more expensive to acquire and maintain and so are often used in a tightly controlled environment

Figure 1.2 Man-in-the-middle attacks. The solid lines represent the actual communications, and the dash line represents the perceived communication between user A and user B

where high levels of security are required. For example, instead of using credit card readers at check-out stands to authenticate credit holders and link payments to their accounts, using fingerprint readers is just as convenient and is more secure.

The third method is implemented in the form of electronic passes authenticated by the issuer. Certain authentication protocols (e.g., Kerberos) use this method to authenticate users.

Authentication using user passwords is the easiest method to implement and so far the most commonly used authentication method.

1.2.4 Identity Spoofing

Identity spoofing attacks allow attackers to impersonate a victim without using the victim's passwords. Common identity spoofing attacks include *man-in-the-middle* attacks, *message replays*, *network spoofing*, and *software exploitation* attacks.

1.2.4.1 Man-in-the-middle Attacks

In a man-in-the-middle attack, the attacker tries to compromise a network device (or installs one of his own) between two or more users. Using this device, the attacker can intercept, modify, or fabricate data transmitted between users. The attacker will then forward them as if they have not been touched by the attacker. For example, the attacker may intercept an IP packet sent by user A, modify its payload, and then send the modified packet to user B as if it comes from user A. This way, both users may still believe that they are directly talking to each other, without realizing that the confidentiality and integrity of the IP packets they receive have already been compromised (see Fig. 1.2).

Encrypting and authenticating IP packets are common measures to thwart man-in-the-middle attacks. This is because the attacker cannot read or modify an encrypted IP packet without decrypting it. Also, the attacker has no way to authenticate a modified or fabricated IP packet to convince the receiver that it comes from a legitimate sender.

1.2.4.2 Message Replays

In a message replay attack, the attacker first intercepts a legitimate message, keeps it intact, and then retransmits it at a later time to the original receiver. In some authentication protocols, for example, after user A proves herself to the system as a legitimate user, she will be given an authentication pass. With this pass, she will be able to obtain services provided by the system. This pass is encrypted, and so it cannot be modified. However, the attacker may intercept it, keep a copy, and use it later to impersonate user A to get the services from the system.

The following are common mechanisms for thwarting message replay attacks:

1. Attach a random number to the message. This number is referred to as a *nonce*. When a user receives a message whose nonce appeared before, he knows that this message is a replay, which is then discarded. This method, however, requires that users keep a record of every nonce they first encounter, which may not be practical.
2. Attach a time stamp to the message. When a user receives a message whose time stamp is old, he knows that this message is a replay. This method, however, requires that all networked computers be synchronized with little error. While not a problem in local area networks, accurate synchronization is difficult to achieve in wide area networks.
3. The best method to thwart message replay attacks is to use a nonce and a time stamp together. Using this method, synchronization does not have to be very accurate, and the user only needs to keep track of the nonces he encounters in a short and fixed time interval. The user stores a nonce in a record with a time stamp when it is first recorded. When this time stamp becomes old, the nonce is removed. The length of the time interval is determined by the worst-case error of an achievable synchronization. A message is considered as a replay only when its nonce is already in the record or its time stamp is out of the time interval.

1.2.4.3 Network Spoofing

IP spoofing is one of the major network spoofing techniques. It consists of *SYN flooding*, *TCP hijacking*, and *ARP spoofing*. ARP spoofing is also referred to as *ARP poisoning*.

SYN Flooding

SYN flooding exploits an implementation side effect of the TCP/IP network protocols. In a SYN flooding attack, the attacker fills the target computer's TCP buffer with a large volume of SYN control packets, making the target computer unable to establish communications with other computers. When this happens, the target computer is called a *muted* computer or a *silenced* computer. The TCP buffer is a set of contiguous memory locations allocated by the underlying network application program. It is used to store TCP packets that have been received but not yet processed.

To launch a SYN flooding attack against a target computer, the attacker sends to it a large number of crafted SYN packets, each requesting to establish TCP connections. The term *crafted SYN packet* means that the source address contained in the SYN packet is a legitimate IP address, but the host computer on that address is not reachable. This host computer may be powered off or taken off the network. We call such a computer a *dead* computer. Detecting whether an IP address is unreachable can be done using the ping command (or

other commands in case a live computer has been hardened to not respond to the `ping` command). If an IP address does not respond to `ping`, then it is probably unreachable. The `ping` command is a common network management tool based on the ICMP protocol. The attacker uses crafted SYN packets to avoid being tracked down. And he uses a legitimate source IP address to ensure that the crafted SYN packets will be delivered to its destination, because the domain name server will drop IP packets with fake IP addresses.

According to the three-way handshake procedure in the TCP protocol, the victim's computer is obliged to send an ACK packet to the source IP address contained in the SYN packet it receives and waits for an ACK packet to be sent back from that IP address. However, the host computer with that source IP address is not reachable, and so it will not respond. Thus, the victim's computer will never receive the ACK packet it is waiting for, forcing the crafted SYN packet to remain in the TCP buffer until its lifetime expires. During this period of time, the TCP buffer is completely occupied by (i.e., flooded with) crafted SYN packets, and so the victim's computer will have no room in the TCP buffer to establish any new connection with another computer. The victim's computer is then considered muted.

TCP Hijacking

Suppose that computer V is a company computer and user A is an employee of that company and is going to log on to computer V from home. User A's computer sends a SYN control packet to V and now suppose that an attacker intercepts this packet. The attacker then uses the SYN flooding attack to mute computer V, so that V cannot complete the three-way handshake protocol with user A's computer. If the attacker can predict the correct TCP sequence number for the ACK packet that is supposed to be sent to A from the muted computer V, then the attacker can craft an ACK packet and send it to user A's computer. The crafted ACK packet uses the correct TCP sequence number and V's IP address as the source IP address. User A's computer receives the ACK packet and verifies that it has the correct TCP sequence number. It then sends an ACK packet to the attacker to complete the three-way handshake procedure with the attacker. Thus, the TCP connection that user A's computer has established is with the attacker, instead of with V.

To see how this works, we note that the TCP protocol uses the sequence number in its TCP header to identify which TCP packets belong to the same communication. Figure 1.3 depicts the TCPv4 header format. As the TCP protocol header does not contain the source IP address, the TCP-layer software would not check the legitimacy of the IP addresses contained in the IP header. See Fig. 1.4 for the standard IPv4 header format. The IP protocol routes

16-bit source port number			16-bit destination port number	
32-bit sequence number				
32-bit acknowledgement number				
4-bit hdr length	6 reserved bits	6 control bits	16-bit window size	
16-bit TCP checksum			16-bit urgent pointer	

Figure 1.3   The standard TCPv4 header format

4-bit version	4-bit hd length	8-bit type of service (TOS)	16-bit total length (in bytes)	
16-bit identification number			3-bit flags	13-bit fragmentation offset
8-bit time to live (TTL)		8-bit protocol	16-bit header checksum	
32-bit source IP address				
32-bit destination IP address				

Figure 1.4   The standard IPv4 header format

the IP packet it receives to the destination on the basis of the information contained in the IP header. It does not keep track of the header information of previous IP packets it received. Thus, checking the source IP address at the IP layer does not help identify whether the source IP address in the current IP packet is the same as those in previous IP packets. This shows that the working of the TCP/IP protocol suite (its early implementation in particular) actually makes TCP hijacking possible. To stop TCP hijacking, it is important to use software (e.g., `TCP wrappers`) that checks IP addresses at the TCP layer.

In 1994, Kevin Mitnick, a resident in North Carolina of the United States, launched TCP hijacking attacks from his home and broke into several major companies' computers a few thousand kilometers away in California. Mitnick was later convicted and sentenced to 5 years in prison for this crime.

ARP Spoofing

Computers are identified by unique media access control (MAC) addresses. MAC addresses are also called physical addresses. ARP is an address resolution protocol at the link layer, which converts the destination IP address in the IP header to the MAC address of the underlying computer at the destination network. In an ARP spoofing attack, the attacker changes the legitimate MAC address of an IP address to a different MAC address chosen by the attacker (see, e.g., Exercise 1.7.2).

To prevent ARP spoofing attacks, checking is the key. In particular, we should strengthen checking procedures of MAC addresses and domain names and make sure that the source IP address and the destination address in an IP packet have not been changed during transmissions.

1.2.5   Buffer-Overflow Exploitations

Buffer overflow, also referred to as *buffer overrun*, is a common software loophole exploited by attackers. A buffer is a set of contiguous memory locations allocated to a process. The size of the buffer is fixed in its declaration in the program. A buffer overflow occurs if the process writes more data into the buffer than it can hold. The following is a simple C program that writes the `buffer` of eight bytes with a string `str` of 34 bytes, causing it to overflow.

```
int main() {
  char buffer[8];
  char *str = "This is a test of buffer overflow.";
  strcpy(buffer, str);
  printf("%s", buffer);
}
```
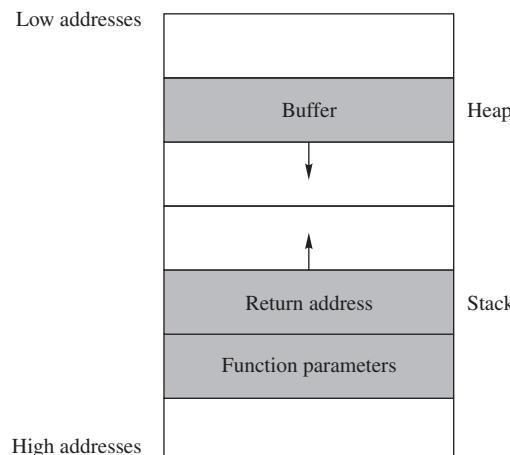
It is possible to exploit buffer overflows to redirect the victim's program to execute attackers' own code located in a different buffer area. Such attacks often exploit function calls in standard memory layout, where the buffer is placed in a *heap* and the return address of the function call is placed in a *stack*. The stack is in the higher end of the memory space, while the heap is in the lower end, where they grow toward each other and shrink away from each other (see Fig. 1.5). The following are general steps of this type of attacks:

1. Find a program that is vulnerable to buffer overflows. For example, programs that use string-based functions (e.g., `strcpy()` and `strcat()`) are vulnerable, for they do not check bounds. These functions would copy as many characters as possible until a NULL byte is encountered.
2. Figure out the address of the attacker's code.
3. Determine the number of bytes that is long enough to overwrite the return address.
4. Overflow the buffer that rewrites the original return address of the function call with the address of the attacker's code.

In reality, exploiting buffer overflows to breach security is often a complex and difficult procedure.

The best way to prevent buffer overflow attacks is to close the doors of overflow. That is, one should always add statements to check bounds when dealing with buffers in a program. Avoid using string functions that do not check bounds.

Figure 1.5   Typical memory layout for function call

1.2.5.1   Compiler Protections for Buffer Overflow
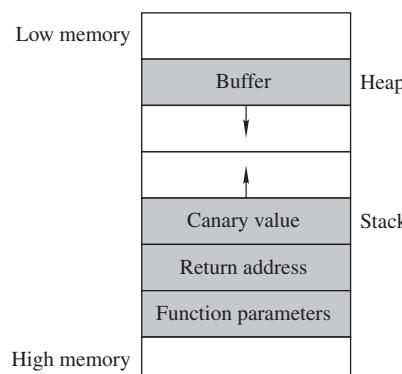
Buffer overflow often has a simple programming fix. Unfortunately, these fixes are often overlooked. To combat this problem, compiler-induced protections have been developed, one of which is the notion of *canary values*. Borrowing its name from the coal-mining practice of lowering a canary into a coal mine to determine if sufficient oxygen exists for miners to enter, a canary value is a special value stored on the programs execution stack, which helps to detect if the return address from a function has been altered. This value is pushed on to the stack immediately after the return address. If a buffer-overflow attack is executed, the heap will likely be overflown into the canary value *and* the return address (see Fig. 1.6). Thus, if the attacker manages to overwrite the return address, then it is likely that they will also overwrite the canary value and thus be detected.

To enable the canary value to protect from buffer overflow, the function prologue and epilogue code generated by the compiler must be modified to deal with the canary value. The prologue must be modified such that the canary value is pushed onto the stack after the return address. The epilogue code must be modified to check that the canary value is valid.

If the same canary value is used in every program, every time a function call is made, the attacker would easily be able to construct buffer overflow attacks. To launch an attack on the system that uses the same canary value for every function call, the attacker merely places the canary value in the correct location in the data used for buffer overflow. The check of the canary in the function epilogue will pass, and thus the return address will vector off to the attacker's malicious code. To correct this problem, a random canary value is often used. A random canary value is chosen at execution and used for just that execution. This means that every time the attacker runs the potentially vulnerable code, the canary is different, and thus the attacker cannot use the attack that works with the fixed canary values.

1.2.6   Repudiation

In some situations, the owner of the data may not want to admit ownership of the data to evade legal consequences. He may argue that he has never sent or received the data in question.

Figure 1.6   Typical memory layout for a function call that uses a canary value

Repudiation is straightforward if the data has not been authenticated. Even if the data has been authenticated, repudiation is still possible when the underlying authentication methods or the communication protocols contain loopholes. The owner of the authenticated data may be able to convince the judge that, because of the loopholes, anyone could have easily fabricated the message and made it look like it was produced by him.

Secure encryption and authentication algorithms are effective mechanisms to counter repudiation attacks.

1.2.7   Intrusion

Intrusion in network security means that an illegitimate user, also known as intruder, gains access to someone else's computer systems. The intruder may turn a victim's computer into his own server, which may result in stolen computing resources and network bandwidth from the victim. The intruder may also steal useful information residing in the victim's computer.

Configuration loopholes, protocol flaws, and software side effects may all be exploited by intruders. Opening TCP or UDP ports that should not be open is a common configuration loophole. TCP and UDP ports are entry points of network application programs.

Intrusion detection is a technology for detecting intrusion incidents. Closing TCP and UDP ports that may be exploited by intruders can also help reduce intrusions.

1.2.7.1   IP Scans and Port Scans
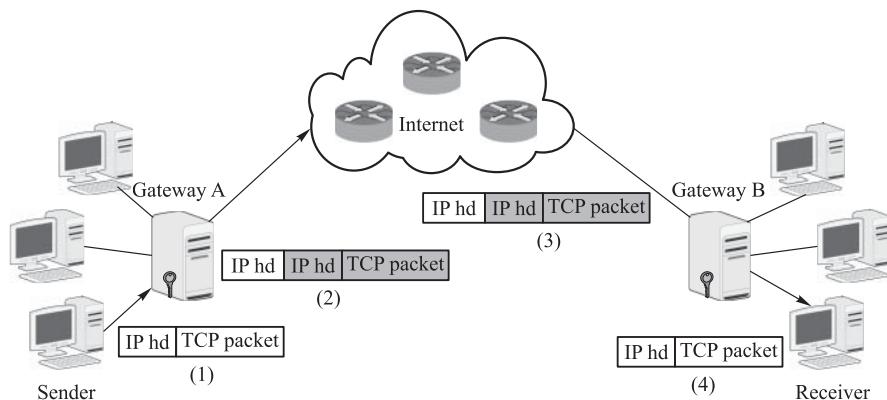
IP scans and *port scans* are common hacking tools. IP scans search for existing IP addresses in the Internet, and port scans search for open ports in a computer. Attackers use IP scans to search for potential targets and port scans to identify open ports that are vulnerable in the targets.

However, IP scans and port scans can also help users to identify in their own systems which ports are open and which ports may be vulnerable. Several such products are available. For example, `ShieldsUP!` of Gibson Research Corporation and `Nessus` of Southwest Research Institute are two such products (see Exercise 1.19).

1.2.8   Traffic Analysis

The purpose of *traffic analysis* is to determine who is talking to whom by analyzing IP packets. Even if the payload of the IP packet is encrypted, the attacker may still obtain useful information from analyzing IP headers. An IP header contains the source IP address and the destination IP address, which reveal who is sending messages to whom. If its payload (i.e., the encapsulated TCP packet) is not encrypted, the port numbers can also be obtained. This information can be used to learn which application program is used to read the message. When preparing for a big event, individuals or organizations may frequently exchange messages before the event takes place. If the traffic analyzer learns this information from analyzing IP headers, an attacker may conclude that something big is about to happen.

The best way to combat traffic analysis is to encrypt IP headers. But an IP packet with an encrypted IP header cannot be routed to the destination. Thus, a new plaintext IP header must be inserted in front of the encrypted IP header for delivery. This may be done using a network *gateway*. A gateway is a special-purpose computer shared by many users in the local network. It

Figure 1.7 Using gateways to encrypt IP packets. (1) Sender forwards an IP packet to gateway A at the sending side. (2) Gateway A encrypts sender's IP packet (the shaded part) and routes it to the next router in the Internet. (3) The IP packet from Gateway A is delivered to gateway B at the receiving side, with certain attributes (e.g., TTL) in the plaintext IP header (shown as the unshaded part) modified. (4) Gateway B removes its header, decrypts the encrypted IP packet of the sender, and forwards it to the receiver
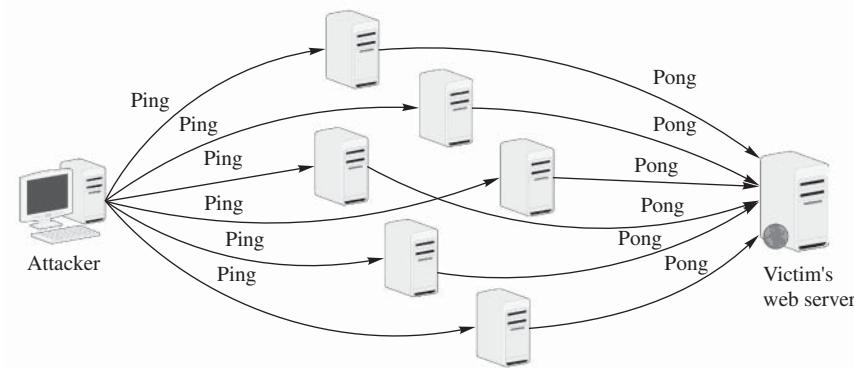
can encrypt a user's IP packet (including its header) at the sending side, decrypt the encrypted IP packet at the receiving side, and forward it to the destination MAC address. If there are no other routers between the sending-side gateway and the sender's computer, and there are no other routers between the receiving-side gateway and the receiver's computer, then traffic analysis can only reveal that the two gateways are talking to each other (see Fig. 1.7), without gaining any information about which user behind one gateway is talking to which user behind the other gateway.

1.2.9 Denial of Service Attacks

The goal of *denial of service attacks* is to block legitimate users from getting services they can normally get from servers. Such attacks often force the target computer to process a large number of useless things, hoping to consume all its critical resources. A denial of service attack, denoted by DoS, may be launched from a single computer, or from a group of computers distributed in the Internet. The latter attack is called a *distributed denial of service attack* and is denoted by DDoS.

1.2.9.1 DoS Attacks

SYN flooding is a typical and effective technique used by DoS attacks. The `smurf` attack is another typical type of DoS attack, where `smurf` is the name of the software used to execute the attack. It sends an excessive number of messages to the target computer and crashes it by consuming all its resources. In a typical `smurf` attack, the attacker sends crafted `ping`

Figure 1.8   Smurf attack

requests to a large number of computers within a short period of time, where the source IP address in the crafted `ping` request is replaced with the victim's IP address. According to the ICMP protocol, a computer that receives a `ping` request will respond to the source IP address with a `pong` message, informing the sender that "I am alive". Therefore, each computer that receives the crafted `ping` request will respond to the victim's computer with a `pong` message. Forced to process a large number of `pong` messages within a short period of time, the victim's computer will use up its computing resources and crash (see Fig. 1.8). Thus, the idea of `smurf` attacks is to crash a single target with a lot of borrowed hammers.
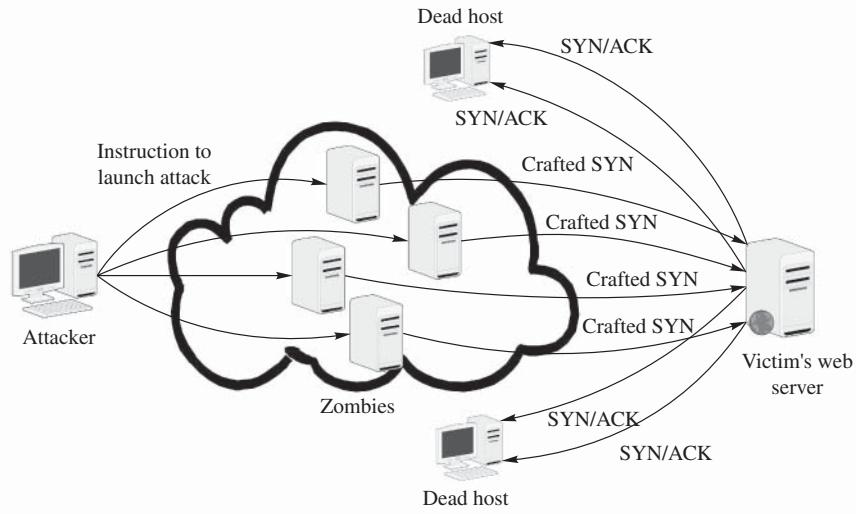
1.2.9.2   DDoS Attacks

A typical DDoS attack proceeds according to the following sequence:

1. Compromise as many networked computers as possible. This may be achieved using Trojans (see Section 1.2.10 for a description of Trojans).
2. Install special software in the compromised computers to carry out a DoS attack at a certain time later. Such software is called *zombie software*, and such a computer is called a *zombie computer* or simply a *zombie*. A collection of zombies is also called a *zombie army*, which is now typically called a *botnet*.
3. Issue an attack command to every zombie computer to launch a DoS attack on the same target at the same time.

Figure 1.9 depicts a DDoS attack. On receiving the attacker's command, each zombie computer uses SYN flooding to mute the victim's Website.

In 2000, for example, a 15-year-old high-school student in Montreal, Canada, with an assumed name "Mafiaboy," launched a DDoS attack against Web servers of several major companies and paralyzed these Web servers for a week. These companies, including Amazon, Cable News Network, eBay, E*Trade, Dell, and Yahoo!, suffered substantial financial losses because of this attack. Mafiaboy was sentenced to spend 8 months in a youth detention center.

Figure 1.9   A DDoS attack using SYN flooding to mute the victim's Website

1.2.9.3   Spam Mail

Spam mails are uninvited emails, which may be commercial messages or phishing messages. While not intended to bring the victim's computer out of service, spam mails do consume computing resources. Spam mails are annoying, particularly when one's mailbox is filled up with them.

Standard electronic messaging systems have made it possible for individuals and companies to send unwanted bulk messages to people. Such individuals or companies are often referred to as *spammers*. Spamming can occur in any form of network applications, but email spam is by far the most common spamming form. According to a recent statistics, about half a billion spam emails are sent in every single day. In other words, each email user is expected to receive about eight spam messages a day. Spamming also occurs in Web search engines, Instant Messaging, blogs, mobile phone messaging, and other network applications.

Spam filters are software solutions to detect and block spam mails from reaching the user's mailbox.

1.2.10   Malicious Software

Software intended to harm computers is *malicious software*. Malicious software is also referred to as *malware*. Common forms of malicious software include *virus*, *worms*, *Trojans*, *logic bombs*, *backdoors*, and *spyware*.

1.2.10.1   Viruses and Worms

A computer virus is a piece of software that can reproduce itself. However, a virus is not a standalone program. In other words, it must attach itself to another program or another file.

A program or file that contains a virus is called an *infected program* (also called an *infected host*). When an infected program is transmitted to another computer, the virus that lives in it is also transmitted along with its host program.

The execution of a virus is initiated by the infected host. Namely, only when an infected program is executed or an infected file is opened, a virus contained in it may get executed. When executed, a virus may do harm (e.g., delete system files) to the system where its host resides or replicate itself to infect other healthy hosts in the system.

A computer worm is also a piece of software that can reproduce itself. Unlike a virus, a worm is a standalone program. In other words, it does not need a host to live in. A worm can execute itself at any time it wishes. When executed, a worm may do harm to the system where it resides or replicate itself to other systems through networks.

There are two common measures to combat viruses and worms. One measure deploys *virus scans* to detect, quarantine, and delete infected hosts and worms. The other measure, consisting of the following rules, blocks viruses and worms from entering a computer:

1. Do not download software (e.g., games) from untrusted Websites or other sources.
2. Do not open any executable file given to you by someone you do not know.
3. Make sure that software patches are installed and up to date.

    Neglecting software patches may be fatal. For example, in the summer of 2001, many systems that run Microsoft Internet Information Services (IIS) were hit by the Code Red worm, the Nimda worm, and the Code Red II worm. These worms made headline news, and they all exploited the same loophole in IIS. Microsoft knew about this problem and provided a patch to correct it a year earlier. However, many system administrators did not install this patch and thus left wide open doors into their systems for the worms to come in and do damage.

1.2.10.2 Trojans

Trojans are also called Trojan horses. The name Trojan horse came from a Greek legend. Legend has it that ancient Greeks, wanting to apprehend a beauty named Helen, attacked the fortified city of Troy but failed. Faking a retreat, the Greeks left behind a huge, hollow wooden horse with a number of soldiers hidden inside. Not suspecting any danger, the Trojans hauled the wooden horse inside the city as a trophy. At night, the Greek army returned, and the soldiers hidden inside the wooden horse went out and opened the city gates for the invasion troops to come in. The city of Troy fell.

In the realm of network security, Trojans are software programs that appear to do one thing but secretly also perform other tasks. Trojans often disguise themselves as desirable and harmless software applications to lure people to download them. When they are executed by the user, the hidden functions contained in them, which now have the user's access rights, do harmful things secretly. Games and network management tools available for free downloads from unknown Websites often are Trojans. Trojans may also use appealing names such as `AntiSPYware.exe` or `Real_Player.exe` (note that the real one is `RealPlayer.exe`) to trap users to use them.

The same measures of combating viruses and worms can also be used to combat Trojans. Virus scans can also detect, quarantine, and delete Trojans.

1.2.10.3   Logic Bombs

Logic bombs are subroutines or instructions embedded in a program. Their execution is triggered by conditional statements. For example, a company employee working on a development project may install a logic bomb inside a program. The bomb will be set off only if the employee has not run the program in a certain period of time. When that condition is met, it would mean that the employee was fired some time before. The logic bomb in this case is used to gain revenge against the employer.

There are three measures to counter logic bombs. First, employers should always do their best to take care of their employees, so that none would be tempted to place a logic bomb. Second, project managers should hire an outside company or form a special team of reviewers from a different group of people other than the developers to review the source code. Third, relevant laws should be established so that employees who planted logic bombs will face criminal charges. With these countermeasures in place, unhappy employees would think twice before planting logic bombs in programs.

1.2.10.4   Backdoors

Backdoors are secret entrance points to a program. They are often inserted by software developers to provide a short cut to enter a password-protected program when attempting to modify or debug code. These backdoors that avoid the typical password entrances of normal users may later be discovered and used by attackers. Attackers who compromise network systems have been known to insert their own backdoors so that they can more easily re-enter later.

We note that, with the increase of outsourcing software development projects and other vital tasks to other countries, the potential for logic bombs and backdoors also increases. The major counter measure of backdoors is to check source code, which should be conducted by an independent team.

1.2.10.5   Spyware

Spyware is a type of software that installs itself on the user's computer. Spyware is often used to monitor what users do and to harass them with popup commercial messages. *Browser hijacking* and *zombieware* are the most disastrous kinds of spyware.

Browser Hijacking
Browser hijacking is a technique that changes the settings of the user's browsers. It may replace the user's default Website with a different Website selected by the attacker. Or it may stop the user from visiting the Websites he or she wants to visit. For example, the Google redirect virus, which affected a lot of people in 2012/2013, redirects the browser to a Website that has nothing to do with the search query entered by the user.

Zombieware
Zombieware is software that takes over the user's computer and turns it into a zombie for launching DDoS attacks or into a relay that carries out harmful activities such as sending spam email or spreading viruses. Therefore, the purpose of zombieware is to hijack computers.

In addition to hijacking browsers and computers, spyware can also do a number of other things, including the followings:

Monitoring
Spyware can be used to monitor and report to a Web server or to the attacker's machine a user's surfing habits and patterns, such as which Web pages the user has browsed and which products the user has purchased.

Password Sniffing
Spyware can be used to sniff user passwords by logging users' keystrokes using a keystroke logger. A keystroke logger is a program that can capture user names and user passwords when the users type them in.

Adware
Adware is software that automatically displays advertising materials on the user's computer screen. The common form of adware is popup windows with commercial material. While not intended to harm users, adware consumes user's precious computing resources and is annoying.

To counter spyware, users may use antispyware software to detect and block spyware. Microsoft's Windows Defender, for example, is such a software tool. Windows Defender is available as a free download.

Most modern antivirus software includes checks for spyware, adware, and hacking tools such as keystroke loggers and network sniffers.

1.3 Attacker Profiles

Attackers are often characterized as *black-hat hackers*, *script kiddies*, *cyber spies*, *employees*, and *cyber terrorists*.

1.3.1 Hackers

Hackers are people with special knowledge of computer systems. They are interested in subtle details of software, algorithms, and system configurations. Hackers are an elite group of well-trained and highly motivated people. Depending on their motives, hackers are further characterized as *black-hat hackers*, *white-hat hackers*, and *grey-hat hackers*.

1.3.1.1 Black-Hat Hackers

Black-hat hackers are people who hack computing systems for their own benefit. For example, they may hack into an online store's computer system and steal credit card numbers stored in it. They may then use the stolen credit card numbers to buy merchandise or sell them to other people. Black-hat hackers are the wicked doers in network security.

Note that, without the "black-hat" modifier, hacker is not a derogatory term. News media, however, have widely used hackers to denote black-hat hackers. To avoid confusions, several authors have suggested to use *crackers* to denote black-hat hackers.

1.3.1.2 White-Hat Hackers

White-hat hackers are hackers who have high moral standards. They hack computing systems for the purpose of searching for security loopholes and developing solutions. They publish security problems and solutions at security conferences, on dedicated Websites, or through special mailing lists. White-hat hackers are the righteous doers in network security.

1.3.1.3 Grey-Hat Hackers

Grey-hat hackers are hackers who wear a white hat most of the time but may also wear a black hat once in a while. For example, when they discover attacks, instead of reporting the incidents to law enforcements, grey-hat hackers may take the matter in their own hands and strike the attackers back themselves. Grey-hat hackers are the Robin Hood type people in the world of network security.

1.3.1.4 Disclosures of Security Problems

When discovering security vulnerabilities in a software product, white-hat hackers and grey-hat hackers would often work directly with the vendors of products to help them fix the problems before they release the details of their discoveries. Whether a full disclosure of their findings should be allowed is an ongoing debate, in part due to the perceived view of the white-hat hackers and the grey-hat hackers that the vendors are not doing enough to fix security problems in a timely manner.

1.3.2 Script Kiddies

Script kiddies are people who use scripts and programs developed by black-hat hackers to attack other people's computers. Such scripts and programs are often referred to as *hacking tools*. Script kiddie is a derogatory term. It is used to indicate that script kiddies only know how to copy and use a hacking tool. They do not understand how it works, and they are not capable of writing any hacking tool themselves. Script kiddies like to crack any target they possibly can, so that they can say to others in the underground cracker community that "I am smarter." Script kiddies may also attack targets with high profiles just to attract the attention of the media.

Although they do not know how to write hacking tools or understand how an existing hacking tool works, script kiddies are dangerous. Many of them are just teenagers who do not care about, or are not mature enough to know, the consequences of their actions. However, they are energetic, and they are everywhere. They launch attacks from unexpected places and at any time, which could inflict serious damages to other people.

1.3.3 Cyber Spies

Cyber espionage takes place at all levels. It could be an individual activity or an organizational effort. Cyber spies collect intelligence through intercepted network communications. They could be working for a good cause or just for money.

Governments run cyber intelligence units to intercept network communications and decipher encrypted messages. The National Security Agency (NSA) and the Central Intelligence Agency (CIA), for example, are the two largest intelligence agencies of the U.S. government. The NSA hires many first-class mathematicians and computer scientists to work for it. Many of them are professors at U.S. universities. They teach during school years and work for NSA during summers. They study encryption algorithms and develop cryptanalysis tools. This sort of work has helped win battles.

During World War II, for example, the intelligence department of the U.S. Pacific Fleet was able to partially decipher Japanese secret code, which helped Admiral Chester W. Nimitz, the Commander in Chief of the Pacific Fleet, deduce the Japanese scheme of invading the Midway Atoll in the mid-Pacific. Nimitz seized the opportunity and ordered his two aircraft carriers to ambush the approaching Japanese invasion forces. With another barely restored carrier joining in the battle a few days later, American aviators sunk four Japanese carriers, with the cost of losing only one carrier. The battle of Midway became a turning point, from a defensive to an offensive campaign for American Pacific naval forces.

1.3.4   Vicious Employees

Vicious employees are people who intentionally breach security to harm their employers. They may plant logic bombs or open backdoors in programs they help develop. They may act as script kiddies to attack company computers to get the attentions of their employers. They may also act as cyber spies to collect and sell company secrets for money.

1.3.5   Cyber Terrorists

Terrorists are extremists who do not hesitate to use extreme means to destroy public property and take innocent life. Cyber terrorists are terrorists who use computer and network technologies to carry out their attacks and produce public fear. Attacks by cyber terrorist have not been reported yet. However, if they did attack, cyber terrorists would be extremely harmful.
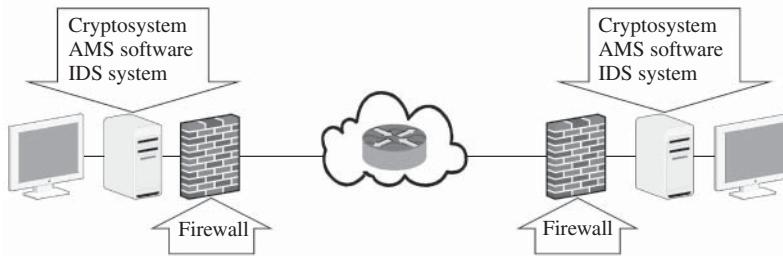
1.3.6   Hypothetical Attackers

The hypothetical attackers this book deals with are black-hat hackers, script kiddies, greedy cyber spies who are willing to betray their countries or organizations for monetary benefits, and vicious employees. Attackers of these four kinds may be wicked, but they are not terrorists. Cyber terrorists, on the other hand, are the die-hard enemies, and so they may need to be dealt with using a different set of measures not addressed in this book.
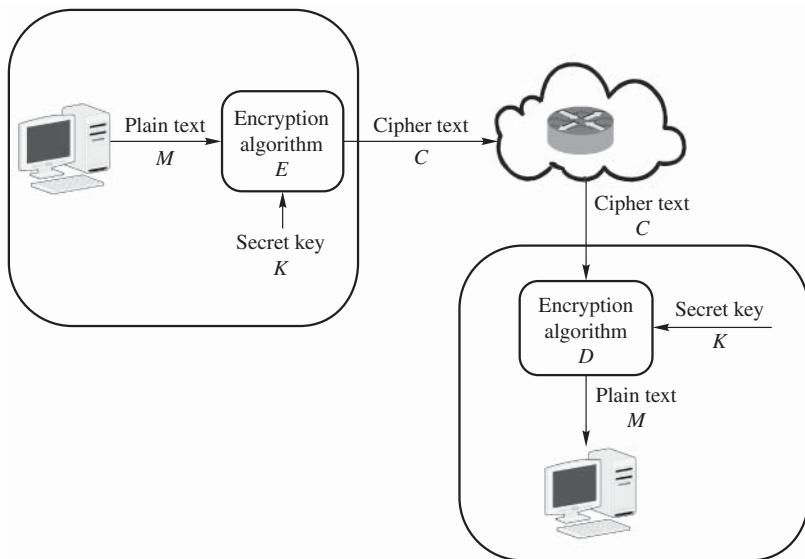
1.4   Basic Security Model

The basic security model consists of four components: *cryptosystems*, *firewalls*, antimalicious-software *software* (AMS software), and intrusion detection systems (IDS system). Figure 1.10 shows this security model.

Cryptosystems use computer cryptography and security protocols to protect data. Security protocols include encryption protocols, authentication protocols, and key management

Figure 1.10 Basic security model

Figure 1.11 Network model of cryptosystem

protocols. Figure 1.11 shows the encryption and decryption components. It is customary to use E to denote an encryption algorithm, D its decryption algorithm, and K the secret key.

Firewalls, AMS software, and IDS systems are used to protect data stored in networked computers. Firewalls are special software packages installed in computers and networking devices that check incoming and outgoing network packets. Certain features of firewalls have also been incorporated into hardware devices to achieve faster processing speeds. AMS software scans system directories, files, and registries to identify, quarantine, or delete malicious code. IDS systems monitor system logins, study user behaviors, and analyze log files to identify and sound alarms when intrusions are detected.

In addition to using firewalls, AMS software, and IDS systems, we may also set up sacrificial decoy machines to lure attackers' attentions away from important computers. Decoy machines are also known as *honeypots*.

This book is centered around these four major components. This book also introduces honeypot technologies.

1.5   Security Resources

Network security is not something that can be taken care of once and for all, because when old security problems are solved, new security problems will appear. Thus, network security defenders will have to fight against the attackers continuously. Network security is an art of defense in digital form. This book covers basic principles, methods, and techniques of network security. It does not and cannot cover every aspect of the area. It does not and cannot tell you what the new attacks are going to be. Fortunately, there are many online security resources available to help you win this fight. The following are a few popular resources.

1.5.1   CERT

Founded in 1988, CERT is a research institute affiliated with Carnegie Mellon University. Its full name is Computer Emergency Response Team. Its budget comes mainly from the U.S. government.

CERT was the earliest organization devoted to studying security problems and offering practical solutions to system administrators to help secure their computer systems. It sends monthly reports to subscribers, free of charge, of any security breach identified in the current month, with recommended solutions. In addition, CERT also trains computer security personnel. Its Website is `www.cert.org`.

1.5.2   SANS Institute

Founded in 1989, SANS Institute is a nonprofit organization devoted to collecting, archiving, and publishing computer security information. It provides this information to users free of charge. SANS stands for SysAdmin, Audit, Network, and Security. In addition, SANS Institute also offers computer security training, issues certification, and funds research. Its Website is `www.sans.org`.

1.5.3   Microsoft Security

Microsoft security is Microsoft's official Website devoted to providing security information for Microsoft products. It provides security updates to Microsoft users. Its Website is `www.microsoft.com/security/default.mspx`.

1.5.4   NTBugtraq

NTBugtraq is a moderated open list service for users to post and discuss security exploits and bugs in Microsoft's products. Its Website is `www.ntbugtraq.com`.

1.5.5   Common Vulnerabilities and Exposures

The Common Vulnerabilities and Exposures (CVE) database is a free database maintained by the Mitre Corporation. CVE tracks software vulnerabilities across all major software products from all major vendors. This is the most widely used collection of information on security vulnerabilities. The vulnerabilities contained within the database are scored and ranked using the Common Vulnerability Scoring System (CVSS), a standard maintained by NIST. The CVE Web site is `www.cve.mitre.org`.

1.6   Closing Remarks

Sun Tzu said: All warfare is based on deception. Attackers may attack us where we are unprepared and appear where they are not expected. Network security is no exception. For example, even if we develop an unbreakable encryption algorithm, if keys are not managed properly, attackers can still break the encryption system, not by attacking the encryption algorithm, but by exploiting loopholes in key management protocols.

We must assume that attackers are capable of using any means available to achieve their objectives. They avoid what is strong and strike at what is weak. Therefore, we must remember that it will only take a small blow at a weak spot to bring down any apparently strong defense system. Also, a defense system would just be an ornament if one could bypass it. The famous Maginot Line, for instance, is an example. During World War II, the French militaries were confident that the Maginot Line of concrete fortifications they spent 10 years to build along the French-German border could stop German aggression. The German invasion forces, however, did not assault the Maginot Line directly as anticipated by the French. Instead, they dispatched motorized troops to quickly cut through the Low Countries of Belgium and the Netherlands and invaded France from unexpected locations in a third country. Lessons like this have taught us that in network security, we must constantly examine our network defense mechanisms from all aspects and fortify any weak point as soon as it is identified.

1.7   Exercises

1.7.1   Discussions

1.1.   Have you experienced any network security attack described in the text? If so, please share your experience with the class. If you have experienced network security attacks not described in the text, please describe them in detail.

1.2.   How did you solve the network security problems you encountered?

1.3.   Why type of attackers do you think attacked you?

1.4.   Networked computers are managed by different types of people. What type of people do you think are most vulnerable to network security attacks?

1.5.   Why do you think phishing and pharming attacks are so common? What measures would your suggest to counter them?

1.6.   Why do you think network security must be a multiple-layer defense mechanism?

1.7.2   Homework

1.1.   This book assumes that the reader has taken a computer network course, or has sufficient experience working with computer networks.

   (a) Describe the major structure of a TCP packet and explain the main functions of the TCP headers.

   (b) Describe the major structure of an IP packet and explain the main functions of the IP headers.

   (c) Explain the three-way handshake protocol in the TCP protocol and describe its main functions.

   (d) Describe the difference between UDP and TCP. Give an example of an application that would use UDP and an application that would use TCP. Justify your answers.

1.2.   On the basis of your understandings of network protocols, answer the following questions:

   (a) Explain the main functions of the ARP protocol.

   (b) Explain the main functions of the ICMP protocol.

   (c) Explain the major functions of routers, switches, and gateways.

   (d) Explain the major functions of the SMTP protocol.

1.3.   Describe the major differences between IPv4 and IPv6.

1.4.   Use network administration tools to familiarize yourself with network configurations.

   (a) In the Windows operating system, `ipconfig`, `ping`, `tracert`, `nslookup`, and `netstat` are common network administration tools. On a machine running Windows, go to the `start` menu, select `run`, and then enter `cmd` to open a command window. Execute these five network administration tools. Explain the results you observe. For each of these admin tools, use option `-?` to list each option of the tool and explain its usage. For example, enter `ipconfig -?` to learn all options of `ipconfig` and explain their usage.

   Execute the following commands and explain the results you observe:

```
ping cs.uml.edu
ping www.google.com
tracert www.yahoo.com
netstat -e
```

   (b) In the UNIX and Linux operating systems, `ping`, `nslookup`, `netstat`, and `arp` are common network administration tools. You may use the `man` tool to find out how to use these tools. For example, enter `man netstat` to list all information about `netstat`. On a machine running UNIX or Linux, execute these tools and explain the results you observe.

   (c) Open a `cmd` window on a Windows machine and execute `ipconfig /all` to list all information of the network setup of your PC. Write down the host name, MAC address of your network adapter, IP address, subnet mask, and default gateway of your PC.

In the UNIX and Linux operating systems, you may find the IP addresses of all hosts in the system in `/etc/hosts`. On a machine running UNIX or Linux, enter `more /etc/hosts` and explain what you see.

(d) Open a `cmd` window on a Windows machine and execute `netstat -ano`. Identify which ports are TCP ports, which ports are listening, which ports have established connections, and which ports are UDP ports. Also identify what programs are running on these ports.

To find out what program is running on a given port number, first identify its PID (process ID), and then open the Windows Task Manager window (e.g., you may open it by pressing the three keys of `Ctrl-Alt-Del` simultaneously). Select `View`, `Select Columns`,···, and `PID`. Then select `Process` and find out which program is running on the PID. For example, suppose that the following line is included in the result returned from `netstat -ano`:

```
Proto Local Address  Foreign Address State      PID
TCP    127.0.0.1:1026 127.0.0.1:1027  ESTABLISHED 664
```

From here, we know that Port 1026 is a TCP port where a connection has been established and its PID is 664. From the Windows Task Manager, we find out that `postgres.exe` has PID 664. Thus, we know that `postgres.exe` is running on Port 1026.

(e) Open a `cmd` window on a Windows machine and execute `arp -a`. It lists the physical address of your router. Compared to the physical address given by `ipconfig /all`, what is the difference between these two physical addresses? On a UNIX machine, enter `arp -a` on the UNIX prompt to list the ARP table in your machine.

1.5. Network sniffers are also referred to as packet sniffers. Network sniffers are software used to monitor network connections and obtain information of network packets. `TCPdump` and `Wireshark` are widely used packet sniffers with free downloads from `www.tcpdump.org` and `www.wireshark.org`, respectively. `TCPdump` has been around for many years. `Wireshark`, formerly known as `Ethereal` until 2006, is newer and has a nicer GUI interface.

If you are using a Windows machine, download from `http://www.wireshark.org/` and install `Wireshark-win64-1.12.0.exe` (64-bit) or `Wireshark-win32-1.12.0.exe` (32-bit) or its newest version. This version contains `WinPCap4.0.1`. You will need to install it as well. If you are using other operating systems, please download and install from the Wireshark Website a corresponding version of `Wireshark`. Then execute `Wireshark`.

We want to sniff ARP packets. For this purpose, on the open window of "The Wireshark Network Analyzer," select `Capture`, `Options`, and then select network card in the `Interface` box. In the `Capture Filter` empty box type in `arp`, and then select `Start` to launch ARP sniffing. At this time, you will see a popup window titled "(the name of the network card): Capturing - Wireshark". To generate ARP packets (so that you have something to sniff), open a Web browser and visit a few Websites. After a short while, you will see that ARP packets have been captured in the popup window. Select `Capture` on the menu bar, then select

`Stop` to stop sniffing. Note that the Wireshark window is divided into three portions. The upper portion shows the ARP packets that have been captured, the middle portion shows the packer headers, and the lower portion shows the contents of the ARP packets in hexadecimal and ASCII code. Explain what you see.

Disclaimer: Network sniffing should only be done on a network where one has permission to do so and all parties are aware that it is (or may be) occurring. Otherwise, it may inadvertently break the Federal electronic eavesdropping and wiretap laws.

1.6. We often want to use a network sniffer to only pick up the types of packets we are interested in.

(a) Execute `Wireshark`. Select `Options` from the menu of `Capture`. A window named "`Wireshark: Capture Options`" will pop up. In the empty box of `Capture Filter`, enter `tcp port 25`, and then click `Start` to begin sniffing. Send yourself an email message. Then click `Capture` on the menu bar and select `Stop`. Explain what you see.

(b) Execute `Wireshark`. Select `Options` from the menu of `Capture`. A window named "`Wireshark: Capture Options`" will pop up. In the empty box of `Capture Filter`, enter `tcp port 80`, and then click `Start` to begin sniffing. Open a Web browser to visit a few Websites. Then select `Capture` on the menu bar and select `Stop`. Explain what you see.

1.7. Finding statistical structures in a cipher text message is a common cryptanalysis method. For example, given a ciphertext message, we first calculate the frequency of each letter occurring in the messages. We then compare these letter frequencies with the letter frequencies one would expect to have in the underlying language. If there is a clear one-to-one correspondence, we will then know which ciphertext letter corresponds to which plaintext letter. This method is especially effective to break earlier designed encryption algorithms.

In the English language, for example, the following table lists the expected frequency of each letter, in the decreasing order of frequencies.

e	t	a	o	i	n	s	h	r	d
12.702	9.056	8.167	7.507	6.996	6.749	6.327	6.094	5.987	4.253
l	c	u	m	w	f	g	y	p	b
4.052	2.782	2.758	2.406	2.360	2.228	2.015	1.974	1.929	1.492
v	k	j	x	q	z				
0.978	0.772	0.153	0.150	0.095	0.074				

If the ciphertext message is not long enough, we may not be able to obtain a frequency curve similar to that of the statistical frequency curve. Thus, we may also want to calculate frequencies of strings of two or more letters, for they may correspond to common letter strings such as er, or, the, and ing. Such information would be useful. Suppose that we have the following ciphertext message with punctuation and space removed, where the plain-text message is written in English:

```
NTCGPDOPANFLHJINTOOFITOVJHJCTMMHIHEMTCPFDWTSOFSHTOGFWTE
TTJJTBTOOFSZOVEOCHCVCHPJHOCGTOHNQMTOCNTCGPDCGFCSTQMFBTO
FBGFSFBCTSHJCGTQMFHJCTYCXHCGFAHYTDDHAATSTJCBGFSFBCTSHJC
GTBHQGTSCTYCCGHONTCGPDQSTOTSWTOCGTMTCCTSASTRVTJBZHJCGTQ
MFHJCTYCFJDOPPJTBFJOTFSBGAPSCGTQMFHJCTYCASPNFIHWTJBHQGT
SCTYCEZBPNQFSHJICGTASTRVTJBZPATFBGMTCCTSFIFHJOCCGTLJPXJ
BPNNPJASTRVTJBZHJCGTVJDTSMZHJIMFJIVFIT
```

(a) Calculate the frequency of each letter.

(b) Compare your calculated letter frequencies with the statistical letter frequencies, and find out the plaintext message properly punctuated and spaced.

1.8. In early versions of UNIX and Linux operating systems, login passwords of the users are stored in the file `/etc/passwd` in the following format:

```
user:password:ID:group-ID:comment:home:shell
```

where the encrypted passwords were readable text strings (e.g., $3/25\#2\%v$), making dictionary attacks possible. Recent versions have fixed this problem by only showing a symbol $*$ or x indicating that the user is required to enter the password. Suppose that your `/etc/passwd` file contains the following entry:

```
nobody:*:65534:10:NFS Nobody (normal):/:/bin/nosh
```

Explain the meaning of each component in this entry.

1.9. Let h be a hash function and r a reduction function. Let T be a rainbow table of k rows for D under h and r, where the jth row is $(w_{j1}, h(w_{jn_j})$ for $1 \leq j \leq k$. Let $Q_0 = h(w)$ and $Q_1 = (h \circ r)^i(Q_0)$, where $i \geq 0$. Suppose $Q_1 = h(w_{jn_j})$ for some $1 \leq j \leq k$ and $i \leq j$. Answer the following questions:

(a) Under what conditions will w appear in the jth chain of $w_{j1}, \cdots, w_{jn_j}$?

(b) Under what conditions will w not appear in the jth chain of $w_{j1}, \cdots, w_{jn_j}$?

(c) We note that in practice, h often maps a shorter password to a longer hash value. Thus, without lost of generality, we may assume that h is one-to-one for a given domain of passwords. It is common practice to use different reduction functions to produce a password chain. Why can this technique help increase the likelihood that w appears in the jth chain of $w_{j1}, \cdots, w_{jn_j}$?

1.10. Two readers of the first edition shared with us their experiences on distributing passwords:

- "I can recall a security incident where the user name and password were accidentally sent off the secure network to an unauthorized email address. While no further security incidents occurred, it was certainly possible for an attacker to recover the username and password and do serious damage to the network."

- "At work, we ONLY give passwords over the phone, and of course only when we know who we are speaking to. Of all the no-no's in network security, sending password via insecure emails has to be at the top of the list."

Describe your practice of distributing passwords and discuss their pros and cons.

1.11. "Early in my career as a Web developer," a reader of the first edition told us, "I created a Website for a friend. I created the FTP login name and password using the same first eight characters of the name of the site. In about 6 month time, somebody hacked into the site and put their own silly page in place of her content. Once I regained control, I created a high-strength password using a combination of uppercase and lowercase letters, numbers, and symbols, with a minimum of eight characters. I have since followed this practice for every Web login I create."

(a) Discuss what the Web developer did before being hacked was problematic.

(b) Do you think that the weak password the Web developer set up was the actual cause of his friend's computer being hacked? Justify your answer.

(c) Do you think that the Web developer's solution to the problem was effective? Justify your answer.

1.12. "Previously when I had DSL and an old router at home, the wireless encryption didn't work and I would occasionally find unauthorized users on my network," a reader told us. "I knew enough not to conduct any sensitive business using the wireless connection, but did once make an online shopping transaction using a credit card (I was being lazy). Within 2 days, there were fraudulent charges on my credit card." Make an educational guess what might happen and justify your answer.

1.13. "My account was compromised by a brute force attack a while back when I was playing an online game," said a reader of the first edition. "In response I purchased an RSA token and linked my account to it, so that even if my password was compromised again my account could never be fully accessed without the token code."

(a) Discuss why playing an online game might breach user accounts.

(b) Research the use of RSA tokens and explain whether using an RSA token would help secure user accounts for playing online games. Justify your answer.

1.14. A reader of the first edition reported the following social engineering attack happened to him: "Sometime ago I received a random phone call from someone (later identified as a fraudster) who wanted to speak to a senior person in my company.

Caller: Hello. Can I speak with the head of operations? (The fraudster did not mention a name, just a common job title, trying to sniff out a name and email address from me if I mistakenly mentioned the name of the person.)

Me: Can you please mention the name of the person you intend to reach, as we have many operation departments and heads around here (Baiting the fraudster)?

Caller: I have lost the business card he gave me and can't remember the details. Can you be kind enough to give me the name, email address, or direct number of one of the heads who might likely be in the same business meeting where I met the person I am trying to reach?

At this point the caller was suspicious enough that I transferred the call to my company's security investigative unit, which took it up from there."

(a) Describe whether you have a similar procedure at work and how you think the procedure could be improved.

(b) If you receive similar phone calls at home, what would you and should you do? Note that some crooks may call you that your tax returns contained errors and you must call a certain number to clear it up; otherwise you will be in trouble.

Others may change the content a little by, for example, telling you that your neighbors reported to the police department that you did something wrong. Anyway, all they try to get you to do is to call a certain number and then scare you to death so that you would provide them information or give them money.

1.15. Good baits are essential for a phishing attack to be successful. Baits are often presented in the form of email messages and Websites that appear to be authoritative. Links contained in phishing messages are traps, leading to Websites controlled by attackers. Discuss how to identify phishing messages and phishing sites.

1.16. The following phishing attacks were experienced by some of the readers. In each instance, describe what you would do if it happened to you.

  (a) "A few years ago one of my network passwords on LinkedIn was compromised, possibly through phishing or pharming. As a result, spam messages spoofing my identity were sent to my connections. I discovered this when some of my connections notified me and said that they knew that I would not send such messages. I changed my passwords (and continue to do so periodically) and as a result the problem has not occurred since."

  (b) "I received phishing emails 2 months ago (around November 2013), claiming to be from FedEX. There were several clues that they were bogus. For example, the content and the Subject Line did not look right, and nowhere did I see anything similar to `fedex.com`. The message was very generic about some complication in delivery, and it urged the recipient to open up a file attachment that looked very suspicious. Sometimes you can tell an email is a phishing attack because the link it gives you in the message does not look right."

  (c) "I have been getting attacked very frequently through emails lately (in early 2014). One example is an email stating that I was offered a job, and asked me to fill out a form with all of my personal data. This is obviously an attempt to get my personal information because legitimate employers wouldn't offer me a job if they didn't know anything about me. My solution to the phishing attacks are never to login to anything through an email, and never giving out information to anyone I can't authenticate or trust. I think one of the main reasons that my phone number and email address were compromised is my resume being posted on sites like `monster.com`. As soon as I find a job I'm taking it down!"

  (d) "I've received tons of phishing emails over the years. When I was a customer of a local bank, I encountered the best phishing email I have ever received. I received an email that looked like it was from the bank with a link to the Website. I clicked the link. When I was about to login, I noticed that the color of the site did not look right. I took a closer look at the URL, and realized that it was not the official Website of the bank. It almost tricked me. I blocked the sender and emailed the bank who then passed it along to the FBI."

  (e) "I've encountered several cleverly disguised email invitations to provide account information. Thankfully, I've never entered personal information that was requested, but I know that many less security conscious people have. The best way to combat phishing is to ignore requests for personal information that emanate from the Web. When in doubt, call the institution directly, and not with the number on the email."

(f) "Just last week (i.e., in mid January 2014), I received a phishing email. It appeared to come from an organization I know, but the actual email address was obviously not, and contained (false) links to reset my password. I reported it to the IT Help Desk."

1.17. Do you agree with the following rule of thumb when dealing with possible phishing emails: "If an email comes from a company or individual I don't recognize, I delete it. If it's really important, they will call me!" Justify your answer.

*1.18. ARP maps an IP address to a MAC address of a computer. Thus, assigning a different MAC address to an IP address redirects message to a different computer. Conduct the following experiment. Let A, B, and C be three PCs connected to the same local area network (LAN) running Microsoft Windows (or Linux). Suppose that you have an user account on each of these computers and you have the same user name `fool` on computers B and C. Suppose that you can modify the ARP table on computer B (e.g., such as what a super user may do). On computer C, run `arp -a` to obtain its MAC address. Then on computer B, run `arp -s` to modify its ARP table to map B's IP address to C's MAC address. Wait for a while or reboot B to let B's new ARP table take effect. Now, send an email message from your account on computer A to your account `fool` on computer B. This message will be redirected to your account `fool` on computer C. Verify this result in your experiment.

1.19. Use port scans to check your computer's open ports.
(a) Use `ShieldsUP!` to scan your computer's open ports for possible loopholes. Visit `www.grc.com` and click the `ShieldsUP!` link. Then move your mouse down to find the `ShieldsUP!` link. Click the link and follow the instructions to scan your computer's open ports.
(b) `Nessus` has features similar to `ShieldsUP!`. It checks open ports and tries to determine what programs are running on them. Visit `www.nessus.org` and download `nessus`. Next, use `nessus` to scan your computer.

1.20. "Port scans are very frequent on our network by outside and inside attackers," a reader told us. "We simply block repeat offenders." Argue that this is a good solution. Can you think of a better approach to counter port scans? Justify your answers.

1.21. Web servers are easy targets of DoS attacks. For example, attackers may bombard a Web server with a large number of login attempts in a short period of time, forcing the Web server to use up its computing resources for checking passwords.

Web servers may use a picture verification service as follows: when receiving a login request, the Website opens a login page that will display, in addition to the usual windows for entering user name and password, a few characters in different colors or shapes, embedded in a small frame of colorful background and a window to enter these characters. To complete the login procedure, the user must also type in these characters. If these characters are not entered correctly, the Web server will not proceed to check the user name and password. This mechanism is typically used to prevent automation of services the Website provides and level the playing field (e.g., Ticketmaster uses this service to prevent scalpers from using a program to purchase tickets).

Explain how automation of services could be used to launch DoS attacks, and why the picture-verification mechanism may help stop DoS attacks.

1.22. A reader of the first edition shared this experience with us: "I sometimes saw employees bringing in a small personal switch and connecting it to the company LAN. Occasionally these switches would cause broadcast storms that resulted in denial of service on the LAN. It was easy to find these switches using tools such as wireshark and then remove them." These are rogue switches. Explain how to use wireshark to identify rogue switches.

1.23. "We had experienced repeated DoS attacks on our corporate Web servers," a reader said. "The attackers were flooding our servers with external communication requests, so much so that the servers could not respond to legitimate traffic. To counter these attacks, we moved to a SaaS solution for our online customer software from AWS (Amazon Web Services), and transitioned to a similar model for our corporate Web servers using a Rackspace provider, beefing up its security and redundancy during the transition."

(a) Conduct a research on AWS, SaaS, and Rackspace.

(b) On the basis of your research, argue that the solution the company took is a good one.

1.24. Sometimes, a legitimate application may affect the performance of your system. Googlebot, for example, is such an application. It is a highly debatable issue whether such applications are considered malware. Googlebot is a Web crawling tool developed by Google, which is also referred to as spider. It is used to crawl the Internet and discover new and updated pages for the Google index. Here is a story shared by a reader: "I worked with a customer who was facing extremely slow performance in their portal at the time of open enrollment for a new service. It was identified that it was Googlebot causing the problem, which was crawling the content on their external facing portal. They then worked with Google and the internal security team to filter the traffic to eliminate the additional crawling time."

Discuss this issue and justify your opinions.

1.25. Microsoft operating systems have become the household operating systems by people in all walks of life. Thus, computers that run Windows operating systems are hackers' major targets. Consequently, loopholes, flaws, and defects have been found one after another.

Use Microsoft Baseline Security Analyzer (MBSA) to analyze security settings of your Windows operating system and other Microsoft products. To do so, first download and install the newest version of MBSA from the following link:

```
www.microsoft.com/technet/security/tools/mbsahome.mspx
```

Then execute MBSA to scan your Windows system.

1.26. Server programs that run in the background of your computer are entry points to your computer from the network. Some of these programs are necessary, some are not, and some are malicious programs downloaded by careless users. Suppose that you are running Windows XP on your computer.

(a) Follow the following procedure to identify which server programs are running and which server programs have been closed: Select `Run` from the `Start` menu, then type in `msconfg`. Press the `OK` button to open the window of `System Configuration Utility`, and click `Services`. For example, is your `DHCP` client running or stopped?

(b) Follow the following steps to find out the usages of XP-supported services: Select `Run` from the `Start` menu, then type in `services.msc`. Press the `OK` bottom to open the window of `Services` and select `Services`. Select each service one at a time and read about its usage. For example, what is the usage of the `DHCP` client?

1.27. Back Orifice is a computer program designed for remote system administration to control a computer running the Windows operating system from a remote location. But it may also be use to log keystrokes easily. Other key-logging tools include hardware keylogger and invisible keylogger. Conduct a survey on keyloggers and write a paper reporting your findings.

1.28. Critical information may be stolen when you shop online. A reader shared with us the following story: "Just last year (i.e., in 2013) I had my credit card information stolen from what I believed to be a keystroke-logging attack. Since then I've beefed up my security and installed an anti-keylogger."

Identify and discuss security vulnerabilities you can think of associated with online shopping.

1.29. As we mentioned in the text, an apparently well-protected network could be brought down via an apparently minor trick. The following is a story shared by a reader of the first edition: "I am a system administrator for a large company with employees worldwide. My site produces sensitive hardware and software products. We have a very strong network security team keeping our network safe. However, about 2 years ago (i.e., in 2012), espionage hackers still managed to get into our network. As secure as our network was, the hackers used Outlook Web Access (OWA) to get into our network, retrieving a large volume of data in 2 days. The attack took the following steps:

1. They first collected information form media and by calling the company disguised as a sales person or government authority. They managed to retrieve email addresses from local users who were assigned to my site.
2. They then used a spoofing method to send emails to users from the known employees to other employees.
3. They would send emails with Trojans only during off hours, so that the email recipient would use OWA at home to access their email and bypass the firewalls and network security protocols at work.

The email spoofing was being done for about 2 weeks until a employee replied to the hacker, thinking it was an employee from a company laptop off hours. When the employee returned to the office the next day the hacker was able to bypass the firewall and get into the network. We had to make major changes to the network from top down including the following:

1. Removed all OWA installations.

2. Spent a large sum of money to purchase firewalls and network security devices and distributed them globally.

3. Hired ten additional network security professionals.

4. Removed all local administrative rights from domain computers.

5. Purchased USB token devices to key staff members with administrator rights on computers. The devise was a custom token that had both a certificate embedded in it associated with the employee and a password management code. For example, SafeNet. Inc sells such products (see `http://www.safenet-inc.com/data-protection/authentication/pki-authentication/`).

6. Required all employees to change passwords every 25 days for a year.

Also as a result of this attack, I had to travel for about a year to multiple locations two to three times a month to give network security training to users. We have not been hacked again so far and we continue to make improvements on our network. We send out intentional spoofing emails every now and then to test our employees and I have to give remote training to those who fail the tests."

(a) Discuss the attacking techniques the attacker was using in this attack.

(b) Discuss how to identify spoofing emails.

1.30. "Since the MafiaBoy attack in 2000, on a regular basis, our own servers have been hit by DDoS attacks on average once every 2 years," said a reader. Have you experienced any DDoS attack? If so, what measures did you take to counter DDoS attacks?

1.31. "I have discovered that DDoS effects can occur by accident on an alarming rate due to improperly configured application software. It is helpful if the network system is configured to shut down the troubled application. Otherwise, it can be difficult to use diagnostic tools to find it." Discuss how you may configure the system to detect misconfigured applications to address this reader's concern.

1.32. "Our servers were taken down with the Code Red and copycat worms in the early 2000s." The reader who shared this experienced also made the following comments: "Everybody I know has suffered from malicious software attacks at one time or another–no matter how careful you are. If you are not completely protected with updated anti-virus/malware software and more importantly, safe browsing habits, it can happen again to almost anyone."

Share your own experiences using one or more concrete examples of malicious software attacks you encountered.

1.33. "Several lab computers I administered were infected with viruses that hijacked the system," a reader told us. "The infected system displayed a message supposedly from the FBI saying that the system was in violation of copyright laws and for a small fee could be cleared up (using a credit card of course). It frankly was too much work to clean it up so we instead just reinstalled the system."

Suggest a way to cleanup such viruses without reinstalling the system. Justify your answer.

1.34. "Back at the dawn of time when I was an undergrad," said a reader, "my university's computers were riddled with viruses. One that I remember in particular was

the Stoned virus. It would attack the file allocation table in the DOS operating system, making the computer unable to find any file. Antivirus tools were not readily available then, so I kept a floppy disk that was just for the university computers. Once I used a disk at school, I marked it and never used it anywhere else. I'm sure that helped spread the virus on the university computers but it kept the viruses off my own PC."

While floppy disks are no longer in use, USB sticks are still widely used today. How do you like the reader's approach to viruses and justify your answer.

1.35. "In early 2013 I built a Website for a local restaurant using Drupal. It was a relatively straightforward site, with no actual commerce function. It didn't have any personal information on it, or in the MySQL database back end. I hid the administrative login for Drupal, but not very well. I just put it somewhere where a site user couldn't navigate to. However, Drupal is set up in such a way that site structures can be guessed by hackers, or perhaps mine was just crawled somehow by a program specializing in this sort of thing. Almost every day I received requests to add users to the site. The restaurant went out of business last week, so I took the site down, which stopped the requests right away."

Can you suggest what happened to the Website and offer a fix if the site were to be run?

1.36. "This past year (e.g., in 2013), I developed a quick and easy site for one of our meetings on a subdomain especially for it outside of our usual security model. One morning, my inbox was flooded with hundreds of error messages (i.e., error messages sent from sites to developers with all the parameters of the requests), all with SQL statements embedded in an open text field's input string. Fortunately, none of the attempts to access the database was successful and that day we came up with a procedure to prevent it from happening in the future by (1) validating all input before it is submitted and (2) blocking any suspicious statements before they get submitted to the database."

Describe how to identify suspicious SQL statements.

1.37. "A few months ago in 2013, a coworker of mine turned on an old PC hooked up to our work network and did not tell anyone. That old PC had been off line for a couple of years. Within a day or two we were having all kinds of network problems, from performance slowdown to other weird issues. Because this PC was behind our firewall it was not picked up right away. It turned out that all these problems were caused simply by an old worm in that old PC. To remedy the situation we first removed that old PC. We then manually scanned all our PCs and servers with multiple antivirus and malware tools, for the worm had also compromised the antivirus software installed on the PCs. We shut down the ports and services the worm was spreading through until we were sure that the network was clean. Once clean we were able to reconnect everything and went back to business as usual. This took about 72 hours to remedy. This incident made us revise our security policies and procedures to prevent things like this from happening again."

What do you think the new security policy should be for this reader's company to avoid similar incidents mentioned in the message from happening again?

1.38. Junk email filters are software tools used to prevent junk email messages from entering your mailbox. Microsoft Office Outlook has this feature. To set it up, open Office Outlook and click `Actions`. Point the mouse to `Junk E-Mail`, click `Junk E-mail options`, `Safe Lists Only`, `Safe Senders`, and `Add`. Type in here the email addresses you wish to receive email messages from, then click `OK`. Likewise, you may also specify the email addresses that you do not want to receive messages from. Describe how this can be done.

1.39. "A server I managed was once compromised by an attacker. The attacker gained root access using buffer overflow and installed a Trojan that replaced standard Linux commands with infected ones, opening up ports for the attacker to attack other locations. We fixed the problem by a complete system reinstall from original media and applied proper security patches."

   Describe what each of these two remedies do.

1.40. **Canary Values.** The GNU Compiler Collection (GCC) supports buffer overflow protection using random canary values.

   (a) Determine what the `-fstack-protector` and `-fstack-protector-all` flags are used for when compiling code using the GNU C (`gcc`) and C++ (`g++`) compilers.
   (b) Compile C code with and without the `-fstack-protector-all` flag and disassemble the executables using the Linux tool `objdump`, with the `-d` option, compare the output and determine what code is responsible for inserting the canary value in the prologue and what code is responsible for checking the canary value in the epilogue.

1.41. "When I was a kid I had problems with adware and Trojans on my Windows PC. Since then I always make sure that my machines have security software installed. Now I am using Norton Internet Security and it seems to get the job done. We also have Norton endpoint security installed on the development VMs at work." Have you experienced any malicious software attack that even the Norton security tools did not help remove them?

1.42. "I had an infection with spyware on my home computer. It popped up a window with an instruction to download Windows antivirus software. It would popup and keep popping up until my computer would freeze because all the opened windows had used up all of the memory. I looked up how to fix the problem but it seemed so involved I finally just took the easy route: I wiped my hard drive, reinstalled the system from scratch, and downloaded antivirus and spyware tools. On a separate note, in my work we use common access cards to log in to computers and we can't even plug in a USB for fear that there might be malware on it."

   (a) What do you think happened to this person's computer?
   (b) Is the USB policy mentioned a good policy? Justify your answer.

1.43. "I have had several instances where my wife's computer became infected with some form of malware or another. She visited several questionable sites that I cautioned her against, but the joy of those sites outweighed my warnings. Of course each time

her computer was infected I would have to fix it and hear about why can't I stop her machine from being infected. To help me avoid this, I run Linux at home which I have found to be much more secure, and less susceptible to viruses."

Do you agree with this reader's last comment about Linux being much more secure and less susceptible to viruses? Justify your answers.

1.44. "I once worked for a guy as a consultant," a reader told us. "The guy started bragging about the logic bomb he created. He set things up so that 3 months after he left the company (due to downsizing), the company screens would be taken over by a faked video of a senior member of the management team having inappropriate relationships with a donkey. The company then called him back in as a consultant (since he knew the system so well) to help find the cause of the problem. He worked at a very high rate of pay for 4 months pretending to solve the problem created by him. I stopped working for him the next week."

Are you aware of any person who planned logic bombs in the software they wrote? If so, please describe it. If not, imagine and describe a situation in which a logic bomb may be planned.

1.45. "In 2012, some syndicates were able to hack into our credit-card payment systems in North America, causing us financial loss of up to $2.7 million dollars. They did this through a combination of password theft, cryptanalysis, and phishing emails. Like the textbook says: The battle against network attacks is a perpetual one as the various attackers constantly device new means to breach our network securities." Can you make an educational guess when the attack this reader mentioned might take place? Justify your answers.

**1.46. When the TCP/IP protocols and the OSI seven-layer model were devised, their designers were only concerned about how to efficiently and reliably transmit data from the source computer to the destination computer. Data security was not a concern at that time. Consequently, the TCP/IP protocols and the OSI model do not contain any built-in security mechanism. When they later realized this security weakness, protocol designers started to add all kinds of security mechanisms into communication protocols. But these early protocols were not designed for data security, and so they may not have the right framework for adding security features. Adding a security feature to a protocol not built for it is like taking out materials from a wall to mend a fence. Thus, network designers have started to investigate the following issue: if one designs a communication protocol all over again, what would be the best native architecture for including the current security mechanisms as well as for adding future security features. Think about this issue when you read the rest of the book, and try to develop a design of your own. This exercise is to be handed in at the end of the course.

2

Data Encryption Algorithms

The history of using secret writing to protect valuable information is probably as long as the history of written language itself. Computer cryptography was created to protect confidential data in digital forms, and it thrives in the Internet era. Data encryption is a critical component of computer cryptography. It uses encryption algorithms and secret keys to transform data from that which is readable to that which is unintelligible. Encryption algorithms must be reversible, so that data can be transformed, using the same secret key, from the unintelligible form back to its original form. Encryption algorithms of this kind are referred to as *conventional encryption algorithms* or *symmetric-key encryption algorithms*.

For example, let $P_0 P_1 \cdots P_{25}$ be a fixed permutation of the 26 English letters, which maps letter A to P_0, B to $P_1, \cdots$, and Z to P_{25}. Replacing each letter in a given English message according to this mapping, we can transform the message to a new form that is unintelligible to an untrained eye. This is a simple encryption algorithm, where the secret key is $P_0 P_1 \cdots P_{25}$. Exercise 1.7 uses this algorithm, where the secret key is `FEBDTAIGHKLMN-JPQRSOCVWXYZU`. Replacing each letter according to the reverse mapping, namely, replacing P_0 with A, P_1 with B, $\cdots$, and P_{25} with Z, the data in the new form can be transformed back to its original form. Thus, devising an encryption algorithm is not difficult. What is difficult is to devise good encryption algorithms.

Good encryption algorithms must satisfy a number of requirements. This chapter first describes design criteria of encryption algorithms. It then presents several common block cipher encryption algorithms, including Data Encryption Standard (DES), triple-DES, and Advanced Encryption Standard (AES). It also introduces common block cipher modes and the RC4 stream cipher. Finally, it describes how to generate secret keys.

2.1 Data Encryption Algorithm Design Criteria

Any message written over a fixed set of symbols can always be represented as a *binary string*. A binary string is a sequence of 0's and 1's. Several standard binary encoding schemes, referred to as *character code sets*, have been established to encode various sets of computer symbols for different written languages. For example, the ASCII code set encodes English letters and other commonly used symbols into binary strings; the GB 2312-80 code set encodes simplified

Chinese characters; the EBCDIC code set encodes western European languages; and the ISO 8859 code set encodes accented Latin and non-Latin European languages, including Greek, Semitic, and Hebrew. The Unicode code set and the ISO 10646 code set intend to unify all code sets to encode all languages. Without loss of generality, we assume that plaintext data and ciphertext data are binary strings.

Binary digits 0 and 1 are called *bits*. To reduce computation overhead costs, encryption algorithms should only use bit operations that are easy to implement on electronic computers. For instance, permuting bits in a binary string is a simple binary operation.

Let X be a binary string. Define the length of X, denoted by $|X|$, to be the number of bits contained in X. If $|X| = \ell$, we say that X is an ℓ-bit binary string.

Let $a \in \{0, 1\}$ and k be a non-negative integer. We use a^k to denote the following k-bit binary string:

$$a^k = \underbrace{aa \cdots a}_{k \ a's}.$$

Let $X = x_1 x_2 \cdots x_\ell$ and $Y = y_1 y_2 \cdots y_m$ be two binary strings, where $x_i, y_i \in \{0, 1\}$. We use XY to denote the concatenation operation of X and Y; that is,

$$XY = x_1 x_2 \cdots x_\ell y_1 y_2 \cdots y_m.$$

For clarity, we may also use $X \parallel Y$ to denote the concatenation operation *XY*.

2.1.1   ASCII Code

The ASCII code set consists of all 7-bit binary strings (see Appendix A), representing non-negative integers from 0 to 127. The first 32 ASCII codes and the last ASCII code are control codes, which are not displayable. ASCII codes from number 32 to 126 encode uppercase and lowercase English letters, decimal digits, punctuation marks, and arithmetic operation notations. Because a byte that is an 8-bit binary string is the basic storage unit in a computer, we often use one byte to represent one ASCII code by prepending a zero. This allows us to expand the ASCII code set to represent up to 128 extra symbols by setting the leftmost bit in each ASCII code from 0 to 1. Sometimes, we also use the leftmost bit as a parity bit for error detection. In any case, using the 8-bit ASCII code set to encode an English message will result in a binary string of length divisible by 8. Using other code sets such as unicode to encode data may result in a binary string of length divisible by 16. Without loss of generality, we assume that any plaintext message is encoded as a binary string of length divisible by 8.

2.1.2   XOR Encryption

The exclusive-OR operation, denoted by $\oplus$ or XOR, is a simple binary operation, where

$$0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1, 1 \oplus 1 = 0.$$

Thus, for any $a \in \{0, 1\}$, we have

$$a \oplus a = 0, a \oplus 0 = a, a \oplus 1 = 1 - a, a \oplus (1 - a) = 1.$$

We can think of the exclusive-OR operation for a single bit as addition modulo 2. Let $X = x_1 x_2 \cdots x_\ell$ and $Y = y_1 y_2 \cdots y_\ell$ be two ℓ-bit binary strings. Define

$$X \oplus Y = (x_1 \oplus y_1)(x_2 \oplus y_2) \cdots (x_\ell \oplus y_\ell).$$

Thus, $X \oplus X = 0^\ell$ and $X \oplus 0^\ell = X$.

We use E, D, and K to denote an encryption algorithm, a decryption algorithm, and a secret key, respectively. When E and D appear in the same context, it is understood that D is the reverse algorithm of E.

Let ℓ be a positive integer divisible by 8 and K an ℓ-bit secret key. Divide the plaintext data M into a sequence of blocks

$$M_1, M_2, \cdots, M_k,$$

where each block is ℓ-bit long, except possibly the last block M_k. If $|M_k| < \ell$, add an 8-bit control code at the end of M_k once or several times to obtain a new block such that its length is exactly ℓ. For example, we may use the control code nl = 00001010 to pad M_k. This procedure is called *padding*. For simplicity, we still use M_k to represent the new block.

An encryption algorithm that encrypts one block at a time is called a *block cipher* algorithm (or simply block cipher). In a block cipher algorithm, the value of ℓ is often selected to be 64 or 128. When ℓ equals the length of the basic code used in the underlying language, for example, when $\ell = 8$, we call the encryption algorithm a *stream cipher* algorithm. Thus, on the surface, the difference between a block cipher and a stream cipher is the length of the block. In stream cipher algorithms, padding is not needed.

We can use the XOR operation to design an encryption algorithm. Let K be a secret key of length ℓ. The encryption algorithm encrypts M_i to produce a ciphertext block C_i as follows:

$$C_i = E_K(M_i) = K \oplus M_i.$$

The decryption algorithm decrypts C_i into M_i as follows:

$$D_K(C_i) = K \oplus C_i = K \oplus (K \oplus M_i) = (K \oplus K) \oplus M_i = 0^\ell \oplus M_i = M_i.$$

XOR encryption is the simplest encryption algorithm. For example, let $\ell = 16$ and $K = 1001101010011011$, then E encrypts FUN as follows:

Plaintext:		F	U	N	(Padding)
ASCII:		01000110	01010101	01001110	00001010
Secret key:	$\oplus$	10011010	10011011	10011010	10011011
Ciphertext:		11011100	11001110	11010100	10010001

The XOR encryption algorithm is simple and fast. But the resulting level of security is low. For example, eavesdroppers can easily calculate the secret key K from a plaintext–ciphertext pair (M_i, C_i) as follows:

$$M_i \oplus C_i = M_i \oplus (M_i \oplus K) = K.$$

Attacks such as this that derive secret keys using a small number of samples of ciphertext data and the corresponding plaintext data are referred to as *known-plaintext attacks*.

To protect XOR encryptions from known-plaintext attacks, users must change encryption keys frequently. If each encryption key is used exactly once, the XOR encryption algorithm offers the best security there is. This security is known as *information theoretic* security. This method is referred to as *one-time pads*. To implement the one-time-pad scheme, one must first generate a long list of encryption keys sufficient for applications in the foreseeable future, make two identical copies, and distribute the list to the sender and the receiver. Both sides must then use the same keys synchronously and remove a key from the list once it is used. The one-time-pad scheme is secure and simple. However, it is unscalable. Implementing one-time pads for network communications would require each pair of users to generate, transmit, and store a huge number of secret keys. This is formidable. Thus, we must explore different methods to devise encryption algorithms that are not only secure, but also practical. On the other hand, although it is not wise to be used alone, the XOR operation still is a major operation employed in all mainstream encryption algorithms.

2.1.3 Criteria of Data Encryptions

Encryption keys must be kept secret at all times. Encryption algorithms may also be kept secret. A secret encryption algorithm is by itself a cryptosystem. For example, during World War II while fighting against the Japanese forces, the U.S. Marine Corps used the language spoken by Navajos, a remote native American Indian tribe, to generate secret codes. This encryption scheme was never broken by the Japanese.

Keeping encryption algorithms secret, however, does not help to study and verify the security of these algorithms, nor does it help to establish encryption standards. Thus, we assume that encryption algorithms are publicly disclosed, an assumption called *Kerchoffs' principle*. Only encryption keys are to be kept secret. To be practical, encryption keys must be reusable.

Good encryption algorithms must satisfy the following criteria.

2.1.3.1 Efficiency

The operations used in encryption and decryption algorithms must be easy to implement on hardware and software. Executing these algorithms should only consume moderate resources. In particular, the time complexity and the space complexity of the algorithms must each be kept within a small constant factor of the input size.

To achieve efficiency, encryption algorithms should only employ operations that are easy to implement on a computer. The following operations are common in mainstream encryption algorithms: exclusive-OR, permutation, substitution, circular shift, and operations on finite fields. Permutation, substitution, and circular shift are unary operations. The circular shift operation is a special form of permutation. A permutation is a one-to-one mapping, while a substitution is a many-to-one mapping.

2.1.3.2 Resistance of Statistical Analysis

Encryption algorithms must destroy any statistical structure in the plaintext data, making any statistical analysis useless. The common way to achieve this goal is to require encryption algorithms to satisfy the *diffusion* and *confusion* requirements.

1. By diffusion, it means that a change of a single bit in the plaintext string will cause a number of bits in the ciphertext string to be changed. These bits should be distributed in the ciphertext string as evenly as possible. In other words, every single bit in the ciphertext data is affected by a number of bits evenly spread across the plaintext string.
2. By confusion, it means that a change of a single bit in the encryption key will cause a number of bits in the ciphertext string to be changed. These bits should be distributed in the ciphertext string as evenly as possible. In other words, every single bit in the ciphertext data is affected by a number of bits evenly spread across the encryption key.

Diffusion and confusion are also referred to as *avalanche effects*.

Diffusion may be achieved by repeatedly executing a fixed sequence of operations for a fixed number of rounds on strings generated from the previous round.

Confusion may be achieved by the following method:

1. Generate a number of subkeys from the encryption key.
2. Use the first subkey to operate on the plaintext string in the first round.
3. Use each subsequent subkey in each subsequent round to operate on the new string generated from the previous round.

Combining these two methods in a coherent way, we may be able to obtain an encryption algorithm that offers both diffusion and confusion.

2.1.3.3   Resistance of Brute-Force Attacks

Suppose that the encryption key is ℓ bits long. After eavesdropping a ciphertext message C, the eavesdropper could use brute force to decipher C by calculating $M' = D_{K'}(C)$ for each ℓ-bit binary string K'. If M' is readable and makes sense, then it would likely be the original plaintext message. As there are 2^ℓ different ℓ-bit binary strings, such a brute force attack incurs a time complexity in the magnitude of 2^ℓ. Thus, ℓ must be sufficiently large to thwart brute-force attacks.

This time complexity of 2^ℓ is often used as a benchmark to determine the effectiveness of a cryptanalysis method. If a cryptanalysis method can break an encryption algorithm with a time complexity much less than 2^ℓ, then this method will be considered useful.

What value of ℓ (the length of the key) would be sufficient depends on computing technologies in the near future. It is a common belief that using $\ell = 128$ would be sufficient for many years to come.

2.1.3.4   Resistance of Other Attacks

Encryption algorithms must also resist other types of attacks. In addition to the known-plaintext attacks, these attacks include *chosen-plaintext attacks* and *mathematical attacks*.

In a chosen-plaintext attack, the attacker chooses a particular plaintext message as a bait to lure his opponents to encrypt it, where the chosen plaintext message contains useful information for the attacker. During World War II, for example, the intelligence department of the U.S. Pacific Fleet suspected that a certain code frequently occurring in the intercepted Japanese

encrypted messages meant "Midway Atoll". To confirm this suspicion, the U.S. intelligence deliberately had a plaintext message sent out, requesting a replacement of a broken facility in Midway to lure the Japanese intelligence to encrypt it. From this, they were able to confirm that their suspicions were indeed correct.

In mathematical attacks, the attacker uses mathematical methods to decipher encrypted messages. These methods include *differential cryptanalysis*, *linear cryptanalysis*, and *algebraic cryptanalysis*. Detailed discussions of these attacks are beyond the scope of this book.

2.1.4 Implementation Criteria

Implementations of encryption algorithms must resist *side channel attacks*. Side channel attacks do not attack the algorithms directly. Instead, they explore loopholes in the implementation environments. For example, the *timing attack* is a common side channel attack. In a timing attack, the attacker analyzes the computing time of certain operations, which may help obtain useful information about the encryption key. Timing attacks could be useful if the run-time of certain operations in the underlying encryption algorithm fluctuates substantially on different bit values in the encryption key.

One way to combat timing attacks is to flatten the computation-time differences between instructions by, for example, executing a few redundant operations on instructions that use much less time to execute.

2.2 Data Encryption Standard

The DES was published by the U.S. National Bureau of Standards (NBS) in 1977. NBS was the predecessor of the U.S. National Institute of Standards and Technology (NIST). DES was based on the Lucifer encryption algorithm developed by an IBM research group led by Horst Feistel. In particular, DES is a concrete realization of the Feistel cipher scheme. Its encryption and decryption structures are symmetrical, and they use four basic operations: exclusive-OR, permutation, substitution, and circular shift. DES was widely used from the mid-1970s to the early 2000s. Although gradually phasing out, DES played an important role in modern cryptography and represents a popular design paradigm for data encryption.
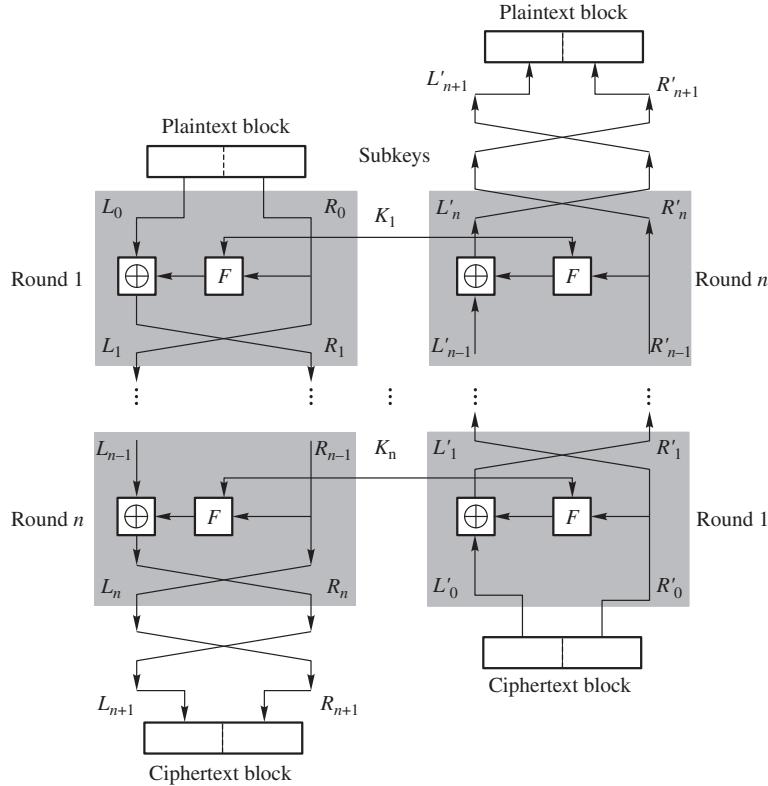
2.2.1 Feistel's Cipher Scheme

The Feistel cipher scheme (FCS) divides the plaintext string into a sequence of blocks, each of which is $2l$ bits long. FCS only uses basic operations of XOR and substitution. Let n be a positive integer. FCS first generates n subkeys of a fixed length from the encryption key K. Let these subkeys be $K_1, \cdots, K_n$. Let F denote the substitution function that takes an l-bit binary string and a subkey as input and generates an l-bit binary string as output. Divide a $2l$-bit plaintext block M into two halves L_0 and R_0 of equal length, where L_0 and R_0 are, respectively, the prefix substring and the suffix substring of M. The FCS encryption and decryption algorithms each executes n rounds of a fixed sequence of operations (see Figure 2.1).

FCS encryption executes the following operations in round i, where $i = 1, \cdots, n$:

$$L_i = R_{i-1}, \tag{2.1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i). \tag{2.2}$$

Figure 2.1   Feistel cipher scheme block diagram

After n rounds, the plaintext block $M = L_0 R_0$ is transformed to $L_n R_n$. Let $L_{n+1} = R_n$ and $R_{n+1} = L_n$ be the output of FCS encryption. That is, the FCS encryption algorithm produces a ciphertext block $C = L_{n+1} R_{n+1}$.

Rewrite C as $C = L'_0 R'_0$; namely, let $L_{n+1} = L'_0$ and $R_{n+1} = R'_0$. The FCS decryption algorithm is symmetrical to the FCS encryption algorithm, except that the subkeys are applied in the reverse order. In particular, the FCS decryption algorithm executes the following operations in round i, where $i = 1, 2, \cdots, n$:

$$L'_i = R'_{i-1}, \tag{2.3}$$

$$R'_i = L'_{i-1}, \oplus F(R'_{i-1}, K_{n-i+1}). \tag{2.4}$$

After n rounds, the ciphertext block $C = L'_0 R'_0$ is transformed to $L'_n R'_n$. Let $L'_{n+1} = R'_n$ and $R'_{n+1} = L'_n$, and let $L'_{n+1} R'_{n+1}$ be the output of FCS decryption.

We now show that the ciphertext block $C = L_{n+1} R_{n+1} = L'_0 R'_0$ is transformed back to the plaintext block $M = L_0 R_0$. Because the output of the FCS decryption is $L'_{n+1} R'_{n+1}$, it suffices to show the following equality:

$$L'_{n+1} R'_{n+1} = L_0 R_0.$$

Here is a proof. Note that $L'_{n+1} = R'_n$ and $R'_{n+1} = L'_n$, and so it suffices to show that $R'_n = L_0$ and $L'_n = R_0$. We can use mathematical induction to show that for any integer i with $0 \leq i \leq n$, we have

$$L'_i = R_{n-i}, \tag{2.5}$$

$$R'_i = L_{n-i}. \tag{2.6}$$

Let $i = n$, and we will get what we need to prove.

The mathematical induction proof is given as follows. We first note that $L'_0 = L_{n+1} = R_n$ and $R'_0 = R_{n+1} = L_n$. Thus, when $i = 0$, Equalities 2.5 and 2.6 hold.

Induction hypothesis: for any positive integer $i \leq n$, we have

$$L'_{i-1} = R_{n-i+1}, R'_{i-1} = L_{n-i+1}.$$

It follows from Equality 2.3, the induction hypothesis, and Equality 2.1 that

$$L'_i = R'_{i-1} = L_{n-i+1} = R_{n-i}.$$

Thus, Equality 2.5 is true.

From Equality 2.4 (the induction hypothesis), Equality 2.2, and Equality 2.1, we have

$$\begin{aligned}
R'_i &= L'_{i-1} \oplus F(R'_{i-1}, K_{n-i+1}) \\
&= R_{n-i+1} \oplus F(L_{n-i+1}, K_{n-i+1}) \\
&= [L_{n-i} \oplus F(R_{n-i}, K_{n-i+1})] \oplus F(R_{n-i}, K_{n-i+1}) \\
&= L_{n-i} \oplus [F(R_{n-i}, K_{n-i+1}) \oplus F(R_{n-i}, K_{n-i+1})] \\
&= L_{n-i} \oplus 0^w \\
&= L_{n-i}.
\end{aligned}$$

Thus, Equality 2.6 is true. This completes the proof of the correctness of the FCS decryption algorithm.

DES is an instance of FCS with $l = 32$. That is, the block size of DES is $\ell = 64$. The length of DES encryption keys is 56 bits. However, a DES encryption key is represented as a 64-bit binary string, where the $8i$th bit ($i = 1, 2, \cdots, 8$) is the parity bit of the seven bits immediately before it. The parity bit is used for error detection. Let K be an encryption key. DES first generates 16 subkeys from K, where each subkey has exactly 48 bits. There are $n = 16$ rounds of executions in DES.

2.2.2   DES Subkeys

Let $K = k_1 k_2 \cdots k_{64}$ be an encryption key of DES. To generate 16 subkeys, DES first removes each $8i$th bit ($i = 1, 2, \cdots, 8$) from K. For convenience, we still use K to denote the new string. DES then permutes the remaining 56 bits using the initial permutation IP_{key} as follows,

where bits are listed row wise:

$$IP_{key}(K) = \begin{matrix} k_{57} & k_{49} & k_{41} & k_{33} & k_{25} & k_{17} & k_9 & k_1 & k_{58} & k_{50} & k_{42} & k_{34} & k_{26} & k_{18} \\ k_{10} & k_2 & k_{59} & k_{51} & k_{43} & k_{35} & k_{27} & k_{19} & k_{11} & k_3 & k_{60} & k_{52} & k_{44} & k_{36} \\ k_{63} & k_{55} & k_{47} & k_{39} & k_{31} & k_{23} & k_{15} & k_7 & k_{62} & k_{54} & k_{46} & k_{38} & k_{30} & k_{22} \\ k_{14} & k_6 & k_{61} & k_{53} & k_{45} & k_{37} & k_{29} & k_{21} & k_{13} & k_5 & k_{28} & k_{20} & k_{12} & k_4 \end{matrix}$$

It is evident that $IP_{key}(K)$ permutes K in the following way: the indexes of the first 28 bits start from 57 such that each next index is equal to the current index minus 8 mod 65. The indexes of the next 24 bits start from 63 such that each next index is equal to the current index minus 8 mod 63. The indexes of the last four bits start from 28 such that each next index is equal to the current index minus 8.

The modular operation is a common operation when dealing with integers in a finite domain. Let m be a positive integer and a be a non-negative integer. Then "$a \bmod m$" is the remainder of dividing a by m. If $a < 0$ and its absolute value $|a| < m$, then $a \bmod m$ is equal to the smallest positive integer b such that $b - a = m$. For instance, $-5 \bmod 65 = 60$.

Let $X = x_1 x_2 \cdots x_{56}$ be a 56-bit binary string, where $x_i \in \{0, 1\}$. Let P_{key} be a compress permutation that takes X as input and produces a 48-bit string as output. P_{key} is defined as follows, where bits are listed row wise:

$$P_{key}(X) = \begin{matrix} x_{14} & x_{17} & x_{11} & x_{24} & x_1 & x_5 & x_3 & x_{28} & x_{15} & x_6 & x_{21} & x_{10} \\ x_{23} & x_{19} & x_{12} & x_4 & x_{26} & x_8 & x_{16} & x_7 & x_{27} & x_{20} & x_{13} & x_2 \\ x_{41} & x_{52} & x_{31} & x_{37} & x_{47} & x_{55} & x_{30} & x_{40} & x_{51} & x_{45} & x_{33} & x_{48} \\ x_{44} & x_{49} & x_{39} & x_{56} & x_{34} & x_{53} & x_{46} & x_{42} & x_{50} & x_{36} & x_{29} & x_{32} \end{matrix}$$

Let Y be a 28-bit binary string. Let $LS_{z(i)}(Y)$ denote the new string obtained by shifting Y circularly to the left $z(i)$ times, where $z(i)$ is defined as follows:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$z(i)$	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Rewrite $IP_{key}(K)$ as $U_0 V_0$, where both U_0 and V_0 are 28-bit binary strings. Then the ith subkey K_i is generated as follows, where $i = 1, 2, \cdots, 16$:

$$U_i = LS_{z(i)}(U_{i-1}),$$
$$V_i = LS_{z(i)}(V_{i-1}),$$
$$K_i = P_{key}(U_i V_i).$$

For instance, let

$$U_0 = 1001101001110110001010011010,$$
$$V_0 = 0110010110001001110101100101.$$

Then

$$U_1 = LS_{z(1)}(U_0) = LS_1(U_0) = 0011010011101100010100110101,$$
$$V_1 = LS_{z(1)}(V_0) = LS_1(V_0) = 1100101100010011101011001010.$$

Thus,

$$K_1 = P_{key}(U_1 V_1)$$
$$= P_{key}(001101001110110001010011010111001011000100111010110011010)$$
$$= 10110011010110011000011000001111011011000011001110.$$

2.2.3   DES Substitution Boxes

The substitution function F in DES is defined using eight special matrices. They are referred to as *substitution boxes* or *S-Boxes* in short. Each S-Box is a 4×16 matrix (see Table 2.1), where each row in each S-Box is a permutation of integers from 0 to 15. We label these S-Boxes as $S_1, S_2, \cdots, S_8$. For each r with $1 \leq r \leq 8$, write

$$S_r = [s_{ij}^{(r)}]_{4 \times 16}, i = 0, \cdots, 3, j = 0, \cdots, 15.$$

Let S be a function that takes a 48-bit string as input and produces a 32-bit binary string as output. In particular, let $Y = y_1 y_2 \cdots y_{48}$ be a 48-bit binary string, where $y_i \in \{0, 1\}$. We use $Y[i, j] \, (i < j)$ to denote the substring $y_i \cdots y_j$. Divide Y into eight 6-bit blocks:

$$Y = Y[1, 6]Y[7, 12]Y[13, 18]Y[19, 24]Y[25, 30]Y[31, 36]Y[37, 42]Y[43, 48].$$

For each 6-bit block $Y[6r - 5, 6r] \, (r = 1, 2, \cdots, 8)$, we use the rth S-Box to generate a 4-bit binary string as output, denoted by $S_r(Y[6r - 5, 6r])$, as follows:

Let $Y[6r - 5, 6r] = b_1 b_2 b_3 b_4 b_5 b_6$, where $b_q \in \{0, 1\}$ for $q = 1, \cdots, 6$. Let $i = b_1 b_6$ denote the binary representation for a row number and $j = b_2 b_3 b_4 b_5$ denote the binary representation for a column number. Then define $S_r(Y[6r - 5, 6r])$ to be the number in the 4-bit binary representation at row $i + 1$ and column $j + 1$ in matrix S_r; namely,

$$S_r(Y[6r - 5, 6r]) = s_{ij}^{(r)}.$$

For example, if $Y[7, 12] = 110010$, then $S_2(110010) = s_{10,1001}^{(2)} = s_{2,9}^{(2)} = 8$.

Let

$$S(Y) = S_1(Y[1, 6])S_2(Y[7, 12]) \cdots S_8(Y[43, 48]).$$

Then $S(Y)$ transforms the 48-bit input Y to a 32-bit output.

The constructions of the S-Boxes followed a clear set of criteria for the purpose of resisting possible attacks. They were also bound by the computing technologies available in the mid-1970s. For example, the reason why an S-Box has a 6-bit input and a 4-bit output is due to the chip technology available at that time. It took several months of computing time at that time to compute the S-Boxes that satisfy all the criteria. Because the set of criteria for the S-Boxes was not published and because NSA played a role in selecting DES, many people suspected that NSA had planted backdoors in these S-Boxes so that NSA can decipher DES-encrypted messages should they decide to do so. This of course was sheer speculation. In the 1990s, the set of criteria used by the DES design team was discovered by cryptanalysts not in the team. After this, IBM finally decided to publish the original set of criteria for constructing the S-Boxes.

Table 2.1   DES S-Boxes

S_1 :	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2 :	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3 :	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4 :	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5 :	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6 :	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7 :	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8 :	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

2.2.4   DES Encryption

DES implements its substitution function F using permutations, exclusive-OR, subkeys, and substitutions from the S-Boxes. In particular, for each 32-bit half block R_{i-1}, DES first uses a function called *expansion permutation*, denoted by *EP*, to expand it into a 48-bit string. It then XORs this string with a 48-bit subkey, takes the resulting 48-bit output as the input of function S to generate a 32-bit string, and permutes this string to generate a 32-bit string L_i.

2.2.4.1 Expansion Permutation

Let $U = u_1 u_2 \cdots u_{32} (u_i \in \{0, 1\})$ be a 32-bit binary string. The expansion permutation EP on U is defined as follows, where bits are listed row wise:

$$EP(U) = \begin{matrix} u_{32} & u_1 & u_2 & u_3 & u_4 & u_5 \\ u_4 & u_5 & u_6 & u_7 & u_8 & u_9 \\ u_8 & u_9 & u_{10} & u_{11} & u_{12} & u_{13} \\ u_{12} & u_{13} & u_{14} & u_{15} & u_{16} & u_{17} \\ u_{16} & u_{17} & u_{18} & u_{19} & u_{20} & u_{21} \\ u_{20} & u_{21} & u_{22} & u_{23} & u_{24} & u_{25} \\ u_{24} & u_{25} & u_{26} & u_{27} & u_{28} & u_{29} \\ u_{28} & u_{29} & u_{30} & u_{31} & u_{32} & u_1 \end{matrix}$$

We note that in $EP(U)$, the indexes of the four middle columns are $1, 2, \cdots, 32$; the indexes of the first column start from 32, where the next index is the current index plus 4 mod 32; and the indexes of the last column start from 5, where the next index is the current index plus 4 mod 32.

2.2.4.2 DES Substitution

Let $V = v_1 v_2 \cdots v_{32}$ be a 32-bit binary string. Permuting V using the following permutation P, where bits are listed row wise:

$$P(V) = \begin{matrix} v_{16} & v_7 & v_{20} & v_{21} & v_{29} & v_{12} & v_{28} & v_{17} & v_1 & v_{15} & v_{23} & v_{26} & v_5 & v_{18} & v_{31} & v_{10} \\ v_2 & v_8 & v_{24} & v_{14} & v_{32} & v_{27} & v_3 & v_9 & v_{19} & v_{13} & v_{30} & v_6 & v_{22} & v_{11} & v_4 & v_{25} \end{matrix}$$

DES defines its substitution function F as follows:

$$F(R_{i-1}, K_i) = P(S(EP(R_{i-1}) \oplus K_i)), \ i = 1, 2, \cdots, 16.$$

2.2.4.3 Encryption Steps

Let $A = a_1 a_2 \cdots a_{64} \ (a_i \in \{0, 1\})$ be a 64-bit binary string. Define a permutation σ as follows: it first reverses A as $a_{64} a_{63} \cdots a_1$. It then lists the prefix $a_{64} a_{63} \cdots a_{33}$ into four columns from right to left, where each column has exactly eight rows. It also lists the suffix $a_{32} a_{31} \cdots a_1$ into four columns from right to left, where each column has exactly eight rows, and inserts them alternately in the prefix columns. That is, according to the listing order, we have

4	8	3	7	2	6	1	5
a_{40}	a_8	a_{48}	a_{16}	a_{56}	a_{24}	a_{64}	a_{32}
a_{39}	a_7	a_{47}	a_{15}	a_{55}	a_{23}	a_{63}	a_{31}
a_{38}	a_6	a_{46}	a_{14}	a_{54}	a_{22}	a_{62}	a_{30}
a_{37}	a_5	a_{45}	a_{13}	a_{53}	a_{21}	a_{61}	a_{29}
a_{36}	a_4	a_{44}	a_{12}	a_{52}	a_{20}	a_{60}	a_{28}
a_{35}	a_3	a_{43}	a_{11}	a_{51}	a_{19}	a_{59}	a_{27}
a_{34}	a_2	a_{42}	a_{10}	a_{50}	a_{18}	a_{58}	a_{26}
a_{33}	a_1	a_{41}	a_9	a_{49}	a_{17}	a_{57}	a_{25}

The permutation enumerates the elements in this list row wise. Denote by σ^{-1} the inverse of σ.

Let $M = m_1 m_2 \cdots m_{64}$ ($m_i \in \{0, 1\}$) be a plaintext block. Define an initial permutation IP by $IP(M) = \sigma^{-1}(M)$. It is straightforward to verify that $IP(M)$ is equal to the following string, where each number i represents bit m_i ($1 \le i \le 64$) and bits are listed row wise:

$$IP(M) = \begin{array}{cccccccccccccccc} 58 & 50 & 42 & 34 & 26 & 18 & 10 & 2 & 60 & 52 & 44 & 36 & 28 & 20 & 12 & 4 \\ 62 & 54 & 46 & 38 & 30 & 22 & 14 & 6 & 64 & 56 & 48 & 40 & 32 & 24 & 16 & 8 \\ 57 & 49 & 41 & 33 & 25 & 17 & 9 & 1 & 59 & 51 & 43 & 35 & 27 & 19 & 11 & 3 \\ 61 & 53 & 45 & 37 & 29 & 21 & 13 & 5 & 63 & 55 & 47 & 39 & 31 & 23 & 15 & 7 \end{array}$$

It is easy to see that the indexes of the first two rows in $IP(M)$ start from 58, where the next index is equal to the current index minus 8 mod 66; and the indexes of the last two rows in $IP(M)$ start from 57, where the next index is equal to the current index minus 8 mod 66.

Let $C = c_1 c_2 \cdots c_{64}$ ($c_i \in \{0, 1\}$). Then $IP^{-1}(C) = \sigma(C)$ is the inverse of $IP(C)$, defined as follows, where each number i represents bit c_i ($1 \le i \le 64$) and bits are listed row wise:

$$IP^{-1}(C) = \begin{array}{cccccccccccccccc} 40 & 8 & 48 & 16 & 56 & 24 & 64 & 32 & 39 & 7 & 47 & 15 & 55 & 23 & 63 & 31 \\ 38 & 6 & 46 & 14 & 54 & 22 & 62 & 30 & 37 & 5 & 45 & 13 & 53 & 21 & 61 & 29 \\ 36 & 4 & 44 & 12 & 52 & 20 & 60 & 28 & 35 & 3 & 43 & 11 & 51 & 19 & 59 & 27 \\ 34 & 2 & 42 & 10 & 50 & 18 & 58 & 26 & 33 & 1 & 41 & 9 & 49 & 17 & 57 & 25 \end{array}$$

It is straightforward to verify that $IP \circ IP^{-1}(M) = IP^{-1} \circ IP(M) = M$. For example, let $C = IP(M)$. Because IP changes m_1 to m_{58} and IP^{-1} changes c_1 to c_{40}, where $c_1 = m_{58}$ and $c_{40} = m_1$, we know that $IP^{-1} \circ IP$ changes m_1 back to m_1.

Let M and K be, respectively, a 64-bit plaintext block and a 64-bit encryption key with added parity bits. Let $K_1, K_2, \cdots, K_{16}$ be the 16 subkeys generated from K as described in Section 2.2.2. The DES encryption steps are given as follows:

1. Rewrite $IP(M) = L_0 R_0$, where $|L_0| = |R_0| = 32$.
2. For $i = 1, 2, \cdots, 16$, execute the following operations in order:

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

3. Finally, let $C = IP^{-1}(R_{16} L_{16})$. (Note that the input of IP^{-1} is $R_{16} L_{16}$, not $L_{16} R_{16}$.)

2.2.5   DES Decryption and Correctness Proof

DES decryption is symmetrical to DES encryption, except that subkeys are applied in the reverse order. The DES decryption steps are given as follows:

1. Rewrite $IP(C) = L_0' R_0'$, where $|L_0| = |R_0| = 32$.
2. For $i = 1, 2, \cdots, 16$, execute the following operations in order

$$L_i' = R_{i-1}',$$
$$R_i' = L_{i-1}' \oplus F(R_{i-1}', K_{17-i}).$$

3. Finally, let $L_{17}' R_{17}' = IP^{-1}(R_{16}' L_{16}')$; we then obtain back the plaintext block M.

To prove the correctness of DES decryption, we need to show that

$$M = IP^{-1}(R'_{16}L'_{16}).$$

Because $L'_0 R'_0 = IP(C) = IP(IP^{-1}(R_{16}L_{16})) = R_{16}L_{16}$, we have $L'_0 = R_{16}$ and $R'_0 = L_{16}$. We note that except IP is used before round 1 starts and IP^{-1} after round 16, the rest of DES is a concrete implementation of FCS. It follows from FCS decryption that $L'_{16} = R_0$ and $R'_{16} = L_0$. Thus,

$$IP^{-1}(R'_{16}L'_{16}) = IP^{-1}(L_0R_0) = IP^{-1}(IP(M)) = M.$$

This completes the proof.

2.2.6   DES Security Strength

The security strength of DES depends on the number of rounds, the length of encryption key, and the construction of the substitution function. A substantial number of experiments have demonstrated that DES encryption provides good diffusion and confusion effects.

It can be shown that if the number of rounds in DES encryption is less than 16, then differential cryptanalysis can break DES encryption in a reasonable amount of time.

The length of a DES encryption key is 56 bits, which was sufficient to resist brute-force attacks in the 1970s to 1980s. However, the 56-bit key length was no longer secure in the late 1990s due to advancements of computer technologies and algorithms. For example, in 1999, the Electronic Frontier Foundation (EFF) based in the United States spent less than $250,000 to build a special-purpose supercomputer, named "DES Cracker," to crack DES encryptions. Working with Distributed.Net and a worldwide network of nearly 100,000 PCs on the Internet, DES Cracker broke in 22 hours the "DES Challenge III" encrypted message. The DES Challenges were a series of DES-encrypted messages posted by RSA Data Security in 1997. DES Challenge III was the last one to be broken. This indicated that the DES era was approaching its end.

Does this mean that all the manpower and resources spent over the years in developing hardware and software DES products were down the drain? It is true that DES encryption keys are too short to resist brute-force attacks, but DES has other good properties that have resisted many other attacks. Thus, it is reasonable to look for ways to effectively extend the DES encryption key length without changing the DES algorithms. Fortunately, it can be shown that DES is not a group. Therefore, applying DES multiple times is different from applying DES a single time. In other words, for any three 56-bit DES encryption keys K_1, K_2, K_3, we have $E_{K_2} \circ E_{K_1} \neq E_{K_3}$, where E_K represents DES encryption with key K. We note that this property may not be true in other encryption algorithms. For example, in XOR encryption, for any given encryption keys K_1 and K_2, let $K_3 = K_1 \oplus K_2$, then

$$E_{K_2}(E_{K_1}(M)) = (M \oplus K_1) \oplus K_2 = M \oplus K_3 = E_{K_3}(M),$$

where E represents XOR encryption.

2.3   Multiple DES

As discussed in Section 2.2.6, applying DES multiple times can effectively extend the length of encryption keys without modifying DES. Multiple DES can therefore be used to resist brute-force attacks. We use kDES to denote a multiple DES scheme of applying DES k times. By applying DES, it means applying either the encryption algorithm E or the decryption algorithm D.

2.3.1   Triple-DES with Two Keys

Triple-DES with two keys, denoted by 3DES/2, is the simplest and reasonably secure method against brute-force attacks. It extends the key length to 112 bits long. Let K_1 and K_2 be two 56-bit encryption keys and M a 64-bit plaintext block. The standard 3DES/2 encryption algorithm applies E_{K_1} on M to obtain $C_1 = E_{K_1}(M)$, then applies D_{K_2} on C_1 to obtain $C_2 = D_{K_2}(C_1)$, and finally applies E_{K_1} on C_2 to obtain $C = E_{K_1}(C_2)$. That is,

$$C = E_{K_1}(D_{K_2}(E_{K_1}(M))). \tag{2.7}$$

For convenience, we denote this scheme by $C = EDE_{K_1,K_2}(M)$.

The following is the 3DES/2 decryption algorithm:

$$M = D_{K_1}(E_{K_2}(D_{K_1}(C))). \tag{2.8}$$

For convenience, we denote this scheme by $M = DED_{K_1,K_2}(C)$.

We note that there are other combinations for 3DES with two keys, such as EEE_{K_1,K_2} or EED_{K_1,K_2}. Any of these combinations would serve the purpose. However, only the combination of EDE_{K_1,K_2} allows us to use 3DES/2 to decrypt ciphertext string produced by applying single DES with key K. This is done as follows: let $C = E_K(M)$ and let $K_1 = K_2 = K$. Then

$$DED_{K,K}(C) = D_K(E_K(M)) = M.$$

A major drawback of 3DES is that its software executions are not as efficient as one would like them to be.

2.3.2   2DES and 3DES/3

In addition to 3DES/2, we may also apply DES twice with two keys, denoted by 2DES/2. For simplicity, we use 2DES to denote 2DES/2. Let K_1 and K_2 be two DES encryption keys and M a 64-bit plaintext block. The standard 2DES encryption algorithm E and decryption algorithm D are described as follows:

$$C = E_{K_2}(E_{K_1}(M)),$$
$$M = D_{K_1}(D_{K_2}(C)).$$

However, 2DES is vulnerable to the *meet-in-the-middle attack* (see Section 2.3.3 for details). Thus, 2DES is considered nonsecure.

We may also apply DES thrice with three keys, denoted by 3DES/3. 3DES/3 has an effective key length of 168 bits. Let K_1, K_2, and K_3 be three encryption keys. The standard 3DES/3 encryption algorithm E and decryption algorithm D are described as follows:

$$C = E_{K_3}(D_{K_2}(E_{K_1}(M))), \tag{2.9}$$

$$M = D_{K_1}(E_{K_2}(D_{K_3}(C))). \tag{2.10}$$

2.3.3 Meet-in-the-Middle Attacks on 2DES

2DES is vulnerable to the meet-in-the-middle attacks. Suppose that the attacker has obtained two plaintext–ciphertext pairs (M_1, C_1) and (M_2, C_2), where

$$C_1 = E_{K_2}(E_{K_1}(M_1)), \quad C_2 = E_{K_2}(E_{K_1}(M_2)).$$

That is,

$$D_{K_2}C_1 = E_{K_1}(M_1), \quad D_{K_2}(C_2)) = E_{K_1}(M_2).$$

The attacker may then be able to identify, with probability close to 1, the encryption keys K_1 and K_2 with time complexity much smaller than 2^{112}. The attack can be carried out as follows:

List all 56-bit strings $U_0, U_1, \cdots, U_{2^{56}-1}$ and calculate, for each pair (U_i, U_j),

$$X_i = E_{U_i}(M_1), \quad Y_j = D_{U_j}(C_1).$$

Note that when $U_i = K_1$ and $U_j = K_2$, we have $X_i = Y_j$. Thus, for each pair (X_i, Y_j) with $X_i = Y_j$, it is possible that $(U_i, U_j) = (K_1, K_2)$. If there is only one such pair, then we have found the encryption keys K_1 and K_2. Otherwise, apply each pair (U_i, U_j) with $X_i = Y_j$ on (M_2, C_2) to obtain

$$X_i' = E_{U_i}(M_2), \quad Y_j' = D_{U_j}(C_2).$$

Again, we note that when $U_i = K_1$ and $U_j = K_2$, we have $X_i' = Y_j'$. Thus, if $X_i' = Y_j'$, then (U_i, U_j) is more likely to be the encryption key pair. Indeed, we can show that the possibility that there exist more than one such pair is very small.

Note that for any plaintext block M and any candidate (U_i, U_j) for the encryption key pair, the ciphertext block $C = E_{U_j}(E_{U_i}(M))$ is uniformly distributed (or close to being uniformly distributed). This is the property any good encryption algorithm should possess. Because $|U_i| = |U_j| = 56$, there are $2^{56} \cdot 2^{56} = 2^{112}$ pairs (X_i, Y_j). As $|X_i| = 64$, the expected number of pairs (U_i, U_j) that satisfy $E_{U_i}(M_1) = X_i = D_{U_j}(C_1)$ is or near

$$2^{112}/2^{64} = 2^{48}.$$

Likewise, the expected number of pairs (U_i, U_j) from these 2^{48} pairs that satisfy $E_{U_i}(M_2) = D_{U_j}(C_2)$ and $E_{U_i}(M_1) = D_{U_j}(C_1)$ is or near

$$2^{48}/2^{64} = 2^{-16}.$$

Thus, the possibility of finding (K_1, K_2) is or near $1 - 2^{-16}$.

The time complexity of executing this attack is in the order of

$$2(2^{56} + 2^{48}) < 2^{58}.$$

This is much smaller than 2^{112}.

2.4 Advanced Encryption Standard

Researchers have never stopped searching for better encryption algorithms that are more efficient, more secure, and more flexible. New encryption algorithms should be able to use longer keys and handle larger blocks. In some applications, people may also want to specify their own key length and block size. Thus, it would be desirable to have key length and block size as parameters.

A number of encryption algorithms have since been devised. Some of the early ones include Blowfish, CAST, GOST (the former Soviet Encryption Standard), International Data Encryption Algorithm (IDEA), LOKI, RC4, RC5, REDOC-II, REDOC-III, SAFER, and Skipjack. Most of these encryption algorithms are Feistel ciphers.

Realizing the urgent need to establish a new encryption standard, NIST launched in 1997 the advanced encryption standard competition for selecting a successor to DES. The following encryption algorithms were submitted for consideration: CAST-256, CRYPTON, DEAL, DFC, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, Rijndael, SAFER+, Serpent, and Twofish, where MARS, RC6, Rijndael, Serpent, and Twofish were selected semifinalists. In November 2001, NIST officially chose Rijndael to be the new AES. Rijndael was devised by Belgian cryptographers Joan Daemen and Vincent Rijmen.

2.4.1 AES Basic Structures

AES is a block cipher, but it is not a Feistel cipher. Its encryption and decryption, although similar, are not symmetrical. The basic computation unit in AES is a byte, rather than a bit as in DES. A byte is an 8-bit binary string. AES divides the plaintext string into 128-bit blocks. AES can use encryption keys of three different key lengths. An AES-encryption key can be 16-byte long, 24-byte long, or 32-byte long. Regardless of what key length is used, AES will generate and use 16-byte subkeys, also called *round keys*. AES can also run a different number of operation rounds. To generate a sufficient number of round keys, AES expands encryption keys depending on the number of rounds and the length of the encryption key specified by the users. Table 2.2 depicts the relations between key lengths, the number of rounds, and the length of expanded encryption keys, where a word is a binary string of length equal to four bytes.

Table 2.2 AES key lengths, the number of rounds, and the length of expanded encryption keys

Key length			Number of Rounds	Expansion key length		
Words	Bytes	Bits		Words	Bytes	Bits
4	16	128	10	44	176	1408
6	24	192	12	52	208	1664
8	32	256	14	60	240	1920

AES treats a 128-bit block as a sequence of 16 bytes and represents it as a 4×4 square matrix, where each element is a byte in the block. In particular, let $M = a_0 a_2 \cdots a_{15}$ be a plaintext block, where each a_i is a byte. Then AES rewrites M as Matrix 2.11:

$$M = \begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}. \tag{2.11}$$

We refer to a 4×4 matrix of bytes as a *state matrix*. AES encryption executes in each round (except the last round) the same sequence of simple operations on state matrices that transforms the plaintext block into a ciphertext block. These operations are *substitute-bytes*, *shift-rows*, *mix-columns*, and *add-round-key*.

1. The operation of substitute-bytes is a nonlinear operation based on a specially designed substitution box. The purpose of this operation is to resist differential cryptanalysis, linear cryptanalysis, and other mathematical attacks.
2. The operation of shift-rows is an elementary operation on state matrices. It is a linear operation. The purpose of this operation is to produce diffusion.
3. The operation of mix-columns is also an elementary operation on state matrices. Its purpose is the same as shift-rows.
4. The operation of add-round-key is a simple set of exclusive-OR operations on state matrices. It is a linear operation. The purpose of this operation is to produce confusion.

It is customary to use AES-128, AES-192, and AES-256 to denote, respectively, AES under 128-bit keys, 192-bit keys, and 256-bit keys. These three variants of AES all have the same encryption and decryption structures. They differ only on the number of rounds, where each round uses a different round key.
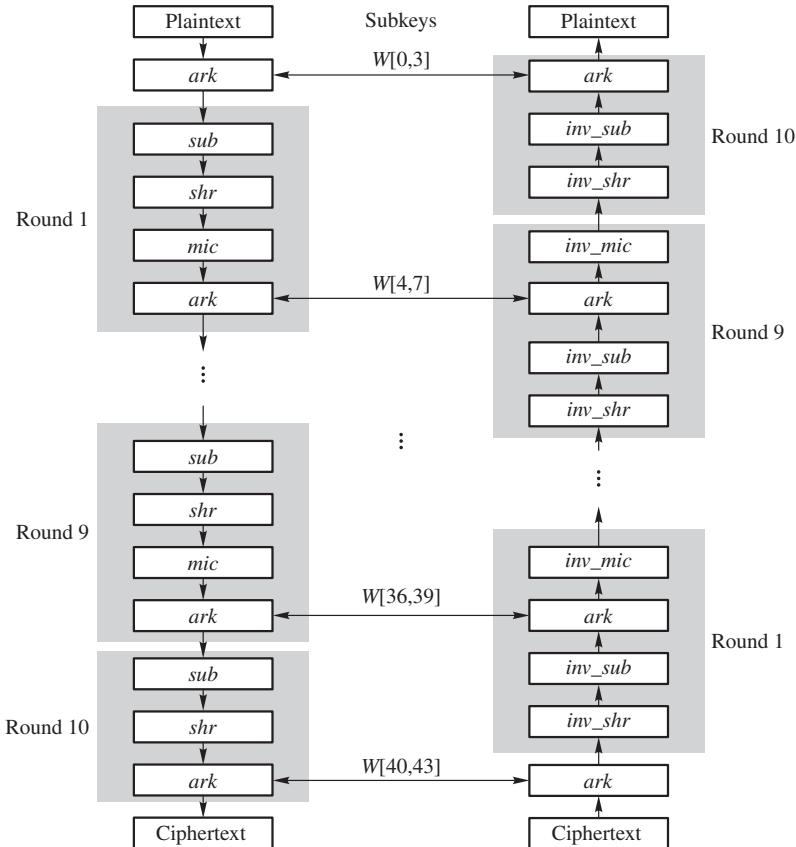
We describe AES using AES-128. AES-128 first expands the 128-bit key into an array $W[0, 43]$ of words. It then rewrites $W[0, 43]$ as a sequence of eleven 128-bit round keys $K_0, K_1, \cdots, K_{11}$. In other words,

$$K_i = W[4i, 4i + 3], \quad i = 0, 1, \cdots, 10,$$

where $W[4i, 4i + 3] = W[4i]W[4i + 1]W[4i + 2]W[4i + 3]$.

For convenience, we use *sub* to denote the operation of substitute-bytes, *shr* the operation of shift-rows, *mic* the operation of mix-columns, and *ark* the operation of add-round-key. Denote by *inv_sub* (or sub^{-1}) the inverse operation of *sub*, *inv_shr* (or shr^{-1}) the inverse operation of *shr*, and *inv_mic* (or mic^{-1}) the inverse operation of *mic*. Figure 2.2 depicts the AES-128 encryption and decryption block diagram.

In the following several subsections, we describe AES S-Boxes, the subkey generation algorithm, the operations of *ark*, *sub*, *shr*, *mic*, and their inverse operations. We then introduce the Galois field $GF(2^8)$ and show how S-Boxes are constructed. Skipping the last two topics will not affect the understanding of AES encryption and decryption operations.

Figure 2.2 AES-128 encryption and decryption diagram

2.4.2 AES S-Boxes

AES uses only one S-Box. It is used to generate subkeys and define the operation of substitute bytes. The AES S-Box is a 16×16 matrix of bytes, which is defined on the basis of the multiplication operation of the Galois field $GF(2^8)$. Unlike the S-Boxes used in DES, the AES S-Box is a permutation of all 256 bytes. Its reverse permutation is called the *reverse S-Box*.

We only present the S-Box and the reverse S-Box in this subsection, where each element and index is represented by two hexadecimal digits. We describe how they are constructed in Section 2.4.11. We use

$$\boldsymbol{S} = [s_{ij}]_{16 \times 16}$$

to denote the AES S-Box and

$$\boldsymbol{S}^{-1} = [s'_{ij}]_{16 \times 16}$$

Table 2.3   The S-Box of AES

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 2.4   The reverse S-Box of AES

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

to denote its reverse. The S-Box is given in Table 2.3, and the reverse S-Box is given in Table 2.4.

Let $w = b_0 \cdots b_7$ be a byte, where each b_i is a bit. Define a byte-substitution function S as follows: let $i = b_0 b_1 b_2 b_3$ denote the binary representation of the row index and $j = b_4 b_5 b_6 b_7$

denote the binary representation of a column index of s_{ij} in the S-Box. Then

$$S(w) = s_{ij}, \tag{2.12}$$

$$S^{-1}(w) = s'_{ij}. \tag{2.13}$$

That is, $S(w)$ is the element on the intersection of the $(i+1)$th row and the $(j+1)$th column in the S-Box S. Likewise, $S^{-1}(w)$ is the element on the intersection of the $(i+1)$th row and the $(j+1)$th column in the inverse S-Box S^{-1}.

For example, let $w = \mathsf{b8}$, then $S(w) = s_{\mathsf{b},8} = \mathsf{6c}$, and $S^{-1}(\mathsf{6c}) = s'_{\mathsf{6,c}} = \mathsf{b8}$.

It is straightforward to see from the S-Box S and its reverse S^{-1} that, for any 8-bit string w, we have

$$S(S^{-1}(w)) = w \text{ and } S^{-1}(S(w)) = w.$$

2.4.3 AES-128 Round Keys

Let $K = K[0,31]K[32,63]K[64,95]K[96,127]$ be a 4-word encryption key, where each $K[32i, 32i+31]$ is a 32-bit binary string, $i = 0, 1, 2, 3$. AES expands K into a 44-word array $W[0,43]$. We first define a byte transformation function $\mathcal{M}$ as follows:

$$\mathcal{M}(b_7b_6b_5b_4b_3b_2b_1b_0) = \begin{cases} b_6b_5b_4b_3b_2b_1b_00, & \text{if } b_7 = 0, \\ b_6b_5b_4b_3b_2b_1b_00 \oplus 00011011, & \text{if } b_7 = 1, \end{cases} \tag{2.14}$$

where each b_i is a bit. We see in Section 2.4.10 that $\mathcal{M}$ represents a multiplication operation of 00000010 and $b_7b_6 \cdots b_0$ over Galois field $GF(2^8)$.

For example,

$$\mathcal{M}(\mathsf{db}) = \mathcal{M}(11011011)$$

$$= 10110110 \oplus 00011011$$

$$= 10101101$$

$$= \mathsf{ad}.$$

Let j be a non-negative number. Define $m(j)$ as follows:

$$m(j) = \begin{cases} 00000001, & \text{if } j = 0, \\ 00000010, & \text{if } j = 1, \\ \mathcal{M}(m(j-1)), & \text{if } j > 1. \end{cases} \tag{2.15}$$

We see in Section 2.4.10 that function $m(j)$ represents the result of multiplying the $GF(2^8)$ element 00000010 by itself for $j-1$ times.

We now define a word-substitution function T that transforms a 32-bit string to a 32-bit string on the basis of parameter j and the AES S-Box. Let $w = w_1w_2w_3w_4$, where each w_i is a byte. Let j be a positive integer. Let

$$T(w, j) = [(S(w_2) \oplus m(j-1)]S(w_3)S(w_4)S(w_1),$$

where S is the byte-substitution function defined by Equality 2.12.

We now expand $K = K_0 K_1 \cdots K_{15}$ to obtain $W[0, 43]$ as follows:

$$W[0] = K[0, 31],$$
$$W[1] = K[32, 63],$$
$$W[2] = K[64, 95],$$
$$W[3] = K[96, 127],$$
$$W[i] = \begin{cases} W[i-4] \oplus T(W[i-1], i/4), & \text{if } i \text{ is divisible by 4,} \\ W[i-4] \oplus W[i-1], & \text{otherwise,} \end{cases}$$
$$i = 4, \cdots, 43.$$

2.4.4 Add Round Keys

Let $K_i = W[4i, 4i+3] = W[4i]W[4i+1]W[4i+2]W[4i+3]$ be an AES-128 round key, where $i = 0, \cdots, 10$. Rewrite K_i as a 4×4 matrix of bytes:

$$\boldsymbol{K}_i = \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix},$$

where each element is a byte and $W[4i+j] = k_{0,j}k_{1,j}k_{2,j}k_{3,j}, \ \ j = 0, 1, 2, 3$.

In what follows we use

$$\boldsymbol{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

to represent the current state matrix. Initially, $\boldsymbol{A} = \boldsymbol{M}$ (see Matrix 2.11). The add-round-key operation *ark* is defined as follows:

$$ark(\boldsymbol{A}, \boldsymbol{K}_i) = \boldsymbol{A} \oplus \boldsymbol{K}_i = \begin{bmatrix} a_{0,0} \oplus k_{0,0} & a_{0,1} \oplus k_{0,1} & a_{0,2} \oplus k_{0,2} & a_{0,3} \oplus k_{0,3} \\ a_{1,0} \oplus k_{1,0} & a_{1,1} \oplus k_{1,1} & a_{1,2} \oplus k_{1,2} & a_{1,3} \oplus k_{1,3} \\ a_{2,0} \oplus k_{2,0} & a_{2,1} \oplus k_{2,1} & a_{2,2} \oplus k_{2,2} & a_{2,3} \oplus k_{2,3} \\ a_{3,0} \oplus k_{3,0} & a_{3,1} \oplus k_{3,1} & a_{3,2} \oplus k_{3,2} & a_{3,3} \oplus k_{3,3} \end{bmatrix}.$$

The inverse operation ark^{-1} is the same as *ark*. That is, $ark^{-1}(\boldsymbol{A}, \boldsymbol{K}_i) = ark(\boldsymbol{A}, \boldsymbol{K}_i)$. It is straightforward to verify that

$$ark(ark^{-1}(\boldsymbol{A}, \boldsymbol{K}_i), \boldsymbol{K}_i) = ark^{-1}(ark(\boldsymbol{A}, \boldsymbol{K}_i), \boldsymbol{K}_i) = \boldsymbol{A}.$$

2.4.5   Substitute-Bytes

The substitute-bytes operation *sub* is defined as follows:

$$sub(\boldsymbol{A}) = [S(a_{ij})]_{4\times4} = \begin{bmatrix} S(a_{0,0}) & S(a_{0,1}) & S(a_{0,2}) & S(a_{0,3}) \\ S(a_{1,0}) & S(a_{1,1}) & S(a_{1,2}) & S(a_{1,3}) \\ S(a_{2,0}) & S(a_{2,1}) & S(a_{2,2}) & S(a_{2,3}) \\ S(a_{3,0}) & S(a_{3,1}) & S(a_{3,2}) & S(a_{3,3}) \end{bmatrix},$$

where S is the byte-substitution function defined in Section 2.4.2 (see formula 2.12).

It is straightforward to verify that the inverse operation sub^{-1} is given by

$$sub^{-1}(\boldsymbol{A}) = [S^{-1}(a_{ij})]_{4\times4},$$

where S^{-1} is defined in Section 2.4.2 (see formula 2.13). This is because for any byte w, we have $S(S^{-1}(w)) = S^{-1}(S(w)) = w$, which implies that

$$sub(sub^{-1}(\boldsymbol{A})) = sub^{-1}(sub(\boldsymbol{A})) = \boldsymbol{A}.$$

2.4.6   Shift-Rows

The shift-row operation *shr* performs a left-circular-shift $i-1$ times on the ith row in the state matrix $\boldsymbol{A}$, where $i = 1, 2, 3, 4$. Its inverse shr^{-1} performs a right-circular-shift $i-1$ times on the ith row. That is,

$$shr(\boldsymbol{A}) = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \end{bmatrix}, \quad shr^{-1}(\boldsymbol{A}) = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,3} & a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,0} \end{bmatrix}.$$

It is straightforward to verify that

$$shr(shr^{-1}(\boldsymbol{A})) = shr^{-1}(shr(\boldsymbol{A})) = \boldsymbol{A}.$$

2.4.7   Mix-Columns

The mix-columns operation *mic* is defined as follows:

$$mic(\boldsymbol{A}) = [a'_{ij}]_{4\times4},$$

where each element in $mic(\boldsymbol{A})$ is determined by the following operations ($j = 0, 1, 2, 3$) :

$$a'_{0,j} = \mathcal{M}(a_{0,j}) \oplus [\mathcal{M}(a_{1,j}) \oplus a_{1,j}] \oplus a_{2,j} \oplus a_{3,j},$$
$$a'_{1,j} = a_{0,j} \oplus \mathcal{M}(a_{1,j}) \oplus [\mathcal{M}(a_{2,j}) \oplus a_{2,j}] \oplus a_{3,j},$$
$$a'_{2,j} = a_{0,j} \oplus a_{1,j} \oplus \mathcal{M}(a_{2,j}) \oplus [\mathcal{M}(a_{3,j}) \oplus a_{3,j}],$$
$$a'_{3,j} = [\mathcal{M}(a_{0,j}) \oplus a_{0,j}] \oplus a_{1,j} \oplus a_{2,j} \oplus \mathcal{M}(a_{3,j}).$$

For example, let

$$A = \begin{bmatrix} \text{db} & \text{2d} & \text{f2} & \text{d4} \\ \text{13} & \text{26} & \text{0a} & \text{d4} \\ \text{53} & \text{31} & \text{22} & \text{d4} \\ \text{45} & \text{4c} & \text{5c} & \text{d5} \end{bmatrix}, \quad \text{then } mic(A) = \begin{bmatrix} \text{8e} & \text{4d} & \text{9f} & \text{d5} \\ \text{4d} & \text{7e} & \text{dc} & \text{d5} \\ \text{a1} & \text{bd} & \text{58} & \text{d7} \\ \text{bc} & \text{f8} & \text{9d} & \text{d6} \end{bmatrix}. \tag{2.16}$$

We verify $a'_{0,0}$ as follows:

$$a'_{0,0} = \mathcal{M}(\text{db}) \oplus [\mathcal{M}(\text{13}) \oplus \text{13}] \oplus \text{53} \oplus \text{45}$$

$$= 10101101 \oplus [00100110 \oplus 00010011] \oplus 01010011 \oplus 01000101$$

$$= 10001110$$

$$= \text{8e}.$$

The reader is asked to verify the rest of the elements.

Let w be a byte and i a positive integer. Define

$$\mathcal{M}^i(w) = \mathcal{M}(\mathcal{M}^{i-1}(w)) \ (i > 1), \quad \mathcal{M}^1(w) = \mathcal{M}(w).$$

Let

$$\mathcal{M}_1(w) = \mathcal{M}^3(w) \oplus \mathcal{M}^2(w) \oplus \mathcal{M}(w), \tag{2.17}$$

$$\mathcal{M}_2(w) = \mathcal{M}^3(w) \oplus \mathcal{M}(w) \oplus w, \tag{2.18}$$

$$\mathcal{M}_3(w) = \mathcal{M}^3(w) \oplus \mathcal{M}^2(w) \oplus w, \tag{2.19}$$

$$\mathcal{M}_4(w) = \mathcal{M}^3(w) \oplus w. \tag{2.20}$$

The inverse operation of mic^{-1} is defined by

$$mic^{-1}(A) = [a''_{ij}]_{4 \times 4},$$

where each column in $mic^{-1}(A)$ is defined as follows:

$$a''_{0,j} = \mathcal{M}_1(a_{0,j}) \oplus \mathcal{M}_2(a_{1,j}) \oplus \mathcal{M}_3(a_{2,j}) \oplus \mathcal{M}_4(a_{3,j}), \tag{2.21}$$

$$a''_{1,j} = \mathcal{M}_4(a_{0,j}) \oplus \mathcal{M}_1(a_{1,j}) \oplus \mathcal{M}_2(a_{2,j}) \oplus \mathcal{M}_3(a_{3,j}), \tag{2.22}$$

$$a''_{2,j} = \mathcal{M}_3(a_{0,j}) \oplus \mathcal{M}_4(a_{1,j}) \oplus \mathcal{M}_1(a_{2,j}) \oplus \mathcal{M}_2(a_{3,j}), \tag{2.23}$$

$$a''_{3,j} = \mathcal{M}_2(a_{0,j}) \oplus \mathcal{M}_3(a_{1,j}) \oplus \mathcal{M}_4(a_{2,j}) \oplus \mathcal{M}_1(a_{3,j}). \tag{2.24}$$

We show in Section 2.4.10 that for any state matrix A the following relation holds:

$$mic(mic^{-1}(A)) = mic^{-1}(mic(A)) = A. \tag{2.25}$$

2.4.8 AES-128 Encryption

Let A_i $(i = 0, 1, \cdots, 11)$ be a sequence of state matrices, where A_0 is the initial state matrix M (i.e., Matrix 2.11), A_i $(i = 1, 2, \cdots, 10)$ represents the input state matrix at round i, and A_{11}

is the ciphertext block C, which is in the form of state matrix. Given below are the encryption steps of AES-128:

$$A_1 = ark(A_0, K_0),$$
$$A_{i+1} = ark(mic(shr(sub(A_i))), K_i), \quad i = 1, 2, \cdots, 9,$$
$$A_{11} = ark(shr(sub(A_{10})), K_{10}).$$

2.4.9 AES-128 Decryption and Correctness Proof

Let C_i $(i = 0, 1, \cdots, 11)$ be a sequence of state matrices, where C_0 is the ciphertext block $C = A_{11}, C_i$ $(i = 1, 2, \cdots, 10)$ represents the input state matrix at round i, and C_{11} is the output state matrix at round 10. The following are the decryption steps of AES-128:

$$C_1 = ark(C_0, K_{10}),$$
$$C_{i+1} = mic^{-1}(ark(sub^{-1}(shr^{-1}(C_i)), K_{10-i})), \quad i = 1, 2, \cdots, 9,$$
$$C_{11} = ark(sub^{-1}(shr^{-1}(C_{10})), K_0).$$

We now show that $C_{11} = A_0$. We first show the following equality using mathematical induction:

$$C_i = shr(sub(A_{11-i})), \quad i = 1, 2, \cdots, 10. \tag{2.26}$$

A proof of Equality 2.26 goes as follows. When $i = 1$, we have

$$\begin{aligned}
C_1 &= ark(A_{11}, K_{10}) \\
&= A_{11} \oplus K_{10} \\
&= ark(shr((sub(A_{10})), K_{10}) \oplus K_{10} \\
&= shr((sub(A_{10})) \oplus K_{10}) \oplus K_{10} \\
&= shr(sub(A_{10})).
\end{aligned}$$

Thus, Equality 2.26 holds. Assume that Equality 2.26 holds for $1 \leq i < 10$. We have

$$\begin{aligned}
C_{i+1} &= mic^{-1}(ark(sub^{-1}(shr^{-1}(C_i)), K_{10-i})) \\
&= mic^{-1}(sub^{-1}(shr^{-1}(shr(sub(A_{11-i}))))) \oplus K_{10-i}) \\
&= mic^{-1}(A_{11-i} \oplus K_{10-i}) \\
&= mic^{-1}(ark(mic(shr(sub(A_{10-i}))), K_{10-i}) \oplus K_{10-i}) \\
&= mic^{-1}([mic(shr(sub(A_{10-i}))) \oplus K_{10-i}] \oplus K_{10-i}) \\
&= shr(sub(A_{10-i})) \\
&= shr(sub(A_{11-(i+1)})).
\end{aligned}$$

Thus, Equality 2.26 is true.

Finally, we have

$$
\begin{aligned}
\boldsymbol{C}_{11} &= ark(sub^{-1}(shr^{-1}(\boldsymbol{C}_{10})), \boldsymbol{K}_0) \\
&= sub^{-1}(shr^{-1}(shr(sub(\boldsymbol{A}_1)))) \oplus \boldsymbol{K}_0 \\
&= \boldsymbol{A}_1 \oplus \boldsymbol{K}_0 \\
&= (\boldsymbol{A}_0 \oplus \boldsymbol{K}_0) \oplus \boldsymbol{K}_0 \\
&= \boldsymbol{A}_0.
\end{aligned}
$$

This completes the correctness proof of AES-128 decryption.

2.4.10   Galois Fields

In addition to the shift-rows operation, the basic operations of AES are based on the XOR operation and a special multiplication operation on 8-bit strings. These two operations and the set of all 8-bit strings form a finite field.

A field is an algebraic structure that consists of a set F and two operations on elements in F. These two operations are addition and multiplication, denoted by $+$ and $\times$, respectively, which satisfy the following conditions:

1. **Closure:** $(\forall a, b \in F)[a + b \in F \text{ and } a \times b \in F]$.
2. **Associativity:** $\forall a, \ b, \ c \in F$:

$$
\begin{aligned}
a + (b + c) &= (a + b) + c, \\
a \times (b \times c) &= (a \times b) \times c.
\end{aligned}
$$

3. **Distributivity:** $\forall a, \ b, \ c \in F$:

$$
\begin{aligned}
a \times (b + c) &= (a \times b) + (a \times c), \\
(a + b) \times c &= (a \times c) + (b \times c).
\end{aligned}
$$

4. **Unit element:** There are elements $e_0, \ e_1 \in F$, where $e_0 \neq e_1$, such that $\forall a \in F$:

$$
\begin{aligned}
a + e_0 &= e_0 + a = a, \\
a \times e_1 &= e_1 \times a = a.
\end{aligned}
$$

   The element e_0 is called the unit element of the addition operation and e_1 the unit element of the multiplication operation.
5. **Inverse:**

$$
\begin{aligned}
\forall a \in F : \ &(\exists a' \in F)[a + a' = a' + a = e_0], \\
\forall a \in F - \{e_0\} : \ &(\exists a'' \in F)[a \times a'' = a'' \times a = e_1].
\end{aligned}
$$

   Elements $a', \ a''$ are called, respectively, the additive inverse (denoted by $-a$) and the multiplicative inverse (denoted by a^{-1}.)
6. **Commutativity:** $(\forall a, b \in F)[a + b = b + a \text{ and } a \times b = b \times a]$.
7. **Nonzero divisor:** If $a, \ b \in F$ and $a \times b = e_0$, then $a = e_0$ or $b = e_0$.

We use $(F, +, \times)$ to denote a field. It is called a finite field if F is a finite set and an infinite field otherwise. The field of real numbers, for example, is the set of all real numbers with the ordinary addition and multiplication operations.

Finite fields have a nice property; namely, any finite field consists of exactly p^n elements for some prime number p and integer $n \geq 2$. A prime number is a positive integer that is divisible only by 1 and by itself. Elements in a finite field of size p^n can be uniquely represented by polynomials of degree $n - 1$ with coefficients in the set $\{0, 1, \cdots, p - 1\}$. A finite field with its elements written in this form is called a *Galois field*, denoted by $GF(p^n)$. Each element in $GF(p^n)$ is represented by the following polynomial

$$b_{n-1}x^{n-1} + \cdots + b_1 x + b_0,$$

denoted in short by $b_{n-1} \cdots b_1 b_0$, where each coefficient

$$b_i \in \{0, 1, \cdots, p - 1\}.$$

The addition operation over $GF(p^n)$ is addition modulo p of coefficients for terms of the same degree. The multiplication operation first multiplies two polynomials in the normal way. If the degree of the resulting polynomial is greater than $n - 1$, then divide it by a fixed *irreducible* polynomial of degree n, and take the remainder as the result of the multiplication. A polynomial is irreducible if it cannot be expressed as a product of two polynomials whose degrees are at least 1.

Modern electronic computers operate on binary operations using memory with bytes as basic units. Thus, choosing $GF(2^8)$ to form the basic operation space for encryption algorithms becomes natural. $GF(2^8)$ consists of all 8-bit binary strings as elements, where each element $b_7 \cdots b_1 b_0$ represents the following polynomial:

$$f(x) = b_7 x^7 + \cdots + b_1 x + b_0,$$

where the addition operation "+" is the exclusive-OR operation $\oplus$, and we use them interchangeably when there is no confusion. Thus, the inverse element of any element b is $-b = b$. We use $\otimes$ to denote the multiplication operation on $GF(2^8)$. The definition of the multiplication operation depends on the chosen irreducible polynomial. AES chooses the following irreducible polynomial:

$$r(x) = x^8 + x^4 + x^3 + x + 1.$$

This irreducible polynomial makes multiplication simple. We use $p(x) \bmod r(x)$ to denote the remainder polynomial of dividing $p(x)$ by $r(x)$. It is straightforward to verify that

$$x^8 \bmod r(x) = x^8 - r(x) = x^4 + x^3 + x + 1.$$

Hence, we have

$$x \otimes f(x) = (b_7 x^8 + b_6 x^7 + \cdots + b_0 x) \bmod r(x)$$

$$= \begin{cases} b_6 x^7 + \cdots + b_0 x, & \text{if } \quad b_7 = 0, \\ (x^4 + x^3 + x + 1) + (b_6 x^7 + \cdots + b_0 x), & \text{if } \quad b_7 = 1. \end{cases}$$

Denoting this formula using binary strings and XOR, we have

$$00000010 \otimes b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 = \begin{cases} b_6 b_5 b_4 b_3 b_2 b_1 b_0 0, & \text{if } b_7 = 0, \\ b_6 b_5 b_4 b_3 b_2 b_1 b_0 0 \oplus 00011011, & \text{if } b_7 = 1. \end{cases}$$

This is the definition of function $\mathcal{M}$ defined in Section 2.4.3 (see Formula 2.14). That is,

$$\mathcal{M}(b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0) = 00000010 \otimes b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 = 02 \otimes b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0.$$

The function $m(j)$ defined in Section 2.4.3 is the result of multiplying 00000010 by itself for $j - 1$ times. That is,

$$m(j) = 00000010 \otimes \cdots \otimes 00000010,$$

where the number of $\otimes$ is $j - 1$.

We now verify Equality 2.25. We first note that $mic(\boldsymbol{A})$ and $mic^{-1}(\boldsymbol{A})$ are the products of the following matrix multiplications:

$$mic(\boldsymbol{A}) = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \otimes \boldsymbol{A}, \tag{2.27}$$

$$mic^{-1}(\boldsymbol{A}) = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \otimes \boldsymbol{A}. \tag{2.28}$$

The matrix multiplication in this case follows the standard rule, with $\oplus$ being the addition operation and $\otimes$ the multiplication operation. In particular, we can verify Equality 2.27 from Equalities 2.17–2.20 by noting that

$$\mathcal{M}(w) \oplus w = (02 \otimes w) \oplus w = (02 \oplus 01) \otimes w = 03 \otimes w.$$

Likewise, we can verify Equality 2.28 from Equalities 2.21–2.24 by noting that

$$\mathcal{M}^i(w) = \mathcal{M}(\mathcal{M}^{i-1}(w)) = x^i \otimes w.$$

Hence,

$$\mathcal{M}_1(w) = (x^3 + x^2 + x) \otimes w = 0e \otimes w,$$

$$\mathcal{M}_2(w) = (x^3 + x + 1) \otimes w = 0b \otimes w,$$

$$\mathcal{M}_3(w) = (x^3 + x^2 + 1) \otimes w = 0d \otimes w,$$

$$\mathcal{M}_4(w) = (x^3 + 1) \otimes w = 09 \otimes w.$$

We can therefore show that $mic(\boldsymbol{A}) \otimes mic^{-1}(\boldsymbol{A}) = mic^{-1}(\boldsymbol{A}) \otimes mic(\boldsymbol{A}) = \boldsymbol{I}_4$ by verifying the following equalities, where $\boldsymbol{I}_4$ is the identity matrix of size 4:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \otimes \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} = \begin{bmatrix} 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \\ 00 & 00 & 00 & 01 \end{bmatrix} = \boldsymbol{I}_4, \tag{2.29}$$

$$\begin{bmatrix} \mathsf{0e} & \mathsf{0b} & \mathsf{0d} & \mathsf{09} \\ \mathsf{09} & \mathsf{0e} & \mathsf{0b} & \mathsf{0d} \\ \mathsf{0d} & \mathsf{09} & \mathsf{0e} & \mathsf{0b} \\ \mathsf{0b} & \mathsf{0d} & \mathsf{09} & \mathsf{0e} \end{bmatrix} \otimes \begin{bmatrix} \mathsf{02} & \mathsf{03} & \mathsf{01} & \mathsf{01} \\ \mathsf{01} & \mathsf{02} & \mathsf{03} & \mathsf{01} \\ \mathsf{01} & \mathsf{01} & \mathsf{02} & \mathsf{03} \\ \mathsf{03} & \mathsf{01} & \mathsf{01} & \mathsf{02} \end{bmatrix} = \begin{bmatrix} \mathsf{01} & \mathsf{00} & \mathsf{00} & \mathsf{00} \\ \mathsf{00} & \mathsf{01} & \mathsf{00} & \mathsf{00} \\ \mathsf{00} & \mathsf{00} & \mathsf{01} & \mathsf{00} \\ \mathsf{00} & \mathsf{00} & \mathsf{00} & \mathsf{01} \end{bmatrix} = \boldsymbol{I}_4. \tag{2.30}$$

For example, the first element at the right-hand side in Equality 2.29 is

$$(\mathsf{02} \otimes \mathsf{0e}) \oplus (\mathsf{03} \otimes \mathsf{09}) \oplus (\mathsf{01} \otimes \mathsf{0d}) \oplus (\mathsf{01} \otimes \mathsf{0b})$$

$$= \mathcal{M}(\mathsf{0e}) \oplus (\mathcal{M}(\mathsf{09}) \oplus \mathsf{09}) \oplus \mathsf{0d} \oplus \mathsf{0b}$$

$$= \mathsf{1c} \oplus (\mathsf{12} \oplus \mathsf{09}) \oplus \mathsf{0d} \oplus \mathsf{0b}$$

$$= \mathsf{01}.$$

2.4.11   Construction of the AES S-Box and Its Inverse

The S-Box S used by AES is constructed as follows:

1. Initially, S is a 16×16 matrix of all 8-bit strings in the lexicographical order. That is, its first row is vector $(\mathsf{00}, \mathsf{01}, \cdots, \mathsf{0f})$, its second row is vector $(\mathsf{10}, \mathsf{11}, \cdots, \mathsf{1f})$, and so on, and its last row is vector $(\mathsf{f0}, \mathsf{f1}, \cdots, \mathsf{ff})$.
2. Keep the first two elements in S (i.e., $\mathsf{00}$ and $\mathsf{01}$) unchanged, and replace any other element w with its inverse w^{-1}.
    For example, as $\mathsf{02}$'s multiplication inverse is $\mathsf{8d}$, the element $\mathsf{02}$ is replaced with $\mathsf{8d}$.
3. Replace each element $b_7 b_6 \cdots b_0$ with $b_7' b_6' \cdots b_0'$, where each bit b_i' ($i = 0, 1, \cdots, 7$) is determined by

$$b_i' = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i,$$

and $c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0 = 01100011$.
    For example, from Step 2 we know that the element $s_{0,2}$ at the intersection of the first row and the third column is $\mathsf{8d} = 10001101 = b_7 \cdots b_1 b_0$. Thus,

$$b_7' = b_7 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \oplus c_7 = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 = 0,$$

$$b_6' = b_6 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus c_6 = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 1,$$

$$b_5' = b_5 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus c_5 = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1,$$

$$b_4' = b_4 \oplus b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus c_4 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1,$$

$$b_3' = b_3 \oplus b_7 \oplus b_0 \oplus b_1 \oplus b_2 \oplus c_3 = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0,$$

$$b_2' = b_2 \oplus b_6 \oplus b_7 \oplus b_0 \oplus b_1 \oplus c_2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 1,$$

$$b_1' = b_1 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_0 \oplus c_1 = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1,$$

$$b_0' = b_0 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_1 \oplus c_0 = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 1.$$

Hence, $s_{0,2} = 01110111 = \mathsf{77}$.

It is required that each element $s'_{i,j}$ in the inverse S-Box $\boldsymbol{S}^{-1}$ must satisfy the following relations:

$$s'_{i,j} = uv, \quad s_{u,v} = ij.$$

Element $s'_{i,j}$, where $i, j \in \{0, 1, \cdots, f\}$, can be calculated as follows: first find the element ij in S. Let $s_{u,v} = ij$. Then let $s'_{i,j} = uv$.

For example, as $s_{6,a} = 02$, we have $s'_{0,2} = 6a$.

2.4.12 AES Security Strength

AES is designed to resist differential cryptanalysis and linear cryptanalysis. It uses 128-bit (or longer) encryption keys to resist brute-force attacks. It modifies each element in the current state matrix in each round, and so it will achieve good diffusion and confusion effects after a few rounds of execution. AES is believed to be superior to DES.

In June 2003, the U.S. government decided that classified government information should be encrypted using AES-128, and for TOP SECRET information, AES-192 or AES-256 must be used. The New European Schemes for Signatures, Integrity, and Encryption (NESSIE) also supported the use of AES. In June 2004, the Institute of Electrical and Electronics Engineers (IEEE) adopted AES in its 802.11i wireless security standard. IEEE 802.11i is also known as Wireless-Fidelity Protected Access 2 (Wi-Fi WPA 2). Today, AES has been used in almost all network security protocols and software products.

No methods have been found that are efficient enough to be considered serious threats to AES, although certain types of side channel attacks have been discovered. Algebraic attacks, however, have attracted attentions. For example, if the attacker knows a pair of AES-128 plaintext and ciphertext blocks, then the attacker may be able to calculate the AES encryption key by solving a system of 1600 quadratic equations of 8000 unknowns. Although solving a system of quadratic equations of this magnitude by today's mathematical theory and computing technology is hopeless, this study opens up a new direction of investigation.

2.5   Standard Block Cipher Modes of Operations

Let ℓ be the block size of a given block cipher (e.g., $\ell = 64$ for DES and $\ell = 128$ for AES). Let M be a plaintext string. Divide M into a sequence of blocks:

$$M = M_1 M_2 \cdots M_k,$$

such that the size of each block M_i is ℓ (using padding for the last block if necessary). There are several methods to encrypt M. Such methods are referred to as *block cipher modes* of operations. The following are the standard block cipher modes of operations:

1. *electronic-codebook mode* (ECB),
2. *cipher-block-chaining mode* (CBC),
3. *cipher-feedback mode* (CFB),
4. *output-feedback mode* (OFB), and
5. *counter mode* (CTR).

Table 2.5 ECB mode

ECB encryption steps	ECB decryption steps
$C_i = E_K(M_i),$ $\quad i = 1, 2, \cdots, k.$	$M_i = D_K(C_i),$ $\quad i = 1, 2, \cdots, k.$

2.5.1 Electronic-Codebook Mode

The ECB mode encrypts each plaintext block independently. Let C_i be the ith ciphertext block. Table 2.5 lists the encryption and decryption steps under the ECB mode.

ECB is often used to encrypt short plaintext messages M.

2.5.2 Cipher-Block-Chaining Mode

When the plaintext message M is long, the possibility that $M_i = M_j$ for some $i \neq j$ will increase. When this happens, their corresponding cipher blocks C_i and C_j are identical under the ECB mode, which will be disclosed to the eavesdropper. The use of the cipher-block-chaining mode can overcome this weakness. Under the CBC mode, the previous ciphertext block is used to encrypt the current plaintext block. At the beginning, CBC uses an initial ℓ-bit block C_0, referred to as an *initial vector*. Table 2.6 lists the encryption and decryption steps under the CBC mode.

CBC is commonly used in practice.

2.5.3 Cipher-Feedback Mode

Under the ECB and CBC modes, the receiver must wait for the entire ciphertext block to arrive before decryption can be started. There are several drawbacks in these schemes:

1. If the ciphertext block is too long, it would hinder the receiver from reading the entire plaintext message M continuously.
2. If padding is used when dividing M into blocks, the actual number of transmitted bits in ciphertext blocks will be larger than the number of bits in M.
3. If a bit error occurs in a ciphertext block during transmission (i.e., a bit is flipped during transmission), it would affect the readability of the plaintext block after decryption because of the effect of diffusion.

The use of CFB mode can overcome these drawbacks. CFB does not divide M into blocks. Instead, it encrypts each basic code one at a time. Let s be the length of the basic code in a

Table 2.6 CBC mode

CBC encryption steps	CBC decryption steps
$C_i = E_K(C_{i-1} \oplus M_i),$ $\quad i = 1, 2, \cdots, k.$	$M_i = D_K(C_i) \oplus C_{i-1},$ $\quad i = 1, 2, \cdots, k.$

given code set. For example, $s = 8$ for ASCII code and $s = 16$ for Unicode. Note that s can also be set to other values, as long as the length of the block is divisible by s. Let

$$M = w_1 w_2 \cdots w_m,$$

where each w_i is an s-bit binary string, and ℓ is divisible by s.

Under CFB mode, the sender and the receiver share the same ℓ-bit initial vector V_0. Encryption begins by encrypting V_0 to produce a ciphertext block U_1. Let $\mathsf{p}_s(U)$ represent the s-bit prefix of U, and $\mathsf{s}_j(U)$ the j-bit suffix of U. The encryption procedure calculates $C_1 = w_1 \oplus \mathsf{p}_s(U_1)$. It then shifts V_0 s bits to the left and fills in the s bits on the right with C_1. Repeat this until C_m is obtained. Table 2.7 lists the encryption and decryption steps under the CFB mode.

CFB is a common method to turn a block cipher algorithm into a stream cipher algorithm.

2.5.4 Output-Feedback Mode

If during the transmission of a CFB cipher string C_i a bit error occurs, then this error not only will affect the correctness of w_i, but also will affect the correctness of $w_{i+1}, w_{i+2}, \cdots, w_{i+\ell/s}$. This is because C_i will be removed from V only after ℓ/s iterations. Output feedback mode can overcome this drawback. OFB is similar to CFB. The only difference is that OFB does not place C_i in V_i. Table 2.8 lists the encryption and decryption steps under the OFB mode.

OFB is also a common method to turn a block cipher algorithm to a stream cipher algorithm. It is commonly used in error-prone environments.

2.5.5 Counter Mode

CTR produces block ciphers. It uses an ℓ-bit counter *Ctr*, which starts from an initial value and increases by 1 each time. Adding 1 to 1^ℓ resets *Ctr* to 0^ℓ. In other words,

Table 2.7 CFB mode

CFB encryption steps	CFB decryption steps
$U_i = E_K(V_{i-1}),$ $C_i = w_i \oplus \mathsf{p}_s(U_i),$ $V_i = \mathsf{s}_{\ell-s}(V_{i-1})C_i,$ $i = 1, \cdots, m-1;$ $U_m = E_K(V_{m-1}),$ $C_m = w_m \oplus \mathsf{p}_s(U_m).$	$U_i = E_K(V_{i-1}),$ $w_i = C_i \oplus \mathsf{p}_s(U_i),$ $V_i = \mathsf{s}_{\ell-s}(V_{i-1})C_i,$ $i = 1, \cdots, m-1;$ $U_m = E_K(V_{m-1}),$ $w_m = C_m \oplus \mathsf{p}_s(U_m).$

Table 2.8 OFB mode

OFB encryption steps	OFB decryption steps
$U_i = E_K(V_{i-1}),$ $C_i = w_i \oplus \mathsf{p}_s(U_i),$ $V_i = \mathsf{s}_{\ell-s}(V_{i-1})\mathsf{p}_s(U_i),$ $i = 1, \cdots, m-1;$ $U_m = E_K(V_{m-1}),$ $C_m = w_m \oplus \mathsf{p}_s(U_m).$	$U_i = E_K(V_{i-1}),$ $w_i = C_i \oplus \mathsf{p}_s(U_i),$ $V_i = \mathsf{s}_{\ell-s}(V_{i-1})\mathsf{p}_s(U_i),$ $i = 1, \cdots, m-1;$ $U_m = E_K(V_{m-1}),$ $w_m = C_m \oplus \mathsf{p}_s(U_m).$

Table 2.9 CTR mode

CTR encryption steps	CTR decryption steps
$Ctr = Ctr_0,$	$Ctr = Ctr_0,$
$\quad C_i = E_K(Ctr^{++}) \oplus M_i,$	$\quad M_i = E_K(Ctr^{++}) \oplus C_i,$
$\quad i = 1, \cdots, k.$	$\quad i = 1, \cdots, k.$

$Ctr \leftarrow Ctr + 1 \mod 2^\ell$. We use Ctr_0 to denote the initial value of *Ctr* and Ctr^{++} to denote $Ctr + 1 \mod 2^\ell$. Table 2.9 lists the encryption and decryption steps under the CTR mode.

CTR is simple, and it overcomes the drawbacks of ECB. It is commonly used in applications that require faster encryption speed.

2.6 Offset Codebook Mode of Operations

The Offset codebook mode (OCB) of operations is a newer but more complex mode of operations for block ciphers. It was devised by Rogaway, Bellare, Black, and Krovetz in 2001. OCB provides encryption and authentication simultaneously, and it has stronger security properties than the standard modes of operations introduced in Section 2.5. OCB is parallelizable, allowing multiple hardware units to execute the algorithm on the same input simultaneously.

2.6.1 Basic Operations

Let a be an ℓ-bit binary string. Denote by $\mathsf{firstbit}(a)$ the first bit of a and $\mathsf{lastbit}(a)$ the last bit of a.

Let $\ell = 128$. Define $f(a)$ and $g(a)$ as follows:

$$f(a) = \begin{cases} a \ll 1, & \text{if } \mathsf{firstbit}(a) = 0, \\ (a \ll 1) \oplus 0^{120}10000111, & \text{if } \mathsf{firstbit}(a) = 1. \end{cases} \tag{2.31}$$

$$g(a) = \begin{cases} a \gg 1, & \text{if } \mathsf{lastbit}(a) = 0, \\ (a \gg 1) \oplus 10^{120}10000111, & \text{if } \mathsf{lastbit}(a) = 1. \end{cases} \tag{2.32}$$

Note that if we treat a as an element in $GF(2^{128})$, that is, if we treat a as a sequence of coefficients of a polynomial of degree 127, then $f(a) = ax$ and $g(a) = ax^{-1}$. Let $\oplus$ and $\otimes$ denote, respectively, the addition operation and the multiplication operation of $GF(2^{128})$, where $\oplus$ is the ordinary exclusive-OR operation on coefficients.

Let i be a positive integer. Let $\mathsf{ntz}(i)$ denote the largest integer z such that 2^z divides i. That is, $\mathsf{ntz}(i)$ is the number of trailing 0's in the binary representation of i.

For example, $\mathsf{ntz}(12) = \mathsf{ntz}(1100) = 1$ and $\mathsf{ntz}(16) = \mathsf{ntz}(10000) = 4$.

Let $i \geq -1$ be an integer. Let $a(i)$ denote $a \otimes x^i$. Thus, $a(-1) = g(a), a(0) = a$, and $a(1) = f(a)$. It is straightforward to see that, for $i > 1$,

$$a(i) = f(f(\cdots f(a) \cdots)) \text{ with } i \text{ many } f\text{'s.} \tag{2.33}$$

Let γ^k denote a sequence of 2^k binary strings of length k as

$$\gamma^k = (\gamma_0^k, \gamma_1^k, \cdots, \gamma_{2^k-1}^k),$$

where $\gamma_0^k = 0^k$ and $\gamma_1^k = 0_1^{k-1}$, such that every two successive strings differ in exactly one place. That is, if we perform XOR on any two successive strings, we get exactly one 1 in the resulting new string. The number of 1's as the result of XORing two strings is referred to as the Hamming distance of the strings. A sequence of strings of equal length is a Gray code if for every two successive strings, their Hamming distance is 1. Thus, γ^k is a Gray code. Let

$$\gamma^{k+1} = (0\gamma_0^k, 0\gamma_1^k, \cdots, 0\gamma_{2^k-2}^k, 0\gamma_{2^k-1}^k, 1\gamma_{2^k-1}^k, 1\gamma_{2^k-2}^k, \cdots, 1\gamma_1^k, 1\gamma_0^k). \tag{2.34}$$

Then γ^{k+1} is also a Gray code.

Let γ denote γ^ℓ. That is, $\gamma_i = \gamma_i^\ell$ for $0 \le i \le 2^\ell - 1$. Then for $1 \le i \le 2^\ell - 1$, we have

$$\gamma_i = \gamma_{i-1} \oplus (0^{\ell-1}1 \ll \mathsf{ntz}(i)). \tag{2.35}$$

This implies that

$$\begin{aligned}
\gamma_i \otimes a &= [\gamma_{i-1} \oplus (0^{\ell-1}1 \ll \mathsf{ntz}(i))] \otimes a \\
&= (\gamma_{i-1} \otimes a) \oplus (0^{\ell-1}1 \ll \mathsf{ntz}(i)) \otimes a \\
&= (\gamma_{i-1} \otimes a) \oplus (a \otimes x^{\mathsf{ntz}(i)}) \\
&= (\gamma_{i-1} \otimes a) \oplus a(\mathsf{ntz}(i)).
\end{aligned}$$

Let n be a non-negative integer and $b(n)$ the binary representation of n. Assume that $|b(n)| \le \ell$. Let

$$\mathsf{ppad}_\ell(n) = 0^{\ell-|b(n)|}b(n),$$

where ppad stands for "prefix padding". Let X be a binary string. Let

$$\mathsf{len}_\ell(X) = \mathsf{ppad}_\ell(|X| \bmod 2^\ell).$$

For example, let X be an 18-bit binary string. Then

$$\mathsf{len}_4(X) = \mathsf{ppad}_4(18 \bmod 2^4) = \mathsf{ppad}_4(2) = 0010.$$

We extend the XOR operation on two binary strings of unequal length as follows: let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$ be two binary strings, where x_i and y_j are bits for $1 \le i \le m$ and $1 \le j \le n$. Let $\kappa = \min\{m, n\}$. Define

$$X \oplus Y = x_1 \cdots x_\kappa \oplus y_1 \cdots y_\kappa = (x_1 \oplus y_1)(x_2 \oplus y_2) \cdots (x_\kappa \oplus y_\kappa).$$

For example,

$$1001 \oplus 010101 = 1001 \oplus 0101 = 1100.$$

2.6.2   OCB Encryption and Tag Generation

We now describe how OCB encrypts data and authenticates data. We assume that the block size of the underlying encryption algorithm E (e.g., AES) is $\ell = 128$. Using an encryption algorithm with a different block size follows the same procedure except that we need to modify

$f(a)$ and $g(a)$ defined in Equalities 2.31 and 2.32. OCB produces a τ-bit tag for authentication, where $\tau \leq \ell$.

Suppose that Alice wants to encrypt and authenticate a plaintext message M to be sent to Bob. Alice first divides M into ℓ-bit blocks as

$$M = M_1 \parallel M_2 \parallel \cdots \parallel M_m,$$

where $|M_1| = |M_2| = \cdots = |M_{m-1}| = \ell$ and $1 \leq |M_m| \leq \ell$.

Let K denote the encryption key of E shared by Alice and Bob. Alice selects an ℓ-bit initial vector N, which is transmitted in plaintext to Bob.

Alice encrypts M and produces a tag T of M as follows:

$$L = E_K(0^\ell),$$
$$R = E_K(N \oplus L),$$
$$Z_i = \gamma_i \otimes (L \oplus R) \ \text{ for } i = 1, \cdots, m,$$
$$C_i = E_K(M_i \oplus Z_i) \oplus Z_i \ \text{ for } i = 1, \cdots, m-1,$$
$$X_m = \mathsf{len}(M_m) \oplus f(L) \oplus Z_m,$$
$$Y_m = E_K(X_m),$$
$$C_m = Y_m \oplus M_m,$$
$$T = \mathsf{p}_\tau(E_K(M_1 \oplus \cdots \oplus M_{m-1} \oplus C_m 0^{\ell - |C_m|} \oplus Y_m \oplus Z_m)).$$

Alice sends $N \parallel C_1 \parallel \cdots \parallel C_m \parallel T$ to Bob.

2.6.3 OCB Decryption and Tag Verification

On receiving $N \parallel C_1 \parallel \cdots \parallel C_m \parallel T$ from Alice, Bob does the following to obtain M and verify the tag.

$$L = E_K(0^\ell),$$
$$R = E_K(N \oplus L),$$
$$Z_i = \gamma_i \otimes (L \oplus R) \ \text{ for } i = 1, \cdots, m,$$
$$M_i = D_K(C_i \oplus Z_i) \oplus Z_i \ \text{ for } i = 1, \cdots, m-1,$$
$$X_m = \mathsf{len}(C_m) \oplus g(L) \oplus Z_m,$$
$$Y_m = E_K(X_m),$$
$$M_m = Y_m \oplus C_m,$$
$$T' = \mathsf{p}_\tau(E_K(M_1 \oplus M_{m-1} \oplus C_m 0^{\ell - |C_m|} \oplus Y_m)).$$

If $T = T'$, then Bob accepts $M_1 M_2 \cdots M_m$ and rejects it otherwise.

2.7 Stream Ciphers

Block ciphers under the CFB or OFB modes can produce stream ciphers. However, they incur extra computation overhead. In certain network applications, particularly in wireless network applications where end devices are hand-held devices (e.g., simple cell phones), executing full-strength block ciphers on these devices may be undesirable because these devices tend to have limited computation capabilities and stringent power supplies. Thus, one would like to use stream ciphers that require less computing power and consume less energy.

The first stream cipher was invented by Gilbert S. Vernam in 1917. We introduce in this section the RC4 (pronounced "arc-four") stream cipher. RC4 was designed by Ron Rivest in 1987 as a trade secret for RSA Security, which was made public in 1994. In particular, RC4 is a major component in the Wired Equivalent Privacy (WEP) protocol adopted in the IEEE 802.11b standard for providing Ethernet-like MAC-layer access for wireless LANs.

2.7.1 RC4 Stream Cipher

RC4 uses encryption keys of variable lengths of $8l$ bits, where $l(1 \leq l \leq 256)$ is an integer chosen by the user. RC4 uses substitution and modular addition operations to generate a sequence of 8-bit subkeys, and XORs the current plaintext character with a new subkey to generate a cipher stream.

2.7.1.1 RC4 Subkey Generation

RC4 uses an array $S[0, 255]$ of 256 bytes to generate subkeys. This array is used to form a new permutation of 8-bit binary strings at each iteration. Let K be an encryption key, where $|K| = 8l, 1 \leq l \leq 256$. Rewrite K as an array $K[0, l-1]$ of l bytes. That is,

$$K = K[0]K[1] \cdots K[l-1].$$

RC4 generates subkey streams as follows:

Key Scheduling Algorithm (KSA)

Initialization:
    for each $i = 0, \cdots, 255$
        set $S[i] \leftarrow i$
Initial permutation:
    set $j \leftarrow 0$
    for each $i = 0, 1, \cdots, 255$
        set $j \leftarrow (j + S[i] + K[i \bmod l]) \bmod 256$
        swap $S[i]$ with $S[j]$

Subkey Generation Algorithm (SGA)

Initialization:
    set $i \leftarrow 0$
    set $j \leftarrow 0$
    set $u \leftarrow 0$
Permutation and generation loop:
    set $u \leftarrow u+1$
        set $i \leftarrow (i+1) \bmod 256$
        set $j \leftarrow (j + S[i]) \bmod 256$
        swap $S[i]$ with $S[j]$
        set $K_u \leftarrow S[(S[i] + S[j]) \bmod 256]$ (See Fig. 2.3)
        repeat

RC4 Encryption and Decryption

Let $M = M_1 M_2 \cdots M_k$ be a plaintext string, where each M_i is an 8-bit binary string.

RC4 encryption: $C_i = M_i \oplus K_i, i = 1, 2, \cdots, k$.
RC4 decryption: $M_i = C_i \oplus K_i, i = 1, 2, \cdots, k$.
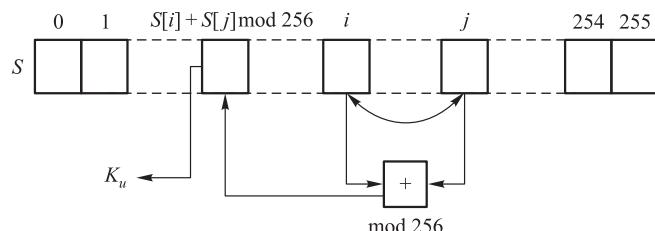
2.7.2 RC4 Security Weaknesses

KSA uses the secret encryption key to generate the initial permutation of S. SGA then generates subkey streams from the initial permutation, which no longer uses the encryption key in any form. This means that knowing the initial permutation is equivalent to breaking RC4 encryption. Even if the initial permutation is only partially revealed, the attacker may still be able to compute a few subkeys using SGA. Thus, KSA is the critical security point of RC4.

2.7.2.1 Weak Keys

Selecting a suitable encryption key to produce a secure initial permutation is difficult. Fluhrer, Mantin, and Shamir showed in 2001 that a large number of $8l$-bit binary strings are weak

Figure 2.3   RC4 subkey generation after KSA is performed and the values of i and j are set

encryption keys in the sense that a small part of the string could determine a large number of bits in the initial permutation and help reveal the secret encryption key.

In particular, let S be the initial permutation. Let b be a positive integer and i an index. Denote by $I_b(S)$ the number of indexes such that $S[i] \equiv i \mod b$. The initial permutation S is said to be *almost b-conserving* if $I_b(S) > N - 1$. Let K be an RC4 encryption key. If for any index i, we have $K[i \mod l] \equiv (1 - i) \mod b$, $K[0] = 1$, and the most significant bit of $K[1]$ is 1, then K is referred to as a *special b-exact key*.

Let K be an $8l$-bit special b-exact key, where $b = 2^q$ for some q with $1 \le q \le 8$. It can be shown that if l is divisible by b, then with a probability of at least 2/5, the initial permutation S produced by $\text{KSA}(K)$ is almost b-conserving. It can be shown that if keys are almost b-conserving, then there is a strong probabilistic correlation between certain bits of the secret key and certain bits of the subkey stream. Detailed analysis of this result is rather involved, which is omitted in this book. On the basis of this correlation, the attacker may be able to deduce the WEP key. We refer this attack to as the Fluhrer–Mantin–Shamir *FMS attack*.

In practice, an encryption key in network protocols using RC4 is often a concatenation of a long-term secret part and a short-term public part. The public component would reveal part of the initial permutation, and the public component may be reused. The reader is referred to Chapter 6 for detailed discussions of security weaknesses in RC4 applications in wireless communications.

2.7.2.2 Attacks from Reusing Subkey Streams

RC4 also requires that the subkey stream be used only once. Otherwise, it is vulnerable to a known-plaintext attack and a *related-plaintext* attack. The known-plaintext attack will reveal the subkey stream used to encrypt the plaintext (see Section 2.1.2).

The related-plaintext attack is intended to obtain the content of two plaintext messages by XORing the corresponding encrypted strings. In particular, Let M_1 and M_2 be two plaintext messages of the same length, where

$$M_1 = m_{11}m_{12}\cdots m_{1n},$$
$$M_2 = m_{21}m_{22}\cdots m_{2n},$$

and each m_{ij} is a binary bit. Suppose that they are encrypted by RC4 using the same encryption key K. That is, first apply RC4 on K to generate a subkey stream $k_1, k_2, \cdots, k_n$, then encrypt M_1 and M_2 to obtain $C_1 = c_{11}c_{12}\cdots c_{1n}$ and $C_2 = c_{21}c_{22}\cdots c_{2n}$, where $c_{ij} = m_{ij} \oplus k_j$. Suppose that the attacker intercepts C_1 and C_2, which allows him to obtain

$$c_{1j} \oplus c_{2j} = (m_{1j} \oplus k_j) \oplus (m_{2j} \oplus k_j) = m_{1j} \oplus m_{2j}, \quad j = 1, 2, \cdots, n.$$

Thus, the attacker obtains the exclusive-OR value of two unknown plaintext strings. From this, he may be able to deduce the original plaintext strings. For example, the attacker may use a statistical analysis to find common words and phrases in certain type of documents. He then performs exclusive-OR on each pair of the words and phrases to produce a list of binary strings that are the exclusive-OR values of two plaintext strings. This list will help deduce, on a given binary string, two plaintext strings whose exclusive-OR value is the same as the string.

2.8 Key Generations

Secret keys are critical components of encryption algorithms. The best way to generate secret keys is to generate them randomly. There are a number of methods to randomly generate encryption keys. For example, one may move the mouse at will on the screen and record its track as a binary sequence. This method may produce truly random binary strings. However, this method needs to interact with users, which is not practical for network applications. The best alternative is to generate pseudorandom strings using deterministic algorithms. Such algorithms are called *pseudorandom number generators* (PRNG). We introduce two PRNG algorithms in this section.

2.8.1 ANSI X9.17 PRNG

We note that a ciphertext block of an encryption algorithm by itself is a pseudorandom binary string. It can therefore be used as an encryption key. Using an encryption algorithm as a PRNG requires an initial key K. For example, we may choose a 128-bit binary string K as an encryption key of AES-128 under OFB mode, and then use $V_1'V_2'\cdots V_{16}'$ as an 128-bit secret key of AES-128, where V_i' is an 8-bit binary string determined by the following recurrence relations:

$$U_i = E_K(V_{i-1}), \ (V_0 \text{ is a fixed initial vector})$$
$$V_i' = \mathsf{p}_8(U_i)$$
$$V_i = \mathsf{s}_{\ell-8}(V_{i-1})V_i'$$
$$i = 1, 2, \cdots, 16.$$

The X9.17 PRNG standard was published in 1985 by the American National Standard Institute (ANSI) for financial institution key management (wholesale). It was reaffirmed in 1991 and updated in 1995. X9.17 PRNG is based on DES. In particular, it uses 3DES/2 with two initial keys K_1 and K_2 and an initial vector V_0. X9.17 also uses two special 64-bit binary strings T_i and V_i to generate a 64-bit pseudorandom string R_i at each round of computation, where T_i represents the current date and time, updated before each round, and V_i, called a *seed*, is determined by the following recurrence relations:

$$R_i = EDE_{K_1,K_2}(V_i \oplus EDE_{K_1,K_2}(T_i)),$$
$$V_{i+1} = EDE_{K_1,K_2}(R_i \oplus EDE_{K_1,K_2}(T_i)),$$
$$i = 0, 1, \cdots$$

2.8.2 BBS Pseudorandom Bit Generator

BBS is a pseudorandom bit generator devised by Lenore Blum, Manuel Blum (1995 Turing Award winner), and Michael Shub in 1986. It generates a pseudorandom bit in each round of computation. In particular, let p and q be two large prime numbers that satisfy

$$p \mod 4 = q \mod 4 = 3.$$

That is, the remainders of dividing p by 4 and dividing q by 4 are both equal to 3. We discuss how to find large prime numbers in Section 3.2.5. Let $n = p \times q$ and let s be a positive number

such that s and p are relatively prime and s and q are *relatively prime*. That is, $\gcd(s, p) = 1$ and $\gcd(s, q) = 1$, where $\gcd(x, y)$ denotes the largest common factor of x and y. Without loss of generality, assume that $x > y \geq 0$. Then $\gcd(x, y)$ can be calculated efficiently using the following recurrence relation, also known as Euclid's algorithm:

$$\gcd(x, y) = \begin{cases} \gcd(y, x \bmod y), & \text{if } y > 0, \\ x, & \text{if } y = 0. \end{cases}$$

It can be shown that Euclid's algorithm only incurs $O(\log\ y)$ recursive calls. It follows from Euclid's algorithm that there are integers a and b such that $\gcd(x, y) = ax + by$.

BBS generates pseudorandom bit as follows:

$$x_0 = s^2 \bmod n,$$
$$x_i = x_{i-1}^2 \bmod n,$$
$$b_i = x_i \bmod 2,$$
$$i = 1, 2, \cdots$$

For example, let $p = 383$ and $q = 503$. It is straightforward to verify that $p \equiv q \equiv 3(\bmod 4)$. Let $s = 101355$. Using Euclid's algorithm it is easy to show that $\gcd(s, p) = \gcd(s, q) = 1$. The first 128 binary bits $b_1, b_2, \cdots, b_{128}$ generated by BBS, for example, may then be used as an AES-128 encryption key.

The difficulty of predicting the $(k + 1)$th BBS bit b_{k+1} from the k previous BBS bits $b_1, \cdots, b_k$ depends on the difficulty of the *integer factorization* problem, also known as *integer factorization*. Integer factorization asks, for a given positive nonprime number n, all prime factors of n. This is a computationally intensive problem. The best known algorithm for integer factorization has a time complexity in the order of

$$e^{\sqrt[3]{\ln n (\ln \ln n)^2}}.$$

It can be shown that, if integer factorization cannot be solved in polynomial time, then a BBS pseudorandom bit cannot be distinguished from a true random bit in polynomial time. This means that any algorithm that can compute, with a probability greater than 1/2, the $(k + 1)$th BBS bit b_{k+1} from the k previous bits $b_1, \cdots, b_k$, its time complexity must be greater than any polynomial of the size of n, where the size of n is $|n| = \lceil \log_2 n \rceil$.

It is a common belief that integer factorization does not have polynomial-time algorithms under conventional computing devices. However, using an unconventional computation model, integer factorization can be solved efficiently. In 1994, Peter Shor, an American computer scientist, showed that integer factorization can indeed be solved in polynomial time using a theoretical model of quantum computers.

2.9 Closing Remarks

The study of encryption algorithms is an active research and development area. Any encryption algorithm, if it is commonly used to encrypt data or is made to be an encryption standard by reputable organizations, will attract attention. Despite intensive studies, however, there have

been no known encryptions algorithms that have been proven secure using mathematical methods. Therefore, people would consider an encryption algorithm secure (i.e., secure for the time being) if it resists all possible attacks one can think of under the current technology. Thus, DES and 2DES are considered insecure, while 3DES/2 and AES are considered secure. In important applications, we should only use encryption algorithms that have been studied extensively and in which no serious security flaws have been found.

In addition to encryption algorithms and key generation algorithms, how to manage encryption keys in local systems and how to distribute them over networks is another critical issue. We discuss this issue in the following chapter.

2.10   Exercises

2.10.1   Discussions

2.1. What are the basic structures and techniques to encrypt data?

2.2. Can you generalize the 64-bit block size and 56-bit key DES to DES+ with 128-bit block size and 128-bit key? How about with 196-bit and 252-bit keys?

2.3. Why do you think AES is a better encryption algorithm?

2.4. What is the best way to apply encryption algorithms to encrypt data?

2.5. What is the best way to generate secret keys?

2.6. How do you think secret keys are distributed between communication parties?

2.10.2   Homework

Programming assignments in this book are assumed to be carried out in the C language. You may also use C++ or Java, as long as you will do so consistently throughout the book.

2.1. A ciphertext message generated by a simple letter permutation maintains the letter frequencies of the plaintext message. To flatten frequencies in the ciphertext message, we may use a generalized XOR encryption. For simplicity, we assume that any plaintext message is a sequence of capital English letters. Firstly, we map all 26 capital letters to integers from 0 to 25, and we use I to denote this mapping. That is, $I(A) = 0, I(B) = 1, \cdots, I(Z) = 25$. Let X and Y be two English letters. Let

$$X + Y = I^{-1}([I(X) + I(Y)] \mod 26),$$

where I^{-1} is the inverse function of I, namely, $I^{-1}(0) = A, I^{-1}(1) = B, \cdots, I^{-1}(25) = Z$. Let $\mathcal{X} = X_1 \cdots X_n$ and $\mathcal{Y} = Y_1 \cdots Y_n$ be two strings of equal length over the English alphabet. Let

$$\mathcal{X} + \mathcal{Y} = (X_1 + Y_1) \cdots (X_n + Y_n).$$

Let K be an arbitrary letter string of length ℓ. We use K as the encryption key. Note that K may contain the same letter multiple times. Let M be a plaintext message. Divide M into blocks $M_1, M_2, \cdots M_k$, where the length of each block M_i $(i < k)$ is ℓ. Let the length of M_k be m. Let K_m denote the first m letters in K.

Define an encryption algorithm E as follows: $E_K(M) = C_1 C_2 \cdots C_k$, where $C_i = K + M_i$, $i = 1, \cdots, k-1$, and $C_k = K_m + M_k$.

(a) Describe the decryption algorithm D.

(b) Let $K =$ BLACKHAT. Encrypt the following passage:

> Methods of making messages unintelligible to adversaries have been necessary. Substitution is the simplest method that replaces a character in the plaintext with a fixed different character in the ciphertext. This method preserves the letter frequency in the plaintext, and so one can search for the plaintext from a given ciphertext by comparing the frequency of each letter against the known common frequency in the underlying language.

(c) Write a program to implement E and D.

2.2. Let

$$IP_{key}(K) = \frac{11010011101011000010110000111}{011010101010100111100010011101,}$$

Compute the DES subkey K_1.

2.3. Draw a block diagram of the DES encryption algorithm and a block diagram of the DES decryption algorithm.

2.4. Let M and K each be 64-bit binary strings, representing a plaintext message WHITE-HAT and an encryption key BLACKHAT.

Each letter in the plaintext message is encoded using an 8-bit ASCII code by adding a leading 0 to its 7-bit ASCII code. For example, the 7-bit ASCII code of letter W is 1010111 (see Appendix A), and its 8-bit ASCII code is 01010111.

Each letter in the encryption key is encoded using its 7-bit ASCII code, with an additional parity bit added at the end such that the total number of 1's (including the parity bit) is an even number. For example, the 7-bit ASCII code of letter B is 1000010 (see Appendix A), and so B is encoded by 10000100.

Carry out the first round of DES encryption. What is $L_1 R_1$?

2.5. Write a program to implement the DES subkey generation algorithm, where the input of the program is an 8-bit string of English letters, while the output is a sequence of 48-bit subkeys.

*2.6. Write a program `encrypt.c` and a program `decrypt.c` to implement, respectively, DES encryption and decryption. The program `encrypt.c` takes an English plaintext file, encoded in ASCII, as an input file and an eight-letter string as an encryption key. It encrypts the plaintext file and writes the ciphertext in a binary file as output.

The program `decrypt.c` takes a binary file as input and decrypts the ciphertext file using the same encryption key and writes the plaintext in ASCII file as output.

**2.7. This exercise involves two users. To do this exercise, you must first complete Exercise 2.6. User A first selects two DES encryption keys $K_{1,1}$ and $K_{1,2}$. User A then selects two 64-bit plaintext blocks $M_{1,j}$, where $j = 1, 2$, computes $C_{1,j} = E_{K_{1,2}}(E_{K_{1,1}}(M_{1,j}))$, and sends $(M_{1,j}, C_{1,j})$ by email to user B. Likewise, user B also selects two DES encryption keys $K_{2,1}$ and $K_{2,2}$, selects two 64-bit plaintext blocks $M_{2,j}$, where $j = 1, 2$, computes $C_{2,j} = E_{K_{2,2}}(E_{K_{2,1}}(M_{2,j}))$, and sends $(M_{2,j}, C_{2,j})$ to user A by email.

From each of the keys $K_{1,j}$, user A selects 10 arbitrary bits and replaces the remaining bits each with a question mark. User A sends the two modified strings to user B. Likewise, user B does the same thing on keys $K_{2,j}$ and sends the corresponding modified strings to user A. User A carries out the meet-in-the-middle attack on 2DES to user B's secret key and vice versa. Do they have the same success rate?

2.8. We have shown that meet-in-the-middle attack is substantially more effective than a brute-force attack. Generalizing this idea, devise a meet-in-the-middle attack on 3DES/2. Is it effective?

*2.9. Let E be the encryption algorithm defined in Exercise 2.1. Show that for any encryption keys K_1 and K_2, there is always an encryption key K_3 such that for any plaintext message M, we have

$$E_{K_2}(E_{K_1}(M)) = E_{K_3}(M).$$

Note that K_1 and K_2 may have different lengths.

2.10. Show that 3DES/3 (see Formula 2.10) can be used to decrypt ciphertext message produced by DES.

2.11. Point out unsymmetrical places between AES encryption and AES decryption.

2.12. Let $K = $ 1234567890abcdef1234567890abcdef be an AES-128 encryption key, represented in hexadecimal. Calculate round key $K_1 = W[4, 7]$.

2.13. Show that generating AES-128 round keys is equivalent to the following pseudo code.

```
KeyExpansion (byte K[16], word W[44]) {
    int i;
    word temp;
    for (i=0; i < 4; i ++)
        W[i] = K[4*i,4*i+3];
    for (i = 4; i < 44; i++) {
        temp = W[i-1];
        if (i mod 4 == 0)
            temp = SubWord(RotWord(temp)) ⊕ Rcon[i/4];
        W[i] = W[i-4] ⊕ temp;
    }
}
```

Here functions SubWord, RotWord, and Rcon are defined as follows. Let $W = w_1 w_2 w_3 w_4$ be a word, where each w_i is a byte. Then

$$\text{SubWord}(W) = S(w_1)S(w_2)S(w_3)S(w_4),$$

$$\text{RotWord}(W) = w_2 w_3 w_4 w_1.$$

Rcon[j] is a round constant, which is a word defined by ($RC[j]$, 0, 0, 0), with

$$RC[j] = \begin{cases} \text{02} \otimes RC[j-1], & \text{if } j > 1, \\ \text{01}, & \text{if } j = 1. \end{cases}$$

2.14. Verify the following elements in matrix $mic(A)$ (see Equality 2.16): $a'_{0,1} = \text{4d}$, $a'_{0,2} = \text{9f}$, $a'_{0,3} = \text{d5}$.

2.15. Let $(a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}) = (\text{8e},\text{4d},\text{a1},\text{bc})$ be the first row in the state matrix A, compute the first column in matrix $mic^{-1}(A)$; that is, compute $(a''_{0,0}, a''_{1,0}, a''_{2,0}, a''_{3,0})$.

2.16. Let w_1 and w_2 be two 8-bit binary strings. Let A and B be two 4×4 byte matrices, that is, each element in the matrix is an 8-bit binary string. Prove the following equalities:
(a) $\mathcal{M}(w_1 \oplus w_2) = \mathcal{M}(w_1) \oplus \mathcal{M}(w_2)$.
(b) $mic^{-1}(A \oplus B) = mic^{-1}(A) \oplus mic^{-1}(B)$.

2.17. Let $K = \text{a0\,a1\,b2\,b3\,c4\,c5\,d6\,d7\,e8\,e9\,fa\,fb\,0c\,0d\,1e\,1f}$ be an AES-128 encryption key, represented in hexadecimal. Execute the first round of AES-128 on the plaintext block $\text{01\,12\,23\,34\,45\,56\,67\,78\,89\,9a\,ab\,bc\,cd\,de\,ef\,f0}$. What is the state matrix A_2 after the first round?

2.18. Let A be a state matrix. Show that shr^{-1} and sub^{-1} commute, that is,

$$shr^{-1}(sub^{-1}(C_i)) = sub^{-1}(shr^{-1}(C_i)).$$

2.19. Let $p(x)$ be a polynomial of degree n in $GF(2^n)$. Show that

$$x^n \bmod p(x) = p(x) - x^n.$$

2.20. Complete the verification of Equality 2.28.

2.21. Prove Equalities 2.29 and 2.30.

2.22. Following the construction algorithm of the AES S-Box, find the fourth element in the first row, that is, $s_{0,3}$, in the S-Box S and the fourth element in the first row, that is, $s'_{0,3}$, in the inverse S-Box S^{-1}.

*2.23. Write a client-server program using socket API to implement AES-128 using an encryption key known to both sides, which is stored in a file. The client program takes a plaintext file and the encryption key file as input, encrypts the plaintext file using AES-128, and sends ciphertext blocks to the server program one block at a time. The server program uses the same encryption key from the encryption key file to decrypt the blocks it receives, one block at a time, and writes the plaintext blocks to a file.

2.24. RC5 is a block cipher with a Feistel structure. Its block size, the number of rounds, and key length may vary. In particular, RC5 takes $2w$-bit block as input, where $w \in \{16, 32, 64\}$; runs for r rounds, where $r \in \{0, 1, \cdots, 255\}$; and uses b-byte keys, where $b \in \{0, 1, \cdots, 255\}$. It is customary to denote RC5-$w/r/b$ an RC5 encryption algorithm with parameters w, r, and b. For example, RC5-32/12/16 takes a 64-bit block as input, runs for 12 rounds, and uses a 128-bit encryption key.

RC5 uses $t = 2r + 1$ subkeys of length $w : S_0, S_1, \cdots, S_{t-1}$, generated by the following algorithm. Let K be a b-byte encryption key: $K_0, \cdots, K_{b-1}$, where K_i is the ith byte in K. Let c be the smallest integer that is greater than or equal to $8b/32$. Let $L_0 L_1 \cdots L_{c-1}$ be a $32c$-bit binary string, where each L_i is a 32-bit binary string. Copy K to L from left to right. Pad the unoccupied locations in L (if any) with 0. Let

$S_0 \leftarrow P_w$
For $i = 1$ to $t - 1$, let
$\quad S_i \leftarrow (S_i - 1 + Q_w) \bmod 2^{32}$
Let $i \leftarrow j \leftarrow A \leftarrow B \leftarrow 0$
Execute the following statements for $3 \times$ max $\{t, c\}$ times:
$\quad A \leftarrow S_i \leftarrow (S_i + A + B) \lll 3$
$\quad B \leftarrow L_j \leftarrow (L_j + A + B) \lll (A + B)$
$i \leftarrow (i + 1) \bmod t$
$j \leftarrow (j + 1) \bmod c$

where $P_w = \mathrm{Odd}[(e - 2)2^w]$, $Q_w = \mathrm{Odd}[(\Phi - 1)2^w]$, $\mathrm{Odd}(x)$ denotes the odd number that is closest to x, Φ is the golden ratio $\frac{1+\sqrt{5}}{2}$, and $x \lll y$ denotes the left-circular-shift operation on x for y bits. In particular, P_w and Q_w are given as follows (in hexadecimal):

w	16	32	64
P_w	b7e1	b7e15163	b7e151628aed2a6b
Q_w	9e37	9e3779b9	9e3779b97f4a7c15

Write a program to generate RC5 keys.

2.25. RC5 encryption and decryption are given as follows. Let $M = LR$, where L and R are, respectively, w-bit binary strings.

RC5 encryption:

$L \leftarrow (L + S_0) \bmod 2^{32}$
$R \leftarrow (R + S_1) \bmod 2^{32}$
For $i = 1$ to r, let
$\quad L \leftarrow (((L \oplus R) \lll R) + S2i) \bmod 2^{32}$
$\quad R \leftarrow (((L \oplus R) \lll L) + S_2 i + 1) \bmod 2^{32}$

RC5 decryption:

> For $i = r$ down to 1, let
> $$R \leftarrow (((R - S_2 i + 1) \bmod 2^{32}) >>> L) \oplus L$$
> $$L \leftarrow (((L - S_2 i) \bmod 2^{32}) >>> R) \oplus R$$
> $$R \leftarrow (R - S_1) \bmod 2^{32}$$
> $$L \leftarrow (L - S_0) \bmod 2^{32}$$

where $x >>> y$ denotes the circular-right-shift operation on x for y bits.

(a) Prove the correctness of RC5 decryption.

(b) Write a program to implement RC5 encryption and decryption, using Exercise 2.24 to generate encryption keys. Here the plain text is an ASCII file, while the encryption keys and cipher text are stored in binary files. Note that RC5 follows the little-endian format to store binary strings (see Exercise 2.26).

2.26. Current computer architecture is based on 32-bit or 64-bit CPU. These computers store information by words and address memory locations by bytes. Thus, one word has four addressable units, whose relative addresses are 0, 1, 2, 3. Let $w = w_3 w_2 w_1 w_0$ be a 4-byte binary string. We have two choices to store w in a word: store w_i at relative address i, or store it at relative address $3 - i$, where $0 \leq i \leq 3$. The first choice is referred to as *little-endian* storage, and the latter *big-endian* storage. In other words, if bytes in a 4-byte string are read from left to right, then in the little-endian storage, the first byte is stored in the location with the largest relative address in a word, the second byte is stored in the location with the second largest relative address, and so on; in the big-endian storage, the first byte is stored in the location with the smallest relative address in a word, the second byte is stored in the location with the second smallest relative address, and so on. Let $w = $ 08040201 (hexadecimal), the following shows how w is stored in the little-endian storage and in the big-endian storage:

For another example, on a 16-bit computer, the basic storage unit is a 2-byte memory unit, where each byte is addressable. Thus, to store UNIX, we get UNIX in the big-endian storage, and we get NUXI in the little-endian storage.

Write a program that can exchange between the little-endian storage and the big-endian storage.

2.27. Show that in the CBC mode, any error occurred in one cipher block during transmission will affect the correctness of two plaintext blocks at the receiving side.

Table 2.10   Little-endian storage and big-endian storage of 08040201

Relative address	Little-endian	Big-endian
0	01	08
1	02	04
2	04	02
3	08	01

2.28. Suppose that we are using AES under the CFB mode with $s = 8$. If a transmission error occurs in one cipher block, how many plaintext blocks will be affected at the receiving side?

2.29. For each of the following cipher-block modes, draw a block diagram for encryption and a block diagram for decryption.

    (a) Electronic codebook mode (ECB).
    (b) Cipher block chaining mode (CBC).
    (c) Cipher feedback mode (CFB).
    (d) Output feedback mode (OFB).
    (e) Counter mode (CTR).

2.30. Show that the OCB decryption and tag verification described in Section 2.6.3 is correct.

2.31. Draw a block diagram to describe OCB encryption and tag generation.

2.32. Draw a block diagram to describe OCB decryption and tag verification.

*2.33. Give two examples to show that OCB has stronger security properties than standard block cipher modes of operations introduced in Section 2.5.

*2.34. In Exercise 2.23, you have written a client-server program to encrypt and decrypt data using AES-128 under ECB. Rewrite this program using CBC, where the initial vector is a pseudorandom binary string generated by BBS.

*2.35. Let $M_1, \cdots, M_k$ be a sequence of plaintext blocks, where each M_i is ℓ-bit long for $1 \leq i < k$, ℓ is the input size of the underlying encryption algorithm E, and M_k is q-bit long for $q < \ell$. Define a *ciphertext stealing mode* (CTS) as follows, where C_0 is an ℓ-bit initial vector and K an encryption key:

$$C_i = E_K(M_i) \oplus C_{i-1}, \quad i = 1, \cdots, k - 2,$$
$$C_k = \mathsf{p}_q(Z_{k-1}), \quad Z_{k-1} = E_K(Y_{k-1}), \quad Y_{k-1} = M_{k-1} \oplus C_{k-2},$$
$$C_{k-1} = E_K(Y_k), \quad Y_k = Z_{k-1} \oplus M_k 0^{\ell - q}.$$

    (a) Describe how to decrypt C_{k-1} and C_k, and prove the correctness of your decryption.
    (b) Draw a block diagram for encryption and a block diagram for decryption under CTS.

2.36. Alice proposes the following method to verify that she and Bob share the same AES-128 key. Alice generates a 128-bit binary string r using BBS, encrypts r, and sends the ciphertext block $r_A = E_{K_A}(r)$ to Bob, where E is the AES-128 encryption algorithm and K_A is Alice's AES-128 encryption key. Bob decrypts r_A to get $r' = D_{K_B}(r_A)$ and sends r' to Alice, where D is the AES-128 decryption algorithm and K_B is Bob's AES-128 encryption key. Alice checks whether $r' = r$. If so, then $K_A = K_B$. Is this protocol secure? Justify your answer.

2.37. Modify RC4 as follows: shorten the array S from 256 cells in RC4 to eight cells and replace each occurrence of 255 in RC4 with 7. This gives a simplified version of RC4. Let $K = 0110010110000011$ be an encryption key. Use this simplified RC4 to encrypt plaintext WHITEHAT.

*2.38. Let $M_1 = m_{11}m_{12}\cdots m_{1n}$ and $M_2 = m_{21}m_{22}\cdots m_{2n}$ be two binary strings that are unknown to you, where each m_{ij} is a binary bit. However, you know

$$M_1 \oplus M_2 = (m_{11} \oplus m_{21})(m_{21} \oplus m_{22}) \cdots (m_{1n} \oplus m_{2n}).$$

Describe how you may be able to deduce M_1 and M_2.

2.39. Let $p = 383$ and $q = 503$. Show that $p \equiv q \equiv 3 \ (\mathrm{mod}\ 4)$. Then let $s = 101355$. Write a program to implement BBS and produce the first 128 pseudorandom bits $b_1, b_2, \cdots, b_{128}$.

2.40. The following result can be used to check whether a PRNG is sufficiently random: for any two positive integers x and y, if they are selected uniformly at random, then the probability that $\gcd(x, y) = 1$ is equal to $6/\pi^2$. Write a program to verify the randomness of the PRNG supported by the operating system of your machine.

3

Public-Key Cryptography and Key Management

To use data encryption algorithms and key generation algorithms in network communications, users involved in a communication must first agree on using the same secret keys. Before public-key cryptography was invented, delivering secret keys from one user to another relied on couriers. For example, one user would generate a secret key and then use a trusted courier to deliver the key to the other users. Or the users would set up a meeting to determine a secret key with all users present. Secret keys may also be delivered using a variety of communication systems, including postal service, email service, and phone service. These methods, however, are insecure and inflexible for network communication applications.

Invented in the 1970s, public-key cryptography (PKC) was a major breakthrough in cryptography. It makes it possible, without sharing prior secrets, to distribute secret keys securely and to authenticate data. The study of PKC also provides new applications to the seemingly unrelated area of number theory. In this chapter, we first introduce the basic concepts of PKC. We then describe several concrete public-key cryptosystems, including Diffie-Hellman key exchange, Elgamal public-key cryptosystem, RSA public-key cryptosystem, and elliptic-curve PKC. These methods use several results in number theory. For convenience, we include a section reviewing these number theoretic results. Finally, we discuss how to transmit secret keys using PKC without sharing prior secrets and how to manage keys. Data authentication methods will be introduced in Chapter 4.

3.1 Concepts of Public-Key Cryptography

In the following sections, we refer to user A as Alice, user B as Bob, user C as Charlie, and the attacker as Malice.

PKC is a new concept. It allows Alice and Bob to exchange secret keys securely and efficiently over public networks without sharing prior secrets. Let us first look at an example. Suppose that Alice wants to send a message M (e.g., M could be an AES-128 encryption key)

confidentially to Bob using the standard postal service. However, Alice and Bob do not share prior secrets, and so if Alice uses a conventional encryption algorithm to encrypt M and sends to Bob the encrypted M, Bob will have no way to decrypt it.

To overcome this obstacle, Bob comes up with the following scheme: Bob first sends an empty box with a lock hasp and an open padlock to Alice. Bob keeps the key in a secure place. After receiving the empty box and the open padlock from Bob, Alice places M in the box, locks it with Bob's padlock, and sends the locked box back to Bob. Bob uses his key to open the lock and reads M. This is a basic idea of PKC. In this example, the open lock serves as the *public key* used for encryption, which is open to the public. The key Bob keeps is the *private key* used for decryption, which is to be kept private.

PKC transforms this idea to a mathematical form suitable for network communications.

Let us consider another example. Assume that Alice has defined two functions f_0 and f_1 such that the following equality holds:

$$f_1(f_0(a, y), x) = f_1(f_0(a, x), y), \tag{3.1}$$

and that it is difficult to derive x from $f_0(a, x)$ and a. On the basis of these two functions, Alice devises a public-key cryptosystem. The purpose of this system is for Alice and Bob to calculate the same encryption key. In particular, let a be a public key known to Alice and Bob. Alice randomly selects a positive number x_1 as her private key, calculates $y_1 = f_0(a, x_1)$, and sends y_1 to Bob. Meanwhile, Bob randomly selects a positive number x_2 as his private key, calculates $y_2 = f_0(a, x_2)$, and sends y_2 to Alice. Alice calculates $K_1 = f_1(y_2, x_1)$ and uses K_1 as her secret key for a conventional encryption algorithm. Bob calculates $K_2 = f_1(y_1, x_2)$ and uses K_2 as his secret key for a conventional encryption algorithm. It follows from

$$f_1(y_2, x_1) = f_1(f_0(a, x_2), x_1) = f_1(f_0(a, x_1), x_2) = f_1(y_1, x_2)$$

that $K_1 = K_2$. Thus, Alice and Bob now share the same secret key K_1, although Alice does not know x_2 and Bob does not know x_1. Although Malice may eavesdrop y_1 and y_2, she cannot obtain x_1 or x_2. Thus, it is difficult for Malice to calculate K_1 or K_2.

PKC provides mathematical constructions of functions f_0 and f_1.

In 1976, two American mathematicians Whitfield Diffie and Martin Hellman published a number-theoretic construction of functions f_0 and f_1. About the same time, a British mathematician Malcolm J. Williamson, then employed by British Government Communications Headquarters (GCHQ), also devised a similar key exchange scheme. However, Williamson's result was not published because of GCHQ regulations.

In 1977, Ronald Rivest, Adi Shamir, and Leonard Adleman published a number-theoretic-based public-key cryptosystem, which is referred to as the RSA public-key cryptosystem. RSA can be used to encrypt and authenticate data. Rivest, Shamir, and Adleman were awarded the 2002 Turing Award for this work. About the same time, a British mathematician Clifford Cocks, then employed by GCHQ, also devised a similar public-key cryptosystem. However, Cocks's result was not published until 1997 because of GCHQ regulations.

In 1985, Neal Koblitz and Victor Miller proposed, independently, elliptic-curve cryptography (ECC). Its functionalities are similar to those of RSA's.

RSA and ECC both have an encryption algorithm and a decryption algorithm. In PKC we still use E to denote an encryption algorithm and D a decryption algorithm. We use K^u and K^r to denote, respectively, a public key and a private key. We often want a PKC to satisfy the following three criteria:

Forward Efficiency

Encryption $C = E_{K^u}(M)$ and decryption $M = D_{K^r}(C)$ must be easy to compute. Moreover, it must be easy to generate a new key pair (K^u, K^r) so that key pairs may be changed from time to time.

Backward Intractability

It must be computationally intractable to compute M from ciphertext C and public key K^u. In other words, the public key K^u must not leak out any useful information about the corresponding private key K^r.

Commutability

The public key K^u and the private key K^r must satisfy the following equalities:

$$\begin{aligned}
M &= D_{K^r}(E_{K^u}(M)) \\
&= D_{K^u}(E_{K^r}(M)) \\
&= E_{K^u}(D_{K^r}(M)) \\
&= E_{K^r}(D_{K^u}(M)).
\end{aligned}$$

The commutability requirement is needed for data authentications and digital signatures. It is not necessary for key exchange.

Public-key cryptosystems often deal with positive integers, instead of arbitrary binary strings as in conventional encryption algorithms. For any given binary string x, we can always insert a digit 1 in the leftmost position to obtain a positive integer $1x$. Let n be a positive integer. We can divide M into a sequence of blocks such that the length of each block is less than $\log_2 n - 1$. Inserting a digit 1 in the leftmost position of each block yields a positive integer less than n. Without loss of generality, we assume that each plaintext block is a positive integer less than a certain value.

Number theory plays a major role in constructing PKCs. We introduce in the following section some of the fundamental concepts and results in number theory.

3.2 Elementary Concepts and Theorems in Number Theory

Number theory is a mathematical branch devoted to studying properties of integers. Integers are formed from prime numbers, and there are infinitely many prime numbers.

Let $f(n)$ and $g(n)$ be functions from positive integers to positive integers. We write $f(n) \sim g(n)$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1.$$

The followings two results are fundamental theorems of integers:

The fundamental theorem of arithmetic. *Any integer that is greater than 1 is a product of prime numbers. Moreover, this product has a unique representation if prime numbers are listed in nondecreasing order.*

Prime number theorem. *Let n be an integer greater than 1 and $\pi(n)$ be the number of prime numbers that are less than n. Then $\pi(n) \sim n/\ln n$.*

Let n be an integer greater than 1. According to the fundamental theorem of arithmetic, the integer n can be uniquely represented by

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_t^{\alpha_t}, \tag{3.2}$$

where $p_1 < p_2 < \cdots < p_t$ are prime numbers, and each $\alpha_i (i = 1, \cdots, t)$ is a positive integer. For example,

$$85 = 5 \cdot 17,$$
$$1200 = 2^4 \cdot 3 \cdot 5^2,$$
$$11011 = 7 \cdot 11^2 \cdot 13.$$

3.2.1 Modular Arithmetic and Congruence Relations

Throughout this subsection, let a and b be integers and m a positive integer. Recall that $a \bmod m$ represents the remainder of dividing a by m. Let $\lfloor x \rfloor$ denote the largest integer that is less than or equal to x. Then

$$a = \lfloor a/m \rfloor \cdot m + (a \bmod m). \tag{3.3}$$

Modular arithmetic has the following properties:

$$(a + b) \bmod m = (a \bmod m + b \bmod m) \bmod m,$$
$$(a - b) \bmod m = (a \bmod m - b \bmod m) \bmod m,$$
$$(a \times b) \bmod m = (a \bmod m \times b \bmod m) \bmod m.$$

Let b be a given positive integer. Let c and d be two positive integers with $c < d$. Suppose that we want to find an integer $a \in [c, d]$ such that a and b are relatively prime. This can be done efficiently as follows: if b is an even number, then choose at random an odd number $a \in [c, d]$ (likewise, if b is an odd number, then choose at random a number $a \in [c, d]$) and check, using Euclid's algorithm, whether $\gcd(a, b) = 1$. If not, repeat this procedure by selecting a that is not previously chosen until $\gcd(a, b) = 1$.

Congruence is a basic relation between integers. In particular, a is said to be *congruent to b modulo m*, denoted by $a \equiv b \,(\bmod\, m)$, if $a - b$ is divisible by m. In other words, $a \equiv b \,(\bmod\, m)$ if and only if there is an integer k (positive or negative) such that $a = b + m \cdot k$.

For example, $29 \equiv 4 \,(\bmod\, 5)$; $-11 \equiv -4 \,(\bmod\, 7)$; $-4 \equiv 3 \,(\bmod\, 7)$.

3.2.2 Modular Inverse

Let a and n be positive integers with $a < n$. If there is a positive integer $b < n$ such that $a \cdot b \equiv 1 \,(\bmod\, n)$, then we say that b is *a's inverse modulo n*, denoted by $a^{-1} \bmod n$. When there is no confusion about the modulo n, we use a^{-1} to denote a's inverse.

Finding a modular inverse is the basic operation in RSA public-key cryptosystem. However, modular inverse does not always exist. For example, let $a = 2$, $n = 4$, then a does not have a

modular inverse modulo n. This is because for any integer b with $1 \leq b < 4$, we have $2 \cdot b \not\equiv 1$ (mod 4).

Euler's theorem states that if $\gcd(a, n) = 1$, then a's inverse modulo n is guaranteed to exist. Moreover, it has a simpler form using Euler's *totient function* ϕ.

Let n be a positive integer. Euler's totient function $\phi(n)$ is defined to be the number of positive integers that are less than or equal to n and relatively prime to n. For example, $\phi(9) = 6$, because each of $1, 2, 4, 5, 7, 8$ is relatively prime to 9, while 3 and 6 are not.

Write n in the form of Expression 3.2, then

$$\phi(n) = [p_1^{\alpha_1-1}(p_1 - 1)][p_2^{\alpha_2-1}(p_2 - 1)] \cdots [p_t^{\alpha_t-1}(p_t - 1)]. \tag{3.4}$$

For instance, $\phi(72) = \phi(2^3 \cdot 3^2) = 2^{3-1}(2 - 1) \cdot 3^{2-1}(3 - 1) = 4 \cdot 6 = 24$.

Euler's theorem. *Let a be a positive integer and n an integer greater than 1 that is relatively prime to a. Then* $a^{\phi(n)} \equiv 1 \, (mod \, n)$.

It follows from Euler's theorem that $a^{-1} = a^{\phi(n)-1} \bmod n$ if $\gcd(a, n) = 1$ and $n > 1$.

Proof of Euler's theorem. Note that $n > 1$. Let $x_1, x_2, \cdots, x_{\phi(n)}$ be an enumeration of all positive integers less than n that are relatively prime to n. As $\gcd(a, n) = 1$ and $\gcd(x_i, n) = 1$, we have $\gcd(ax_i, n) = 1$. Thus, $\gcd(ax_i \bmod n, n) = 1$. In other words, $ax_i \bmod n$ is equal to x_j for some j. Note that $x_i \neq x_j$ if and only if $ax_i \bmod n \neq ax_j \bmod n$. We have

$$\{ax_1 \bmod n, \; ax_2 \bmod n, \cdots, ax_{\phi(n)} \bmod n\} = \{x_1, x_2, \cdots, x_{\phi(n)}\}.$$

Thus,

$$\prod_{i=1}^{\phi(n)} x_i \equiv \prod_{i=1}^{\phi(n)} ax_i \, (\bmod \, n),$$

$$\prod_{i=1}^{\phi(n)} x_i \equiv a^{\phi(n)} \prod_{i=1}^{\phi(n)} x_i \, (\bmod \, n).$$

This implies that $a^{\phi(n)} \equiv 1 \, (\bmod \, n)$. This completes the proof.

When n is a prime number, Euler's theorem is often referred to as Fermat's little theorem.

Fermat's little theorem. *Let p be a prime number and a a positive number not divisible by p. Then* $a^{p-1} \equiv 1 \, (mod \, p)$.

If follows from Fermat's little theorem that $a^{-1} = a^{p-2} \bmod p$, if $\gcd(a, p) = 1$ and p is a prime number.

In addition to Euler's theorem and Fermat's little theorem, we can also calculate modular inverse using the following two methods:

1. Let a and u be positive integers with $n > 1$. Write $a \cdot u = n + 1$, then $a \cdot u \equiv 1 \, (\bmod \, n)$. Thus, $a^{-1} = u \bmod n$.

   For instance, because $4 \cdot 6 = 24 = 23 + 1$, we know that $4^{-1} \bmod 23 = 6$.

2. Let a and n be positive integers with $n > 1$. Let $A = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & n \end{bmatrix}$. If A can be transformed

to $\begin{bmatrix} u & v & 1 \\ w & x & y \end{bmatrix}$ using elementary matrix transformations, then we have $au + nv = 1$. Thus, $a \cdot u \equiv 1 \pmod{n}$. That is, $a^{-1} = u \bmod n$.

For instance, let $a = 3$ and $n = 5$. Let $A = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \end{bmatrix}$. Multiplying the first row by 2 and

then subtracting the second row, we get $\begin{bmatrix} 2 & -1 & 1 \\ 0 & 1 & 5 \end{bmatrix}$. Thus, $3^{-1} \bmod 5 = 2$.

3.2.3 Primitive Roots

Let a and n be positive integers that are relatively prime, where $n > 1$. It follows from Euler's theorem that $a^{\phi(n)} \equiv 1 \pmod{n}$. If $a^m \not\equiv 1 \pmod{n}$ for any positive integer $m < \phi(n)$, then a is a *primitive root modulo n*. We also say that n has a primitive root a.

For instance, let $n = 2 \cdot 5 = 10$. We have $\phi(10) = 4$ and $9^4 \equiv 1 \pmod{10}$. But 9 is not a primitive root modulo 10, for $9^2 \equiv 1 \pmod{10}$. We can show that 10 has primitive roots 3 and 7.

It is straightforward to verify that if n has a primitive root a, then the following $\phi(n)$ modular exponentiations are different pairwise:

$$a \bmod n, \quad a^2 \bmod n, \cdots, a^{\phi(n)} \bmod n.$$

This is the longest cycle of exponentiations modulo n without repetition and forms a multiplicative group modulo n, denoted by Z_n^*. A primitive root a modulo n is also referred to as a generator of the group Z_n^*.

In particular, if n is prime, denoted by p, and p has a primitive root a, then the following $p - 1$ modular exponentiations modulo p

$$a \bmod p, a^2 \bmod p, \cdots, a^{p-1} \bmod p$$

are different pairwise.

Not every integer n has a primitive root. For instance, 12 does not have a primitive root. It can be shown that only the numbers of the following forms have primitive roots: $2, 4, p^\alpha$, and $2p^\alpha$, where p is an odd prime number and α is a positive integer.

3.2.4 Fast Modular Exponentiation

Let a, x, n be positive integers with $a < n$. Calculating modular exponentiation $a^x \bmod n$ is a common operation in PKC. If we compute $y = a^x$ first and then compute $y \bmod n$, then the time complexity of this computation would be too high, for the value of y may be large. This computation is unnecessary because $a^x \bmod n < n$. Indeed, we can compute $a^x \bmod n$ much more efficiently. To see how this can be done, we first assume that x is a power of 2; namely, $x = 2^m$ for some non-negative integer m. Let $r(m) = a^{2^m} \bmod n$. We have the following recursive relation:

$$r(m) = \begin{cases} r^2(m-1) \bmod n, & \text{if } m > 0, \\ a \bmod n, & \text{if } m = 0. \end{cases}$$

Thus, we can start from $i = 0$, square $r(i)$, and then take the modulo n to yield $r(i+1)$. Repeating this procedure m times, we get $r(m)$. It should be noted that $r(i) < n$ for each i, which means that the squaring operation is always performed on positive integers smaller than n.

Now let us assume that x is an arbitrary positive integer represented in the binary form $b_k \cdots b_1 b_0$, where $k = \lfloor \log_2 x \rfloor, b_i \in \{0, 1\}$. Then

$$x = b_k \cdot 2^k + \cdots + b_1 \cdot 2^1 + b_0 \cdot 2^0 = \sum_{b_i = 1} 2^i.$$

Thus,

$$a^x \bmod n = a^{\left(\sum_{b_i=1} 2^i \right)} \bmod n$$

$$= \left[\prod_{b_i=1} a^{2^i} \right] \bmod n$$

$$= \left[\prod_{b_i=1} (a^{2^i} \bmod n) \right] \bmod n.$$

On the basis of this, we can derive the following fast modular exponentiation algorithm, where g_0 (the output of the algorithm) is equal to $a^x \bmod n$:

1. Let $g_k = a$.
2. For each integer i from $k - 1$ down to 0:
3. Let $g_i = (g_{i+1} \times g_{i+1}) \bmod n$;
4. If $b_i = 1$ let $g_i = (g_i \times a) \bmod n$.

This algorithm will undergo at most $2k = 2 \lfloor \log_2 x \rfloor$ multiplication operations on positive integers that are less than n.

For instance, let $x = 37$. Converting it to binary representation, we get $x = 100101$. Thus,

$$a^{37} \bmod n = (a^{2^5} \cdot a^{2^2} \cdot a) \bmod n = [(a^{2^3} \cdot a)^{2^2} \cdot a] \bmod n.$$

Applying the fast modular exponentiation algorithm, we get

$$g_5 = a \bmod n,$$
$$g_4 = a^2 \bmod n,$$
$$g_3 = g_4^2 \bmod n = a^{2^2} \bmod n,$$
$$g_2 = ((g_3^2 \bmod n) \cdot a) \bmod n = a^{2^3} \cdot a \bmod n,$$
$$g_1 = g_2^2 \bmod n = (a^{2^3} \cdot a)^2 \bmod n,$$
$$g_0 = [(g_1^2 \bmod n) \cdot a] \bmod n = [(a^{2^3} \cdot a)^{2^2} \cdot a] \bmod n.$$

Therefore, $g_0 = a^{37} \bmod n$. Let $a = 7$ and $n = 11$. Then $g_5 = 7, g_4 = 7^2 \bmod 11 = 5$, $g_3 = 5^2 \bmod 11 = 3, g_2 = [(3^2 \bmod 11) \cdot 7] \bmod 11 = 8$, and $g_1 = 8^2 \bmod 11 = 9$. Hence,

$$7^{37} \bmod 11 = [(9^2 \bmod 11) \cdot 7] \bmod 11 = 4 \cdot 7 \bmod 11 = 6.$$

3.2.5   Finding Large Prime Numbers

Constructing a PKC often needs to use large prime numbers that consist of hundreds of bits. Finding a prime number in a given range can be done by checking each odd number in the range until a prime number is found. For example, we can find a k-bit prime number by checking k-bit odd numbers one at a time until a prime number is found. Such binary numbers can be represented as a regular expression of $1(0+1)^{k-2}1$, where $(0+1)^{\ell}$ denotes any ℓ-bit binary string. It follows from the prime number theorem that we can expect at least one prime number among any $\ln 2^{k+1} = (k+1) \ln 2$ consecutive k-bit binary positive integers. Thus, we could find a k-bit prime number by checking $(k+1) \ln 2/2$ many k-bit positive odd numbers. For instance, if $k = 300$, then we may be able to find a prime number by only checking about $301 \cdot \ln 2/2 < 105$ many 300-bit consecutive positive odd numbers. Such computations can be carried out easily.

Therefore, the primary task becomes how to efficiently determine whether a given odd number n is prime. One way to do this is to use the classic sieve, which checks whether n has a factor that is greater than 1 but less than or equal to $\sqrt{n}$. However, the time complexity of a sieve is equal to $O(\sqrt{n}) = O(2^{\frac{1}{2}\log n})$, which is exponential of the length of n. So when n is large, using sieve is not practical.

There exists a polynomial-time algorithm to determine whether a given integer is a prime number, but it takes much more time to run than Miller–Rabin's *primality test*. Miller–Rabin's primality test is a probabilistic algorithm that uses the following property of prime numbers to determine whether a given integer is prime with large probability:

> *Let p be an odd prime number. It follows from the fundamental theorem of arithmetic that there is a positive integer k such that $p - 1 = 2^k q$, where q is an odd number. Let a be an integer with $1 < a < p - 1$. Then either $a^q \bmod p = 1$ or there is a non-negative integer $j < k$ such that $a^{2^j q} \bmod p = -1$.*

This property can be proven as follows. From Fermat's little theorem, we know that $a^{p-1} \bmod p = 1$. That is, $a^{2^k q} \bmod p = 1$. Let us consider the following sequence of integers:

$$a^{2^0 q} \bmod p, \ a^{2^1 q} \bmod p, \cdots, a^{2^{k-1} q} \bmod p, \ a^{2^k q} \bmod p,$$

where the last integer is equal to 1 and each number in the sequence is a square of the previous number modulo p, that is,

$$a^{2^j q} \bmod p = [a^{2^{j-1} q}]^2 \bmod p = [a^{2^{j-1} q} \bmod p]^2 \bmod p.$$

Thus, if $a^q \bmod p = 1$, then every integer in this sequence is equal to 1. If $a^q \bmod p \neq 1$, then as $a^{2^k q} \bmod p = 1$, there must be a non-negative integer $j < k$ such that $a^{2^j q} \bmod p \neq 1$ and $a^{2^{j+1} q} \bmod p = 1$. As $[a^{2^j q} \bmod p]^2 \bmod p = a^{2^{j+1} q} \bmod p = 1$, we must have $a^{2^j q} \bmod p = -1$.

Thus, if there is an integer a with $1 < a < n - 1$ such that $a^q \bmod n \neq 1$ and $a^{2^j q} \bmod n \neq -1$ for all j from 1 to $k - 1$, then n is not prime.

3.2.5.1   Miller–Rabin's Primality Test

Let n be an odd number greater than 1. Let k be an integer in the expression $n - 1 = 2^k q$, where q is an odd number.

1. Choose at random an integer a with $1 < a < n - 1$.
2. If $a^q \bmod n \neq 1$ and $a^{2^j q} \bmod n \neq -1$ for all j with $1 \leq j \leq k - 1$, then output "n is not prime" and halt.
3. Otherwise, output "n is likely to be prime". Then choose at random another integer a with $1 < a < n - 1$, and repeat Step 2.

As long as it does not halt, that is, as long as it does not output "n is not prime," this procedure is repeated for m times, where m is a positive integer.

Miller–Rabin's primality test returns false information only when it outputs "n is likely to be prime" while n is not a prime number. It can be shown that the probability that this can happen is less than 2^{-2m}. If $m = 20$, then this probability is less than $2^{-40} < 10^{-12}$. Therefore, if one runs Miller–Rabin's primality test with a sufficiently large integer m and the test still does not halt, then n will very likely be prime.

3.2.6 The Chinese Remainder Theorem

The Chinese remainder theorem, first studied by the ancient Chinese, finds a solution to a set of simultaneous congruence equations.

Let i be a positive integer. Let $Z_i = \{0, \cdots, i - 1\}$. Let $n_1, n_2, \cdots, n_k$ be positive integers that are pairwise relatively prime. That is, $\gcd(n_i, n_j) = 1$ for all $i \neq j$ with $1 \leq i, j \leq k$. Let $n = n_1 \times n_2 \times \cdots \times n_k$.

The Chinese remainder theorem.. *For any given set of simultaneous congruence equations* $x \equiv a_i \,(\bmod\ n_i)$, *where* $i = 1, \cdots, k$, *it has the following unique solution in* Z_n:

$$x = \left(\sum_{i=1}^{k} a_i b_i \right) \bmod n,$$

where $b_i = m_i(m_i^{-1} \bmod n_i)$ *and* $m_i = n/n_i$.

Note that $m_i^{-1} \bmod n_i$ exists because m_i and n_i are relatively prime. As an example of using the Chinese remainder theorem, let $(n_1, n_2, n_3) = (3, 5, 7)$, $n = 3 \times 5 \times 7 = 105$. Then the following set of simultaneous congruence equations of $x \equiv 2 \,(\bmod\ 3), x \equiv 3 \,(\bmod\ 5)$, and $x \equiv 2 \,(\bmod\ 7)$ has the unique solution $x = 23$ in Z_{105} for the following reasons: $m_1 = 5 \times 7 = 35, m_2 = 3 \times 7 = 21$, and $m_3 = 3 \times 5 = 15$. Thus,

$$m_1^{-1} \bmod n_1 = 35^{-1} \bmod 3 = 2,$$
$$m_2^{-1} \bmod n_2 = 21^{-1} \bmod 5 = 1,$$
$$m_3^{-1} \bmod n_3 = 15^{-1} \bmod 7 = 1.$$

This implies that $b_1 = 35 \times 2 = 70$, $b_2 = 21 \times 1 = 21$, and $b_3 = 15 \times 1 = 15$. Hence,

$$(2 \times 70 + 3 \times 21 + 2 \times 15) \bmod 105 = (35 + 63 + 30) \bmod 105 = 23.$$

The Chinese remainder theorem has a special form for the case of $a_1 = a_2 = \cdots = a_k = a$. It states that for all integers x and a, if $x \equiv a \,(\bmod\ n_i)$ for $i = 1, \cdots, k$, then $x \equiv a \,(\bmod\ n)$.

This result can be proven as follows. As $x \equiv a \,(\bmod\ n_i)$ for all i from 1 to k, we know that $x - a$ is divisible by each of these n_i's. As n_i's are pairwise relatively prime, $x - a$ must be divisible by the product of these n_i's. Namely, $x \equiv a \,(\bmod\ n)$.

3.2.7   Finite Continued Fractions

Finite continued fractions are fractional numbers of the following form:

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{\ddots}{\quad + \cfrac{1}{a_k}}}}}$$

where a_0 is an integer (which could be zero), and $a_1 \cdots, a_k$ are nonzero integers. It is custom-
ary to use $[a_0; a_1, \cdots, a_k]$ to denote such a finite continued fraction.

   Continued fractions are representations of real numbers. Given a real number x, we can
construct a continued fraction (possibly infinite) to represent x as follows.

3.2.7.1   Construction of Continued Fractions

1. Set $x_0 \leftarrow x, a_0 \leftarrow \lfloor x_0 \rfloor, \; i \leftarrow 0$.
2. If $x_i = a_i$, then halt. Otherwise, set
3. $x_{i+1} \leftarrow \frac{1}{x_i - a_i}$,
4. $a_{i+1} \leftarrow \lfloor x_{i+1} \rfloor$.
5. Set $i \leftarrow i + 1$ and goto Step 2.

If this algorithm generates a finite sequence $a_0, \; a_1, \cdots, a_k$, then

$$[a_0; a_1, \cdots, a_k] = x,$$

and so x is a rational number. On the other hand, if x is a rational number, then the algo-
rithm will halt, producing a finite continued fraction $[a_0; a_1, \cdots, a_k]$ for some k such that $x =
[a_0; a_1, \cdots, a_k]$. If we let $x = m/n$ (where m and n are nonzero integers with $\gcd(m, n) = 1$),
then it can be shown that $k \leq \log_2 n$.

   If the algorithm generates an infinite sequence $a_0, a_1, \cdots$, then it can be shown that for any
$k > 1$ and any $1 \leq j < k$, the finite prefix $[a_0; a_1, \cdots, a_k]$ is closer to x than $[a_0; a_1, \cdots, a_j]$.
The following are two examples that represent nonrational numbers as infinite continued
fractions:

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \ddots}}}} = [1; 2, 2, 2, 2, \cdots],$$

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \ddots}}}} = [3; 7, 15, 1, 292, \cdots].$$

The following theorem states that if a rational number y is sufficiently close to x, then y must be a prefix of x's continued fraction representation.

Finite continued fraction approximation theorem. *Let x be a real number and $[a_0;$ $a_1, \cdots]$ a continued fraction representation of x that may or may not be infinite. If there is a non-negative integer r and a positive integer s such that $|x - \dfrac{r}{s}| < (\sqrt{2}s)^{-2}$, then there must be k such that $\dfrac{r}{s} = [a_0; a_1, \cdots, a_k]$.*

3.3 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange protocol provides a concrete construction of functions f_0 and f_1 defined in Section 3.1. It uses primitive roots and modular exponentiation operations. In particular, it uses two global parameters (p,a) known to all parties involved in the communication, where p is a large prime number and a is a primitive root modulo p. The functions f_0 and f_1 are defined as follows:

$$f_0(p, a; x) = a^x \bmod p,$$
$$f_1(x, b) = x^b \bmod p,$$

where x and b are positive integers. We have

$$f_1(f_0(p, a; y), x) = (a^y \bmod p)^x \bmod p = a^{yx} \bmod p = f_0(p, a; x \cdot y),$$
$$f_1(f_0(p, a; x), y) = (a^x \bmod p)^y \bmod p = a^{xy} \bmod p = f_0(p, a; x \cdot y).$$

Thus, $f_1(f_0(p, a; y), x) = f_1(f_0(p, a; x), y)$. This shows that the functions f_0 and f_1 satisfy Equality 3.1.

3.3.1 Key Exchange Protocol

Diffie-Hellman key exchange allows Alice and Bob to create a common secret key without sharing prior secrets. The algorithm proceeds as follows: Alice randomly selects a positive number $X_A < p$ as her private key and calculates

$$Y_A = f_0(p, a; X_A) = a^{X_A} \bmod p$$

as her public key. In the meantime, Bob randomly selects a positive number $X_B < p$ as his private key and calculates

$$Y_B = f_0(p, a; X_B) = a^{X_B} \bmod p$$

as his public key. Alice then sends Y_A to Bob and Bob sends Y_B to Alice. Finally, both Alice and Bob independently perform secret key calculations. Alice calculates $K_A = f_1(Y_B, X_A) = Y_B^{X_A} \bmod p$, and Bob calculates $K_B = f_1(Y_A, X_B) = Y_A^{X_B} \bmod p$. We have already shown in Section 3.1 that $K_A = K_B$. Therefore, Alice and Bob now share the same secret key $K = K_A = K_B$.

For example, let the global parameters be $(p, a) = (541, 2)$. Suppose that Alice selects her private key to be $X_A = 137$ and sends her public key $Y_A = 2^{137} \bmod 541 = 208$ to Bob. Suppose that Bob selects his private key to be $X_B = 193$ and sends his public key $Y_B = 2^{193} \bmod 541 = 195$ to Alice. Next, Alice and Bob calculate, respectively,

$$K_A = Y_B^{X_A} \bmod 541 = (195)^{137} \bmod 541 = 486,$$
$$K_B = Y_A^{X_B} \bmod 541 = (208)^{193} \bmod 541 = 486,$$

to obtain the same secret key $K = 486$.

As modular exponentiations can be calculated using the fast modular exponentiation algorithm, Diffie-Hellman key exchange satisfies the efficiency requirement. Its intractability relies on the difficulty of solving x from $y = a^x \bmod p$, where $x < p$. This problem is referred to as the *discrete logarithm* problem, or the *discrete log* problem for short. It is a common belief that discrete log cannot be solved in polynomial time on conventional computing devices. Thus, as long as the prime number p is sufficiently large, Diffie-Hellman key exchange is considered secure. On the other hand, Peter Shor, an American scientist, showed in 1994 that discrete log can be solved in polynomial time on a theoretical model of quantum computers. Discrete log will be used again in Chapter 4 when we discuss digital signatures.

3.3.2 Man-in-the-Middle Attacks

Suppose that Malice eavesdrops Y_A or Y_B. As there are no known efficient algorithms that can solve discrete log, Malice has no ways to solve X_A or X_B. However, Malice could launch a man-in-the-middle attack to establish a key K_{mA} with Alice and a key K_{mB} with Bob, while Alice thinks that she shares K_{mA} with Bob and Bob thinks that he shares K_{mB} with Alice. Therefore, when Alice encrypts data using K_{mA}, Malice can decrypt it. Likewise, when Bob encrypts data using K_{mB}, Malice can decrypt it.
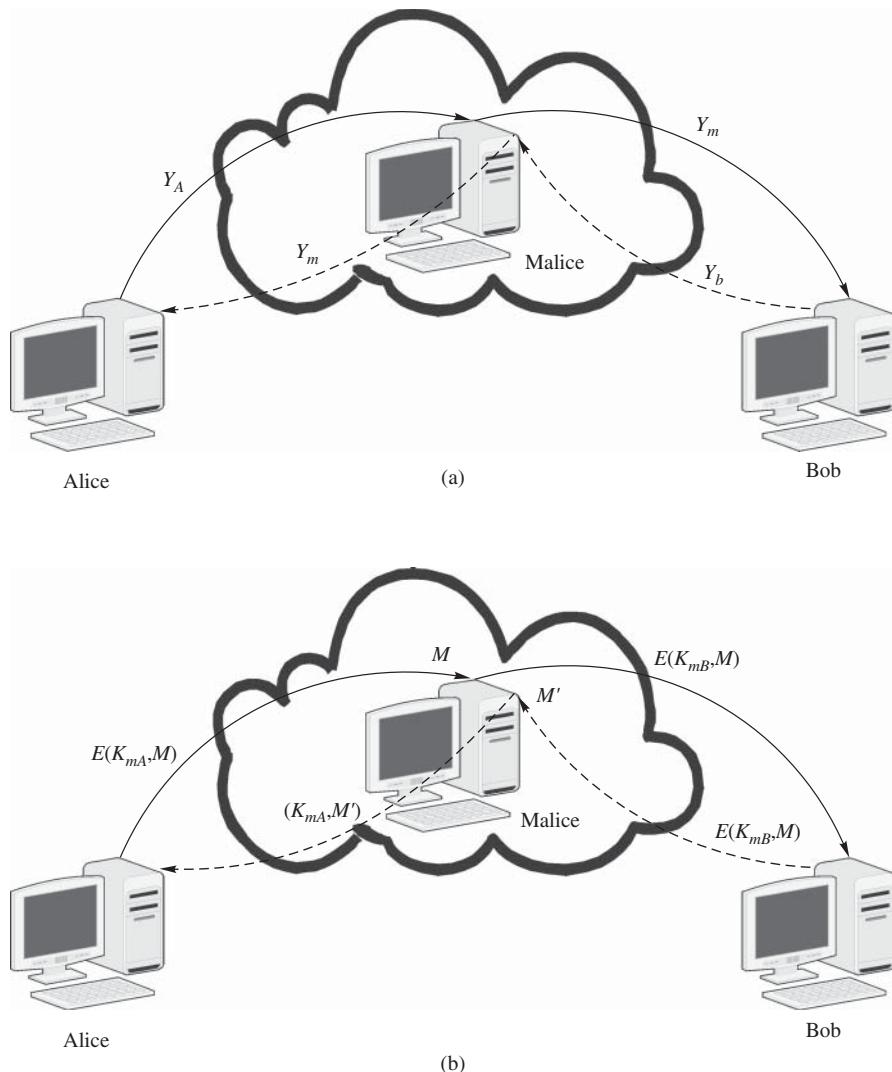
To launch a man-in-the-middle attack, Malice selects at random a positive integer $X_m < p$ and calculates $Y_m = a^{X_m} \bmod p$. Malice places a sniffer on the communication channel used by Alice and Bob. When she intercepts Y_A sent from Alice to Bob and Y_B sent from Bob to Alice, Malice sends Y_m to Bob as it were Y_A and sends Y_m to Alice as it were Y_B (see Figure 3.1).

Suppose that Alice does not know that Y_B has been replaced with Y_m and Bob does not know that Y_A has been replaced with Y_m. Following the Diffie-Hellman key exchange protocol, Alice and Bob proceed to compute, respectively,

$$K_A = Y_m^{X_A} \bmod p = a^{X_m \cdot X_B} \bmod p,$$
$$K_B = Y_m^{X_B} \bmod p = a^{Y_m \cdot X_B} \bmod p.$$

Malice computes

$$K_{mA} = Y_A^{X_m} \bmod p = a^{X_A \cdot X_m} \bmod p = K_A,$$
$$K_{mB} = Y_B^{X_m} \bmod p = a^{X_B \cdot X_m} \bmod p = K_B.$$

Figure 3.1   Man-in-the-middle attack being carried out on the Diffie-Hellman key exchange. (a) Malice intercepts Alice's public key Y_A and Bob's public key Y_B. Malice then sends Y_m to Bob as if it were Y_A and sends Y_m to Alice as if it were Y_B. (b) Because Alice shares K_{mA} with Malice and Bob shares K_{mB} with Malice, Malice can read encrypted data encrypted using K_{mA} or K_{mB}

Thus, Alice and Bob have not established any common secret key. Instead, Malice has established with Alice a secret key $K_{mA} = K_A$ and established with Bob a secret key $K_{mB} = K_B$. When Alice encrypts M using encryption key K_A and sends the ciphertext $C_A = E_{K_A}(M)$ to Bob, Malice intercepts and decrypts it using K_{mA} to obtain M. Malice then uses K_{mB} to

encrypt M and sends the ciphertext $C_m = E_{K_{m_B}}(M)$ to Bob. Bob then uses K_B to decrypt C_m and get M. Likewise, Malice can read any encrypted message sent from Bob encrypted using K_B. In addition, Malice may also modify or fabricate M.

We note that this man-in-the-middle attack will fail if Alice and Bob can authenticate each other. The RSA public-key cryptosystem introduced in Section 3.4 provides a mechanism for authenticating users.

3.3.3 Elgamal PKC

Taher Elgamal, an Egyptian-American cryptographer, devised in 1985 a public-key cryptosystem on the basis of the Diffie-Hellman key exchange protocol. It is often referred to as *Elgamal PKC*. Note that "Elgamal" has also been written as "ElGamal" (with a capital G in the middle). Elgamal PKC uses two global parameters p and a, just like those used in the Diffie-Hellman key exchange protocol, where p is a prime number and a is a primitive root modulo p.

Alice randomly selects a positive integer $X_A < p$ as her private key and calculates $Y_A = a^{X_A} \bmod p$ as her public key.

Bob randomly selects a positive integer $X_B < p$ as his private key and calculates $Y_B = a^{X_B} \bmod p$ as his public key.

Let M be a positive integer less than p that represents a block to be encrypted. Alice encrypts M as follows:

1. Select a positive integer k at random with $k < p$;
2. Compute $K = (Y_B)^k \bmod p$;
3. Compute $C_1 = a^k \bmod p$, $C_2 = (K \cdot M) \bmod p$, and send (C_1, C_2) to Bob.

After receiving (C_1, C_2), Bob can decrypt it by calculating

$$M = (C_2 \cdot (C_1^{X_B} \bmod p)^{-1}) \bmod p. \tag{3.5}$$

The proof of its correctness is left to the reader (see Exercise 3.13).

3.4 RSA Cryptosystem

The basic operation of the RSA public-key cryptosystem is modular exponentiation. Decryption takes place by finding the modular inverse.

3.4.1 RSA Key Pairs, Encryptions, and Decryptions

Suppose that Alice wants to set up an RSA cryptosystem. She first selects two large prime numbers p and q, and calculates $n = p \cdot q$. She then selects a positive integer d such that $1 < d < \phi(n)$ and $\gcd(d, \phi(n)) = 1$. Finally, she computes the inverse of d modulo $\phi(n)$, denoted by e. That is, she finds e such that $de \equiv 1 \pmod{\phi(n)}$. Alice publishes (e,n) as her public key. She keeps d, p, q, and $\phi(n)$ secret and uses (d,n) as her private key.

Suppose that Bob wants to encrypt M (some positive integer less than n) and send the encrypted message to Alice without sharing a prior secret key with Alice. Bob uses RSA

encryption to encrypt M as follows:

$$\text{RSA encryption}: C = M^e \bmod n. \tag{3.6}$$

After receiving C, Alice uses RSA decryption to decrypt C as follows:

$$\text{RSA decryption}: M = C^d \bmod n. \tag{3.7}$$

We now prove the correctness of RSA decryption; namely, we want to show that Equality 3.7 is true. We present two different proofs. One proof uses the Chinese remainder theorem, while the other does not use it.

Proof 1: We prove Equality 3.7 using the Chinese remainder theorem. As $n = p \cdot q$, we have $\phi(n) = (p-1) \cdot (q-1)$. It follows from $de \equiv 1 \pmod{\phi(n)}$ that there is an integer k such that

$$de = k \cdot \phi(n) + 1.$$

Case 1: M is not divisible by p, that is, $\gcd(M, p) = 1$. It follows from Fermat's little theorem that $M^{p-1} \equiv 1 \pmod{p}$. Hence,

$$M^{de} \equiv M^{k\phi(n)} M \equiv (M^{p-1})^{k(q-1)} M \equiv (1)^{k(q-1)} M \equiv M \pmod{p}.$$

Case 2: M is divisible by p, that is, $\gcd(M, p) = p$. Then $M \equiv 0 \pmod{p}$ and $M^{de} \equiv 0 \pmod{p}$. This implies that $M^{de} \equiv M \pmod{p}$. Hence, we always have $M^{de} \equiv M \pmod{p}$.

Likewise, we can show that $M^{de} \equiv M \pmod{q}$. By the special form of the Chinese remainder theorem, we have $M^{de} \equiv M \pmod{pq}$. As $M < n$, we have $M^{de} \bmod n = M$. This completes the proof.

Proof 2: If $\gcd(n, M) = 1$, then M is not divisible by p or q. Hence, by Euclid's theorem, we have $M^{\phi(n)} \bmod n = 1$. This implies that

$$M^{k\phi(n)} \bmod n = \left(M^{\phi(n)} \bmod n\right)^k \bmod n = 1^k \bmod n = 1.$$

Thus,

$$
\begin{aligned}
C^d \bmod n &= (M^e \bmod n)^d \bmod n \\
&= M^{ed} \bmod n \\
&= M^{k\phi(n)+1} \bmod n \\
&= \left[\left(M^{k\phi(n)} \bmod n\right) \cdot M \bmod n\right] \bmod n \\
&= (1 \cdot M \bmod n) \bmod n \\
&= M.
\end{aligned}
$$

If $\gcd(n, M) \neq 1$, then it follows from $M < n$ and $n = p \cdot q$ that M is divisible either by p or by q, but not by both p and q (otherwise, $M \geq n$). Without loss of generality, assume that M is divisible by p (the case that M is divisible by q is similar). This implies that M is not

divisible by q. That is, $M = l \cdot p$ for a positive integer l and $\gcd(M, q) = 1$. By Fermat's little theorem, we have $M^{q-1} \bmod q = 1$. Thus,

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

$$= 1^{k(p-1)} \bmod q$$

$$= 1.$$

Therefore, there is an integer u such that $M^{k\phi(n)} = 1 + u \cdot q$. As $M = \ell \cdot p$, we have

$$M^{k\phi(n)+1} = M + M \cdot u \cdot q = M + l \cdot u \cdot p \cdot q = M + l \cdot u \cdot n \equiv M \bmod n,$$

namely, $M^{k\phi(n)+1} \bmod n = M$. Thus,

$$C^d \bmod n = M^{k\phi(n)+1} \bmod n = M.$$

This completes the proof.

For example, let $p = 13, \; q = 19$. Then

$$n = p \cdot q = 247,$$

$$\phi(n) = 12 \cdot 18 = 216.$$

Choose $d = 173$ and compute $e = 5$, where $de = 865 \equiv 1 \bmod 216$. Let $M = 85$. We have

$$C = M^e \bmod n = 85^5 \bmod 247$$

$$= [((85^2 \bmod 247)^2 \bmod 247) \cdot 85] \bmod 247$$

$$= ((62^2 \bmod 247) \cdot 85) \bmod 247 = (139 \cdot 85) \bmod 247$$

$$= 206.$$

On the other hand, $C^d \bmod n = 206^{173} \bmod 247$. As

$$173 = 2^7 + 2^5 + 2^3 + 2^2 + 1,$$

we have

$$C^{173} \bmod n = ((((C^{2^2} \cdot C)^{2^2} \cdot C)^2 \cdot C)^{2^2} \cdot C) \bmod n.$$

Applying the fast modular exponentiation algorithm, we get

$$g_7 = C = 206,$$

$$g_6 = g_7^2 \bmod n = 206^2 \bmod 247 = 199,$$

$$g_5 = (g_6^2 \bmod n \cdot C) \bmod n = ((199^2 \bmod 247) \cdot 206) \bmod n = 137,$$

$$g_4 = g_5^2 \bmod n = 137^2 \bmod 247 = 244,$$

$$g_3 = ((g_4^2 \bmod n) \cdot C) \bmod n = ((244^2 \bmod 247) \cdot 206) \bmod 247 = 125,$$

$$g_2 = ((g_3^2 \bmod n) \cdot C) \bmod n = ((125^2 \bmod 247) \cdot 206) \bmod 247 = 93,$$

$$g_1 = g_2^2 \bmod n = 93^2 \bmod 247 = 4,$$

$$g_0 = ((g_1^2 \bmod n) \cdot C) \bmod n = ((4^2 \bmod 247) \cdot 206) \bmod 247 = 85 = M.$$

From Section 3.2.5, we know that finding large prime numbers p and q can be done efficiently. Once p and q are determined, finding a positive integer d such that $\gcd(d, \phi(n)) = 1$ is straightforward using Euclid's algorithm. From the discussions in Sections 3.2.2 and 3.2.3, we can find $e = d^{-1} \bmod \phi(n)$ efficiently. Using the fast modular exponentiation algorithm, we can carry out RSA encryption and decryption efficiently. Thus, RSA satisfies the efficiency requirement.

RSA also satisfies the commutability requirement because

$$M = D_{d,n}(E_{e,n}(M)) = (M^e \bmod n)^d \bmod n$$
$$= E_{d,n}(D_{e,n}(M)) = (M^d \bmod n)^e \bmod n$$
$$= D_{e,n}(E_{d,n}(M))$$
$$= E_{e,n}(D_{d,n}(M)).$$

The intractability of RSA depends on the difficulty of integer factorization discussed in Section 2.8.2. It is a consensus that if RSA parameters p, q, d are appropriately selected and changed from time to time, then RSA cryptosystem is secure. How to select these parameters appropriately to avoid possible attacks is an important issue. We discuss several common parameter attacks in the following section.

3.4.2   RSA Parameter Attacks

This section discusses several common attacks on RSA that take advantage of inappropriately selected parameters. This will serve as a guideline for choosing correct RSA parameters. Attacks against RSA may use the following methods:

1. Try all possible parameters d to decrypt an encrypted block.
2. Factor n.
3. Conduct time analysis to find d.
4. Derive RSA parameters from partial information of these parameters.

The first method is a brute-force method, which is infeasible when n and d are sufficiently large.

How to factor n efficiently is a long-standing open problem. Despite intensive efforts, it is still not known whether integer factorization can be solved in polynomial time on a conventional computer.

Time analysis on RSA execution is possible because the execution time of modular exponentiation differs a great deal on the basis of the current bit in the exponent. This difference may be exploited to deduce d. In particular, under fast modular exponentiation, d is represented as a binary string $d_k \cdots d_1 d_0$. Its execution time on $d_i = 1$ is substantially more than that on $d_i = 0$. If this difference is measurable, then d could be derived. Executing a few redundant instructions when $d_i = 0$ could thwart this time analysis.

We discuss several methods that may breach RSA security if partial information of RSA parameters is known.

3.4.2.1 Small Exponent Attacks

Small values of e and d should be avoided. For example, assume that Alice and Bob happen to use the same value of $e = 2$, but with different values of n_A and n_B, where n_A and n_B happen to be relatively prime. Suppose that Charlie wants to send the same message M to Alice and Bob using their public keys to encrypt them, where $M < \min\{n_A, n_B\}$. Namely, Charlie sends $C_A = M^2 \bmod n_A$ to Alice and $C_B = M^2 \bmod n_B$ to Bob. If Malice intercepts C_A and C_B, then she can use the Chinese remainder theorem to solve the following two simultaneous congruences:

$$x \equiv C_A \,(\bmod\ n_A),$$
$$x \equiv C_B \,(\bmod\ n_B).$$

Let $x_0 \in Z_n$ be a solution, where $n = n_A n_B$. Then $x_0 = M^2 \bmod n$. As $M^2 < n$, we have $x_0 = M^2$, and so $M = \sqrt{x_0}$.

For another example, assume that $d < \frac{1}{3} n^{1/4}$ and $q < p < 2q$, then one can compute d in polynomial time of $\log_2 n$ as follows:

We note that $q^2 < p \cdot q = n$, which means that $q < \sqrt{n}$. As

$$n - \phi(n) = p \cdot q - (p-1)(q-1) = p + q - 1$$

and $q < p < 2q$, we have

$$4 \leq n - \phi(n) < 3q < 3\sqrt{n}.$$

From $de \equiv 1 \,(\bmod\ \phi(n))$, we know that there is a positive integer k such that $de = k\phi(n) + 1$. As $e < \phi(n)$, we have

$$\phi(n)k < de < \frac{1}{3}\phi(n)n^{1/4}.$$

This implies that $k < \frac{1}{3} n^{1/4}$. It follows from $kn - de = k(n - \phi(n)) - 1$ that

$$0 < kn - de < k(n - \phi(n)) < \frac{1}{3} n^{1/4}(3\sqrt{n}) = n^{3/4}.$$

Dividing both sides of this inequality by dn, we get

$$0 < \frac{k}{d} - \frac{e}{n} < \frac{1}{dn^{1/4}} < \frac{1}{3d^2} < \frac{1}{2d^2}.$$

Hence, $| \, e/n - k/d \, | < (\sqrt{2}d)^{-2}$.

By the finite continued fraction approximation theorem (see Section 3.2.7), we know that k/d is a prefix of e/n's continued fraction. Namely, if $e/n = [a_0; a_1, \cdots, a_m]$, then there is a positive integer $j \leq m$ such that

$$k/d = [a_0; a_1, \cdots, a_j].$$

We can compute $[a_0; a_1, \cdots, a_i], i = 1, \cdots, m$, using the continued fraction construction algorithm introduced in Section 3.2.7 in polynomial time of $\log_2 n$. Let $A_i/B_i = [a_0; a_1, \cdots, a_i]$. Then A_i/B_i is a candidate of k/d. To determine whether $A_i/B_i = k/d$, we first verify whether $C_i = (eB_i - 1)/A_i$ is an integer. If not, we choose the next i. If $C_i = (eB_i - 1)/A_i$ is an

integer, then it is possible that $A_i = k$ and $B_i = d$. To verify whether this is the case, we solve the following quadratic equation:

$$x^2 - (n - C_i + 1)x + n = 0. \tag{3.8}$$

As $C_i > 0$, solutions to Equation 3.8 cannot be equal to 1 or n. Let r_1, r_2 be two solutions to Equation 3.8 . That is,

$$x^2 - (n - C_i + 1)x + n = (x - r_1)(x - r_2).$$

If we let $x = 0$, then we get

$$n = r_1 \cdot r_2.$$

If both r_1 and r_2 are integers, then we get $\{r_1, r_2\} = \{p, q\}$, from which we can calculate $\phi(n)$. From $\phi(n)$ and e, we can calculate d using Euclid's algorithm. If r_1 or r_2 is not an integer, we then choose the next i and repeat the aforementioned steps until d is found. Note that solving Equation 3.8 can be done in polynomial time of $\log_2 n$. Thus, we can find d in polynomial time of $\log_2 n$.

3.4.2.2   Partial Information Attacks

When partial information of parameters p, q, and d leaks out, we must select new parameters, because partial information of these parameters may be exploited by attackers. For example, let the length of the decimal representation of n be m. It can be shown that if the prefix (or suffix) $m/4$ bits of p (or q) leak out, then n can be factored efficiently. For another example, it can be shown that if $m/4$ bits in the suffix of d leak out, then d can be found efficiently.

  If the parameter d is compromised, then we must not use the original secret parameters p and q to generate a new pair of d and e, for p and q may no longer be secret. This is because from the compromised d and the corresponding e, we have

$$de \equiv 1 \, (\bmod \, \phi(n)).$$

Thus, there is a positive integer k such that $de - 1 = k\phi(n)$, from which we can factor n as follows:

  Let a be an arbitrary positive integer less than n. Compute $\gcd(a, n)$ using Euclid's algorithm. If $\gcd(a, n) > 1$, then we know that $\gcd(a, n) \in \{p, q\}$. If $\gcd(a, n) = 1$, then it follows from Euclid's theorem $a^{de-1} \equiv 1 \, (\bmod \, n)$. Let $u = de - 1$. Note that $\phi(n)$ is an even number. Thus, $u = k\phi(n)$ must be an even number. We have

$$(a^{u/2} + 1)(a^{u/2} - 1) \equiv 0 \, (\bmod \, n). \tag{3.9}$$

If $a^{u/2} \not\equiv \pm 1 \, (\bmod \, n)$, then gcd $((a^{u/2} + 1) \bmod n, n)$ or $\gcd((a^{u/2} - 1) \bmod n, n)$ is a prime factor of n. Otherwise, we have the following three cases:

Case 1: $a^{u/2} \equiv -1 \, (\bmod \, n)$. That is, $a^{u/2} \equiv (n - 1) \, (\bmod \, n)$ and $(a^{u/2} + 1) \bmod n = 0$.
  Choose a different value for a and repeat the aforementioned procedure.
Case 2: $a^{u/2} \equiv 1 \, (\bmod \, n)$ and $u/2$ is an odd number. Choose a different value for a and repeat the aforementioned procedure.
Case 3: $a^{u/2} \equiv 1 \, (\bmod \, n)$ and $u/2$ is an even number. Set $u \leftarrow u/2$ and start from Equality 3.9 until case 1 or case 2 occurs.

3.4.2.3 Other Attacks

We should avoid encrypting plaintext M that contains a prime factor p or q. This is because if $M < n$ and n is not relative prime, then n can be factored efficiently. Without loss of generality, assume that $\gcd(M, n) = p$. Thus, $\gcd(C, n) = p$. Using Euclid's algorithm, we can calculate p efficiently.

If M is short and M is a product of two integers whose lengths are close to each other, then Malice can use meet-in-the-middle attack to compute M. For instance, let the length of binary representation of M be ℓ and $M = m_1 \cdot m_2$, where m_1 and m_2 are two integers, and the length of m_1 and the length of m_2 are all less than or equal to $\ell/2$. Malice intercepts $C = M^e \bmod n$, computes the following two arrays of values, and then sorts them into nondecreasing order:

1. Array 1: For each positive integer $x \le 2^{\ell/2+1}$, compute $Cx^{-e} \ (\bmod\ n)$.
2. Array 2: For each positive integer $y \le 2^{\ell/2+1}$, compute $y^e \ (\bmod\ n)$.

If there are integers x and y such that

$$Cx^{-e} \ (\bmod\ n) = y^e \ (\bmod\ n),$$

then $C \equiv (xy)^e \ (\bmod\ n)$. Thus, $M \equiv C^{-e} \equiv xy \ (\bmod\ n)$.

The time of complexity of this attack is in the order of $2^{\ell/2+2}$, which is much smaller than the complexity of 2^ℓ in a brute-force attack. For instance, if M is a 128-bit encryption key that is a product of two 64-bit integers, then Malice can compute M using meet-in-the-middle attack in the order of 2^{66} time. A simple way to combat meet-in-the-middle attack is to break up the product. For example, one may throw in a few useless symbols at the beginning or at the end of the plaintext message so that the new string cannot be written as the product of two integers that have about the same length.

3.4.3 RSA Challenge Numbers

A number that is equal to the product of two prime numbers is often referred to as a *semiprime*. The ultimate security of an RSA cryptosystem rests on how difficult it is to factor semiprimes. To stimulate this line of research, the RSA designers and the RSA security company publish, respectively, an old list and a new list of semiprimes, called *RSA challenge numbers*, soliciting solutions from the public. These numbers contain from 100 to 617 decimal digits. Early published RSA challenge numbers were named by RSA-l_d, where l_d is decimal length of the number. For instance, RSA-200 consists of 200 decimal digits, which was factored in May 2005. Later published RSA challenge numbers were named by RSA-l_b, where l_b is the binary length of the number. For instance, RSA-576 consists of 576 bits, which was factored in December 2003. RSA-640 was factored in November 2005. The following is the decimal representation of RSA-640:

RSA-640: 31074182404900437213507500358885679300373460228427
27545720161948823206440518081504556346829671723286
78243791627283803341547107310850191954852900733772
48227835257423864540146917366602477652346609

Table 3.1   Status of RSA challenge numbers

Challenge number	Decimal length	Prize (USD)	Status	Date
RSA-576	174	$10,000	Factored	2003–12–03
RSA-640	193	$20,000	Factored	2005–11–02
RSA-704	212	$30,000	Factored	2012–07–02
RSA-768	232	$50,000	Factored	2009–12–12
RSA-896	270	$75,000	Not factored	
RSA-1024	309	$100,000	Not factored	
RSA-1536	463	$150,000	Not factored	
RSA-2048	617	$200,000	Not factored	

Table 3.1 lists the status of RSA Factoring Challenge. RSA Security was acquired by EMC^2 in 2007, but the prizes offered by RSA Security for factoring RSA challenge numbers are still honored.

The competition of factoring RSA challenge numbers has led to two conclusions. Firstly, we should change semiprimes from time to time, where a particular semiprime should only be used in a time interval shorter than the time required to factor an RSA challenge number of a similar length. Secondly, we should use semiprimes that consist of more than 200 decimal digits.

However, there is a practicality issue on the length of semiprimes: if semiprimes are required to have long length to avoid being factored, then it may compromise the efficiency requirement. Thus, finding an alternative becomes important. This effort has led to the development of elliptic-curve cryptography.

3.5   Elliptic-Curve Cryptography

The mathematics used in elliptic-curve cryptography is deep. This section only provides a brief introduction.

In general, an elliptic curve is a plane curve defined by an equation of the form

$$y^2 + a_1xy + a_2y = x^3 + a_3x^2 + a_4x + a_5,$$
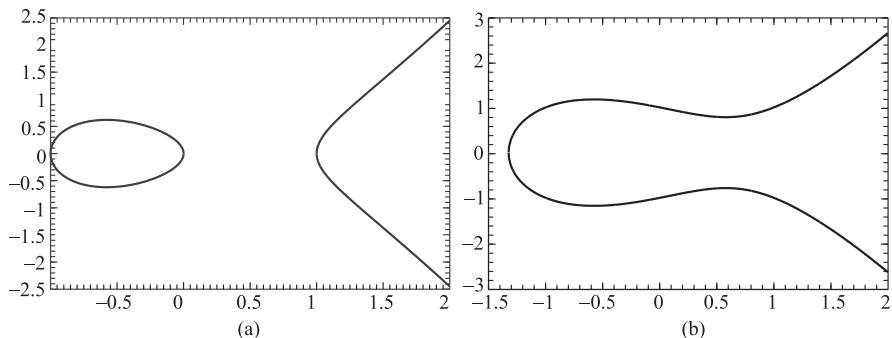
where coefficients a_1, a_2, a_3, a_4, a_5 are real numbers. Note that an elliptic curve may not have the shape of an ellipse.

We are particularly interested in the following special form of elliptic curves, with $a_1 = a_2 = a_3 = 0$ in Equation 3.10 and with a_4 renamed to a and a_5 to b (Fig. 3.2 provides two examples):

$$y^2 = x^3 + ax + b, \quad \text{where } 4a^3 + 27b^2 \neq 0. \tag{3.10}$$

3.5.1   Commutative Groups on Elliptic Curves

Let $\mathsf{E}(a, b)$ denote the set of points on the elliptic curve defined by Equation 3.10 . Then $\mathsf{E}(a, b)$ is additive. This property can be used to construct a commutative group. A commutative group,

Figure 3.2 (a) $y^2 = x^3 - x$. (b) $y^2 = x^2 - x + 1$

or Abelian group, is a set of elements G with an addition operation "+" satisfying the following five conditions:

1. **Closure:** $(\forall x, \ y \in G)[x + y \in G]$.
2. **Associativity:** $(\forall x, \ y, \ z \in G)[x + (y + z) = (x + y) + z]$.
3. **Unit element:** There is an element in G, denoted by 0, such that

$$(\forall x \in G)[x + 0 = 0 + x = 0].$$

4. **Inverse:** For any element $x \in G$, there is an element $x' \in G$ such that

$$x + x' = x' + x = 0,$$

where x' is often denoted by $-x$. We use $x - y$ to denote $x + (-y)$.
5. **Commutativity:** $(\forall x, \ y \in G)[x + y = y + x]$.

In a commutative group, the unit element is also called the *zero element.*

Let $X, \ Y \in \mathsf{E}(a, b)$. We have the following two cases.

Case 1: $X \neq Y$. Let L be a straight line connecting X and Y. If L is not perpendicular, then L must intersect with a unique point Z in $\mathsf{E}(a, b)$, where $Z \neq X$ and $Z \neq Y$.

Case 2: $X = Y$. Let L be a tangent line to the elliptic curve on point X. If L is not perpendicular, then L must intersect with a unique point Z in $\mathsf{E}(a, b)$, where $Z \neq X$.

In either case, if L is perpendicular, then L will not intersect with any other point in $\mathsf{E}(a, b)$. However, we introduce an imaginary point O that intersects with L at imaginary locations infinitely far away. This imaginary point O will play the role of the unit element, called a zero point.

Let $\mathsf{E}'(a, b) = \mathsf{E}(a, b) \cup \{O\}$. Define an addition operation "+" on points in $\mathsf{E}(a, b)'$ as follows:

1. For any $X \in \mathsf{E}'(a, b)$, let $X + O = X$.
2. For any $X, Y \in \mathsf{E}(a, b)$, if $X \neq Y$ but they have the same x-coordinate, then it follows from Equation 3.10 that X and Y are images on the x-axis; namely, $X = (x, y)$ and $Y = (x, -y)$. Let $X + Y = O$. Thus, we have $-X = (x, -y)$.

3. For any $X, Y \in \mathsf{E}(a,b)$, if their x-coordinates are different, then let L be a straight line connecting these two points.

    (a) If L is not a tangent line to the curve, then L must intersect with a unique different third point $Z \in \mathsf{E}(a,b)$. Let $X + Y = -Z$, namely, $X + Y$ is the image of Z on the x-axis.

    (b) If L is a tangent line to the curve at point X, let $X + Y = -X$.

    (c) If L is a tangent line to the curve at point Y, let $X + Y = -Y$.

4. For any $X \in \mathsf{E}(a,b)$, let L_X be a tangent line to the curve at point X. Let $Y \in \mathsf{E}(a,b)$ be another point that is also on L_X. Let $X + X = -Y$.

It can be shown that $(\mathsf{E}'(a,b), +)$ is a commutative group (see Exercise 3.8.2).

3.5.2 Discrete Elliptic Curves

For the sake of encoding data, we only consider integral points (x,y) in $\mathsf{E}(a,b)$; namely, x and y are both integers. Moreover, we consider integral points modulo a prime number p.

Let $Z_p = \{0, 1, \cdots, p-1\}$. If $4b^3 + 27c^2 \bmod p \neq 0$, let

$$\mathsf{E}_p(a,b) = \mathsf{E}(a,b) \cap \{(x,y) \mid x \in Z_p,\ y \in Z_p\}$$

$$\mathsf{E}'_p(a,b) = \mathsf{E}_p(a,b) \cup \{O\}.$$

Define an addition operation "+" over $\mathsf{E}'_p(a,b)$ to be the same as the addition operation over $\mathsf{E}'(a,b)$ for the first two conditions (i.e., replace $\mathsf{E}'(a,b)$ with $\mathsf{E}'_p(a,b)$). For the last two conditions, because a straight line connecting two points in $\mathsf{E}_p(a,b)$ may not intersect with the curve at an integral point in $\mathsf{E}_p(a,b)$, we modify these two conditions as follows:
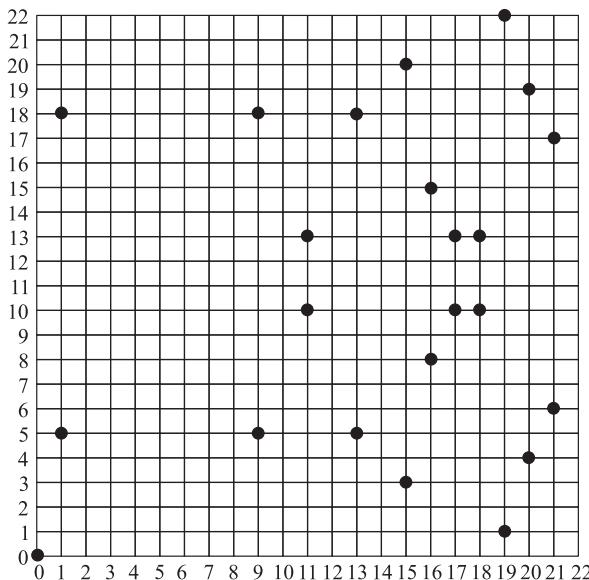
3′. For any $X, Y \in \mathsf{E}_p(a,b)$, if their x-coordinates are different, then let $X + Y = (x_3, y_3)$, where

$$x_3 = (\lambda^2 - x_1 - x_2) \bmod p,$$

$$y_3 = (\lambda(x_1 - x_3) - y_1) \bmod p,$$

$$\lambda = \frac{y_1 - y_2}{x_1 - x_2} \bmod p.$$

4′. For any $X = (x,y) \in \mathsf{E}_p(a,b)$, let $X + X = (x', y')$, where

$$x' = (\lambda^2 - 2x) \bmod p,$$

$$y' = (\lambda(x - x') - y) \bmod p,$$

$$\lambda = \frac{3x^2 + b}{2y} \bmod p.$$

It can be shown that $(\mathsf{E}'_p(a,b), +)$ is a commutative group.

Figure 3.3   Point distribution in $\mathsf{E}_{23}(1, 0)$

As an example, let $p = 23$, $b = 1$, and $c = 0$. We have

$$\mathsf{E}'_{23}(1,0) = \begin{cases} (\infty, \infty), & (0,0), & (1,5), & (1,18), & (9,5), & (9,18), \\ (11,10), & (11,13), & (13,5), & (13,18), & (15,3), & (15,20), \\ (16,8), & (16,15), & (17,10), & (17,13), & (18,10), & (18,13), \\ (19,1), & (19,22), & (20,4), & (20,19), & (21,6), & (21,17)\}, \end{cases}$$

where $O = (\infty, \infty)$. Figure 3.3 displays how points in $\mathsf{E}_{23}(1, 0)$ are distributed.

3.5.3   ECC Encodings

To encrypt plaintext M, we first represent it as a positive integer as in RSA cryptography. We then encode the integer representation as a point in $\mathsf{E}_p(a, b)$ in a way that it can be reversed. That is, M can be obtained from its point representation in $e_p(a, b)$. It is not known whether there is a polynomial-time algorithm to generate such an encoding, but an encoding can be obtained efficiently using a probabilistic algorithm. Although there is no guarantee that the algorithm can always generate a valid encoding, it can be shown that the probability that the algorithm fails to generate one is very small.

Suppose that M is a positive integer much smaller than p. Let $x = M$. Check whether

$$M^3 + bM + c$$

is equal to the square of some integer modulo p. If not, append a few digits at the end of M, and modify these digits if necessary, to obtain a new number $M' < p$, so that $M'^3 + bM' + c$

is equal to the square of some integer modulo p. This can be done using the following probabilistic algorithm.

Let γ be a large integer such that

$$\epsilon = 2^{-\gamma}$$

is very small, and $(M + 1)\gamma < p$. Let

$$x_j = M\gamma + j,$$

where $0 \leq j < \gamma$. For each j from 0 to $\gamma - 1$, compute

$$y_j = \sqrt{(x_j^3 + bx_j + c) \bmod p}.$$

If y_j is an integer, let $P_M = (x_j, y_j)$ be M's encoding. Otherwise, increase the value of j and repeat the same computation until an integral y_j is found or $j = \gamma$. If $j = \gamma$, it means that the algorithm failed to find an encoding for M. It can be shown that for each j, the probability that y_j is not an integer is approximately equal to $1/2$. Thus, the probability that the algorithm fails to find an encoding for M is only about ϵ. Given $P_M = (x, y)$, it is easy to verify that $M = \lfloor x/\gamma \rfloor$. We will call γ the *encoding parameter*.

For example, let $(p, b, c, \gamma) = (179, 3, 34, 15)$. Then $(4 \cdot b^3 + 27 \cdot c^2) \bmod p = 174 \neq 0$. It follows from $(M + 1)\gamma < 176$ that $1 \leq M \leq 12$. Let $M = 10$. We have $x_j = M\gamma + j = 150 + j$, where $0 \leq j < 15$. When $j = 0$, we have $x_0 = 150$ and $(x_0^3 + bx_0 + c) \bmod p = (150^3 + 3 \cdot 150 + 34) \bmod 179 = 81 = 9^2$. Thus, $y_0 = 9$ is an integer. Hence, $P_{10} = (150, 9)$ is an encoding for $M = 10$ over $\mathsf{E}'_{179}(3, 34)$. As $\lfloor 150/15 \rfloor = 10$, we can obtain $M = 10$ from $(150, 9)$.

3.5.4   ECC Encryption and Decryption

It is customary to refer to elliptic-curve encryption as ECC encryption and to elliptic-curve decryption as ECC decryption.

Let k be an integer greater than 1. For any $X \in \mathsf{E}'_p(a, b)$, let

$$kX = X + (k - 1)X.$$

The *elliptic-curve logarithm* problem is to find k from kX. There is no known efficient algorithm to solve this problem. The security of ECC encryption rests on the difficulty of solving this problem.

Similarly to Diffie-Hellman key exchange, ECC encryption also requires that Alice and Bob share the same parameters. Let $G \in \mathsf{E}_p(a, b)$ and γ be an encoding parameter. Let $(\mathsf{E}_p(a, b), G, \gamma)$ be the parameters shared by all parties.

Alice randomly selects a positive integer k_A as her private key. She then computes $P_A = k_A G$ as her public key and publishes P_A. Suppose that Bob wants to encrypt a plaintext block M using ECC encryption using Alice's public key K_A, where M is a positive integer satisfying $(M + 1)\gamma < p$. Bob uses the following ECC encryption procedure.

Bob randomly selects a positive integer k, encodes M to a point $P_M = (x, y) \in \mathsf{E}_p(a, b)$, and computes the following two points in $\mathsf{E}_p(a, b)$ as the ciphertext of M:

$$C = (kG, P_M + kP_A).$$

For convenience, we use $\pi_0(C)$ to denote kG and $\pi_1(C)$ to denote $P_M + kP_A$.

After receiving C from Bob, Alice uses the following ECC decryption procedure to decrypt C:

$$P_M = \pi_1(C) - k_A\pi_0(C) = (x, y). \tag{3.11}$$

She then computes $M = \lfloor x/\gamma \rfloor$.

The correctness of Equality 3.11 is shown as follows:

$$\begin{aligned}
\pi_1(C) - k_A\pi_0(C) &= (P_M + kP_A) - k_A(kG) \\
&= P_M + k(k_AG) - k_A(kG) \\
&= P_M.
\end{aligned}$$

3.5.5   *ECC Key Exchange*

ECC can also be used to exchange keys. Similarly to ECC encryption, Alice and Bob must agree on the same parameters $(\mathsf{E}_p(b, c), G, \gamma)$ in ECC key exchange. Let n be the smallest positive integer satisfying $nG = O$.

To obtain the same secret key through ECC key exchange, Alice selects at random a positive integer $k_A < n$ as her private key. She then computes $P_A = k_AG \in \mathsf{E}_p(a, b)$ as her public key and sends P_A to Bob. In the meantime, Bob selects at random a positive integer $k_B < n$ as his private key, computes $P_B = k_BG \in \mathsf{E}_p(a, b)$ as his public key, and sends P_B to Alice. Alice then computes $K_A = k_AP_B$ as her secret key, and Bob computes $K_B = k_BP_A$ as his secret key. It is easy to see that

$$K_A = k_AP_B = k_A(k_BG) = k_B(k_AG) = k_BP_A = K_B.$$

This key exchange scheme is also referred to as the elliptic-curve Diffie-Hellman (ECDH) scheme.

3.5.6   *ECC Strength*

The security of ECC rests on the difficulty of solving the elliptic-curve logarithm problem, which has not been studied as intensively as the discrete logarithm problem. However, the requirements of ECC parameters are not as rigid as those of RSA. In other words, using a prime number p consisting of a few hundred bits appears to be sufficient in ECC, while in RSA, the modulo n is required to have more than a thousand bits. On the other hand, ECC cryptanalysis has not been conducted as deeply or broadly as RSA cryptanalysis has. And so, our understanding of ECC is far less than that of RSA.

3.6   **Key Distributions and Management**

PKC takes substantially more time to encrypt data than conventional encryption algorithms, and so PKC is not suitable for encrypting long data. PKC is often used to encrypt secret keys for conventional encryption algorithms and other short messages. Suppose that Alice and Bob have already agreed on a symmetric-key encryption algorithm. When Alice wants to exchange confidential information with Bob; Alice generates a secret key K using a PRNG, encrypts K

using Bob's public key, and then sends the encrypted key C to Bob. Bob uses his private key to decrypt C to obtain K.

3.6.1 Master Keys and Session Keys

Suppose that Alice and Bob have agreed on a secret key. If they use this secret key to encrypt other secret keys during a certain period of time, then this key is referred to as a *master key*, denoted by K_m. To protect the master key from being compromised by Malice, Alice and Bob should use it sparingly to reduce exposure of the master key. Instead, they should generate a new secret key for each new communication session and use the master key to encrypt it for distribution. Such keys are referred to as *session keys*, denoted by K_s. In particular, suppose that Alice wants to send a confidential message to Bob, and they share a master key K_m. Alice first generates a session key K_s, encrypts it using K_m, and then sends $E_{K_m}(K_s)$ to Bob. After receiving $E_{K_m}(K_s)$, Bob decrypts it using K_m and obtains K_s. From now on, Alice and Bob will use K_s to encrypt and decrypt data for the current communication session until the session is terminated.

The lifetime of a session key is much shorter than that of a master key. A session key is typically used to encrypt a message (e.g., an email message) or to encrypt packets in a particular TCP connection from the time a TCP connection is established to the time the connection is closed. The lifetime of a master key is longer, depending on the underlying applications.

For example, suppose that Alice wants to log on to her company's computer from home using a secure remote login application program. This is a client-server program that does, among other things, the following: when Alice opens the client program on her home computer, the program generates a master key K_m, uses the public key of her company to encrypt K_m, and sends the encrypted K_m to the server program. The server program decrypts the encrypted master key sent from Alice using its private key to obtain K_m. The client program then generates a session key K_s, encrypts it using K_m, and sends $E_{K_m}(K_s)$ to the server program. This session key K_s will be used to encrypt all communications between the client program and the server program for the current login session, including Alice's user name, password, control messages, and data. When Alice logs out, the session key K_s becomes obsolete. However, the master key K_m remains valid for the next login session. The master key K_m will become obsolete when Alice exists the client program.

3.6.2 Public-Key Certificates

In addition to encrypting secret keys, RSA, as well as ECC, can also be used to authenticate data. Authentication helps to maintain data integrity and data nonrepudiation. Suppose that Alice wants to prove to Bob that a message M is indeed from her. She first uses her private key K_A^r to encrypt M and sends (M, C_A) to Bob, where $C_A = E_{K_A^r}(M)$. Bob uses Alice's public key K_A^u to decrypt C_A and then compare the decrypted message with M. If the two are identical, then Bob is convinced that M is indeed from Alice, because only she knows K_A^r. Using this procedure, Alice can prove her identity to Bob using a special message M that says "I am Alice."

To use PKC to encrypt or authenticate data, one must first obtain the receivers' public keys. Public keys may be published and distributed in a number of ways. No matter what method

is used, the user needs to be confident that a published public key really does belong to the purported owner.

Public keys can be published at a special Website in the form of a directory, listing owner names and their public keys. They also may be published at an owner's own Websites or they may be distributed in email messages. These methods are simple, but they are not secure, because there is no way to ensure true ownership of a public key. For instance, Malice may pretend to be Bob and lure Alice to use his public key as if it were Bob's. Therefore, we need to find a way to authenticate public keys. A common method is to use a trusted organization as an authenticator. For example, similarly to the domain name service that provides IP address lookups, a trusted organization may create a public-key service to provide public-key lookups. This scheme may be practical if each IP address only has one public key. In reality, however, the host of an IP address may be a machine with many user accounts, where each user may have a number of public keys. Thus, this method will cause substantially more network traffic, because each user might need to communicate with the public-key server for the receiver's public key. This high volume of potential network traffic makes the idea of a public-key service impractical.

The use of *public-key certificates* is a practical and simple method to authenticate public keys. Public-key certificates are also referred to as *digital certificates*. For simplicity, we sometimes use *certificates* to denote public-key certificates. Public-key certificates are issued by trusted organizations, which are referred to as *certificate authorities*, denoted by CAs. A CA uses PKC to authenticate certificates. It publishes its public key on its Website. It issues a certificate for each user, encrypted by CA's private key for authentication. In particular, a certificate is a digital document including the following information: user name, user's public key, issue date, CA name, and valid period of the key. Because a certificate may be long, a CA typically only encrypts a hash value of the certificate using its private key. This process is known as signing the certificate. When Alice needs to use Bob's public key, she first asks Bob to send her his certificate, and then uses CA's public key to verify that the certificate is indeed issued by the CA. From Bob's certificate, Alice obtains Bob's public key and its expiration period.

The use of public-key certificates helps to thwart data-repudiation attacks. If Bob possesses a digital document encrypted by Alice's private key and Alice's certificate, then Alice will have a difficult time denying that she did not sign the document, unless she can prove to the court beyond reasonable doubt that her private key was already stolen before she digitally signed the document.

We describe in Section 5.2 an industry standard of public-key certificates.

3.6.3   CA Networks

In addition to issuing certificates to users, a CA also needs to keep track of which certificates are out of date and which certificates have been canceled. It may become a problem when a CA has many users. To solve this problem, multiple CAs may be needed to form a *CA chain*.

We use $\text{CA}\langle K_X^u \rangle$ to denote the certificate issued by CA to user X whose public key is K_X^u.

Let CA_1 and CA_2 denote two different CAs. Assume that Alice is CA_1's user, but not CA_2's user. Alice possesses a certificate $\text{CA}_1\langle K_A^u \rangle$ issued by CA_1. Assume that Bob is CA_2'user, but not CA_1's user. Bob possesses a certificate $\text{CA}_2\langle K_B^u \rangle$ issued by CA_2. Alice does not know CA_2's public key or how to use it even if she did know it, for she is not CA_1's subscriber. Likewise, Bob does not know CA_1's public key nor how to use it. Thus, Alice and Bob have difficulties verifying each other's certificate.

To allow Alice to verify Bob's certificate, we require that CAs be able to authenticate each other's public keys. That is, CA_1 issues a certificate $CA_1 \langle K_{CA_2}^u \rangle$ for CA_2 and makes it available to its users. Also, CA_2 issues a certificate $CA_2 \langle K_{CA_1}^u \rangle$ for CA_1 and makes it available to its users. When Alice sends Bob her certificate $CA_1 \langle K_A^u \rangle$, she also sends $CA_2 \langle K_{CA_1}^u \rangle$ along with it. Bob first uses CA_2's public key to verify CA_1's public key, then uses CA_1's public key to verify Alice's public key. Likewise, Bob sends to Alice two certificates $CA_2 \langle K_B^u \rangle$ and $CA_1 \langle K_{CA_2}^u \rangle$, so that Alice can first verify CA_2's public key and then use it to verify Bob's public key.

We can use a directed graph, called a *CA network* to represent the relations between CAs, where a vertex represents a CA and an edge from vertex CA_i to vertex CA_j means that CA_j is CA_i's user; namely, CA_i has issued a certificate $CA_i \langle K_{CA_j}^u \rangle$ to CA_j. We call a non-CA user a regular user. Thus, the aforementioned example can be represented by the CA network shown in Figure 3.4.

When a CA network consists of more than two CAs, we call a path from one CA to another CA a *certificate path*. Figure 3.5 is such an example. In this example, the path from Alice (denoted by A in the figure) to Bob (denoted by B in the figure) has two certificate paths: $CA_1 \rightarrow CA_5 \rightarrow CA_4$ and $CA_1 \rightarrow CA_3 \rightarrow CA_5 \rightarrow CA_4$. However, from Bob to Alice, there is only one certificate path $CA_4 \rightarrow CA_2 \rightarrow CA_1$.

In the CA network shown in Fig. 3.5, regular users of CA_1 and CA_2 can verify each other's certificate as follows: suppose that Alice is a regular user of CA_1 and Bob a regular user of CA_4. When Bob sends his certificate $CA_4 \langle K_B^u \rangle$ to Alice, he must also send her all the certificates of the CAs on a certificate path from CA_4 to CA_1. In this case, there is only one certificate path: $CA_4 \rightarrow CA_2 \rightarrow CA_1$, and so Bob needs to send $CA_2 \langle K_{CA_4}^u \rangle$ and $CA_1 \langle K_{CA_2}^u \rangle$ to Alice, allowing Alice to use CA_1's public key to verify CA_2's public key, use CA_2's public key to verify CA_4's public key, and finally use CA_4's public key to verify Bob's public key.
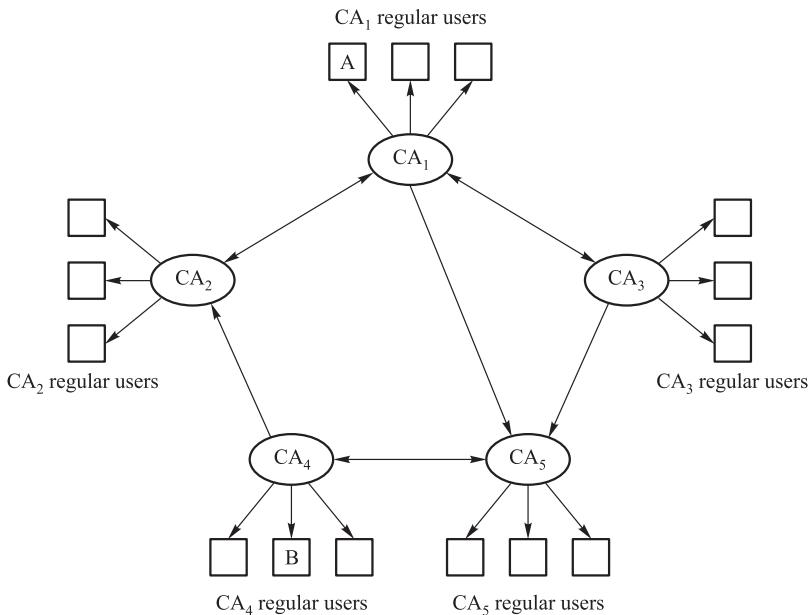
Multiple certificate paths may exist from one CA to another CA. Naturally, we would like to find the best certificate path, which should be the shortest and most trustworthy. However, these two measures may compete with each other, for different CAs may adopt different security policies and use different security protections, resulting in different levels of trust. Thus, a longer certificate path may turn out to be more trustworthy. How to measure the trustworthiness of a certificate path is a complex issue, which is beyond the scope of this book.

3.6.4 Key Rings

In a host computer, there may be many user accounts in which each user may have one or more public and private key pairs. When a user in the system obtains public-key certificates of users outside of the system, he should store these certificates and the corresponding public keys for

Figure 3.4   A CA network consisting of two CAs that can verify each other's public key

Figure 3.5 A CA network consisting of more than two CAs

Table 3.2 A sample private-key ring, where P_A represents Alice's login password and $H(P_A)$ represents the secret key generated from P_A

Key ID	Owner's name	Public key	Encrypted private key	Time stamp
$K_A^u \bmod 2^\ell$	Alice	K_A^u	$E_{H(P_A)}(K_A^r)$	T_A
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

later use for himself and for other users in the system, which avoids duplicate communications and computations. Thus, we need to find ways to store and manage these public and private key pairs. One way to do this is to use a centralized data structure to store all these pairs acquired by the different users in the system. This data structure is referred to as *public-key ring*. Likewise, for a particular security application, we may also use a centralized data structure to store all the public and private key pairs of each user in the system. This data structure is referred to as *private-key ring*.

3.6.4.1 Private-Key Rings

A private-key ring is a table (see Table 3.2) in which each row represents a record of a particular user, which includes the following attributes: key ID, owner's name, public key, encrypted private key, and time stamp.

The following are explanations of these attributes:

1. The key ID is an ℓ-bit binary string for a fixed ℓ to identify a public key. For example, we may use $K^u \bmod 2^\ell$ to represent an ℓ-bit ID of the public key K^u.
2. The owner's name is the user name who owns the public key in the record, which may be represented by user's login name or user's email address.
3. The public key is the public-key component in the user's public and private key pair.
4. The encrypted private key is the ciphertext of the private-key component in the user's public and private key pair, encrypted using a secret key generated from user's login password. The encrypted private key is to be decrypted using the user's login password.
5. The time stamp is the date and time sequence when the public and private key pair was generated.

A private-key ring may, of course, contain other attributes on the basis of the requirements of a specific application.

3.6.4.2   Public-Key Rings

A public-key ring is also a table in which each row represents a record of a particular user, which includes the following attributes: key ID, owner's name, public key, CA name, CA trust, and time stamp. The first three attributes have the same meanings as those in a private-key ring. The following are explanations of the rest of the attributes:

1. CA name is the name of the CA that issued the owner's public-key certificate.
2. CA trust is a numerical value, indicating the degree of trust given to the CA.
3. Time stamp is a date and time sequence at which the record was created.

A public-key ring may contain other attributes on the basis of the requirements of a specific application.

3.7   Closing Remarks

PKC is a major breakthrough in computer cryptography. It makes it possible to distribute secret keys confidentially from one user to the other without requiring users to share prior secrets. It also makes it possible to authenticate data. The security of PKC depends on the difficulties of solving certain mathematical problems. These problems are believed to be intractable on conventional computers, but there have been no formal proofs. The quest for a new and better public-key cryptosystem will continue.

3.8   Exercises

3.8.1   Discussions

3.1.   What is the major problem in distributing secret keys in network communications and why does PKC solve the problem?

3.2. Comparing Diffie-Hellman with RSA as key distribution schemes, which one do you think is better?

3.3. Comparing Elgamal PKC with RSA as encryption schemes, which one do you think is better?

3.4. Comparing Diffie-Hellman with ECC as key exchange schemes, which one do you think is better?

3.5. Comparing RSA with ECC as encryption schemes, which one do you think is better?

3.6. Discuss the concept of key chains.

3.8.2 Homework

3.1. Alice wants to send a plaintext file M to Bob confidentially. However, Alice and Bob do not share any prior secret. What Alice has is a box with two lock hasps, while Alice and Bob each have a lock with a key. Alice figures out a way to lock M inside this box and transmit the locked box to Bob such that during the transmission, the box is always locked. Bob can get M while no one else can, and no keys are transmitted. Describe Alice's method.

3.2. Let a, b, c, d, and n be integers with $n \neq 0$. Prove the following properties.
(a) $a \equiv a \,(\bmod\, n)$.
(b) $a \equiv 0 \,(\bmod\, n)$ if and only if n is divisible by a.
(c) $a \equiv b \,(\bmod\, n)$ if and only if $b \equiv a \,(\bmod\, n)$.
(d) If $a \equiv b \,(\bmod\, n)$ and $b \equiv c \,(\bmod\, n)$, then $a \equiv c \,(\bmod\, n)$.
(e) If $a \equiv b \,(\bmod\, n)$ and $c \equiv d \,(\bmod\, n)$, then

$$a + c \equiv b + d \,(\bmod\, n),\ a - c \equiv b - d \,(\bmod\, n),\ ac \equiv bd \,(\bmod\, n).$$

3.3. Calculate $\phi(12)$, and then find all positive integers $a < \phi(12)$ such that $a^{\phi(12)} \equiv 1$ $(\bmod\, 12)$. Prove that there is no primitive root modulo 12.

3.4. Let p be a prime number and $n < p$ be a positive integer. Show that $a^2 \bmod p = 1$ if and only if $a \bmod p = 1$ or $a \bmod p = -1$.

3.5. The basic operation in the fast modular exponentiation algorithm is the modular squaring operation $a^2 \bmod n$. If we let a be an ℓ-bit binary integer, then the length of a^2 is about 2ℓ. If ℓ is close to $\log_2 n$ (the length of the binary representation of n), then computing a^2 will require about $2\log_2 n$ space. To reduce the memory requirement, devise an algorithm to compute $a^2 \bmod n$ so that the binary representation of the largest number during the computation has at most $3\ell/2$ bits.

3.6. Use the fast modular exponentiation algorithm to compute $101^{124} \bmod 110$.

3.7. Write a program to implement the fast modular exponentiation algorithm.

3.8. Let $x = m/n$ be a rational number, where m and n are positive integers and $\gcd(m, n) = 1$. Let $[a_0; a_1, \cdots, a_k]$ be the finite continued fraction of x, generated by the continued fraction construction algorithm. Show that $k \le \log_2 n$.

3.9. If we let prime number $p = 353$ then $a = 3$ is a primitive root modulo p. Use these two numbers to construct a Diffie-Hellman key exchange system.
  - (a) If Alice selects a private key $X_A = 97$, what is the value of Alice's public key Y_A?
  - (b) If Bob selects a private key $X_B = 233$, what is the value of Bob's public key Y_B?
  - (c) What is the value of the secret key agreed on by both Alice and Bob?

3.10. Let $p = 13$.
  - (a) Show that $a = 2$ is a primitive root modulo p. Use these two parameters to construct a Diffie-Hellman key exchange system.
  - (b) If Alice's public key is $Y_A = 7$, what is the value of her private key X_A?
  - (c) If Bob's public key is $Y_B = 11$, what is the value of his private key X_B?

3.11. Let p be a prime and a be a positive integer. Alice selects at random a private key X_A and uses $Y_A = X_A^a \bmod p$ for her public key. Bob selects at random a private key X_B and uses $Y_B = X_B^a \bmod p$ as his public key. How do Alice and Bob agree on the same secret key? Is this method secure? Justify your answer.

3.12. Write a client-server program using the socket API to implement the Diffie-Hellman key exchange protocol. To complete this problem, you should first complete Exercise 2.39 and Exercise 3.7.

3.13. Prove the correctness of Elgamal decryption. That is, prove Equality 3.5.

3.14. Can the man-in-the-middle attack that can be carried out against the Diffie-Hellman key exchange be used successfully to attack Elgamal PKC? Justify your answer.

3.15. Let $p = 61$, $q = 53$, $d = 2753$. Find e such that $de \equiv 1 \bmod \phi(pq)$.

3.16. Let $n = 187 = 11 \times 17$.
  - (a) Let $e = 7$, $M = 89$. Calculate the RSA ciphertext C.
  - (b) From C calculated in (a), compute plaintext M.
  - (c) Let $e = 7$, $M = 88$. Calculate the RSA ciphertext C. Can this C be used to factor $n = 187$? Justify your answer.

3.17. Describe how to attack RSA if a small value of $e = 3$ is used. Can you come up a simple way to prevent such attacks? Justify your answer.

3.18. Alice uses the following method to encrypt English plaintext messages, where only capital letters are used: map each English capital letter to numbers from 100 to 125; namely, map A to 100, B to 101, $\cdots$, and Z to 125. She then encrypts such integers one at a time. She uses large values of n and e. Is Alice's method secure? Justify your answer.

3.19. If $C = M^e \bmod n$ in RSA encryption, show that $(2^e C)^d \equiv 2M \bmod n$.

3.20. Assume that Alice encrypts a message M using RSA with public keys $n = 437$ and $e = 3$, which yields ciphertext $C = 75$. If someone tells Malice that $M \in \{8, 9\}$, then Malice can determine the true value of M without factoring n. How can Malice do this?

3.21. Assume that Alice and Bob use the same RSA number n, where Alice's public exponent e_A and Bob's public exponent e_B are relatively prime. Now Charlie wants to send a message M to both Alice and Bob. Charlie encrypts M using Alice's public key to get $C_A = M^{e_A} \bmod n$ and encrypts M using Bob's public key to get $C_B = M^{e_B} \bmod n$. He then sends C_A to Alice and C_B to Bob. Malice intercepts both C_A and C_B, which he uses to calculate M. How can Malice do this?

3.22. Write a client-server program to implement RSA encryption and decryption, where RSA parameters are given. To do this exercise, you should first complete Exercise 3.7.

**3.23. RSA-576 and RSA-640 have been factored. Conduct a literature search and write a paper describing how these numbers were factored.

3.24. Let $y^2 = x^3 - x + 1$ be an elliptic curve. Let $X = (1, 1)$ and $Y = (-1, -1)$. Compute $X + Y$ and $2Y$.

3.25. Compute $\mathsf{E}_{23}(0, 1)$ and $\mathsf{E}_{17}(1, 1)$.

*3.26. Show that $(\mathsf{E}'(a, b), +)$ defined in Section 3.5.1 is a commutative group.

*3.27. Show that $(\mathsf{E}'_p(a, b), +)$ defined in Section 3.5.2 is a commutative group.

3.28. Show that ECC encryption and decryption satisfy commutativity.

3.29. Let the global parameters of an ECC system be $\mathsf{E}_{23}(1, 1), G = (3, 10)$, and $\gamma = 4$. Assume that Alice selects $k_A = 5$ as her private key.
(a) Compute Alice's public key P_A.
(b) Let $M = 4$. Compute M's encoding P_M using $\mathsf{E}_{23}(1, 1)$.
(c) Bob selects $k = 3$ and encrypts P_M using Alice's public key to get C. What are the values of the two coordinates in C?
(d) Show how Alice decrypts C to get M.

3.30. Let the global parameters of an ECC system be $\mathsf{E}_{11}(1, 6), G = (2, 7)$, and $\gamma = 2$. Assume that Alice selects $k_A = 6$ as her private key.
(a) Compute Alice's public key P_A.
(b) Let $M = 2$. Compute M's encoding P_M in $\mathsf{E}_{11}(1, 6)$.
(c) Bob selects $k = 5$ and encrypts P_M using Alice's public key to get C. What are the values of the two coordinates in C?
(d) Show how Alice decrypts C to get M.

3.31. Discuss whether it is useful to have a data structure of a *secret-key ring* in a system. If yes, give an example to show that this is a useful concept.

3.32. The use of PKC and public-key certificates to distribute secret keys is a simple and secure method. Without PKC, we can also use symmetric-key encryption algorithms to distribute secret keys using a *key distribution center (KDC)*. A KDC is a trusted organization. Each user of a KDC must first register with the KDC and establish a master key with the KDC. When Alice wants to communicate with Bob confidentially, Alice first requests her KDC to generate a session key. After receiving such a request from Alice, KDC generates a session key, encrypts it using the master key shared with Alice, and sends it back to Alice.

(a) Devise a secure session key distribution protocol for a KDC.

*(b) Improve your protocol in (a) so that it can resist man-in-the-middle attacks and message replay attacks and allows Alice and Bob to authenticate each other's identity. Moreover, your protocol should cut down communication overhead as much as possible and incorporate TCP three-way handshake protocol to establish a protected connection between Alice and Bob.

(c) How does the KDC manage all the master keys? Is a secret-key ring data structure useful in this application?

(d) Before using the KDC, each user must first register with the KDC, prove his identity to the KDC, and then establish a shared master key. Without using PKC, how can this be done?

(e) A KDC can easily become a bottleneck when it has to handle many requests from many users in a short period of time. Design a hierarchical KDC system to help solve this problem.

(f) Analyze the pros and cons of using a KDC to distribute secret session keys.

3.33. Similar to setting up a KDC for distributing secret keys, we can also establish a *public-key authority center (PKA)* to obtain a user's public key without using his public-key certificate. A PKA is a trusted organization. To use PKA, each user must first register his public key with the PKA.

(a) Devise a secure public-key distribution protocol for a PKA.

*(b) Improve your protocol in (a) so that it can resist man-in-the-middle attacks and message replay attacks and allows Alice and Bob to authenticate each other's identity. Moreover, your protocol should cut down communication overhead as much as possible, and incorporate the TCP three-way handshake protocol to establish a protected connection between Alice and Bob.

(c) How does a PKA manage users' public keys? Can a user prove his identity to the PKA without registering with the PKA?

(d) Before using a PKA, each user must first register with the PKA and prove his identity to the PKA. How can this be done?

(e) A PKA can easily become a bottleneck when it has to handle many requests from many users in a short period of time. Design a hierarchical PKA system to help solve this problem.

(f) Analyze the pros and cons of using a PKA to distribute public keys.

*3.34. Suppose that Alice knows an integer i and Bob knows an integer j. They want to know whether $i \leq j$ without revealing their own integers to each other. This is often referred to as *Yao's millionaire problem*. Devise a security protocol to solve this problem.

*3.35. Alice and Bob will go to two separate places in a region governed by a dictator, where the only way they can communicate is through a government-censored network. To use this network, messages are not allowed to be encrypted because government agents want to reveal the content of the messages. But the sender is allowed to sign the message. Alice and Bob do share a secret key, but they will not be able to use any encryption algorithms once they are there. They figure out a way to communicate using the government-censored network, so that the real data is well protected with confidentiality and integrity. This is often referred to as a *subliminal channel*. Try to describe their method and justify your answer. Note: there are a number of ways to build a subliminal channel. Alice and Bob should use a method that is cost-effective and secure (if they were caught cheating, they might be executed).

4

Data Authentication

Data authentication has two purposes: certify the origin of the data and convince the user that the data has not been modified or fabricated. Data authentication is a critical mechanism to maintain data integrity and nonrepudiation. Data authentication may be achieved either using conventional encryption algorithms or using public-key cryptography.

Suppose that Alice and Bob share a common secret key K. Alice wants to send a data string M to Bob and convince Bob that M does indeed come from her without being modified during transmission. This can be done as follows: Alice sends M together with C to Bob, where $C = E_K(M)$ and E is a conventional encryption algorithm agreed on by Alice and Bob. Because only Alice and Bob know K, Bob can use K to decrypt C to get M'. Bob will be convinced that M indeed comes from Alice and that M has not been modified during transmission if and only if $M' = M$. This method, however, allows Alice to deny to Charlie that M comes from her, for it could have come from Bob who shares the same secret key K with her. Public-key cryptography overcomes this obstacle. Section 3.6.2 has introduced how to use public-key cryptography to authenticate data and provide data nonrepudiation.

If M is short, one may encrypt M directly to authenticate it. However, if M is long, encrypting the entire M for the purpose of authenticating it may be overkill because it incurs extra computation and traffic overhead. To authenticate a long data string M, it suffices to compute a short representation h of M and encrypt h.

A short representation of M generated without using any secret key is often referred to as a *digital digest* or a *digital fingerprint*, which can be obtained using a *cryptographic hash function*. A short representation of M generated using a secret key is referred to as a *message authentication code* (*MAC*) or a *tag*, which can be obtained using an *encrypted checksum algorithm*. We can also combine a cryptographic hash function and an encrypted checksum algorithm to generate a *keyed-hash message authentication code* (*HMAC*).

This chapter introduces hash function, MAC, and HMAC algorithms, as well as digital signature standard algorithms. It also introduces a dual signature scheme for electronic transactions and a blind signature scheme for producing electronic cash.

4.1 Cryptographic Hash Functions

A *hash function* takes a long string as input, breaks it into pieces, mixes them up, and produces a new string with a short length. Not every hash function is suitable for generating a digital

fingerprint. For example, let us look at a simple hash function $H_{\oplus}$ that uses the XOR operation to transform an input string of arbitrary length to a 16-bit long output string. In particular, let $M = M_1 M_2 \cdots M_k$, where each M_i (except possibly the last block M_k) is a 16-bit binary string. If the last block is shorter than 16 bits, pad it with 1's to make it a 16-bit block. For convenience, we still use M_k to denote this block. Let

$$H_{\oplus}(M) = M_1 \oplus M_2 \oplus \cdots \oplus M_k.$$

This hash function is ill suited for generating fingerprints because it is easy to find sentences with different meanings that have the same hash value under this hash function. For example, consider the following two sentences with different meanings: "He likes you but I hate you" and "He hates you but I like you". Let S_1 and S_2 denote, respectively, the binary strings obtained from the first and the second sentences by encoding English letters using 8-bit ASCII codes and removing white spaces between words. It is straightforward to verify that $H_{\oplus}(S_1) = H_{\oplus}(S_2)$.

To be well suited for generating fingerprints, a hash function must satisfy several criteria.

4.1.1 Design Criteria of Cryptographic Hash Functions

Let H be a hash function to be constructed. We first set an upper bound Γ for the length of input strings (measured by bits), where Γ is a very large number. The output length γ is fixed, where γ is much less than Γ. For example, we may choose $\Gamma = 2^{64} - 1$ and $\gamma = 160$. We require that each γ-bit string be selected with the same likelihood as a hash value of the hash function if input strings are selected uniformly and independently at random. Thus, it follows from the pigeonhole principle that for any input string x, there must be several input strings y with $H(x) = H(y)$. To generate a good digital fingerprint, H must possess the *one-way property* and the *computational uniqueness property*. Such a hash function is referred to as a *cryptographic hash function* (*CHF*).

4.1.1.1 One-Way Property

The one-way property assures that computing a digital fingerprint for a given string is easy but finding a string that has a given fingerprint is hard. In other words, for any binary string x with $|x| \leq \Gamma$, it is easy to compute $H(x)$ (e.g., in linear time of $|x|$), but for any binary string h with $|h| = \gamma$, it is hard to find a binary string x such that $h = H(x)$ (e.g., finding such an x requires exponential time of $|x|$).

4.1.1.2 Computational Uniqueness Property

The computational uniqueness property assures that it is computationally difficult to find two different strings with the same fingerprint. There are two types of computational uniqueness; they are *collision resistance* and *strong collision resistance*.

Collision Resistance

A hash function H is *collision resistant*, if for any binary string x with $|x| \leq \Gamma$, it is computationally intractable (e.g., it requires exponential time of $|x|$) to find a different string y with $|y| \leq \Gamma$ such that $H(x) = H(y)$, although we know that such strings y exist.

Strong Collision Resistance

A hash function H is *strongly collision resistant* if it is computationally intractable to find two binary strings x and y with $|x| \leq \Gamma$ and $|y| \leq \Gamma$ (e.g., it requires exponential time of $|x| + |y|$) such that $H(x) = H(y)$ in practice for most situations.

It is straightforward to verify that if a hash function is not collision resistant, then it is not strongly collision resistant. However, the opposite is not true. In other words, even if a hash function is not strongly collision resistant, it may still be collision resistant in practice for most situations.

4.1.2   Quest for Cryptographic Hash Functions
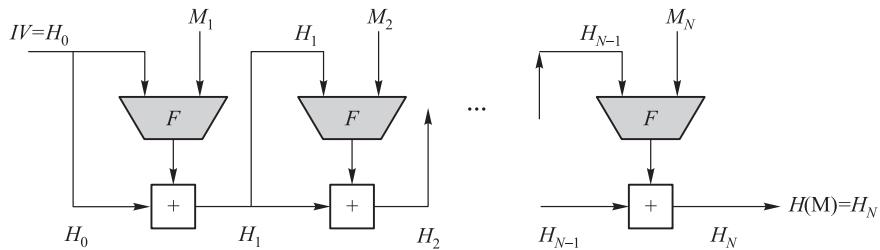
Despite intensive efforts, it is still not known whether cryptographic hash functions exist with the one-way property and the computational uniqueness property. Over the years, several candidates of cryptographic hash functions have been constructed and used in practice, although there is no mathematical proof that these hash functions indeed possess the desired properties. These hash functions may contain subtle loopholes that are exploitable by the attacker. Thus, it is important to identify such weaknesses to help devise stronger hash functions.

In 2004, for example, a Chinese mathematician Xiaoyun Wang and her group showed that several widely used hash functions at that time, including MD4, MD5, HACAL-128, and RIPEMD, do not satisfy the requirement of strong collision resistance. This is contrary to prior beliefs. They also showed in 2005 that another commonly used hash function SHA-1's collision resistance is not as strong as people thought it was. They developed a method that can find two different strings x and y, with time complexity in the order of 2^{69}, such that SHA-1(x) = SHA-1(y). Prior to this, it was commonly believed that the time complexity of finding such a pair of strings is in the order of 2^{80} (see Section 4.4). Later that year, Xiaoyun Wang, Andrew Yao (2000 Turing Award winner), and Francis Yao further reduced this time complexity to the order of 2^{63}. These results again tell us what was believed to be strong and secure may no longer be so because of advancements of technologies and methodologies. New findings will stimulate new designs of hash functions to overcome the problems that have been identified.

This section introduces two standard hash functions. They are SHA-512 and WHIRLPOOL. They are strong candidates of cryptographic hash functions.

SHA stands for *Secure Hash Algorithms*. SHA-512 and SHA-1 hash functions were devised by NSA and were made standards for cryptographic hash functions by NIST in, respectively, 1995 and 2002. Exercises 4.8 and 4.9 present a description of SHA-1. Between SHA-1 and SHA-512, there are SHA-256 and SHA-384. The set of SHA-256, SHA-384, and SHA-512 is also referred to as the SHA-2 series.

WHIRLPOOL, named after M51 (Whirlpool) Galaxy in Canes Venatici, was devised by a Brazilian cryptographer Paulo SLM Barreto and a Belgium cryptographer Vincent Rijmen

Figure 4.1   The basic structure of hash functions, where M is a plaintext block, IV is an initial vector, F is a compression function, and "+" is some form of modular addition operation

(co-author of AES) in 2000, which was recommended by NESSIE and adopted by ISO and the International Electrotechnical Commission (IEC) as part of the joint ISO/IEC international standard in 2003.

4.1.3   Basic Structure of Standard Hash Functions

The SHA-1, SHA-2, and WHIRLPOOL hash algorithms all have the same basic structure. This structure was proposed by Ralph C. Merkle in 1978. The heart of this basic structure is a *compression function F*. Different hash algorithms use different compression functions. The basic structure is a CBC mode of repeated applications of F without using secret keys (see Figure 4.1).

4.1.4   SHA-512

In SHA-512, $\Gamma = 2^{128} - 1$ and $\gamma = 512$, whereas in SHA-1, $\Gamma = 2^{64} - 1$ and $\gamma = 160$.

4.1.4.1   Initial Process

Let M be a binary string with $|M| \leq \Gamma$. Let L be the length of M. We represent L as a 128-bit binary string and denote it by $b_{128}(L)$. We pad M to produce a new binary string M' as follows:

$$M' = M \parallel 10^{\ell} \parallel b_{128}(L), \quad \ell \geq 0,$$

such that the length of M' is divisible by 1024, where $\parallel$ represents concatenation. Let L' denote the length of M'. We have

$$L' = L + (1 + \ell) + 128 = L + \ell + 129.$$

It follows from Equality 3.3 that

$$L = 1024 \cdot \left\lfloor \frac{L}{1024} \right\rfloor + L \mod 1024.$$

Hence, we can determine ℓ as follows:

$$\ell = \begin{cases} 895 - L \bmod 1024, & \text{if } 895 \geq L \bmod 1024, \\ 895 + (1024 - L \bmod 1024), & \text{if } 895 < L \bmod 1024. \end{cases}$$

It is straightforward to verify that there is a positive integer N such that $L' = 1024N$. So we can write

$$M' = M_1 M_2 \cdots M_N,$$

where each M_i is a 1024-bit binary string.

SHA-512 uses a 512-bit initial vector $\boldsymbol{IV}$. Let $r_1, r_2, r_3, r_4, r_5, r_6, r_7$, and r_8 be variables, each of which represents a 64-bit binary string. We view each r_i as a 64-bit register. Their initial values are, respectively, the 64-bit binary string in the prefix of the fractional component of $\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{7}, \sqrt{11}, \sqrt{13}, \sqrt{17}$, and $\sqrt{19}$. Representing these values in hexadecimal, we have

$$r_1 = \texttt{6a09e667f3bcc908}, \quad r_5 = \texttt{510e527fade682d1},$$

$$r_2 = \texttt{bb67ae8584caa73b}, \quad r_6 = \texttt{9b05688c2b3e6c1f},$$

$$r_3 = \texttt{3c6ef372fe94f82b}, \quad r_7 = \texttt{1f83d9abfb41bd6b},$$

$$r_4 = \texttt{a54ff53a5f1d36f1}, \quad r_8 = \texttt{5be0cd19137e2179}.$$

All SHA algorithms store binary strings in the big-endian format (see Exercise 2.26 for the definition of the big-endian format).

4.1.4.2 SHA-512 Compression Function

The compression function F of SHA-512 takes two inputs; one is a 1024-bit plaintext block M_i, and the other is a 512-bit string H_{i-1}, where $1 \leq i \leq N$ and H_{i-1} is the current content in $r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8$.

Divide M_i into sixteen 64-bit blocks $W_0, W_1, \cdots, W_{15}$, where

$$W_i = M[64i, 64i + 64], i = 0, 1, \cdots, 15$$

Then generate sixty-four 64-bit binary strings $W_{16}, W_{17}, \cdots, W_{79}$ as follows:

$$W_t = [\sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}] \mod 2^{64},$$
$$t = 16, \cdots, 79,$$
$$\sigma_0(W) = (W \ggg 1) \oplus (W \ggg 8) \oplus (W \lll 7),$$
$$\sigma_1(W) = (W \ggg 19) \oplus (W \ggg 61) \oplus (W \lll 6),$$

where $W \ggg n$ denotes a shift operation that shifts W circularly to the right n times, and $W \lll n$ denotes a shift operation that shifts W linearly to the left n times (with the n-bit suffix of W filled with 0's).

Let $X = x_1 x_2 \cdots x_\ell$ and $Y = y_1 y_2 \cdots y_\ell$ be two binary strings of equal length, where $x_i, y_i \in \{0, 1\}$. Define

$$X \wedge Y = (x_1 \wedge y_1)(x_2 \wedge y_2) \cdots (x_\ell \wedge y_\ell),$$
$$X \vee Y = (x_1 \vee y_1)(x_2 \vee y_2) \cdots (x_\ell \vee y_\ell),$$
$$\overline{X} = \overline{x}_1 \overline{x}_2 \cdots \overline{x}_\ell,$$

where $\wedge$ denotes the logical conjunction, that is,

$$0 \wedge 1 = 1 \wedge 0 = 0 \wedge 0 = 0, 1 \wedge 1 = 1;$$

$\vee$ denotes the logical disjunction, that is,

$$0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1, 0 \vee 0 = 0;$$

and $\overline{x}$ denotes the logical negation, that is,

$$\overline{0} = 1, \overline{1} = 0.$$

Let $Z = z_1 z_2 \cdots z_\ell$ be a binary string, where each z_i is a bit. Let $ch(X, Y, Z)$ denote the conditional predicate "if X, then Y else Z." That is,

$$ch(X, Y, Z) = (X \wedge Y) \vee (\overline{X} \wedge Z).$$

Let $maj(X, Y, Z)$ denote the majority predicate, that is,

$$maj(X, Y, Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z).$$

Let $K_0, K_1, \cdots, K_{79}$ denote the sequence of SHA-512 constants, where each constant is a 64-bit binary string (see Appendix B). Let T_1 and T_2 denote temporary variables representing 64-bit binary strings. Let r denote a 64-bit register. Let

$$\Delta_0(r) = (r \ggg 28) \oplus (r \ggg 34) \oplus (r \ggg 39),$$
$$\Delta_1(r) = (r \ggg 14) \oplus (r \ggg 18) \oplus (r \ggg 41).$$

The SHA-512 compression function $F(M_i, H_{i-1})$ for each i is executed 80 rounds of the same operations. In particular, for t from 0 to 79, it does the following:

$$T_1 \leftarrow [r_8 + ch(r_5, r_6, r_7) + \Delta_1(r_5) + W_t + K_t] \mod 2^{64},$$
$$T_2 \leftarrow [\Delta_0(r_1) + maj(r_1, r_2, r_3)] \mod 2^{64},$$
$$r_8 \leftarrow r_7,$$
$$r_7 \leftarrow r_6,$$
$$r_6 \leftarrow r_5,$$
$$r_5 \leftarrow (r_4 + T_1) \mod 2^{64},$$
$$r_4 \leftarrow r_3,$$

$$r_3 \leftarrow r_2,$$
$$r_2 \leftarrow r_1,$$
$$r_1 \leftarrow (T_1 + T_2) \mod 2^{64}.$$

After 80 rounds of executions, the 512-bit string in $r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8$ is the output of $F(M_i, H_{i-1})$.

4.1.4.3 SHA-512 Algorithm

Let $X = X_1 X_2 \cdots X_k$ and $Y = Y_1 Y_2 \cdots Y_k$, where each X_i and Y_i is an ℓ-bit binary string. Generalize the bitwise-XOR operation to an ℓ-bitwise-XOR operation as follows:

$$X \oplus_\ell Y = [(X_1 + Y_1) \mod 2^\ell][(X_2 + Y_2) \mod 2^\ell] \cdots [(X_k + Y_k) \mod 2^\ell].$$

Clearly, $\oplus_1$ is the standard bitwise-XOR operation $\oplus$. Then M's digital fingerprint is determined by $H(M) = H_N$, where H_N is calculated recursively as follows:

$$H_0 = \boldsymbol{IV},$$
$$H_i = H_{i-1} \oplus_{64} F(M_i, H_{i-1}),$$
$$i = 1, 2, \cdots, N.$$

4.1.5   WHIRLPOOL

In WHIRLPOOL, $\Gamma = 2^{256} - 1$ and $\gamma = 512$.

4.1.5.1   Initial Process

Let M be a binary string with $|M| < 2^{256}$. Let L be the length of M. We represent L as a 256-bit binary string and denote it by $b_{256}(L)$. Similarly to SHA-512, we pad M to produce a new binary string M' as follows:

$$M' = M \parallel 10^\ell \parallel b_{256}(L), \quad \ell \geq 0,$$

such that the length of M' is divisible by 512. Let L' denote the length of M'. We have

$$L' = L + (1 + \ell) + 256 = L + \ell + 257.$$

It follows from Equality 3.3 that

$$L = 512 \cdot \left\lfloor \frac{L}{512} \right\rfloor + L \mod 512.$$

Hence, we can determine ℓ as follows:

$$\ell = \begin{cases} 255 - L \mod 512, & \text{if } 255 \geq L \mod 512, \\ 255 + (512 - L \mod 512), & \text{if } 255 < L \mod 512. \end{cases}$$

It is straightforward to verify that there is a positive integer N such that $L' = 512N$. So we can write

$$M' = M_1 M_2 \cdots M_N,$$

where each M_i is a 512-bit binary string.

4.1.5.2 Compression Function

The heart of the WHIRLPOOL compression function F is an encryption algorithm W. It takes a 512-bit plaintext block X and a 512-bit key K as input, manipulates X in a way similar to AES, and produces a 512-bit output $W(X, K)$. In this algorithm, a 512-bit binary string is divided into a sequence of 64 bytes and is treated as an 8×8 state matrix of bytes, where elements are listed row wise. Recall that AES deals with 4×4 matrices of bytes.

WHIRLPOOL's compression function is defined as follows:

$$F(X, K) = W(X, K) \oplus X.$$

The WHIRLPOOL fingerprint of M is obtained using a CBC mode on M_i defined as follows:

$$
\begin{aligned}
H_0 &= 0^{512}, \\
H_i &= H_{i-1} \oplus F(M_i, H_{i-1}) \\
&= H_{i-1} \oplus W(M_i, H_{i-1}) \oplus M_i, \\
i &= 1, 2, \cdots, N,
\end{aligned}
$$

where $H(M) = H_N$.

We describe in the following section how round keys are generated and how $W(X, K)$ is constructed.
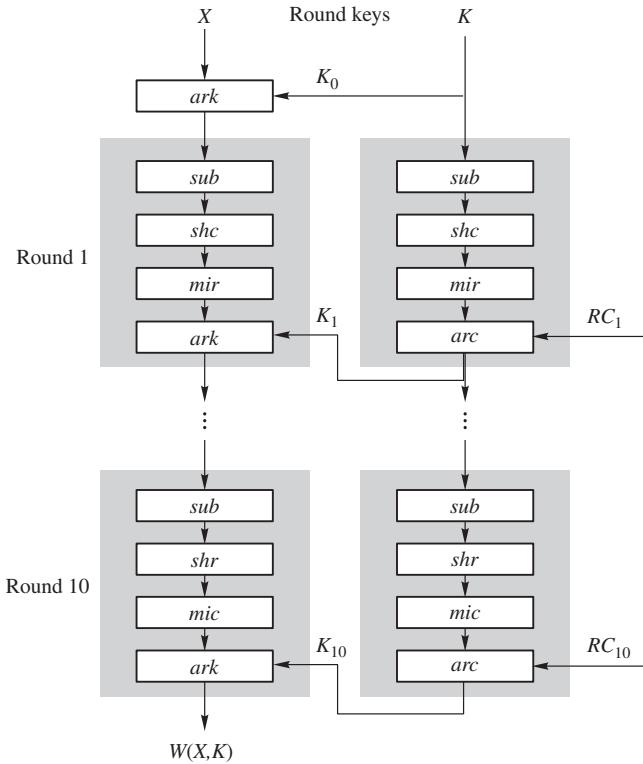
Generating Round Keys
A total of eleven 512-bit round keys are generated from K, denoted by $K_0, K_1, \cdots, K_{10}$. In particular, $K_0 = K$ and $K_i (1 \le i \le 10)$ is generated using the same sequence of four basic operations on K_{i-1}. These four operations are *substitute-bytes*, denoted by *sub*; *shift-columns*, denoted by *shc*; *mix-rows*, denoted by *mir*; and *add-round-constant*, denoted by *arc*. In other words, We will treat K_i as an 8×8 state matrix of bytes for these operations and so we will denote it by bolditalic $\boldsymbol{K}_i$.

$$\boldsymbol{K}_i = arc(mir(shc(sub(\boldsymbol{K}_{i-1}))), \boldsymbol{RC}_i),$$

where $\boldsymbol{RC}_i$ is an 8×8 state matrix RC_i representing a 512-bit constant string obtained directly from WHIRLPOOL's S-Box (see Table 4.1). In particular, the first eight bytes in RC_i are the ith eight bytes in the S-Box, where the rest of the bytes are 0's. That is, if we write RC_i as a sequence of bytes with $RC_i[j]$ denoting the jth byte in RC_i, where $0 \le j \le 63$, and list the elements in WHIRLPOOL's S-Box row wise as $s_0, s_1, \cdots, s_{255}$, then

$$
RC_i[j] = \begin{cases} s_{8(i-1)+j}, & \text{if } 0 \le j \le 7, \\ 00000000, & \text{if } 8 \le j \le 63, \end{cases}
$$

Figure 4.2   Block diagram of W

where $i = 1, 2, \cdots, 10$. For instance, the first eight bytes in RC_1 are 18 23 c6 e8 87 b8 01 4f, and the first eight bytes in RC_8 are e4 27 41 8b a7 7d 95 c8.

Encryption Structure

After the round keys are generated, the algorithm W writes the 64-byte string X in the form of a state matrix $\boldsymbol{A} = (a_{u,v})_{8\times 8}$, where

$$a_{u,v} = x_{8u+v},$$

and $u, v = 0, 1, \cdots, 7$. It then performs the *add-round-key* operation, denoted by *ark*, on $\boldsymbol{A}$ and $\boldsymbol{K}_0$ to generate a new string A_0. It repeats the same sequence of four operations for 10 rounds. In particular, for each round i with $1 \le i \le 10$,

$$\boldsymbol{A}_i = ark(mir(shc(sub(\boldsymbol{A}_{i-1}))), \boldsymbol{K}_i),$$

and $W(X, K) = \boldsymbol{A}_{10}$.

Figure 4.2 depicts the block diagram of W.

Substitute Bytes

WHIRLPOOL's operation of substitute-bytes uses a 16×16 S-Box defined in Table 4.1.

Table 4.1 WHIRLPOOL S-Box (in hexadecimal values)

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	18	23	c6	e8	87	b8	01	4f	36	a6	d2	f5	79	6f	91	52
1	60	bc	9b	8e	a3	0c	7b	35	1d	e0	d7	c2	2e	4b	fe	57
2	15	77	37	e5	9f	f0	4a	ca	58	c9	29	0a	b1	a0	6b	85
3	bd	5d	10	f4	cb	3e	05	67	e4	27	41	8b	a7	7d	95	c8
4	fb	ee	7c	66	dd	17	47	9e	ca	2d	bf	07	ad	5a	83	33
5	63	02	aa	71	c8	19	49	c9	f2	e3	5b	88	9a	26	32	b0
6	e9	0f	d5	80	be	cd	34	48	ff	7a	90	5f	20	68	1a	ae
7	b4	54	93	22	64	f1	73	12	40	08	c3	ec	db	a1	8d	3d
8	97	00	cf	2b	76	82	d6	1b	b5	af	6a	50	45	f3	30	ef
9	3f	55	a2	ea	65	ba	2f	c0	de	1c	fd	4d	92	75	06	8a
a	b2	e6	0e	1f	62	d4	a8	96	f9	c5	25	59	84	72	39	4c
b	5e	78	38	8c	c1	a5	e2	61	b3	21	9c	1e	43	c7	fc	04
c	51	99	6d	0d	fa	df	7e	24	3b	ab	ce	11	8f	4e	b7	eb
d	3c	81	94	f7	b9	13	2c	d3	e7	6e	c4	03	56	44	7f	a9
e	2a	bb	c1	53	dc	0b	9d	6c	31	74	f6	46	ac	89	14	e1
f	16	3a	69	09	70	b6	c0	ed	cc	42	98	a4	28	5c	f8	86

For the construction of WHIRLPOOL's S-Box, the reader is referred to Exercises 4.9.10, 4.9.11, and 4.12.

Let $A = (a_{i,j})_{8 \times 8}$ be an 8×8 state matrix of bytes. Let

$$x = x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7$$

be an eight-bit string, where each $x_i \in \{0, 1\}$. Let $\pi_1(x)$ denote the decimal value of the binary string $x_0 x_1 x_2 x_3$ and $\pi_2(x)$ denote the decimal value of the binary string $x_4 x_5 x_6 x_7$. Define a substitution function S on x by

$$S(x) = s_{\pi_1(x), \pi_2(x)},$$

where $s_{u,v}$ is the byte at the uth row and the vth column in WHIRLPOOL's S-Box and $0 \leq u, v \leq 7$. WHIRLPOOL's operation *sub* of substitute-bytes is defined as follows:

$$sub(A) = (S(a_{i,j}))_{8 \times 8}.$$

Shift Columns

The shift-columns operation *shc* in WHIRLPOOL is similar to the shift-rows operation *shr* in AES, except that the columns rather than the rows are shifted. In particular, the jth column is circularly shifted down j bytes, where $j = 0, 1, \cdots, 7$.

Mix Rows

The mix-rows operation *mir* in WHIRLPOOL is similar to the mix-columns operation *mic* in AES. In particular, it uses the following constant matrix, where each row, starting from the

second row, is a circular right shift of the previous row:

$$\mathbf{\Delta} = \begin{bmatrix} 01 & 01 & 04 & 01 & 08 & 05 & 02 & 09 \\ 09 & 01 & 01 & 04 & 01 & 08 & 05 & 02 \\ 02 & 09 & 01 & 01 & 04 & 01 & 08 & 05 \\ 05 & 02 & 09 & 01 & 01 & 04 & 01 & 08 \\ 08 & 05 & 02 & 09 & 01 & 01 & 04 & 01 \\ 01 & 08 & 05 & 02 & 09 & 01 & 01 & 04 \\ 04 & 01 & 08 & 05 & 02 & 09 & 01 & 01 \\ 01 & 04 & 01 & 08 & 05 & 02 & 09 & 01 \end{bmatrix}$$

Then *mir* is defined by

$$mir(\mathbf{A}) = \mathbf{A} \cdot \mathbf{\Delta},$$

where the scalar addition operation and the scalar multiplication operation on bytes are the same as those defined in AES.

Add Round Constant and Add Round Key

The add-round-constant operation *arc* and the add-round-key operation *ark* in WHIRLPOOL are the same as the add-round-key operation in AES. In particular,

$$arc(\mathbf{A}, \mathbf{RC}_i) = \mathbf{A} \oplus \mathbf{RC}_i,$$

$$ark(\mathbf{A}, \mathbf{K}_i) = \mathbf{A} \oplus \mathbf{K}_i,$$

where the $\oplus$ operation on two matrices is the $\oplus$ operation on bytes of corresponding elements in the matrices.

4.1.6   SHA-3 Standard

In 2007, NIST launched a competition to define a secure hash algorithm to provide an alternative to SHA-2. The SHA-2 hash functions are still considered secure by the NIST, and so NIST required that the SHA-3 standard be drop-in compatible with any system that uses SHA-2. This means that SHA-3 must support output sizes of 224, 256, 384, and 512 bits.

   In 2012, NIST announced that the KECCAK family of hash functions devised by Bertoni, Daemen, Peeters, and Van Assche was selected as the SHA-3 standard. This construction deviates from the traditional CBC mode of repeated applications of compression structure used by SHA-2 and WHIRLPOOL. Instead, it uses a sponge construction originating from the work of the KECCAK team.

4.1.6.1   Sponge Functions

A sponge function takes a variable length binary string as input and constructs an arbitrary length binary string as output. The heart of a sponge function is a fixed length permutation. In

particular, a sponge function consists of three phases: a *setup phase*, an *absorb phase*, and a *squeeze phase*.

Let M be the input string and the hash output length be r. Let $b = 25 \times 2^\ell$ with $0 \leq \ell \leq 6$. Thus, $b \in \{25, 50, 100, 200, 400, 800, 1600\}$. Given a value of b, write $b = r + c$, where both r and c are positive integers, r is called the *rate*, and c is called the *capacity*. In general, the selection of r is arbitrary, but the capacity c should be sufficiently large to make the hash function secure. SHA-3 specifies that $c = 2\gamma$, where γ is the length of the hash value.

For example, if we want the hash value to have length $\gamma = 224$, then we have $c = 2\gamma = 448$, and so we may choose $b = 800$ with $r = 352$.

Setup

In the setup phase, we first pad M by appending at most $r - 1$ bits in the form of 10^*1 for a new string M' such that $|M|'$ is divisible by r. Let $N = |M|'/r$. Divide M' into N blocks such that each block is r-bit long. Denote these blocks by

$$M_1, M_2, \cdots, M_N.$$

Let A be a b-bit string. Divide A into 25 substrings of length $m = 2^\ell$, and rewrite the 25 substrings into a 5×5 state matrix listed row wise, denoted by $\boldsymbol{A} = (a_{i,j})_{5 \times 5}$. Let $a_{i,j,k}$ denote the kth bit in $a_{i,j}$.

Absorb

In the absorb phase, each block is absorbed by the XOR operation with the current state, followed by the application of a fixed-length permutation f_b on b-bit inputs, which will be described later. In particular, let A_i be the current state, which is a binary string from the current state matrix $\boldsymbol{A}_i$ by listing all the entries row wise. Initially, $A_0 = 0^b$. Let M_i be the current block. Recall that the size of M_i is r, $\mathrm{p}_r(X)$ returns the r-bit prefix of string X, and $\mathrm{s}_c(X)$ returns the c-bit suffix of X. Then

$$A_i = f_b((\mathrm{p}_r(M_i \oplus A_{i-1}) \parallel \mathrm{s}_c(A_{i-1})),$$
$$i = 1, \cdots, N.$$
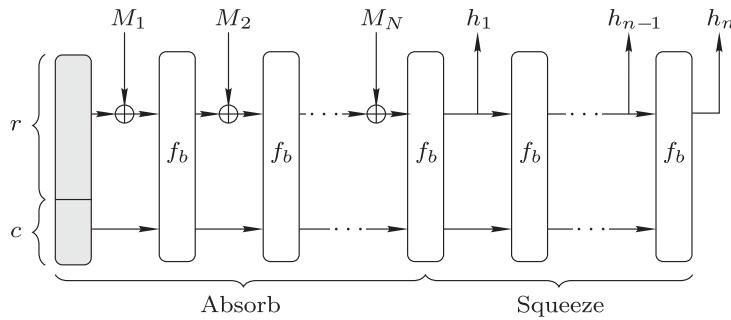
Squeeze

In the squeeze phase, the permutation f_b iterates itself a few times on the initial input of A_N until the output string is at least as long as the desired hash value. Let n denote the number of iterations. Each iteration generates a new string of length b as follows:

$$A_{N+i} = f_b(A_{N+i-1}),$$
$$i = 1, \cdots, n.$$

Let

$$h_i = \mathrm{p}_r(A_{N+i-1}),$$
$$i = 1, \cdots, n.$$

The output of the squeeze phase is $h_1 \cdots h_n$ with length rn. For example, suppose that the length of the desired hash value is $\gamma = 224$, then because in this case $r > \gamma$, we can simply

Figure 4.3 The sponge function construction using a b-bit permutation f_b, where $b = r + c$. The state is initially set to 0^b and the result is an rn-bit string

choose $n = 1$. For another example, if $\gamma = 160$, then $c = 2\gamma = 320$, then we may choose $b = 400$ with $r = 80$. In this case, we may choose $n = 2$.

Basic Sponge Structure
Figure 4.3 depicts the basic structure of the sponge function.

Hash Value
Let H_γ denote the SHA-3 hash function with output length γ. Then

$$H_\gamma(M) = \mathrm{p}_\gamma(h_1 \parallel \cdots \parallel h_n).$$

4.1.6.2 The KECCAK Family of Permutations

We now describe the KECCAK family of permutations f_b, where $b = 25 \times 2^\ell$ with $0 \le \ell \le 6$, and $b = r + c$.

The permutation f_b takes a 5×5 state matrix $\boldsymbol{A}$ as input and carries out the following five operations, where indices are computed modulo 4 and 2^ℓ where appropriate.

1. *Diffusion*: For all $0 \le i, j \le 4$ and $0 \le k \le 2^\ell - 1$, compute

$$a_{i,j,k} = a_{i,j,k} \oplus \bigoplus_{y=0}^{4} a_{i-1,y,k} \oplus \bigoplus_{y=0}^{4} a_{i+1,y,k-1}.$$

2. *Dispersion* (of bits in words): This operation is the following sequence of 24 steps:
    (a) Set $i = 1$ and $j = 0$.
    (b) For $t = 0$ to 23 do
        i. $a_{i,j} = a_{i,j} \oplus ((t+1)(t+2)/2)$.
        ii. Set $i = j$ and $j = (2i + 3j) \bmod 5$.
3. *Dispersion* (of words): For all $0 \le i, j \le 4$, compute

$$a_{i,(2i+3j) \bmod 5} = a_{i,j}.$$

Table 4.2 Round constants for the symmetry disruption phase of the f_b for $\ell = 6$. For $\ell < 6$, use the prefix of these round constants to obtain the round constants of appropriate length

l	Value	l	Value
0	0x0000000000000001	12	0x000000008000808B
1	0x0000000000008082	13	0x800000000000008B
2	0x800000000000808A	14	0x8000000000008089
3	0x8000000080008000	15	0x8000000000008003
4	0x000000000000808B	16	0x8000000000008002
5	0x0000000080000001	17	0x8000000000000080
6	0x8000000080008081	18	0x000000000000800A
7	0x8000000000008009	19	0x800000008000000A
8	0x000000000000008A	20	0x8000000080008081
9	0x0000000000000088	21	0x8000000000008080
10	0x0000000080008009	22	0x0000000080000001
11	0x000000008000000A	23	0x8000000080008008

4. *Nonlinear Map*: For each $0 \leq i, j \leq 4$, compute

$$a_{i,j} = a_{i,j} \oplus (\overline{a_{(i+1) \bmod 5, j}} \wedge a_{(i+2) \bmod 5, j}).$$

This map provides resistance to linear cryptanalysis.

5. *Symmetry Disruption*: During the lth round, compute

$$a_{0,0} = a_{0,0} \oplus RC_l,$$

where RC_l is the round constant for the lth round.

Table 4.2 lists the round constants for $\ell = 6$, where the length of each word is 64-bit long.

Generating Round Constants

The round constants for the permutation f_b are generated using a Linear Feedback Shift Register (LFSR). We call the initial value of the word, stored in the register, the *seed*.

A Galois LFSR is used to produce the round constants using a homomorphism

$$\varphi : GF(2)[x]/(x^8 + x^6 + x^5 + x^4 + 1) \rightarrow GF(2).$$

In particular, let $RC_{l,k}$ denote the k-bit of RC_l. Then we have, for $0 \leq j \leq \ell$,

$$RC_{l,2^j-1} = rc_{j+7l},$$

where rc_t is the output of the LFSR defined by

$$rc_t = (x^t \bmod x^8 + x^6 + x^5 + x^4 + 1) \bmod x.$$

All other bits of RC_r are zero.

For example, rc_9 is computed as follows:

$$rc_9 = (x^9 \bmod x^8 + x^6 + x^5 + x^4 + 1) \bmod x$$
$$= (-x^6 - x^5 - x^4 - x) \bmod x$$
$$= (x^6 + x^5 + x^4 + x) \bmod x$$
$$= 0.$$

4.2 Cryptographic Checksums

Checksums are commonly used to detect transmission errors in network communications. For instance, the IP header in IPv4 contains a 16-bit 1's complement sum and the Ethernet frame contains a 32-bit Cyclic Redundancy Check (CRC). However, these checksums cannot be used to authenticate data or be used as fingerprints, for it is easy to find a different string to have the same checksum as that of the given string. We can, however, use symmetric-key encryption algorithms to generate cryptographic checksums to authenticate data. Cryptographic checksums are also known as Message Authentication Codes (MAC).

4.2.1 Exclusive-OR Cryptographic Checksums

Let $M = M_1 M_2 \cdots M_k$ be a message, where each M_i (after appropriate padding) is a 128-bit binary string. Let E denote the AES-128 encryption algorithm and K an AES-128 secret key. Let

$$H_\oplus(M) = M_1 \oplus M_2 \oplus \cdots \oplus M_k.$$

Then $\mathrm{MAC}(M) = E_K(H_\oplus(M))$ is the MAC code for M. However, this method is insecure, for it is vulnerable to a man-in-the-middle attack. For example, suppose that Alice and Bob share the same AES-128 key K. If Alice sends $(M, E_K(H_\oplus(M)))$ to Bob to authenticate M and Malice intercepts it, then Malice can use $E_K(H_\oplus(M))$ to impersonate Alice to authenticate almost any message. This can be done as follows: let $M' = Y_1 Y_2 \cdots Y_\ell$ be an arbitrary message, where each Y_i (after appropriate padding) is a 128-bit binary string. Let

$$Y = Y_1 \oplus Y_2 \oplus \cdots \oplus Y_\ell \oplus H_\oplus(M),$$
$$M'' = M' \parallel Y.$$

Note that M'' is basically the same message as M'. Malice sends

$$(M'', E_K(H_\oplus(M)))$$

to Bob to make him believe that M'' comes from Alice as follows. According to the authentication protocol, Bob first computes

$$H_\oplus(M'') = Y_1 \oplus Y_2 \oplus \cdots \oplus Y_\ell \oplus Y = H_\oplus(M).$$

He then decrypts $E_K(H_\oplus(M))$ to get $H_\oplus(M) = H_\oplus(M'')$, and so Bob would have to believe that M'' comes from Alice.

4.2.2 Design Criteria of MAC Algorithms

Let $\mathrm{MAC}_K(M)$ denote M's MAC code, where K is a secret key. We require that $\mathrm{MAC}_K(M)$ satisfy the following four criteria:

1. *Forward efficiency*: Computing $\mathrm{MAC}_K(M)$ is easy and efficient.
2. *Backward intractability:* It is computationally difficult to compute M from $\mathrm{MAC}_K(M)$.
3. *Computational uniqueness:* It is computationally difficult to find $M' \neq M$ from $(M, \mathrm{MAC}_K(M))$ such that $\mathrm{MAC}_K(M') = \mathrm{MAC}_K(M)$.
4. *Uniform distribution:* Let k be the length of the MAC code. Let M be a string selected uniformly at random. Let $M' \neq M$ be a string, where M' is either selected at random independently of M or transformed from M (e.g., using a transformation function f such that $M' = f(M)$). Then the probability of $\mathrm{MAC}_K(M') = \mathrm{MAC}_K(M)$ is 2^{-k}.

Despite intensive efforts, there have been no known MAC algorithms proven to satisfy these four criteria. We can, however, use standard encryption algorithms and standard hash functions to construct message authentication codes to meet the needs in practical applications.

4.2.3 Data Authentication Algorithm

In 1985, the NIST established a data authentication code standard DAC. It is based on DES under CBC mode.

Let $M = M_1 M_2 \cdots M_k$, where each M_i (after appropriate padding) is a 64-bit binary string. Let K be a DES key and E be DES encryption algorithm. Let

$$C_1 = E_K(M_1),$$
$$C_i = E_K(M_i \oplus C_{i-1}),$$
$$i = 2, \cdots, k.$$

Then $\mathrm{DAC} = C_k$.

As DES is phasing out, DAC has been replaced with a new authentication scheme called *Keyed-Hash Message Authentication Code* (*HMAC*).

4.3 HMAC

HMAC is an algorithmic scheme. It uses a hash function and a symmetric-key encryption algorithm to generate authentication codes. The basic idea of HMAC is to embed the secret information of the key into the data and then compute a hash value from it.

4.3.1 Design Criteria of HMAC

1. Any reasonable hash function can be deployed directly, that is, without any modification, in HMAC.

2. Any cryptographic hash function deployed in HMAC should maintain its basic properties, including the one-way property (i.e., forward efficiency and backward intractability) and the computational uniqueness property.
3. The use of secret keys is simple.
4. Analysis of the strength of a HMAC code can be obtained from analyzing the strength of the hash function deployed.

4.3.2   HMAC Algorithm

HMAC takes the following parameters:

H: a hash function to be embedded (e.g., SHA-512 and WHIRLPOOL).
IV: the initial vector of H.
M: the message to be authenticated.
L: the number of blocks of M.
ℓ: the output length of $H(M)$.
b: the number of bits in a block, which is divisible by 8. It is required that $b \geq \ell$.
K: the secret key with a length $\leq b$. (If $|K| > b$, then let $K \leftarrow H(K)$ such that $|K| = \ell$.)
K': $K' = 0^{b-|K|}K$ is the prefix padding of K with $|K'| = b$.
ipad: ipad $= (00110110)^{b/8}$.
opad: opad $= (01011100)^{b/8}$.
K'_0: $K'_0 = K' \oplus$ ipad. (K'_0 reverse one-half of the bits in K'.)
K'_1: $K'_1 = K' \oplus$ opad. (K'_1 also reverses one-half of the bits in K'.)

The HMAC algorithm is given as follows:

$$\text{HMAC}(K, M) = H(K'_1 \parallel H(K'_0 \parallel M)). \tag{4.1}$$

We use HMAC-SHA-512 to denote the HMAC implementation with SHA-512 as the embedded hash function. Likewise, we use HMAC-WHIRLPOOL to denote the HMAC implementation with WHIRLPOOL as the embedded hash function.

It can be shown that the strength of an HMAC implementation is closely related to that of the underlying hash function it deploys.

4.4   Birthday Attacks

Suppose that we want to know whether in a group of people there are two persons who were born on the same day in the same month. If each person has the same likelihood of being born on any one of the 365 days in a year (for simplicity, we assume there are no leap years), we can show that in a group of 23 people, the probability that there are at least two persons born on the same day in the same month is greater than 1/2. To see why this is true, let us fix an order to these 23 people. Then the probability that the second person's birthday differs from the first person's is $364/365$. Likewise, the probability that the ith person's birthday differs from any of the previous $i - 1$ persons' birthdays is $(365 - i + 1)/365$. Thus, the probability

that none of the 23 people has the same birthday is

$$\left(\frac{364}{365}\right) \times \left(\frac{363}{365}\right) \times \cdots \times \left(\frac{343}{365}\right) < 0.493.$$

Hence, the probability that in a group of 23 people there are at least two people with the same birthday is greater than $1 - 0.493 = 0.507 > 1/2$. We generalize this idea to study the complexity upper bound of breaking the strong collision resistance of hash functions.

4.4.1   Complexity of Breaking Strong Collision Resistance

Let H be a hash function with a fixed output length. Thus, H has a fixed number of different outputs. Without loss of generality, we assume that H has exactly n different outputs. Note that n is a power of 2 if H is a cryptographic hash function, but it does not have to be so in other cases.

Select k inputs uniformly and independently at random and list them as $y_1, y_2, \cdots, y_k$. Then the probability that there are two indexes i and j with $j < i$ such that $H(y_i) = H(y_j)$ is $(i-1)/n$. Hence, for any i with $2 \leq i \leq k$, the probability that $H(y_i)$ differs from any $H(y_j)$ with $j < i$ is

$$1 - \frac{i-1}{n}.$$

Therefore, the probability that none of these k strings collides, that is, $H(y_i) \neq H(y_j)$ for any $j < i \leq k$, is

$$\left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right)\cdots\left(1 - \frac{k-1}{n}\right).$$

Let $P(n,k)$ denote the probability that a collision occurs in k strings under H. That is, $P(n,k)$ is the probability that there are two strings y_i and y_j with $i \neq j$ such that $H(y_i) = H(y_j)$. Then

$$P(n,k) = 1 - \left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right)\cdots\left(1 - \frac{k-1}{n}\right). \tag{4.2}$$

To compute $P(n,k)$, we use the following inequality:

$$1 - x < e^{-x}, \tag{4.3}$$

where x is any positive real number. The proof of this inequality can be found in most college calculus textbooks.

It follows from Equality 4.2 and Inequality 4.3 that

$$P(n,k) > 1 - e^{-1/n}e^{-2/n}\cdots e^{-(k-1)/n}$$

$$= 1 - e^{-\frac{k(k-1)}{2n}}.$$

Let $1 - e^{-\frac{k(k-1)}{2n}} = 1/2$. Then $e^{-\frac{k(k-1)}{2n}} = 1/2$. Thus,

$$\frac{k(k-1)}{2n} = \ln 2. \tag{4.4}$$

Solving Quadratic Equation 4.4, we get

$$k = \frac{1 + \sqrt{1 + 8\ln 2 \cdot n}}{2}. \tag{4.5}$$

When $n \geq 52$, because $52 > 36/\ln 2$, we have

$$\sqrt{8\ln 2 \cdot n} < 1 + \sqrt{1 + 8\ln 2 \cdot n} < \sqrt{9\ln 2 \cdot n}. \tag{4.6}$$

Thus, from Equality 4.5 and Inequality 4.6 we get

$$\frac{\sqrt{8\ln 2 \cdot n}}{2} < k < \frac{\sqrt{9\ln 2 \cdot n}}{2}, \tag{4.7}$$

$$1.17\sqrt{n} < k < 1.25\sqrt{n}. \tag{4.8}$$

This implies that when $1.17\sqrt{n} < k < 1.25\sqrt{n}$, the probability that a collision occurs in k inputs under H is greater than 1/2. For example, when $n = 356$, we have $1.25\sqrt{365} < 23.89$ and $1.17\sqrt{365} > 22.35$. Any integer between 22.35 and 23.89 can only be 23. This verifies what we showed at the beginning of this section: the probability of at least two persons who were born on the same day and in the same month in a group of 23 people is greater than 1/2.

The computations in this section give rise to the following *birthday paradox*.

Birthday Paradox. *From a basket of n balls of different colors, pick k ($k < n$) balls uniformly and independently at random and record their colors (i.e., after a ball is picked, record its color, and put it back in the basket before the next ball is picked). If $1.17\sqrt{n} < k < 1.25\sqrt{n}$, then with probability at least 1/2, there is at least one ball that is picked more than once.*

Let H be a cryptographic hash function with output length ℓ. Then $n = 2^\ell$. Thus, if we select $1.25 \cdot 2^{\ell/2} \approx 2^{\ell/2}$ input strings uniformly and independently at random, the probability that there are two different strings x and y such that $H(x) = H(y)$ is greater than 1/2. This indicates that the complexity upper bound of the strong collision resistance of H does not exceed the complexity of searching $2^{\ell/2}$ random strings.

For example, the complexity upper bound of strong collision resistance for SHA-1 is $2^{160/2} = 2^{80}$ and for SHA-512 is $2^{512/2} = 2^{256}$.

4.4.2   Set Intersection Attack

Suppose that we select uniformly and independently at random two sets of integers from $\{1, 2, \cdots, n\}$, with k integers in each set, where $k < n$. What is the probability $Q(n, k)$ that these two sets intersect?

Let $A = \{a_1, a_2, \cdots, a_k\}$ and $B = \{b_1, b_2, \cdots, b_k\}$ denote these two sets. As the probability that $b_1 = a_1$ is $1/n$, we know that the probability that $b_1 \neq a_1$ is $1 - 1/n$. Likewise, the probability that none of the k elements in B equals a_1 is $(1 - 1/n)^k$. Thus, the probability that B and A disjoin is

$$\left[\left(1 - \frac{1}{n}\right)^k\right]^k = \left(1 - \frac{1}{n}\right)^{k^2} < (e^{-1/n})^{k^2},$$

where the inequality at the right-hand side follows from Inequality 4.3. Hence,

$$Q(n, k) = 1 - \left(1 - \frac{1}{n}\right)^{k^2} > 1 - e^{-k^2/n}.$$

Let $1 - e^{-k^2/n} = 1/2$. Then $\ln 2 = k^2/n$. Hence, $k = \sqrt{\ln 2 \cdot n}$. That is,

$$0.69\sqrt{n} < k < 0.7\sqrt{n}. \tag{4.9}$$

In other words, if $0.69\sqrt{n} < k < 0.7\sqrt{n}$, then $Q(n, k) > 1/2$.

The set intersection attack is a form of birthday attack. In this attack, the attacker first uses a legitimate document $\mathcal{D}$ to obtain AU's signature, where AU is an authentication authority (e.g., a company CEO). That is, the attacker uses $\mathcal{D}$ to obtain $C = \langle H(\mathcal{D}) \rangle_{K^r_{\text{AU}}}$, where $\langle H(\mathcal{D}) \rangle_{K^r_{\text{AU}}}$ represents AU's signed copy of $H(\mathcal{D})$ using his private key K^r_{AU}.

The attacker then produces a new document $\mathcal{F}$ that has different meanings from $\mathcal{D}$ such that $H(\mathcal{F}) = H(\mathcal{D})$. The attacker can then use $(\mathcal{F}, C)$ to show that $\mathcal{F}$ is endorsed by AU.

Suppose that AU is willing to sign any legitimate document. The attacker may launch a set intersection attack as follows:

1. By assumption, AU will sign any legitimate document $\mathcal{D}$ or any document that has the same meaning as $\mathcal{D}$. Let the output length of H be ℓ bits, which is fixed regardless of the length of the input. The attacker prepares a set S_1 of $2^{\ell/2}$ different documents, all having the same meaning as $\mathcal{D}$. These documents can be obtained, for example, using one or more of the following methods:

   (a) Replace a word or a phrase in $\mathcal{D}$ with a synonym or a synonymic phrase.
   (b) Rephrase sentences in $\mathcal{D}$.
   (c) Use different punctuation.
   (d) Reorganize the structure of $\mathcal{D}$.
   (e) Change passive tense to active, or active to passive.

2. By assumption, AU will refuse to sign any malicious document. To obtain AU's signature on a malicious document $\mathcal{F}$, the attacker prepares a set S_2 of $2^{\ell/2}$ different documents, all having the same meaning as $\mathcal{F}$.

3. The attacker computes

$$H(S_1) = \{H(X) \mid X \in S_1\},$$
$$H(S_2) = \{H(X) \mid X \in S_2\}.$$

Although the documents in S_1 (similarly in S_2) are not generated uniformly or independently at random, their hash values are distributed uniformly and independently. This is the property of a cryptographic hash function. Thus, the probability that there is a document $\mathcal{D}' \in S_1$ and a document $\mathcal{F}' \in S_2$ such that $H(\mathcal{D}') = H(\mathcal{F}')$ is greater than 1/2.

Assuming the attacker has found such two documents, he presents $\mathcal{D}'$ to AU for AU's signature.

If such a pair of documents $(\mathcal{D}', \mathcal{F}')$ does not exist in this round, the attacker repeats the same procedure until he finds such a pair.

The set intersection attack also indicates that the complexity of breaking the strong collision resistance of any cryptographic hash function has an upper bound in the magnitude of $2^{\ell/2}$.

4.5   Digital Signature Standard

PKC is the most effective mechanism to produce a digital signature for a given document. Suppose that Alice wants to sign a document M. She first computes $H(M)$, where H is a cryptographic hash function. She then encrypts $H(M)$ with her private key K_A^r to get $E_{K_A^r}(H(M))$. For Bob to verify that she has signed the document M, Alice will present $(M, E_{K_A^r}(H(M)))$ to Bob, together with her public-key certificate $\mathrm{CA}\langle K_A^u\rangle$. Bob retrieves K_A^u from Alice's certificate, decrypts $E_{K_A^r}(H(M))$ to get h using K_A^u, computes $H(M)$ on the copy of M he receives, and verifies that $h = H(M)$ to confirm Alice's signature.

RSA and ECC would be natural choices of PKC for digital signatures. RSA, however, was under patent protection from 1978 to 2000. NIST, being a government agency, should not establish standards using patented algorithms to benefit patentees, and so it established a digital signature standard (DSS) using a different digital signature algorithm. DSS was first published in 1991. It was modified once in 1994 and once in 1996. After RSA's patent protection ended in 2000, NIST included RSA and ECC as part of DSS. The DSS algorithm introduced in this section is the original DSS published in 1996.

Note that DSS can only be used to generate digital signatures. That is, it cannot be used to encrypt data. DSS uses SHA-1 to compute a 160-bit hash value. The reader is referred to Exercises 4.8 and 4.9 for a description of SHA-1. In particular, DSS uses the following three global parameters:

p: a prime number with $2^{L-1} < p < 2^L$, where $512 < L < 1024$
  and L is divisible by 64.
q: a prime number and a factor of $p - 1$, with $2^{159} < q < 2^{160}$.
g: $g = h^{(p-1)/q} \bmod p$, where $1 < h < p - 1$ is an integer with $g > 1$.

To use DSS, each user must first select at random a positive integer $x < q$ as a private key. After this selection, the user then computes $y = g^x \bmod p$ as the public key and obtains a public-key certificate $\mathrm{CA}\langle y\rangle$ for y.

4.5.1   Signing

Suppose that Alice's private key is x_A and her public-key is y_A. To digitally sign a document M, Alice first selects at random a positive integer $k_A < q$. She then computes

$$r_A = (g^{k_A} \bmod p) \bmod q,$$
$$k_A^{-1} = k_A^{q-2} \bmod q,$$
$$s_A = [k_A^{-1} \cdot (H(M) + x_A \cdot r_A)] \bmod q,$$

and uses (r_A, s_A) as M's digital signature. We call this procedure a *signing procedure* or simply *signing*.

4.5.2 Signature Verifying

Suppose that Bob obtains $(M', (r'_A, s'_A))$ from Alice and her certificate $\text{CA}\langle y_A \rangle$. Bob first obtains Alice's public-key y_A using CA's public key to decrypt $\text{CA}\langle y_A \rangle$. He then carries out the following computations and verifies Alice's digital signature:

$$w = (s'_A)^{-1} \quad \text{mod } q = (s'_A)^{q-2} \text{ mod } p,$$

$$u_1 = (H(M') \cdot w) \text{ mod } q,$$

$$u_2 = (r'_A \cdot w) \text{ mod } q,$$

$$v = [(g^{u_1} \cdot y_A^{u_2}) \text{ mod } p] \text{ mod } q.$$

If $v = r'_A$, then Bob can be confident that M'_A does indeed bear Alice's signature. This computation is referred to as *signature verification*.

4.5.3 Correctness Proof of Signature Verification

We now prove the correctness of signature verification. That is, we show that if $M' = M, r'_A = r_A, s'_A = s_A$, then $v = r'_A$.

Note that $\gcd(h, p) = 1$, for $1 < h < p - 1$. It follows from Fermat's little theorem that $h^{p-1} \text{ mod } p = 1$. Hence,

$$g^q \text{ mod } p = (h^{(p-1)/q} \text{ mod } p)^q \text{ mod } p$$

$$= h^{p-1} \text{ mod } p$$

$$= 1.$$

If m and n are positive integers and $m = n \text{ mod } q$, then there is an integer k such that $m = n + kq$. Hence,

$$g^m \text{ mod } p = g^{n+kq} \text{ mod } p$$

$$= (g^n g^{kq}) \text{ mod } p$$

$$= [(g^n \text{ mod } p)(g^q \text{ mod } p)^k] \text{ mod } p$$

$$= (g^n \text{ mod } p) \cdot 1^k \text{ mod } p$$

$$= g^n \text{ mod } p.$$

As $M' = M, s'_A = s_A$, and $r'_A = r_A$, we have

$$w = s_A^{-1} \text{ mod } q, \tag{4.10}$$

$$u_1 = (H(M) \cdot w) \text{ mod } q, \tag{4.11}$$

$$u_2 = (r_A \cdot w) \text{ mod } q. \tag{4.12}$$

It follows from Equality 4.11 that

$$g^{u_1} \text{ mod } p = g^{H(M) \cdot w} \text{ mod } p.$$

It follows from Equality 4.12 that

$$x_A \cdot u2 = x_A \cdot r_A \cdot w \text{ mod } q.$$

Thus,

$$y_A^{u_2} \bmod p = g^{x_A \cdot u_2} \bmod p = g^{x_A \cdot r_A \cdot w} \bmod p.$$

We therefore have

$$
\begin{aligned}
v &= [(g^{u_1} \cdot y_A^{u_2}) \bmod p] \bmod q \\
&= \left[\left(g^{H(M) \cdot w} \cdot g^{x_A \cdot r_A \cdot w}\right) \bmod p\right] \bmod q \\
&= \left[\left(g^{H(M) \cdot w} \cdot g^{x_A \cdot r_A \cdot w}\right) \bmod q\right] \bmod p \\
&= \left[\left(g^{(H(M) + x_A \cdot r_A)w}\right) \bmod q\right] \bmod p.
\end{aligned}
$$

As $s_A = [k_A^{-1}(H(M) + x_A \cdot r_A)] \bmod q$, we have

$$w = s_A^{-1} \bmod q = [k_A(H(M) + x_A \cdot r_A)^{-1}] \bmod q.$$

This implies that $(H(M) + x_A \cdot r_A)w = k_A \bmod q$. Therefore,

$$v = (g^{k_A} \bmod p) \bmod q = r_A.$$

This completes the proof.

4.5.4   Security Strength of DSS

The security strength of DSS rests on the strength of SHA-1 and the difficulty of solving discrete log. The complexity of breaking the strong collision resistance of SHA-1 has been reduced from 2^{80} to 2^{63}. However, breaking the collision resistance (i.e., given $(M, H(M))$ find $M' \neq M$ such that $H(M') = H(M)$) is harder. Intractability of discrete log ensures that it is difficult to compute k_A or x_A from r_A and s_A.

4.6   Dual Signatures and Electronic Transactions

Suppose that Alice has a message I_1 and wants Bob to act according to what it says. So she sends I_1 to Bob. However, for Bob to act on I_1, he must wait for Charlie to tell him it is okay to do so. Charlie, on the other hand, needs Alice to send him a separate message I_2 to convince him that he can give Bob the go-ahead message in which I_2 must be linked with I_1, but Charlie is not allowed to see what I_1 is. Moreover, all messages must be properly authenticated and remain confidential during transmissions.

  Dual signature is an interactive authentication protocol for solving this problem. In particular, it allows Alice to encrypt both I_1 and I_2, sign them using her private key, and send them to Bob. However, she only allows Bob to read I_1. In other words, Bob cannot decrypt I_2. Bob verifies that he received both I_1 and I_2 from Alice. He then transmits them to Charlie, but he only allows Charlie to read I_2. Charlie verifies that both I_1 and I_2 come from Alice through Bob. He then sends a receipt R_C to Bob, telling Bob whether he approves I_2. Bob verifies that he received R_C from Charlie. He then sends a receipt R_B to Alice, telling Alice whether he will act on I_1.

4.6.1   Dual Signature Applications

Dual signature may be used in online shopping applications to provide security and privacy protections, where Alice is a customer, Bob a merchant, and Charlie a banker.

Alice browses Bob's online store, creates an order list I_1, fills in her payment information I_2 (credit card number, name, expiration date, etc.), and sends both I_1 and I_2 to Bob. Bob first verifies that they indeed come from Alice. He then reads Alice's order list I_1 and sends both I_1 and I_2 (which he cannot read) to Charlie. Charlie first verifies that I_1 and I_2 come from Alice through Bob. He then reads Alice's payment information I_2. Note that Charlie cannot read Alice's order list. Depending on whether the information provided in I_2 is correct and whether Alice has a sufficient credit line, Charlie issues a receipt R_C and sends it to Bob, telling Bob whether he will be paid for selling the items to Alice on her order list I_1. Bob verifies that R_C indeed comes from Charlie. If R_C says payment will be made, Bob creates a receipt R_B and sends it to Alice, informing her that her order has been filled.

Requiring that both I_1 and I_2 be linked together prevents separation of a payment from an order, so that nobody can use this payment to pay for a different order. Disallowing Bob from reading I_2 and Charlie from reading I_1 gives Alice better privacy protection than what the current practice provides. In the current practice, merchants can read customers' credit card information, and banks can find out what customers have purchased. Requiring that all messages be authenticated and remain confidential during transmissions ensures that Malice cannot obtain any useful information by eavesdropping and cannot modify or fabricate order information or payment information.

Dual signature is used in the Secure Electronic Transaction Protocol (SET), which was devised in 1996 by two major U.S. credit card companies: Visa and Mastercard. SET is a complex protocol, which has not been deployed in practice.

4.6.2   Dual Signatures and Electronic Transactions

Alice, Bob, and Charlie first agree on a hash function H and a PKC encryption algorithm E (e.g., RSA). Let (K_A^u, K_A^r), (K_B^u, K_B^r), and (K_C^u, K_C^r) be, respectively, the public–private key pair for Alice, Bob, and Charlie. Moreover, they all know each other's public key; for example, they may pass their public-key certificates to each other. Let D be the decryption algorithm of E.

Alice carries out the following steps:

1. Compute

$$s_B = E_{K_B^u}(I_1),$$
$$s_C = E_{K_C^u}(I_2),$$
$$h_B = H(s_B),$$
$$h_C = H(s_C),$$
$$ds = D_{K_A^r}(H(h_B \parallel h_C)),$$

where *ds* is a dual signature.

2. Transmit (s_B, s_C, ds) to Bob.
3. Wait for Bob's receipt R_B, which is in the form of $E_{K_A^u}(D_{K_B^r}(R_B))$ (see Bob's step 5).
4. Decrypt $E_{K_A^u}(D_{K_B^r}(R_B))$ using Alice's private key to get $D_{K_B^r}(R_B)$, and verify Bob's signature using Bob's public key to get R_B. This completes the protocol.

Bob carries out the following steps:

1. Compute $H(H(s_B) \parallel h_C)$ and $E_{K_A^u}(ds)$, and check whether they are identical. If identical, then it implies that (s_B, s_C, ds) indeed comes from Alice.
2. Compute $D_{K_B^r}(s_B)$ to get I_1.
3. Transmit (s_B, s_C, ds) to Charlie.
4. Wait for Charlie's receipt R_C, which is in the form of $E_{K_B^u}(D_{K_C^r}(R_C))$ (see Charlie's step 3).
5. Decrypt $E_{K_B^u}(D_{K_C^r}(R_C))$ using Bob's private key to get $D_{K_C^r}(R_C)$, and verify Charlie's signature using Charlie's public key to get R_C. Issue a receipt R_B and sends $E_{K_A^u}(D_{K_B^r}(R_B))$ to Alice.

Charlie carries out the following steps:

1. Compute $H(H(s_B) \parallel H(s_C))$ and $E_{K_A^u}(ds)$, and check whether they are identical. If identical, then it implies that (s_B, s_C, ds) indeed comes from Alice.
2. Compute $D_{K_C^r}(s_C)$ to get I_2.
3. Issue a receipt R_C and send $E_{K_B^u}(D_{K_C^r}(R_C))$ to Bob.

Note that s_B can only be decrypted by Bob and s_C can only be decrypted by Charlie. In other words, Bob cannot see I_2 and Charlie cannot see I_1. Because of the dual signature, Malice or Bob cannot modify or fabricate Alice's I_1 or I_2. Likewise, Malice or Charlie cannot modify or fabricate Alice's I_1 or I_2 without being caught.

4.7 Blind Signatures and Electronic Cash

A *blind signature* is a signature signed on a message presented in an unintelligible form (i.e., the signer cannot review its content), but the signature can be verified against the original message. In the nondigital world, for example, one can achieve this effect using an envelope: Alice puts a message in an envelope, seals it, and presents the sealed envelope to Bob for him to sign his name along the seal. Because the envelope is sealed before Bob signs it, this has the effect that Bob signs Alice's message enclosed in the envelop without being able to see the content of the message. A digital blind signature can be achieved using the RSA cryptosystem and a *blind factor* as follows, where a blind factor is used to make a message unintelligible.

4.7.1 RSA Blind Signatures

Let n, d, and e be Bob's RSA parameters, where n and e are published. Suppose that Alice wants Bob to sign her message M blind, where $M < n$. She chooses a pseudorandom number

$r < n$ as a blind factor with $\gcd(n, r) = 1$, calculates

$$M_r = M \cdot r^e \bmod n,$$

and presents M_r to Bob. (The r^e in this case is equivalent to a sealed envelop containing M.) Bob signs M_r and produces a signature

$$s_r = M_r^d \bmod n.$$

As $de \equiv 1(\bmod \phi(n))$ and $\gcd(n, r) = 1$, it follows from Fermat's little theorem that

$$r^{de} \equiv r(\bmod n),$$

and that $r^{-1} \bmod n$ exists. Thus, Alice can remove the blind factor from s_r by calculating $(s_r \cdot r^{-1}) \bmod n$ and obtain

$$
\begin{aligned}
s_M &= s_r \cdot r^{-1} \bmod n \\
&= M^d \cdot r^{de} \cdot r^{-1} \bmod n \\
&= M^d \cdot r \cdot r^{-1} \bmod n \\
&= M^d \bmod n.
\end{aligned}
$$

Alice now obtains Bob's signature on M, and Bob does not know the content of M.

4.7.2 Electronic Cash

Using credit cards to pay for goods or services will expose the identities of the credit card holders. This is a major difference between using credit cards and cash, for cash does not link to its owner. Cash payment is anonymous: cash can be owned by anyone, and cash does not reveal its owner's identity. In addition, cash can be circulated. Circulating from one owner to another, cash leaves no obvious trace (except perhaps the owner's fingerprints) to identify who has owned it. Cash can also be divided into cash of smaller values.

Electronic cash is a note in a digital form issued by a bank. It circulates in the world of networks, to be used to pay for goods and services. Anyone who owns electronic cash may go to the bank that issues it to obtain paper money of the same value. Thus, electronic cash should satisfy the following requirements:

1. *Anonymous and untraceable*: Electronic cash may be owned by anyone, and it does not reveal its owner's identity. Electronic cash circulated in the networks should leave no trace to its owners. A person or a bank receiving electronic cash will not be able to identify who has been the owner of it.
2. *Secure*: Electronic cash can be circulated safely in the networks. That is, it cannot be modified or fabricated.
3. *Convenient*: Electronic cash payments do not need to go through any financial institution.
4. *Nonreplicable*: Electronic cash cannot be replicated. That is, there is only one true copy of electronic cash. When electronic cash changes hands, the old owner can no longer use it (although he might possess a digital replica).

5. *Transferable*: Electronic cash can be transferred from one owner to another.
6. *Dividable*: An electronic cash note with a larger value can be divided into several electronic cash notes of smaller values such that the summation of these smaller values is equal to the value of the original note.

Despite intensive efforts, no electronic cash protocols have been devised to satisfy all these requirements. On the other hand, it is possible to devise a electronic cash protocol that satisfies some of the most important requirements. For example, eCash is such a protocol. Devised by David Chaum in the 1980s, eCash uses blind signatures to ensure that it is anonymous and untraceable.

4.7.2.1   eCash

Let B denote a financial institution. Let n, d, and e be B's RSA parameters. For convenience, we assume that B only issues one-dollar-value eCash notes, called an *eCash dollar*, which is equivalent to the worth of one dollar.

Suppose that Alice wants to obtain an eCash dollar from B. She and B perform the following computations:

1. Alice generates a sequence number m to represent this eCash dollar and a pseudorandom number $r < n$ to be used as a blind factor. She then calculates

$$x = mr^e \bmod n,$$

   and sends both x and her bank account number to B.
2. B charges one dollar (plus a service fee) to Alice's account, calculates

$$y = x^{\,d} \bmod n,$$

   and sends y to Alice.
3. Alice calculates
$$z \equiv y \cdot r^{-1} \equiv m^d (\bmod\ n).$$

   That is, z is m with B's signature. Then (m, z) is an eCash dollar for Alice.

Note that B is not able to see m when it signs x, and so B is unable to link m to Alice. This eCash dollar (m, z) is free to change hands. Any person who owns (m, z) may cash it for a dollar from B. This can be done as follows. Suppose that Charlie has (m, z). He transmits (m, z) to B. After receiving (m, z), B first verifies whether z has its signature. That is, B verifies whether

$$z^e \equiv m^{de} \equiv m (\bmod\ n).$$

If the answer is yes and the sequence number m has not been presented, then B records m and sends a dollar to Charlie. Otherwise, (m, z) is invalid.

If Alice pays Bob with this eCash dollar (m, z) to exchange for goods or service that Bob is providing, then Bob should check with B whether z is a signed copy of m and m has not been recorded. If the answer is yes, then Bob knows that this eCash dollar is good. Otherwise, Bob should reject it.

Because only Alice knows r, B or any other person will not be able to connect m to x, and so nobody knows Alice created m. After (m, z) changes hands, it leaves no trace to its previous owner. Thus, eCash is anonymous and untraceable. However, Alice may still keep a copy of (m, z) after she gives it to Bob, and so she may cash it before Bob does.

4.7.3 Bitcoin

Bitcoin is a network protocol. It can be viewed as a mining game, where players are awarded for their successes with prizes. These prizes are called Bitcoins, which posses certain properties of currency. From the currency point of view, Bitcoin differs from eCash in the sense that it is a completely decentralized currency. In other words, there is no trusted bank to maintain balances or issue new currency. This means that there is no centralized trust in the currency. Bitcoins, instead of using a trusted bank to give Bitcoins a real value, use a group of cooperating players to form a *peer-to-peer* network called Bitcoin network. The Bitcoin network maintains a global distributed ledger of transactions. Each transaction itemizes a payment from one player to another. This ledger is called the *block chain*. The Bitcoin network is also responsible for verifying each transaction as it enters the block chain.
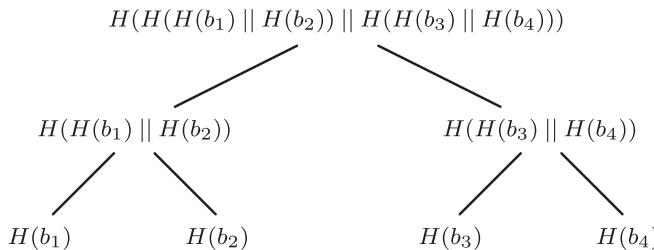
Bitcoin digitally signs individual transactions, not a particular Bitcoin, which differs from eCash. Let BTC denote the Bitcoins unit. Suppose that Alice wants to pay Bob a certain amount of BTC. To do so, she must collect transactions she owns that sum up to a value greater than or equal to the amount she wishes to pay Bob. For example, suppose that Alice wants to pay Bob with c BTC. To do so, Alice must first gather transactions belonging to herself from the global distributed ledger. These transactions must sum up to a value greater than or equal to c BTC. Alice then broadcasts a signed transaction message to all users in the Bitcoin network, listing Bob – represented by Bob's payment address – as the receiver of the payment. The payment address is a function of Bob's public key. The entire signed transaction message is called a *transaction record*. This transaction becomes a new item in the global distributed ledger, and consists of the following components:

- A list of transactions destined to Alice that sum to at least c. These transactions are lines in the global distributed ledger.
- A hash of each transaction that Alice is going to use for payment.
- The payment address for Bob, along with how much BTC from the transactions should be payed to Bob.
- The payment address for Alice, along with how much change Alice should receive from the transaction.
- Finally, the above-mentioned items (except for the pay outs) are hashed and signed with Alice's private key.

A hash is needed in all cases. The Bitcoin protocol currently uses SHA-256. The new transaction record, as previously mentioned, is broadcasted to all members of the network.

4.7.3.1 Mining

To obtain Bitcoins, players must first collect a certain set of transactions from the Bitcoin network. These set of transactions is called a block, and this process is referred to as *mining*.

$$H(H(H(b_1) \parallel H(b_2)) \parallel H(H(b_3) \parallel H(b_4)))$$

$$H(H(b_1) \parallel H(b_2)) \qquad\qquad H(H(b_3) \parallel H(b_4))$$

$$H(b_1) \qquad H(b_2) \qquad\qquad H(b_3) \qquad H(b_4)$$

Figure 4.4 An example of Merkle tree, where H is a hash function and b_i represents a block

The type of players who mine the Bitcoin network is referred to as *miners*. A block consists of the following components:

- A list of verified transactions.
- A hash of those transactions.
- A link to the previous transaction in the block chain.
- A proof that the transactions have been verified.

Miners then need to add the new block to the block chain to be awarded a certain number of Bitcoins.

The transactions listed in the new block have all been verified by the miner. To verify a transaction, the miner checks that the signatures work and that no coins were spent twice.

The transactions in the new block are combined into a single hash value using a *Merkle tree*. A Merkle tree is a hashing scheme that organizes blocks of data into a binary tree where only the leaves contain the data. Every leaf node in the Merkle tree is a hash of the block at that leaf, and every internal node is the hash of its two children (see Fig. 4.4).

To link the new block to the previous block in the block chain, the transaction must contain a reference to the previous block, which is done using a proof-of-work (POW). A POW is, intuitively, a value that is *extremely* hard to generate to meet a certain requirement but very easy to verify. This scheme makes the process of acquiring Bitcoins from the Bitcoin network immensely difficult, and so some people are willing to pay real money to acquire Bitcoins from someone who own them. Some stores also accept Bitcoins as cash payments. In late 2013, a bitcoin could sell for about 6000 Chinese yuan in the Chinese market (about 1000 US dollars).

Proof-of-Work

A POW p for a new block with a difficulty level of k is obtained as follows, where k is a positive integer:

1. Let r be the root of the Merkle tree of the new block, r' the root of the Merkle tree of the previous block, and p a binary string. Let H be a hash function. Compute $H \circ H(r, r', p) = h$, where h is called a double hash.
2. If h begins with k consecutive 0's, then use h as the hash value of the new block and the number p as the proof. Both values are included in the block.
3. If h does not begin with k consecutive 0's, then select a new p and go to Step 1.

When k is large, for example, when $k = 100$, the miner may have to try, in the worst case, at least $2^{100} > 10^{10}$ different values of p to find a proof, making it computationally intractable. On the other hand, if a proof is given, then it is easy to verify that the double hash value is equal to h.

Adding New Blocks

The new block, once constructed, is broadcasted to all other miners in the Bitcoin network. This is done so that all users can agree on the new block being added to the block chain. In the case where two miners each broadcast a new block and one block is a subset of the other, only the block with more transactions is kept. In the rare case that these two blocks have the same number of transactions (i.e., they are the same block), there is a temporary split in the block chain that is eventually resolved by the Bitcoin network. The resolution occurs when one of the forked chains contains more transactions than the competing chain, as miners would always favor the longer chain.

Miners add new Bitcoins using a special transaction called a *coinbase transaction*. When the miner is building a new transaction block, he adds this special coinbase transaction. This transaction lists the miner as the recipient of a certain amount of Bitcoins. At the time of writing this is 25 BTC. The source of this transaction is a special zero address. Recall that the miner only gets paid if the new block gets added to the block chain.

4.8 Closing Remarks

The developments of PKCs, public-key infrastructures, and cryptographic hash functions have made data authentications easy routines. This signature mechanism is more flexible and more reliable than using shared secrets to authenticate data. Thus, it is natural to establish digital signature standards using PKCs and cryptographic hash functions.

4.9 Exercises

4.9.1 Discussions

4.1. Why is PKC a better method to authenticate data?

4.2. Why do we want to compute cryptographic hashing?

4.3. What is the advantage of HMAC?

4.4. Why is birthday attack an efficient method for breaching security?

4.5. Can you think of a different application of blind signature?

4.6. Do you think that Bitcoin could become a popular electronic currency?

4.9.2 Homework

4.1. Find two English sentences with different meanings (substantially different from the example given in Section 4.1), which have the same hash value under the 16-bit XOR-hash function $H_{\oplus}$.

4.2. Let $h = 1001101000111010$ be a 16-bit binary string. Find four different binary strings such that they have the same hash value h under the 16-bit XOR-hash function $H_\oplus$.

4.3. In the initial process of SHA-512, it pads M to obtain a new string M'. Explain how to obtain M from M'.

4.4. Draw a flow diagram of SHA-512 that transforms a 1024-bit block M_i to eighty 64-bit binary strings $W[0, 79]$.

4.5. Draw a flow diagram showing the computations of the SHA-512 compression function $F(M_i, H_{i-1})$. You should show the first round, the last round, and the ith round of computations, where i represents a round between the first and the last round.

4.6. Draw a flow diagram to show the computation in one round of the SHA-512 compression function $F(M_i, H_{i-1})$.

*4.7. How does SHA-512 attempt to achieve the one-way property and the computational uniqueness property? Can you find any weakness? Justify your answers.

4.8. Let M be a L-bit binary string with $L < 2^{64}$. Pad M to yield the following new binary string

$$M' = M \parallel 10^\ell \parallel b_{160}(L), \quad \ell \geq 0$$

such that the length of M' (measured by bits) is divisible by 512. How do you determine ℓ?

4.9. SHA-1 is much simpler than SHA-512. In SHA-1, $\gamma = 160$ bits and $\Gamma = 2^{64} - 1$ bits. Let M be an input string with $|M| \leq \Gamma$. SHA-1 first pads M to produce a new string M' as described in Exercise 4.8, where each block is 512-bit long. There are 80 rounds of computations in the SHA-1 compression function, where basic operations are on 32-bit binary strings. Let X, Y, and Z be 32-bit binary strings. Let

$$F_t(X, Y, X) = \begin{cases} (X \wedge Y) \vee (\overline{X} \wedge Z), & \text{if } 0 \leq t < 20, \\ X \oplus Y \oplus Z, & \text{if } 20 \leq t < 40, \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), & \text{if } 40 \leq t < 60, \\ X \oplus Y \oplus Z, & \text{if } 60 \leq t < 80. \end{cases}$$

Define 80 constants $K_0, K_1, \cdots, K_{79}$ as follows:

$$K_t = \begin{cases} \text{5a827999}, & \text{if } 0 \leq t < 20, \\ \text{6ed9eba1}, & \text{if } 20 \leq t < 40, \\ \text{8f1bbcdc}, & \text{if } 40 \leq t < 60, \\ \text{ca62c1d6}, & \text{if } 60 \leq t < 80. \end{cases}$$

Let r_1, r_2, r_3, r_4, and r_5 be variables, each of which represents a 32-bit binary string (160 bits totally), where their initial values are, in hexadecimal,

$$r_1 = \textsf{67452301},$$

$$r_2 = \textsf{efcdab89},$$

$$r_3 = \textsf{98badcfe},$$

$$r_4 = \textsf{10325476},$$

$$r_5 = \textsf{c3d2e1f0}.$$

Let the initial vector **IV** of SHA-1 be the concatenation of the initial values of r_1, r_2, r_3, r_4, and r_5.

Recall that $|M'| = N \times 512$ bits, where N is a positive integer. Let $M' = M_1$ $M_2 \cdots M_N$, where each M_i is a 512-bit binary string. SHA-1 produces a hash value SHA-1$(M) = H_N$ using the following recurrence relation:

$$H_i = H_{i-1} \oplus_{32} F(M_i, H_{i-1}),$$

$$i = 1, 2, \cdots, N,$$

$$H_0 = \textbf{IV},$$

where $F(M_i, H_{i-1})$ is a compression function, defined as follows:

1. Let $M_i = W_0 W_1 \cdots W_{15}$, where each W_j is a 32-bit binary string.
2. For t from 16 to 79, let

$$W_t = [(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}] \lll 1.$$

3. For t from 0 to 79, let

$$T \leftarrow [(r_1 \lll 5) + F_t(r_2, r_3, r_4) + r_5 + W_t + K_t] \bmod 2^{32},$$

$$r_5 \leftarrow r_4,$$

$$r_4 \leftarrow r_3,$$

$$r_3 \leftarrow r_2 \lll 30,$$

$$r_2 \leftarrow r_1,$$

$$r_1 \leftarrow T.$$

  (a) Draw a flow diagram of SHA-1 computation, including how W_i is generated and the computation of the SHA-1 compression function.

  **(b) Explain why using the order of 2^k computations with $k < 80$ could allow you to find $M_0 \neq M_1$ such that SHA-1$(M_0) = $ SHA-1(M_1).

*4.10. The entries in WHIRLPOOL's S-Box (see Table 4.3) are calculated using finite-field operations defined on $GF(2^4)$ under an irreducible polynomial $r(x) = x^4 + x + 1$. In particular, let u be a hexadecimal digit. Let

$$E(u) = \begin{cases} \left(x^3 + x + 1\right)^u \bmod r(x), & \text{if } u \neq \textsf{f}, \\ 0, & \text{if } u = \textsf{f}. \end{cases}$$

Table 4.3 The E mini-box for constructing the S-Box of WHIRLPOOL

u	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
E(u)	1	b	9	c	d	6	f	3	e	8	7	4	a	2	5	0

Show that E is determined by Table 4.3.

*4.11. Let E^{-1} denote the inverse function of E defined in Exercise 4.9.2. Show that E^{-1} is determined by Table 4.4.

Table 4.4 The E^{-1} mini-box for constructing the S-Box of WHIRLPOOL

u	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$E^{-1}(u)$	f	0	d	7	b	e	5	a	9	2	c	1	3	4	8	6

4.12. Let R be a random permutation of $0, 1, \cdots ,$ f defined in Table 4.5.

Table 4.5 The R mini-box for constructing the S-Box of WHIRLPOOL

u	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
R(u)	7	c	b	d	e	4	9	f	6	3	8	a	2	5	1	0

The entry $s_{i,j}$ in WHIRLPOOL's S-Box is calculated using the following tweaked procedure:

$$i, j = 0, 1, \cdots , f,$$
$$y = E(i) \oplus E^{-1}(j),$$
$$z_1 = R(y) \oplus E(i),$$
$$z_2 = R(y) \oplus E^{-1}(j),$$
$$s_{i,j} = E(z_1)E^{-1}(z_2),$$

where i and j are four-bit binary strings.

For example, consider $s_{0,0}$. We have

$$y = E(0000) \oplus E^{-1}(0000) = 0001 \oplus 1111 = 1110.$$

We further have $R(1110) = 0001, z_1 = 0001 \oplus 0001 = 0000, z_2 = 0001 \oplus 1111 = 1110$. Thus, we have $ss_{0,0} = E(0000)E^{-1}(1110) = 18$.

Compute the fifth row of WHIRLPOOL's S-Box (see Table 4.1). That is, compute $s_{4,0}, s_{4,1}, \cdots , s_{4,f}$.

4.13. What is the value of rc_{10}?

4.14. Write a program to implement SHA-3 for $\ell = 6$.

4.15. Draw a flow diagram of HMAC computations.

4.16. Show that Equality 2.33 is true.

4.17. Show that Equality 2.34 is true.

4.18. Show that Equality 2.35 is true.

4.19. Find the value of k so that in a group of k randomly selected people, the probability that there are at least two persons who were born on the same date in the same month is greater than 3/4? For convenience, assume that there are no leap years.

4.20. Randomly select k students from the first grade, second grade, and the third grade students. How big must k be to ensure that the probability of at least two students among the selected having the same birthday (i.e., they were born on the same date, in the same month, and in the same year) is greater than 1/2? Assume that students in the same grade were born in the same year, and none of them was born in a leap year.

4.21. In Section 4.4.1, we have determined that if we randomly select $\sqrt{n}$ strings, then the probability that there are at least two strings $x \neq y$ such that $H(x) = H(y)$ is greater than 1/2, where H is a hash function and n is the number of different hash values.

Let x be fixed. Then select k strings randomly and independently. What is the probability that there is at least one string $y \neq x$ in the selected strings such that $H(y) = H(x)$? What should k be for the probability to be greater than 1/2?

4.22. Prove that when $n \geq 52$, Inequality 4.6 holds. That is,

$$1 + \sqrt{1 + 8\ln 2 \cdot n} < \sqrt{9\ln 2 \cdot n}.$$

4.23. Give a concrete application of the set intersection attack.

4.24. Let E be a symmetric-key encryption algorithm, where it takes a ℓ-bit data block and a ℓ-bit key as input. Let $M = M_1 M_2 \cdots M_N$, where each M_i is a ℓ-bit binary string (after appropriate padding if necessary). Define a hash function H as follows:

$$H_0 = \ell - \text{bit initial vector},$$
$$H_i = E_{H_{i-1}}(M_i),$$
$$i = 1, 2, \cdots, N,$$
$$H(M) = H_N.$$

Show that if Malice can obtain one pair $(M', H(M'))$, then she can find a message $M'' \neq M'$ such that $H(M'') = H(M')$ using the method of set intersection attack.

4.25. The UNIX operating system (as well as Linux) uses a symmetric-key encryption algorithm named crypt(3) to hash users passwords and store the hash values in a file. An early version of crypt(3) transforms a user password w to a 56-bit binary string k_w as a secret key for DES. It then randomly selects a 12-bit binary string s, referred to as *salt*. It modifies DES to obtain a new encryption algorithm DES[s], where s is the

salt value and DES[s] does everything the same as in DES except in the output of the expansion permutation *EP* (see Section 2.2.4 for a description of *EP*): If the ith bit in s is 1, swap the ith bit with the $(i + 24)$th bit in the output of *EP*. crypt(3) computes w's hash value $h_{s,w}$ as follows

$$C_0 = \text{DES}[s]_{k_w}(0^{64}),$$
$$C_i = \text{DES}[s]_{k_w}(C_{i-1}),$$
$$i = 1, 2, \cdots, 24,$$
$$h_{s,w} = C_{24}.$$

The operating system stores $h_{s,w}$ and its salt value s according to user names in a file (see Table 4.6)

Table 4.6 File structure for storing user passwords

User name	Salt	Hash value of user password
Alice	s	$h_{s,w}$
$\vdots$	$\vdots$	$\vdots$

After a user enters his user name u and user password w, the operating system computes a 56-bit secret key k_w of w. It searches for u's record in the password file and finds the salt value s associated with u. It then computes $h_{s,w}$ and compares it with the hash value stored in the record. The user is allowed to log on if and only if these two values are identical.

(a) Explain why salt values are needed.

(b) Under crypt(3), how long is the effective length of a user's password?

(c) Analyze the security strength of crypt(3).

*4.26. Early versions of crypt(3) did not support users selecting passwords with arbitrary length, which makes it vulnerable to dictionary attacks. To improve security, crypt(3) was later modified to compute hash values of user passwords using MD5, which allows users to select passwords with arbitrary length.

(a) Search the literature for a detailed description of the modified crypt(3) algorithm. Then write a short paper (about 4000 words) describing this algorithm and analyzing its security strength.

(b) MD5 has been shown to have a weakness (i.e., it does not satisfy the requirement of strong collision resistance). Will this mean that using MD5 in crypt(3) is no longer secure? Justify your answer.

*4.27. Microsoft Windows XP, unlike UNIX or Linux, stores user names and user passwords in the registry. Search the literature for a detailed description of how this is done and write a short paper (about 4000 words) describing this algorithm and analyzing its security strength.

4.28. Draw a flow diagram of the DSS signing algorithm and a flow diagram of the DSS signature verifying algorithm.

4.29. If Alice's random number k_A used to sign a document using DSS is stolen, what would happen?

4.30. In addition to securing online shopping, can you think of any other applications for which dual signatures can be used?

*4.31. Can you modify the eCash scheme presented in Section 4.7.2 so that it also satisfies the dividability requirement? Justify your answer.

4.32. Explain which requirements of electronic cash listed in Section 4.7.2 is met by eCash and which requirements is not met by eCash.

*4.33. Alice creates an electronic document and wants her boss Bob to sign it using RSA. However, Alice does not want Bob to see the document he is asked to sign. Bob, on the other hand, agrees to sign any legitimate document Alice presents to him. Devise a blind signature scheme to solve this problem.

4.34. Devise a double signature scheme using RSA to sign a document with two signatures. In particular, the two signatures are signed sequentially. Only after the second signatory verifies the first signature should he sign the document with the first signatory's signature. The document must be verifiable by the public that it indeed has two signatures.

4.35. The double signature scheme described in Exercise 4.34 is a special application of *multiple-key public-key cryptography* (*MKPKC*). An MKPKC encryption algorithm uses several keys, some of which are public keys, while the remaining keys are private. Generalize RSA from one public key and one private key to a multiple public-key RSA. Justify your answer. That is, prove the correctness of the multiple public-key RSA decryption algorithm.

*4.36. An *undeniable signature* is a signature signed on a document using signer's private key such that the signature cannot be verified without signer's permission. Moreover, the signer can prove forged signatures so that he cannot falsely deny a genuine signature of his.

 The first property allows the signer to restrict who may verify his signature; so that if an authorized user obtains a copy of the document, he will be unable to verify the signature.

 In 1989, David Chaum and Hans van Antwerpen devised a scheme for undeniable signatures. Do a literature search and write a paper of up to 4000 words to describe their scheme.

4.37. How does the Bitcoin network prevent a miner from adding fake blocks in the block chain? Justify your answer.

4.38. Does the number of Bitcoins have an upper bound? Justify your answer.

5

Network Security Protocols in Practice

Computer cryptography provides building blocks for constructing network security protocols. These building blocks include symmetric-key encryption algorithms, public-key encryption algorithms, key-generation and key-exchange algorithms, cryptographic hash functions, authentication algorithms, digital signatures, and public-key infrastructures. We call these building blocks *cryptographic algorithms*.
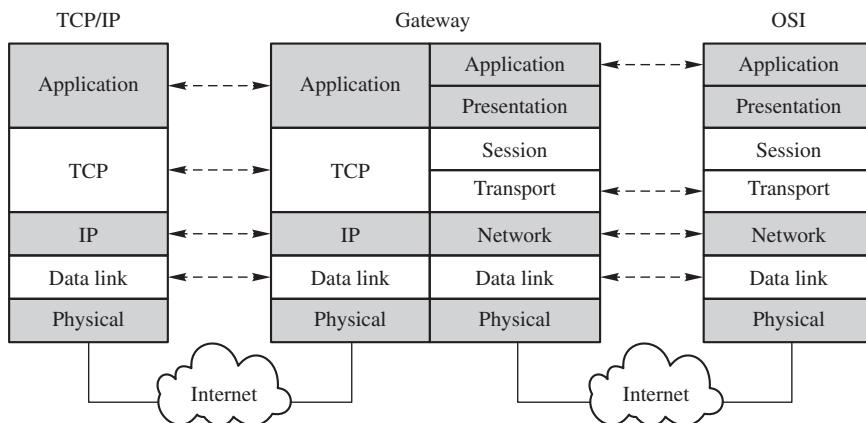
To protect network communications, one may deploy cryptographic algorithms at any layer in the network architecture. The use of cryptographic algorithms at different layers offers different degrees of protection. This technique of placing algorithms within the different network layers is the first issue discussed in this chapter.

We then introduce common network security protocols used in practice. These protocols include the X.509 public-key infrastructure (PKI), the IP security protocol at the network layer (IPsec), the Secure Sockets Layer protocol at the transport layer (SSL/TLS), and several application-layer security protocols, including Pretty Good Privacy (PGP), Secure/Multipurpose Internet Mail Extension (S/MIME), Kerberos, Secure Shell (SSH), and an electronic voting protocol.

5.1 Crypto Placements in Networks

TCP/IP is the dominant networking technology today. It is a five-layer architecture. These layers are, from top to bottom, the application layer, the transport layer (TCP), the network layer (IP), the data-link layer, and the physical layer. In addition to TCP/IP, there also are other networking technologies. For convenience, we use the OSI network model to represent non-TCP/IP network technologies. Different networks are interconnected using gateways. A gateway can be placed at any layer.

The OSI model is a seven-layer architecture. The OSI architecture is similar to the TCP/IP architecture, except that the OSI model specifies two additional layers between the application layer and the transport layer in the TCP/IP architecture. These two layers are the presentation layer and the session layer. Figure 5.1 shows the relation between the TCP/IP layers and the

Figure 5.1 Correspondence between layers of the TCP/IP architecture and the OSI model. Also shown are placements of cryptographic algorithms in network layers, where the dotted arrows indicate actual communications of cryptographic algorithms

OSI layers. The application layer in TCP/IP corresponds to the application layer and the presentation layer in OSI. The transport layer in TCP/IP corresponds to the session layer and the transport layer in OSI. The remaining three layers in the TCP/IP architecture are one-to-one correspondent to the remaining three layers in the OSI model.

The functionalities of OSI layers are briefly described as follows:
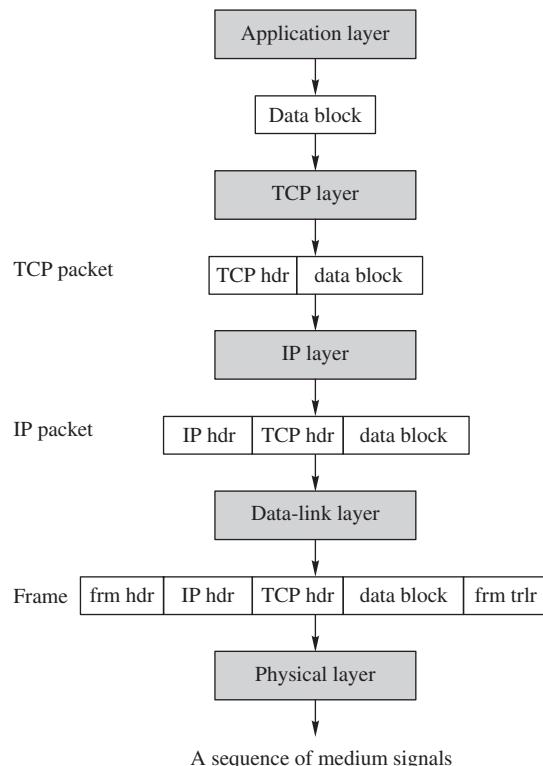
1. The application layer serves as an interface between applications and network programs. It supports application programs and end-user processing. Common application-layer programs include remote logins, file transfer, email, and Web browsing.
2. The presentation layer is responsible for dealing with data that is formed differently. This protocol layer allows application-layer programs residing on different sides of a communication channel with different platforms to understand each other's data formats regardless of how they are presented.
3. The session layer is responsible for creating, managing, and closing a communication connection.
4. The transport layer is responsible for providing reliable connections, such as packet sequencing, traffic control, and congestion control.
5. The network layer is responsible for routing device-independent data packets from the current hop to the next hop.
6. The data-link layer is responsible for encapsulating device-independent data packets into device-dependent data frames. It has two sublayers: logical link control and media access control.
7. The physical layer is responsible for transmitting device-dependent frames through some physical media.

Starting from the application layer, data generated from an application program is passed down layer-by-layer to the physical layer. Data from the previous layer is enclosed in a new envelope at the current layer, where the data from the previous layer is also just an envelope

containing the data from the layer before it. This is similar to enclosing a smaller envelope in a larger one. The envelope added at each layer contains sufficient information for handling the packet. Application-layer data are divided into blocks small enough to be encapsulated in an envelope at the next layer.

Application data blocks are "dressed up" in the TCP/IP architecture according to the following basic steps. At the sending side, an application data block is encapsulated in a TCP packet when it is passed down to the TCP layer. In other words, a TCP packet consists of a header and a payload, where the header corresponds to the TCP envelope and the payload is the application data block. Likewise, the TCP packet will be encapsulated in an IP packet when it is passed down to the IP layer. An IP packet consists of a header and a payload, which is the TCP packet passed down from the TCP layer. The IP packet will be encapsulated in a device-dependent frame (e.g., an Ethernet frame) when it is passed down to the data-link layer. A frame has a header, and it may also have a trailer. For example, in addition to having a header, an Ethernet frame also has a 32-bit cyclic redundancy check (CRC) trailer. When it is passed down to the physical layer, a frame will be transformed to a sequence of media signals for transmission. Figure 5.2 demonstrates this process.

At the destination side, the medium signals are converted by the physical layer into a frame, which is passed up to the data-link layer. The data-link layer passes the frame payload (i.e.,


Figure 5.2 Flow diagram of packet generation

the IP packet encapsulated in the frame) up to the IP layer. The IP layer passes the IP payload, namely, the TCP packet encapsulated in the IP packet, up to the TCP layer. The TCP layer passes the TCP payload, namely, the application data block, up to the application layer. When a packet arrives at a router, it only goes up to the IP layer, where certain fields in the IP header are modified (e.g., the value of TTL is decreased by 1). This modified packet is then passed back down layer-by-layer to the physical layer for further transmission.

Deploying cryptographic algorithms at different layers has different security effects. For convenience, we use the term *crypto placement* to mean deployment of cryptographic algorithms.

5.1.1 Crypto Placement at the Application Layer

Deploying cryptographic algorithms at the application layer provides end-to-end security protection. Data is encrypted or authenticated at this layer. The encrypted or authenticated data then goes through each layer below as if it were normal data. That is, it does not need to be decrypted or checked for signatures at any layer.

On the other hand, TCP headers and IP headers are not encrypted or authenticated because these headers are added within the lower layers, making it possible for an attacker to analyze traffic and modify header information. For example, Malice may change the destination IP address in the IP header to have the modified packet delivered to a different person.

5.1.2 Crypto Placement at the Transport Layer

Deploying cryptographic algorithms at the transport layer provides security protections for TCP packets. The payload of a TCP packet or the entire TCP packet itself (i.e., both header and payload) can be encrypted or authenticated at this layer.

Crypto placement at the transport layer does not affect the application data received from the application layer. Therefore, users do not need to modify any application programs.

The IP header encapsulating the encrypted or authentication TCP packet is not encrypted, making it possible for the attacker to analyze traffic using information from IP headers. If the TCP header is not encrypted, the attacker may further obtain additional information such as TCP sequencing numbers. This makes it possible for the attacker to figure out how TCP sequencing numbers might be generated. This information is needed if the attacker wants to hijack a TCP connection.

5.1.3 Crypto Placement at the Network Layer

Deploying cryptographic algorithms at the network layer provides link-to-link security protection. At this layer, the payload of the IP packet or the entire IP packet itself (i.e., both headers and payloads) can be encrypted or authenticated. Applying cryptographic algorithms on payloads does not affect the routing functionality, and it is referred to as the *transport mode* application. Applying cryptographic algorithms on the entire packet requires a network-layer gateway to route tunnel mode IP packets, which is equivalent to hiding the whole IP packets

inside the gateway, and it is referred to as the *tunnel mode* application. In particular, when a tunnel-mode IP packet arrives, the gateway first deciphers the encrypted IP packet (or verifies its signature). This allows the gateway to read the IP header. If the gateway is not in the destination edge network, the gateway re-encrypts (or re-authenticates) the whole IP packet, adds a plaintext IP header of its own, and routes the new IP packet to the next IP gateway. Thus, the IP packets are visible to the gateways at the end points of a tunnel but remain invisible inside the tunnel that may contain other routers.

Unlike the encryption that takes place within an application layer where applications need to incorporate cryptographic algorithms specifically, deploying cryptographic algorithms at the network layer does not require modifications to the existing application programs. Thus, implementing link-to-link security protection places no extra work on application programs.

A transport-mode IP packet leaves the original IP header in plaintext format, but its payload, that is, the TCP header and the TCP payload, is encrypted, making it possible for the attacker to analyze network traffic. However, the attacker will not be able to obtain TCP sequencing numbers or other information contained in the TCP header. A tunnel-mode IP packet might leave the IP header of the gateway in plaintext format, and so the attacker can only observe traffics between IP gateways, instead of between users. This can be further protected using nested tunnels, that is, a tunnel wrapping around another tunnel, to make traffic analysis more difficult.

5.1.4   *Crypto Placement at the Data-Link Layer*

Deploying cryptographic algorithms at the data-link layer provides security protections for frames. Payloads of the frames are encrypted or authenticated at this layer. Deploying cryptographic algorithms at the data-link layer also does not require modifications to the existing application programs. We note that frames travel only one link. Thus, traffic analysis on encrypted frames would not yield much information.

In this chapter, we focus on crypto placements at the network layer, at the transport layer, and at the application layer. We introduce crypto placements at the data-link layer in Chapter 6 in the context of wireless security.

5.1.5   *Implementations of Crypto Algorithms*

Cryptographic algorithms may be implemented on hardware, using Application Specific Integrated Circuit (ASIC) technologies, or in software. In general, it is common to implement cryptographic algorithms at the application layer using software and at the data-link layer using hardware. Implementations of cryptographic algorithms at other layers can be performed using software, or hardware, or both. Hardware implementations offer the best performance, but they are inflexible to change, hard to port to different platforms, and may cost more to develop. Software implementations, on the other hand, are flexible to change, easier to port, and may cost less to develop, but their performance will not match the performance of hardware implementations. Recent advances of programmable network processors (e.g., Intel IXP network processors), however, may offer both hardware performance and software flexibility.

5.2 Public-Key Infrastructure

To deploy cryptographic algorithms in network applications, we need a way to distribute secret keys using open networks. Public-key cryptography is the best way to distribute these secret keys. In order to use public-key cryptography, we need to build a public-key infrastructure (PKI) to support and manage public-key certificates and certificate authority (CA) networks. In particular, PKIs are set up to perform the following functions:

1. Determine the legitimacy of users before issuing public-key certificates to them.
2. Issue public-key certificates upon user requests.
3. Extend public-key certificates valid time upon user requests.
4. Revoke public-key certificates upon users' requests or when the corresponding private keys are compromised.
5. Store and manage public-key certificates.
6. Prevent digital signature signers from denying their signatures.
7. Support CA networks to allow different CAs to authenticate public-key certificates issued by other CAs.

5.2.1 X.509 Public-Key Infrastructure
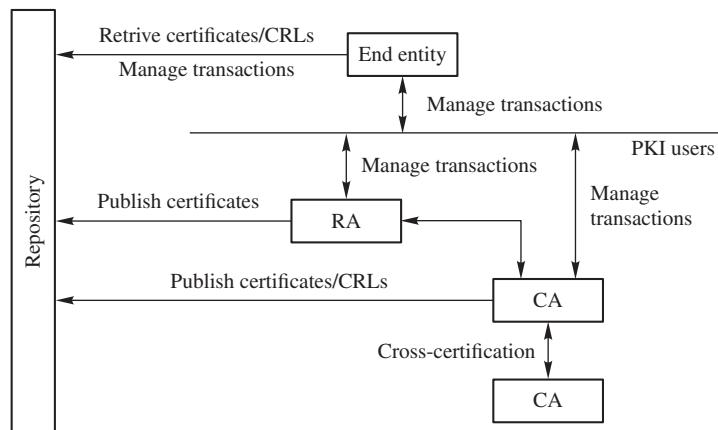
Recommended by the Internet Engineering Task Force (IETF), X.509 is a public-key infrastructure established by the Telecommunication Standardization Sector of the International Telecommunication Union (ITU) in 1988. It is also referred to as the ITU-T PKI standard, and we denote it by PKIX. PKIX consists of the following four basic components: *end entity*, *certificate authority (CA)*, *registration authority (RA)*, and *repository*. An entity means any user of public-key certificates or any device (e.g., servers and routers) that supports PKIX. These components have the following functionalities:

1. The CA is responsible for issuing and revoking public-key certificates.
2. The RA is responsible for verifying identities of owners of public-key certificates.
3. The Repository is responsible for storing and managing public-key certificates and certificate revocation lists (CRLs). A CRL is a list of certificates revoked by CA.

Figure 5.3 shows the architecture of PKIX.

Transaction management between the end entity, CA, RA, and repository includes the following items:

1. *Registration*: Users register with CA or RA before certificates are issued to them. Users may register their certificates directly or indirectly through RA.
2. *Initialization*: Users obtain initial information, including public keys of CAs and RAs, signature algorithms, and information.
3. *Certificate issuing and publication*: CA or RA issues and publishes certificates in the repository for users.
4. *Key recovery*: CA or RA provides necessary mechanisms for users to recover lost private keys.
5. *Key generation*: CA or RA periodically generates new key pairs for users.

Figure 5.3 PKIX architecture

6. *Certificate revocation*: Users inform CA or RA to revoke their certificates if they lose private keys, if they change names/addresses, or in case of any other events that may jeopardize the security of their private keys.
7. *Cross-certification*: Different CAs should be able to authenticate certificates issued by each other.

5.2.2   X.509 Certificate Formats

X.509 certificate formats have gone through three different versions. X.509 version 1 was first released in 1988. X.509 version 2 was not used widely. X.509 version 3 was released in 1996 and is the most common certificate format used today. An X.509 certificate consists of the following components:

1. *Version*: It indicates which version the certificate is using.
2. *Serial number*: It is a unique number assigned to the certificate within the same CA.
3. *Algorithm*: It lists the name of the hash function and the public-key encryption algorithm used to generate the signature for the certificate. For example, the name `sha1RSA` indicates that the signature of the certificate is generated by applying RSA on the hash value of the certificate produced by SHA-1.
4. *Issuer*: It gives the issuer's name.
5. *Validity period*: It gives a time interval when the certificate is valid.
6. *Subject*: It gives the certificate owner's name.
7. *Public key*: It gives the subject's public-key information and parameter information (if any) and what algorithm this key is to be used with.
8. *Extension*: It gives other information such as what the subject's key is used for. Only version 3 offers this component.
9. *Properties*: It gives the encrypted hash value of the certificate using CA's private key (i.e., the signature of the certificate) and other information.

Table 5.1 An X.509 certificate generated by Adobe Acrobat Pro, where c, email, ou, o, and cn are X.509 names, representing, respectively, certificate owner's country, email address, organization, employer, and name

Name	Value
Key usage	Sign transaction, Encrypt document
SHA1 digest of public key	C2139EE0B1CD6C22F485650EA5B1EB23D2882487
Public key	RSA (1024 bits)
	`30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01`
	`01 01 05 00 03 81 8D 00 30 81 89 02 81 81`
	`00 C5 DD 2D 97 2F 1F A5 4E 16 6A 32 FE 37`
	`77 44 4C 0C 2F 03 E0 02 05 64 AB C3 52 F0`
	`A9 5E 4F 32 1A E6 3E 77 83 C7 56 8F B8 A1`
	`FF 1F 15 F5 9C DA 7E DF C3 F3 92 80 A0 B7`
	`EB 2B 14 3E 6C 6D CA D2 4F 92 C1 7C 7F 43`
	`B4 F6 15 63 07 ED C0 7E 5A F7 4F 0E 13 75`
	`2C 9C 9E 59 FD DA 4F 71 F3 B0 35 0B EA F0`
	`60 D2 33 45 BD 5A DA DD 09 42 AF EB C4 40`
	`38 4A F0 DC 42 79 05 56 BC DE A5 CF 50 8D`
	` 8A C5 02 03 01 00 01`
Validity ends	2017/12/11 19:48:34 -04'00'
Validity starts	2012/12/11 19:48:34 -04'00'
Serial number	`47 D2 ED 7D 82 FF 40 21 08 F5`
Issuer	c = US, email = `wang@cs.uml.edu`
	ou = Department of Computer Science
	o = UMass Lowell, cn = Jie Wang
Subject	c = US, email = `wang@cs.uml.edu`
	ou = Department of Computer Science
	o = UMass Lowell, cn = Jie Wang
Signature algorithm	SHA1 RSA (1.2.840.113549.1.1.5)
Version	3

Table 5.1 shows an example of the components in an X.509 certificate generated by Adobe Acrobat X Pro, which can be found in option of Details under the Adobe Acrobat Pro Certificate Viewer.

Suppose that Alice wants to send a master key K_{AB} to Bob and prove to Bob that it indeed comes from Alice without being modified during transmission. Alice first obtains from PKIX Bob's certificate $\text{CA}\langle K_B^u \rangle$, verifies CA's signature, and extracts Bob's public key K_B^u from the certificate. Alice then sends the following message to Bob:

$$D_{K_A^r}(M) \parallel \text{CA}\langle K_A^u \rangle \parallel M, \tag{5.1}$$

where

$$M = t_A \parallel r_A \parallel ID_B \parallel E_{K_B^u}(K_{AB}),$$

E is a public-key encryption algorithm agreed on by both Alice and Bob with D being its decryption algorithm, (K_A^u, K_A^r) is Alice's public-private key pair, t_A is a time stamp, r_A is a nonce, ID_B is Bob's identity, and $\text{CA}\langle K_A^u \rangle$ is Alice's certificate.

After receiving Message 5.1, Bob first verifies the signature of Alice's certificate using CA's public key. If confirmed, Bob retrieves Alice's public key K_A^u from the certificate and uses K_A^u to encrypt $D_{K_A^r}(M)$ to get M. Bob then extracts from M the time stamp, the nonce, ID_B, and $E_{K_B^u}(K_{AB})$. If ID_B is correct, and the time stamp and the nonce are valid, then Bob uses his own private key K_r^B to decrypt $E(K_B^u, K_{AB})$ to get K_{AB}.

5.3   IPsec: A Security Protocol at the Network Layer

IPsec is a major security protocol at the network layer. Some authors write it as IPSec. It is written as IPsec in RFC documents. IPsec provides a potent platform for constructing *virtual private networks* (VPN). VPNs are private networks overlayed on public networks.

As mentioned in Section 5.1, the purpose of deploying cryptographic algorithms at the network layer is to encrypt or authenticate IP packets (either just the payloads or the whole packets). IPsec specifies how this is to be done. IPsec also specifies how to exchange keys. Thus, IPsec consists of authentication protocols, encryption protocols, and key exchange protocols. They are referred to, respectively, as *authentication header* (AH), *encapsulating security payload* (ESP), and *Internet key exchange* (IKE).

1. AH is an authentication format. It is used to authenticate the origin of the IP packet and ensure its integrity. In addition, AH uses the sliding window technique to detect message replays. Values in certain fields in the IP header (e.g., TTL) are updated at each hop during transmission, but values in most of the fields remain unchanged. For each IP packet to be authenticated, AH authenticates its payload and the fields with unchanged values in its header.
2. ESP is an encryption format. It is used to encrypt IP packets, either just their payloads or the whole packets. It can also be used to authenticate IP packets.
3. IKE is a key exchange format. It is used to establish secret keys for the sender and the receiver.

IPsec supports a number of encryption algorithms for users to choose from. When Alice wants to communicate with Bob using IPsec, Alice must first select a set of encryption algorithms and parameters and then inform Bob about her selection. Bob may accept Alice's selection or negotiate with Alice for a different set of algorithms and parameters. Once the algorithms and parameters are selected, IPsec establishes a *security association* (SA) between Alice and Bob for the rest of the session.

5.3.1   Security Association

A security association provides the following information:

1. *Security parameters index* (SPI): It is a 32-bit binary string used to identify a particular set of algorithms and parameters, as well as a particular communication session. SPI is included in AH and ESP to ensure that both sides will use the same algorithms and parameters.
2. *IP destination address*: It specifies which host the underlying SA is established for.
3. *Security protocol identifier*: It specifies whether the underlying SA is established for AH or ESP. IPsec disallows AH and ESP to use the same SA simultaneously.

A SA has a particular lifetime. SAs for different communication sessions that use the same set of algorithms and parameters, with the same IP destination address and security protocol identifier, are different SAs that are specified with different SPIs.

SAs can be established dynamically by IKE or statically by the IPsec manager of the host computer.

When both ESP and AH are applied to an IP packet, IPsec applies ESP before authentication. That is, AH is in front of the ESP header. The reason for this is that deciphering a packet would take more time than verifying a signature of it. If authentication fails in the first place, then there is no need to decipher the packet in the second place. This means that the authentication SA comes in front of the ESP SA. A sequence of SAs is referred to as an *SA bundle*.

IPsec has several built-in mechanisms to facilitate the use of SAs. They are *security association database* (SAD), *security policy database* (SPD), and *SA selectors* (SAS).

5.3.1.1 Security Association Database

In order to facilitate searching (after an SA relation is established between users), IPsec stores the SA information in the SAD at a user's local machine. Therefore, including SPI in the IPsec packet header allows IPsec to look for the SA information within the SAD to process the packets.

5.3.1.2 Security Policy Database

IPsec is placed at the network layer, and so it needs to handle TCP packets from different users. Not every packet needs encryption or authentication. For IPsec on the sending host to know what to do when a TCP packet is passed down from the transport layer, the IPsec manager at the host computer must create and maintain a list of rules, which is referred to as the security policy. To facilitate searching, a security policy is stored in the SPD at the host computer. On the basis of the information contained in the TCP header, IPsec finds the corresponding security policy in the SPD. On the basis of this security policy, IPsec will encrypt the packet, authenticate the packet, or do nothing.

5.3.1.3 SA Selectors

IPsec allows users to assign a set of rules to an SA that determines which packets the SA is applied to. Such a set of rules is referred to as an SA selector. For example, one may give a certain SA a selector so that the SA only handles packets whose destination IP addresses fall into a certain range.

SAs are directional. Some SAs can handle outgoing packets, some can only be used to handle incoming packets, and others can handle both.

5.3.2 Application Modes and Security Associations

IPsec supports the transport-mode and the tunnel-mode applications of cryptographic algorithms.

Establishing SAs in transport mode is straightforward. For each outbound TCP packet passed down to the network layer at the sending host, IPsec checks the security policy stored in the local SPD. If the TCP packet is to be encrypted, IPsec encrypts it, adds an ESP header in front of the encrypted TCP packet, and specifies an SA. Likewise, if the TCP packet is to be authenticated, IPsec signs it with a digital signature, adds an authentication header in front of it, and specifies an SA. A normal IP header will then be added to the resulting packet for transmission. IPsec at the receiving host finds the SA in its SAD according to the SPI in the ESP header or in the authentication header and processes the packet accordingly.

Establishing SAs in tunnel mode is more involved. It depends on how many IPsec gateways there are on the path from the sending host to the receiving host. Between adjacent IPsec gateways, there could be other routers.

5.3.2.1   Single Tunnel
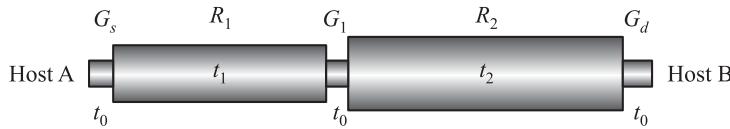
The simplest tunnel-mode SA is a one-layer tunnel. For each outbound IP packet that asks for tunnel-mode encryption (the case for tunnel-mode authentication is similar), the IPsec gateway G_s at the sending side encrypts the entire IP packet, adds an IP header of its own, and forwards the resulting packet to the next IPsec gateway G_1. In order for it to forward the resulting packet, G_s needs to establish a security association $SA_{s,1}$ with G_1 to ensure that G_1 knows how to decipher the IP packet it receives. G_1 checks the IP header of the deciphered IP packet, obtains its destination IP address, and finds the next IPsec gateway G_2 to forward this packet to. G_1 then encrypts the entire IP packet, adds an IP header of its own, and forwards the resulting packet to G_2. To carry out this forwarding action, G_1 needs to establish a security association $SA_{1,2}$ with G_2. This process continues until the destination IPsec gateway G_d is reached.

5.3.2.2   Nested Tunnels

In the above-mentioned example, the original IP packet can be read by each of the IPsec gateways on the path. To disallow reading of the original IP packets by certain IPsec gateways, IPsec can wrap another tunnel around the tunnel. For example, assume that G_s, G_1, and G_d are three IPsec gateways between host A and host B, where G_s is the sending-side gateway, G_1 is the next gateway of G_s, and G_d, the destination gateway, is the next gateway of G_1. Suppose that G_s supports encryption algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, G_1 supports encryption algorithms $\mathcal{A}_2$ and $\mathcal{A}_3$, and G_d supports encryption algorithms $\mathcal{A}_1$ and $\mathcal{A}_3$, where $\mathcal{A}_1$ is the weakest algorithm. Suppose that host A needs to send an IP packet P to host B using the strongest possible encryption algorithms during transmission but does not want G_1 to read P, where P's header has host A's IP address as the source and host B's IP address as the destination. This objective can be achieved using nested tunnels as follows (see Figure 5.4):

1. G_s establishes a security association $SA_{s,d}$ with G_d, specifying that $\mathcal{A}_1$ is the chosen encryption algorithm. G_s encrypts P using $\mathcal{A}_1$ to get an ESP packet denoted by $\mathcal{A}_1(P)$. This creates a tunnel t_0 between G_s and G_d. G_s adds an IP header $\text{IPh}_{s,d}$ in front of $\mathcal{A}_1(P)$, with G_s's IP address as the source and G_d's IP address as the destination. Let P' denote this new packet. That is,

$$P' = \text{IPh}_{s,d} \parallel \mathcal{A}_1(P).$$

Figure 5.4 A demonstration of multiple layers of nested tunnels, where R_1 denotes other routers on the path from IPsec gateway G_s to IPsec gateway G_1, and R_2 denotes other routers on the path from IPsec gateway G_1 to IPsec gateway G_d

2. G_s establishes a security association $SA_{s,1}$ with G_1, specifying that $\mathcal{A}_2$ is the chosen encryption algorithm. G_s encrypts P' using $\mathcal{A}_2$ to get an ESP packet denoted by $\mathcal{A}_2(P')$. This creates a tunnel t_1 between G_s and G_d that wraps around tunnel t_0. G_s adds an IP header $\mathrm{IPh}_{s,1}$ in front of $\mathcal{A}_2(P')$, with G_s's IP address as the source and G_1's IP address as the destination. Let P_1 denote this new packet, that is,

$$P_1 = \mathrm{IPh}_{s,1} \parallel \mathcal{A}_2(P').$$

G_s forwards P_1. (Note that P_1 may go through other routers between G_s and G_1.)
3. On receiving P_1, G_1 first uses $\mathcal{A}_2$, obtained from $SA_{s,1}$ through the SPI contained in the ESP header of $\mathcal{A}_2(P')$, to decipher P_1 to get P'. Realizing from $\mathrm{IPh}_{s,d}$ that the payload of P', that is, $\mathcal{A}_1(P)$, is to be forwarded to G_d, G_1 establishes a security association $SA_{1,d}$ with G_d, specifying that $\mathcal{A}_3$ is the chosen encryption algorithm. G_1 encrypts P' using $\mathcal{A}_3$ to get an ESP packet denoted by $\mathcal{A}_3(P')$. This creates a tunnel t_2 between G_1 and G_d that wraps around tunnel t_0. G_1 adds an IP header $\mathrm{IPh}_{1,d}$ in front of $\mathcal{A}_3(P')$, with G_1's IP address as the source and G_d's IP address as the destination. Let P_2 denote this new packet, that is,

$$P_2 = \mathrm{IPh}_{1,d} \parallel \mathcal{A}_3(P').$$
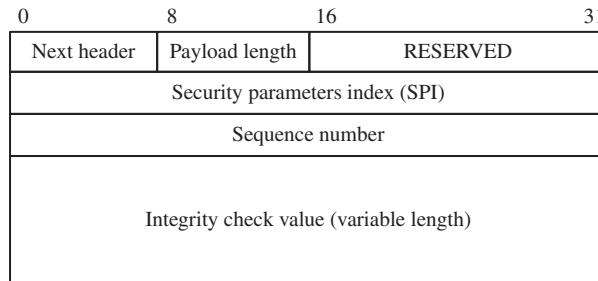
G_1 forwards P_2 to G_d. (Note that P_2 may go through other routers between G_1 and G_d.)
4. On receiving P_2, G_d first uses $\mathcal{A}_3$, obtained from $SA_{1,d}$ through the SPI contained in the ESP header of $\mathcal{A}_3(P')$, to decipher P_2 to get P'. Realizing from $\mathrm{IPh}_{s,d}$ that G_d is the final destination for the payload of P', G_d uses $\mathcal{A}_1$, obtained from $SA_{s,d}$ through the SPI contained in the ESP header of $\mathcal{A}_1(P)$, to decipher the encrypted $\mathcal{A}_1(P)$ to get P, and forwards P to host B.

5.3.3   AH Format

Figure 5.5 shows IPsec's authentication header format, where each field is defined as follows:

The field of "next header" is used to indicate the type of header immediately after the authen-
   tication header. For example, if an ESP header is immediately after the AH header, then this
   field is used to indicate the type of ESP.
The field of "payload length" is used to specify the number of words in the field of "integrity
   check value" (ICV) plus 1. For example, if the ICV is a 96-bit HMAC code, then the value in
   the "payload length" field equals $\lceil 96/32 \rceil + 1 = 4$, where $\lceil x \rceil$ denotes the smallest integer
   greater than or equal to x. Equivalently, the payload length is equal to the number of words
   contained in the header minus 2.

0	8	16	31

Next header	Payload length	RESERVED

Security parameters index (SPI)

Sequence number

Integrity check value (variable length)

Figure 5.5 Authentication header format

The field of "RESERVED" is reserved for future applications, which is occupied with 0's at the current version.

The field of SPI has already been explained in Section 5.3.1.

The field of "sequence number" is used to identify an authentication header, where the sequence number is a 32-bit counter. It is used to foil message replays.

In particular, the sending host sets the sequence number to 0 when it establishes an SA with the receiving host. The number is increased by 1 each time the SA is used until it is equal to $2^{32} - 1$, at which time, the SA must be terminated. A new SA must be established if there still are packets to be sent in the same session.

5.3.3.1 Sliding Window

To resist message replay attacks, IPsec at the receiving host uses a sliding window to determine which packets should be processed and which packets should be dropped. A sliding window is a buffer that can hold w sequence numbers; the default value of w is 64. IPsec at the destination host sets up a sliding window $SW[1, w]$ for each SA it creates. Initially, $SW[i]$ is unmarked for all i from 1 to w.

When the first packet associated with the SA arrives, the right end of the window $SW[w]$ is marked, representing the highest sequence number received so far. Let n represent this number.

After receiving a packet with sequence number i, the destination IPsec finds the SA window the packet belongs to and does the following:

1. If $n - w + 1 \leq i \leq n$, that is, if i is within the window, check if $SW[i + w - n]$ is marked. If it is marked, the packet is a replay; drop the packet. If it is not marked, check the signature of the packet. If the signature is valid, mark $SW[i + w - n]$ and process the packet; otherwise, drop the packet.
2. If $i \leq n - w$, the packet is old; drop the packet.
3. If $i > n$, check the signature of the packet. That is, unmark everything that is shifted out of the window. If the signature is valid, shift everything in the window to the left $i - n$ times, set $n \leftarrow i$, mark $SW[w]$, and unmark every $SW[j]$ with $w - n - i < j \leq w - 1$. If the signature is not valid, drop the packet.

5.3.3.2 Integrity Check Value

The field of "integrity check value (ICV)" holds a hash value of the data to be authenticated. IPsec at the receiving host decrypts the authenticated data, computes the hash value of the decrypted data, and compares it against the ICV to verify its signature. In the transfer mode, the data to be authenticated is the IP payload, and the authentication header is placed between the IP header and the TCP header. In the tunnel mode, it is the entire IP packet, except the fields in the IP header whose values (such as TTL and checksum) are updated dynamically during the transmission. The authentication header is placed in front of the entire IP packet. ICV may be a prefix of the value of a standard hash function. For example, under HMAC-SHA-1-96, ICV is the 96-bit prefix of the hash value produced by HMAC-SHA-1.

5.3.4   ESP Format

Figure 5.6 shows the ESP format.

The SPI field has the same meaning as the SPI field in the authentication header (see Section 5.3.3).
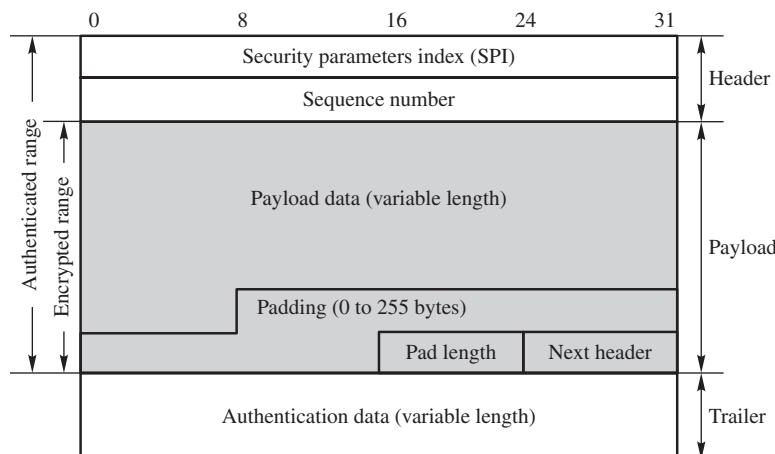
The "sequence number" field has the same meaning as the "sequence number" field found in the authentication header.

Within the "payload data" field, data is placed to be encrypted. Under the transport mode, the encrypted data is the IP payload, that is, the TCP packet. Under the tunnel mode, the encrypted data is the entire IP packet.

The "padding" field is used to pad the encrypted data to the desirable length according to the underlying encryption algorithms.

The "pad length" field is used to indicate how many bytes are included in the "padding" field.

The "next header" field points to the first header succeeding the ESP header.

Figure 5.6   ESP format

The length of the binary string

$$\text{payload data} \parallel \text{padding} \parallel \text{pad length} \parallel \text{next header}$$

must be divisible by 32. This binary string is to be encrypted, and we call the encrypted string the *encrypted component* of the ESP packet.

The "authentication data" field stores the ICV value of the following binary string:

$$\text{SPI} \parallel \text{sequence number} \parallel \text{payload data} \parallel \text{padding} \parallel \text{pad length} \parallel \text{next header}.$$

Its length is divisible by 32. It is used to check the integrity of the encrypted component of the ESP packet.

Thus, an ESP packet has a header, a payload, and a trailer. The header consists of SPI and the sequence number; the payload consists of the payload data, padding, pad length, and next header; and the trailer consists of the authentication data.

5.3.5   Secret Key Determination and Distribution

To use encryption algorithms to create ESP packets and to use HMAC authentication algorithms to authenticate packets, the sending host and the receiving host must first agree on using the same secret keys. This involves key determination and key distribution. Secret keys are set up automatically using key exchange protocols, although they can also be set up manually by system administrators. IPsec uses *Oakley key determination protocol* (KDP) and *Internet security association and key management protocol* (ISAKMP). Oakley KDP is Diffie–Hellman key exchange with authentication and several other security measures. However, it does not specify formats. ISAKMP, on the other hand, specifies key exchange formats, but it does not specify key exchange algorithms.

5.3.5.1   Oakley KDP

We have shown in Section 3.3.2 that the Diffie–Hellman key exchange scheme is vulnerable to the man-in-the-middle attack. However, this attack can be prevented if all parties involved in a key exchange can authenticate each other. Oakley KDP uses authentication methods to combat man-in-the-middle attacks.

In addition to the man-in-the-middle attack, the Diffie–Hellman key exchange scheme is also vulnerable to the *clogging attack*. The clogging attack is a form of denial-of-service attacks. It forces users to engage in a large number of expensive operations with the purpose of crashing their computer systems. In the context of Diffie–Hellman key exchange, the attacker sends to the victim a large number of public keys Y_i in crafted IP packets, where the source IP addresses contained in the crafted packets are IP addresses of unreachable hosts, forcing the victim's computer to compute secret keys $K_i = Y_i^X \bmod p$. Because modular exponentiations are expensive computations, computing a large number of modular exponentiations at the same time would use up the CPU cycles of the computer.

Oakley KDP uses *cookie exchange* to resist clogging attacks. In particular, suppose that Alice receives a message from Bob, which may contain Bob's Diffie–Hellman public-key Y_B,

requesting to use Diffie–Hellman key exchange to determine a secret key. In this example, Bob is the initiator and Alice the responder. Instead of carrying out the exponentiation $Y_B^{X_A}$ mod p right away, Alice generates a (pseudo) random number, called a *cookie*, sends it to Bob, and waits for Bob's acknowledgement. Only after she receives the acknowledgement will Alice carry out the exponentiation operation. Because in a crafted packet the host on the source IP address used in a clogging attack is not reachable, the victim's computer will not receive any acknowledgement of the cookie it sent, and so it will not execute the exponentiation operations it was asked to perform. To save time, the initiator will send a cookie along with his initial request. A cookie can be extended to include extra information. Thus, a cookie is a string that contains a random number as a substring, and the rest of the string represents other information.

Cookie exchange, Diffie–Hellman key exchange, and authentication are the three major components in Oakley KDP. In addition to these three components, Oakley KDP also uses nonce to thwart message-replay attacks.

To describe how Oakley KDP users exchange information, we define the following notations.

1. CKY_I denotes the initiator's cookie, and CKY_R denotes the receiver's cookie.
2. OK_KEYX denotes that the underlying message is for key exchange.
3. NIDP denotes that the succeeding part in the message is not encrypted, while IDP denotes that the succeeding part in the message is encrypted.
4. GRP denotes Diffie–Hellman parameters p and a, where p is a prime number and a is a primitive root modulo p. Note that Oakley KDP provides default values of p and a, but it also allows the initiator and the responder to negotiate a new set of parameters.
5. g^x and g^y denotes, respectively, a^x mod p and a^y mod p, where p and a are specified in GRP.
6. EHAO denotes the list of encryption algorithms, hash functions, and authentication algorithms supported by the initiator. EHAO is provided to the responder.
7. EHAS denotes the encryption algorithm, hash function, and the authentication algorithm selected by the responder from EHAO.
8. ID_I and ID_R denote, respectively, the initiator's name and the responder's name.
9. N_I and N_R denote, respectively, the initiator's nonce and the responder's nonce.
10. $S_{K_I^r}(X)$ and $S_{K_R^r}(X)$ denote, respectively, the initiator's signature of X and the responder's signature of X.

The following are the basic interactions between the initiator and the responder using Oakley KDP:
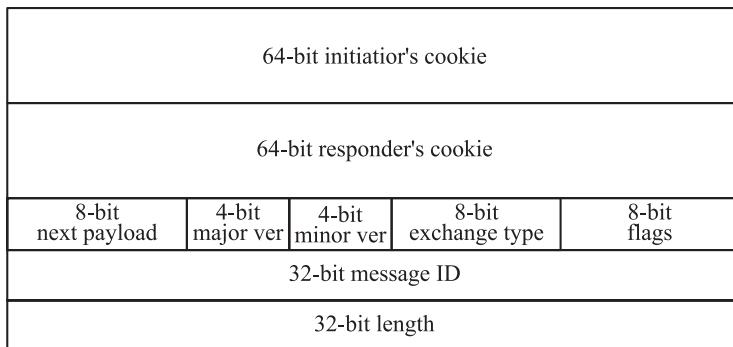
$I \rightarrow R:$ CKY_I, OK_KEYX, GRP, g^x, EHAO, NIDP, ID_I, ID_R, N_I,
           $S_{K_I^r}(ID_I \parallel ID_R \parallel N_I \parallel GRP \parallel g^x \parallel EHAO)$.

$R \rightarrow I:$ CKY_R, CKY_I, OK_KEYX, GRP, g^y, EHAS, NIDP, ID_R, ID_I, N_R, N_I,
           $S_{K_R^r}(ID_R \parallel ID_I \parallel N_R \parallel N_I \parallel GRP \parallel g^y \parallel g^x \parallel EHAS)$.

$I \rightarrow R:$ CKY_I, CKY_R, OK_KEYX, GRP, g^x, EHAS, NIDP, ID_I, ID_R, N_I, N_R,
           $S_{K_I^r}(ID_I \parallel ID_R \parallel N_I \parallel N_R \parallel GRP \parallel g^x \parallel g^y \parallel EHAS)$.

| 64-bit initiatior's cookie				
64-bit responder's cookie				
8-bit next payload	4-bit major ver	4-bit minor ver	8-bit exchange type	8-bit flags
32-bit message ID				
32-bit length				

Figure 5.7 ISAKMP header

5.3.5.2 ISAKMP Formats and Exchanges

ISAKMP specifies packet formats used for key exchange and other types of information exchange. An ISAKMP packet consists of a header and a payload. ISAKMP supports several types of payloads.

ISAKMP Header Format
Figure 5.7 shows the ISAKMP header format.

The 64-bit "cookie" field (of the initiator's or the responder's) contains, in addition to a random number, information to establish, notify, or delete a security association.

The 8-bit "next payload" field indicates the type of the first payload in the message.

The 4-bit "major version" field indicates the major version of ISAKMP being used. Likewise, the 4-bit "minor version" field indicates the minor version of ISAKMP being used.

The 8-bit "exchange type" field indicates the type of exchange.

The 8-bit "flags" field is used to specify options.

The 32-bit "message ID" field is a unique identifier of the underlying message.

The 32-bit "length" field specifies the number of bytes in the entire packet (i.e., header and all payloads).

ISAKMP Payload Types
ISAKMP specifies a number of payload types. They are *SA*, *proposal*, *transform*, *key-exchange*, *identification*, *certificate-request*, *certificate*, *hash*, *signature*, *nonce*, *notification*, and *delete* payloads.

1. The SA payload is used to establish a security association.
2. The proposal payload is used to negotiate an SA.
3. The transform payload specifies encryption and authentication algorithms.
4. The key-exchange payload specifies a key-exchange algorithm.
5. The identification payload carries information for identifying peers.
6. The certificate-request payload is used to request a public-key certificate.

  7. The certificate payload contains a public-key certificate.
  8. The hash payload contains the hash value of a hash function.
  9. The signature payload contains the output of a digital signature function.
 10. The nonce payload contains a nonce.
 11. The notification payload notifies the status of the other types of payloads (e.g., "invalid signature" is a notification).
 12. The delete payload is used to notify the receiver that the sender has deleted an SA or several SAs.

The payload in an ISAKMP packet may be one type of payload or a sequence of payloads of different types. For more information about these payload types, the reader is referred to RFC 2408 and other relevant RFC documents.

Each type of payload begins with a payload header of the same form (see Figure 5.8).

The 8-bit "next payload" field specifies the type of the succeeding payload. It is equal to 0 if it is the last payload.

The 16-bit "payload length" field specifies the number of bytes in the current payload and its payload header (i.e., not the payloads before or after it).

A Sample ISAKMP Exchange
The following is an ISAKMP payload exchange example:

1. I → R: SA, proposal, transfer, nonce
2. R → I: SA, proposal, transfer, nonce
3. I → R: key-exchange, identification, signature
4. R → I: key-exchange, identification, signature

In this example of payload exchange, the initiator first sends an SA payload, a proposal payload, a transfer payload, and a nonce payload to the responder for the purpose of setting up a security association. The responder selects an encryption algorithm and an authentication algorithm from the list provided by the initiator (i.e., the algorithms contained in the initiator's transfer payload). The responder then sends an SA payload, a proposal payload, a transfer payload, and a nonce payload to the initiator for the purpose of completing a security association. The initiator then sends a key-exchange payload, an identification payload (with the initiator's identity), and a signature payload (for authentication) to the responder for the purpose of determining a secret key. The responder sends a key-exchange payload, an identification payload (with the responder's identity), and a signature payload (for authentication) to the initiator to complete the key exchange procedure.

This example is the basic type of exchange. There are other types of exchange described in relevant RFC documents.

8-bit next payload	8-bit RESERVED	16-bit payload length

Figure 5.8   ISAKMP payload header

5.3.5.3 Internet Key Exchange

The IKE is heavily influenced by Oakley KDP. It has two distinct versions called version 1 and version 2. We only describe version 1, which is documented in RFC 2409.

The IKE protocol consists of two distinct phases. Phase one is responsible for authenticating and establishing session keys. Phase two is responsible for setting up SAs.

Phase one consists of two different authentication methods called the Main Mode and the Aggressive Mode. The Main Mode method must be implemented by all IKE software. Phase one is designed on top of the ISAKMP specification. The sample ISAKMP exchange described in the previous section is exactly phase one of IKE version 1. The key exchange used by IKE is Diffie–Hellman.

Phase two, sometimes called Quick Mode, is a three-message protocol used to establish an SA given the association created in Phase one. In particular, the SA established in Phase one is used to protect Quick Mode communications. The messages exchanged are all encrypted using the SA established in Phase one. The exchange of messages is as follows:
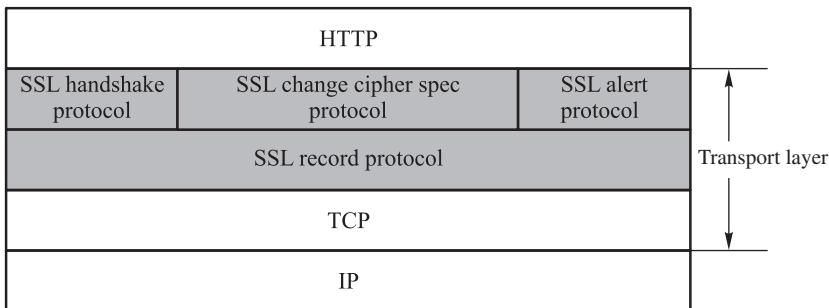
1. $I \rightarrow R$: Propose cryptographic parameters for the SA. This message contains the SPI for I, a proposal, and a nonce.
2. $R \rightarrow I$: Acceptance of the cryptographic proposal for the SA. This message contains the SPI for R, acceptance of the cryptographic parameters, and a nonce.
3. $I \rightarrow R$: Acknowledgment by I that R has accepted the cryptographic parameters.

5.4 SSL/TLS: Security Protocols at the Transport Layer

The SSL and the TLS are common transport-layer security protocols used in practice. Designed and developed by Netscape in 1994, SSL is used to protect World-Wide-Web applications and electronic transactions. The World Wide Web is a client-server application program. Thus, placing cryptographic algorithms at the transport layer (i.e., just below the application layer) to protect Web applications is a reasonable choice. TLS is a revised version of SSL version 3, which was published in 1999 as the transport-layer security standard by The Internet Engineering Task Force (IETF). There are only minor differences between TLS and SSLv3. This section describes SSL.

SSL consists of two components. The first component is referred to as the *record protocol*, which is placed on top of transport-layer protocols. The second component consists of the *handshake protocol*, the *change-cipher-spec protocol*, and the *alert protocol*. The second component is placed between application-layer protocols (such as HTTP) and the record protocol. Figure 5.9 shows how the SSL protocol structure exists between the application-layer protocol (in this case, HTTP) and the transport-layer protocol within the TCP/IP protocol stack. In particular, HTTPS specifies the HTTP protocol over SSL.

The handshake protocol establishes cryptographic algorithms, a compression algorithm, and parameters to be used by both sides during the encrypted exchange. After this, the record protocol takes over the communications. In particular, it is responsible for dividing a message into blocks, compressing each block, authenticating them, encrypting them, adding a record header to each block, and then transmitting the resulting blocks. The change-cipher-spec protocol allows communicating parties to change algorithms or parameters during a communication session. The alert protocol is a management protocol; it notifies communicating parties when problems occur.

HTTP		
SSL handshake protocol	SSL change cipher spec protocol	SSL alert protocol
SSL record protocol		
TCP		
IP		

Figure 5.9   SSL structure

Data can often be compressed to reduce size without losing information. ZIP is a widely used data compression algorithm in network communications. Invented by Phil Katz in the mid-1980s, ZIP is based on an universal algorithm for sequential data compression devised by Jacob Ziv and Abraham Lempel in 1977. This universal algorithm is often referred to as the LZ77 compression algorithm. Published as open format in 1989, ZIP has since been used in various popular data compression products, including PKZIP, WinZip, WinRAR, and gzip. See Appendix C for a detailed description of data compression using ZIP.

5.4.1   SSL Handshake Protocol

The handshake protocol is a complicated protocol. It allows the client and the server to negotiate and select a set of cryptographic algorithms and to exchange keys. It also allows the client and the server to authenticate each other. Because of the complexity of the protocol, we use an example of online shopping to describe the handshake protocol. The client program and the server program exchange information in four phases.

For convenience, we use "client" to denote "the client program" or "the user of the client program". Likewise, we use "server" to denote "the server program". In SSL applications of the World Wide Web, the client program is the Web browser and the server program a Web server.

5.4.1.1   Phase 1: Select Cryptographic Algorithms

The client initiates a conversation with a `client_hello` message to the server, and the server responds with a `server_hello` message to the client. The `client_hello` message consists of the following information:

1. *Version number v_c*: It is the highest SSL version installed at the client-side computer (e.g., $v_c = 3$.)
2. *Pseudorandom string r_c*: It is a 32-byte string consisting of a 4-byte time stamp and a 28-byte nonce generated by a pseudorandom number generator at the client-side computer. This string is used to resist message-replay attacks.
3. *Session ID S_c*: The value of S_c may be any number. If $S_c = 0$, it means that the client wants to start a new SSL connection using a new session. An SSL connection is determined by

the cryptographic algorithms, parameters, and hash functions agreed on by the client and the server. If $S_c \neq 0$, it means that the client wants to start a new SSL connection using the current session, or update the parameters of the current SSL connection.

4. *Cipher suite*: It is a list of public-key encryption algorithms, symmetric-key encryption algorithms, and hash functions supported by the client-side system, listed in decreasing order of preference. For example, the client may supply the following cipher suite:

$$\langle \text{ RSA, ECC, Diffie–Hellman, Elgamal};$$
$$\text{AES-128, 3DES/3, MARS, RC6, Serpent, Twofish};$$
$$\text{SHA-512, Whirlpool, SHA-384, SHA-256, SHA-1 } \rangle.$$

Each item in the list also comes with a description of how to use it.

5. *Compression method*: It is a list of compression algorithms supported by the client-side system, listed in decreasing order of preference. For example, the client may supply the following list of compression methods:

$$\langle \text{ ZIP, WinZip, PKZIP } \rangle.$$

The `server_hello` message contains the cryptographic algorithms selected by the server. In particular, it consists of the following information:

1. *Version number v_s*: $v_s = \min\{v_c, v\}$, where v is the highest SSL version installed at the server-side computer.
2. *Pseudorandom string r_s*: It is a 32-byte string consisting of a 4-byte time stamp and a 28-byte nonce generated by a pseudorandom number generator at the server-side computer.
3. *Session ID S_s*: If $S_c = 0$, then S_s is equal to the new session ID; otherwise, $S_s = S_c$.
4. *Cipher suite*: It is a list of a public-key encryption algorithm, a symmetric-key encryption algorithm, and a hash function selected by the server from the client's cipher suite. For example, the server may select

$$\langle \text{ RSA, AES-128, Whirlpool } \rangle$$

as its cipher suite.

5. *Compression method*: It is a compression method selected by the server from the list of client's compression methods. For example, the server may select WinZip as its compression method.

5.4.1.2 Phase 2: Authenticate Server and Exchange Key

The server sends the following information to the client:

1. Server's public-key certificate.
2. Server's key-exchange information.
3. Server's request of client's public-key certificate.
4. Server's closing statement of `server_hello`.

Because the client may not have a public-key certificate, and because the client's identity can be verified through the client's credit card information and the standard methods of authenticating credit card holders, Step 3 is often omitted. If the server selects RSA to exchange keys, then Step 2 can also be omitted.

5.4.1.3   Phase 3: Authenticate Client and Exchange Key

The client responds to the server with the following information:

1. Client's public-key certificate.
2. Client's key-exchange information.
3. Client's ICV of its public-key certificate.

The key-exchange information between the server and the client is used to generate a master key between them.

If the server did not ask for the client's public-key certificate, then the first item and the third item are omitted in the client's reply.

If the server in Phase 1 chooses RSA to exchange secret keys, then the client generates and exchanges a secret key as follows: the client first verifies the signature of the server's public-key certificate. If verified, the client obtains the server's public key K_s^u. It then generates a 48-byte pseudorandom string s_{pm}, referred to as a *pre-master secret*. It then uses the server's public key K_s^u to encrypt s_{pm} using RSA and sends the ciphertext string as key-exchange information to the server. Thus, the client and the server both have the following strings:

$$ r_c, \quad r_s, \quad s_{pm}. $$

As s_{pm} is encrypted before it is transmitted, only the end users, namely, the client and the server, know the value of s_{pm}.

The client and the server calculate the *master secret* s_m as follows:

$$
\begin{aligned}
s_m = &\, H_1(s_{pm} \parallel H_2('A' \parallel s_{pm} \parallel r_c \parallel r_s)) \parallel \\
&\, H_1(s_{pm} \parallel H_2('BB' \parallel s_{pm} \parallel r_c \parallel r_s)) \parallel \\
&\, H_1(s_{pm} \parallel H_2('CCC' \parallel s_{pm} \parallel r_c \parallel r_s)),
\end{aligned}
$$

where H_1 and H_2 are hash functions (note that SSL uses MD5 as the default hash function for H_1 and SHA-1 as the default hash function of H_2), $'A'$, $'BB'$, and $'CCC'$ denote, respectively, the ASCII code of A, BB, and CCC.

5.4.1.4   Phase 4: Complete Handshake

The client and the server send to each other a `change_cipher_spec` message and a `finish` message to close the handshake protocol. Both sides determine whether they have calculated the same master secret. For this purpose, the `finish` message sent from each side must contain a hash value of the master secret s_m it has calculated.

After the handshake protocol is ended, both sides calculate a secret-key block K_b using the same method for calculating the master secret s_m. The only difference is to replace s_{pm} with s_m. That is,

$$
\begin{aligned}
K_b = {} & H_1(s_m \parallel H_2('A' \parallel s_m \parallel r_c \parallel r_s)) \parallel \\
& H_1(s_m \parallel H_2('BB' \parallel s_m \parallel r_c \parallel r_s)) \parallel \\
& H_1(s_m \parallel H_2('CCC' \parallel s_m \parallel r_c \parallel r_s)) \parallel \\
& H_1(s_m \parallel H_2('DDDD' \parallel s_m \parallel r_c \parallel r_s)) \parallel \\
& \cdots
\end{aligned}
$$

SSL then divides K_b into six blocks, each of which forms a secret key. The six secret keys that are obtained are put into two groups:

$$\text{Group I: } (K_{c1}, K_{c2}, K_{c3})$$
$$\text{Group II: } (K_{s1}, K_{s2}, K_{s3})$$

That is,

$$K_b = K_{c1} \parallel K_{c2} \parallel K_{c3} \parallel K_{s1} \parallel K_{s2} \parallel K_{s3} \parallel Z,$$

where Z is the remaining substring. The first group of secret keys is used to protect packets from the client to the server, denoted by

$$(K_{c1}, K_{c2}, K_{c3}) = (K_{c,\text{HMAC}}, K_{c,E}, IV_c),$$

where $K_{c,\text{HMAC}}$ is used as the secret key for an HMAC algorithm, $K_{c,E}$ as the secret key for a symmetric-key encryption algorithm, and IV_c as the initial vector for running the encryption algorithm under the CBC mode. The second group of secret keys is used to protect packets from the server to the client, denoted by

$$(K_{s1}, K_{s2}, K_{s3}) = (K_{s,\text{HMAC}}, K_{s,E}, IV_s).$$
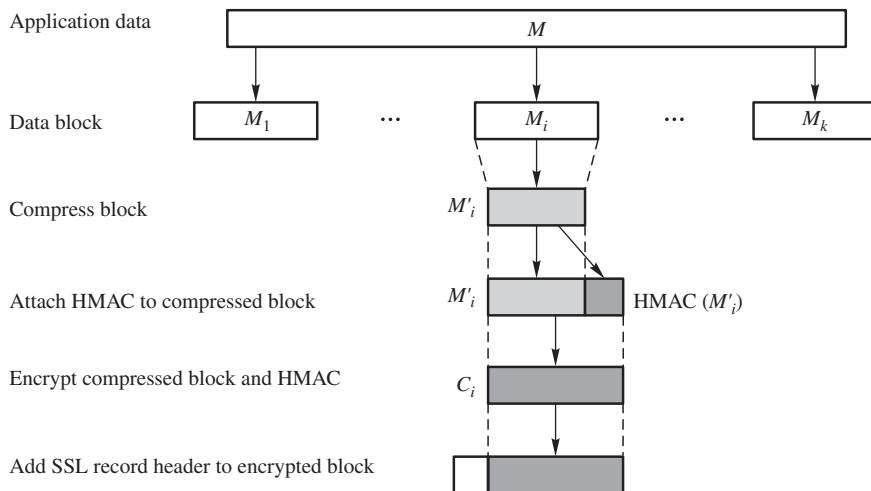
The usage of each of these keys is the same as that of the corresponding key for the client.

The handshake protocol is responsible for establishing a secure communication session between the client and the server. After this, the client and the server will use the SSL record protocol to protect their communications.

5.4.2 SSL Record Protocol

The handshake protocol determines what encryption algorithms, parameters, secret keys, and compression algorithms are to be used in the underlying communication session. The record protocol uses these algorithms, parameters, secret keys, and compression algorithms to protect data. Let M be a message to be sent from the client to the server. The SSL record protocol at the client site will first divide M into a sequence of data blocks

$$M_1, M_2, \cdots, M_k.$$

Figure 5.10 SSL record protocol

It will then compress, authenticate, and encrypt each data block (see Figure 5.10) and transmit encrypted blocks to the server.

In particular, let CX, H, and E be, respectively, the compression algorithm, the HMAC algorithm, and the symmetric-key algorithm selected by both sides during the SSL handshake protocol. For each data block $M_i, i = 1, 2, \cdots, k$, the client does the followings:

1. Compress M_i to get $M_i' = CX(M_i)$.
2. Authenticate M_i' to get $M_i'' = M_i' \parallel H_{K_{c,\mathrm{HMAC}}}(M_i')$.
3. Encrypt M_i'' to get $C_i = E_{K_{c,E}}(M_i'')$.
4. Encapsulate C_i to get $P_i = [\text{SSL record header}] \parallel C_i$.
5. Transmit P_i to the server.
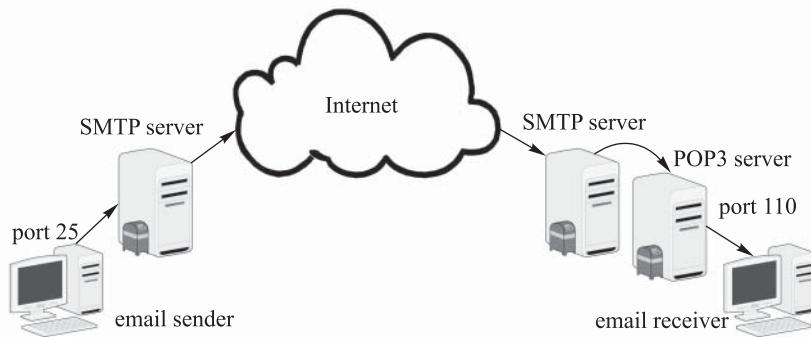
When the server receives a P_i from the client, it first extracts C_i from P_i. It then decrypts C_i to get M_i'', extracts M_i' and $H_{K_{c,\mathrm{HMAC}}}(M_i')$, verifies the authentication code, and decompress M_i' to get M_i.

Under SSL, any data sent from the server to the client also goes through the same process. This provides data confidentiality and integrity for data transmitted between the client and the server.

5.5   PGP and S/MIME: Email Security Protocols

There are a number of security protocols at the application layer. The most used of these protocols are email security protocols and remote login security protocols. The former includes PGP and S/MIME. The latter includes SSH. In addition, Kerberos authentication for local area networks is also popular.

Simple Mail Transfer Protocol (SMTP) and Post Office Protocol (POP) are the basic email protocols. Both SMTP and POP are TCP protocols. POP3 is the commonly used version of

Figure 5.11   SMTP and POP3 flow diagram

POP, where POP3 and POP use different port numbers. Other than this, we use POP and POP3 interchangeably in this book, unless otherwise stated. SMTP is responsible for transmitting email, while POP3 is responsible for receiving email (see Figure 5.11).

SMTP was designed to transmit text messages encoded using 7-bit ASCII codes. This presents a problem, for encrypted email messages are in binary format. To solve this problem, one needs to devise a method to convert a binary string to a character string for transmission and convert it back to the original binary string at the destination.

As certain binary strings represent ASCII control codes, it is difficult to represent binary strings directly using ASCII codes. However, it is possible to represent a binary string using hexadecimal digits, because the basic storage of binary data is byte. Each hexadecimal digit is represented by an ASCII code. This method, however, is not economical (see Exercise 5.23). As the number of characters used in English is over 2^6 but less than 2^7, using a 6-bit binary string to represent an English character becomes a natural choice. This gives rise to Base64 encoding, a.k.a. Radix-64 encoding. For a detailed description of Base64 encoding, the reader is referred to Appendix D.

5.5.1   Basic Email Security Mechanisms

Email security is a classic application of cryptographic algorithms. Let E and D denote a symmetric-key encryption algorithm and its decryption algorithm. Let $\hat{E}$ and $\hat{D}$ denote a public-key encryption algorithm and its decryption algorithm. When there is no confusion in the context, the hat may be omitted.

Suppose that Alice wants to prove to Bob that a certain email message M he receives is from Alice. She can do so by sending the following string to Bob:

$$M \parallel \hat{E}_{K_A^r}(H(M)) \parallel \mathrm{CA}\langle K_A^u \rangle,$$

where K_A^u and K_A^r are Alice's public key and private key, respectively. After receiving

$$M \parallel S_M \parallel \mathrm{CA}\langle K_A^u \rangle$$

from Alice, where S_M is the signed copy of M using Alice's private key. Bob first verifies CA's signature on the public-key certificate $\mathrm{CA}\langle K_A^u \rangle$ and extracts K_A from it. He then extracts

M and verifies whether $S_M = \hat{E}_{K_A^r}(H(M))$. If so, then Bob is convinced that M is indeed from Alice.

Suppose that Alice wants to ensure that M remains confidential during transmission and she knows Bob's public key K_B^u. She sends the following string to Bob:

$$E_{K_A}(M) \parallel \hat{E}_{K_B^u}(K_A),$$

where K_A is Alice's secret key. After receiving this string from Alice, Bob first decrypts $\hat{E}_{K_B^u}(K_A)$ using his private key to obtain K_A; that is, he computes

$$\hat{D}_{K_B^r}(\hat{E}_{K_B^u}(K_A)) = K_A.$$

He then uses K_A to decrypt $E_{K_A}(M)$ to obtain M; that is, he computes

$$D_{K_A}(E_{K_A}(M)) = M.$$

Phil Zimmermann incorporated cryptographic algorithms and mechanisms in an email system he called PGP and made it easy to use. He published the PGP source code in 1991. Today, PGP is owned by PGP Corporation, which continues the tradition of publishing the source code of each new version of PGP for peer review. This practice helps to ensure product integrity. PGP is now owned by Symantec Corporation.

5.5.2   PGP

PGP implements all major cryptographic algorithms, the ZIP compression algorithm, and the Base64 encoding algorithm. It can be used to authenticate a message, encrypt a message, or both. PGP follows the following general format: authentication, ZIP compression, encryption, and Base64 encoding. The Base64 encoding procedure makes the message ready for SMTP transmission. Figure 5.12 shows the general format of a PGP message Alice sends to Bob.
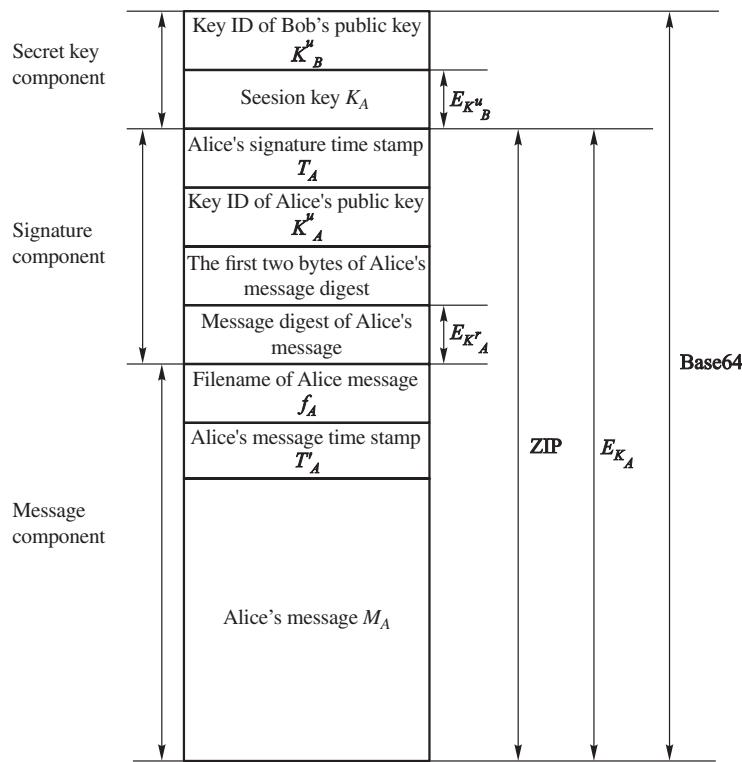
Alice and Bob each maintain a public-key ring and a private-key ring.

The secret-key component consists of Alice's session key K_A encrypted using Bob's public key K_B^u and the key ID of K_B^u. The encryption ensures the confidentiality of K_A, and the key ID informs which public key in his key ring Bob should use to decrypt it.

The signature component consists of a time stamp when Alice signs the message, the key ID of Alice's public key K_A^r to inform Bob which key in his key ring he should use to verify Alice's signature, the first two bytes of the message digest of Alice's message for Bob to verify the message digest, and the encrypted message digest using Alice's private key K_A^r that serves as Alice's signature on the message.

The message component consists of the file name of the message, the time stamp for when the message was created, and the message itself.

The signature component and the message component are compressed using ZIP and encrypted using the session key K_A Alice generated. The new binary string produced by this process is then attached to the secret-key Component, and the whole string is converted to a Base64 string. This format is used for Alice to provide authentication and confidentiality of her message. If she only wants to authenticate her message, Alice does not need to include the secret-key component and the encryption part using the session key. If she only wants to provide confidentiality of the message, Alice does not need to include the signature component.

Figure 5.12   The general format of a PGP message Alice sends to Bob

5.5.3   S/MIME

SMTP can only handle 7-bit ASCII text messages. While POP can handle other content types besides 7-bit ASCII, POP may, under a common default setting, download all the messages stored in the mail server to the user's local computer. After that, POP will remove these messages from the mail server. This makes it difficult for the user to read his messages from multiple computers. Neither SMTP nor POP3 can authenticate or encrypt email messages.

The Multipurpose Internet Mail Extension protocol (MIME) was designed to support sending and receiving email messages in various formats, including nontext files generated by word processors, graphics files, sound files, and video clips. Moreover, MIME allows a single message to include mixed types of data in any combination of these formats.

The Internet Mail Access Protocol (IMAP), operated on TCP port 143, stores incoming email messages in the mail server until the user deletes them deliberately. This allows the user to access his mailbox from multiple machines and download messages to a local machine without deleting it from the mailbox in the mail server.

To solve the third problem and to support multimedia email messages at the same time, RSA Security extended MIME in 1999 to include cryptographic algorithms for authenticating and encrypting messages. The new protocol is referred to as S/MIME. S/MIME version 3 was designated by IETF to be the email security standard. In terms of security functionality, S/MIME

is similar to PGP. It can automatically authenticate, encrypt, or authenticate and encrypt all outgoing email message. It can also authenticate, encrypt, or authenticate and encrypt a specific email message. However, unlike PGP, S/MIME requires signatories to possess public-key certificates.

S/MIME specifies encryption algorithms and encoding formats. It uses X.509 PKI and public-key certificates. It supports standard symmetric-key encryption algorithms, public-key encryption algorithms, digital signature algorithms, cryptographic hash algorithms, and compression functions. S/MIME uses MIME formats to enclose encrypted messages.

5.6   Kerberos: An Authentication Protocol
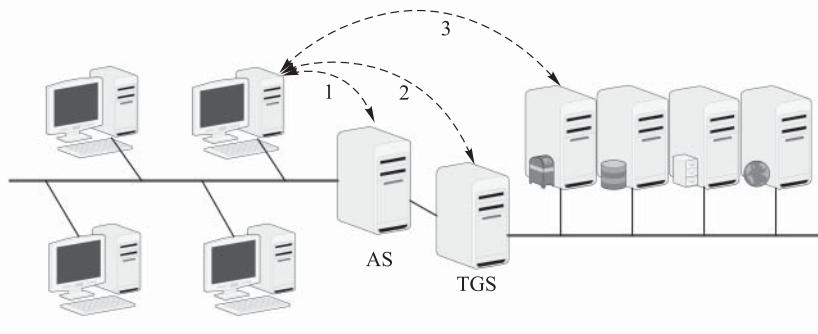
The use of public-key certificates is arguably the best method to authenticate users across networks. This method, however, requires a PKI, which incurs substantial overhead costs. For users in the same local area network, this method is not necessary, because users in the local area networks have their user names and password information stored in the local server, which are used to authenticate users. Kerberos, named after a monstrous three-headed, snake-tailed guard dog of Hades in Greek mythology, is an authentication protocol designed for users in the same local area network without using public-key cryptography.

5.6.1   Basic Ideas

Kerberos was designed and developed in the late 1980s by a research team at Massachusetts Institute of Technology, led by Steve Miller and Clifford Neuman, as part of Project Athena. The goal of Kerberos is to make it easy for users to authenticate themselves to various servers at the local network (e.g., email server, Web server, and file server) for obtaining services, without needing to type in their passwords every time before they use the service. When a user wants to use a certain service provided by a server in his local network, he needs to prove to the server that he is a legitimate user. On the other hand, servers should also authenticate themselves to users so that users know that they are using a legitimate service. While users may use their user names and passwords to authenticate themselves to a server each time they are using it, this practice is obviously cumbersome. In the meantime, each server must also maintain an up-to-date record of user names and passwords, which makes it arduous to manage. Kerberos uses symmetric-key encryption algorithms and electronic passes to solve this problem. An electronic pass is referred to as a *ticket*.

A ticket is used to authenticate its holder's identity. Kerberos uses two different types of tickets and two special servers to issue tickets to users. One is called *authentication server* (*AS*), and the other is called the *ticket-granting server* (*TGS*). AS manages users; it keeps and maintains records of user login names and their password information (e.g., cryptographic hash values of the passwords). TGS manages servers; it shares a different master key with each server. TGS knows the names of all the users, which can be made public in the local network, but only AS knows user password information.

When first logging on to the network, the user must prove to AS his identity by typing his user name and password. After AS authenticates the user, AS issues a TGS-ticket to the user. When he wants to use a service provided by server V, the user presents his TGS-ticket to TGS. TGS verifies the legitimacy of the TGS-ticket and issues a V-ticket to the user. This type of

Figure 5.13   Single-realm Kerberos, where the three dash lines indicate, respectively, the first-phase, the second-phase, and the third-phase communications

ticket is referred to as a *server ticket*, which is tied to that specific server. The user uses his V-ticket to request service from V.

Kerberos can be used in a single LAN with one AS and one TGS. It can also be used across several local LANs. The first kind is referred to as *single-realm Kerberos* and the latter as *multiple-realm Kerberos*. In other words, a Kerberos realm is a set of users and servers, all using the same AS for authentication.

5.6.2   Single-Realm Kerberos

Figure 5.13 shows the flow diagram of single-realm Kerberos protocol, which consists of three phases. In the first phase, the user requests a TGS-ticket. In the second phase, the user requests a server ticket. In the third phase, the user presents his server ticket to the server to obtain service.

We use the notations specified in Table 5.2 to describe Kerberos protocol steps.

5.6.2.1   Phase 1: AS Issues a TGS-Ticket to User

1. $U \rightarrow AS$: $ID_U \parallel ID_{TGS} \parallel t_1$;
2. $AS \rightarrow U$: $E_{K_U}(K_{U,TGS} \parallel ID_{TGS} \parallel t_2 \parallel LT_2 \parallel \mathsf{Ticket}_{TGS})$,
   $\mathsf{Ticket}_{TGS} = E_{K_{TGS}}(K_{U,TGS} \parallel ID_U \parallel AD_U \parallel ID_{TGS} \parallel t_2 \parallel LT_2)$.

5.6.2.2   Phase 2: TGS Issues a Server Ticket to User

1. $U \rightarrow TGS$: $ID_V \parallel \mathsf{Ticket}_{TGS} \parallel \mathsf{Auth}_{U,TGS}$,
   $\mathsf{Auth}_{U,TGS} = E_{K_{U,TGS}}(ID_U \parallel AD_U \parallel t_3)$;
2. $TGS \rightarrow U$: $E_{K_{U,TGS}}(K_{U,V} \parallel ID_V \parallel t_4 \parallel \mathsf{Ticket}_V)$,
   $\mathsf{Ticket}_V = E_{K_V}(K_{U,V} \parallel ID_U \parallel AD_U \parallel ID_V \parallel t_4 \parallel LT_4)$.

5.6.2.3   Phase 3: User Requests Service from Server

1. $U \rightarrow V$: $\mathsf{Ticket}_V \parallel \mathsf{Auth}_{U,V}$,
   $\mathsf{Auth}_{U,V} = E_{K_{U,V}}(ID_U \parallel AD_U \parallel t_5)$;
2. $V \rightarrow U$: $E_{K_{U,V}}(t_5 + 1)$.

Table 5.2 Notations and their meanings used to describe Kerberos protocol
steps

Notation	Meaning
U	User
V	Server
ID_{U}	U's ID
ID_{TGS}	TGS's ID
t_i	Time stamp
E_K	Symmetric-key encryption with secret key K
K_{U}	The secret key derived from user U's password
$K_{\mathrm{U,TGS}}$	The session key generated by AS to be used by U and TGS
K_{TGS}	The master key shared by AS and TGS
K_{V}	The master key shared by TGS and V
$K_{\mathrm{U,V}}$	The session key generated by TGS to be used by U and V
LT_i	Expiration time
$\mathsf{Ticket}_{\mathrm{TGS}}$	TGS-ticket issued to U by AS
$\mathsf{Ticket}_{\mathrm{V}}$	Server ticket for using server V issued to U by TGS
AD_{U}	U's MAC address
$\mathsf{Auth}_{\mathrm{U,TGS}}$	Authentication code generated using secret key $K_{\mathrm{U,TGS}}$
$\mathsf{Auth}_{\mathrm{U,V}}$	Authentication code generated using secret key $K_{\mathrm{U,V}}$

5.6.2.4   Dissection

In the first phase, the request sent from U to AS is not encrypted, where the time stamp is used
to resist message replay attacks. We note that Kerberos is mainly used in local area networks
and that in local area networks it is reasonable to assume that all the clocks on networked
computers are synchronized. Thus, time stamps alone are sufficient to resist message replay
attacks. On the basis of what it receives, AS finds user's password P_{U}. It then computes a
secret key K_{U} on the basis of P_{U}. AS then generates a session key $K_{\mathrm{U,TGS}}$ for U and TGS and
encrypts it using the master key K_{TGS} shared by AS and TGS. AS encrypts

$$K_{\mathrm{U,TGS}} \parallel ID_{\mathrm{U}} \parallel AD_{\mathrm{U}} \parallel ID_{\mathrm{TGS}} \parallel t_2 \parallel LT_2$$

and generates a TGS-ticket, where ID_{U} is U's ID, used to show TGS U's login name; AD_{U}
is U's MAC address, used to specify that this TGS-ticket can only be used by the computer
whose MAC address is AD_{U}. Time stamp t_2 and expiration time LT_2 are used to prevent
eavesdroppers from reusing this TGS-ticket.

After receiving

$$E_{K_{\mathrm{U}}}(K_{\mathrm{U,TGS}} \parallel ID_{\mathrm{TGS}} \parallel t_2 \parallel LT_2 \parallel \mathsf{Ticket}_{\mathrm{TGS}})$$

from AS, U uses the same method used by AS to compute K_{U} from P_{U} (where U needs to type
in P_{U} on the computer with MAC address AD_{U}). U then uses K_{U} to decrypt what he received
from AS to get the session key $K_{\mathrm{U,TGS}}$ and TGS-ticket $\mathsf{Ticket}_{\mathrm{TGS}}$. U uses the TGS-ticket
to request a server ticket from TGS. He can repeat this request as many times as he wants
before LT_2 expires. For example, U may want to send an email message, browse a Website,

and fetch a file. Therefore, he needs to authenticate himself to the local email server, the local Webserver, and the local file server. Suppose that U wants to obtain service from server V. He is now entering the second phase of Kerberos.

In the second phase, U sends his ID, V's ID, his TGS-ticket, and the encrypted string, using the session key $K_{U,TGS}$, of his ID, his MAC address, and a new time stamp t_3 to TGS. TGS checks that the TGS-ticket is still valid by checking the time stamp and the expiration time contained in it. TGS then verifies U's ID and U's MAC address against those contained in the TGS-ticket, and that the time stamp t_3 did not occur before. TGS then generates a session key $K_{U,V}$ for U and V and a server ticket Ticket_V for V. The server ticket is encrypted using the master key K_V shared by TGS and V, which will allow V to authenticate that the ticket indeed was issued by TGS.

In the third phase, U sends to V the TGS-ticket Ticket_V he obtained from TGS, and the encrypted string of U's ID, U's MAC address, and a new time stamp t_5. V then verifies all the information to determine whether U is a legitimate user. V then adds t_5 by 1, encrypts it using the session key $K_{U,V}$, and sends the encrypted string to U to indicate that authentication is completed and U is going to receive the service he has requested.
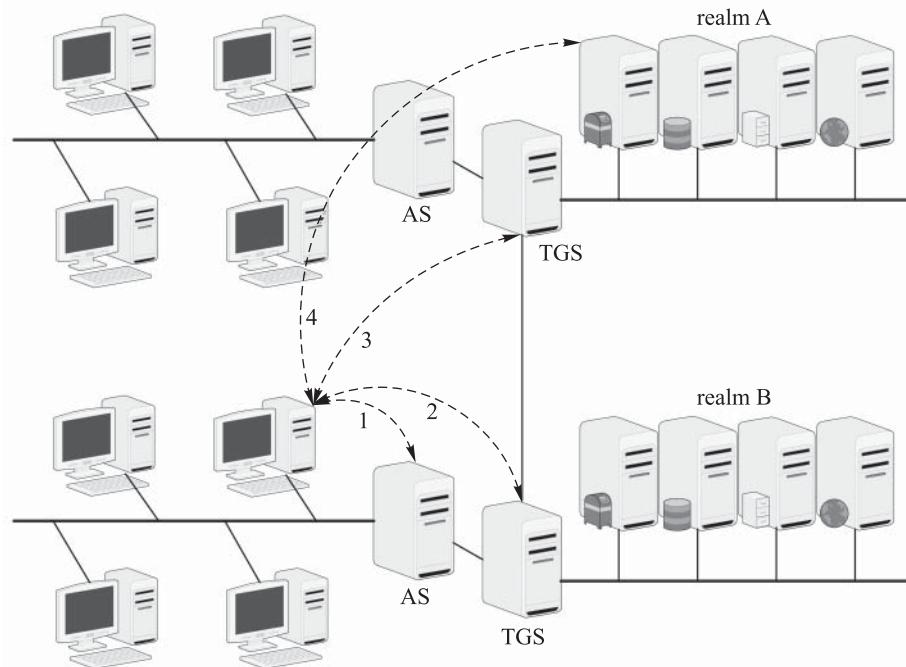
5.6.3   Multiple-Realm Kerberos

Suppose that the Department of Computer Science and the Department of Computer Engineering at the same university occupy two buildings nearby and have each installed single-realm Kerberos in their LANs, both of which are connected to the university network. Suppose that the Department of Computer Science has installed a new software package and several computer engineering professors and students want to use it, which is covered by the licensing agreement. However, the computer engineering professors and students need to be authenticated to the computer science server. This may be done by creating new accounts for these professors and students in the computer science Kerberos, which would add to the system management burden, or by using multiple-realm Kerberos through mutual authentication of each department's TGS.

Multiple-realm Kerberos is based on single-realm Kerberos with a slight modification. Suppose that a user U in a single-realm Kerberos A wants to use a service provided in a different but nearby single-realm Kerberos. The proximity ensures time synchronization needed in the protocol. Multiple-realm Kerberos consists of four phases. In the first phase, U sends a request to the local AS for a ticket of the local TGS. After verifying that U is a legitimate user, the local AS grants U's request. In the second phase, U uses the local TGS-ticket to request the local TGS to grant him a TGS-ticket of the neighboring TGS. The local TGS issues a TGS-ticket of the neighboring TGS. In the third phase, U uses his TGS-ticket of the neighboring TGS to request a server ticket for the server in the neighboring network. TGS in the neighboring network issues a server ticket to U. In the forth phase, U uses the server ticket of the neighboring network to obtain service provided in the neighboring network.

Figure 5.14 shows a flow diagram of multiple-realm Kerberos.

We use AS to denote the authentication server in the local realm and AS' the authentication server in the neighbor realm, and we call them, respectively, *local authentication server* and *neighbor authentication server*. We define TGS and TGS' in a similar manner. Multiple-realm Kerberos consists of four phases.

Figure 5.14   Flow diagram of multiple-realm Kerberos

5.6.3.1   Phase 1: Local AS Issues a Local TGS-Ticket to User

1. $U \rightarrow AS$: $ID_U \parallel ID_{TGS} \parallel t_1$;
2. $AS \rightarrow U$: $E_{K_U}(K_{U,TGS} \parallel ID_{TGS} \parallel t_2 \parallel LT_2 \parallel \mathsf{Ticket}_{TGS})$,
   $\mathsf{Ticket}_{TGS} = E_{K_{TGS}}(K_{U,TGS} \parallel ID_U \parallel AD_U \parallel ID_{TGS} \parallel t_2 \parallel LT_2)$.

5.6.3.2   Phase 2: Local TGS Issues a Neighbor TGS-Ticket to User

1. $U \rightarrow TGS$: $ID_V \parallel \mathsf{Ticket}_{TGS} \parallel \mathsf{Auth}_{U,TGS}$,
   $\mathsf{Auth}_{U,TGS} = E_{K_{U,TGS}}(ID_U \parallel AD_U \parallel t_3)$;
2. $TGS \rightarrow U$: $E_{K_{U,TGS}}(K_{U,TGS'} \parallel ID_{TGS'} \parallel t_4 \parallel \mathsf{Ticket}_{TGS'})$,
   $\mathsf{Ticket}_{TGS'} = E_{K_{TGS'}}(K_{U,TGS'} \parallel ID_U \parallel AD_U \parallel ID_{TGS'} \parallel t_4 \parallel LT_4)$.

5.6.3.3   Phase 3: Neighbor TGS′ Issues a Server Ticket to User

1. $U \rightarrow TGS'$: $ID_V \parallel \mathsf{Ticket}_{TGS'} \parallel \mathsf{Auth}_{U,TGS'}$,
   $\mathsf{Auth}_{U,TGS'} = E_{K_{U,TGS'}}(ID_U \parallel AD_U \parallel t_5)$;
2. $TGS' \rightarrow U$: $E_{K_{U,TGS'}}(K_{U,V} \parallel ID_V \parallel t_6 \parallel \mathsf{Ticket}_V)$,
   $\mathsf{Ticket}_V = E_{K_V}(K_{U,V} \parallel ID_U \parallel AD_U \parallel ID_V \parallel t_6 \parallel LT_6)$.

5.6.3.4   Phase 4: User Requests Service from Neighbor Server

1. U → V: $\text{Ticket}_\text{V} \parallel \text{Auth}_{\text{U,V}}$,
          $\text{Auth}_{\text{U,V}} = E_{K_{\text{U,V}}}(ID_\text{U} \parallel AD_\text{U} \parallel t_7)$;
2. V → U: $E_{K_{\text{U,V}}}(t_7 + 1)$.

Dissection of multiple-realm Kerberos is left as exercise (see Exercise 5.32).
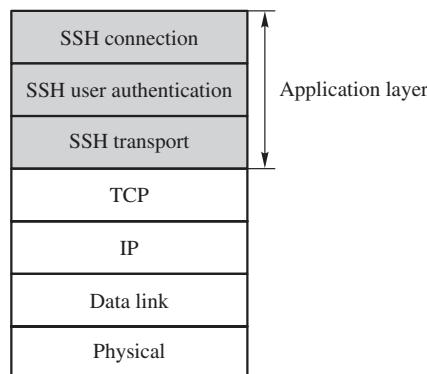
5.7   SSH: Security Protocols for Remote Logins

Telnet, rlogin, rsh, rcp, and FTP were once popular application-layer protocols for users to log on to a remote computer and to transfer or copy files between different computers. These protocols, however, transmit data in plaintext without any cryptographic protection, and so they are vulnerable to password sniffing, eavesdropping, IP spoofing, and other types of security attacks.

To protect remote logins from security attacks, a Finnish researcher Tatu Ylönen devised in 1995 a security protocol called SSH. SSH creates a secure connection between two computers using authentication and encryption algorithms. It also supports data compression. SSH also provides security protection for file transfers (SFTP) and file copy (SCP).

SSH is a client-server application protocol. It is divided into three layers that are housed in the application layer of the TCP/IP network model. They are the *connection* layer, the *user authentication* layer, and the *transport* layer. Figure 5.15 shows the SSH architecture.

The SSH transport layer is the bottom layer. It is used to authenticate server, exchange keys in the initial phase, and set up encryption and compression algorithms. The user's computer ensures that it is connecting to the same server computer during subsequent sessions. Subsequent packets transmitted between the client and the server are all encrypted using a symmetric-key encryption algorithm.

Figure 5.15   SSH architecture

The SSH user authentication layer is the next layer on top of the SSH transportation layer. It is used to authenticate the user (i.e., the client) to the server. The user may authenticate himself using password or public-key cryptography.

The SSH connection layer is the highest layer. It is used to set up multiple channels for different applications in a single SSH connection, each transferring data in both directions.

As SSH encrypts everything it sends and receives, SSH provides data confidentiality and data integrity between the client computer and the server computer and resists IP spoofing attacks.

SSH is a widely used security protocol with several free downloads (see Exercise 5.35).

5.8 Electronic Voting Protocols

The goal of electronic voting protocols is to allow a group of distributed parties to vote in an election. These elections somewhat parallel traditional elections. In a traditional election, voters must be registered with the municipality. On the voting day, a voter must authenticate himself to a trusted third party, which then provides the voter with a ballot. The voter then goes into a cubicle and indicates his selection. The voter places the completed ballot through a machine that adds the ballot to the tally.

In electronic voting, a similar experience is provided. Voters must still authenticate to the system. However, the ballots are protected through the use of cryptography. Moreover, the election is publicly auditable at every step in the process. Electronic voting protocols generally seek to provide the following two guarantees:

1. *Ballot casting assurance*: Each voter gains personal assurance that his ballot is correctly cast.
2. *Universal verifiability*: Any observer can verify that all ballots are properly tallied.

Traditionally, electronic voting protocols are divided into two main phases: *ballot preparation* and *ballot tallying*. In the ballot preparation phase, a voter prepares an encrypted ballot that represents his choice. In the *ballot tallying* phase, the set of encrypted ballots is cryptographically processed to produce a tally and a proof of correctness of that tally. To help satisfy the security guarantees outlined previously, every encrypted ballot and the associated voter identification are posted to a public bulletin board for easy auditing.

To understand the details of electronic voting protocols, we must first understand the following three cryptographic primitives: interactive proofs, re-encryption schemes, and threshold cryptography. We describe each topic in turn and conclude with a discussion of electronic voting protocol constructions.

5.8.1 Interactive Proofs

An interactive proof is a protocol that consists of two parties named Peggy and Victor. Peggy is in possession of some secret. Victor wants to be convinced that Peggy actually possesses the secret. However, Peggy does not want to just reveal the secret. In order for Peggy to prove to Victor that she knows the secret, she agrees to interact with Victor. Victor will repeatedly challenge Peggy to answer queries that she would be able to answer only if she knew the secret. Once Peggy has succeeded a polynomial number of times, then Victor is convinced that

Peggy knows the secret. Victor and Peggy are subject to a polynomial-time bound (possibly probabilistic).

We explain the ideas of interactive proofs using the canonical example of the graph isomorphism problem. Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a *bijection* $\varphi : V_1 \rightarrow V_2$. A mapping φ is called a bijection if it is *injective* (one-to-one) and *surjective* (onto). A map is injective if every element of the domain is mapped to a distinct element in the range. A map is surjective if every element y in the range has a corresponding element x in the domain such that $\varphi(x) = y$. When φ is a bijection for the graph isomorphism problem, it must additionally be the case that for all vertices in V_1, the edge $(u, w) \in E_1$ if and only if $(\varphi(u), \varphi(w)) \in E_2$.

Assume for the moment that Peggy knows that graph G_1 and G_2 are isomorphic, denoted by $G_1 \simeq G_2$, which implies that Peggy knows the associated bijection φ. Peggy wishes to convince Victor that she knows that $G_1 \simeq G_2$. The following is one round of the interactive protocol Peggy and Victor:

1. **Setup**: Peggy constructs a graph H such that $G_1 \simeq H$. By construction Peggy knows the bijection $\varphi' : G_1 \rightarrow H$. Using φ and φ' Peggy constructs two maps: $\sigma_0 = \varphi'$ and $\sigma_1 = \varphi' \circ \varphi^{-1}$. Peggy finishes by sending H to Victor.
2. **Selection**: Victor flips a fair coin and sends the digit 1 to Peggy if the coin comes up head. If the coin comes up tail, Victor sends the digit 0 to Peggy.
3. **Verification**: Peggy sends to Victor σ_i where i is the value Victor generated in the selection step. Victor then verifies that σ_i is a valid bijection that shows $G_{i+1} \simeq H$.

With a little effort, we can verify that if Peggy does in fact know the correct bijection φ that shows $G_1 \simeq G_2$, then no matter how Victor's coin lands, he will obtain the correct result. In the case that Victor's coin comes up tails, Peggy sends the bijection σ_0 to Victor and by construction $G_1 \simeq H$. In the case that Victor's coin comes up heads, Peggy sends the bijection $\sigma_1 = \varphi' \circ \varphi^{-1}$. As Victor is trying to show that $G_2 \simeq H$, we apply σ_1 to G_2 map G_2 to G_1 and then use the isomorphism φ' to map G_1 to H.

5.8.2 Re-encryption Schemes

Re-encryption is a cryptographic primitive used in electronic voting protocols. A re-encryption scheme seeks to allow users to create a new ciphertext whose plaintext is equivalent to an existing ciphertext's plaintext. Moreover, this new ciphertext can be created without knowing the plaintext message.

We can construct a re-encryption scheme using the Elgamal encryption system. Recall from Chapter 3 that the Elgamal cryptographic system works over the multiplicative group Z_p^* for a prime number p. As a primitive root of p is a generator of the multiplicative group Z_p^*, it is also customary to use g to denote a generator. We use this notation here (as opposed to the use of a in Chapter 3).

The public key for the scheme is $g^X \bmod p$, and the private key is $X < p$, which is randomly selected. An example encryption of a message $M < p$ is

$$(C_1, C_2) = \left(g^k \bmod p, M g^{Xk} \bmod p \right),$$

where $k < p$ is a number chosen at random.

To re-encrypt an Elgamal ciphertext, it is sufficient to select a new random value k' and produce from the ciphertext $(C_1' C_2')$ a new ciphertext

$$\left(C_1 g^{k'}, C_2 g^{Xk'} \right).$$

It remains to show that the re-encryption scheme does produce a ciphertext with plaintext equivalent to (C_1, C_2). We have

$$\begin{aligned}
(C_1', C_2') &= \left(C_1 g^{k'} \bmod p, C_2 g^{Xk'} \bmod p \right) \\
&= \left(g^k g^{k'} \bmod p, M g^{Xk} g^{Xk'} \bmod p \right) \\
&= \left(g^{k+k'} \bmod p, M g^{Xk+Xk'} \bmod p \right) \\
&= \left(g^{k+k'} \bmod p, M g^{X(k+k')} \bmod p \right).
\end{aligned}$$

Note that $k + k'$ is still a random number, This lends evidence to the fact that these re-encryption schemes produce ciphertexts that have the same plaintexts. In addition, if the party executing the re-encryption protocol were to reveal k', then it would be publicly verifiable that the two ciphertexts represent the same plaintext messages. In fact, under this notion, the integer k' can be thought of as a proof.

5.8.3 Threshold Cryptography

Threshold cryptography is a form of PKC, where a predetermined number of parties must cooperate to decrypt a ciphertext. To construct a system of this form, the participating parties must jointly produce a public key. Each participant begins by generating and publishing an encryption key. These keys are then aggregated in some way to form a public key, which will be published to the world.

We use an Elgamal-based system to demonstrate a construction of this primitive. To understand this primitive, one must first understand the notion of a *secret sharing scheme*. A secret sharing scheme is a cryptographic primitive that allows n parties to share a secret and at some later time allows m parties ($m \leq n$) to cooperate and recover the secret (technically, this is called a threshold scheme).

The following secret sharing scheme was devised by Adi Shamir in 1979. Let s be a secret to be shared by n parties, where s is a number. Construct an $(m-1)$-degree polynomial with $m \leq n$:

$$p(x) = a_{m-1} x^{m-1} + a_{m-2} x^{m-2} + \cdots + a_1 x^1 + s.$$

A special party, called the *dealer*, sets up this polynomial and is responsible for distributing the *shares* of the secret. To each of the n parties in the system, the dealer provides a point on the curve generated by the $m-1$ degree polynomial. For example, party 3 may be given the point $(3, p(3))$. Once this process is complete, the dealer destroys the polynomial.

To recover the secret, at least m parties must cooperate to reconstruct the polynomial. In particular, these m parties use interpolation to recover the polynomial and then evaluate the

recovered polynomial at $x = 0$ to obtain s. The common interpolation technique is called Lagrange Interpolation. The Lagrange Interpolation process is given by the following two equations:

$$L(x) = \sum_{j \in \Delta} \ell_{j,\Delta}(x) y_j, \tag{5.2}$$

$$\ell_{j,\Delta}(x) = \prod_{\substack{k \in \Delta \\ k \neq j}} \frac{x - x_k}{x_j - x_k}, \tag{5.3}$$

where Δ is a set of point indices and each point is of the form (x_i, y_i). The number $\ell_j(x)$ is commonly called the Lagrange coefficient.

Combining the Shamir Secret Sharing scheme and the Elgamal PKC in a certain way known as the Pedersen system, we achieve a threshold cryptographic scheme. The Pedersen system consists of a key-selection phase and a key distribution phase.

In the key-selection phase, each user P_i selects a random number $r_i < p$ and publishes $h_i = g^{r_i}$. Once all users have published their values, the public key for the threshold system is derived by

$$h = \prod_{i=1}^{n} h_i = \prod_{i=1}^{n} g^{r_i}.$$

Let r be the private key associated with h.

In the distribution phase, a group of users works collectively to distribute shares of r to all users using the Shamir secret sharing scheme. The process is described as follows:

1. User P_i chooses a random polynomial $f_i(z)$ of degree $m - 1$, where m is the minimum number of users who have to cooperate to recover r. Define f_i by

$$f_i(z) = f_{i,m-1} z^{m-1} + f_{i,m-2} z^{m-2} + \cdots + f_{i,1} z + r_i.$$

   Clearly, $f_i(0) = r_i$, which is user P_i's share of r.
2. User P_i computes and broadcasts $F_{i,j} = g^{f_{i,j}}$ for all $1 \leq j \leq m - 1$.
3. User P_i secretly sends a signed message containing $s_{ij} = f_i(j)$ to user P_j for all $1 \leq j \leq n$ with $j \neq i$.
4. User P_i verifies the share received from P_j by first checking the signature and then verifying the equality

$$g^{s_{ij}} = \prod_{t=0}^{k-1} \left(F_{j,t}\right)^{i^t} = \prod_{t=0}^{k-1} \left(g^{(f_{j,t}) i^t}\right)$$

   for all $j \neq i$.
5. Once satisfied with the signatures and the equality for all j, user P_i computes her share s_i of r as $s_i = \sum_{j=1}^{n} s_{ji}$ and signs h to mark her agreement with the public key for the group.

To decrypt the ciphertext (C_1, C_2), users must cooperate in the following way:

1. Each user P_i broadcasts $w_i = C_1^{s_i}$.

2. Each user computes

$$M = \frac{C_2}{\prod_{j \in \Delta} w_j^{\ell_{j,\Delta}(0)}},$$

where Δ is the set of user identifiers involved in the decryption. Recall that every user can compute the Lagrange coefficient as $x_j = j$ for all j. Therefore, the computation of $\ell_{j,\Delta}(0)$ is well defined.

5.8.4 The Helios Voting Protocol

We describe the Helios voting protocol as a concrete example of electronic voting protocols. The Helios consists of the following five phases:

1. *Vote phase*: Each voter, represented by Alice, needs to confirm that the device she is using to encrypt her vote is operating correctly. When she is satisfied, she can cast her encrypted ballot. To have her ballot counted, Alice *must* authenticate herself to the system.
2. *Publish phase*: The system posts Alice's encrypted vote along with her name to a public bulletin board. This way, Alice, and everyone else, can confirm that Alice did indeed vote.
3. *Shuffle phase*: The shuffle phase is performed after the voting has concluded. The system decouples the votes from the names and mixes the votes.
4. *Tally phase*: The system first tallies the votes in the public view (e.g., on the bulletin board) and then destroys the votes.
5. *Audit phase*: This is an optional phase where any auditor may choose to download all the election data and verify the shuffle and tally phases.

5.8.4.1 Vote and Publish

In the voting phase, Alice is presented with a list of candidates she could select and confirm. Once the selection is confirmed, the device she is using encrypts her ballot using the Threshold Elgamal system. Alice is then given a hash (e.g., SHA-256) of her encrypted vote as a commitment from the system. Alice is then provided with an option to audit or cast her ballot. If Alice chooses to audit her ballot, then she will be provided with the ciphertext of her ballot and the random number used in the Elgamal encryption.

To cast a vote after auditing, Alice must re-vote and obtain a new ciphertext and hash. The system then discards all random values it generated for the Threshold Elgamal encryption. Alice will then be prompted to authenticate herself to the system. Helios is currently implemented as a Webpage and uses username and password to authentical users. Once Alice has authenticated herself, the system will post the encrypted ballot with Alice's identifier to the bulletin board.

5.8.4.2 Shuffle

A *mix network* is used to preserve the anonymity of ballots. A mix network, or mixnet, is a network of servers called *mix servers*, where each server takes in a set of data items, mixes

(permutes) the set, and passes the resulting set to the next server. Every mix server is further required to prove that its resulting mix is genuine using interactive proof.

Assume that the ballots have all been collected in a set $\mathcal{B}$. The mixnet proceeds as follows for each mix server:

1. Re-encrypt each ballot $B_i \in \mathcal{B}$ using the Elgamal re-encryption scheme to produce ballot B'_i.
2. Shuffle (permute) the set of re-encrypted ballots $\{B'_1, B'_2, \cdots, B'_n\}$ to form a ballot set $\mathcal{B}'$.
3. Construct m additional random Ballot sets $\mathcal{B}_1, \mathcal{B}_2, \cdots \mathcal{B}_m$, processed in the same way as $\mathcal{B}'$. Note that in Step 1, $\mathcal{B}_i$ starts as $\mathcal{B}$.
4. Interact with a user to get a random sequence of length m of challenge bits: $c_1 c_2 c_3 \cdots c_m$.
5. For all $1 \leq i \leq m$, where $c_i = 0$, show that the ballot set $\mathcal{B}_i$ is equivalent to the ballot set $\mathcal{B}$. To do so, it suffices to reveal the permutation and the re-encryption random values for every ballot. Note that the ballot itself may be leaked, but *not* what the ballot contains!
6. For all $1 \leq i \leq m$, where $c_i = 1$, show the ballot set $\mathcal{B}_i$ is equivalent to the ballot set $\mathcal{B}'$ by
   (a) computing the pairwise difference of the re-encryption data used to produce $\mathcal{B}'$ and $\mathcal{B}_i$; and
   (b) computing the composition of the inverse permutation used to produce $\mathcal{B}_i$ and the permutation used to produce $\mathcal{B}'$.

   Both compositions are then furnished to the verifier.

The transcripts of the proofs are posted to the bulletin board.

5.8.4.3 Tally

In the tally phase, the ballots are decrypted and tallied. This tally as well as proof of the decryption is posted to the bulletin board. To prove that the threshold encrypted ballots are decrypted correctly, an interactive proof strategy is used. In particular, this protocol is called the proof of decryption protocol.

The proof of decryption protocol used in the Helios voting protocol is the following Chaum-Pedersen protocol, which proves that for a given Elgamal ciphertext (α, β) and the plaintext claim M, we have

$$\log_g(g^x) = \log_\alpha \left(\frac{\beta}{M} \right).$$

The following protocol between Peggy and Victor can be used to demonstrate the veracity of that statement.

1. *Selection*: Peggy selects a $w \in Z_p^*$ and sends $A = g^w$ and $B = \alpha^w$.
2. *Challenge*: Victor challenges Peggy with a random number $c \in Z_p^*$.
3. *Response*: Peggy responds with $t = w + xc$.
4. *Verification*: Victor check that $g^t = Ag^{xc}$ and $\alpha^t = B\left(\frac{\beta}{M} \right)^c$.

   If it does match, then Victor agrees that the decryption was correct. Otherwise, Victor does not agree on the decryption.

To see why this proof works, it suffices to note that for Peggy to construct t, she would either be the owner of the private key x or have to solve the discrete log problem to compute x from the public key g^x. Because discrete log is intractable according to common beliefs, we conclude that Peggy is the owner of the private key, and thus her decryption is correct. The proof is then posted to the bulletin board.

5.9   Closing Remarks

How to construct practical security protocols to protect network communications is a critical issue in network security. Security flaws in security protocols are often caused by improper handling of a cryptographic algorithm, not by the algorithm itself. Improper key management and careless implementations of security protocols are common reasons that security loopholes exist. Whether a security protocol will do exactly what it is intended to do can only be tested through practice. This is a process of discovering defects, patching loopholes, and devising new security protocols. The security protocols we described in this chapter are for the three top layers of the Internet (i.e., above the data-link layer). We introduce wireless network security protocols at the data-link layer in the following chapter.

5.10   Exercises

5.10.1   Discussions

5.1. In your opinion, should the construction of PKI facilities be under close control of the government?

5.2. Do you think that the development of global economy will demand the establishment of a global PKI authority? How do you imagine such a system will work?

5.3. Provide an example attack that IPsec cannot protect against.

5.4. From your experience, if a Website uses SSL to transmit your credit card information when you do online shopping, how much do you trust that it is secure in the scale from 1 to 10 with 10 being 100% secure. What security problems would you be concerned about?

5.5. From your experience, why do you think one must use a secure remote login protocol. Even if SSH is used, how much do you trust that it is secure in the scale from 1 to 10 with 10 being 100% secure. What security problems would you be concerned about?

5.6. Which network security protocols do you use more often than the others? What security protocols do you wish to have that are not currently available?

5.10.2   Homework

5.1. Placing cryptographic algorithms in the transport layer has a different security impact than placing them in the network layer. Explain the differences.

5.2. Placing cryptographic algorithms in the application layer has a different security impact than placing them in the data-link layer. Explain the differences.

5.3. Suppose that cryptographic algorithms are deployed at the IP layer. Thus, when a TCP packet is passed down to the IP layer, its payload or the entire packet is encrypted or authenticated. Does it make sense to encrypt or authenticate only the header? Justify your answer.

5.4. Suppose that a TCP header is encrypted at the TCP layer. Can this TCP packet be delivered to the destination without using some type of TCP gateway? When does the encrypted TCP header need to be deciphered? Justify your answers.

5.5. Explain why one may want to encrypt the entire frame at the data-link layer. What can one get from performing traffic analysis on such frames?

5.6. Explain why one may want to authenticate the entire frame at the data-link layer.

5.7. If only the payloads of frames are encrypted (i.e., frame headers and trailers are not encrypted), what can one get from performing traffic analysis on such frames?

5.8. Users of Microsoft Windows XP may look at the public-key certificates and the list of revoked certificates stored in the system following these instructions: Click Start then Run. Enter mmc and click OK. In the popup window titled "Console1" click in succession File, Add/Remove Snap-in, Add, Certificate, Add, My user account (skip it if it is already selected), Finish, Close, and OK. Click the "+" sign at the left-hand side of the Certificate – Current User window and answer the following questions.
  (a) What does each item mean?
  (b) In which items do revoked certificates appear? Which certificates have been revoked?

5.9. If you have Adobe Acrobat version 6.0 or a later version installed on your computer, you can create public-key certificates and use public keys to authenticate documents. Suppose that the document you want to sign is a PDF file. Open this file using Acrobat. Click Advanced then Manage Digital IDs. Point the mouse to My Digital ID Files, then click in succession Select My Digital ID File, New Digital ID File, and Continue. Fill in the blank box with relevant information. Click Create. A window titled The New Self-Sign Digital ID File will pop up. This is your public-key certificate. Click Save.

   Click in succession Advanced, Manage Digital IDs, My Digital ID Files, My Digital ID File Settings, and Export. Select Save the data to a file. This is your public key. Select a directory and a file name, then click in succession Save, OK, and Close.

   Click in succession File, Save as Certified Document, and OK. Select Disallow any changes to the document, then click Next. Select Do not show Certification on document, then click in succession Next, Add Digital ID, Create a self-signed digital ID, and Continue. Select Add as a "Windows Trusted Root" Digital ID, then click Create and OK. Select the public-key certificate you just created, and

click OK and View digital ID. You will see a certificate similar to Table 5.1. List your certificate and explain each item.

Finally, click Close and Sign and Save as. Enter the file name and click OK.

5.10. Describe the differences between IPsec transport mode and the IPsec tunnel mode.

5.11. Download the latest version of OpenSSL (available for both Windows and Linux) from `http://www.openssl.org` and use it to create a self-signed certificate for use as a CA and then create a user certificate signed by the CA.

Can you load the CA certificate into the Windows and use the client certificate to send S/MIME emails?

5.12. The early versions of OpenSSL suffered from the Heartbleed bug attack that allows anyone on the Internet to read the memory of the systems protected by the early versions of OpenSSL, including the memory of the secret keys. Conduct a research on Heartbleed bug and write a report of up to 4000 words on your findings.

5.13. Use the "Heartbleed Bug checker" at `https://filippo.io/Heartbleed/` to check if your OpenSSL server is still vulnerable to the Heartbleed bug.

5.14. Use the "LastPass' SSL date checker" at `https://lastpass.com/heart bleed/` to check if your OpenSSL server has updated its SSL certificate. If the update was done after April 4, 2014, then it is likely that the system administrator of your OpenSSL server has patched the Heartbleed bug vulnerability.

5.15. The IPsec transport mode and the IPsec tunnel mode can be mixed together. Describe the pros and cons of different combinations of these two modes.

5.16. Describe using a diagram the working of the sliding window in AH.

5.17. Explain the meanings in the basic interactions in Oakley KDP.

5.18. The following is an ISAKMP payload exchange example:
1. I → R: SA, proposal, transfer, nonce
2. R → I: SA, proposal, transfer, nonce
3. I → R: key-exchange, nonce
4. R → I: key-exchange, nonce
5. I → R: identification (of I), signature
6. R → I: identification (of R), signature

Explain what this exchange is trying to accomplish.

5.19. IPsec has been implemented with IPv4 and IPv6. However, it is easier to implement IPsec with IPv6. Explain why this is the case.

**5.20. Can you design a network protocol so that IPsec can be implemented in a more natural way than IPv6? Justify your answer.

5.21. The later versions of Linux includes IPsec. If you are running Linux, install IPsec on your Linux machine.

5.22. Translate the following binary strings to Base64 strings:
  (a) 10010010
  (b) 1010110110010010
  (c) 101100100110110110100011
  (e) 010011011001001001011100010110010

5.23. Because binary data is a full 8-bit per byte, it is possible to convert a binary string to a text string using hexadecimal digits. That is, one can represent each byte using two hexadecimal digits, where each hexadecimal digit is represented by an ASCII code. Explain why this method is not economical.

5.24. According to your experience of online shopping, describe how SSL is executed.

5.25. Draw a flow diagram describing the SSL handshake protocol.

5.26. Describe how the receiving site of SSL executes the SSL record protocol.

5.27. Use `Wireshark` to identify packets used for executing the SSL handshake protocol. (Note: you should finish Exercise 1.5 first.) If your bank supports online banking, use `Wireshark` to obtain your login packets and check whether they are encrypted.

5.28. In this hands-on drill, you will install PGP on a PC and use PGP with Outlook Express to send secure email messages. Firstly, download PGP Freeware 8.0 (or the latest version) from
`http://www.pgpi.org/products/pqp/versions/freeware`
for your operating system. For example, for Windows XP, go to
`http://www.pgpi.org/products/pqp/versions/freeware`
`/winxp`.
   Click the file you downloaded and install it. Select `I am a New User`, and enter your name and organization. Select PGPMail for Microsoft Outlook Express and press the `Next` button. After PGP is installed, do the followings:
  (a) Generate a public/private key pair: click the `PGPtray` icon and select `PGP-keys`. Then select `keys` and `New Key`. Now you will enter the PGP key generation wizard. Click `Next`. Enter your name and your email address. The key pair you will generate will be associated with the name and the email address you enter here. Select algorithm (e.g., RSA) and key length (select a number between 1024 and 4096). Then enter an expiration date. Click the `Next` button and enter a pass phrase of your choice (you need to remember this pass phrase). Click `Finish`. Now a public key should be shown on your screen.
  (b) Distribute your public key: send your public key to your correspondents (e.g., your classmates). You may simply drag the key to the body of a mail message and send it.
  (c) Obtain your correspondent's public key: ask your correspondent to mail you his key and put it in your key ring. To do so, click `PGPtray` (the PGP executable), select `Current Window` and `Decrypt and Verify`. You may

also obtain your correspondent's public key from the key server (if your corre-
spondent has sent his public key to it). Validate your correspondent's public key
and sign the key. To sign a key, select `Keys` from `PGPtray`, then click `New`
`Keys` and `Sign`. Now you are ready to use your correspondent's public key.

(d) Send an authenticated email message or an encrypted email message to your
correspondent: at the top right "new message" window of Outlook Express, you
will see two buttons: `Encrypt Message Before Sending` and `Sign`
`Message Before Sending`. Choose accordingly (e.g., choose either one
of them or both). Then send the message.

(e) Receive an authenticated email message or an encrypted email message: Simply
select the `decrypt and verify` option.

5.29. Microsoft Office Outlook and Outlook Express both support S/MIME. To set it
up, one must first install a digital ID (i.e., a digital certificate) following these
steps: Open Office Outlook or Outlook Express, then click in succession Tools,
Options, Security, and Get a Digital ID. Select your country or region. Click
VeriSign Web Site then Click here (for 60-day free trial). Enter your email
address, and click in succession Accept and OK. If Get a Digital ID does not
work, go to the VeriSign Website directly using the following URL: `https:`
`//digitalid.verisign.com/cgi-bin/OEenroll.exe?name`
`=& email=`, and obtain a free-trial digital ID from there. After you receive your
digital ID, click in succession Continue, Install, and Yes.

(a) Follow these steps to sign a message: Open Office Outlook or Outlook Express.
Click in succession Tools, Options, Security, Add digital signature to out-
going message, and OK. Send a message to yourself. Click Send then OK.
Describe how to authenticate a sender's identity.

(b) Follow these steps to encrypt a message: Open Office Outlook or Outlook
Express. Click in succession Message, New Message, and Encrypt. Send a
message to yourself and explain how to decrypt a message.

5.30. Explain the advantages of separating AS and TGS into two entities in Kerberos.

5.31. Explain why $\text{Auth}_{\text{U,TGS}}$ is added in Phase 3 of single-realm Kerberos. What attacks
can it help protect?

5.32. Provide a dissection of multiple-realm Kerberos protocol steps.

5.33. Draw a flow diagram to describe single-realm Kerberos dialogs.

5.34. Draw a flow diagram to describe multiple-realm Kerberos dialogs.

5.35. You will need to use two networked computers running Microsoft Windows to do
this exercise. Download from `http://www.ssh.com` the evaluation version of
the latest SSH client program and SSH server program. Install the client program on
one machine, install the server program on the other machine, and run SFTP.

*5.36. SSH has two different versions. The original version is called SSH1, and the revised
version is called SSH2. These are two different protocols. Search the literature and

write a short paper of up to 4000 words describing these two versions of SSH. Explain why SSH2 is more secure than SSH1.

5.37. Visit the Helios voting protocol Webpage: `https://vote.heliosvoting .org/`, and describe potential security flaws in deploying the voting protocol for a student government election.

5.38. In the electronic voting protocol described in Problem 5.37, is it really necessary for Alice to verify the voting machine correctly encrypted her ballot?

*5.39. Describe an interactive proof protocol between Peggy and Victor where Peggy, knowing the three coloring of a graph, can prove this fact to Victor. As in the text, Peggy does not want to actually reveal the three coloring to Victor.

6

Wireless Network Security

Wireless computer networks are playing a major role in modern communications. Laptop computers, cell phones, and embedded systems in common appliances and automobiles may be connected to form an ad hoc network or to a fixed network infrastructure such as the Internet through wireless access points. Wireless access points have been installed in office buildings, homes, airports, hotels, highway service stations, and other facilities, providing people with unprecedented conveniences and flexibilities to exchange information and enjoy online entertainment. People today, for example, can connect their laptop computers to the Internet while sitting in their own backyards or while waiting for flights in the airports. Wireless computer networks have started a new revolution in the information industry.

This chapter introduces the IEEE 802.11 wireless network standard for wireless local area networks (WLANs), wireless network security vulnerabilities, and common wireless security protocols. The latter includes the Wired Equivalent Privacy (WEP) protocol, the Wi-Fi Protected Access (WPA) protocol, WPA2, the IEEE 802.11i protocol, and the IEEE 802.1X authentication protocol. This chapter also introduces the Bluetooth protocol and the ZigBee protocol for wireless personal area networks (WPANs) and their security mechanisms. Finally, it introduces several security issues in wireless mesh networks (WMNs).

6.1 Wireless Communications and 802.11 WLAN Standards

Wireless networks transmit data in the air via radio waves of various frequencies. Transmitting radio waves in the open air, however, makes it easy for any person on the street to intercept wireless data, to connect his computing devices to a nearby wireless network, or to inject new packets to existing wireless networks. To do these, what the attacker needs is a radio transmitter and receiver with the same radio frequency of the underlying wireless network. The attacker may also jam a particular wireless channel using a jamming device. Channel jamming may also be unintentional, for common wireless networks are operated on the same frequency.

Media access in wireless networks is fundamentally different from the media access mechanisms in wired networks, where one has to hook up a computing device to a network cable for transmitting and receiving data, and the cables are physically protected by walls, ceilings,

doors, pipes, and other forms of physical structures. Thus, how to provide wired equivalent media access in wireless networks becomes a unique security issue.

Providing physical-layer security protections for wireless networks is difficult, for anyone may disturb or jam a selected radio frequency. Jamming a radio frequency is in a way equivalent to cutting a network cable in a wired network. Attacks of this kind are easy to implement by the attacker but difficult to prevent by the defender. To counter such attacks, the spread spectrum technology has been developed to make radio signals more difficult to detect and more difficult to jam.

Most wireless security protocols deal with media access at the data-link layer, including implementations of encryption algorithms, authentication algorithms, and integrity-check algorithms. This approach provides network access with WEP. With security mechanisms implemented at the data-link layer, higher-layer protocols and applications can be used without any modification.

The IEEE 802.11 standards specify a suite of protocols and specifications for WLAN communications at the physical layer with appropriate security protections at the data-link layer.

6.1.1   WLAN Architecture

A WLAN may attach itself to a wired infrastructure. Such WLANs are referred to as infrastructure WLANs. A WLAN may also be formed without attaching itself to a fixed infrastructure. Such WLANs are referred to as ad hoc WLANs (or peer-to-peer WLANs).
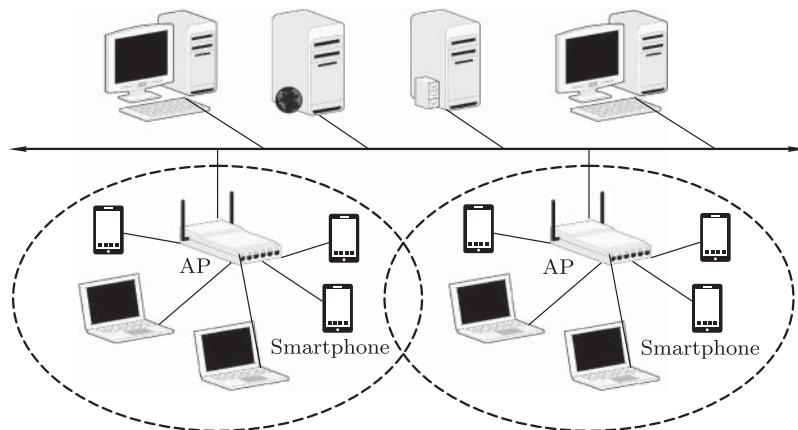
6.1.1.1   Infrastructure WLANs

An infrastructure WLAN consists of one or more wireless access points (WAP) and a number of wireless-enabled computing devices (e.g., laptop computers, tablets, and smartphones). WAP is often referred to as AP in short. A wireless-enabled computing device is referred to as a mobile station (STA) or a wireless node (WN). An AP is equipped with a radio transmitter and receiver, an antenna, and a standard port for wired connection. At one end of an AP is a wired link connected to a wired LAN. At the other end of an AP is a radio transmitter and receiver used to establish radio connections between the AP and an STA. A single AP may establish wireless connections with multiple STAs at the same time through time division multiplexing access. Thus, an AP in a WLAN is similar to a switch in a LAN, and an infrastructure WLAN is similar to a traditional star network. Figure 6.1 shows a schematic of infrastructure WLANs.

The AP in an infrastructure WLAN is fixed, which serves as the center of the network. An STA will need to select an AP within its communication range and connect itself to the AP to become a new member of the WLAN.

Each STA in the 802.11 standards is identified by a 48-bit MAC address. APs can be configured so that they can be accessed only by STAs from a given list of MAC addresses. This is called *MAC-address filtering*.

Each AP is associated with a Service Set Identifier (SSID), which serves as the name of the WLAN. Each AP announces regularly its SSID and other information needed for an STA to establish a connection with it. This process is known as *beaconing*. An STA waits for a beacon frame from an AP and joins a WLAN by sending a request frame to the corresponding AP with the AP's SSID it receives. This process is known as *scanning*.

Figure 6.1   A schematic of infrastructure wireless local area networks

6.1.1.2   Wi-Fi Networks

WLANs that meet the 802.11 standards may be certified as Wi-Fi networks by the Wi-Fi Alliance. Wi-Fi stands for Wireless Fidelity. The Wi-Fi Alliance, established in 1999, is a nonprofit organization for promoting the worldwide adoption of the IEEE 802.11 standards for high-speed WLANs.

6.1.1.3   Wi-Fi Hotspots
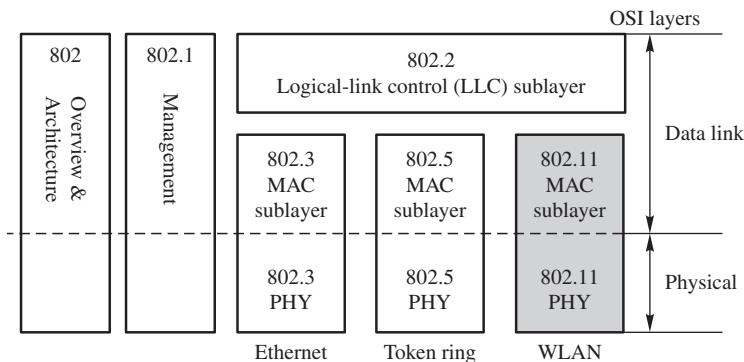
The geographic region covered by a Wi-Fi AP is often referred to as a *Wi-Fi hotspot* . The AP is connected to the Internet, allowing STAs in the hotspot to connect to the Internet through the AP.

6.1.1.4   Ad Hoc WLANs

An ad hoc WLAN is formed without using any fixed wired infrastructure, where an STA may communicate with another STA directly within communication range. If the destination STA is not within the communication range of the source STA, the source STA will use intermediate STAs to relay data to the destination STA. Thus, an ad hoc WLAN is similar to a wired peer-to-peer network.

6.1.2   802.11 Essentials

802.11 is the wireless counterpart of 802.3 and 802.5 in the IEEE 802 protocol family, where 802.3 is the IEEE standard for Ethernet LANs and 802.5 is the IEEE standard for token ring LANs. The data-link layer consists of the logical-link control (LLC) sublayer and the media-access (MAC) sublayer. 802.11 specifies communications and security mechanisms for WLAN at the MAC sublayer and at the physical layer. Figure 6.2 shows the schematic of the IEEE 802 family.

Figure 6.2 A schematic of the IEEE 802 family

The MAC sublayer of 802.11 uses the carrier sense multiple access with collision avoidance (CSMA/CA) scheme. 802.11 consists of a series of protocols. The first protocol was named 802.11, and the subsequent protocols were named 802.11 followed by a lower-case letter, which ranged from 802.11a to 802.11u. Of these subprotocols, 802.11a, 802.11b, 802.11g, and 802.11i have been used widely, where 802.11b supports a data rate of up to 11 Mbps with a transmission range of about 35 m indoor and about 110 m outdoor, while 802.11g supports a data rate of up to 54 Mbps with the same transmission range as 802.11b. WEP is defined in 802.11b, and WPA2 is defined on the basis of 802.11i.

802.11b and 802.11g operate on the same radio frequency of 2.4 GHz, while 802.11a operates on frequency of 5 GHz. As most cordless phones operate on the frequency of 2.4 GHz, 802.11a will not be interfered with by cordless phones, microwaves, or Bluetooth devices. In addition, 802.11a channels do not overlap with 802.11b and 802.11g channels. In the United States, 802.11b and 802.11g consist of 11 useable channels, where channels 12–14 are reserved for emergency responders.

Built on previous 802.11 standards, 802.11n supports devices with multiple-input multiple-output (MIMO) capacity. These devices use multiple transmitter and receiver antennas to improve system performance.

In ad hoc WLANs, network bandwidths are often reduced by a factor of 2, for receiving nodes need to forward data toward the destination.

6.1.3 Wireless Security Vulnerabilities

Wireless technologies have the following weaknesses:

1. Wireless communications could be easily sniffed.
2. Radio signals could be easily disturbed or injected to the network.
3. Wireless hand-held computing devices and embedded systems may not have sufficient computing resources or power supply to carry out complex computations that require fast CPUs and large memory space.

These weaknesses make wireless communications vulnerable to eavesdropping attacks, denial-of-service attacks, message-replay attacks, STA-spoofing attacks, and AP-spoofing attacks.

   STAs and APs in early wireless network protocols were identified only by MAC addresses. MAC addresses were transmitted in plaintext, and so impersonating STAs and APs was straightforward. The attacker may impersonate a legitimate user and deliberately inject certain malicious packets to the network for the purpose of achieving false authentication or breaking existing wireless connections between legitimate STAs and legitimate APs.

6.2   Wired Equivalent Privacy

The WEP protocol, published in 1999, is the security component at the data-link layer of 802.11b.

   WEP requires that all STAs and APs in the same WLAN share the same preset secret key K, referred to as the *WEP key*. A WEP key may be 40-bit or 104-bit long. Some WEP products may even support 232-bit WEP keys. WEP allows each WLAN device to share more than one WEP key. WEP keys are identified using a 1-byte key ID, denoted by *keyID*.

   WEP does not specify how to generate or distribute secret keys. Thus, secret keys are often selected by the system administrator and distributed using land-line communications or other methods. In general, WEP keys are not changed once they are installed.

6.2.1   Device Authentication and Access Control

WEP uses a *challenge-response* authentication scheme to authenticate STAs. That is, for an STA to get access to an AP, the STA must authenticate itself to the AP as follows:

1. *Request*: The STA transmits a request for connection to the AP in the WLAN.
2. *Challenge*: The AP generates 128-bit pseudorandom string

$$cha = a_1 a_2 \cdots a_{16},$$

   where each a_i is an 8-bit string for $1 \leq i \leq 16$, and sends *cha* to the STA.
3. *Response*: The STA generates a 24-bit initialization vector V and encrypts *cha* using RC4 with key $V \parallel K$. That is, the STA applies RC4 on $V \parallel K$ to generate a subkey stream $k_1, k_2, \cdots, k_{16}$, where each k_i is an 8-bit string. It then computes $r_i = a_i \oplus k_i$ for $i = 1, 2, \cdots, 16$. The STA sends

$$res = V \parallel r_1 r_2 \cdots r_{16}$$

   to the AP.
4. *Verification*: The AP applies RC4 on $V \parallel K$ to generate the same subkey stream $k_1, k_2, \cdots, k_{16}$, computes $a_i' = r_i \oplus k_i$, and verifies whether $a_i' = a_i$, where $i = 1, 2, \cdots, 16$. If yes, the STA is authenticated and the connection request is granted. If not, the STA is denied access to the AP.

6.2.2   Data Integrity Check

Let M be the packet passed down from the network layer to the data-link layer for transmission. WEP at the LLC sublayer calculates a 32-bit Cyclic Redundancy Check (CRC-32) value of M as the integrity check value.

6.2.2.1 Cyclic Redundancy Check

CRC is a simple binary polynomial division procedure that takes a binary string as input and outputs a binary string of a fixed length as error-detection code. WEP uses CRC to check data integrity. In particular, let M be an n-bit binary string. Let P be a binary polynomial of degree k appropriately chosen, represented as a $(k + 1)$-bit binary string. To obtain a k-bit CRC value, treat $M0^k$ as a binary polynomial of degree at most $n + k - 1$. Divide $M0^k$ by P to obtain a k-bit remainder R, which is the CRC value, denoted by $\mathrm{CRC}_k(M)$. IEEE 802.3 selects

$$P = 100000100110000010001110110110111$$

to be the CRC-32 polynomial, that is,

$$P(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1.$$

The new polynomial $M \parallel \mathrm{CRC}_k(M)$ is divisible by P. This can be proved as follows. Firstly, write M as a polynomial $M(x)$ of degree at most $n - 1$. Then $M0^k$ represents the polynomial $M(x)x^k$. We have

$$M(x)x^k \bmod P(x) = R(x),$$

where $R = \mathrm{CRC}_k(M)$. Thus, because adding binary coefficients is the same as exclusive-OR, we have

$$(M(x)x^k + R(x)) \bmod P(x)$$
$$= (R(x) + R(x)) \bmod P(x)$$
$$= 0 \bmod P(x)$$
$$= 0.$$

Thus, if $M \parallel \mathrm{CRC}_k(M)$ is not divisible by P, it implies that M has been modified.

To calculate a CRC value, let $T = M$, align P to T at the leftmost bit that is 1, and perform XOR on P and the corresponding bits in T. Let T denote the new string. Repeat the same procedure until P goes out of the right-hand side of T. The rightmost k bits are the remainder, which is used as the integrity check value (ICV).
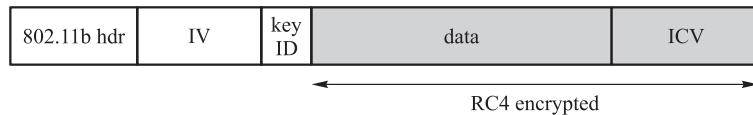
We use an example to demonstrate the CRC procedure. For simplicity, let $n = 8$ and $k = 4$. The standard CRC_4 polynomial is $x^4 + x + 1$. That is, $P = 10011$. Let $M = 11001010$, then $\mathrm{CRC}_4(M) = 0100$ (see Figure 6.3).

6.2.3 LLC Frame Encryption

Let M be an 802.11b LLC frame to be transmitted by the sender, which includes the LLC-frame header and the packet passed down from the upper layer. An LLC frame is also referred to as MAC Service Data Unit (MSDU).

WEP calculates the $\mathrm{CRC}_{32}(M)$ and encrypts $M \parallel \mathrm{CRC}_{32}(M)$ at the MAC sublayer using the RC4 stream cipher. In particular,

	1	1	0	0	1	0	1	0	0	0	0	0
$\oplus$	1	0	0	1	1							
		1	0	1	0	0	1	0	0	0	0	0
$\oplus$		1	0	0	1	1						
			1	1	1	1	0	0	0	0	0	0
$\oplus$			1	0	0	1	1					
				1	1	0	1	0	0	0	0	0
$\oplus$				1	0	0	1	1				
					1	0	0	1	0	0	0	0
$\oplus$					1	0	0	1	1			
								0	1	0	0	

Figure 6.3   A sample CRC_4 calculation

802.11b hdr	IV	key ID	data	ICV

RC4 encrypted

Figure 6.4   802.11b MAC sublayer frame layout

1. Let
$$M \parallel \mathrm{CRC}_{32}(M) = m_1 m_2 \cdots m_\ell,$$
where each m_i is an 8-bit binary string.

2. The sender's MAC sublayer generates a 24-bit initialization string V, and uses RC4 on input $V \parallel K$ to generate a sequence of 8-bit subkeys $k_1, k_2, \cdots, k_\ell$. Let
$$c_i = m_i \oplus k_i.$$

3. The sender's MAC sublayer adds a header to the payload
$$V \parallel keyID \parallel c_1 c_2 \cdots c_\ell$$
and transmits it in the air for the receiver. Figure 6.4 shows the layout of an 802.11b MAC sublayer frame.

For convenience, we denote this encryption algorithm by
$$C = (M \parallel \mathrm{CRC}_{32}(M)) \oplus \mathrm{RC4}(V \parallel K).$$

The initialization string V is used to produce a different subkey stream for a different LLC frame. The initialization string is typically referred to as initialization vector. Thus, the key $V \parallel K$ is also referred to as a *per-frame key*. Note that the initialization vector V is transmitted in plaintext, which is used by the receiver to calculate the same subkey sequence $k_1, k_2, \cdots, k_\ell$ for decrypting c_i to get m_i. Removing the rightmost 32 bits from $m_1, m_2, \cdots, m_\ell$, the receiver gets M.

WEP encryption is intended to provide WEP protection. WEP does not provide confidentiality against legitimate STAs in the network. Thus, application-layer encryptions are needed to provide confidentiality for user data against STAs.

6.2.4 Security Flaws of WEP

WEP, although used widely, was a hastily designed security protocol. It contains serious security flaws in device authentication, frame integrity check, frame encryption, and access control.

6.2.4.1 Authentication Flaws

The challenge-response authentication scheme used in WEP is a simple exclusive-OR scheme, which is vulnerable to the known-plaintext attack as described in Section 2.1.2. In particular, the attacker may use a sniffer to intercept a challenge–response pair (cha, res) between the AP and a legitimate STA, where

$$cha = a_1 a_2 \cdots a_{16},$$
$$res = V \parallel r_1 r_2 \cdots r_{16},$$
$$r_i = a_i \oplus k_i,$$
$$i = 1, 2, \cdots, 16.$$

Thus, Malice, the attacker, can compute $k_i = a_i \oplus r_i$ for $i = 1, 2, \cdots, 16$. This allows Malice to authenticate her device to the AP as follows:

1. Send a request to the AP.
2. Wait for the challenge string cha' from the AP.
3. Use the previously computed key streams $k_1, k_2, \cdots, k_{16}$ to XOR the challenge string cha' to get a response string res'.
4. Send the previously captured initialization vector V and the response string res' to the AP.

The AP applies RC4 to $V \parallel K$ to generate the same key stream $k_1, k_2, \cdots, k_{16}$, verifies that $k_1 k_2 \cdots k_{16} \oplus res' = cha'$, and authenticates Malice's device.

6.2.4.2 Integrity Check Flaws

CRC has been widely used at the data-link layer to detect transmission errors in data frames, but it is a poor choice for checking frame integrity. This is because CRC has the following two weaknesses:

1. *Linear property*: For any two strings x and y,

$$\mathrm{CRC}(x \oplus y) = \mathrm{CRC}(x) \oplus \mathrm{CRC}(y). \tag{6.1}$$

   The proof of this property is straightforward (see Exercise 6.7).
2. *No secret key involved*: CRC is generated without using secret keys.

The first weakness allows the attacker to modify a message so that the CRC of the modified message is the same as that of the original message. The second weakness allows the attacker to inject new messages.

Message Tampering

Suppose that M is the original packet that Alice wants to send to Bob. According to the encryption scheme, Alice's STA sends the following string C to Bob's STA:

$$C = (M \parallel \mathrm{CRC}_{32}(M)) \oplus \mathrm{RC4}(V \parallel K).$$

Suppose that Malice intercepts C. Then Malice can modify C in anyway she wants to, and the modification cannot be detected by integrity check. This can be done as follows. Let Γ be an arbitrary frame. Malice modifies C by

$$C' = (\Gamma \parallel \mathrm{CRC}_{32}(\Gamma)) \oplus C,$$

and sends it to Bob. It follows from Equality 6.1 that

$$
\begin{aligned}
C' &= (\Gamma \parallel \mathrm{CRC}_{32}(\Gamma)) \oplus C \\
&= [(\Gamma \parallel \mathrm{CRC}_{32}(\Gamma)) \oplus (M \parallel \mathrm{CRC}_{32}(M))] \oplus \mathrm{RC4}(V \parallel K) \\
&= [(\Gamma \oplus M) \parallel (\mathrm{CRC}_{32}(\Gamma) \oplus \mathrm{CRC}_{32}(M))] \oplus \mathrm{RC4}(V \parallel K) \\
&= [(\Gamma \oplus M) \parallel (\mathrm{CRC}_{32}(\Gamma \oplus M))] \oplus \mathrm{RC4}(V \parallel K) \\
&= (M' \parallel \mathrm{CRC}_{32}(M')) \oplus \mathrm{RC4}(V \parallel K),
\end{aligned}
$$

where $M' = \Gamma \oplus M$. Thus, Bob will receive a new message M' with the correct integrity check value of $\mathrm{CRC}_{32}(M')$.

Message Injections

RC4 is vulnerable to the known-plaintext attack. That is, if a legitimate plaintext–ciphertext pair (M, C) is known, then performing $M \oplus C$ will yield the subkey stream used to encrypt M. As the initialization string V is transmitted in plaintext and CRC is generated without using any secret key, the attacker can inject a message and have it authenticated if the same initialization vector can be reused. What he needs to do is to generate a message Θ he wants to inject to the network, compute $\mathrm{CRC}_{32}(\Theta)$, encrypt $\mathrm{CRC}_{32}(\Theta) \parallel \Theta$ using the subkey stream he obtains from a known-plaintext attack, and inject

$$V \parallel (\Theta \parallel \mathrm{CRC}_{32}(\Theta)) \oplus \mathrm{RC}_4(V \parallel K)$$

to the network.

Fragmentation Attacks

The fragmentation attack takes advantage of the unique structure in the 802.11b LLC frame header to inject messages to the network. In particular, the first eight bytes in the header of any LLC frame have the following fixed values (represented in hexadecimal):

$$\text{AA AA 03 00 00 00 08 00}$$

if it is an IP packet, or

$$\text{AA AA 03 00 00 00 08 06}$$

if it is an ARP packet, where each ARP packet has a fixed length of 36 bytes. Thus, it is straightforward to distinguish an ARP packet from an IP packet.

These eight bytes are the first to be encrypted. Thus, the attacker may obtain the first eight subkeys

$$k_1, k_2, \cdots, k_8$$

from any intercepted MAC frame by performing a simple XOR operation. Let V be the initialization string.

The 802.11b MAC sublayer may divide an LLC frame up to 16 segments. Thus, the attacker may inject a 64-byte LLC frame by segmenting it into sixteen 4-byte fragments, use V and the subkey stream $k_1, k_2, \cdots, k_8$ to encrypt each 4-byte fragment and its 4-byte integrity check value (ICV), put it in a MAC frame, and inject it to the network.

6.2.4.3 Confidentiality Flaws

Recall that RC4 (see Section 2.7.1) is a stream cipher that first expands the encryption key to generate an initial permutation of a 256-byte array, then keeps swapping elements in the array to generate subkey streams.

Repeating Initialization Vectors

In Section 2.7.2, we have discussed that the RC4 stream cipher is vulnerable to the related-plaintext attack if a subkey stream is reused. To avoid regenerating the same subkey stream, WEP generates independently at random a 24-bit initialization string V for each frame to form a frame key $V \parallel K$, which allows RC4 to generate a different subkey stream.

However, as there are only $2^{24} = 16,777,216$ different initialization vectors, it follows from the birthday paradox (see Section 4.4.1) that after $1.25\sqrt{2^{24}} = 5120$ frames, the chance that there is at least one initialization vector appearing more than once is greater than 50%.

Moreover, as 802.11b has a bandwidth of 11 Mbps, it is easy to have more than 2^{24} frames transmitted in a busy network in a short period of time, during which some initialization vectors will be repeated.

RC4 Weak Keys

In Section 2.7.2, we have discussed that knowing the initial permutation is equivalent to breaking RC4 encryption. Even if the initial permutation is partially revealed, the attacker may still be able to obtain information about the encryption key. Thus, it is important to select a suitable encryption key to produce a secure initial permutation. But selecting a suitable encryption key may be difficult, for many binary strings form weak encryption keys.

In WEP, the 24 bits of the initialization vector in the per-frame encryption key are open to the public. The Fluhrer–Mantin–Shamir (FMS) attack described in Section 2.7.2 collects weak initialization vectors used with the same WEP key (i.e., these vectors are almost 2^q-conserving for $1 \leq q \leq 8$) and uses them to deduce the WEP key.

A number of WEP cracking software tools have since been developed on the basis of the FMS attack. These tools include WEPCrack, WEPLab, WEPWedgie, WEPAttack, AirSnort, AirJack, and AirCrack. Appendix E describes a WEPCrack experiment to crack WEP keys.

6.3 Wi-Fi Protected Access

The WPA protocol, published in 2003 by the Wi-Fi Alliance, was designed on the basis of an early version (draft 3) of the IEEE 802.11i standard. WPA has three major objectives. The first objective is to correct all the security problems found in WEP. The second objective is to make the existing hardware that supports WEP also support WPA. The third objective is to ensure that WPA is compatible with the 802.11i standard to be announced.

In particular, WPA uses a specifically designed integrity check algorithm, called the *Michael* algorithm, to produce Message Integrity Code (MIC) for preventing forgeries. Although it still uses RC4 to encrypt LLC frames, WPA uses a new key structure to generate per-frame keys that prevents message replays and de-correlates public initialization vectors from weak RC4 keys. The new key structure uses a new initialization vector generation scheme and a key mixing algorithm. All of these mechanisms are specified in the *Temporal Key Integrity Protocol* (TKIP).
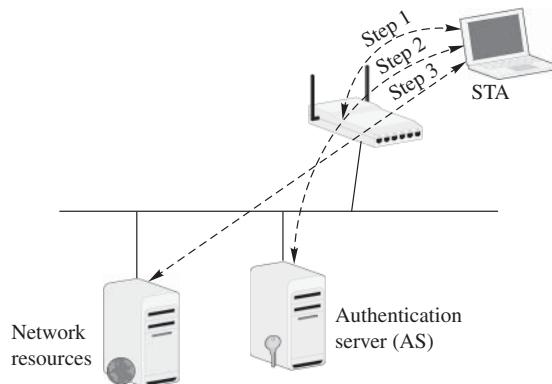
6.3.1 Device Authentication and Access Controls

WPA supports two methods for authenticating STAs. The first method uses a preset secret key in the same way as WEP. This method is intended to secure home and small-office WLANs. Thus, this method is referred to as *Home-and-Small-Office WPA*. The second method is more sophisticated, which is intended to secure corporate WLANs. Thus, it is also referred to as *Enterprise WPA*. It uses an authentication server (AS) and a preshared secret between the AS and an STA, where different STAs share different preshared secrets with the AS. Preshared secrets are often presented in the form of passwords.

Enterprise WPA adopts the 802.1X Port-Based Network Access Control protocol to authenticate STAs. The AS is connected to a wired local area network. The AS may be a separate server or implemented inside an AP. 802.1X was originally developed for authenticating dial-up devices. Using the early terminology of 802.1X, an STA is also referred to as a *supplicant*, an AP an *authenticator*, and the AS a *Remote Authentication Dial-In User Service* (RADIUS) server.

6.3.1.1 802.1X in A Nutshell

802.1X specifies a procedure for authenticating an STA when the STA wants to obtain access to a local area network (see Figure 6.5).

Figure 6.5 Schematic of 802.1X authentication steps, where dash lines represent the actual connections. Connections 2 and 3 are through the AP

1. The STA sends a request to the AP it wants to get access to. The AP asks for the identity of STA.
2. The STA sends to the AP its identity and its signature using its master key shared with the AS. The AP passes it to the AS. AS verifies STA's signature and grants the access permission to the STA if the signature is validated and passes its decision to AP. The AP then informs the STA about AS's decision.
3. The STA is granted access to the WLAN.

6.3.2   TKIP Key Generations

Suppose that an STA and an AP have completed the 802.1X authentication process with an AS, where the STA shares a presecret key with the AP and the AP shares a presecret key with the AS. The AS generates a 256-bit *pairwise master key* (PMK) and sends it confidentially to the AP using the preshared secret key between the AS and the AP. The AP then sends the PMK confidentially to the STA using the preshared secret key between the AP and the STA. In the case where 802.1X authentication is not required, the preshared master key between the STA and the AP is used to generate a PMK directly at both sides.

Different STA shares with the AP a different PMK. In the case where several STAs want to use the same master key for group communication, the key is referred to as a *group master key* (GMK), or simply a group key.

TKIP first generates a PMK for each STA. It then generates, on the basis of PMK and other information about the devices involved, four 128-bit secret pairwise transient keys (PTKs).

6.3.2.1   Pairwise Transient Keys

When an STA wants to connect to the AP, TKIP generates four 128-bit temporal subkeys for data encryptions, data integrity checks, EAPoL encryptions, and EAPoL integrity checks to

be used in the connection, where EAPoL stands for "Extensible Authentication Protocol over LAN". They are the *Data Encryption* key, the *Data MIC* key, the *EAPoL Encryption* key, and the *EAPoL MIC* key. These four subkeys are referred to as the *pairwise transient keys* (PTK). In particular, the first two keys are used to encrypt data and produce MIC of the data. The last two keys are used to secure communications between the AP and the STA during the initial handshake procedure. PTKs are temporal, for they are generated with different values each time an STA is associated with the AP. In other words, PTKs are session keys.

PTKs are generated using a pseudorandom number generator on the seed value made up of PMK, the MAC address of the STA, the MAC address of the AP, the nonce generated by the STA, and the nonce generated by the AP. We use AMAC and ANonce to denote, respectively, the MAC address and the nonce of the AP. Likewise, we use SMAC and SNonce to denote, respectively, the MAC address and the nonce of the STA.

6.3.2.2 Four-Way Handshakes

Both of the STA and the AP must have the same input to generate the same PTK. That is, the STA and the AP must exchange their MAC addresses and nonces after the STA is authenticated using 802.1X. TKIP uses the following four-way handshake procedure to complete this exchange.

802.1X introduces the notion of *robust security networks* (RSN) and the notion of *security network associations* (RSNA). RSNA specifies that in an RSN, APs can only be connected by RSN-enabled STAs.

Assume that the AS has informed the AP that the STA is authenticated and the PMK has been computed. Now the STA sends a special packet called *robust security network information element* (RSN IE) RSNIE_{STA} to the AP, where RSN IE contains authentication and pairwise cipher suite selectors, group key cipher suite selector, RSN capabilities, and other RSNA parameters. The AP responds to the STA with an RSN IE RSNIE_{AP} to inform the STA which algorithms and parameters it wants to use. Figure 6.6 shows a schematic of a typical RSN IE, where AKM stands for authentication and Key management algorithms.

Step 1: AP Sends ANonce to STA
The AP generates an ANonce, a sequence number *sn*, and sends

$$\text{message}_1 = (\text{AMAC, ANonce}, sn) \tag{6.2}$$

in plaintext to the STA. The STA generates an SNonce and computes the PTK from the PMK, SMAC, SNonce, AMAC, and ANonce.

Element ID	Length	Version	Group cipher suite	Pairwise cipher suite	Pairwise cipher suite list	AKM suite count	AKM suite list	RSN capabilities	PMKID count	PMKID list

Figure 6.6   Schematic of an RSN IE

Step 2: STA Sends SNonce to AP

The STA computes MIC of (SNonce, sn) using the EAPoL MIC key it generated in Step 1 and sends

$$\text{message}_2 = (\text{SMAC, SNonce, } sn) \parallel \text{MIC(SNonce, } sn) \parallel \text{RSNIE}_{STA} \qquad (6.3)$$

to the AP. The MIC ensures that the AP and the STA have the same PMK.

Step 3: AP Acknowledges STA

The AP extracts SNonce and SMAC from message_2 it receives from the STA and computes the PTK using the PMK shared with the STA, SMAC, SNonce, AMAC, and ANonce. It then uses the EAPoL MIC key to validate the MIC it receives. If the validation fails, the AP stops the handshake procedure. Otherwise, the AP sends the following acknowledgement message to the STA:

$$\text{message}_3 = (\text{AMAC, ANonce, } sn + 1) \parallel$$
$$\text{MIC(ANonce, } sn + 1) \parallel \text{RSNIE}_{AP}. \qquad (6.4)$$

This message indicates that the AP is ready to use the new PTK.

Step 4: STA Acknowledges AP

After receiving the acknowledgement message_3 the AP sends to the STA in Step 3, the STA sends the following acknowledgement to the AP:

$$\text{message}_4 = (\text{SMAC, } sn + 1) \parallel \text{MIC}(sn + 1) \qquad (6.5)$$

This message indicates that the STA is also ready to use the new PTK. This completes the four-way handshake procedure.

After the four-way handshake is completed, the AP and the STA will each generate and install PTK. Figure 6.7 shows a schematic of the four-way handshake procedure.

In addition to the MAC addresses, nonces, and the sequence numbers, the STA in step 2 will also send to the AP an RSN IE to establish security relation.

6.3.3   TKIP Message Integrity Code
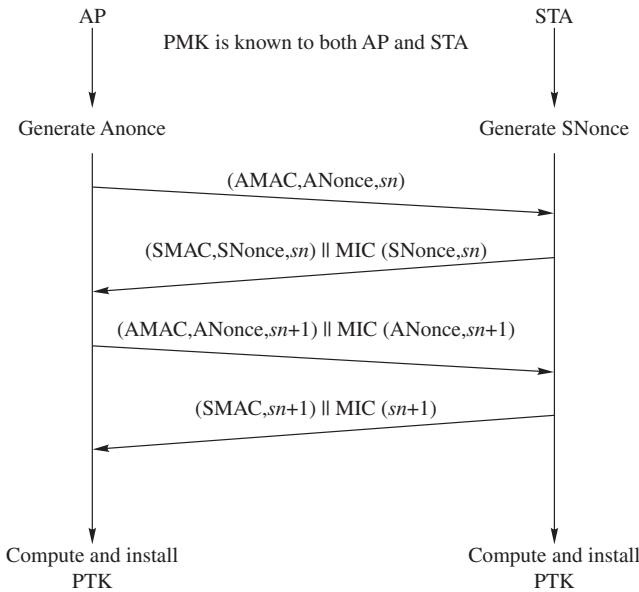
TKIP uses the Michael algorithm to generate MIC. Michael, designed solely for WPA by a Dutch cryptographic engineer Niels Ferguson, generates a 64-bit message authentication code using a 64-bit secret key. In particular, one half of the Data MIC key is used as the 64-bit secret key for authenticating messages sent from the AP to the STA, and the other half is used as the 64-bit secret key for authenticating messages sent from the STA to the AP.

TKIP stores strings in the little-endian storage format. For convenience, we assume that items (bits, bytes, words, etc.) with small indexes are stored in low memory locations.

Let K be a 64-bit secret key shared between the STA and the AP. Divide K into two halves K_0 and K_1 of equal length. Let

$$M = M_1 \cdots M_n$$

be an LLC frame to be transmitted, where each M_i is a 32-bit block.

Figure 6.7   Schematic of a 4-way handshake procedure

Michael generates MIC for M using K as follows:

$$(L_1, R_1) = (K_0, K_1),$$
$$(L_{i+1}, R_{i+1}) = F(L_i \oplus M_i, R_i),$$
$$i = 1, 2, \cdots, n$$
$$\mathrm{MIC} = L_{n+1}R_{n+1},$$

where F is a Feistel type of substitution. Let l and r be two 32-bit strings. Then $F(l, r)$ is defined as follows:

$$r_0 = r,$$
$$l_0 = l,$$
$$r_1 = r_0 \oplus (l_0 \lll 17),$$
$$l_1 = l_0 \oplus_{32} r_1,$$
$$r_2 = r_1 \oplus \mathsf{XSWAP}(l_1),$$
$$l_2 = l_1 \oplus_{32} r_2,$$
$$r_3 = r_2 \oplus (l_2 \lll 3),$$
$$l_3 = l_2 \oplus_{32} r_3,$$
$$r_4 = r_3 \oplus (l_3 \ggg 2),$$

$$l_4 = l_3 \oplus_{32} r_4,$$
$$F(l, r) = (l_4, r_4),$$

where $l \oplus_{32} r = (l + r) \bmod 2^{32}$, $\textsf{XSWAP}(l)$ swaps the left-half of l with the right-half of l. For example, representing numbers in hexadecimal, we have

$$\textsf{XSWAP}(12345678) = 56781234.$$

Michael as an MIC is much more secure than CRC_{32}, for Michael uses a 64-bit secret key on a Feistel encryption structure. But Michael MIC is still vulnerable to security attacks. For example, the attacker could create any message and attach a 64-bit binary string to it as a possible MIC, trying to find the correct MIC without knowing the secret key. Trying all of the 2^{64} strings will surely find the correct MIC for the message. While 2^{64} is large, there is a differential cryptanalysis attack that requires only 2^{29} tries. To prevent the attacker from keeping trying, TKIP specifies that if two failed forgeries are detected within a second, the STA should delete its keys and disengage with the AP. And the STA should wait for a minute before connecting to the AP again.

6.3.4 TKIP Key Mixing

TKIP generates a per-frame key using a key mixing algorithm for each frame that an STA wants to send to the AP. Key mixing uses a 48-bit initialization string V, generated for each frame using a 48-bit counter, which is referred to as the *TKIP sequence counter* (TSC). Divide V into three 16-bit blocks V_2, V_1, V_0.

The key mixing algorithm consists of two mixing phases, denoted by mix_1 and mix_2, where mix_1 takes a 128-bit input and outputs an 80-bit string, while mix_2 takes a 128-bit input and outputs a 128-bit string. Each mixing phase is a Feistel structure, involving a sequence of additions, XORs, and substitutions. The substitution function, denoted by S, uses two S-boxes, each of which is a table consisting of 256 bytes. Let a^t denote the 48-bit MAC address of the transmitter (i.e., the source device), k^t the 128-bit data encryption key of the transmitter, pk_1 the output of mix_1, and pk_2 the output of mix_2. Then

$$pk_1 = mix_1(a^t, V_2 V_1, k^t),$$
$$pk_2 = mix_2(pk_1, V_0, k^t),$$

where pk_2 is a 128-bit per-frame key for RC4.

6.3.4.1 S-Boxes

TKIP uses two S-boxes S_0 and S_1 to substitute a 16-bit string with a 16-bit string. In particular, divide X into two bytes: X_0 and X_1. That is, $X = X_1 X_0$. Treat X_0 and X_1 each as an index value from 0 to 255. Use S_0 to substitute X_0, the lower byte of X; and use S_1 to substitute X_1, the upper byte of X. Then $S_1(X_1)$ is the value in S_1 at index X_1 (see Table 6.1), and $S_0(X_0)$

Table 6.1 S_1: the TKIP S-box for the upper byte

```
c6 f8 ee f6 ff  d6 de 91 60 02 ce 56 e7 b5 4d ec 8f 1f 89 fa ef b2 8e fb
41 b3 5f 45 23 53 e4 9b 75 e1 3d 4c 6c 7e f5 83 68 51 d1 f9 e2 ab 62 2a
08 95 46 9d 30 37 0a 2f 0e 24 1b df cd 4e 7f ea 12 1d 58 34 36 dc b4 5b
a4 76 b7 7d 52 dd 5e 13 a6 b9 00 c1 40 e3 79 b6 d4 8d 67 72 94 98 b0 85
bb c5 4f ed 86 9a 66 11 8a e9 04 fe a0 78 25 4b a2 5d 80 05 3f 21 70 f1
63 77 af 42 20 e5 fd bf 81 18 26 c3 be 35 88 2e 93 55 fc 7a c8 ba 32 e6
c0 19 9e a3 44 54 3b 0b 8c c7 6b 28 a7 bc 16 ad db 64 74 14 92 0c 48 b8
9f bd 43 c4 39 31 d3 f2 d5 8b 6e da 01 b1 9c 49 d8 ac f3 cf ca f4 47 10
6f f0 4a 5c 38 57 73 97 cb a1 e8 3e 96 61 0d 0f e0 7c 71 cc 90 06 f7 1c
c2 6a ae 69 17 99 3a 27 d9 eb 2b 22 d2 a9 07 33 2d 3c 15 c9 87 aa 50 a5
03 59 09 1a 65 d7 84 d0 82 29 5a 1e 7b a8 6d 2c
```

Table 6.2 S_0: the TKIP S-Box for the lower byte

```
a5 84 99 8d 0d bd b1 54 50 03 a9 7d 19 62 e6 9a 45 9d 40 87 15 eb c9 0b
ec 67 fd ea bf f7 96 5b c2 1c ae 6a 5a 41 02 4f 5c f4 34 08 93 73 53 3f
0c 52 65 5e 28 a1 0f b5 09 36 9b 3d 26 69 cd 9f 1b 9e 74 2e 2d b2 ee fb
f6 4d 61 ce 7b 3e 71 97 f5 68 00 2c 60 1f c8 ed be 46 d9 4b de d4 e8 4a
6b 2a e5 16 c5 d7 55 94 cf 10 06 81 f0 44 ba e3 f3 fe c0 8a ad bc 48 04
df c1 75 63 30 1a 0e 6d 4c 14 35 2f e1 a2 cc 39 57 f2 82 47 ac e7 2b 95
a0 98 d1 7f 66 7e ab 83 ca 29 d3 3c 79 e2 1d 76 3b 56 4e 1e db 0a 6c e4
5d 6e ef a6 a8 a4 37 8b 32 43 59 b7 8c 64 d2 e0 b4 fa 07 25 af 8e e9 18
d5 88 6f 72 24 f1 c7 51 23 7c 9c 21 dd dc 86 85 90 42 c4 aa d8 05 01 12
a3 5f f9 d0 91 58 27 b9 38 13 b3 33 bb 70 89 a7 b6 22 92 20 49 ff 78 7a
8f f8 80 17 da 31 c6 b8 c3 b0 77 11 cb fc d6 3a
```

is the value in the S_0 at index X_0 (see Table 6.2). S_1 is also referred to as the TKIP S-Box for the upper byte and S_0 the TKIP S-Box for the lower byte. Define $S(X)$ by

$$S(X) = S_1(X_1)S_0(X_0).$$

For example, let $X = 0102$, then $S(X) = S_1(01)S_0(02) = f899$.

6.3.4.2 Computation of Phase 1

We use the following notations to describe the computation of phase 1 and phase 2:

a_n^t : the n th byte of a^t, where a_5^t is the highest byte and a_0^t the lowest byte

k_n^t : the n th byte of k^t, where k_{15}^t is the highest byte and k_0^t the lowest byte

Divide pk_1 into five 16-bit blocks as

$$pk_1 = pk_{14}pk_{13}pk_{12}pk_{11}pk_{10},$$

where each $pk_{1i}(i = 0, 1, \cdots, 4)$ is a 16-bit binary string. The phase 1 function $mix_1((a^t, V_2 V_1, k^t)$ is defined as follows:

$$pk_{10} \leftarrow V_1$$
$$pk_{11} \leftarrow V_2$$
$$pk_{12} \leftarrow a_1^t a_0^t$$
$$pk_{13} \leftarrow a_3^t a_2^t$$
$$pk_{14} \leftarrow a_5^t a_4^t$$

for $i \leftarrow 0$ **to** 3 **do**
$$pk_{10} \leftarrow pk_{10} \oplus_{16} S[pk_{14} \oplus (k_1^t k_0^t)]$$
$$pk_{11} \leftarrow pk_{11} \oplus_{16} S[pk_{10} \oplus (k_5^t k_4^t)]$$
$$pk_{12} \leftarrow pk_{12} \oplus_{16} S[pk_{11} \oplus (k_9^t k_8^t)]$$
$$pk_{13} \leftarrow pk_{13} \oplus_{16} S[pk_{12} \oplus (k_{13}^t k_{12}^t)]$$
$$pk_{14} \leftarrow pk_{14} \oplus_{16} S[pk_{13} \oplus (k_1^t k_0^t)] + \oplus_{16} i$$

$$pk_{10} \leftarrow pk_{10} \oplus_{16} S[pk_{14} \oplus (k_3^t k_2^t)]$$
$$pk_{11} \leftarrow pk_{11} \oplus_{16} S[pk_{10} \oplus (k_7^t k_5^t)]$$
$$pk_{12} \leftarrow pk_{12} \oplus_{16} S[pk_{11} \oplus (k_{11}^t k_{10}^t)]$$
$$pk_{13} \leftarrow pk_{13} \oplus_{16} S[pk_{12} \oplus (k_{15}^t k_{14}^t)]$$
$$pk_{14} \leftarrow pk_{14} \oplus_{16} S[pk_{13} \oplus (k_3^t k_2^t)] + \oplus_{16} 2i \oplus_{16} 1$$

6.3.4.3 Computation of Phase 2

Let *pt* denote an 80-bit binary string as a temporary variable. Divide *pt* into 16-bit blocks as

$$pt = pt_5 pt_4 pt_3 pt_2 pt_1 pt_0,$$

where each pt_i is a 16-bit string.

Let $X = X_1 X_0$ be a 16-bit string, where X_1 and X_0 are bytes. Denote by $ub(X) = X_1$ and $lb(X) = X_0$.

Let RC4Key denote the 128-bit output of $mix_2(pk_1, V_0, k^t)$, which is the per-frame key for RC4 to generate subkey stream. Divide RC4Key into 16 bytes as

$$\text{RC4Key} = \text{RC4Key}_{15} \ \text{RC4Key}_{14} \cdots \text{RC4Key}_0.$$

RC4Key is computed as follows:

for $i \leftarrow 0$ **to** 5 **do**
$$pt_i \leftarrow pk_{1i}$$
for $i \leftarrow 0$ **to** 5 **do**
$$pt_i \leftarrow pt_i \oplus_{16} S[pt_{(5+i) \bmod 6} \oplus (k_{2i+1}^t k_{2i}^t)]$$
for $i \leftarrow 0$ **to** 1 **do**
$$pt_i \leftarrow pt_i \oplus_{16} ([pt_{(5+i) \bmod 6} \oplus (k_{2i+13}^t k_{2i+12}^t)] \ggg 1)$$
for $i \leftarrow 2$ **to** 5 **do**
$$pt_i \leftarrow pt_i \oplus_{16} (pt_{i-1} \ggg 1)$$

$$\text{RC4Key}_0 \leftarrow ub(V_0)$$
$$\text{RC4Key}_1 \leftarrow (ub(V_0) \vee 00100000) \wedge 01111111$$

$\text{RC4Key}_2 \leftarrow lb(V_0)$
$\text{RC4Key}_3 \leftarrow lb(pt_5 \oplus [(k_1^t k_0^t) \gg 1])$
for $i \leftarrow 0$ **to** 5 **do**
$\quad \text{RC4Key}_{2i+4} \leftarrow lb(pt_i)$
$\quad \text{RC4Key}_{2i+5} \leftarrow lb(pt_i)$

Note that the lowest three bytes of RC4Key, denoted by *IV*, are

$$IV = ub(V_0) \parallel U \parallel lb(V_0), \tag{6.6}$$

where $U = 0 v_{13} 1 v_{12} \cdots v_8$. *IV* is to be transmitted as the initialization vector in WEP, where $ub(V_0)$ is to be transmitted first. The remaining 104-bit string is used as a WEP key.

6.3.5   WPA Encryption and Decryption

At the transmission end, WPA encrypts MSDU (i.e., an LLC frame) by reusing the WEP encryption block. In particular, the ICV of an MSDU is computed first and then attached at the end of the MSDU. This new string is then fragmented into several smaller blocks according to the MAC sublayer specification. The initialization vector $V_2 V_1 V_0$ is included in the MAC Protocol Data Unit (MPDU) (i.e., the MAC sublayer frame) and transmitted in the public, where V_0 can be obtained from *IV* by removing the middle byte U. The middle byte U in the WEP *IV* is used to avoid a certain type of RC4 weak keys. Figure 6.8 shows a schematic of WPA encryption.

At the receiving end, WPA strips off the MAC sublayer header, extracts the initialization vector V, and computes the transient keys. It then decrypts fragmented MSDU and reassembles them back to the original MSDU and its ICV.
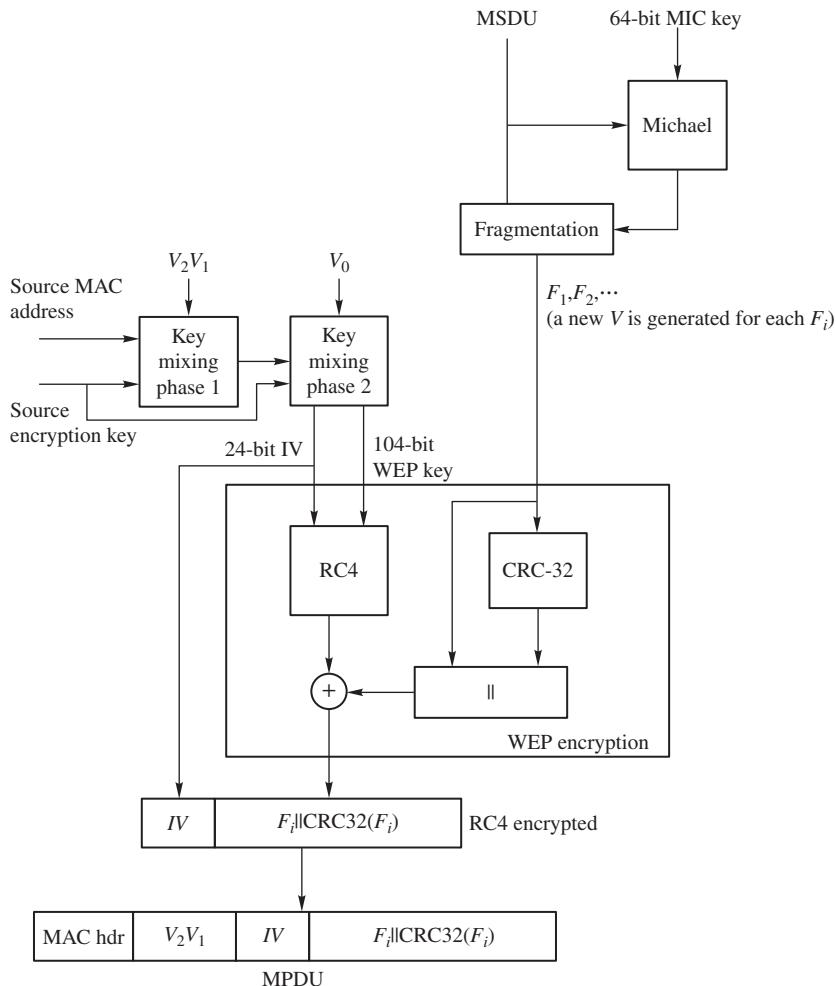
The transmitter increments its initialization vector by 1, starting from 0, for each fragmented MSDU to be sent. Out-of-order frames will be dropped at the destination. The TSC counter for the initialization vector will be reset to 0 for a new connection with a new data encryption key. This mechanism prevents message replays.

6.3.6   WPA Security Strength and Weaknesses

WPA is superior to WEP in a number of ways. WPA uses 802.1X to authenticate devices and uses TKIP to generate temporal keys for encrypting LLC frames and producing MICs of the frames to be sent. TKIP, in particular, is the major security product that reuses the WEP encryption mechanism. Users may be able to upgrade their existing WEP devices to run WPA with minimal costs.

However, the security strength of TKIP has not been analyzed, and so flaws unknown of at this point may be discovered later to attack WPA.

WPA is vulnerable to DoS attacks. Let M be an LLC frame. WPA computes the MIC of M and includes it in the payload. WPA then computes fragments of $M \parallel \text{ICV}(M)$ to $F_1, F_2, \cdots$, according to the MAC sublayer protocol. For each fragment F_i, WPA generates a 48-bit initialization vector V_i and uses it to generate a WEP initialization vector and a WEP key. Because the values of the initialization vector are always increased and because it is transmitted in plaintext, the attacker may intercept an MAC frame and replace the initialization vector contained in it with a larger value. As it cannot be decrypted correctly, the encrypted frame will be discarded.

Figure 6.8   A schematic of WPA encryption, where only the major components in an MPSU are shown

This will cause legitimate MAC frames that arrive at a later time to be rejected because the value of its initialization vector has already been used.

Another type of DoS attacks takes advantage of the TKIP specification with MIC verifications. Recall that if two MSDUs with forged MICs are detected within a second, TKIP will discard these MSDUs and disconnect the STA from the AP. Thus, the attacker may simply keep sending forged MSDUs to prevent legitimate STAs from being connected to the AP.

6.4   IEEE 802.11i/WPA2

WPA, published in 2002, was a rush solution directed at solving urging security problems in WEP using existing WEP hardware. WPA was based on draft 3 of IEEE 802.11i. In 2004,

IEEE published the official version of the 802.11i standard. The Wi-Fi Alliance subsequently published WPA2 on the basis of 802.11i, which is interoperable with 802.11i. This section is focused on 802.11i.

802.11i defines a *counter mode-CBC MAC protocol* (CCMP) using AES-128 to encrypt data and compute the MIC of the data. 802.11i also uses 802.1X to authenticate STAs.

As AES-128 is used for both encryption and authentication, encryption keys can be reused. Thus, initialization vectors transmitted in plaintext are no longer needed to generate per-frame keys.

However, unlike WPA that can reuse WEP cards to support WPA, most existing Wi-Fi WPA cards cannot be upgraded to support 802.11i, for 802.11i uses AES as its underlying encryption algorithm.

6.4.1 Key Generations

IEEE 802.11i has the same key hierarchy as WPA. That is, 802.11i generates a 256-bit PMK and four temporal 128-bit PTKs for WPA. In addition, 802.11i also generates a 384-bit temporal key for each session between an STA and the AP, used to carry out CCMP operations. This 384-bit key is generated using a pseudorandom number generator on the STA's MAC address, the STA's nonce, the AP's MAC address, and the AP's nonce, which are exchanged following a 4-way handshake protocol as in WPA.

The 384-bit key is then divided into three 128-bit transient keys, two of which are used to establish connection between the STA and the AP. The other is used as a session key for carrying out the AES-128 encryption algorithm.

6.4.2 CCMP Encryptions and MIC

CCMP uses AES counter mode to encrypt MSDUs. That is, it uses an 128-bit counter *Ctr*, starting from an initial value and increases 1 each time. Let M be an MSDU. Divide M into a sequence of 128-bit blocks:
$$M = M_1 M_2 \cdots M_k.$$

Let Ctr_0 denote the initial value of the counter. Let K denote the 128-bit AES encryption key and $\text{AES-128}_K(X)$ the AES-128 encryption algorithm on 128-bit string X with key K. Then the encryption of M is carried out as follows:
$$Ctr = Ctr_0,$$
$$C_i = \text{AES-128}_K(Ctr + 1) \oplus M_i,$$
$$i = 1, \cdots, k.$$

CCMP uses cipher-block chaining message authentication code (CBC-MAC) to authenticate MSDUs and perform integrity checks. CBC-MAC using AES-128 is defined as follows:
$$C_0 = 0^{128},$$
$$C_i = \text{AES-128}_K(C_{i-1} \oplus M_i),$$
$$i = 1, 2, \cdots, k.$$

Unlike in WPA that the MIC is calculated on an MSDU before it is fragmented at the MAC sublayer, 802.11i fragments the MSDU first. In particular, 802.11i calculates the CBC-MAC of each fragment and appends the MIC to the fragment. It then encrypts each fragment together with its MIC.

6.4.3 802.11i Security Strength and Weaknesses

The cryptographic algorithms and security mechanism used in 802.11i are superior to those used in WPA and WEP. 802.11i eliminates the use of public initialization vectors in generating session keys. In addition to using CCMP for encryption and authentication, 802.11i also recommends using the mode of OCB (described in Section 2.6) as an alternative for encryption and authentication. However, OCB is patented, and so adopting it officially in an industry standard is inappropriate.

While CCMP is believed to be secure, 802.11i is still vulnerable to a number of security attacks. Most of the security vulnerabilities in 802.11i are from communication protocols. For example, 802.11i is vulnerable to rollback attacks, and the four-way handshake protocol is vulnerable to RSN IE poisoning attacks. Moreover, as we mentioned earlier, wireless networks are vulnerable to DoS attacks, and 802.11i is no exception. Most DoS attacks are targeted at MAC-layer protocols. We describe in this subsection three DoS attacks: the rollback attacks, the RSN IE poisoning attacks, and the de-association attacks. The reader is referred to Exercises 6.18, 6.19, and 6.20 for several other possible DoS attacks.

6.4.3.1   Rollback Attacks

IEEE 802.11i is meant to establish only RSNs. However, to accommodate existing WEP and WPA devices, 802.11i allows RSN devices to communicate with pre-RSN devices. This makes *rollback attacks* possible, for the attacker may be able to trick an RSN device to roll back to WEP. For example, the attacker may impersonate a legitimate RSN AP to broadcast a message announcing that it is a WEP AP, or impersonate a legitimate RSN STA to request a WEP connection with an RSN AP.

To counter rollback attacks, we may configure RSN APs to decline WEP or WPA connections. This countermeasure, however, will block WEP or WPA devices from using RSN APs. Thus, a better measure would be for RSN APs to decline WEP or WPA connections for critical applications where security is a primary concern and allow WEP or WPA connections for applications where weaker security protections are acceptable.

6.4.3.2   RSN IE Poisoning Attacks

RSN IE poisoning is a DoS attack against the four-way handshake protocol. Recall that $message_2$ sent by the STA to the AP in Step 2 of the four-way handshake protocol (see Message 6.3) contains $RSNIE_{STA}$. The AP verifies MIC and discards $message_2$ if the MIC is incorrect. Otherwise, the AP compares bit-by-bit $RSNIE_{STA}$ with the local record the AP received prior to the handshake. If they are not identical, the AP stops the handshake procedure and de-authenticates the STA.

In Step 3, the AP sends $\mathrm{RSNIE_{AP}}$ to the STA in $\mathrm{message_3}$. The STA then checks the RSN IE before verifying MIC. If the received RSN IE is not identical to the local record, the STA stops the handshake procedure and de-authenticates the AP. Checking RSN IE before verifying MIC presents a design flaw, for the attacker can forge $\mathrm{message_3}$ with the wrong RSN IE and force the STA to disconnect from the AP.

6.4.3.3   De-Association Attacks

De-association attacks use forged MAC-layer management frames to break an existing connection between an STA and an AP. For example, suppose that an STA has already established a connection with an AP. The attacker sends a forged de-authentication frame to the AP to de-associate the STA from the AP. The AP, however, believes that the connection still exists, making it possible for the attacker to impersonate the STA to connect to the AP.

6.5   Bluetooth Security
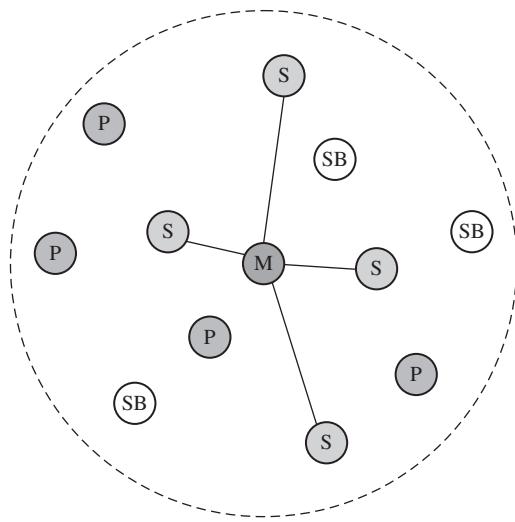
Bluetooth is a communication technology for building ad hoc WPANs. It allows wireless devices with low power, for example, cellular phones, PDAs, and embedded systems, to communicate with each other within a short range. The IEEE 802.15 standard for WPANs is based on the Bluetooth technology.

On the basis of the Mobile Communication architecture developed by Ericsson, Bluetooth was proposed in 1998 as an industrial standard by the Special Interest Group (SIG) formed by Ericsson, Intel, IBM, Nokia, and Toshiba. Bluetooth allows wireless devices of different platforms made by different vendors to communicate with each other. It was named after Harald Bluetooth, the 10th century Danish king, who advocated negotiations to solve regional conflicts. Designed to support wireless devices with low power, that is, with limited computing capabilities and power supplies, Bluetooth is restricted to cryptographic algorithms that do not require much computing resources to execute.
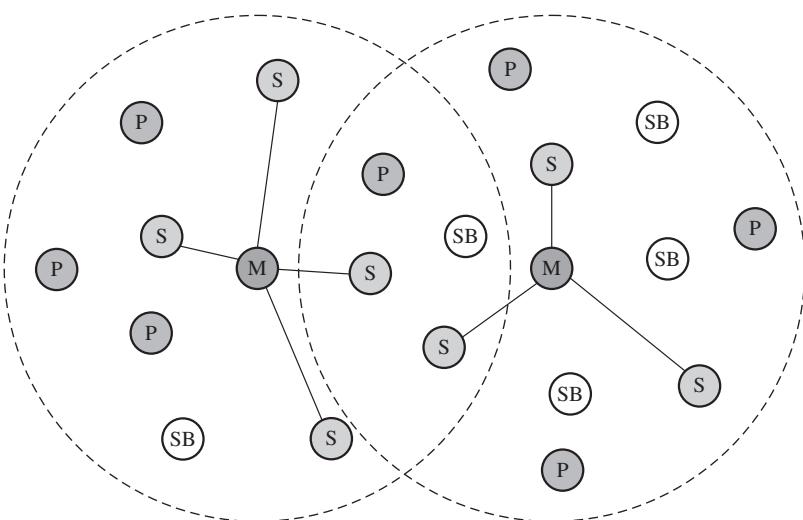
6.5.1   Piconets

Bluetooth is implemented on *piconets*, which are self-configured and self-organized ad hoc wireless networks. Piconets are formed dynamically, allowing new devices to join in and current devices to leave at will without using access points or other infrastructure devices. In particular, a piconet may consist of up to eight active devices that use the same physical channel. All devices in a piconet are peers, that is, they can communicate with each other directly. Exactly one of these peers is designated as the *master* node for the purpose of synchronizing other nodes. The other nodes are referred to as the *slave* nodes. Slave nodes are synchronized with the master node.

A Bluetooth device may also be in the parked state. A device in the parked state can become active quickly. Other devices are said to be in the standby state. A standby device takes a longer time to become active. A piconet may consist of up to 255 parked devices. Figure 6.9 shows a schematic of a piconet.

Figure 6.9   Piconet schematic. M denotes the "master node," S "slave nodes," P "parked-state devices" and SB "standby devices"

Figure 6.10   Scatternet schematic

When a Bluetooth device wants to set up a piconet, it sends out a special packet and becomes the master node of the piconet. When a Bluetooth device wants to join an existing piconet, it sends out a special request-to-join packet and becomes a slave node.

Several piconets may overlap, which form a scatternet. Figure 6.10 shows a schematic of a scatternet. A device can only belong to one piconet at a time.

6.5.2 Secure Pairings

Suppose that two Bluetooth devices D_A and D_B in the same piconet want to communicate securely. For convenience, we will assume that D_A is the device that initiates the communication. Initially, D_A and D_B must share a personal identification number (PIN). A PIN code is a string of characters of up to 16 characters, which may be entered by the user or prestored in a device if the device has no input functions.

D_A and D_B will then generate shared secret keys to authenticate each other. This is referred to as a *secure pairing*. Thus, authentication is the most important part of a secure pairing.

At the beginning of a secure pairing session, Bluetooth generates a 128-bit initialization key for each device on the basis of the PIN code and other information. The devices then each generate a 128-bit *link* key, also called a *combination* key. Bluetooth uses the link key to authenticate devices and generate encryption keys to encrypt packets.

Bluetooth uses a stream cipher called E_0 to encrypt packet payloads, which is re-synchronized for each payload to be encrypted. Bluetooth uses a block cipher SAFER+ and a modified version of SAFER+ to construct three algorithms, denoted by E_1, E_{21}, and E_{22}, to generate subkeys and authenticate devices.

This section is focused on Bluetooth authentication.

6.5.3 SAFER+ Block Ciphers

SAFER+ is a block cipher used to authenticate Bluetooth devices in secure pairings. SAFER+ is an enhancement of SAFER devised by James L. Massey in 1993. SAFER, standing for Secure And Fast Encryption Routine, is a Fiestel cipher with a 64-bit block size. SAFER+ is a Fiestel cipher with a 128-bit block size. As in AES, SAFER+ allows 128-bit, 192-bit, and 256-bit key lengths. Bluetooth uses SAFER+ with 128-bit keys, denoted by SAFER+ K-128.

As in any Fiestel cipher, SAFER+ K-128 consists of a key scheduling component and an encryption component. The SAFER+ K-128 encryption component consists of eight identical rounds and an output transformation, which need a total of 17 subkeys: two for each round and one for the output transformation.

6.5.3.1 SAFER+ Subkeys

For convenience, for any k-byte string $X = x_1 x_2 \cdots x_k$, we use $X[i]$ to denote the ith byte x_i and $X[i:j]$ to denote the substring $x_i \cdots x_j$, where $0 \le i \le j \le k$ and x_i is a byte.
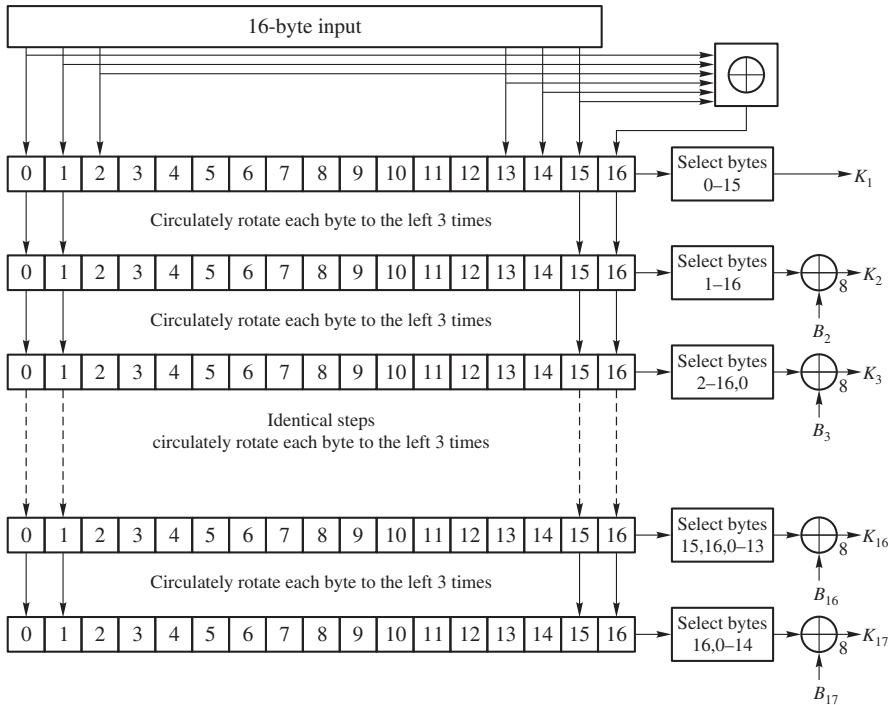
Let $K = k_0 k_1 \cdots k_{15}$ be a 128-bit encryption key, where k_i is a byte for $i = 0, 1, \cdots, 15$. Let

$$k_{16} = k_0 \oplus k_1 \oplus \cdots \oplus k_{15}.$$

Let X be a byte. Recall that $LS_k(X)$ denotes the new string obtained by shifting X circularly to the left k times. SAFER+ generates seventeen 128-bit subkeys $K_1, K_2, \cdots, K_{17}$ as follows:

The first subkey K_1 is K. Let

$$K \leftarrow K \parallel k_{16}$$

Figure 6.11 Schematic of SAFER+ subkey generation

be a 17-byte expanded key. To generate each of the remaining subkeys $K_i, i = 2, \cdots, 17$, first perform LS_3 on each byte of the expanded key. Then select 16 bytes from the resulting string in a left-circular shift manner, and perform an XOR operation on the selected 16-byte string and a constant 16-byte string B_i, called a *bias* vector to produce K_i, where B_i is obtained as follows:

$$B_i[j] = \left(45^{\left(45^{17i+j+1} \bmod 257 \right)} \bmod 257 \right) \bmod 256,$$

$$j = 0, 1, \cdots, 15,$$

$$B_i = B_i[0]B_i[1] \cdots B_i[15],$$

$$i = 2, 3, \cdots, 17.$$

The following procedure generates K_i (see Figure 6.11):

$$K_1 \leftarrow k_0 k_1 \cdots k_{15}$$
for $j = 0, 1, \cdots, 16$ **do**
$\qquad k_j \leftarrow LS_3(k_j)$
$K_2 \leftarrow k_1 k_2 \cdots k_{16} \oplus_8 B_2$
for $i = 3, 4, \cdots, 17$ **do**
$\qquad$ **for** $j = 0, 1, \cdots, 16$ **do**
$\qquad\qquad k_j \leftarrow LS_3(k_j)$
$\qquad K_i \leftarrow k_{i-1} k_i \cdots k_{16} k_0 \cdots k_{i-3} \oplus_8 B_{i-3}$

where $\oplus_8$ denotes bytewise addition $\mod 2^8$. That is, let $X = x_1 x_2 \cdots x_k$ and $Y = y_1 y_2 \cdots y_k$ be two strings, where x_i and y_i are bytes, $i = 1, 2, \cdots, k$. Then

$$X \oplus_8 Y = x_1 \oplus_8 y_1 \parallel x_2 \oplus_8 y_2 \parallel \cdots \parallel x_k \oplus_8 y_k.$$

6.5.3.2 SAFER+ Encryption

SAFER+ K-128 encrypts a 128-bit plaintext block through eight identical rounds of operations and one output transformation.

Encryption Rounds

Operations in each round consist of Pseudo Hadamard Transform (PHT), Armenian Shuffles (ArS), table lookups on two S-boxes e and l, and the $\oplus$ and $\oplus_8$ operations with two subkeys.

PHT takes two bytes x and y as input and produces two-byte output as follows:

$$\text{PHT}(x, y) = (2x + y) \mod 2^8 \parallel (x + y) \mod 2^8.$$

Let $X = x_1 x_2 \cdots x_{2k-1} x_{2k}$ be a string, where each x_i is a byte, $i = 1, 2, \cdots, 2k$. Define

$$\text{PHT}(X) = \text{PHT}(x_1, x_2) \parallel \text{PHT}(x_3, x_4) \parallel \cdots \parallel \text{PHT}(x_{2k-1}, x_{2k}).$$

ArS permutes bytes in a 16-byte string $X = x_0 x_1 \cdots x_{15}$ as follows, where each x_i is a byte for $i = 0, 1, \cdots, 15$:

$$\text{ArS}(X) = x_8 x_{11} x_{12} x_{15} x_2 x_1 x_6 x_5 x_{10} x_9 x_{14} x_{13} x_0 x_7 x_4 x_3.$$

The S-Box e is a table with 2^8 entries, which is used to substitute an input byte x with a new byte $e(x)$ from the table, where $e(x)$ is defined using an exponentiation function as follows:

$$e(x) = (45^x \mod (2^8 + 1)) \mod 2^8.$$

The S-Box l is an inverse box of e. Namely, l has 2^8 entries, and the value of each entry $l(y)$ is determined by

$$l(y) = x \quad \text{if} \quad e(x) = y.$$

Let Y_i denote the 128-bit input to the ith round, where $1 \le i \le 8$. That is, Y_i ($i > 1$) is the output of the $(i - 1)$th round and Y_1 is the original plaintext block. Let K_{2i-1} and K_{2i} be the two subkeys. Let Z_1 and Z_2 be two 128-bit temporary strings. The ith round in SAFER+ performs the following computations:

$$Z_0 = Y_i,$$
$$Z_1[2j - 2] = e(Z_0[2j - 2] \oplus K_{2i-1}[2j - 2]),$$
$$Z_1[2j - 1] = l(Z_0[2j - 1] \oplus_8 K_{2i-1}[2j - 1]),$$
$$Z_1[2j] = l(Z_0[2j] \oplus_8 K_{2i-1}[2j]),$$
$$Z_1[2j + 1] = e(Z_0[2j + 1] \oplus K_{2i-1}[2j + 1]),$$
$$j = 1, 3, 5, 7.$$

$$Z_2[2j - 2] = l(Z_1[2j - 2] \oplus_8 K_{2i}[2j - 2]),$$
$$Z_2[2j - 1] = e(Z_1[2j - 1] \oplus K_{2i}[2j - 1]),$$
$$Z_2[2j] = e(Z_1[2j] \oplus K_{2i}[2j]),$$
$$Z_2[2j + 1] = l(Z_1[2j + 1] \oplus_8 K_{2i}[2j + 1]),$$
$$j = 1, 3, 5, 7.$$
$$Y_{i+1} = \text{PHT(ArS(PHT(ArS(PHT(ArS(PHT}(Z_2))))))).$$

Output Transformation

After eight identical rounds are finished, the output transformation component applies K_{17} to Y_9 as applying K_{2i-1} to Y_i without using S-boxes and generates ciphertext block C. That is,

$$C[2j - 2] = Y_9[2j - 2] \oplus K_{17}[2j - 2],$$
$$C[2j - 1] = Y_9[2j - 1] \oplus_8 K_{17}[2j - 1],$$
$$C[2j] = Y_9[2j] \oplus_8 K_{17}[2j],$$
$$C[2j + 1] = Y_9[2j + 1] \oplus K_{17}[2j + 1],$$
$$j = 1, 3, 5, 7.$$

6.5.4 Bluetooth Algorithms E_1, E_{21}, and E_{22}

Bluetooth uses SAFER+ and a modified version of SAFER+ to construct E_1. It uses the modified version of SAFER+ to construct E_{21} and E_{22}. The modified version of SAFER+ combines the input of round 1 to the input of round 3 using $\oplus$ and $\oplus_8$ operations in the same way of combining K_5 to Y_3 without using the S-boxes, making the algorithm noninvertible. In particular, the new Y_3 is modified as follows:

$$Y_3[2j - 2] \leftarrow Y_3[2j - 2] \oplus Y_1[2j - 2],$$
$$Y_3[2j - 1] \leftarrow Y_3[2j - 1] \oplus_8 Y_1[2j - 1],$$
$$Y_3[2j] \leftarrow Y_3[2j] \oplus_8 Y_1[2j],$$
$$Y_3[2j + 1] \leftarrow Y_3[2j + 1] \oplus Y_1[2j + 1],$$
$$j = 1, 3, 5, 7.$$

For convenience, we use A_r to denote the original SAFER+ encryption algorithm and A'_r to denote the modified version of the SAFER+ encryption algorithm.

6.5.4.1 E_1

Let K be a 16-byte key, ρ a 16-byte string, and α a 6-byte address. Define $\tilde{K}$ as follows:

$$\tilde{K}[0] = K[0] \oplus_8 233, \qquad \tilde{K}[1] = K[1] \oplus 229,$$
$$\tilde{K}[2] = K[2] \oplus_8 223, \qquad \tilde{K}[3] = K[3] \oplus 193,$$
$$\tilde{K}[4] = K[4] \oplus_8 179, \qquad \tilde{K}[5] = K[5] \oplus 167,$$
$$\tilde{K}[6] = K[6] \oplus_8 149, \qquad \tilde{K}[7] = K[7] \oplus 131,$$
$$\tilde{K}[8] = K[8] \oplus 233, \qquad \tilde{K}[9] = K[9] \oplus_8 229,$$
$$\tilde{K}[10] = K[10] \oplus 223, \qquad \tilde{K}[11] = K[11] \oplus_8 193,$$
$$\tilde{K}[12] = K[12] \oplus 179, \qquad \tilde{K}[13] = K[13] \oplus_8 167,$$
$$\tilde{K}[14] = K[14] \oplus 149, \qquad \tilde{K}[15] = K[15] \oplus_8 131.$$

Define an expansion function E that expands α cyclically to a 16-byte string as follows:

$$E(\alpha) = \alpha \parallel \alpha \parallel \alpha[0:3].$$

E_1 takes r, K, and α as input and produces the following 16-byte string as output:

$$E_1(K, \rho, \alpha) = A'_r(\tilde{K}, [A_r(K, \rho) \oplus \rho] \oplus_8 E(\alpha)). \tag{6.7}$$

6.5.4.2 E_{21}

The E_{21} function takes a 16-byte random string ρ and a 6-byte address α as input. Let

$$\rho' = \rho[0:14] \parallel (\rho[15] \oplus b(6)),$$

where $b(6)$ denotes the 8-bit string of 6: 00000110. Then

$$E_{21}(\rho, \alpha) = A'_r(\rho', E(\alpha)). \tag{6.8}$$

6.5.4.3 E_{22}

The E_{22} algorithm takes a 16-byte random string ρ, a 6-byte address α, and an ℓ-byte PIN code p as input, where $1 \le \ell \le 16$. Let

$$\text{PIN}' = \begin{cases} \text{PIN} \parallel \alpha[0] \parallel \cdots \parallel \alpha[\min\{5, 15 - \ell\}], & \text{if } \ell < 16, \\ \text{PIN}, & \text{if } \ell = 16. \end{cases}$$

Let $\ell' = \min\{16, \ell + 6\}$. Let

$$\kappa = \begin{cases} \text{PIN}' \parallel \text{PIN}' \parallel \text{PIN}'[0:1], & \text{if } \ell' = 7, \\ \text{PIN}' \parallel \text{PIN}'[0:15-\ell'], & \text{if } 8 \le \ell' < 16, \\ \rho, & \text{if } \ell' = 16, \end{cases}$$

$$\rho' = \rho[0:14] \parallel (\rho[15] \oplus b(\ell')),$$

where $b(\ell')$ denotes the 8-bit presentation of ℓ'. Then

$$E_{22}(\text{PIN}, \rho, \alpha) = A'_r(\kappa, \rho'). \tag{6.9}$$

6.5.5 Bluetooth Authentication

Recall that each Bluetooth device has an ℓ-byte PIN code, where $1 \le \ell \le 16$. To create a pairing between two devices D_A and D_B, where D_A initiates the communication with D_B, both devices must have the same PIN code, which may be prestored in the devices or entered by the users. Each Bluetooth device also has a 6-byte device address, denoted by BD_ADDR.

Bluetooth generates an *initialization* key K_{init} and a link key K_{AB} for performing device authentications.

6.5.5.1 Initialization Key

When D_A initiates the communication with D_B, D_A generates a 16-byte pseudorandom string, denoted by IN_RAND$_A$ and sends it to D_B in plaintext. Then both D_A and D_B compute the initialization key K_{init} as follows:

$$K_{init} = E_{22}(\text{PIN}, \text{IN\_RAND}_A, \text{BD\_ADDR}_B),$$

where BD_ADDR$_B$ is the 6-byte device address of D_B.

6.5.5.2 Link Key
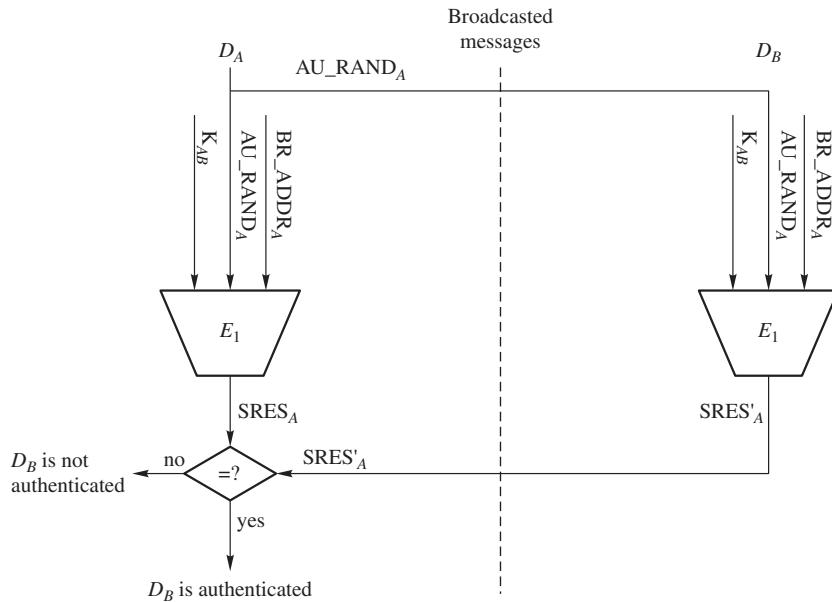
Both D_A and D_B will create the link key K_{AB} after they have created K_{init}. In particular, D_A generates a 16-byte pseudorandom string LK_RAND$_A$, and D_B generates a 16-byte pseudorandom string LK_RAND$_B$. D_A then transmits LK_RAND$_A \oplus K_{init}$ to D_B and D_B transmits LK_RAND$_B \oplus K_{init}$ to D_A, which allow D_A to get LK_RAND$_B$ and D_B to get LK_RAND$_A$.

D_A and D_B each compute K_{AB} as follows:

$$K_{AB} = E_{21}(\text{LK\_RAND}_A, \text{BD\_ADDR}_A) \oplus E_{21}(\text{LK\_RAND}_B, \text{BD\_ADDR}_B).$$

6.5.5.3 Device Authentication

Bluetooth devices use the challenge-response scheme to authenticate peers. For D_A to authenticate D_B, D_A generates a 16-byte pseudorandom string AU_RAND$_A$ and sends it to D_B in

Figure 6.12   Bluetooth device D_A authenticates device D_B

plaintext. D_B calculates a 4-byte *singed response* (SRES) as follows:

$$\mathsf{SRES}_A = E_1(K_{AB}, \mathsf{AU\_RAND}_A, \mathsf{BD\_ADDR}_B)[0:3].$$

It then sends it to D_A. D_A verifies the correctness of the SRES_A it receives from D_B to validate that D_A possesses the same K_{AB} to authenticate D_B. Figure 6.12 shows a schematic of device authentication.

6.5.6   A PIN Cracking Attack

The Bluetooth authentication protocol is vulnerable to the PIN cracking attack that is similar to the meet-in-the-middle attack in 2DES. In particular, assume that Malice eavesdrops an entire pairing and authentication session between D_A and D_B as shown in Table 6.3.

Malice may crack the PIN out by brute force as follows: she first enumerates all 2^{48} possible values of PIN. For each PIN enumerated (denoted by PIN′), Malice can use $\mathsf{IN\_RAND}_A$ from Message 1 and $\mathsf{BD\_ADDR}_B$ to compute a candidate

$$K'_{init} = E_{22}(\,\mathrm{PIN}', \mathsf{IN\_RAND}_A, \mathsf{BD\_ADDR}_B).$$

She uses K'_{init} to XOR Message 2 and to XOR Message 3 to obtain, respectively, $\mathsf{LK\_RAND}'_A$ and $\mathsf{LK\_RAND}'_B$. She then computes a candidate

$$K'_{AB} = E_{21}(\mathsf{LK\_RAND}'_A, \mathsf{BD\_ADDR}_A) \oplus E_{21}(\mathsf{LK\_RAND}'_B, \mathsf{BD\_ADDR}_B).$$

Table 6.3 Messages between D_A and D_B during the entire pairing and authentication session eavesdropped by the attacker

Message	Source	Destination	Data	Length	Notes
1	D_A	D_B	IN_RAND$_A$	128 bits	Plaintext
2	D_A	D_B	LK_RAND$_A \oplus K_{init}$	128 bits	
3	D_B	D_A	LK_RAND$_B \oplus K_{init}$	128 bits	
4	D_A	D_B	AU_RAND$_A$	128 bits	Plaintext
5	D_B	D_A	SRES$_A$	32 bits	Plaintext
6	D_B	D_A	AU_RAND$_B$	128 bits	Plaintext
7	D_A	D_B	SRES$_B$	32 bits	Plaintext

Using the candidate K'_{AB} and the last four messages, Malice can verify whether the candidate PIN code is indeed the PIN code used by D_A and D_B. In particular, Malice uses AU_RAND$_A$ from Message 4, K'_{AB}, and BD_ADDR$_B$ to compute

$$\text{SRES}'_A = E_1(\text{AU\_RAND}_A, K'_{AB}, \text{BD\_ADDR}_B)[0:3],$$

and verifies whether SRES$'_A$ is identical to SRES$_A$ from Message 5. If yes, then Malice has a good chance to have found PIN. She may further use Messages 6 and 7 to confirm the PIN code.

We note that cracking a 4-digit PIN on a desktop computer can be done in only a fraction of a second. Four-digit PINs were commonly used in Bluetooth devices. Thus, one should avoid using short PIN codes.

6.5.7 Bluetooth Secure Simple Pairing

The Bluetooth Core Specification 2.1 + EDR, introduced in 2007, uses a new pairing protocol to improve Bluetooth security. The protocol is called the secure simple pairing (SSP) protocol. It uses the elliptic-curve Diffie–Hellman (ECDH) key-exchange algorithm (see Section 3.5.5) to negotiate shared secret keys between peers to replace user PIN codes, which makes SSP resistant to the PIN cracking attack.

Initially, each Bluetooth device generates its own ECDH public–private key pair (K^u, K^r) for all future SSP pairings.

SSP first exchanges public keys as follows: device D_A (the initiator) first sends its public key K_A^u to device D_B. Device D_B then sends its public key K_B^u to D_A.

This public-key exchange process is vulnerable to the man-in-the-middle attack described in Section 3.3.2. In particular, the attacking device may intercept both K_A^u and K_B^u and inject its own key to replace K_A^u and K_B^u, respectively, for D_B and D_A.

To counter this man-in-the-middle attack, it is desirable to use public-key certificates to authenticate the owners of the public keys. Bluetooth, however, does not specify link-layer public-key infrastructure.

Assume that D_B has obtained D_A's public key K_A^u and D_A has obtained D_B's public key K_B^u. That is, no man-in-the-middle attacks occurred during the key-exchange phase. SSP

specifies two authentication phases after the key-exchange phase to counter other forms of man-in-the-middle attacks. SSP then calculates the link key on the basis of the public keys to complete the pairing process.

6.6 ZigBee Security

The ZigBee protocol is a standard for low-power wireless personal area networks similar to Bluetooth. The protocol is built on top of the IEEE 802.15.4 communications standard. Devices powered by the ZigBee protocol are widely used in health monitoring devices, home security systems, and home automation systems, to name just a few.

The ZigBee protocol consists of the following four protocol layers: the physical layer, the medium access control layer, the network layer, and the application layer. The physical layer and medium access control are strictly specified by the IEEE 802.15.4 standard.

The network layer in the ZigBee is responsible for supporting the network topology. The topologies supported by ZigBee are star, tree, and mesh networks. The use of mesh networks is most common. When a ZigBee device wants to create a personal area network, it declares itself as a *ZigBee coordinator* and starts a new WPAN. The ZigBee coordinator is also referred to as the *trust center*. All other devices that join the new WPAN are called *ZigBee end devices*.

The application layer is responsible for providing a framework on which new ZigBee devices are developed. Within the application layer, there is a special sublayer called the application support sublayer. The application support sublayer is responsible for directly interfacing with the network layer.

The security of a ZigBee network consists of many of the same facets as a Bluetooth piconet. Similarly to Bluetooth, ZigBee has the ability to join networks (pairing in the Bluetooth parlance), authenticate, and secure the communication between devices.

6.6.1 Joining a Network

Similarly to Bluetooth, a ZigBee device joins a ZigBee network as an unauthenticated member. The basic protocol consists of the following four messages that occur between the ZigBee device D and the ZigBee coordinator C:

1. $D \rightarrow C$: Request a beacon.
2. $C \rightarrow D$: Respond with a beacon.
3. $D \rightarrow C$: Request to associate (join) the network.
4. $C \rightarrow D$: Respond to the association request.

The association messages contain the addresses of D and C as well as the identifier of the WPAN. As part of this process, a special key called the *network key* is sent from the coordinator to the device in plaintext. This network key is used to secure communication destined for all devices in the network. Note that this is clearly not a secure practice. However, because data is sent at low power and the communication range of the ZigBee network is very small, it is considered "secure enough" to transmit the network key briefly to the newly joined node. It remains for the newly joined device to be authenticated to the WPAN.

6.6.2 Authentication

Before any secure communication can take place between ZigBee devices, devices must authenticate to each other. This authentication can occur between any two devices in the ZigBee network, including the coordinator. We refer to the two devices involved in authentication as Alice (A) and Bob (B). The protocol is straightforward:

1. $A \to B$: Alice sends to Bob a random string C_A, called the challenge.
2. $B \to A$: Bob sends to Alice a random string C_B, also called the challenge, with a message authentication code computed on the message

$$(02)^{16} \parallel A \parallel B \parallel C_B \parallel C_A$$

   using the network key.
3. $A \to B$: Alice sends to Bob the message authentication code, computed with the network key, of the following message:

$$(03)^{16} \parallel A \parallel B \parallel C_A \parallel C_B.$$

6.6.3 Key Establishment

Next, the ZigBee devices need to establish link keys for communicating with each other. There are three methods for establishing link keys:

1. *Preinstallation*: Keys are placed on to the devices using an out-of-band method. For example, keys may be set at the factory or entered by a user.
2. *Transport*: A trusted third party called the trust center securely sends a key.
3. *Establishment*: Devices coordinate with the trust center to establish keys with both ends of the link *without* sending the keys.

The method of Establishment is of the most interest. The protocol that ZigBee uses to obtain these keys is called the Symmetric-Key Key Establishment (SKKE) protocol. Using the SKKE protocol, Alice (A) and Bob (B) establish a link key as follows:

1. $A \to B$: Alice sends to Bob a random bit string C_A, called the challenge.
2. $B \to A$: Bob sends to Alice a random bit string C_B, also called the challenge, along with a message authentication code computed on the message

$$(02)^{16} \parallel A \parallel B \parallel C_B \parallel C_A$$

   using a key provided to Alice and Bob by the trust center. This key is called the *master key*,
3. $A \to B$: Alice sends to Bob the message authentication code, computed with the master secret, of the following message:

$$(03)^{16} \parallel A \parallel B \parallel C_A \parallel C_B.$$

Once the exchange of messages has occurred, and the message authentication codes have been verified, a link key L is derived as follows: Each party computes

$$L = MAC_{MK}(A \parallel B \parallel C_A \parallel C_B),$$

where MAC_{MK} denotes the message authentication code with key *MK*. We note that the SKKE protocol is the same as the authentication protocol, except the use of the master key and the derivation of the link key L.

6.6.4   *Communication Security*

Communications between ZigBee devices can be secured at the network layer or the application layer. In either case, the encryption uses a symmetric encryption algorithm. A link key is used if the communication to be secured is just between two ZigBee devices, and a network key is used if the message is to be broadcast to the entire WPAN.

Regardless of the key, the ZigBee protocol requires the use of AES-128 for all encryptions. The mode of operation used, however, is not a standard mode. Instead, the ZigBee protocol defines the CCM* mode of operation. From this mode of operation, an encryption method and a cryptographic hash scheme can be defined. In particular, the CCM* mode allows the user to either encrypt, hash, or both encrypt and hash. When used in the encrypt-and-hash form, the CCM* mode operates as follows:

1. Compute the hash.
   (a) Divide the message into 16-byte blocks $b_1, b_2, \cdots, b_n$. The last block may be padded.
   (b) Compute $X_i = E(K, X_{i-1} \oplus b_{i-1})$ for $2 \leq i \leq n$, where E is the encryption function and K is the key. We bootstrap this process by computing

$$X_1 = E(K, b_1).$$

   (c) The hash is the first M bytes of X_n, where M is the requested digest size.
2. Encrypt the message by constructing a stream cipher.
   (a) For each block b_i, compute a corresponding stream block s_i as

$$s_i = E(K, \text{Flag} \parallel \text{Nonce} \parallel i),$$
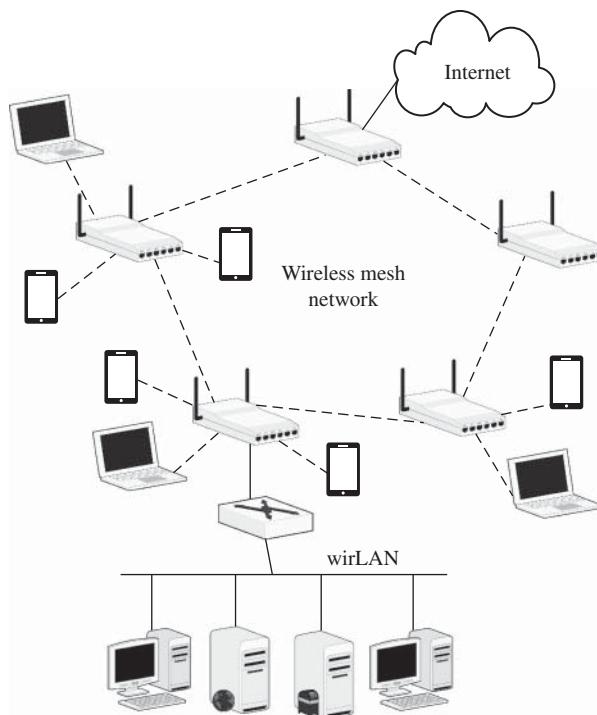
   where Flag is a set of flags (see ZigBee specification for a description of the flags) and Nonce is a random value. Moreover, the length of $\text{Flag} \parallel \text{Nonce} \parallel i$ must be exactly 128 bits.
   (b) The ciphertext corresponding to block b_i is $c_i = b_i \oplus s_i$.

6.7   Wireless Mesh Network Security

A WMN is a wireless network consisting of several wireless routers (or STAs that can forward messages to other STAs) such that for every pair of wireless routers, either they are within direct communication range or there are a sequence of wireless routers between them such that adjacent wireless routers are within direct communication range. A wireless router provides address assignment, routing, DNS lookup, and other networking services. For convenience, we may view a wireless router as an enhanced AP, and so we still use APs to denote wireless routers. A sequence of APs in which each pair of adjacent APs are within direct communication range is referred to as a wireless communication *path*.

An AP in a WMN may or may not connect to a wired network infrastructure. In a WMN, each STA is (dynamically) connected to one AP. Thus, the major difference between WMNs

Figure 6.13 A wireless mesh network

and wireless local area networks is that WLANs are star networks and WMNs are multihop networks. Figure 6.13 shows what a wireless mesh network looks like.

As a special case, we remark that an ad hoc wireless network may be formed simply by allowing both APs and STAs to work in the ad hoc mode.

Most implementations of mesh networks use two sets of radio frequencies for access points. They will typically utilize 802.11b/g for clients and 802.11a to link the APs. This reduces interference and will typically result in greater bandwidth.

Inside a WMN, an AP and all the STAs connected to it are referred to as a *region*. A region can be viewed as a WLAN. Thus, in a WMN region, we may apply the 802.11i/WPA2 security standard to authenticate STAs to the AP, to control access to the AP, and to encrypt packets between STAs and the AP.

Likewise, we may apply the 802.11i/WPA2 standard to secure communications between any two APs that are within direct communication range. Moreover, if several APs are within direct communication range pairwise, then the 802.11i/WPA2 standard may also be used to secure communications between them.

As different APs may be controlled by different administrators, transmitting and managing secret keys between different APs would require a wireless public-key infrastructure (WPKI) or a trusted wireless key distribution center (WKDC).

For convenience, we refer an AP in a WMN that relays packets as a *mesh router*. If a mesh router is allowed to join a WMN without authentication, there will be a number of

routing protocol threats and route disruption attacks. This section briefly describes several such attacks.

6.7.1   Blackhole Attacks

In a *blackhole* attack, the attacker impersonates a legitimate mesh router and drops packets it receives, instead of relaying them to the next hop, thus denying service to legitimate users. The attacker may also coax users to use his mesh router by, for example, advertising that his mesh router provides service of higher quality or faster speed.

There are two kinds of blackhole attacks. The first kind drops all packets, making the destination AP unreachable. The second kind drops selected packets. For example, it may drop a packet to certain destinations, a packet in a certain number of packets, or a packet in a certain period of time, which has the effect of downgrading communication quality.

Blackhole attacks of the second kind are also referred to as *grayhole* attacks.

6.7.2   Wormhole Attacks

In a wormhole attack, the attacker reroutes the packets sent from an STA in one region to an STA in a different region on a communication path under the control of the attacker. In particular, let D_A and D_B be two STAs in two different regions. Suppose that D_A sends a packet to D_B via a normal path of mesh routers determined by the routing protocol. The attacker may intercept this packet and then send the packet, without any modification, to D_B using a faster route. This has the effect that the attacker creates an extraneous tunnel, referred to as a *worm tunnel* between D_A and D_B controlled by the attacker. The attacher may, for example, drop tunneled packets or remove the tunnel at a later time, to disrupt communications between D_A and D_B. How to effectively and efficiently detect wormhole attacks is a major research topic.

6.7.3   Rushing Attacks

Rushing attacks are targeted at on-demand routing protocols. A typical on-demand routing protocol stipulates that each mesh router must forward the first route-request packet it receives from a source and drop the subsequent route-request packets from the same source to reduce clutter. The attacker may take advantage of this mechanism by rushing an impersonated route-request packet before the legitimate one arrives, resulting a denial of service to legitimate users.

6.7.4   Route-Error-Injection Attacks

In a route-error-injection attack, the attacker simply injects certain forged route-error packets into the network to break a normal communication path. Because typical route errors are stateless, injecting route-error packets would be less difficult to do, for it does not require dynamic information of which states the underlying routing protocol is in.

6.8   Closing Remarks

Wireless computer networks have provided new platforms for data communications. Compared to the wired networks, the wireless world has certain unique limitations. In particular, mobile STAs are typically powered by batteries, the standard wireless communication ranges are short, and wireless networks transmit packets by broadcasting. Thus, while cryptographic algorithms and infrastructures used to secure wired communications may be borrowed to secure wireless communications, there are many new issues in wireless security that require new algorithms, techniques, and infrastructures. WPA, 802.11i/WPA2, Bluetooth, and ZigBee are industry standards that provide solutions to some of these issues. However, even with these standards in place, different vendors may still introduce implementation-specific variances in their products, which may cause interoperability problems. Wireless security will remain a major playground for academic research and industry development in many years to come.

6.9   Exercises

6.9.1   Discussions

6.1. If your wireless router only supports WEP, do you trust that you are secure? Explain why.

6.2. Why is WPA more secure than WEP? What are the major differences between WPA and WPA2?

6.3. What are the major differences between the Wi-Fi security and the Bluetooth security?

6.4. What are the major differences between the Bluetooth security and the ZigBee security?

6.5. Suppose that you are asked to design wireless networks to connect fast-moving vehicles, what would be the major issues on security?

6.6. WeChat is arguably the most popular smartphone application in China today, which makes it very easy for people to communicate in groups. What do you think the major security concerns would be in WeChat?

6.9.2   Homework

6.1. Explain why in wireless network communications cryptographic algorithms should be placed in the data-link layer.

6.2. Explain why wireless communications are vulnerable to eavesdropping attacks and the denial-of-service attacks.

6.3. Explain why wireless communications are vulnerable to message-replay attacks.

6.4. If in the WEP challenge-response authentication scheme a pseudorandom challenge is reused, what will happen?

6.5. Let $P = 10011$ be a CRC_4 polynomial. Let $M = 10010110$. What is $CRC_4(M)$?

6.6. Let $P = 10011$ be a CRC_4 polynomial. Let $M = 11001010$. We note that $CRC_4(M) = 0100$. Show that the polynomial $M \parallel CRC_4(M)$ is divisible by P.

6.7. For any two strings x and y, show that

$$CRC(x \oplus y) = CRC(x) \oplus CRC(y).$$

6.8. Summarize the major security weaknesses in WEP.

6.9. Explain why WEP misuses RC4.

*6.10. The attacker would need to collect a sufficient number of packets to crack a WEP key. Using a "bit-flipping" technique to cause an STA to keep retransmitting ARP packets, the attacker may be able to collect sufficient information to crack the WEP key in about 5 to 6 minutes. In this attack, the attacker first uses a sniffer to capture the probe packets an STA sends out in searching for a legitimate AP. The attacker then configures his own AP and claims that it is the AP the STA is looking for and have the faked AP carry out a faked authentication process with the STA, making the STA connect to the faked AP. Suppose that the victim STA wants to get an IP address using DHCP, but the faked AP will not provide it. After the DHCP times out, the victim STA will use a built-in address (169.254.x.x). Once it gets an IP address, the victim STA will send out an ARP broadcast packet. The attacker then flips the bits in the encrypted ARP packet to create a new packet that is destined for the client address. This requires the attacker to know the WEP key. The victim STA sees this new ARP packet, which is still properly encrypted, and responds with a new AEP packet. The attacker will keep doing this until a sufficient number of packets are collected.

   Provide a detailed description of this attack.

6.11. Describe a birthday attack on the MIC generated by the Michael algorithm.

6.12. Explain the major differences between WEP and WPA.

6.13. Describe the decryption algorithm of CCMP used in 802.11i.

*6.14. Analyze the strength and weaknesses of 802.11i encryption algorithms.

*6.15. Analyze the strength and weaknesses of 802.11i authentication and integrity checks.

6.16. CBC-MAC is often used for authentication. Can it be used as an encryption algorithm as well? Justify your answer.

6.17. How to prevent RSN IE poisoning?

6.18. In the four-way handshake protocol, suppose that the STA and AP have finished the protocol half-way through. That is, the STA has received $message_1$ from the AP and

the AP has received message_2 from the STA. Now the attacker impersonates the AP and sends (AMAC, ANonce', *sn*) to the STA before the STA receives message_3 from the AP. Explain what will happen? Suggest an efficient way to fix the problem.

*6.19. RSN IEs are exchanged in plaintext in the four-way handshake protocol. Does this cause security concerns? If yes, how do you suggest fixing the problem?

6.20. Explain how the attacker may use the fact that authentication is done only after the association is done to launch a DoS attack to de-associate STAs from the AP.

6.21. Summarize the security parameters (e.g., encryption algorithms, MIC algorithms, key sizes, per-frame keys, and initialization vectors) of WEP, WAP, and 802.11i.

*6.22. There have been attempts to use Rainbow tables of passwords and commonly used Extended Service Set Identifiers (ESSIDs) in wireless networks to crack WPA and WPA2 keys. Describe how a Rainbow table be used to crack WPA and WPA2 keys. The Shmoo group offers tools and Rainbow tables in its Website `http://www.shmoo.com/projects.html`. Rainbow tables can be obtained from `http://umbra.shmoo.com:6969/` using BitTorrent.

6.23. Why is it difficult to detect wormhole attacks in WMNs?

*6.24. Search the literature and write a short paper describing plausible techniques to detect wormhole attacks.

6.25. Calculate the S-Boxes e and l in SAFER+.

6.26. Draw a block diagram to represent the ith round of operations in SAFER+.

6.27. Draw a block diagram to represent the output transform component.

6.28. Describe the decryption algorithm of SAFER+ and provide a correctness proof.

6.29. Draw a block diagram describing how the initialization K_{init} is generated by D_A and D_B.

6.30. Assuming K_{init} is created, draw a block diagram describing how the link key K_{AB} is generated by D_A and D_B.

6.31. Draw a block diagram describing the mutual authentication process in Bluetooth.

*6.32. Let ℓ be the length of a PIN code. If the Bluetooth PIN cracking procedure finds a candidate PIN code that passes the validation of SRES_A, what is the probability that the candidate PIN code is the original PIN code?

Likewise, if the same candidate PIN code also passes the validation of SRES_B, what is the probability that the candidate PIN code is the original PIN code?

**6.33. The PIN cracking procedure described in Section 6.5.6 requires that the attacker obtains the entire pairing and authentication session. This may not be practical. For example, if D_A and D_B store a link key K_{AB} for future use, then they do not need to

go through the entire pairing and authentication again. Thus, the PIN cracking procedure will not have sufficient information to work on. Devise a method by exploiting the connection establishment protocol in Bluetooth to force D_A and D_B, although they both have stored a link key K_{AB}, to start the entire pairing and authentication session again.

6.34. In Bluetooth secure pairing, the attacker may initiate the authentication procedure with a legitimate device over and over again and eventually guess the correct PIN. To prevent this from happening, each Bluetooth device often maintains a blacklist of bad device addresses that failed to authenticate themselves. The attacker can take advantage of this mechanism to launch a denial-of-service attack. Describe this attack.

*6.35. Read about SSP from Bluetooth v2.1 + EDR volume 2 specifications and write a short paper describing how SSP establishes pairing. Is SSP secure against man-in-the-middle attacks?

6.36. Describe a scenario where an attacker can capitalize on the fact that the Zigbee network key is broadcast in plaintext to a newly joined node.

*6.37. Research an application of Zigbee technology. Write a short paper to discuss how the security of Zigbee is influential in the adoption of the device.

*6.38. Ad hoc wireless sensor networks (WSNs) are wireless networks of wireless sensor nodes and base stations. A wireless sensor node is a small microelectronic device consisting of a sensor unit, a wireless communication unit, a battery power unit, and a programmable embedded processor. A base station may be a laptop computer or other computing devices. The sensing range of a sensor is typically modeled as a disk in the 2D space, or as a sphere in the 3D space, with the sensor located in the center. The communication range of a sensor is modeled in the same way. Communication range is typically much larger than sensing range. In an ad hoc WSN, sensor nodes may communicate with other sensor nodes within communication range, and communicate with nodes that are in the communication range through intermediate nodes. Thus, ad hoc WSNs are similar to wireless mesh networks introduced in Section 6.7. Once a sensor node collects a new data and detects a new event, it will want to transmit the information back to a base station. WSNs can be used to monitor a large set of targets spread across a geographical region, with important applications in military, intrusion detection, environmental monitoring, precision farming, and other areas.

In a WSN where sensor nodes are randomly deployed (e.g., sensor nodes may be dropped by aircrafts or shelled by artilleries), it is easy for attackers to insert their own nodes in the network. Thus, it is important to use an efficient and secure authentication mechanism to authenticate sensor nodes. Design an authentication protocol for WSNs and argue that why your design is efficient and secure.

7

Cloud Security

Data and computing during the last decade are moving from the edge of the network into the center of the network known as cloud storage and cloud computing. This platform shift has reduced the infrastructure and management cost for end users, adding an important new dimension to computing that we have known for decades. Cloud computing, different from the model of computing centers decades ago, has four basic service models. They are the software-as-a-service model, the platform-as-a-service model, the infrastructure-as-a-service model, and the storage-as-a-service model. We discuss these models in detail in this chapter. We also discuss virtualization and other technologies that enable these infrastructures.

The decrease in infrastructure costs, on the other hand, has also been traded for increased security concerns. When carrying out computation and storing data on the clouds, the security of user data and computation lies in the control of the cloud providers. This is a *significant* amount of trust placed on a third party. We discuss the security concerns and describe solutions to the security problems for each of the cloud service models. We describe access controls in untrusted clouds using the proxy re-encryption schemes. We also discuss other security issues including proofs of storage and secure multiparty computations. Finally, we describe search over symmetric encryptions for the honest-but-curious clouds and the semi-honest-but-curious clouds.

7.1 The Cloud Service Models

The concept of cloud computing can be characterized by a number of models. In particular, we may characterize clouds on the basis of four service models. They are *software-as-a-service* (SaaS) clouds, *platform-as-a-service* (PaaS) clouds, *infrastructure-as-a-service* (IaaS) clouds, and *storage-as-a-service* (*STaaS*) clouds. Each model offers unique features and incurs different security concerns. All but the last models are officially defined by NIST.

Access to the cloud in all service models is provided through Internet services. Most clouds interact with the user through the Hypertext Transfer Protocol (HTTP), following a set of design guidelines known as the Representational State Transfer architecture, which is referred

to the REST architecture or RESTful architecture. The most common REST architecture uses the uniform resource identifier (URI) and the HTTP functions of PUT, POST, and GET to provide an appropriate interface to the cloud.

7.1.1   The REST Architecture

The REST architecture capitalizes on the notion that a URI can be used to represent actions that Web servers should perform. It combines the existing HTTP infrastructure with results in the Hypertext Markup Language (HTML), the Extensible Markup Language (XML), or the JavaScript Object Notation (JSON) format.   As an example, suppose that we wish to determine the CPU time we have consumed on a cloud. Our cloud provider may inform us that the URI for our CPU time is `http://sky.cloud/user/johndoe/usage`. We can simply execute an HTTP GET request for this URI, and the CPU usage data will be returned to us in a certain format. In this case, we assume that the format is JSON.

In what follows, we use the terms of users and clients interchangeably.

7.1.2   Software-as-a-Service

The SaaS clouds provide specific applications to all users. That is, users of SaaS clouds can only use the applications provided by the clouds. Compared to the PaaS and IaaS clouds, the SaaS clouds provide the least amount of access freedom to their users for the cloud resources. These applications are maintained solely by the cloud provider, thus alleviating expensive maintenance costs for the clients.

The SaaS service is one of the oldest forms of cloud computing. For example, users of Webmail such as Gmail and Yahoo! mail are interacting with SaaS clouds. Gmail users are interacting with the SaaS cloud for Gmail and Yahoo! mail users with the SaaS cloud for Yahoo! mail. Other common applications include online social networks such as Facebook, Twitter, and Google+.

7.1.3   Platform-as-a-Service

The PaaS clouds allow users to deploy their own applications in the clouds. The cloud maintains control and manages the entire cloud infrastructure. Common PaaS clouds include Google AppEngine and a subset of Amazon Web Services (AWS).

7.1.4   Infrastructure-as-a-Service

The IaaS clouds allow users to deploy their own operating systems on top of the machines they lease from the clouds. The cloud provider maintains the hardware of its machines, but they will not provide support for the operating systems and application software the users deploy on top of them. Common IaaS clouds include the Amazon Elastic Compute Cloud (EC2).

7.1.5   Storage-as-a-Service

The STaaS model is not a cloud architecture approved by the NIST. But the technology of storage-as-a-service, also known as cloud storage, has enjoyed a growing popularity. A STaaS cloud is a cloud architecture where the cloud provider maintains a collection of bulk storage accessible through the use of a REST protocol. Common STaaS clouds include Amazon S3, Dropbox, and Google Drive.

7.2   Cloud Security Models

As in any other field of security, we must define the power of the adversary. In the setting of cloud computing, the adversary is the cloud provider itself. In what follows, we outline three adversarial models of the clouds. They are the trusted-third-party (TTP) clouds, honest-but-curious (HBC) clouds, and semi-honest-but-curious (SHBC) clouds.

7.2.1   Trusted-Third-Party

In the TTP model, users must completely trust the cloud providers. This trust is backed by the service-level agreements (SLA) with the cloud provider or by other policies offered by the cloud provider. The trust may also reside in brand recognition. No protection, however, is offered against the individuals who manage the cloud infrastructure or the perpetrators who hack in the cloud system. Nevertheless, the TTP model is the only one in practice today, despite offering the weakest form of security.

7.2.2   Honest-but-Curious

The HBC model describes a cloud that interacts with the users in the following way:

1. Honestly store the data (if needed).
2. Honestly follow the steps of the protocol.
3. Try to learn as much information as possible from the computations and interactions with the users.

Considerably more flexible than the TTP model, the HBC model itself is akin to the classic eavesdropping security in the setting of encrypted communications.

7.2.3   Semi-Honest-but-Curious

The HBC model has served well for a number of cloud-based protocols. However, it does not take into consideration the desire of the cloud to do as little work as possible. The SHBC model captures a more liberal set of requirements, where the cloud acts in the following way:

1. Honestly store the data (if needed).

2. Honestly execute requested operations *or a fraction of them*.
3. Return a nonzero fraction of the results of the operations.
4. Try to learn as much information as possible from the interactions.

7.3 Multiple Tenancy

In a cloud computing environment, a number of different clients are expected to have data or applications reside on the same hardware in the same data center. The problems that occur because of this setting are known as the *multitenancy problem*. From the security standpoint, we want to know what the tenants can learn about each other from their coexistence on the same hardware. We are also concerned with how the tenants may affect each other.

7.3.1 Virtualization

Virtualization is a technology that supports software-based emulation or provisioning of computer hardware resources. This may involve emulating the entire computer configurations so that multiple operating systems can simultaneously coexist on the same physical hardware. This process is known as constructing *virtual machines*. In particular, virtual machines are software emulators of computers, allowing a computer to run different operating systems concurrently without any modification of the operating systems it emulates. Moreover, virtual machines provide a level of isolation between applications running on different virtual machines. Both software-based virtualization and hardware-assisted virtualization are available, which present a revolutionary breakthrough in cloud computing.
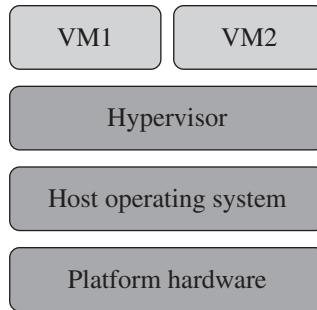
Because of the flexibility offered by virtualization, virtual machines are particularly useful for the IaaS clouds. For example, Amazon EC2 allows users to create a virtual machine that hosts different operating systems.

Virtual machines are enabled at a special layer called the *hypervisor*. The hypervisor layer is responsible for providing access to resources according to how resources are provisioned to the virtual machine. These resources could be *logical* or *physical*. In the case of a *logical* resource, an actual physical resource is shared among multiple virtual machines. In the case of a *physical* resource, the actual hardware resource is in complete control of the virtual machine it is assigned to.
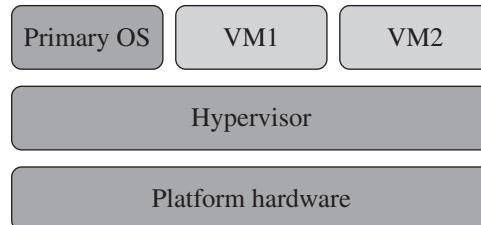
We categorize virtualization technologies on the basis of the implementation of hypervisor into *software-based virtualization* and *hardware-assisted virtualization*.

7.3.1.1 Software-based Virtualization

In software-based virtualization, the hypervisor runs on top of a true operating system (see Fig. 7.1). The operating system that runs the hypervisor is called the *host operating system*, and the emulated operating systems that run on top of the hypervisor are called the *guest operating systems*. The hypervisor is responsible for mediating access to the underlying hardware resources. Oracle Virtual Box and VMWare are examples of software virtualization.

Figure 7.1 Software-based virtualization

Figure 7.2 Hardware-assisted virtualization

7.3.1.2 Hardware-Assisted Virtualization

In hardware-assisted virtualization, the hypervisor runs as firmware, which handles the provisioning of hardware resources from the *primary operating system* (see Fig. 7.2). The primary operating system is responsible for providing the management interface to the hypervisor. Initially, the primary operating system has control of all hardware resources in the system. It then elects to relinquish control of some resources to create new guest operating systems. The Oracle VM Server for SPARC is an example of hardware-assisted virtualization. Unlike software-based virtualization, hardware-assisted virtualization requires specific hardware to run.

7.3.1.3 Hypervisor Security

Virtualization, however, also opens a door for multitenancy problems to take place, particularly in the IaaS clouds, as IaaS clouds expose virtual machines to their clients. Most hypervisors are designed with basic security protections. Recall that it is the responsibility of the hypervisor to provide an interface to partition the hardware to a given virtual machine. As the hypervisor interposes the virtual machine and hardware, it is poised to include security mechanisms. In particular, most hypervisors are able to monitor both storage- and computation-bound

operations at a coarse-grained level. For example, coarse-grained monitoring of the storage can include what disks are accessed by the virtual machine and when.

Recent advances in technologies have enabled finer-grained hypervisor security through the process of *introspection*. Introspection can be used to monitor network traffic, memory usage, process status, and other parts of a virtual machine. Using introspection, hypervisors are able to enforce security policies and perform more advanced tasks such as firewalls and intrusion detection systems. (These topics are discussed in the following chapter.) In particular, introspection enables better separations between multiple tenants of the same cloud.

7.3.2   Attacks

A number of different types of attacks can occur in a multitenant cloud. *Side-channel attacks* and *co-residency attacks* are the major types of attacks, and they often work together.

7.3.2.1   Side-channel Attacks

A side-channel attack in cloud computing is designed to learn information that leaks from normal operations of the cloud. For example, the network activity destined to a virtual machine may allow the attacker to glean the relative importance of the virtual machine.

7.3.2.2   Co-residency Attacks

A co-residency attack seeks to create a new virtual machine to co-reside on the same physical hardware with the virtual machine the attacker wishes to attack. To launch this attack, the attacker would need to exploit software bugs or exhaustively spin up virtual machines. Once an attacker successfully spins up a virtual machine that co-resides with the target virtual machine, the attacker can then launch a side-channel attack.

7.4   Access Control

Access control in the cloud is a mechanism to control user access to hardware resources. *Discretionary access control* and *mandatory access control* are the two common forms of access control mechanisms. In discretionary access control, users are given privileges and the ability to grant privileges to other users. The granting of privileges to a resource can only be given by the resource owner. In mandatory access control, an access policy is setup for all resources in the system; only the policy manager can grant and revoke privileges.

To enforce an access policy, users must first authenticate themselves to the system. Because there are several ways to authenticate a user to a system (e.g., Kerberos), we discuss them as they arise. After authentication, the access policy is enforced by the system that the user is authenticated to. For example, a typical computer system uses password-based authentication together with the operating system to provide access control to user files. In particular, Alice enters her username and password to prove her identity to the system. The file system of the operating system then enforces the access-control policy for Alice's files.

7.4.1 Access Control in Trusted Clouds

If users trust the cloud provider, then they may simply rely on the cloud to enforce access control for their files. The resources provided to the user is generally managed by the mandatory access control mechanism. Specifically, the resources in the client management interface are protected using a password-based authentication mechanism. The username serves as an assertion of the user identity, and the password serves as the proof of that assertion. Once the user has authenticated to the system, the user will be governed by the appropriate access control policy. For example, the user may be allowed to grant the resources given to him by the cloud to other users of the system.

7.4.1.1 The Open Authentication Protocol
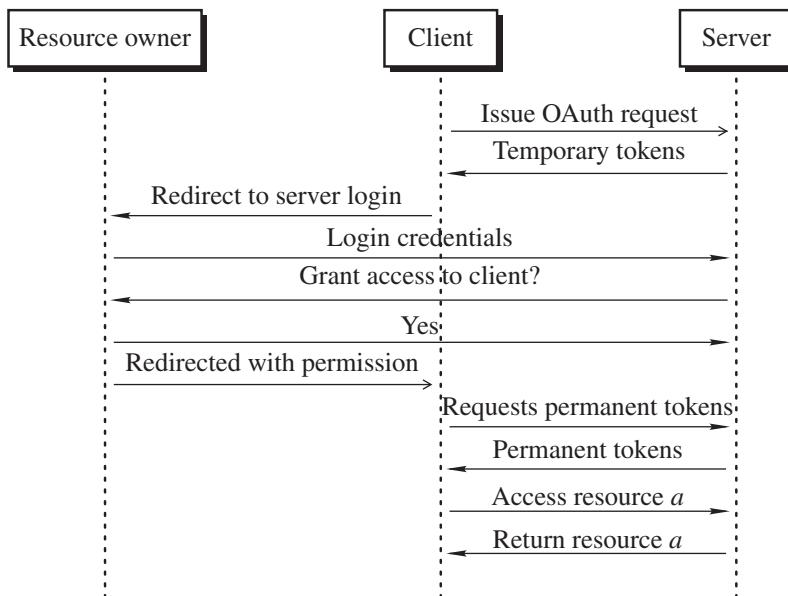
During the course of interfacing with the cloud, it may be desirable for users to delegate access to some cloud resources they are in control of without releasing their credentials. This is a form of discretionary access control. The Open Authentication (OAuth) protocol offers this ability, which works on top of the existing HTTP protocol. The protocol is described in detail in RFC 5849 (OAuth version 1.0).

In the OAuth protocol, we call the resource being protected by the authentication mechanism the *protected resource*. An example of a protected resource is a storage bucket on the Amazon S3 cloud. A storage bucket is a single object stored in the S3 cloud.

The OAuth protocol specifies three types of principals: the server, the client, and the resource owner. The server is the principal that can accept OAuth requests. The client is the principal that can create OAuth requests. The resource owner is the principal that can both access and provide access to a protected resource.

Figure 7.3 depicts the flow of the OAuth protocol at a high level, which consists of the following steps:

1. A client creates an OAuth request for temporary credentials to the server.
2. The server responds with a temporary access token.
3. The client then redirects the resource owner to a URI that includes the temporary access token. This URI points the resource owner at the password authentication screen for the resource. Logging in to the service results in a prompt for granting of permission to the client. If permission is granted, the client will be able to access the resource.
4. If the resource owner approves, they are redirected to a callback URI provided in the initial OAuth request.
5. The client is now able to request a set of tokens, from the server, using its temporary access tokens. This is done over an SSL connection. Moreover, the request is signed using a key shared between the client and server.
6. The server responds with a signed message containing the requested tokens. Again, the message is signed using a key shared between the client and server.
7. The client uses the requested token in a signed message to request access to the delegated resource. Each time the client uses the tokens, the server validates that token provided. At a later time, the resource owner may choose to revoke the token to prevent access to the resource.

Figure 7.3 An overview of the OAuth protocol

The signatures in the OAuth interactions are based on plaintext, HMAC-SHA1, or RSA-SHA1. The plaintext method signifies that no signature method is used. The latest version of OAuth is OAuth version 2.0 described in RFC 6749, which removes signatures completely. This created controversies from the security standpoint, for without signatures, the authentication protocol is subject to the following attacks. Clients can be manipulated into sending their requests to malicious servers. As long as the token is valid, this malicious server can use the token to request a resource. Specifically, if the client is directed to the wrong server that collects its credentials, then, as the requests are not signed, the server can impersonate the client and access its resources. Nevertheless, the OAuth version 2.0 protocol was standardized in 2012 by the Internet Engineering Task Force (IETF).

7.4.2 Access Control in Untrusted Clouds

In the case where clouds are untrusted, both authentication and access control must be managed without the involvement of the cloud. Cloud applications can provide their own authentication and access control to their software via conventional authentication mechanisms and access control policies. However, when we apply access control mechanisms over the data stored in the untrusted cloud, we will need to deal with new security issues.

Using basic cryptographic primitives, we can offer a simple access control policy. That is, encrypt the data before storing it on the cloud. In this case, the access control to the data is enforced by positioning the appropriate cryptographic key to decrypt the data in question. What about the case when we want to model group-level access control in the cloud? To solve this problem, we may use a cryptographic primitive known as *proxy re-encryption* (PRE). The

PRE primitive allows an HBC proxy to convert a piece of ciphertext decryptable by Alice to a piece of ciphertext decryptable by Bob, but the proxy is unable to learn the contents of the ciphertext, and Bob is not able to learn Alice's secret key. To understand how this can be done, we first consider the Blaze, Bleumer, and Strauss (BBS) proxy re-encryption system.

7.4.2.1 BBS Proxy Re-Encryption

The PRE primitive was first devised by Matthew Blaze, Gerrit Bleumer, and Martin Strauss in 1998, which uses a proxy to re-encrypt a piece of ciphertext encrypted for Alice to a piece of ciphertext encrypted for Bob. The proxy is trusted to carry out the operations according to the protocol, but there is no guarantee that the proxy will only perform what it is asked to do. Thus, it is required that the proxy be unable to see the plaintext as part of the re-encryption operations. It should also be the case that the proxy cannot recover the private key of Alice or Bob from performing the re-encryption operations. Likewise, Bob should not be able to recover Alice's private key. In this Scenario, Alice plays the role of the delegator and Bob the role of the delegatee.

PRE schemes can be characterized into *unidirectional PRE* and *bidirectional PRE*. In the unidirectional PRE scheme, the generation of a re-encryption key from Alice to Bob does not imply a re-encryption key from Bob to Alice. In the bidirectional PRE, a re-encryption key from Alice to Bob implies a re-encryption key from Bob to Alice. The BBS PRE is bidirectional and uses the Elgamal PKC cryptographic primitive. The BBS PRE scheme consists of the following components:

Setup
Select a prime number p and a generator g for the group $\mathbb{Z}_p^*$. Publish g and p as public parameters for the entire system.

KeyGen
Alice, the delegator, selects uniformly at random a positive integer $a < p$ as Alice's private key. Suppose that Bob is a delegatee. Alice then selects a positive integer $b < p$ as Bob's private key and passes b to Bob through a secret channel. Alice computes $g^a \bmod p$ as her public key, and Bob computes his public key $g^b \bmod p$.

Encrypt
Alice encrypts a message m using her public key $g^a \bmod p$ as in the Elgamal PKC:

1. Select uniformly at random a positive integer k.
2. Compute the encrypted message $(mg^{ak} \bmod p, g^k \bmod p)$.

Alice then sends the ciphertext to Eve, the proxy.

Decrypt
Alice can decrypt the ciphertext at any time using her own private key a as in the Elgamal PKC: given a ciphertext tuple $(mg^{ak} \bmod p, g^k \bmod p)$, Alice computes

$$(mg^{ak} \bmod p) \cdot (g^{ka} \bmod p)^{-1} = m.$$

ReKeyGen

Suppose that Alice wants to delegate Bob to decrypt the ciphertext using Bob's private key and she does not want to decrypt it first and then encrypt it using Bob's public key. Alice sends to Eve $\frac{b}{a}$ as her *re-encryption key* for Bob and asks Eve to re-encrypt the message for Bob using the re-encryption key.

ReEncrypt

Eve takes as input Alice's ciphertext $(mg^{ak} \bmod p, g^k \bmod p)$ and the re-encryption key $\frac{b}{a}$, re-encrypts it as

$$\left(mg^{ak\left(\frac{b}{a}\right)} \bmod p, g^k \bmod p \right),$$

and sends it to Bob. Bob can then decrypt it using his private key b as in the Elgamal PKC.

   We note that the BBS PRE system is a cryptographically weak system. For example, if the proxy and Bob collude, then they can discover Alice's private key. Improved PRE schemes (e.g., unidirectional PRE) have been devised, using a computationally expensive pairing function, which hinders the practicality of these new schemes.

7.4.2.2   Using PRE for Access Control

Consider for the moment a self-storage facility as an example. Users would want to keep people they distrust out of their storage bays, including the owners and the staff of the facility. Users further want significant control over who has access to their storage bays. They want to be able to change padlocks at any time. They may also want to delegate access to their storage bays to certain people. How do they delegate this access without compromising the aforementioned security concerns? We ponder the following situation.

   Alice stores her belongings in a UHaul self-storage facility owned by Eve, and she knows that Eve is a reliable service provider, but she may also look through unlocked storage bays to satisfy her curiosity. To prevent Eve from looking through her possession, Alice locks her storage bay with a padlock to which she holds the only key. Alice would like Bob (and possibly other people) to access her storage bay. She has three options: (1) unlock the lock herself to let Bob in; (2) designate a trusted third party to do her work; (3) make Bob a copy of her key.

   Alice, however, does not want to take any of these three options for the following reasons:

1. She cannot guarantee that she will always be available at the storage facility.
2. She does not want to use a third party, for it is not easy to find a completely trusted third party, and the third party, like herself, may not always be available.
3. She does not want to make Bob a copy of her key, for doing so would make it easy for him to distribute her key to other people for them to access her storage bay without her knowledge.
4. She wants to change padlocks at will without affecting Bob. That is, Bob does not need to know or do anything when Alice changes padlocks.

   To resolve this enigmatical situation, Alice designs a special keysafe that can hold a key to the padlock for the storage bay. This keysafe has a special locking mechanism with replaceable cylinders of three different types: the original, unlockable, and user cylinders. Alice can unlock the keysafe with the original cylinder using her own private key. She can also use her private key to replace the original cylinder with the unlockable cylinder. The keysafe with the unlockable cylinder in it cannot be unlocked by any key. The unlockable cylinder can be replaced with

the original cylinder or a user cylinder using a special key known as the swap key. Any user cylinder placed in the keysafe can unlock the keysafe with a user key specifically made for the user. User cylinders in the keysafe can only be replaced with the unlockable cylinder using the same user key and cannot be replaced with any other cylinders.

Alice constructs a user cylinder and a user key for Bob and sends to Bob his user key. She then gives the swap key and the user cylinder for Bob to Eve to keep in her office. Alice locks her storage bay with a padlock and places the key to the padlock in the keysafe. She then locks the keysafe and replaces the original cylinder with the unlockable cylinder.

When Bob wishes to access Alice's storage bay, he asks Eve for the swap key and the user cylinder for him. Bob then uses the swap key to swap the unlockable cylinder in the keysafe with the user cylinder for him, uses his user key to unlock the keysafe, and uses the key in the keysafe to unlock the padlock. When he is finished in the storage bay, Bob locks the storage bay with the padlock, places the key back in the keysafe, and replaces the user cylinder currently in the keysafe with the unlockable cylinder.

In the aforementioned scenario, Alice and Bob form a group with access to a shared resource, namely, Alice's storage bay. The confidentiality of the contents is preserved via the use of the keysafe. The group access is provided via the cylinders and the keysafe. In addition, Alice has dynamic control over who has access to her storage bay by creating a new user cylinder for her keysafe. Alice is thus always aware of who has access to her storage bay. Moreover, the fact that Bob can replace the replaceable cylinder with his user cylinder eliminates the need for a trusted third party to unlock the lock for him. Alice, of course, can change padlocks anytime without needing to change Bob's user cylinder or Bob's user key.

As far as trust is concerned, neither Alice nor Bob would place much trust in Eve. They only trust that Eve will keep Alice's swap key and user keys and give them to anybody who asked for them.

We may use PRE to realize the idea of keysafe and establish a UNIX-like group-level access control system for files stored in the cloud for the data owner. In particular, the data owner may provide group-level access control to the different groups of users in the following way:

1. Construct a public and private key pair for each group.
2. For each file to be accessed by a specific group of users, encrypt the file using a conventional encryption algorithm with a secret key K and encrypt K using the owner's public key. Upload the encrypted files and the encrypted K to the cloud.
3. To add a new user to the group, re-encrypt the encrypted K for the group, which is decryptable using the group's private key, to a ciphertext that is decryptable using the new user's private key. The cloud is in possession of the re-encryption key.
4. To read a file in the cloud, the user asks the cloud to re-encrypt the file using the re-encryption key for the group the user belongs to.

Explanations that this protocol meets what Alice's wants are left to the reader as exercise Problem 7.10.

7.5   Coping with Untrusted Clouds

After a model of the cloud has been determined, coping with the security issues can begin. In addition to the concerns around virtualized systems, we are concerned with two major security issues related to the computation offloaded to the cloud and the sensitive data stored in it.

7.5.1 Proofs of Storage

The simplest security guarantee we desire out of the STaaS cloud is the guarantee of data storage. We want the cloud to certify that what we have requested the cloud to store has actually been done as asked. We would therefore want the cloud to provide an uncorruptable and unforgeable proof guaranteeing the storage of the data. This proof of storage should be significantly smaller in size than the actual data being stored.

One way to provide a proof of storage is the challenge and response scheme, where the challenger is trying to ascertain if the responder knows some information. The password-based authentication system, for example, is the most common challenge and response system. To provide a proof of storage, the challenge–response system works as follows:

1. Challenger (the user) challenges the responder (the cloud) to prove that it possess a file f.
2. Responder returns to the challenger a short proof that it has file f. If the responder does not return a proof, then it is assumed that the responder does not have file f.
3. Challenger verifies the proof returned by the responder. If verification succeeds, then the responder has possession of file f; otherwise, the responder does not have file f.

There is a publicly verifiable proof and there is a privately verifiable proof, both relying on PKC. A proof of storage is called *publicly verifiable* if anyone knowing a client's public key can verify the cloud's storage of the client's file. A proof of storage is called *privately verifiable* if only the original client can verify the cloud's storage of the client's file.

7.5.1.1 A Proof-of-Storage Protocol

Using the mechanisms of the Diffie–Hellman key exchange, one can devise a proof-of-storage protocol. The protocol consists of a setup phase and a challenge phase. In the setup phase, the client asks the cloud to store a file f on the cloud and stores a small amount of information of fixed size on the client's own computer related to the file. In the challenge phase, the client challenges the cloud to prove that it has indeed stored the file.

In what follows, assume that n is the product of two primes and g a generator of the multiplicative group of Z_n^*.

Setup Phase:

1. The client stores $a = g^f \bmod n$ in the client's computer.
2. The client sends f to the cloud.

Challenge Phase:

1. *Challenge*: The client selects a value r uniformly at random from Z_n^* and sends g^r to the cloud.
2. *Response*: The cloud sends the proof $p = (g^r)^f \bmod n$ to the client.
3. *Verification*: The client computes $a^r \bmod n$ and compares it to p. If the two values are equal, the proof of storage succeeds; otherwise, it fails.

We note that in both the setup phase and the challenge phase of the proof-of-storage protocol, the file f is used as a number, and the time complexity of computing $g^f \bmod n$ is quadratic in the size of f. As the size of f could be *very* large, we will need to modify the protocol to make it practical, which is left to the reader as an exercise problem (see Problem 7.15).

7.5.2 Secure Multiparty Computation

Secure multiparty computation is a technique that allows multiple untrusting party to jointly compute a function of shared data without revealing their individual inputs and allows them to see the result. The problem was first considered by Andrew C. Yao in the 1980s. This notion is also useful in offloading computation to the cloud in such a way that the cloud can perform the computation but cannot learn anything beyond some function of the result. This particular case is sometimes referred to a *secure function evaluation*.

As an example of the situation, consider for the moment that two millionaires, Alice and Bob, meet on the street. They are both curious as to which of them is richer, but neither is inclined to reveal to the other his/her net worth. Using secure multiparty computation, they can both achieve this. In essence, Alice and Bob wish to jointly compute the following function

$$f(a, b) = \begin{cases} 1 & \text{if } a > b, \\ 0 & \text{otherwise,} \end{cases}$$

where a is the amount of money that Alice has and b is the amount of money that Bob has.

7.5.2.1 Garbled Circuits

One way to solve the Millionaire's problem is through the use of a garbled circuit invented by Andrew C. Yao (although Yao himself never published this result). The idea is to model the truth table of circuits in a way that obscures both the input *and* their output. The use of Boolean circuits is a natural choice, as any computation can be described using one. The following is a high-level view of how the garbled circuit protocol works:

1. *Circuit Construction*: Alice constructs a Boolean circuit (using 1 for true and 0 for false) that corresponds to the function f being computed. The circuit itself has two inputs, one for Alice and one for Bob.
2. *Garble Circuit*: Alice proceeds to "garble" the truth table for each gate of the circuit. This garbling hides the true correspondence between gate inputs and their outputs.
3. *Send Circuit to Bob*: Alice sends the garbled circuit along with the values for the input wires that correspond to Alice's input to the circuit.
4. *Bob Converts Input*: Bob uses a process called *oblivious transfer* to get Alice to give him the correct garbled values for his input *without* revealing to Alice what garbled value he is looking for.
5. *Bob Evaluates Circuit*: Bob evaluates the circuit using the truth tables and the garbled input to obtain the output. Bob then shares the output with Alice.

Table 7.1 Truth table for the exclusive-or operation

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Table 7.2 First phase of garbling the exclusive-or gate

a	b	$a \oplus b$
k_a^0	k_b^0	$k_{a \oplus b}^0$
k_a^0	k_b^1	$k_{a \oplus b}^1$
k_a^1	k_b^0	$k_{a \oplus b}^1$
k_a^1	k_b^1	$k_{a \oplus b}^0$

Garble Circuit

To garble a circuit, Alice must garble each individual gate. To illustrate the process, we use the XOR operation as an example. Recall that the XOR operation is defined by

$$a \oplus b = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{otherwise} \end{cases}$$

Suppose that we are given a truth table in Table 7.1.

Garbling occurs using the truth table representation of the operation.

Alice starts by choosing a symmetric-key algorithm and generating four keys for the algorithm, where each key represents one possible input value for each of the variables. The keys for the inputs are k_a^0, k_a^1, k_b^0, and k_b^1, where k_i^v is the key for input i having value $v \in \{0, 1\}$. In addition, there needs to be a key for every possible output value. In this case, there is only one output with one of two possible values. Thus, we have the keys $k_{a \oplus b}^0$ and $k_{a \oplus b}^1$. If we make the appropriate substitution in the truth table, then we obtain the truth table given in Table 7.2.

Alice proceeds to produce the garbled value of the circuit by double encrypting each output value with the appropriate key for b, followed by encrypting with the appropriate key for a. The resulting truth table is given in Table 7.3.

Finally, Alice removes all the headers and permutes the orders of the row to prevent information leakage on the basis of a "natural order" of gate inputs and outputs (see, e.g., the truth table in Table 7.4).

Alice has completed garbling one gate. If she needs to garble additional gates, she will use the keys for outputs to help hook the output of the gate to the input of a subsequent gate. After all the gates have been hooked to each other, the first, second, and third columns of each table are removed. Alice must maintain

- the mapping between the keys of the outputs and the values they represent;
- the mapping between the keys of the inputs and the values they represent.

Table 7.3 Second phase of garbling the exclusive-or gate

a	b	$a \oplus b$	Garbled value
k_a^0	k_b^0	$k_{a \oplus b}^0$	$E_{k_a^0}(E_{k_b^0}(k_{a \oplus b}^0))$
k_a^0	k_b^1	$k_{a \oplus b}^1$	$E_{k_a^0}(E_{k_b^1}(k_{a \oplus b}^1))$
k_a^1	k_b^0	$k_{a \oplus b}^1$	$E_{k_a^1}(E_{k_b^0}(k_{a \oplus b}^1))$
k_a^1	k_b^1	$k_{a \oplus b}^0$	$E_{k_a^1}(E_{k_b^1}(k_{a \oplus b}^0))$

Table 7.4 Final phase of garbling the exclusive-or gate

k_a^1	k_b^0	$k_{a \oplus b}^1$	$E_{k_a^1}(E_{k_b^0}(k_{a \oplus b}^1))$
k_a^0	k_b^0	$k_{a \oplus b}^0$	$E_{k_a^0}(E_{k_b^0}(k_{a \oplus b}^0))$
k_a^0	k_b^1	$k_{a \oplus b}^1$	$E_{k_a^0}(E_{k_b^1}(k_{a \oplus b}^1))$
k_a^1	k_b^1	$k_{a \oplus b}^0$	$E_{k_a^1}(E_{k_b^1}(k_{a \oplus b}^0))$

To allow Bob to learn the result of the computation, Alice must send Bob the mapping between the keys of the outputs and the values. This is optional, as one can envision cases where you do not want Bob to learn the unencrypted output (e.g., in the case of offloading a computation to the cloud).

Oblivious Transfer

To obtain the correct garbled inputs, Bob must ask Alice which key represents 0 and which key represents 1 for a given wire. However, Alice does not want to give what Bob asks. On the other hand, Bob does not want to reveal the value he is looking for either. Surprisingly, this problem can be solved using *oblivious transfer*. Technically speaking, this case is known as the 1-out-of-2 oblivious transfer, denoted by OT_2^1. Formally, an OT_2^1 protocol is a protocol where Alice has two messages m_1 and m_2, and Bob wants to retrieve one of those messages. However, Bob does not want to tell Alice what message he wants, and Alice does not want to reveal both messages to Bob. This task can be achieved using a method invented by Shimon Even, Oded Goldreich, and Abraham Lempel, and we call it the EGL protocol. Instantiated with the RSA PKC with modulus n, public key e, and private key d, the EGL protocol works as follows:

1. Alice has two secret messages m_1 and m_2 in the message space for RSA. She generates two random strings x_0 and x_1 in the message space of RSA. Alice sends to Bob the values of n, e, x_0, and x_1.
2. Bob selects a message $b \in \{0, 1\}$, which is represented by x_b. In addition, he selects a random number k in the message space of RSA. Bob sends to Alice the message $r = (k^e + x_b) \bmod n$.

3. Alice computes two values k_0 and k_1 to blind the messages m_1 and m_2, where

$$k_i = (r - x_i) \bmod n,$$
$$i = 0, 1.$$

Alice sends to Bob two messages $\text{MAC}'_0 = m_0 + k_0$ and $m'_1 = m_1, k_1$.

4. Bob takes the received messages (m'_0, m'_1) as input and uses his known choice m'_b to unmask the message m_b. This is accomplished by computing $m_b = m'_b - k$. Note that this k only works with the choice of b that Bob was committed to when he requested the message.

Garbled Circuit Evaluation

Bob evaluates the garbled circuit, provided by Alice, using the input values for a gate to decrypt the output values for a gate. This is done through a series of decryptions following the structure of the circuit. Recall that for each gate, all Bob knows is a set of ciphertexts in a random order. This means that Bob must attempt to decrypt all ciphertexts to learn the key for the gate's output wire. In a standard symmetric cipher, such as AES, all decryptions would yield a valid plaintext. There are, however, encodings Alice could use to signal that a decryption is valid. One way to achieve this is to append to the random values a significantly long fixed string such as a string of 0's. Then a decryption is valid only when that significantly long fixed string appears in the output (the string will be ignored when determining the key). To initiate the process, Bob simply uses the values on the input wires to decrypt the correct output value and then uses that value (key) as input to the next gate. When the output wires are reached, Bob maps the keys to the binary values and sends those binary values to Alice.

7.5.3 Oblivious Random Access Machines

The computation model of Random Access Machines (RAMs) is the prototypical model used for traditional single processor computers. The RAM model consists of a machine with a single CPU and a finite number of registers, a unbounded random-access memory, and primitive instructions. In this model of computation, each instruction executes in a constant amount of time, and there are no instructions that execute in parallel. Every location in the memory can be directly accessed (directly addressable) in a constant amount of time.

From the security point of view, we observe that algorithms in the RAM model leak the location in the memory that the algorithm is using. The sequence of memory locations read from or written to by an algorithm is called the algorithm's *access pattern*. By observing the access patterns of an algorithm, one can easily deduce what operations are performed and what data is valuable.

When a program is executing in the context of an untrusted cloud, we would want to hide the access pattern, so that we will not leak any information the algorithm deems important. The most naïve solution to this problem is as follows: given a RAM model with m cells of memory. For each access requested, each of the m cells is both read from and written to. Moreover, this is done in a predetermined order. Ignoring the extreme inefficiencies of such solutions, we can observe that a passive adversary is unable to determine (1) what cell is of current interest and (2) what operation was requested. We call such a scheme *oblivious*, as its accesses are the same for all input of size n.

The Oblivious Random Access Machine (ORAM) model of computation hides the access pattern from any adversary observing an algorithms execution. In particular, an ORAM is a RAM that reveals no information about an algorithm's input beyond the running time on the input. The goal in studying ORAMs is to seek efficient simulations for algorithms in the RAM model using an Oblivious RAM.

7.5.3.1 Generic Oblivious RAM Simulation

The simplest simulation of an ORAM is the "Square Root" solution. In this simulation, we assume that we have access to a probabilistic RAM. We will call the memory access requested by the algorithm a *virtual memory access*. This notion of virtual memory access differs from the notion of virtual memory in operating systems.

In the "Square Root" simulation, the virtual memory access pattern is completely hidden. Specifically, the simulation additionally hides (1) which virtual locations are accessed and in what order, and (2) how many times a particular virtual location is accessed. To achieve these properties, the simulation requires $m + 2\sqrt{m}$ cells of memory, where m is the total number of memory cells needed by the RAM model algorithm.

The memory used by the simulation is divided into three chunks, denoted by, respectively, the *main section*, the *dummy section*, and the *shelter section*.

1. The main section is the first chunk of m memory cells to provide all the memory needed by the algorithm in the RAM model.
2. The dummy section is the chunk of $\sqrt{m}$ memory cells following the main section.
3. The shelter section is the chunk of $\sqrt{m}$ memory cells following the dummy section, to serve as a form of cache for the recently accessed cells.

The simulation operates in an infinite loop as follows:

1. *Obliviously* permute the first $m + \sqrt{m}$ cells of memory in the main section and the dummy section using a random permutation.
2. Simulate $\sqrt{m}$ virtual memory accesses using the shelter section as a cache.
3. After $\sqrt{m}$ accesses, use the shelter to update the affected cells obliviously.

In what follows, we assume that each memory cell has (1) a tag field, (2) a value field, (3) a virtual address field, and (4) a shelter bit. The virtual address field will default to the virtual address that the cell represents, or ∞ if it is a dummy cell. The shelter bit will be set to 0 for all memory locations in the main and dummy sections.

Obliviously Permutation
We obliviously permute the element of memory using a random permutation π. In particular, we select a random function

$$f : \{1, 2, \ldots, m + \sqrt{m}\} \rightarrow \{1, 2, \cdots, m^2 + 2m\sqrt{m} + m\},$$

which is stored internally in the CPU. The tag for the memory cell i is $f(i)$. We then construct a permutation $\pi(i) = k$ for each cell, where $\pi(i) = k$ if and only if $f(i)$ is the kth smallest

element in $\{f(j)|1 \le j \le m + \sqrt{m}\}$. We then use an *oblivious sort* to order the memory contents in increasing order by tag.

An *oblivious sort* is a sorting algorithm in which the same compare–exchange operations are executed in the same order for every input of size n. In other words, the compare–exchange operations are independent of the elements being sorted. There are a number of such sorting algorithms, and we use a simple, asymptotically inefficient oblivious insertion sort for easy exposition. The oblivious insertion sort algorithm on array A of size n works by looping through the array with index j and then comparing all the elements at indices $i < j$ with their right neighbor at location $i + 1$. If the element at location i is greater than the element at location $i + 1$, the elements are swapped (this is called a compare and exchange operation). Once all elements at position $i < j$ are inspected, j is incremented and the process is repeated.

It is straightforward to note that regardless of the initial permutation of the elements of A, the oblivious insertion sort algorithm *always* performs the same compare–exchange operations.

We also note that $\pi(i)$ is not easy to compute. In particular, to determine $\pi(i)$, one must perform a binary search on tags to find $f(i)$. The binary search algorithm is also oblivious as you will always, in this case, perform $O(\lg (m + \sqrt{m}))$ accesses.

Simulate $\sqrt{m}$ Memory Accesses

To simulate a virtual memory access for the cell i, we first scan the entire shelter section and look for the ith cell. If the ith cell is found in the shelter, we then access one of the dummy locations not previously accessed. This should be done using the random permutation. In other words, we will access the memory cell of $\pi(m + 1)$ through $\pi(m + \sqrt{m})$. If, on the other hand, the ith cell is *not* found, then we retrieve the memory cell $\pi(i)$.

We can simulate up to $\sqrt{m}$ memory accesses. Let *mem* denote the location in memory and *count* be the number of memory accesses made.

Search for the memory location in the shelter section; if it is found, we will store the value in that memory location in a CPU register. If the memory location was found in the shelter, we access $\pi(m + count)$ using an oblivious binary search. Otherwise, we access location $\pi(i)$ directly. Finally, we iterate over the entire shelter section accessing and updating every location. Specifically, we are sure to add the currently accessed cell to the shelter. Once we exceed $\sqrt{m}$ accesses, we *must* move to the next step as the shelter is full.

Update Permuted Memory

After $\sqrt{m}$ memory accesses, the shelter must be written back to the permuted memory. This must be done obliviously. This can be done by sorting all $m + 2\sqrt{m}$ memory locations. We obliviously sort the virtual addresses in decreasing order to break ties using the shelter bit. This organizes all the duplicate virtual addresses next to each other. We then traverse the $m + 2\sqrt{m}$ memory cells and write ∞ to the second occurrence of any virtual address.

Adding Confidentiality

It is a simple addition, to any Oblivious RAM simulation, to provide confidentiality of the value stored in memory. To do this, we will use an encryption algorithm that is secure against a chosen-plaintext attack. In other words, the encryption must be CPA-Secure. A CPA-Secure cipher guarantees that there are many possible ciphertexts for every plaintext and key combination. In essence, the encryption is randomized. Examples of CPA-Secure ciphers include

Elgamal PKC and any block cipher in CBC or OFB mode using random initialization vectors. The modifications would mean that every write to memory would require the writing of a new ciphertext to each cell of memory accessed as a result of the write operation. In the case of the "Square Root" simulation, this would mean that $\sqrt{m}$ writes would be performed for each write. This is the case as only the shelter section is written to in the course of a normal access.

7.6 Searchable Encryption*

Imagine for the moment that Alice has a large collection of documents, $\mathcal{D}$, that she wishes to store in a distributed storage environment owned by Bob. Bob has been known to be nosy, which means that Alice must encrypt all the documents in her document collection *before* uploading them to Bob's distributed storage environment. Assume, now, that Alice wants to read the documents in $\mathcal{D}$ that contain a certain word or phrase. What does she do? Trivially, she could ask Bob to send her all the files, decrypt them locally, and then search for the documents that contain the information she is looking for. Retrieving all the files and then decrypting them, however, will incur a great cost in both communication and time. It would be far more efficient, for Alice, if Bob could perform the search and only send her the documents that match her queries. Alice's problem is known as the searchable encryption problem.

Dawn Xiadong Song, David Wagner, and Adrian Perrig, in 2000, offered the first glimpse of a solution to Alice's problem under the HBC model. They introduced Searchable Symmetric Encryption (SSE). This new SSE construction allows for Alice to ask Bob to query the encrypted document collection for a specific word or phrase. Alice enables Bob to perform the search by providing Bob, at query time, with some special information known as a trapdoor. Bob then returns the results of the query to Alice. The guarantees that they provided are that the queries remain unknown to Bob (query privacy) and any information beyond the number of results and size of the encrypted documents is unknown to Bob (query result privacy).

Although not its original intention, we can adapt the searchable encryption to cloud storage. We assume that a collection of encrypted documents, $\mathcal{D}$, is stored in the cloud such that a search query can be executed over all the documents in the collection. The cloud is responsible for both executing the query and returning the results. We have the added security guarantee that the cloud should be unable to learn the nature of the query. If one uses only symmetric cryptography in the solution, the problem is called the Symmetric Searchable Encryption problem.

7.6.1 Keyword Search

The simplest type of query that Alice could send to Bob is a query for what documents contain a certain keyword. This problem is known as keyword search. It has been extensively studied in the context of both symmetric and asymmetric cryptography. In this section, we consider example constructions on the basis of symmetric cryptography. For solutions based on asymmetric cryptography, we point the reader to work by Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano from 2004.

7.6.1.1 Basic Symmetric-Encryption-Based Techniques

We consider two types of searches distinguished between *hidden searches* and *nonhidden searches*. In a hidden search, the query submitted to the cloud is constructed in such a way that the cloud is unable to ascertain the meaning of the query (i.e., query privacy). In a nonhidden search, the query is known to the cloud. For our application, we will only be concerned with hidden searches, as we do not want to reveal the keyword to the cloud.

We use the following assumptions:

1. A document d consists of a sequence of words.
2. There exists a family of pseudo-random functions $F_{k_i} : \{0,1\}^{n-m} \rightarrow \{0,1\}^n$, for any n and m. A pseudo-random function is a member of the family of functions where the behavior of one function, drawn randomly from the family, is computationally indistinguishable from any other random function.
3. There exists a family of functions as a set of keyed functions $F : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^l$, where $k, n, l > 1$.
4. There exists a family of pseudo-random permutations (encryption function) $E_{k_i} : \{0,1\}^n \rightarrow \{0,1\}^n$, for any n. If the domain and range of a pseudo-random function are the same, we have a notion called a pseudo-random permutation.
5. There exists a family of pseudo-random generators G with output contained in $\{0,1\}^m$, for any m. A pseudo-random generator is a function provided with an n-bit input that expands its input to a longer sequence in a way that the distribution generated by the pseudo-random generator is computationally indistinguishable from being truly random.
6. $f_{k'} : \{0,1\}^* \rightarrow \mathcal{K}$ is a pseudo-random function that maps arbitrary binary strings to a key space $\mathcal{K}$.

Using these functions and the document collection $\mathcal{D}$, we construct an SSE system that consists of two basic operations Encrypt and Search. The Encrypt operation encrypts a document in such a way that at a later time, the cloud can run Search and obtain an answer to a keyword query. The scheme requires two secret keys k' and k'' maintained by the owner of the collection.

Encrypt
For each word w_i of n-bit long in the document d, the data owner does the following:

1. Encrypt word w_i as $x_i = E_{k''}(w_i)$.
2. Split x_i as $L_i \parallel R_i$, where $|L_i| = n - m$ and $|R_i| = m$.
3. Choose $k_i = f_{k'}(L_i)$.
4. Generate a pseudo-random value s_i for word w_i using the pseudo-random number generator G with output length $n - m$.
5. Let $T_i = s_i \parallel F_{k_i}(s_i)$, where F outputs a binary string of length m.
6. Write $C_i = E_{k''}(w_i) \oplus T_i$ to the file that will be uploaded to the cloud.

Essentially, the Encrypt operation encrypts each word in the document and then encrypts the encrypted word with a stream cipher. Recall that a stream cipher can be generated using a

block cipher under the CFB or OFB mode of operations. Similarly, a stream cipher can also be generated using a pseudo-random number generator.

Search

To search for a keyword w, the data owner sends a query consisting of $x = E_{k''}(w)$ and $k = f_{k'}(L)$ to the cloud, where L is the leftmost $n - m$ bits of x. The cloud proceeds as follows for every word in the document d:

1. Compute $T_i = C_i \oplus x$
2. Parse T_i as $s \parallel v$.
3. If $v = F_k(s)$, then the word is found and stop. Place the document d is in the list of documents to send back to the data owner.

Essentially, the Search operation uses the encryption of the keyword to serve as a token. The key is needed for computation of the pseudo-random function. The pseudo-random function itself serves as a check condition for a match. Statistically speaking, the only way the search token will match is if the pseudo-random function applied to the $n - m$ bit string matches with what is computed by XORing with the token.

While this system achieves the basic properties of SSE, we note that it leaks statistical information including the access pattern and the results. Moreover, this scheme also suffers from efficiency. If we want to find all documents in the collection $\mathcal{D}$ that have keyword w (expressed by $\mathcal{D}(w)$), we must perform a search with complexity linear in the number of documents and words in each document.

7.6.1.2 Index-Based Approaches

To move away from linear search in the size of the document collection, we first assign a unique numeric identifier to every document in the collection $\mathcal{D}$. We can then use an *inverted index* to reduce the search complexity. An inverted index is a data structure that maps a keyword to a document identifier that contains the keyword. Essentially, every entry in the inverted index pairs the keyword w with $\mathcal{D}(w)$. Therefore, when an inverted index is searched for a keyword, all identifiers of documents that contain the keyword are returned *without* consulting the actual document collection $\mathcal{D}$. To use the inverted index for SSE, we must encrypt the index in some form. We call this form of the inverted index the *encrypted index*.

We use the following four algorithms to construct an indexed-based searchable symmetric encryption:

1. KeyGen: which is used to generate any secret keys needed by the system.
2. BuildIndex: which is used to construct the encrypted index.
3. TrapGen: which is used to generate a trapdoor of the keyword. The trapdoor allows the cloud to search for the keyword in the encrypted index *without* revealing the keyword.
4. Search: which takes a trapdoor and an index and returns the identifiers of all documents that match the keyword. This is the only operation run by the cloud.

As an example of inverted index, we construct a lookup table that matches a keyword w to a list that represents $\mathcal{D}(w)$. The construction requires

- A pseudo-random function F that maps a word to a binary string of length that is a polynomial in the size of the largest document set. The size of a document set is the total number of words contained in the document set.
- A pseudo-random function G that maps a word to a binary string of length k.

The four algorithms are instantiated as follows:

- KeyGen: Generate three keys K_1, K_2, and K_3. The keys are pseudo-random binary strings of length k. Every document in $\mathcal{D}$ is symmetrically encrypted with key K_3.
- BuildIndex: Construct the index using a lookup table T, which is a dictionary data structure. In particular, for each word w in the dictionary Δ:
  1. Create a search key $k_w = G_{K_2}(w)$ for word w.
  2. Construct the list of document identifiers of documents that matches keyword w, and XOR the list with the output of pseudo-random function F on the word w. In other words, compute
  $$L_w = \mathcal{D}(w) \oplus F_{K_1}(w).$$
  3. Store L_w in T using key k_w.

  The encrypted index is then sent directly to the cloud along with the document collection encrypted under key K_3.
- TrapGen: The client sends the pair $\tau = (F_{K_1}(w), G_{K_2}(w))$ to the cloud as a trapdoor, which allows the cloud to search the index.
- Search: Given the trapdoor τ and the index T, the cloud parses τ as (f, g). The cloud then accesses location g in T, denoted by $T[g]$, and returns $T[g] \oplus f$ to the client. At this point, the cloud will either return the list of document identifiers to the client *or* return all the encrypted documents to the client. The decision is an implementation choice. Once the user has the encrypted documents that match the query, the client has to just use K_3 and the decryption algorithm to retrieve the plaintext.

This indexed-based mechanism is significantly more efficient than the nonindexed approach. The system, however, is not without flaws. Firstly, it leaks the access pattern akin to the nonindexed approach. Secondly, it has a drawback that documents cannot be dynamically added to the collection. The only way to add documents to a collection is to publish a new encrypted index. The nonindexed approach does not have this drawback.

7.6.2   *Phrase Search*

The process of searching for phrases over encrypted data was first considered by Yinqi Tang, Dawu Gu, Ning Ding, and Haining Lu. They solved the problem by presenting a two-phase protocol to handle the search over the encrypted data. In the first phase, the cloud retrieves the document identifiers for documents that contain all the words in the phrase provided by the client and returns the identifiers to the client. This phase relies on a global index, namely, the index shared among all documents in the cloud. In the second phase, the client sends the phrase query and a list of document identifiers to the cloud. The cloud searches for an exact phrase match for each document in the per document index (phrase table) and returns to the client the actual encrypted documents that match the phrase.

We first construct a phrase table for each document, allowing the cloud to determine if the phrase occurs in a specific document without learning what the phrase is. To construct the table, we must assume the existence of the following three keyed pseudo-random functions:

$$\Psi : \{0,1\}^{\lambda} \times \{0,1\}^{*} \rightarrow \{0,1\}^{n},$$
$$h : \{0,1\}^{\lambda} \times \{0,1\}^{*} \rightarrow \{0,1\}^{u},$$
$$f : \{0,1\}^{\lambda} \times \{0,1\}^{*} \rightarrow \{0,1\}^{\lambda}.$$

The table has the dimensions of $w_c \times (d+1)$, where w_c is the number of distinct words in the document and d is the highest frequency (for any word) that occurs in the document collection $\mathcal{D}$.

We then construct a phrase matching lookup table as follows:

1. Associate a random number r_i with the ith word in the document.
2. Store in the first column of the lookup table the value $\Psi_z(w_i \parallel \mathsf{id}(D))$, where $\mathsf{id}(D)$ denotes the identifier associated with document D.
3. For each remaining element of the lookup table, store $h_s(r_{i-1}) \parallel r_i$ if the word r_{i-1} precedes r_i. It is required that key s be distinct for two coherent words (i.e., $s = f_k(w_{i-1} \parallel w_i \parallel \mathsf{id}(D))$). The first word in the document is handled in a special way by computing $h_s(r^*) \parallel r_i$, where r^* is a random number.
4. After all of the relationships are placed in the lookup table, all unfilled slots are filled with random numbers of the same size as the output of h_s and the size of the random number.
5. Finally, permute the contents of each row in the table (starting from the second element) and sort the rows on the basis of the first element of each row.
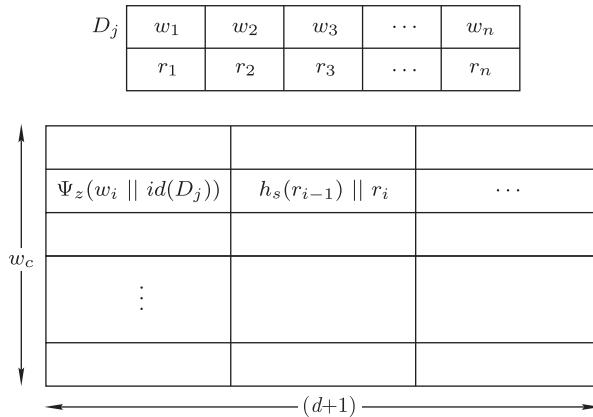
An example of the construction is given in Figure 7.4 for document D_j

To search this phrase lookup table, the cloud will use each $\Psi_z(w_i \parallel \mathsf{id}(d))$ value in the order it appears. The $\Psi_z(w_i \parallel \mathsf{id}(d))$ part of the trapdoors is constructed by the client as part of conducting the phrase search. The cloud proceeds as follows:

1. Use binary search to find the row for $\Psi_z(w_i \parallel \mathsf{id}(d))$.
2. Search the row for $f_k(w_{i-1} \parallel w_i \parallel \mathsf{id}(d))$. If there is a word in the phrase that is not found, return false. If all pieces of the phrase are found, return true.

7.6.3 Searchable Encryption Attacks

There are two major types of attacks on searchable encryption. They are the *chosen-query attack*(CQA1) and the *adaptive chosen-query attack*(CQA2). In a CQA1 attack, we assume that the cloud (as an adversary) is allowed to query the index using queries that do not depend on the results of previous queries. In a CQA2 attack, we assume that the cloud is allowed to make adaptive queries to the data structure. In either case, we seek to guarantee that the cloud can only learn the access pattern (what was accessed) and the results of the queries (the set of document identifiers that match the search). Everything else is to be kept secret.

Figure 7.4 An example of a phase matching lookup-table-based index

A guarantee against CQA1 attacks is a relatively weak security guarantee. One can think of this security model as the case where the client issues batch queries. A guarantee against CQA2 attacks is a much stronger notion of security over that of CQA1.

There are examples that satisfy both security models. For example, the phrase search scheme is a CQA1-secure system. At the time of this writing, there is no known CQA2 secure encrypted phrase search scheme. But the indexed-based keyword search system is a CQA2-secure system.

7.6.4 Searchable Symmetric Encryptions for the SHBC Clouds

The solutions presented previously for searchable symmetric encryptions are for the HBC cloud model. The first searchable symmetric encryption scheme secure in the SHBC cloud model is due to Qi Chai and Guang Gong in 2012. The system we present here is the corrected system made by the authors of this book. We often refer to a scheme secure in the SHBC model as a verifiable scheme. Formally, we define the scheme as follows:

Definition 1 (Verifiable SSE). A Verifiable SSE scheme is a collection of five polynomial-time algorithms

$$\mathsf{SSE} = (\mathsf{Keygen}, \mathsf{BuildIndex}, \mathsf{Token}, \mathsf{Search}, \mathsf{Verify}).$$

The explanations of these algorithms are given as follows:

$\mathsf{Keygen}(1^\lambda)$ is a probabilistic key generation algorithm run by the data owner O (who is also the client). It takes a unary notation of λ (as security parameter) as input and returns a secret key K_O such that the length of K_O is polynomially bounded in k.

$\mathsf{BuildIndex}(K_O, \mathcal{D})$ is a (possibly probabilistic) algorithm run by the data owner O. It takes as input the secret key K_O and a document collection $\mathcal{D}$ that is polynomially bounded in λ and returns an index $\mathcal{I}$ such that the length of $\mathcal{I}$ is polynomially bounded in λ.

$\mathsf{Token}(K_O, w)$ is run by the data owner O. It takes the secret key K_O and a word w as inputs and returns a token T_w.

Search$(\mathcal{I}, T_w)$ is run by the cloud S. It takes an index $\mathcal{I}$ for a collection $\mathcal{D}$ and the Token T_w for word w as inputs and returns the set of identifiers of documents containing w, denoted by $\mathcal{D}(w)$ and proof of correctness proof$(\mathcal{D}(w))$. The proof must be polynomially bounded in λ and unforgeable.

Verify$(\mathcal{D}(w), \text{proof}(\mathcal{D}(w)), w, K_O)$ is run by the data owner to verify that the results in $\mathcal{D}(w)$ are correct via proof proof$(\mathcal{D}(w))$. The Verify algorithm will return true if the proof is correct and false otherwise.

7.6.4.1   Tries

The keyword indexing mechanism used in this scheme is a data structure called trie, which was devised by Edward Fredkin in 1960. It supports two main operations Insert and Search. Both operations take a word $w \in \Sigma^*$ as input. A trie is a $|\Sigma \cup \{\$\}|$-ary tree, where $\$ \notin \Sigma$ is a special symbol. Each internal node of the tree is labeled with an element of Σ, and each leaf node is labeled with $\$$. A root-to-leaf path through the tree denotes a word $w \in \Sigma^*$.

The Insert operation appends a $\$$ to the input w. Starting at the root node of the tree, we use w to create a path. The first time we reach a node that does not have the current corresponding letter in w, we add a subpath as a child to the current node. Moreover, we label this subpath appropriately with the remaining letters of w, terminating the path with a $\$$.

The Search function uses input w as a path through the tree. The function first adds a $\$$ to the path. If that path ends in a leaf, that is, if the path is a root-to-leaf path, then the search is successful. Otherwise, the word does not exist in the dictionary.

7.6.4.2   A Privacy-Preserving Trie

In what follows, we denote a trie by T and a node by $T_{i,j}$, where j is the depth of the node and i the left-to-right placement of the node. We denote the access to values stored in a node of T by $T_{i,j}[s]$, where s denotes the name of the field. We denote the parent of a node $T_{i,j}$ by parent$(T_{i,j})$.

To build a privacy-preserving trie, assign each node with three fields: l, h, and e, where l holds the symbol in Σ of the given node, h stores a globally unique value for the node, and e stores a bit map of the children of the node. In the case that a node is a leaf, the field e actually holds a list of identifiers of documents that contain the word on the path from the root to the leaf. This e field will often be described as a "verification tag." The construction requires a keyed hash function $\mathcal{F} : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^z$, a semantically secure block cipher $(G, \mathcal{E}, D)$, and a function ord $: \Sigma \rightarrow \mathbb{Z}^+$ which, when given a letter in Σ, returns the index of the associated letter in the alphabet Σ.

We first use the Insert operation of the trie data structure to insert every word appearing in $\mathcal{D}$ into the trie. Next, in a level order traversal of the trie, every internal node $T_{i,j}$ has its h field set to

$$\mathcal{F}_{k_1}(l \parallel j \parallel \text{parent}(T_{i,j})[h])$$

and its e field set to

$$E_{k_2}(h \parallel b),$$

where b is the bitmap of the children of the node. Every leaf node $T_{i,j+1}$ has its e field set to

$$(\mathcal{D}(w) \parallel E_{k_2}(h \parallel \mathcal{D}(w)))$$

for word w that is represented by the root-to-leaf path through the trie. The leaf node also has its h field set to $\mathcal{F}_{k_1}(\$ \parallel j + 1 \parallel \mathsf{parent}(T_{i,j+1})[h])$. Each node of the trie permutes its children and removes its associated symbol stored in l. Figure 7.5 shows the construction of a privacy-preserving trie on the dictionary $\Delta = \{\mathrm{cat}, \mathrm{car}, \mathrm{do}, \mathrm{dog}\}$.

Given a method for constructing a privacy-preserving trie, the five operations for the Verifiable SSE scheme is as follows:

Keygen(1^λ):
Select a key k_1 uniformly at random from $\{0, 1\}^\lambda$ and use G to generate a key k_2 of λ bits. Return the key $K_O = (k_1, k_2)$.

BuildIndex($K_O, \mathcal{D}$):
Using K_O and $\mathcal{D}$, construct a secure trie, $\mathcal{I}$, as described in section 7.6.4. Return the tuple $(\{E_{k_2}(d_i) | d_i \in \mathcal{D}\}, \mathcal{I})$.

Token(K_O, w):
The client generates a privacy-preserving query π for the cloud to use in searching for the keyword in the index. The query π for a word w is constructed by setting $\pi_i = \mathcal{F}_{k_1}(w_i \parallel i \parallel \pi_{i-1})$ for $i \geq 1$, where w_i is the ith letter in word w. The value π_{i-1} is the hash of (1) the previous character in the word, (2) its position in the word, and (3) the its parents hash value. We bootstrap this by setting $\pi_0 = 0$. The value $T_w = \pi$ is returned. This means that we are essentially building a chained hash along the root to leaf path that exist in the nodes of the trie.
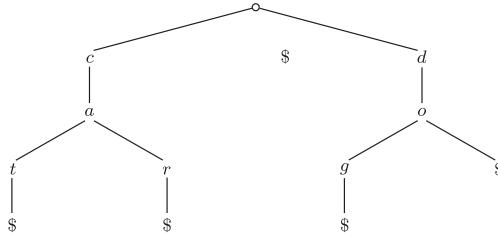
Search($\mathcal{I}, T_w$):
To search the index for the token T_w, we use the Search procedure of a Trie matching the h values along the root-to-leaf path. At each node $T_{i,j}$, along the root-to-leaf path, we add to $\mathsf{proof}(\mathcal{D}(w))$ the value $T_{i,j}[e]$. When we reach the leaf, we have recovered $\mathcal{D}(w)$. If the leaf node is reached, return $(\mathcal{D}(w), \mathsf{proof}(\mathcal{D}(w)))$. Otherwise, return $(\perp, \mathsf{proof}(\mathcal{D}(w)))$.
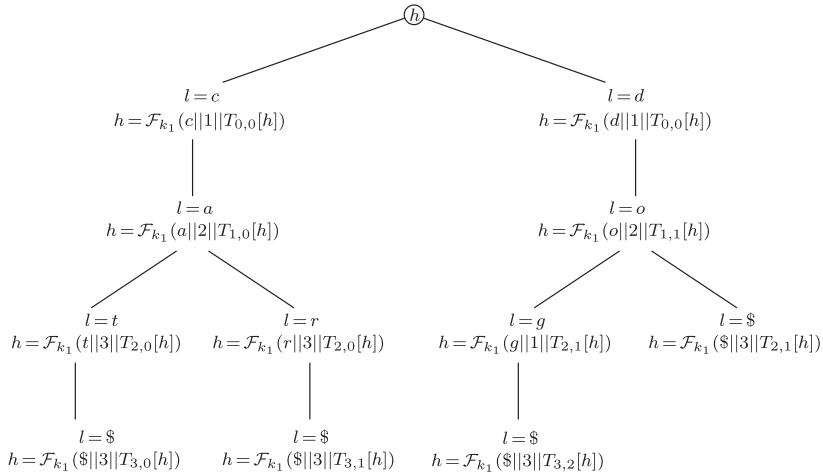
Verify($\mathcal{D}(w), \mathsf{proof}(\mathcal{D}(w)), w, K_O$):
To verify the results, we first call $T_w = \mathsf{Token}(K_O, w)$ to generate a token $T_w = \pi$. For each t_i in the sequence $\mathsf{proof}(\mathcal{D}(w)) = \langle t_1, t_2, \ldots t_\ell \rangle$:

1. If $i \neq |w| + 1$, then compute $(r_1, b) = D_{k_2}(t_i)$. Otherwise, parse $t_{|w|+1}$ as $\alpha \parallel \beta$ and compute $(r_1, \delta) = D_{k_2}(\beta)$.
2. Verify that $r_1 = \pi_i$.
3. If $i \neq |w| + 1$, then verify that $b[\mathsf{ord}(w_{i+1})] = 1$. Otherwise, verify that $\alpha = \delta$ (i.e., both are $\mathcal{D}(w)$).
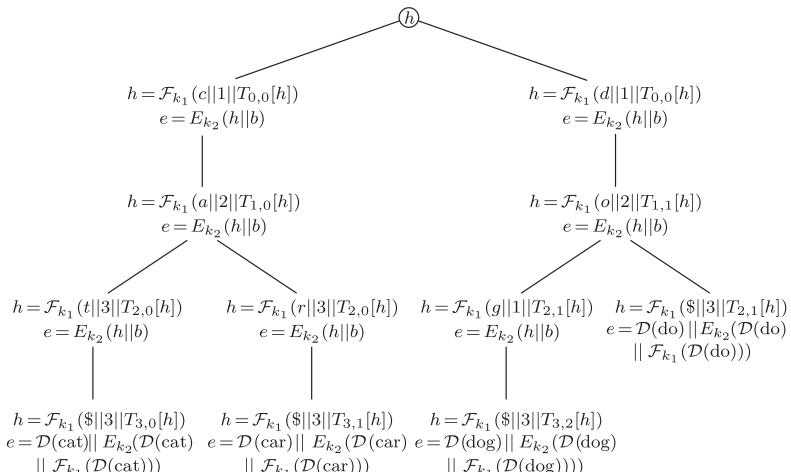
Assuming that $\mathcal{D}(w) = \perp$, it should be the case that Step 3 will fail when processing tag t_ℓ. If it does, Verify returns true. Otherwise, Verify returns false.

(a) Insert the words "cat", "car", "dog", and "do" into the trie

(b) Using a level order traversal, construct the appropriate hash chains

(c) Using a level order traversal, construct the appropriate verification tags and delete the values in the l fields

Figure 7.5 Steps for constructing an encrypted trie

Assuming $\mathcal{D}(w) \neq \perp$, the Verify routine only returns true when the following conditions are met:

- $\ell = |w| + 1$.
- Parsing $t_{|w|+1}$ as $(\mathcal{D}(w), \mathcal{F}_{k_1}(\mathcal{D}(w)))$ results in $\mathcal{D}(w)$ matching $\mathcal{F}_{k_1}(\mathcal{D}(w))$.
- Step 2 and 3 do not fail for any t_i where $i < \ell$.

Otherwise, Verify returns false.

7.7 Closing Remarks

In this chapter, we toured the various parts of cloud computing where security is a concern. We investigated the basics of virtual machine architecture and the attacks that capitalize on that architecture. We considered passive side-channel attacks. We concluded by looking at clouds we do not fully trust and how to deal with those untrusted clouds. Cloud security will continue to be a major concern.

7.8 Exercises

7.8.1 Discussions

7.1. Describe your experience in cloud computing and indicate which cloud service providers you have worked with.

7.2. Describe a few concrete examples of STaaS applications and the security concerns of these applications.

7.3. Why is the SHBC model a better model from the security point of view? Can you think of a more realistic cloud model?

7.4. In the PRE scheme, is the proxy necessary and why?

7.5. Can you think of a STaaS application where providing nonadaptive security is sufficient?

7.6. Think about how to generalize search over encrypted data to relational databases.

7.8.2 Homework

7.1. Provide an example of when you would want to use each of the following clouds:
  - (a) A SaaS cloud.
  - (b) A PaaS cloud.
  - (c) An IaaS cloud.

7.2. Many companies are hesitant to move their operations into the cloud, even if the provider has a service-level agreement. Explain why this is the case and justify your answers.

7.3. Install an operating system virtualization product such as Oracle Virtual Box. Experiment with installing and managing a guest operating system.

7.4. Describe an attack that leverages the multitenancy of clients in the cloud infrastructure.

7.5. Discuss what benefits there are from using hardware-assisted virtualization versus pure software virtualization.

7.6. Conduct a research on a REST API for cloud storage products such as Dropbox (`http://www.dropbox.com`), and write a report of up to 4000 words.

7.7. Give an example of a side-channel attack that a virtual machine co-resident with another may perform.

7.8. Conduct research on a recent co-residency attack and write a report of up to 4000 words.

*7.9. Research and implement a library interposer in Linux that audits `fopen` calls. Your program should record what file was opened *and* the mode of access used to open it. *Hint*: You will find the man pages for `dlopen`, `dlsym`, and `dlclose` helpful. You will also find it necessary to read up on the LD_PRELOAD shell variable.

7.10. Explain why the PRE protocol for access control meets Alice's needs. In particular, explain which components correspond to the keysafe, various types of cylinders, and keys. Justify your answers.

*7.11 In the chapter, it was mentioned that there exist unidirectional PRE schemes that use bilinear pairings. Research bilinear pairings and show how you can recover a session key in the Diffie–Hellman protocol by eavesdropping if you have a bilinear pairing.

7.12. Is the Linux file permission system an example of discretionary or mandatory access control? Justify your answer.

7.13. Describe why computer security experts might disagree on removing the signatures from the OAuth protocol in OAuth 2.0.

7.14. Recall the PRE protocol discussed in the chapter. Is it possible for the proxy server and another user to collude (work together) to recover the delegator's private key? If so please explain the attack.

7.15. The Diffie–Hellman-based proof of storage protocol presented in the chapter treats the file f as a *very* large number. It was mentioned in the text that this is impractical. Using a cryptographic hash function, describe a simple modification to this protocol to make it more practical.

7.16. Construct the garbled circuit for each of the following gates:
  (a) The AND gate.
  (b) The OR gate.
  (c) The NAND gate. Recall that A NAND B is equivalent to $\overline{A \wedge B}$.

7.17. Is bubble sort an oblivious sorting algorithm? Justify your answer.

7.18. Research and describe a different oblivious sorting algorithm that can be used in the ORAM simulation we discussed. Is your new algorithm more efficient? Justify your answer.

7.19. We define the *overhead* of an ORAM simulation by the ratio of the number of accesses done by the ORAM to the number of original accesses. If the Square-Root Solution is used, what is the overhead? Justify your answer.

7.20 Why are we concerned about leaking access patterns when we deal with data? What can the attacker do with this access pattern?

7.21 The text states that the searchable encryption system for keyword search devised by Song, Wagner, and Perrig is not very efficient. Explain why and justify your answer.

7.22. Why is it desirable to have a CQA2-secure searchable encryption scheme opposed to a CQA1-secure searchable encryption scheme? Justify your answer.

7.23. It is known that Oblivious RAMs provide the most secure solution the SSE problem. Devise a solution to the SSE problem using an Oblivious RAM.

7.24. Why are researchers still looking for solutions to the SSE problem if the most secure solution is already available using Oblivious RAMs?

7.25. In the original privacy-preserving trie construction due to Qi Chai and Guang Gong, the leafs of the tries had e values of $\mathcal{D}(w) \parallel \mathcal{F}_{k_1}(\mathcal{D}(w))$ for word w. Recall that w is the root-to-leaf path through the trie. Describe an attack that allows the cloud to return a different bitmap, $\mathcal{D}(w)$, that still verifies.

7.26. After solving problem 7.25, do you see why the values kept in the e fields of the leaves prevent this attack?

7.27. In the chapter, we discussed 1-out-of-2 oblivious transfer. There is also 1-out-of-n oblivious transfer where the Bob gets only one value out of a possible n values without telling Alice which one. Research an application for 1-out-of-n oblivious transfer.

*7.28. Implement the privacy-preserving trie, as described in this chapter, in your language of choice.

7.29. In the access control mechanism we described in Section 7.4.2, the cloud must posses the re-encryption key as well as perform the re-encryption. It may be the case that we instead wish for the cloud to only archive the re-encryption keys and have the clients re-encrypt the ciphertexts on their own. Identify and discuss a security flaw that appears if this is allowed.

7.30. We say that a PRE scheme is *nontransferable* if the proxy and a set of colluding delegates cannot re-delegate decryption rights. In other words, they cannot produce a valid re-encryption key. Demonstrate that the BBS PRE scheme does have the non-transferability property.

8

Network Perimeter Security

Local area networks (LANs), personal area networks (PANs), wireless local area networks (WLANs), and wireless sensor networks (WSNs) exist on the edges of the Internet. These edge networks that are contained within individual organizations and households can be found at various locations across the Internet. In the early days of the Internet when most users were researchers, setting up a strong defense mechanism to protect an edge network was not a priority. However, within the current Internet that is much larger and deeper, edge networks with no protection or with limited protection are invaded time and time again by malicious intruders. In some cases, the intruder simply walks right into an unprotected edge network, and at other times, the intruder finds a way to break into one that is weakly defended.

Protecting an edge network against intruders, regardless of how well each individual computer is protected, is similar to protecting a city against intrusions in ancient times. In those days, a fortified wall was built around the city as a barrier separating the inside from the outside. There were three layers of defense. The first layer of defense to protect the internal networks was *perimeter security*, where entrance and exit points were reduced to only a few, and armed guards were posted at each of these points to check and question people when they tried to enter or leave the city. People who did not possess the appropriate documents would be stopped from entering or leaving the city. The second layer of defense was *street patrolling*, where armed guards were scheduled to patrol the city streets to identify intruders who somehow got pass the entrance points. The third layer of defense was *house cleaning*, where security specialists checked individual homes to remove foreign objects and mend security loopholes.

The *firewall* technology, the intrusion detection systems, and the anti-malicious-software technology are, respectively, successful adaptations of perimeter security, street patrolling, and house cleaning in the era of computer networks. The firewall technology allows system administrators to focus on a few powerful computers on which they can set up strong defense mechanisms to check incoming and outgoing packets. These computers are placed at entrance and exit points of the edge networks to protect host computers against malicious packets. This chapter presents the basic principles of firewalls and common firewall configuration schemes. Intrusion detection systems and the art of anti-malicious-software are discussed in the following two chapters.
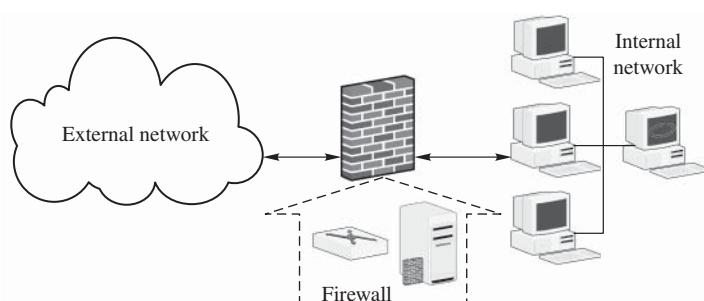
8.1   General Firewall Framework

Firewalls are needed because encryption algorithms cannot effectively stop malicious packets from getting into an edge network. This is because IP packets, regardless of whether they are encrypted, can always be forwarded into an edge network. Authentication algorithms, on the other hand, can be used to help determine whether an incoming IP packet comes from a trusted user, and so can be used to help stop malicious packets from getting into an edge network. However, not all host computers in an edge network have the resources to run authentication algorithms. To make matters worse, host computers in an edge networks are often managed by different users with different skill levels. Therefore, some computers may be managed appropriately with no obvious security flaws, while other computers may be managed poorly with ample security loopholes.
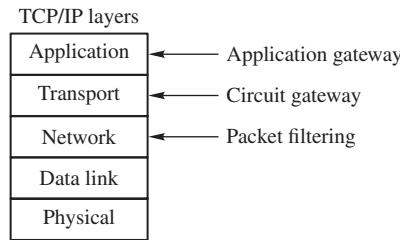
Firewalls that were developed in the 1990s are important instruments to help restrict network access. A firewall may be a hardware device, a software package, or a combination of both. It is used as a barrier between the Internet and an edge network. An edge network is also referred to as an *internal network* because it is controlled by its owner. The rest of the Internet that connects to the edge network is controlled by other owners, and so it is also referred to as an *external network*. Figure 8.1 shows an example of a firewall setup. Firewalls examine incoming and outgoing packets and determine whether to allow them to pass through or to block them. A packet that is blocked will be removed from the network.

Packets flowing into the internal network from the outside should be evaluated before they are allowed to enter. One of the critical elements of a firewall is its ability to examine packets without imposing a negative impact on communication speed while providing security protections for the internal network. Hardware firewalls are devices using integrated circuits customized for filtering packets. This technology is in general referred to as application-specific integrated circuit (ASIC). Today, firewalls have been embedded in commonplace networking devices, including routers, switches, modems, and wireless access points. Hardware firewalls are fast, but they are difficult to update. Software firewalls, on the other hand, are slower, but they are easier to update. Software firewalls can also run under different platforms. Some of these software firewalls have become a standard feature of popular operating systems (e.g., Linux 2.2 kernel and up and Microsoft Windows XP SP2).

The packet inspection that is carried out by firewalls can be done using several different methods. On the basis of the particular method used by the firewall, it can be characterized as

Figure 8.1   Schematic of a firewall

TCP/IP layers

Figure 8.2   Firewall placements at different layers

either a *packet filter*, *circuit gateway*, *application gateway*, or *dynamic packet filter*. In general, packet filters are placed at the network layer, circuit gateways are placed at the transport layer, and application gateways are placed at the application layer. Figure 8.2 shows a schematic of firewall placements at different layers. This firewall characterization based on protocol levels, however, may sometimes get blurred. Packet filtering, for example, often inspects both of the IP headers and the TCP headers. A dynamic packet filter is a hybrid firewall. It combines a packet filter and a circuit gateway into a firewall system.

These types of firewalls are standalone firewalls that are typically controlled by local administrators. In a large organization with a large number of firewalls, instead of letting each individual firewall administrator set up its firewall policy, the organization may want to establish a firewall policy for the entire organization. In such a case, the organization may want to set up a distributed firewall system to store the policy on a central location and have each firewall administrator enforce the policy on each firewall.

8.2   Packet Filters

Packet filtering is the basic firewall technology. It inspects ingress packets coming to an internal network from outside and inspects egress packets going outside from an internal network. The former is often referred to as *ingress filtering* and the latter as *egress filtering*. Packing filtering only inspects IP headers and TCP headers, not the payloads generated at the application layer. A packet filtering firewall uses a set of rules to determine whether a packet should be allowed or denied to pass through.

Packet filtering can be either *stateless filtering* or *stateful filtering*. Stateful filtering inspects the state of network connections. The firewall allows a packet to pass through if it belongs to an existing connection state or it is a legitimate request for creating a connection. The firewall blocks it otherwise.

8.2.1   Stateless Filtering

Stateless filtering is the simplest and the most widely used firewall technology. It treats each packet as an independent object, and it does not keep track of any previously processed packets. In other words, stateless filtering inspects a packet when it arrives and makes a decision without leaving any record of the packet being inspected. Stateless filtering is analogous to courier sorting in a postal office, where the sorting machine or a human sorter inspects each envelope to ensure that it has a valid destination address.

In particular, stateless filtering often inspects the source IP address and the destination IP address in an IP header against a predetermined set of rules. It may also inspect the source port and the destination port in a TCP header or a UDP header. The set of rules is often referred to as an *access control list* (*ACL*). Table 8.1 shows a simple ACL for ingress filtering, while Table 8.2 shows an ACL for egress filtering, where a.b.c.d denotes an IP address and * denotes any address or any port. If an IP address or a TCP/UDP port is untrustworthy or unwelcome, the IP address or the TCP/UDP port is blocked.

As the network layer already inspects IP headers to deliver the packets, implementing packet filtering at the network layer is convenient and does not incur too much computation overhead.

Rules in an ACL list are checked one at a time from top to bottom. If there is no rule in ACL to process a packet being inspected, for example, if its IP address or port does not appear in ACL, then this packet is blocked by default. In other words, at the end of a typical ACL list, there is a default rule that blocks every packet as shown in Table 8.3. This default rule does not need to be listed explicitly. We note that depending on the type of firewall and administration desires, the default rule may also be set to "allow every packet."

In addition to blocking ingress packets from a certain IP address or a certain port, or blocking egress packets to a certain IP address or a certain port, stateless filtering should also block the following types of packets:

1. An ingress packet having an internal address as the source IP address. A packet such as this is possibly a crafted packet for the purpose of disguising itself as a legitimate packet in the internal network to worm its way into the internal network.
2. A packet (ingress or egress) that specifies which routers are to be used. Such a packet is possibly a crafted packet for the purpose of bypassing certain firewalls.

Table 8.1 Sample ACL rules for ingress filtering, where "int" represents "internal," "ext" represents "external," and "addr" represents "address"

int addr	int port	ext addr	ext port	Action	Comment
*	*	a.b.c.d	*	Block	Block packets from this IP address
192.63.8.254	110	*	*	Allow	Open internal POP3 port

Table 8.2 Sample ACL rules for egress filtering

int addr	int port	ext addr	ext port	Action	Comment
*	*	a.b.c.d	*	Block	Block packets to this IP address
*	*	*	25	Allow	Allow packets to external SMTP port
*	*	*	> 1023	Allow	Allow packets to nonstandard port

Table 8.3 The default rule at the end of ACL

int addr	int port	ext addr	ext port	Action	Comment
*	*	*	*	Block	ACL default rule: block everything

3. A packet with a small payload that may be used to foil ACL filtering. The purpose is to make the payload so small that the TCP header in the payload will be split into two or more parts. For example, encapsulate the source port and the destination port in different IP packets. Such attacks are referred to as *TCP fragmentation attacks*.

In addition to blocking malicious ingress packets from entering internal networks, stateless filtering should also block certain internal packets from going out to external networks. In particular, control packets for establishing communications in internal networks should be blocked from going outside. These include control packets for carrying out the bootstrap protocols (Bootp), the Dynamic Host Configuration Protocol (DHCP), the Trivial File Transfer Protocol (TFTP), the Network Basic Input/Output System (NetBIOS), the Common Internet File System (CIFS), the Line Printer Remote protocol (LRP), and the Network File System (NFS) protocol.

Bootp enables a networked computer to boot without using its own hard drive. That is, it enables a networked computer to obtain an IP address and a boot image before loading the operating system. DHCP is a protocol developed on the basis of Bootp, and it often supports Bootp. TFTP is the simplest file transfer protocol that does not use much memory. TFTP is typically used to update firmware on embedded systems. NetBIOS enables networked computers in the LAN to communicate with each other. CIFS, LPR, and NFS enable networked computers in local area networks to share files and printers.

Stateless filtering is easy to implement, for it only inspects IP headers and TCP headers. However, stateless filtering does not block malicious packets that exploit application-layer software loopholes. As every packet must be examined against the entire ACL, stateless filtering could become a bottleneck on a high-speed network, resulting in inadvertent packet drops.

8.2.2 Stateful Filtering

Stateful filtering, also referred to as *connection-state filtering*, keeps track of connections between an internal host and an external host. A connection state (or state, for short) indicates whether it is a TCP connection or a UDP connection and whether the connection is established. Connection states are stored in a *state table*. When a packet arrives, whether it is an ingress packet or an egress packet, the stateful filtering firewall checks whether the packet belongs to an existing connection against its state table. If yes, the firewall allows the packet to pass through and saves the information (such as its TCP sequence number) for later use. If the packet is a SYN packet, the firewall creates a new entry in the state table. If the packet does not belong to an existing connection and it is not a SYN packet, the firewall will discard it. When a network connection ends, the connection state is removed from the state table.

A port number is a positive integer used to identify a particular program. Any port opened by an internal host (e.g., a server) is typically specified by a port number less than 1024 by default. Port numbers less than 1024 are referred to as standard ports. Also by default, external hosts will use port numbers between $2^{10} = 1024$ and $2^{16} - 1 = 65535$ to establish TCP connections with internal hosts. External port numbers may be generated dynamically. Table 8.4 shows an example of a state table.

Stateful filtering and stateless filtering are often used together. When it is difficult to determine whether a packet should be blocked on the basis of connection states alone, ACLs will become useful to help make a more accurate decision.

Table 8.4 Example of connection state table

Client addr	Client port	Server addr	Server port	Connection state	Protocol
219.22.101.32	1030	129.63.24.84	25	Established	TCP
219.22.101.54	1034	129.63.24.84	161	Established	UDP
210.99.201.14	2001	129.63.24.87	80	Established	TCP
24.102.129.21	3389	129.63.24.87	110	Established	TCP

Keeping a history of connection states, however, may require sophisticated data structures and search algorithms. One needs to determine, for example, how much information should be kept in a state table, how to manage it, and how to search for information from it. Executing these tasks may consume significant amounts of storage space and CPU cycles, which not only slows down network traffic, but also creates a negative security side effect. For instance, attackers may flood a large number of crafted packets into a targeted stateful filtering firewall, forcing it to execute excessive computations and therefore break the normal connection between the internal network and the external network.

Thus, when using stateful filtering, one needs to make sure that the time and space complexities of running it are manageable. For example, instead of keeping track of the entire history of a connection, one may only keep track of a connection for a fixed period of time.

8.3   Circuit Gateways

Circuit gateways, also referred to as *circuit-level gateways*, are typically operated at the transportation layer (although there are exceptions). They evaluate the information of the IP addresses and the port numbers contained in TCP (or UDP) headers and use it to determine whether to allow or to disallow an internal host and an external host to establish a connection.

It is common practice to combine packet filters and circuit gateways to form a dynamic packet filter (DPF).
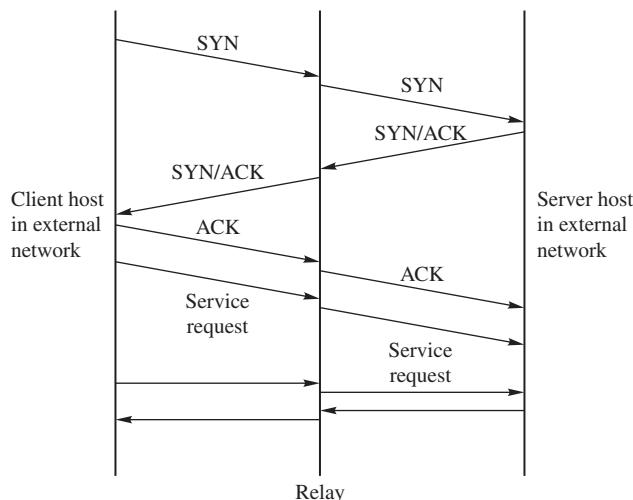
8.3.1   Basic Structures

The objective of a circuit gateway is to relay a TCP connection between an internal host and an external host. Thus, a circuit gateway is also referred to as a *transparent proxy firewall*. In particular, a circuit gateway first validates a TCP (or a UDP) session. It then establishes separately a connection with the internal host and a connection with the external host. It maintains a table of valid connections and checks an incoming packet (in either direction) against the information contained in the table. The gateway allows the packet to pass through if it belongs to an existing connection maintained in the table and blocks it otherwise. When a session ends, the corresponding entry is removed from the table and the circuit is closed.
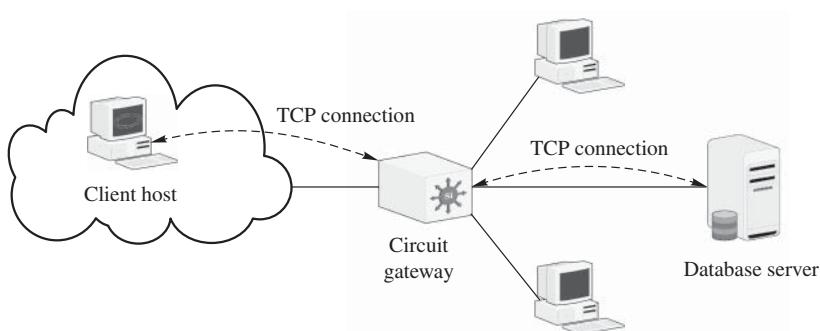
In other words, when an external host wants to establish a connection with an internal host in a network protected by a circuit gateway, the external host cannot establish a connection directly with the internal host. Instead, the external host can only establish a connection with the gateway. The gateway will then establish a connection with the internal host if such a connection is allowed. For example, suppose that the internal host is a server, and the external host is a client. The client establishes a TCP connection with the circuit gateway, and the

gateway establishes a TCP connection with the server. The server does not need to know the client's name or address to establish a connection, for it only establishes the connection with the gateway. Likewise, the client does not need to know the server's name or address to establish a connection, for it only establishes the connection with the gateway. That is, the gateway keeps internal computers from being seen from outside. When a connection is established, the circuit gateway will relay packets between the external host and the internal host without filtering these packets. Figure 8.3 shows how a circuit gateway is used as a relay node.

In practice, it is common for an organization to separate its internal network from the external networks using a circuit gateway, where the circuit gateway uses a public IP address reachable from outside, and the host machines in the internal network use private IP addresses unreachable from the Internet. For example, if a client from an external network wants to use the database server in the internal network, the client first makes a connection request to the gateway. The gateway then validates the request and establishes a connection, if the request is legitimate, with the database server in the internal network (see Figure 8.4).

Figure 8.3   Circuit gateway acts as connection relay

Figure 8.4   Schematic of a circuit gateway

After establishing a network connection with the external host and a network connection with the internal host, a circuit gateway will simply play a role as a relay node without inspecting packets passing through. Thus, an internal user may open a port on an internal host and instruct the gateway to establish connections between external hosts and the internal host. Malicious packets could therefore enter the internal network through such a channel. Thus, circuit gateways should be used together with packet filtering firewalls. In addition, a circuit gateway should keep a log to record information of the packets (ingress or egress) it validates, including the source IP address, source port, destination IP address, destination port, and the length of each packet. Such a log file could help identify problems.

8.3.2 SOCKS
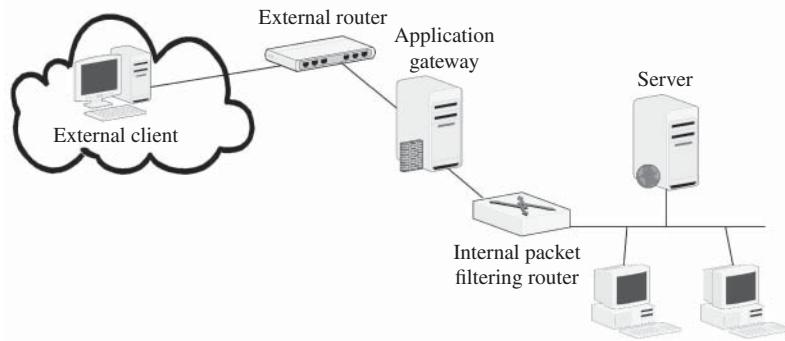
SOCKS (short for SOCKetS) is a network protocol for implementing circuit gateways. The first version of SOCKS was implemented by Dave Koblas and Michelle Koblas in the early 1990s. SOCKS consists of three components: the SOCKS server, the SOCKS client, and the SOCKS client library.

The SOCKS server runs on a packet filtering firewall through port 1080. The SOCKS client library runs on an internal host, and the SOCKS client runs on an external client host. The SOCKS client executes the modified versions of FTP and other standard TCP-based client application programs, which are modified for SOCKS.

When an external client wants to obtain service from an internal server under the protection of SOCKS, the client must first establish a TCP connection with the SOCKS server. It then negotiates with the SOCKS server to select an authentication algorithm, provides information for authentication, and submits a relay request. The SOCKS server verifies the information submitted for authentication and determines whether to establish a relay connection with the internal server as requested. Even if the external client just wants to send a UDP packet to an internal host, the client still needs to establish a TCP connection with the SOCKS server and submits information for authentication. Only after the client's request is granted should the client be allowed to forward the UDP packet to the internal host using the SOCKS server as a relay node. This packet is forwarded through the TCP connection that is established between the SOCKS server and the client.

8.4 Application Gateways

Application gateways, also referred to as *application-level gateways* (ALG) or *proxy servers*, are software packages installed on a designated computer. An ALG acts like a proxy for internal hosts, processing service requests from external clients. An ALG performs deep inspections on each IP packet (ingress or egress). In particular, an ALG inspects application program formats contained in the packet (e.g., MIME format and SQL format) and examines whether its payload is permitted. Thus, an ALG may be able to detect a computer virus contained in the payload. Because an ALG inspects packet payloads, it may be able to detect malicious code and quarantine suspicious packets, in addition to blocking packets with suspicious IP addresses and TCP ports. On the other hand, an ALG also incurs substantial computation and space overheads.

Figure 8.5   Schematic of an application gateway

8.4.1   Cache Gateways

Suppose that a certain organization wants to set up a Web server, where legitimate Internet users are allowed to obtain Web pages from the Web server. To protect the Web server from being compromised, a common approach is to set up an application gateway as a proxy for the Web server, called a *Web proxy server*. The Web proxy server receives requests at port 80 from external clients and performs deep packet inspections on packet payloads. Only after the packet payloads pass the security check should the Web proxy server pass the packets to the Web server. In addition, the Web proxy server also checks Web pages that the Web server sends to the external client and stores them in its cache. If other clients also request these pages, the Web proxy server could directly forward these pages from its cache to the clients without visiting the Web server. Proxy servers of this kind are referred to as *cache gateways*.

An application gateway is often used with a packet filtering router, where the router is placed behind the gateway (see Figure 8.5) to further protect connections between the gateway and the internal hosts.

Application gateways are specific to applications. For example, a Web proxy server is only used with Web servers; it does not apply to other types of applications.

8.4.2   Stateful Packet Inspections

Stateful packet inspection (SPI) extends stateful packet filtering (SPF) to also inspect packet payloads. SPI inspects whether a packet belongs to a legitimate connection and whether the format of its content matches the type of the service the connection is intended to provide. SPI is not the same as SPF. For example, the payload of a packet that belongs to a Web connection should be in a common Web format. If not, the packet will be blocked by the firewall.

8.5   Trusted Systems and Bastion Hosts

An application gateway is a computer placed between the internal network and the external network. It is exposed to attackers from the Internet. Thus, gateway computers need stronger

security protections. There are two common measures. The first measure fortifies the operating system of gateway computers to make it become a *trusted operating system*. The second measure fortifies gateway computers to become *bastion hosts*.

8.5.1 Trusted Operating Systems

A Trusted Operating System (TOS) is an operating system that meets a particular set of security requirements. Whether an operating system can be trusted or not depends on a number of elements. For example, for an operating system on a particular computer to be certified trusted, one needs to validate that, among other things, the following four requirements are satisfied:

1. Its system design contains no defects;
2. Its system software contains no loopholes;
3. Its system is configured properly; and
4. Its system management is appropriate.

The first two elements are the problems that system designers and developers need to resolve. Making source code publicly available (such as Linux) may help validate these elements. The last two elements are problems for system administrators and users to take care of. Determining how to access data and what access rights are given to users of different levels is critical.

Users read and write data through programs. Different users may be given different access rights to read or write files. Although they are using the same program, the access rights given to the program must be consistent with those of the users. To make an operating system running on a particular computer a trusted system, the system administrator must define a set of rules, specifying which program can be executed by which level of users and what access rights can be given to what level of users.

Each user of a TOS is assigned a certain level of clearance. Likewise, each file, program, and directory is given a certain level of secrecy. To prevent users from divulging data intentionally or unintentionally, TOS must carry out the following two rules rigorously:

1. *No read-up*: Users of a lower level of clearance cannot execute programs of higher level of secrecy. Programs of lower level of secrecy cannot read files of higher level of secrecy.
2. *No write-down*: Users of higher level of clearance cannot use programs of lower level of secrecy to write data to a file. Likewise, programs of higher levels of secrecy cannot write data into files of lower levels of secrecy.

The no-read-up rule is evident. To understand the no-write-down rule, assume that Alice has a high level of clearance, and so she can read files with a high level of secrecy. If she is allowed to write down, Alice can write data with high level of secrecy into a file with lower level of secrecy, allowing users of lower levels of clearance to read these data. Thus, no-read-up and no-write-down rules must be used together.

The Common Criteria, developed by government agencies of Canada, France, Germany, the Netherlands, the UK, and the United States in the late 1990s, defines a set of requirements for users to establish security certification of a system or a product.

Trusted Platform Modules

A Trusted Platform Module (TPM) is a cryptographic co-processor on modern motherboards. A TPM is a passive device, for it will not run unless users make a request. It provides basic

cryptographic operations such as RSA signatures, asymmetric cryptography using RSA, SHA-1 hash, and HMAC operations. While there do exist symmetric ciphers on the TPM, they are not exposed. TPMs also have the ability to provide some level of secure storage using special purpose registers and off TPM storage techniques. Of interest to the TOS is TPM's ability to audit the boot process.

A TPM can be used to audit the computer's boot software stack. This is achieved by offloading hash computations to the TPM. Specifically, each software module in the boot stack will trigger the computation of a hash of the software module that directly follows it in the boot process. The hash values collectively form a *trust chain*. The hash of the boot block for the BIOS of the machine is called the *root* of the trust chain. The root of a trust chain must be trusted, as the trust of all modules above it in the stack depends on the actions of the root. These trust chains can be used in one of two ways:

1. *Platform Validation*: The sequence of hashes can at a later time be extracted from the TPM and verified, thus providing reliance on the integrity of the system.
2. *Network Protection*: Once the machine is connected to the network, the chain is signed by the TPM and the client reports the value to a server that maintains a list of acceptable configurations for the machine. If the configuration does not match, then it means that the configuration has been compromised and therefore should be blocked.

This boot integrity process alone has brought the TPMs under scrutiny for allowing providers to restrict platform independence on their networks. The TPM itself is still a powerful measure of security that far outweighs, in most cases, the limiting of platform independence. TPMs have other uses, which can be found in the TPM ISO specification.

8.5.2 Bastion hosts and Gateways

Bastion hosts are computers with strong defense mechanisms. They often serve as host computers for implementing application gateways, circuit gateways, and other types of firewalls. A bastion host is operated on a trusted operating system that must not contain unnecessary functionalities or programs. This measure helps to reduce error probabilities and makes it easier to conduct security checks. Only those network application programs that are absolutely necessary, for example, SSH, DNS, SMTP, and authentication programs, are installed on a bastion host.

Gateways operated on bastion hosts must satisfy the following conditions:

1. Gateway software should be written using only small modules. Small modules are easier to check for security loopholes. Small modules are also easier to be reused.
2. A bastion host may authenticate users at the network layer. That is, it validates the source IP address and the destination IP address contained in an IP packet. Gateways running on a bastion host should authenticate users independently at a higher layer.
3. A bastion host should be connected to the smallest possible number of internal hosts, so that the number of internal hosts that could be affected because of security breaches of the bastion host is kept to the minimum. This measure also makes management of the bastion host network easier.
4. Bastion hosts should keep logs of how the systems are used, including connection state of each TCP session and how long each session lasts. These logs could help system administrators identify problems.

5. If multiple gateways are running on a single bastion host, these gateways must operate independently. If one gateway goes wrong, the system administrator can simply shut it down without affecting other gateways.

6. Bastion hosts should avoid writing data to their hard disks for the purpose of reducing the chance for the malicious codes (e.g., viruses, worms, and Trojan hosts) to enter the systems.

7. Gateways running on a bastion host should not be given system administration rights. In other words, gateway programs should be run under a well-protected directory on the bastion host. In so doing, even if a gateway is compromised, other gateways running under other directories can still be operational.

8.6 Firewall Configurations

Gateways running on a bastion host are often used with packet filters. Without loss of generality, we assume that routers have built-in packet filters. For the sake of convenience, we use a bastion host to represent a proxy server, where a bastion host may run several proxy servers simultaneously and independently. We introduce in this section several common firewall configurations. They are *single-homed bastion host system* (*SHBH*), *dual-homed bastion host system* (*DHBH*), and *screened subnets*. The latter is often referred to as *demilitarized zones* (*DMZ*).

8.6.1 Single-Homed Bastion Host System
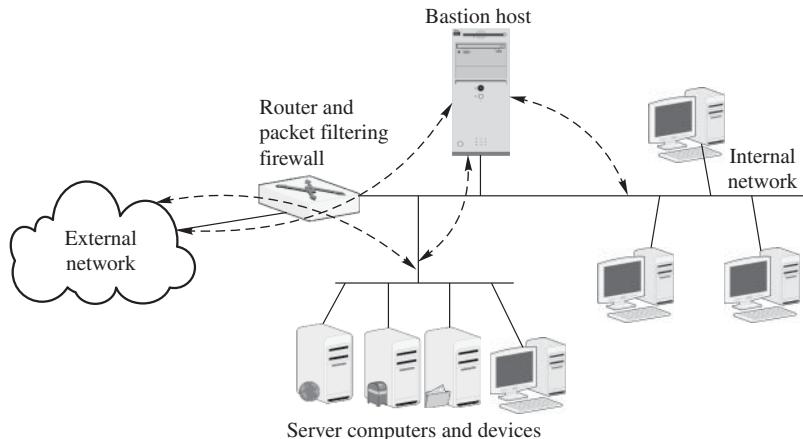
A single-homed bastion host system consists of a packet-filtering router and a bastion host, where the router connects the internal network to external networks and the bastion host is inside the internal network. The router announces to the public the IP addresses and the port numbers of the internal server computers. However, the router does not forward ingress packets directly to server computers. Instead, the router inspects an ingress packet. If the packet passes the inspection, the router passes it to the bastion host. The bastion host inspects the ingress packet. If it passes the inspection, the bastion host then determines which internal server this packet should be forwarded to. Egress packets going out of the internal network will also go though the bastion host. The packet filtering firewall inspects each egress packet and blocks it if its source address is not the IP address of the bastion host or if it fails other filtering rules.

Note that certain servers may not require strong security protections. For example, Web servers that offer general information about the degree programs in a university do not need strong security protection. Packets to such server computers do not need to go though the packet-filtering router. That is, ingress packets to these servers and egress packets from these servers are allowed without inspection. However, communications between these servers and the internal hosts still need to go through the bastion host. Figure 8.6 shows a schematic of a single-homed bastion host network.

In an SHBH system, if Malice compromises the packet-filtering router, she can modify ACL rules to bypass the bastion host. That is, Malice could modify the ACL rules to forward packets directly to internal hosts, making the bastion host become nothing but an ornament. This problem can be solved using a dual-homed bastion host.
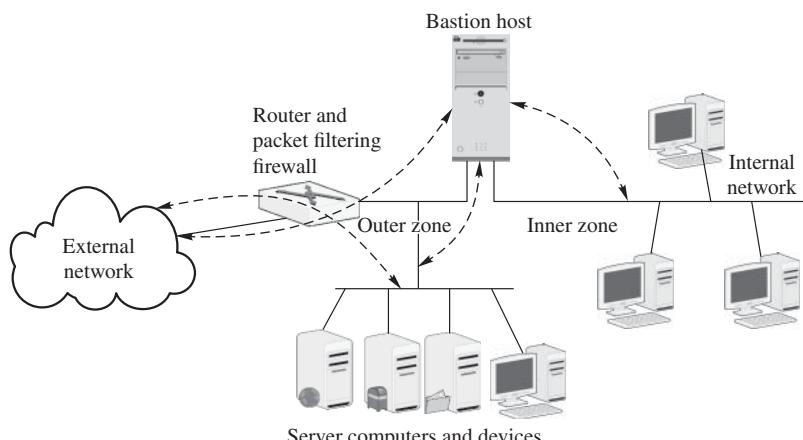
8.6.2 Dual-Homed Bastion Host System

A dual-homed bastion host network divides the internal network into two zones. These two zones are referred to as the *inner zone* and the *outer zone*. The inner zone is also referred to as

Figure 8.6   Schematic of a single-homed bastion host network, where the dotted arrow lines show the actual communications and the solid lines show the physical network connections

the *private zone*. The IP addresses of the host computers in the private zone are not reachable from the external network. The IP addresses of the host computers in the outer zone may be directly reachable from the Internet. In particular, the router is placed between the external network and the outer zone, and between the external network and the bastion host. Unlike in SHBH, the inner zone in DHBH is connected to the bastion host only. Thus, host computers in the inner zone are protected by both the bastion host and the packet-filtering router. The server machines in the outer zone are protected by the packet-filtering router. Similarly to the SHBH system, a DHBH allows the server computers in the outer zone to communicate to the Internet without going through the bastion host. In other words, the ACL in the router allows each inbound packet to pass through if its source addresses are allowed, and its destination IP address and port number match with the IP address of a server computer and an open port on that server. Figure 8.7 shows a schematic of a dual-homed bastion host network.

Figure 8.7   Schematic of a dual-homed bastion host network, where the dotted arrow lines show the actual communications and the solid lines show the physical network connections
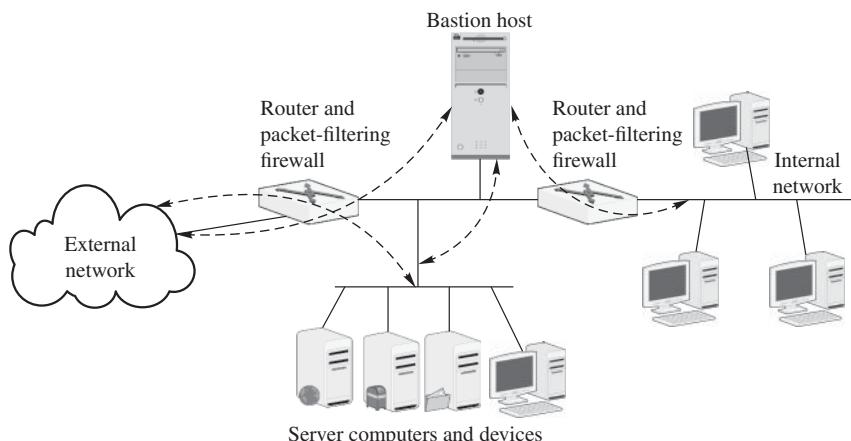
In a DHBH system, if Malice compromises the packet-filtering router, she still cannot bypass the bastion host.

8.6.3 Screened Subnets

Screened subnets are the most secure firewall configurations. A screened subnet consists of a bastion host and two packet-filtering routers. In particular, a screened subnet is a SHBH network with a second packet-filtering router (i.e., the inner router) inserted between the bastion host and the internal network. That is, in a screened subnet, one router is placed between the Internet and the bastion host, and the other router is placed between the bastion host and the internal network. The two packet-filtering firewalls create an isolated, screened subnetwork in between. Server computers and devices that do not require strong security protection are often placed in the screened subnetwork. Figure 8.8 shows a schematic of a screened subnet system.

The outer router announces to the public the IP addresses and port numbers of the server computers and devices connected to the screened subnetwork. The inner router announces to the internal network the IP addresses and port numbers of the server computers and devices connected to the screened subnetwork. Thus, the structure of the internal network is hidden from the outside world. The internal hosts can only communicate with external hosts through server computers or devices in the screened subnetwork.

We may move some of the server computers, for example, database servers, from the screened subnetwork to the internal network to provide a stronger protection; and place the corresponding proxy servers, for example, database proxies, in the screened subnetwork. This configuration, while increasing security, may reduce processing speed. In a particular application, we need to consider this trade-off and find the optimal configuration.

Figure 8.8 Schematic of a screened subnet system, where the dotted arrow lines show the actual communications and the solid lines show the physical network connections

8.6.4   Demilitarized Zones
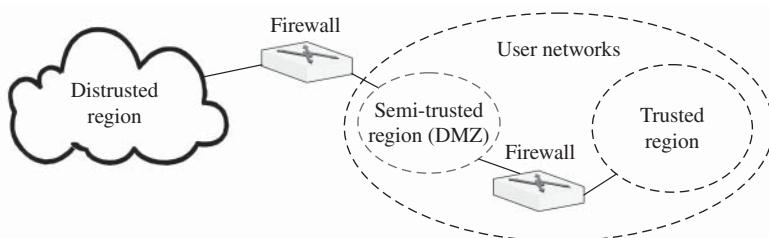
A subnetwork between two firewalls in the internal network is often referred to as a *demilitarized zone* (*DMZ*)[1]. The external firewall protects the DMZ subnetwork from external networks, and the internal firewall protects the internal network from the DMZ. A DMZ subnetwork may or may not have a bastion host. Server computers and devices that do not require strong security protections are placed in the DMZ subnetwork so that they will not be exposed to the external networks without any security protection.

The concept of a single-layer DMZ can be generalized to a multiple-layer DMZ, where a DMZ may also contain a sub-DMZ. Computers needing the most security protections are placed in a subnetwork that is connected to the innermost firewall. Computers needing the least security protections are placed in the outermost DMZ, which is protected only by the outermost firewall. Other computers are placed in a DMZ in between. An ingress packet that fails the inspection at the outermost firewall is blocked from entering the outermost DMZ. Likewise, an ingress packet that passes the inspection of the outermost firewall but fails the inspection at the second-layer firewall is blocked from entering the second DMZ. Organizing an internal network into a hierarchy of subnetworks increases security, for the attackers will have to compromise more firewalls to reach to a computer in a deeper subnetwork. Moreover, subnetworks are relatively easier to manage because of their smaller sizes.

8.6.5   Network Security Topology

Firewalls can be used to divide networks into three separate areas: distrusted region, semitrusted region, and trusted region (see Figure 8.9).

The distrusted region is the external network outside of the outer firewall. The semitrusted region is the DMZ between the outer firewall and the inner firewall. This area can include a bastion host and other server computers. The trusted region is the internal network behind the inner firewall. The concept of DMZ is a relative concept. For example, certain directories and files in a bastion host may be designated as DMZ.

Figure 8.9   A schematic of network security topology

---

[1] Demilitarized zone was a term first used by the United Nations in the 1950s to denote the strip of land along the 38th parallel that separated the Korean Peninsula into the North and South Korea. No troops or military activities of any kind are allowed in DMZ.

8.7 Network Address Translations

The network address translation (NAT) protocol divides the IP addresses into two groups. The first group consists of public IP addresses that are reachable from external networks. The second group consists of private IP addresses that are not reachable from outside hosts directly. We often refer to an internal network using private IP addresses as a *private network*. This collection of private IP addresses is broken up into three classes, which are defined in Table 8.5.

Using NAT, an edge network only needs a small number of public IP addresses to connect its gateways and routers to the Internet. The private IP addresses are then used by the internal hosts, which remain hidden behind the gateways and the routers. Different edge networks may share the same private IP addresses. When the internal network is organized into a hierarchy of subnetworks, it allows for better use of the available private address space to connect more computers. As NAT allows edge networks to use the same private addresses over and over again, it stretches the limited 32-bit address space in IPv4, allowing it to accommodate substantially more than 2^{32} hosts and devices.

8.7.1 Dynamic NAT

Dynamic NAT is a widely used network technology. It assigns a small number of reachable IP addresses dynamically to a large number of private networks.
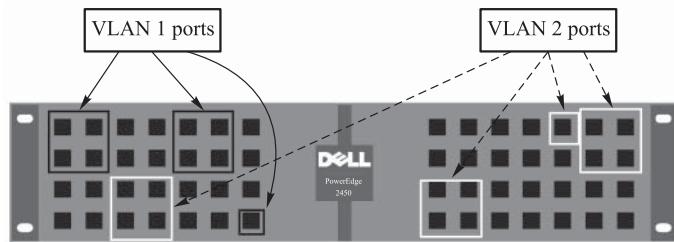
The port address translation (PAT) protocol, a variant of NAT, allows several private networks to share one public IP address. It is a common network technology for homes and small companies. For example, suppose that two internal hosts want to use port 25 to send email to external hosts at the same time, and their private addresses are 192.168.0.3 and 192.168.0.4. PAT does this as follows: it translates the source addresses in the packets sent from these two hosts to the same public IP address of the router with two different port numbers. For example, it may use 61003 as a port number to indicate a packet sent from host 192.168.0.3 and 61004 as a port number to indicate a packet sent from host 192.168.0.4. When it receives a returned packet from the destination, the PAT router first checks the port number in the packet. If it is 61003, the PAT router translates the destination address contained in the packet back to 192.168.0.3 and the port number back to port 25 and forwards the packet to 192.168.0.3:25. If it is 61004, the PAT router does the same thing for host 192.168.0.4.

8.7.2 Virtual Local Area Networks

A virtual local area network (VLAN) is a network technology for creating several independent logical LANs over the same physical network. VLANs can be created by configuring switches

Table 8.5 Private network address classes

Address class	Starting address of this class	Ending address of this class
A	10.0.0.0	10.255.255.255
B	172.16.0.0	172.31.255.255
C	192.168.0.0	192.168.255.255

Figure 8.10   Logical groupings of switch ports for creating independent VLANs

using software. That is, switch ports can be logically segmented into several groups, where each group is used for creating an independent VLAN (see Figure 8.10). The VLAN technology allows a network administrator to logically divide a physical LAN into different broadcast domains, or to logically configure host computers on different LANs to the same VLAN.

It is convenient to use the VLAN technology to form and modify logical networks on the basis of temporary needs. For example, a university can form a VLAN for its tenure and promotion committee. The host computers connected to this VLAN are those used by the committee members. A firewall or a bastion host may be used to protect this VLAN. Committee members are elected from each college within the university, and they are changed from year to year. Thus, the membership of this VLAN will also be changed from year to year.

The IEEE 802.1q standard has been established to support VLANs. We note that vulnerabilities were found in certain 802.1q implementations that could compromise security.

8.7.3   Small Office and Home Office Firewalls
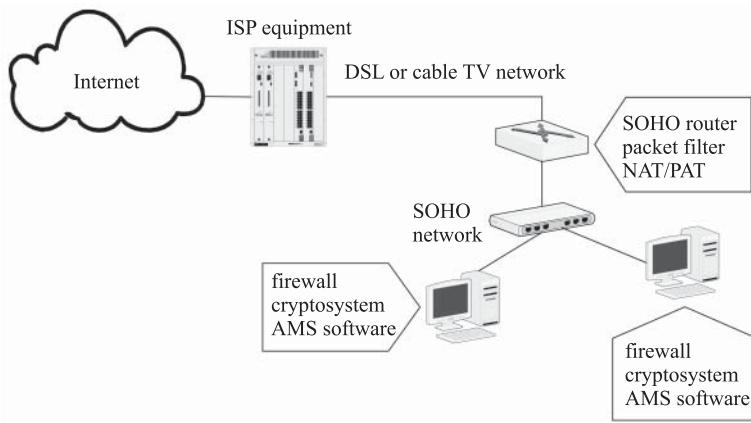
Setting up and managing firewalls may require special training, which means it may be impractical for ordinary users to set up a firewall in a small office or home office (SOHO). We note that SOHO users often use digital subscriber lines (DSL) to connect their computers to telephone company's Internet service equipment, or use coaxial cables to connect their computers to cable television company's Internet service equipment. The router used for either connection typically supports NAT/PAT and packet filtering. Such a router is sometimes referred to as a *SOHO firewall* (see Figure 8.11).

Modern SOHO routers often support IEEE 802.1q, although a firmware update may be required.

8.8   Setting Up Firewalls

Microsoft Windows operating systems are shipped with a built-in firewall. To set it up, open Windows Firewall under the Control Panel and click the firewall on.

For Linux and UNIX operating systems, the user may build a firewall using built-in programs. For example, Linux users may use the iptables program to build a personal stateless packet filter, while FreeBSD UNIX users may use the pf program to build an organizational firewall with a DMZ.

Figure 8.11   Schematic of a SOHO firewall network

8.8.1   Security Policy

To set up a firewall, we must first form a security policy to specify what is allowed and what is not allowed. In general, we may want to allow internal users to access the Internet freely or with minor restrictions. For example, the firewall should allow DNS query packets and initial egress TCP connection packets to pass through. However, internal users are not allowed to offer any Internet service directly to the external users. This means that ingress packets of Internet services must go through appropriate gateways. For example, ingress HTTP packets must go through a Web server, while ingress mail packets must go through a mail server.

No matter how complex it might evolve, a security policy must start from a small set of simple rules. On the other hand, for a security policy to be useful, it must be easy to understand and it must be implementable.

8.8.2   Building a Linux Stateless Packet Filter

The iptables command is a built-in program in Linux for building a stateless packet filter. It supports NAT, routes IP packets, and filters IP packets on the basis of addresses, port numbers, and flags. In particular, iptables organizes ACL rules into several subsets of rules called *chains*. There are three system chains. They are the *input* chain, the *output* chain, and the *forward* chain. The input chain is the set of rules for ingress packets, the output chain for egress packets, and the forward chain for routing packets. The basic syntax of iptables is as follows:

```
iptables <option> <chain> <matching criteria> <target>
```

When building a firewall, the common option is -A, which means to append a new rule at the end of the chain. In addition to -A, the following option of -I may also be used, meaning to insert a new rule at the beginning of the chain.

Recall that ACL rules are processed in order. Thus, it is important to ensure that rules are implemented in the correct order.

Suppose that we want to create a simple stateless packet filter on a Linux host with IP address 129.63.8.109 on the basis of a security policy that permits TCP connections initiated

by internal users and denies ingress telnet packets. For simplicity, we only create rules for the input chain. This can be achieved using the following command lines:

```
iptables -A INPUT -p TCP -s 129.63.8.109 -j ACCEPT
iptables -A INPUT -p TCP ! -syn -d 129.63.8.109 -j
  ACCEPT
iptables -A INPUT -p TCP -d 129.63.8.109 telnet -j
  DROP
```

The options of -j, -p, -s, and -d specify how each ACL rule is to be operated, where -j (means to jump to the target) specifies what to do on the packet under inspection if the matching criteria is met, -p specifies what protocol the ACL rule applies to, -s specifies the source IP address, and -d specifies the destination address. The operator ! means negation, and the option of -syn specifies the SYN flag in the packet.

Thus, the first rule specifies that egress IP packets for TCP connections from host 129.63.8.109 are always allowed. The second rule specifies ingress IP packets for TCP connections with host 129.63.8.109 are allowed, provided that they are not TCP SYN packets. The third rule specifies that ingress telnet packets to 129.63.8.109 are blocked.

8.9   Closing Remarks

Firewalls are used to block malicious ingress packets from entering internal networks and block malicious egress packets from going out to external hosts. However, no matter how strong firewalls are, attackers may still be able to find ways to enter internal networks. For example, attackers may disguise themselves as legitimate users to go into internal networks. Therefore, we will need to know how to identify these intruders. Intrusion detection systems are technologies developed to meet this need, which are discussed in the following chapter.

8.10   Exercises

8.10.1   Discussions

8.1. Describe your experience in setting up a firewall on your home computers or work computers.

8.2. If you are familiar with the setup procedure of a particular type of firewall products (such as Cisco firewalls), share your experiences.

8.3. If you are familiar with Linux firewall setup steps, share your experiences.

8.4. "Home computers may run slower when firewalls are installed. I have found that it is more helpful to become as security savvy as possible," said a reader. "I read reports from Symantec, BugTraq, Microsoft Security Bulletin, and other security forums to keep up with the latest malware reports and what they do. I check the Registry keys for strange modifications. I also view the Task Manager to see if there are strange or malicious processes running." Share your thoughts and experiences how to become security savvy.

8.5. Discuss why the following firewall configurations are important: (1) Create ACL deny rules. (2) Enable a global policy for FTP inspection. (3) Hide NATs from outside.

8.6. Share your experience in setting up a DMZ (if you have done it).

8.10.2   Homework

8.1. Suppose that an ACL contains the following rules for processing ingress packets:

int addr	int port	ext addr	ext port	Action	Comments
*	25	*	*	Allow	Allow ingress SMTP packets
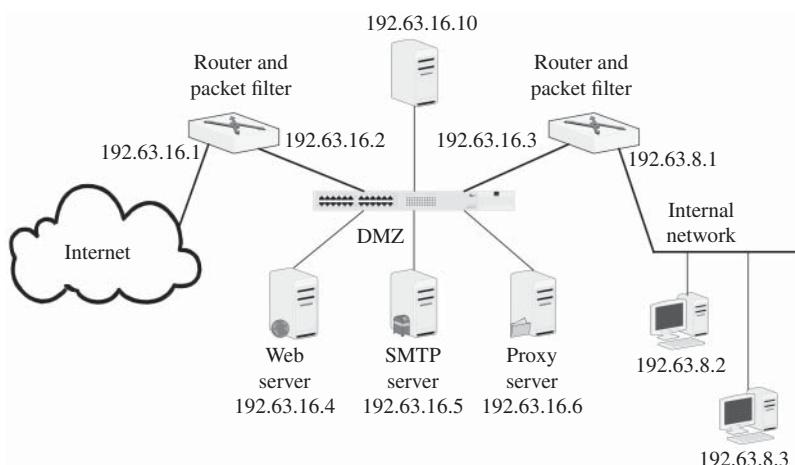
Is this ACL rule secure? Justify your answer.

8.2. Can encrypted packets be relayed through a circuit gateway? Justify your answer.

8.3. Table 8.6 lists common communication protocols used to establish local area network service. Construct ACL rules to block packets that carry out these protocols from going out to external networks.

8.4. Requiring a TCP packet as an IP payload to be longer than a certain fixed length can help resist TCP fragmentation attacks. However, IP packets may not arrive at the destination in the original order. If a TCP header is divided into two halves, the IP packet that contains the second half of the TCP header may arrive earlier than the the IP packet containing the first half of the TCP header. How do you suggest to handle this situation? Justify your answer.

8.5. Suppose that in the schematic of the screened subnet shown in Figure 8.8 we want to upgrade the security protection of the SMTP server computer. Describe one or more methods to solve this problem.

Table 8.6   Communication protocols used for establishing LAN

Port	Transport-layer protocol	Application
67/68	UDP	Bootp/DHCP
69	UDP	TFTP
135, 137, 138, 139	TCP and UDP	NetBIOS
445	TCP and UDP	CIFS
515	TCP	LPR
2049	UDP	NFS

8.6. Figure 8.12 is a screened subnet firewall system, where DMZ contains three server computers. The IP addresses of the outer router, the inner router, and server computers are shown in the figure. Construct ACL rules such that external hosts can directly communicate with DMZ server computers but cannot establish direct communications with any internal host. Justify your construction.

8.7. In Figure 8.12, assume that the outer router uses ACL rules given in Table 7.7 and the inner router uses ACL rules given in Table 7.8. In addition to port 25, other ports in the tables are defined as follows: port 80 is used for Web server program HTTP, port 7 is used for server program echo, port 23 is used for server program `telnet`, and port 22 is used for server program SSH.
  (a) Explain what each ACL rule is intended to do.
  (b) Point out which ACL rule is used for egress packets and which ACL rule is used for ingress packets.

8.8. The internal network in Figure 8.12 uses private addresses for its hosts. Suppose that the inner router supports the PAT protocol. If two hosts in the internal network send egress packets from port 80 simultaneously, where one host uses private address 192.168.8.2 and the other host uses private address 192.168.8.3. Describe how to use PAT and one public IP address 192.63.16.3 to accomplish this task.

8.9. If in an ingress packet from an external network to the internal network, its source address is an internal IP address or is a private network address, should this packet be allowed or blocked? Why?

*8.10. To set up a firewall to deal with DNS packets, one should allow any egress DNS queries and filter ingress DNS responds, for external servers should not be trusted. Discuss the advantages and disadvantages of using packet filtering and DNS proxy to filter DNS packets.

Figure 8.12   The firewall system used for Exercises 8.6 and 8.7

Table 8.7   The ACL rules contained in the outer router in Figure 8.12

Line number	Source addr	Source port	Dest addr	Dest port	Action
1	*	*	192.63.16.10	> 1023	Allow
2	*	*	192.63.16.1	*	Block
3	*	*	192.63.16.2	*	Block
4	192.63.16.1	*	*	*	Block
5	192.63.16.2	*	*	*	Block
6	192.63.16.10	*	*	*	Allow
7	*	*	192.63.16.5	110	Allow
8	*	*	192.63.16.10	7	Block
9	*	*	192.63.16.10	23	Block
10	*	*	192.63.16.10	22	Allow
11	*	*	192.63.16.4	80	Allow
12	*	*	*	*	Block

Table 8.8   The ACL rules contained in the inner router in Figure 8.12

Line number	Source addr	Source port	Dest addr	Dest port	Action
1	*	*	192.63.16.10	> 1023	Allow
2	*	*	192.63.16.10	25	Allow
3	*	*	192.63.16.3	*	Block
4	*	*	192.63.8.1	*	Block
5	192.63.16.3	*	*	*	Block
6	192.63.8.1	*	*	*	Block
7	192.63.8.2	*	*	*	Allow
8	192.63.8.3	*	*	*	Allow
9	192.63.16.6	*	192.63.8.2	*	Block
10	192.63.16.6	*	192.63.8.3	*	Block
11	*	*	*	*	Block

*8.11.   Suppose that you telnet from an internal host to an outside host on which you don't have an account. Your `telnet` session should fail, but it doesn't. The internal host you use to launch `telnet` has an Ethernet port connected to the local gateway. Explain how the `telnet` packets can reach the outside host and allow you to create a connection. Explain how to prevent this scenario from happening. (*Hint:* suppose that there is another router connected to the local gateway, which emits a default entry to the internal network. This is sometimes referred to as a *route leak*.)

8.12.   If an ingress packet has 127.0.0.1 as both of its source IP address and its destination IP address, should the packet filter block this packet? Why? (Note: 127.0.0.1 is the address of localhost.)

8.13.   If an ingress packet has 0.0.0.0 as its source IP address or its destination IP address, should the packet filter block this packet? Why? (Note: 0.0.0.0 is the address for broadcasting messages.)

8.14. If an ingress packet belongs to an existing TCP connection, should this packet be blocked? Why?

8.15. If an ingress packet has 25 or 80 as its destination port number, should this packet be blocked? Why?

8.16. Explain why SPI is different from SPF.

8.17. If an ingress packet has the IP address of an internal host as its destination address and has 7 or 23 as its port number, should this packet be blocked? Why?

8.18. In general, egress HTTP packets should be allowed to pass through the firewalls. Ingress HTTP packets, however, should go through firewall inspections. Suppose that the Web server is placed in DMZ. If an ingress HTTP packet has an internal IP address as its destination address, should it be blocked? Why?

8.19. Should egress SMTP traffic be filtered? Why?

8.20. Should an internal host be allowed to connect to an external POP3/IMAP server? Why?

8.21. If an ingress packet has the IP address of an internal host as its destination address and has 22 as its port number, should this packet be blocked? Why?

8.22. Microsoft Windows uses port numbers 135–139 and 445 for NetBIOS and file sharing. If an ingress packet has one of these numbers as its port number, should this packet be blocked? Why?

8.23. If the source address of an egress packet is not an internal address, should this packet be blocked? Why?

8.24. If the destination address of an egress packet is a private network address, should this packet be blocked? Why?

8.25. If the destination port of an egress packet is 53, but its source address is not the address of a DNS server, should this packet be blocked? Why?

8.26. In Microsoft Windows XP or 2000, open Internet Explorer, and then click Tools and Internet Options. Browse carefully the following options: Security, Content, Privacy, and Advanced.
   (a) There are two options for Trusted Sites and Restricted Sites under Security. Explain what they mean and how to use them.
   (b) Explain what the option of Privacy means and how to use it.
   (c) Explain what each of the options for Content mean and how to use it.
   (d) Explain what each of the options for Advanced mean and how to use it.

8.27. In Microsoft Windows XP or 2000, open Internet Explorer, and then click Tools and Internet Options. Use Security and Privacy to accomplish the following tasks:
   (a) Set up a reverse firewall.
   (b) Block Web page cookies from entering your system.

8.28. What do the following iptables rules do?

```
iptables -A INPUT -i $INTERNET -s $BROADCAST_DEST -j
  LOG
iptables -A INPUT -i $INTERNET -s $BROADCAST_DEST -j
  DROP
```

where the option `-i` specifies the interface name the rule applies to, and the `-j LOG` logs packets that match the rule.

8.29. Give two examples of attacks that can be resisted using firewalls alone.

8.30. Give two examples of attacks that cannot be resisted using firewalls alone.

8.31. Using the NAT technology, one can connect substantially more than 2^{32} hosts and devices to the Internet with 32-bit IP addresses. Explain why.

8.32. Under Microsoft Windows 7, DNS search can be done as follows: Click successively Start and Run. Then type nslookup. Under the prompt $>$ in the popup window, enter an IP address and press the return key. For example, if you enter cs.uml.edu, then you will see the following output after you press the return key:

```
> cs.uml.edu
Server: DD-WRT
Address: 192.168.11.1

Non-Authoritative Answer:
Name: cs.uml.edu
Address: 129.63.8.2

>
```

(a) Under the prompt $>$ type google.com. Explain what you see.
(b) Under the prompt $>$ enter 66.94.234.13. Explain what you see.
(c) Certain firewalls may block this type of DNS search. If so, you may use the service provided by `http://www.kloth.net/services/nslookup.php` to carry out a DNS search.

*8.33. Search the literature and describe in detail a real instance of attacking a firewall. Explain why this attack works. Suggest a solution to help resist such an attack.

8.34. Under Microsoft Windows, you may use route print to show the routing table on your computer. Use, instead, route or ip route show under Linux or UNIX. Show how to use them and explain what you see.

*8.35. Read RFC 3089 at `http://www.ietf.org/rfc/rfc3089.txt` and write a paper of about 4000 words to describe the SOCKS protocol.

8.36. Some employees in Company X were using company time to bid on goods at eBay. The company wants to set up ACL rules to block online bidding on eBay.

    (a) Use nslookup to find the IP address of eBay.

    (b) Create ACL rules that will block internal hosts from communicating with eBay servers.

    *(c) Can eBay do some simple modification so that the ACL rules you created in (b) will not work?

8.37. "I work in a financial institution," said a reader. "So there are times we need to allow a third-party's regulatory application servers to access our network. We have to modify our firewall ACL to open up ports (ftp, proxy, tcp, and http, among others), for they may be required for the connections to happen." What types of firewalls do you think this financial institution has. Can you suggest improvement?

8.38. "I use the Windows firewall on my PC," a reader told us. "But I have to disable my firewall every now and then as it does not allow my USB Internet modem to work. This is not the best security practice, but I have been lucky so far as regards any attack." Can you suggest a better way to do this?

8.39. "At home I have setup a Cisco ASA-550X firewall with SPI, ACL, port forwarding and triggering features to protect my small internal network," a reader told us. "I also have Cisco SSL VPN enabled and setup ACL to allow only certain traffic on the network and specify what networks/subnet the VPN clients is able to access. In this setup I created an ACL policy to allow VPN client to connect and communicate with specified computers." Argue that this protection would be sufficient for a small internal network.

8.40. "At work I used both E5500 and Cisco ASA-5510 firewalls to protect the corporate network," said a reader. "I used address objects, NAT, and the firewall policies for traffic shaping to include multiple site-to-site VPNs using IPsec and Dynamic Multipoint VPN (DMVPN) between branch sites and vendor sites. A few of our systems were within the DMZ, and these systems were protected with antivirus software, static routing configurations, and multiple ACL policies." Do you think these firewall measures are sufficient to protect a corporate network? Justify your answer.

8.41. "I am running Mac OS X 10.9.2 on my personal laptop," a reader told us. "When enabled, the firewall prevents unauthorized applications, programs, and services from accepting incoming connections (Mac OS X Window). Recently I discovered that the firewall was off. I must have disabled it in the past to do something specific and forgotten to re-enable it. I wonder how long ago that was!" Have you experienced similar situations? Explain what security consequences you might have to face should this happen.

9

Intrusion Detections

Network perimeter security cannot stop attackers from entering the internal networks if they obtain authenticated access to target computers and log on to them as legitimate users. Attackers may be able to obtain login information of legitimate users through, for example, identity spoofing and phishing attacks. Attackers of this kind are intruders.

Thus, it is desirable, and often is necessary, to detect intrusion activities by monitoring ingress packets that have passed through firewalls and analyze how users use their computers, so that system administrators can take appropriate actions against intrusions. It is also possible to prevent intrusions from entering important systems by using sacrificial decoy assets, called *honeypots*, which lure attackers' attention away from the computers that need protection. This chapter introduces common intrusion detection techniques and honeypot techniques.

9.1 Basic Ideas of Intrusion Detection

Building automated systems to detect intrusion activities was initiated by Dorothy Denning and Peter Neumann in the mid-1980s. They observed that intruders often acted differently from the legitimate users they impersonated. Moreover, behavior differences may be measured to allow quantitative analysis. Their seminal work has evolved into a fruitful branch of network security.

The goal of intrusion detection is to identify intrusion activities that already occurred or are currently occurring inside an internal network. In particular, intrusion detection wants to detect intrusion activities as quickly as possible so that appropriate actions can be taken to minimize damages caused by the intrusions. It also wants to trace intruders and collect evidence to indict the criminals. A common approach to detecting intrusions is to find ways to identify abnormal events, such as finding behavior discrepancies between the intruder and the legitimate user impersonated by the intruder. This can be done by building automated tools on the basis of operating system administrations, network protocols, computational statistics, and data mining. Automated tools for detecting intrusions are referred to as intrusion detection systems.

An intrusion detection system (IDS) is an automated alarm system that searches for direct or indirect intrusion indications, including intrusions that already happened or are currently taking place, and notifies the system administrators to take appropriate actions. Traditional IDS systems may only detect intrusions and alert system administrators. But intrusion detection systems may also be more proactive to automatically respond to detected intrusion activities. Such systems are often referred to as intrusion prevention systems (IPS). An IPS could automatically modify network perimeter rules, isolate the affected systems, or shut down services. For simplicity, we use IDS to denote both traditional IDS and IPS.
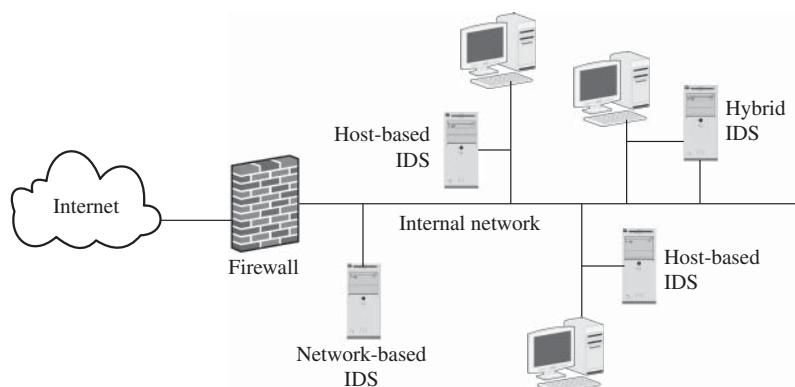
9.1.1 Basic Methodology

The basic methodology of detecting intrusions is to log system events and analyze them using appropriate methods. For example, one may build a simple IDS as follows: log all the packets passing through a router (or a firewall) using a packet sniffer/logger, and analyze the log to identify suspicious events on the basis of a given set of rules that specify what events are unacceptable. Such analysis may be done manually if the log is small. However, typical log files tend to be as large as several mega bytes. It is formidable to analyze a log file of large size manually. This calls for automated tools. Snort, for example, is an open source automated tool (http://www.snort.org) that can log and analyze IP packets in real time. Analyzing logs is often referred to as *auditing*.

Keeping logs is important, not only for intrusion detection, but also for post-incident forensics and recovery. Logs may also be required as evidence for prosecuting intruders.

Intrusion detection may be carried out at the network level, or at the host level, or both. That is, an IDS may detect anomalies in the LAN, or in networked computers, or both. The first type of intrusion detection is referred to as *network-based detection* (*NBD*), the second type as *host-based detection*, and the third type as *hybrid detection* (*HBD*). Figure 9.1 shows a schematic of an intrusion detection system layout with a firewall in place.

NetRanger, for example, is a network-based IDS product made by Cisco Systems, Intruder Alert is a host-based IDS product made by Axent Technologies, and CyberSafe

Figure 9.1 A schematic of an intrusion detection system layout with a firewall in place

is a hybrid IDS product made by CyberSafe Corporation. A hybrid system may simply consist of a network-based system and a host-based system that can operate independently. If a hybrid system is not available, organizations should deploy both network-based systems and host-based systems and merge alerts from both systems to achieve hybrid-like system effects.

9.1.2   Auditing

Security auditing is a routine security process. There are two kinds of auditing. The first kind audits static configuration information, which is also called *security profiles*. The second kind audits dynamic events.

9.1.2.1   Security Profiles

A security profile is a set of preconfigured values of certain security parameters, such as how long a user password should remain valid and how often a user password should be changed. Table 9.1 shows a simple security profile of a system on password and login parameters.

9.1.2.2   Events

A typical event record should consist of the following fields:

1. *Subject*: The subject field provides information of the initiator of the event.
2. *Action*: The action field provides information of the operation carried out by the subject.
3. *Object*: The object field provides information of the action receiver.
4. *Exception condition*: The exception-condition field specifies exception condition of the event.
5. *Resource usage*: The resource-usage field provides quantitative information about the use of computing resources of the event.
6. *Time stamp*: The time-stamp field specifies when the event takes place.

Most events are sequences of elementary actions, where an elementary action could be a single memory-access operation, a single arithmetic operation, or a single logic operation. For

Table 9.1   A sample security profile on password and login parameters

	Parameters	Values
	Minimum length (bytes)	8
Password	Lifetime (days)	90
	Expiration warning (days)	14
	Maximum number of unsuccessful attempts allowed	3
Login session	Delay between login attempts (seconds)	20
	Time an account is allowed to remain idle (hours)	12

Table 9.2 A sample event recorded for detecting intrusion auditing, where "byte-r" represents the number of bytes been read and "byte-w" represents the number of bytes been written

Subject	Action	Object	Exception condition	Resource usage	Time stamp
Alice	Executes	cp	None	CPU: 00001	Tue 11/06/07 20:18:33 EST
Alice	Opens	./myprog	None	byte-r: 0	Tue 11/06/07 20:18:33 EST
Alice	Writes	etc/myprog	Write fails	byte-w: 0	Tue 11/06/07 20:18:34 EST

example, suppose that Alice, a user of a UNIX system, tries to copy her program myprogram to the system directory /etc by issuing the following command:

cp myprog/etc

This event consists of three records based on its elementary actions (see Table 9.2). Assuming Alice is not a super user, and so she does not have write-permission in /etc, the command is aborted.

9.1.3 IDS Components

A typical IDS system should consist of three components: assessment, detection, and alarm.

9.1.3.1 Assessment

The assessment component evaluates security needs of a system and produces a security profile for the target system.

9.1.3.2 Detection

The detection component collects system usage events and analyzes these events to detect intrusion activities, where each record in the event log should contain information useful for detecting intrusions. The objective of auditing events is to identify anomalies on the basis of certain rules and quantitative measures. For example, one may characterize user activities in a computer system as a time series of discrete events, define a *user profile*, and define acceptable variations of the user profile. If a series of events executed by a user is found to be significantly different from any acceptable variation of the user's user profile, then the user is likely to be an intruder.

In addition to detecting user behaviors, an IDS may also analyze *program behaviors*. For example, we note that a typical Web server daemon should not launch programs outside of its CGI directory. Thus, if something happens otherwise, then it may indicate that a worm is attempting to infect the Web server.

User Profiles

A user profile may be created by analyzing user's activities for a certain period of time. It may, for instance, contain the following information:

1. When does the user usually log on and log off?
2. What programs, and what ordering of these programs, will the user normally run?
3. How long will each program be executed?

For example, suppose that Betty is a secretary of the Biology department in a university. Her user profile may look like the following: Betty normally logs on her computer at 8:05 a.m. She spends about 1 hour reading and replying to email. After that, she uses Microsoft Word and Excel to work on certain documents under certain directories until 4:00 p.m. She would occasionally print a file, copy a file, create a new directory, open her mailbox reading and replying to email, or let her computer idle for a short while during these hours. But she never installs software or probes system directories. She normally logs out at 4:05 p.m. Thus, if the system detects that Betty is logging in to her account at midnight and is installing software, then it will be considered as abnormal activity, which may be the act of an intruder impersonating Betty.

On the basis of user profiles, the IDS system may allow certain reasonable variations of the user profile. For example, it may allow justifiable activities (e.g., to open or edit PDF files) to be included in Betty's user profile, although such activities have not been recorded. An acceptable variant may also remove certain events from the user profile or change the ordering of observable events. What constitutes acceptable behaviors depends on individual users. Different users may have different sets of acceptable behaviors.

9.1.3.3   Alarm

When an attacker impersonates a legitimate user to log on to the user's account, the attacker's behaviors would likely be different from the true user's behaviors, which would be considered unacceptable and will therefore trigger the IDS to alarm the user or the system administrator. The alarm component specifies how this is to be done. It classifies alarms and specifies how the system should respond to an alarm.

9.1.4   IDS Architecture
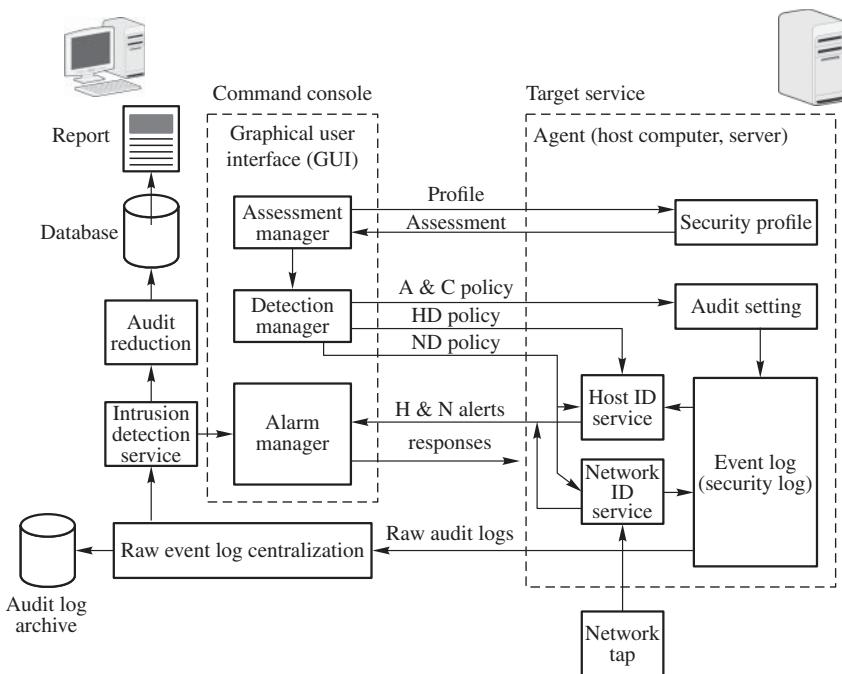
An IDS is an automated alarm system. A typical IDS may consist of a command console and the targets to be monitored. Figure 9.2 shows a block diagram of an IDS system.

9.1.4.1   Dissection of the IDS Architecture

Command Console

The command console, a.k.a. a detection center, should be run on a separate computer. It controls and manages the target systems. The target systems in an IDS system are host computers or server computers to be protected. A typical command console may consist of the following components:

1. *Assessment manager*: It manages and assesses security profiles of the target systems.
2. *Detection manager*: It maintains connections with the detection component of the target systems.

Figure 9.2 A block diagram of an IDS system, where "A & C policy" stands for "audit and collection policy," "HD Policy" stands for "host-based detection policy," and "ND policy" stands for "network-based detection policy"

3. *Alarm manager*: It collects and manages suspicious information and determines what responses are appropriate. For example, it may instruct the target system to do one or more of the followings: disconnect network connection, cancel user login sessions, remove user accounts, reset firewall ACLs, and shut itself down.
4. *Raw event log centralization*: It collects log files sent from the target service.
5. *Intrusion detection service*: It checks the log files collected by the raw event log centralization. When it finds problems, it informs the alarm manager to handle the problems.
6. *Audit reduction*: It simplifies and organizes raw event log information for storage.
7. *Audit log archive*: It stores raw event log data.
8. *Database*: It stores processed and organized event log data.

Target Service
The target service is performed on a host computer or on a server computer to detect intrusions on these devices. It may consist of the following components:

1. *Security profile*: The assessment manager at the command console assesses the target system and creates a security profile of the system, which is stored in the target system.
2. *Audit setting*: The detection manager at the command console sets up what is to be audited at the target system.
3. *Event log*: The target system generates an event log according to the audit setting.

4. *Host-based intrusion detection service*: It checks the event log to search for anomalies and informs the alarm manager at the command console.
5. *Network-based intrusion detection service*: It uses a network sniffer/logger (a.k.a. a network tap) to log packets, searches for anomalies, and informs the alarm manager at the command console.

The command console should maintain network connections only with the target systems. That is, the command console should not be reachable from external networks. One way to ensure this is to place the command console behind a firewall that blocks all packets from or to external networks. For home or small office users, however, the command console and the target service may be installed on the same computer.

One may also construct a distributed IDS by distributing the command console components, part or all of them, across several target systems (see Exercise 9.5).

9.1.5   Intrusion Detection Policies

Intrusion detection policies (IDP) are used to identify intrusion activities. They specify what data must be protected and how well they should be protected. They also specify what kinds of activities are considered intrusions and how to respond when suspicious activities are identified.

Ideal IDPs should be simple, effective, and easy to implement, with minimum false alarm rates. That is, IDPs should not allow IDS to detect something that is normal as abnormal. Returning something that is normal as abnormal is also known as *false positive detection* or *false positive alarm*. On the other hand, IDS should not detect something that is abnormal as normal. Returning something that is abnormal as normal is also known as *false negative detection*.

9.1.5.1   False Positives Versus False Negatives

False positive detections and false negative detections are common scenarios in IDS, for the boundaries between normal and abnormal activities are not always clear. False positive detections and false negative detections may be competing with each other. To reduce false positive detections, one may want to revise IDPs to accept additional types of activities as normal, which means it will detect smaller types of abnormal activities, resulting in an increase of false negative detections.

On the other hand, to reduce false negative detections, one may want to revise IDPs to classify more types of activities as abnormal, which, at the same time, will likely make normal activities detected as abnormal, resulting in an increase of false positive detections.

Thus, how to formulate and fine tune an IDP to balance between false positives and false negatives is a challenging issue in IDS research and applications.

9.1.5.2   Behavior Classifications

To help reduce both false positives and false negatives, we may characterize behaviors as *green-light behaviors*, *red-light behaviors*, and *yellow-light behaviors*.

A green-light behavior is a normal behavior acceptable by the system. For example, any reasonable behavior from a legitimate user is a green-light behavior.

A red-light behavior is an abnormal behavior that must be rejected by the system. When a red-light behavior is detected, the ID service at the target system must issue a red-light alarm to the alarm manager at the detection center. Red-light behaviors are unacceptable behaviors.

A yellow-light behavior is a behavior that the system cannot determine, with the information it has so far, whether it is a green-light behavior or a red-light behavior. When a yellow-light behavior is detected, the ID service at the target system should issue a yellow-light alarm to the alarm manager at the detection center.

An IDS system should also specify how to respond to yellow-light behavior detections and red-light behavior detections. The following are possible reactions:

1. Pay further attention to the user with yellow-light behaviors, hoping to collect additional information to make a better determination.
2. Terminate the login session of the user with red-light behaviors.
3. Disconnect the network connection for the computer where red-light behaviors are found.
4. Shut down the computer.

9.1.6   Unacceptable Behaviors

When a user logs on to a networked computer and uses its resources, his activities may be viewed as a sequence of events. These events include using system software (e.g., browsing directories and copying files), using standard application software (e.g., using Microsoft Office, browsing the Web, sending email, and managing systems), and using user-produced software. A behavior is a sequence of events or a collection of several sequences of events. When a legitimate user uses computing resources following the system security policy, the sequence of events incurred is an acceptable behavior. An unacceptable behavior is a sequence of events that violate the system security policy.

Building an IDS faces the following two challenging issues:

1. How to define what behaviors are acceptable and what behaviors are not acceptable?
2. How to model and analyze behaviors using quantitative methods?

9.2   Network-Based Detections and Host-Based Detections

NBD and HBD are the two major detection mechanisms. NBDs analyze network packets. Host-based detections analyze system events and user behaviors. A hybrid IDS supports both NBDs and HBDs.

Depending on when detections are carried out, there are *real-time detections*, *batch detections*, and *periodic detections*. Real-time detections analyze data when it arrives, batch detections analyze data when the set of collected data has reached at a certain size, and periodic detections analyze data periodically at certain preset time.

9.2.1   Network-Based Detections

An NBD is responsible for checking packets in the network, identifying which packets are yellow-light behaviors and which are red-light behaviors, and sending warning messages to the alarm manager in the command console. It also logs packets in the event log for future analysis.

A typical NBD consists of two major components: a *network tap* and a *detection engine*. The network tap is responsible for tapping the network at selected points to gather information passing through these points. The detection engine is responsible for analyzing packets and sending warning messages to the alarm manager in the command console.

There are two types of NBDs: *network-node detections* and *network-sensor detections*. Both have the same structure. The only difference is where they are placed.

9.2.1.1   Network-Node Detections

A network-node detection NBD is placed inside a target computer, checking ingress packets and egress packets. Figure 9.3 shows a schematic of a network-node detection component.
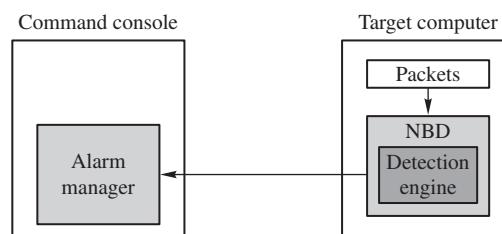
9.2.1.2   Network-Sensor Detections

A network-sensor detection NBD is placed at a selected point of the network, checking packets passing by. It needs to use a network tap (e.g., a network sniffer). Figure 9.4 shows a schematic of a network-sensor detection component.
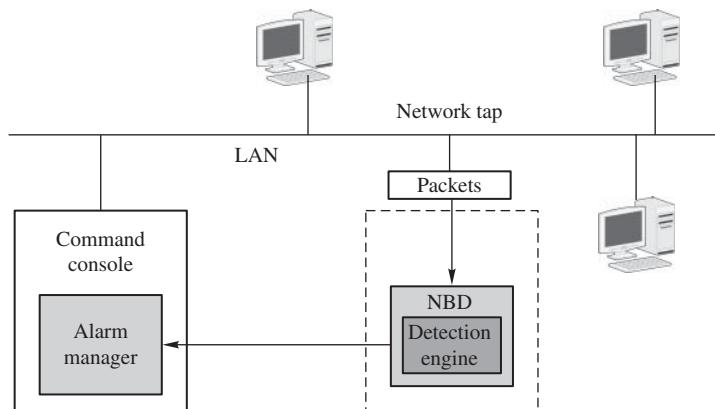
9.2.1.3   NBD Advantages

The use of NBDs has the following three advantages:

1. *Low cost*: In a well-designed large-scale LAN, one only needs to install network-sensor detection NBDs at a small number of selected points to monitor the entire network.
2. *No interference*: NBDs monitor packets passively and forward them to detection engines for analysis. Thus, NBDs do not interfere normal network traffics.
3. *Intrusion resistant*: An NBD is a small system, which can be easily made to resist intrusion. Also, it is easy to hide network-sensor detection devices in a network so that intruders will not be able to find them.

Figure 9.3   Network-node detection

Figure 9.4   Network-sensor detection

9.2.1.4   NBD Disadvantages

The use of NBDs has the following disadvantages:

1. NBDs may not be able to analyze encrypted packets, and so they cannot analyze IPsec, SSL, SSH, or other security protocol packets.
2. NBDs may not be able to handle large volume of network traffics in time, causing an increase of false negative detections.
3. Some intrusion activities, such as fragmentation-attack packets, are hard to identify by NBDs.
4. Even if it detects an intrusion behavior, it is hard for an NBD to determine whether the intrusion activity has been successfully carried out (see Section 9.3.2 for information about compound signatures).
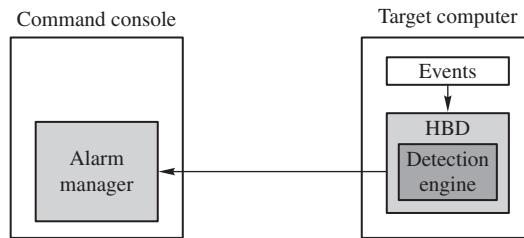
9.2.2   *Host-Based Detections*

HBDs are installed in target systems (i.e., host computers or server computers). An HBD installed in a target system checks the event log of the system and alerts the alarm manager in the command console of any red-light or yellow-light behaviors it identifies. Figure 9.5 shows a schematic of an HBD component.

The detection engine in an HBD checks the event log to identify suspicious behaviors. It also checks system logs, including any activities that try to establish, modify, or delete system files. Thus, an HBD is also referred to as a *system integrity verifier* (SIV). It creates a record of system files, including their sizes, locations, and the time they were created, and uses this record to identify intrusion behaviors.

The detection engine may also check system configurations, such as the .ini documents, .cfg documents, .dat documents, and the Windows registry.

An HBD may also keep a copy of the event log in its own storage, so that even if the intruder modifies the event log, the HBD may still uses its own event log to identify intrusions.

Figure 9.5   Host-based detection

9.2.2.1   HBD Advantages

The use of HBDs has the following advantages:

1. HBDs can detect data that are encrypted during transmissions, because encrypted data will eventually be decrypted inside the host computer.
2. HBDs can detect intrusion behaviors that cannot be detected by NBDs. For example, HBDs can detect fragment attack packets.
3. HBDs do not need special hardware devices.
4. HBDs check system logs and so can analyze system behaviors more accurately.

9.2.2.2   HBD Disadvantages

The use of HBDs has the following disadvantages:

1. HBDs are installed in the target systems, and so they require extra system managing.
2. HBDs may consume extra computing resources, including CPU time and storage space, and so they may affect normal computations.
3. Attacks that affect host computers or server computers may also affect HBDs.
4. HBDs cannot be installed in routers or switches.

9.3   Signature Detections

Signature detections and statistical analysis are major intrusion detection techniques used in both NBDs and HBDs. Signature detections are also referred to as *operational detections*. Signature detections inspect current events (i.e., events that occurred in a small interval of the current time) and decide whether these events are acceptable. Signature detections are often associated with a set of rules. Thus, signature detections are also referred to as *rule-based detections*.

For example, one may specify a set of behavior rules as follows:

1. System files, particularly the password files, should not be copied by users.
2. Disks should be accessed only by operating system utilities. That is, users should not access disks directly.
3. Users should not probe other users' personal directories.

4. Users should not copy files to other users' personal directories or to system directories.
5. Users should not modify other users' files.
6. Users should not keep trying to log on to their accounts if three attempts have failed.
7. Users with higher levels of clearance should not copy files from directories with higher level secrecy to directories with lower level secrecy.
8. Users with lower levels of clearance should not read files in directories with higher level secrecy.

Signature detections include *network signatures* and *host-based signatures*.

9.3.1   Network Signatures

Network signatures provide information of packet behaviors that may affect the normal execution of the system. Network signatures consist of *header signatures* and *payload signatures*. Payload signatures are also referred to as *content signatures*.

9.3.1.1   Payload Signatures

Checking packet contents is a basic intrusion detection technique in NBD systems. It uses payload signatures to determine which packets are acceptable and which packets are not acceptable. We illustrate this using an example. Suppose that an intruder attempts to use the standard FTP application program from a remote computer, denoted by so.com, to execute a selected program on the target computer, denoted by de.com. This example illustrates a common intrusion technique to read a system file in the target system. The following commands are payloads of IP packets transmitted from so.com to de.com:

```
so.com -> de.com ETHER TYPE=0800 (IP), SIZE=68 bytes
so.com -> de.com IP D=129.63.8.1 S=129.63.8.12 LEN=54,
         ID=44340
so.com -> de.com TCP D=21 S=28613 ACK=2132480783
         SEQ=1358787809 LEN=14 WIN=61320+
so.com -> de.com FTP C PORT=28113 SITE exec cat+\verb
         +/etc/passwd\r\n
```

The first three lines in this example appear innocent, and so they may be considered acceptable. The last line is unacceptable, for it attempts to download the /etc/passwd system file of the target system.

9.3.1.2   Header Signatures

Checking packet headers is another basic intrusion detection technique in NBD systems. It uses header signatures to identify malicious packets. For example, in a *broadcast attack*, the attacker sends packets to the targets in which the source address and the destination address are the same. Broadcast attacks may cause the target system to crash. Thus, when it finds that a packet has the same source address as the destination address, the IDS should inform the alarm manager about it.

9.3.2 Host-Based Signatures

Host-based signatures provide information of event behaviors that may affect the normal execution of the system. For example, "three consecutive failed logins" is a host-based signature, which means that someone enters passwords consecutively three times, but none of them is successful. This event may occur if a legitimate user forgets his password, or if an intruder tries to log on to the computer as a legitimate user. When a "three consecutive failed logins" is detected, the HBD component should inform the alarm manager about it.

Host-based signatures can be characterized as *single-event signatures*, *multievent signatures*, *multihost signatures*, and *compound signatures*.

9.3.2.1 Single-Event Signatures

A suspicious behavior in a single command is a single-event signature. The following are several activities with single-event signatures:

1. A single command that modifies a system file.
2. A single command that reads another user's personal directory.
3. A single command that modifies another user's file.
4. A single command that performs direct disk I/O. That is, it does not use disk drivers provided by the underlying operating system to perform disk I/O's.
5. A single command that duplicates system files.

9.3.2.2 Multievent Signatures

A sequence of several single-event signatures forms a multievent signature. For example, the "three consecutive failed logins" is a multievent signature, for it consists of three single events. This is a suspicious multievent signature. For another example, the signature that "users would often use the same documents they worked on in previous logins" is a multievent signature. This is an acceptable signature.

9.3.2.3 Multihost Signatures

A multihost signature is a signature formed from a sequence of single-event signatures and multievent signatures on several different hosts. For example, the following activities have a multihost signature:

> A user attempted to log on to machine α and failed. He then attempted to log on to machine β and failed again. After this, he attempted to log on to machine γ and still failed.

9.3.2.4 Compound Signatures

Some intrusion behaviors are hard to identify if only network-based signatures or host-based signatures are available. This is because a host-based attack launched from a remote computer

Table 9.3 Examples of compound signatures

Network-based activities	Host-based activities	Compound signatures
A user uses FTP to log on to the system and uses cd and ls commands	A user browses the etc directory and reads the passwd file	A user browses system files from a remote computer
A user uses FTP to log on to the system and uses the put command	The files uploaded to the system have virus and Trojan horse signatures	A user uploads malicious software to the system from a remote computer
A user uses FTP to log on to the system and uses the put command	A user modifies system files and registry entities	A user modifies system files from a remote computer
A certain Web attack	Read system executable files	A Web attack is successful

may not provide the source of the attack. For example, suppose that an attacker is able to impersonate a system administrator of a target computer to log on to the target system from a remote computer. Using host-based signatures, the host-based IDS may only detect that the intruder is modifying certain system files, which is acceptable because the intruder is impersonating the system administrator. However, if network-based signatures are also available to allow the IDS to detect that this login is from a remote computer different from any of the remote computers the system administrator would normally use, then the IDS may be able to determine that this is an intrusion act. Note that in this case, if only network-based signatures are available, then even if the IDS detects that the login is from a remote computer, it may not be able to detect that this login is an intrusion act without knowing that system files are modified, if remote logins to the target system are allowed.

Thus, it is necessary to combine network-based signatures and host-based signatures to form *compound signatures* to more accurately detect intrusion behaviors. Table 9.3 provides several examples of compound signatures.

9.3.3 Outsider Behaviors and Insider Misuses

People who have authenticated access to a computer system are referred to as insiders of the system. People who do not have authenticated access to the system are referred to outsiders of the system. An outsider of the system becomes an insider once he obtains authenticated access to the system, either legitimately or illegitimately. Thus, an attacker is an outsider before he obtains authenticated access to the system. But he becomes an insider once he gains authenticated access to the system.

Malicious outsiders and insiders behave differently from legitimate users. Outsider behaviors and insider misuses can be used to detect the presence of intruders.

9.3.3.1 Use Outsider Behaviors to Detect Intrusions

Malicious outsiders, wanting to become insiders, may try to gain authenticated access to the system. They may, for example, plant a Trojan horse in the system to allow them change system configurations to get into the system. They may also try to hijack a TCP connection to enter the system. Another example of outsider behaviors is a *sweeping attack*, in which the malicious

outsider, wanting to find weak links in the victim's firewall, "sweeps" the firewall by sending a series of probing packets. Sweeping attacks often demonstrate well-known packet patterns that could be detected easily.

9.3.3.2 Use Insider Misuses to Detect Intrusions

Once a malicious outsider becomes an insider, he can do anything a legitimate user is allowed to do. Because he is malicious, he would do things that legitimate users would not normally do, which can be used to detect the presence of a malicious insider. For example, a malicious insider may try to copy system files, change system configurations, or read mission-critical objects.

9.3.4 Signature Detection Systems

Typical intrusion detection systems use one of the following three methods for modeling signatures: built-in system, programming system, and expert system.

9.3.4.1 Built-in System

This type of intrusion detection systems stores a set of detection rules inside the system and provides an IDS editor to the user, allowing users to select rules on the basis of their needs.

Examples of built-in IDS systems include CyberSafe Corporation's Centrax and Internet Security Systems' SafeSuite.

9.3.4.2 Programming System

This type of intrusion detection systems provides a set of default rules and a programming language (or a scripting language), allowing users to select default rules and write their own rules. This model provides users with flexibilities, but it also requires users to learn to use the programming language.

Examples of programming IDS systems include Axent Technologies' ITA and Haystack's Stalker, where ITA allows users to use a special scripting language designed by Axent Technologies to write their own detection rules, while Stalker allows users to use the C language to write detection rules.

9.3.4.3 Expert System

This type of intrusion detection systems is directed toward meeting special needs for a special organization. Such a system requires domain experts to define detection signatures.

For example, to build an IDS to detect intrusions related to the banking business, it may need special knowledge from banking experts. Expert detection rules will then be used to construct a special-purpose detection engine.

Early IDS systems were expert systems, of which SRI International's real-time intrusion detection expert system (IDES), funded by U.S. Navy, has influenced the research and development of intrusion detections in many ways. The precursor of SRI International is Stanford Research Institute created by Stanford University in the 1940s.

9.4   Statistical Analysis

When the difference between acceptable events and unacceptable events can be quantified, we can use statistical analysis to identify unacceptable events on the basis of quantified event measures. There are two common approaches. One approach uses threshold values of certain measures. The other approach uses user profiles.

Using threshold values, an IDS system counts the number of occurrences of certain types of events during a period of time and considers it an intrusion behavior when the count exceeds a predetermined threshold value regardless of who the users are. Threshold detection is simple to implement, but it is inaccurate.

Using user profiles to detect intrusions is more accurate. It collects past events of a user or a group of users to create user profiles on the basis of certain quantified measures. Certain parameters of events can be quantified in a natural way. The following are some examples of quantifiable events:

1. The time a particular event occurs.
2. The number of times a particular event occurs in a period of time.
3. The current values of system variables.
4. The utilization rate of system resources.

On the basis of these parameters, we may define the following four event measures: event counter, event gauge, event timer, and resource utilization.

9.4.1   Event Counter

We may use an integer variable for each type of events to record the total number of times this type of events occurs in a fixed period of time. This variable is referred to as an *event counter*. Different types of events use different event counters. The value of an event counter, starting from zero, is increased by one each time an event of the same type occurs.

Event types are defined on the basis of particular situations. For example, we may treat all login events of a user during a fixed period of time the same type of events and view each login from the user a single event. Likewise, we may treat all login events (which may come from different users) that occurred on the same host computer during a fixed period of time the same type of events and view each login on the system a single event. We may also treat the number of executions of a particular command during a fixed period of time the same type of events.

Note that some events could cause event counters to reset. For example, an event counter of a user login could be reset if a login is successful after two or three unsuccessful login attempts.

9.4.2   Event Gauge

We may use an integer variable for each measurable object in the system to denote the current value of the object. This variable is referred to as an *event gauge*. For example, the buffer space used by a TCP server is a measurable object, where the current number of packets stored in the buffer is the current value of the variable. Different objects use different event gauges. The values of an event gauge are non-negative integers, which may be increased and may be decreased.

9.4.3 Event Timer

We may use an integer variable for two related events in the system to denote the time difference of the occurrences of the first event and the second event. This variable is referred to as an *event timer*. For example, we may use an event timer to record the time difference between two consecutive logins of the same user. We may also use an event timer to record the time difference between a user login and the first program the user executes after login.

9.4.4 Resource Utilization

We may use a variable for each resource in the system to record the utilization of the resource during a fixed period of time. This variable is referred to as *resource utilization*. For example, the CPU time needed to execute a program, the number of times a user prints in each login, and the number of email messages a user sends out in each login.

9.4.5 Statistical Techniques

On the basis of the event measures introduced in Sections 9.4.1–9.4.4, we may use statistical methods to analyze events and identify unacceptable events. Common statistical methods include the mean and standard deviation, multivariate analysis, Markov process, and time series analysis.

The mean and standard deviation is the simplest statistical method. We may use it to identify intrusion activities by comparing average event frequency and its standard deviation with the normal average event frequency and its standard deviation. For example, we may calculate the mean and standard deviation of login frequency and execution frequency of a particular user at a certain time period and compare them with the normal login frequency and execution frequency. Any substantial discrepancy would indicate intrusion acts.

We may use multivariate analysis to analyze two or more related variables at the same time to identify anomalies. For example, we may obtain more information by considering the CPU time and resource utilization at the same time.

Markov process can be used to calculate the probability that the system is changed from one state to another state. Time series analysis can be used to study event sequences to find out anomalies.

Large corporations often hire special groups of experts whose sole responsibilities are to analyze IDS alerts, refine detection algorithms, and provide information to the incident response teams.

9.5 Behavioral Data Forensics

Behavioral data forensics studies how to use data mining techniques to analyze event logs and search for useful information. Any information that indicates past, current, and future intrusion activities would be particularly interesting.

The goal of data mining is to devise algorithms to search for useful information from large data sets. In the scope of intrusion detection, after collecting raw data (events) from various target computers, the command console needs to mine these data to identify intrusion behaviors.

Raw events may have been recorded in various formats. For example, the raw events collected by NBDs are raw TCP/IP packets, the raw events collected by HBDs may be in operating system formats, and the system logs may be in the ASCII format.

The amount of raw data collected from a server computer may reach 10 MB per day. Thus, if an organization has 100 servers, then the total amount of data collected per day may reach 1 GB. Analyzing a large volume of data such as this requires efficient data mining tools.

9.5.1   Data Mining Techniques

Data mining uses the following common techniques:

Data Refinement
Data refinement is a technique for improving data representations to help find new information. This technique is used to make useful features stand out.

Contextual Interpretation
Contextual interpretation is a technique to interpret data according to its context to help find new information. Interpreting data from a different point of view would often yield new meanings of the data that were undiscovered previously.

Source Combination
Source combination is a technique that combines different types of data sources to find new information from new perspectives.

Out-of-Band Data
Out-of-band data is a technique that combines data outside of the scope of intrusion detection to help find new information.

Drill Down
Drill down is a technique that starts from a higher level of activities. Once a specious behavior is spotted, look for lower levels of activities to find out more information.

9.5.2   A Behavioral Data Forensic Example

Suppose from analyzing a system log that Cathy, the system manager in a company located in Chelmsford, Massachusetts, found that user John had 203 successful logins during the last 30 days, which was substantially larger than his average number of logins in any given 30-day period. This triggered Cathy to drill down and find out what John was doing on these logins by checking a lower level activity report. Cathy found that John logged in regularly around 8:30 p.m., executed a program named mytest, and logged out 2 hours later. Using an out-of-band data resource, Cathy learned that John actually went to see a ballet show from 8:00 p.m. to 10:00 p.m. in the Wang Center for the Performing Arts in Downtown Boston on a day that John was also recorded logging in at 8:30 p.m. This made Cathy believe that John's account had been compromised by an intruder. Further investigation on what the program mytest did confirmed Cathy's suspicion.

9.6 Honeypots

The use of decoy machines to direct intruders' attention away from the machines under protection is a major technique to preclude intrusion attacks. Any device, system, directory, or file used as a decoy to lure attackers away from important assets and to collect intrusion behaviors is referred to as a *honeypot*.

A honeypot may be implemented as a physical device or as an emulation system. The idea is to set up decoy machines in a LAN, or decoy directories/files in a file system and make them appear important, but with several exploitable loopholes, to lure attackers to attack these machines or directories/files, so that other machines, directories, and files can evade intruders' attentions. A decoy machine may be a host computer or a server computer. Likewise, we may also set up decoy routers or even decoy LANs.

Honeypots can also be used for researchers to study intrusion techniques, formulate event signatures, and design intrusion countermeasures. Such honeypot is often referred to as *research honeypot*. A honeypot deployed in production networks or systems in an organization, which is not used for research purpose, is referred to as *production honeypot*. While a honeypot may be used for both research and production, in general, a research honeypot is not required to be an industry-grade product.

Thus, a honeypot is set out to do two things:

1. Help its owner to know the enemies.
2. Sacrifice itself to save the other assets.

Honeypots are deliberately set up to deceive and trap intruders to reveal their motivations, intentions, tactics, techniques, and tools through packet capturing, analysis, and controls. Honeypots have become an important component in the network security infrastructure.

9.6.1 Types of Honeypots

Early honeypots, developed in 1990, were physical systems. They were simply host computers connected to unprotected LANs with real IP addresses. They were operated on unpatched operating systems with default configurations. However, physical systems often require high-level interactions between the honeypot daemon and the operating system running it. It may also require substantial efforts to maintain a physical honeypot.

Since the late 1990s, researchers have developed new software techniques to construct virtual honeypots by emulating operating systems or network services. They are easy to deploy and require low-level interactions between the honeypot daemon and the local hard disk. For example, Honeyd is a network-based emulation honeypot and KFSensor is a host-based emulation honeypot for the Windows operating system. Other honeypot emulation software includes CyberCop's StingandDeception Toolkit.

The MWCollect Alliance (`http://www.mwcollect.org`) and the Honeynet Project (`http://www.honeynet.org`) are international organizations of white-hat hackers devoted to studying and developing honeypot technologies.

MWCollect projects include Nepenthes, Honeytrap, and HoneyBow.

Honeynet projects include Honeywall CDROM, Sebek, and Interaction Honeypot Analysis Toolkit (HIHAT).

Specialty honeypots are designed specifically to deal with a certain type of intrusions. For example, honeypots that are designed to handle spam mails are referred to as *spam honeypots* or *spam traps*.

Honeypot functionalities may also be distributed among several honeypots to form a distributed honeypot. The common architecture of a distributed honeypot consists of a centralized cluster of high-interaction honeypots and a distributed low-interaction honeypots across the local area network.

9.6.1.1   Interaction Levels

The interaction level of a honeypot is characterized as follows:

Low Interaction
A honeypot has low interaction if its daemon only writes to the hard disk of the local host.

Mid Interaction
A honeypot has mid interaction if its daemon reads from and writes to the hard disk of the local host.

High Interaction
A honeypot has high interaction if its daemon interacts with the operating system of the local host and through the operating system interacts with the local hard disk and other resources.
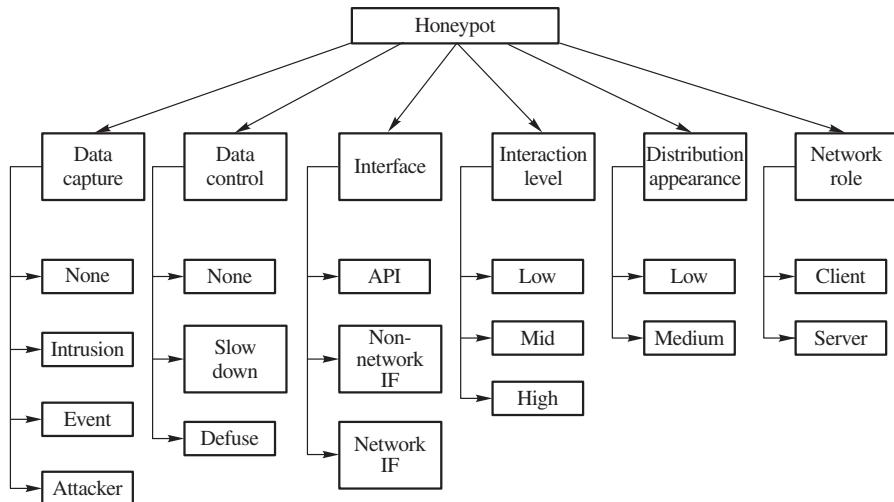
9.6.1.2   Honeypot Functionalities and Characterizations

A typical honeypot should consist of the following components: data capture, data control, and interface. The data capture component is used to capture intrusion activities, events, or attackers. The data control may either slow down intrusion activities or defuse attackers. The interface component may provide an API, a non-network implementation facilitator (IF), or a network IF.

Honeypots may be characterized by their interaction levels, distribution appearances, and network roles. In the last characterization, there are client honeypots and server honeypots. Honeypot functionalities and characterizations are summarized in Fig. 9.6.

9.6.2   Honeyd
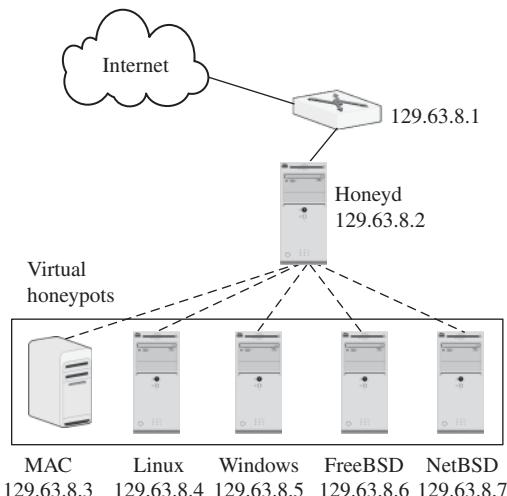
Honeyd is an engine for running virtual IP protocol stacks in parallel. It provides a lightweight framework for constructing virtual honeypots at the network level. A single instance of the Honeyd daemon can simulate standard network services (such as SMTP, FTP, and ICMP) running different operating systems on several different virtual hosts simultaneously. Honeyd is designed to detect and disable worms, distract intruders, and prevent the spread of spam mails.
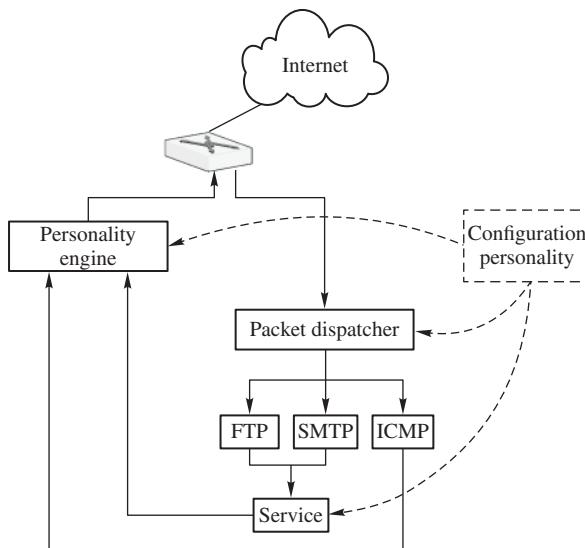
Figure 9.6 Honeypot functionalities and characterizations

9.6.2.1 Virtual Framework

Honeyd virtual honeypots appear to run on real IP addresses, but these addresses are not physically allocated. Honeyd receives network packets for virtual honeypots via a router or a Proxy ARP. Honeyd replies to network packets sent to virtual honeypots. Figure 9.7 shows a schematic of Honeyd.

Figure 9.7 Schematic of Honeyd. A single instance of Honeyd daemon simulates the network stack behaviors of different operating systems on different virtual hosts

Figure 9.8   A block diagram of Honeyd architecture

Let $A.B.C.x$ and $A.B.C.y$ denote, respectively, the IP addresses of the router and the Honeyd host. For simplicity, assume that all virtual honeypots are installed in the same domain with IP addresses

$$A.B.C.v_1, \cdots, A.B.C.v_k.$$

Suppose that an intruder sends an ingress packet to the honeypot at $A.B.C.v_i$ from the Internet. The local router receives the packet and forwards the packet to the virtual honeypot residing in the Honeyd host with IP address $A.B.C.y$. This can be done by modifying relevant routing entries in the router to direct packets sent to $A.B.C.v_i$ to the Honeyd host. If no special route is provided, the router will use a Proxy ARP to allocate the MAC address of the virtual honeyhost, which configures the Honeyd host to reply to ARP requests for $A.B.C.v_i$ with the Honeyd host's own MAC address. Honeyd is a mid-interaction honeypot.

9.6.2.2   Personality Engines

As attackers may use a fingerprinting tool (e.g., Xprobe) to find out whether a target system is a honeypot, it is important to make a honeypot look like a valuable target when it is fingerprinted. Honeyd does this by simulating the network stack behavior of the underlying operating system supposed to run on the target system. This is referred to as the personality of a virtual honeypot.

Different virtual honeypots have different personalities. Honeyd provides virtual honeypot personality through a personality engine. In particular, the personality engine introduces appropriate changes to the headers of every egress packets to make them meet the expectation of the operating system supposedly running on the target system, which is now just a virtual honeypot. Figure 9.8 shows a block diagram of the Honeyd architecture.

Ingress packets are dispatched by a packet dispatcher to the corresponding protocol handler. For TCP (such as SMTP and FTP) and UDP protocols, a service component is needed. Egress

packets will then be modified by the personality engine so that they appear to come from the correct network stacks as the intruder would expect.

9.6.3 MWCollect Projects

Nepenthes, Honeytrap, and HoneyBow are honeypot tools developed by the MWCollect Alliance, which are specifically used to lure malicious software away from attacking the assets under protection. Nepenthes and Honeytrap are low-interaction honeypots, while HoneyBow is a high-interaction tool. Nepenthes and Honeytrap are used to set up virtual honeypots to collect autonomously spreading, worm-like malware. HoneyBow is a high-interaction physical honeypot.

While Nepenthes can only trap known malicious software, Honeytrap can trap unknown attacks. Honeytrap handles ingress packets to unbound TCP ports. When it detects a connection attempt to an unbound TCP port, the Honeytrap daemon will take over the network server to handle the TCP connection. In particular, Honeytrap can extract TCP connection attempts from a network stream. This may be done using a sniffer to catch TCP reset packets with a zero sequence number, for such packets indicate that the corresponding TCP connection requests are denied.
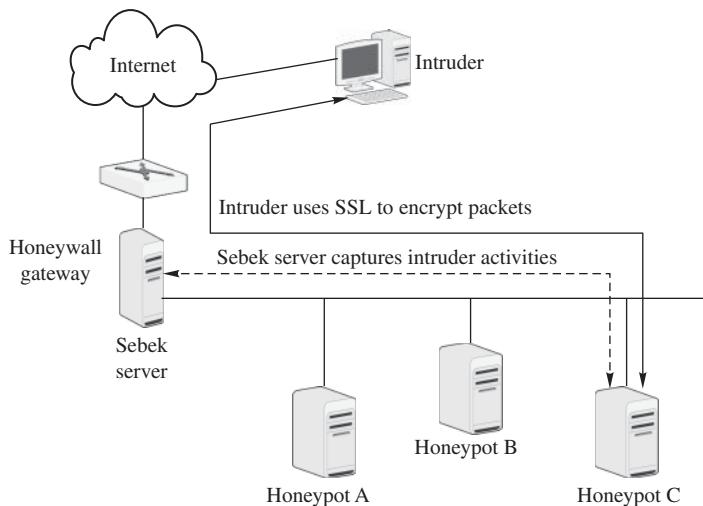
9.6.4 Honeynet Projects

A honeynet is a network of real honeypots that operates on real operating systems. While it may be costly and time consuming, honeynets are capable of capturing more information. In particular, a honeynet is a reachable decoy network. It uses a stealth inline network device, called a *honeywall*, to monitor and control ingress and egress packets to and from real honeypots in the network. A honeywall is similar to a firewall, except that honeywall never blocks packets. The following are several common Honeynet tools:

1. *Honeywall CDROM*: A honeywall is a hardware device for constructing a Honeynet. Honeywall CDROM is a bootable CD. It contains all the functionality of Honeywall. When booted from a Honeywall CDROM, the host computer copies the functionality of Honeywall and constructs a Honeywall. It allows users to capture, control, and analyze packets.
2. *Sebek*: Sebek is used to capture attacker activities on real honeypots, even if they are encrypted by IPsec, SSL, or SSH. In particular, Sebek can recover keystrokes, passwords, and uploaded files. Figure 9.9 shows a schematic of a Sebek honeypot.
3. *High-Interaction Honeypot Analysis Toolkit (HIHAT)*: This is a Web tool that can transform arbitrary PHP (Hypertext Preprocessor) applications into Web-based high-interaction Honeypots. HIHAT can be used to monitor and analyze packets to and from the honeypot. Moreover, it can generate an IP-based geographical mapping of the attack sources.
4. *HoneyBow*: HoneyBow is a high-interaction malware collection honeypot sensor.

9.7 Closing Remarks

How to detect intrusion activities effectively and efficiently is an active research area. Understanding operating systems and network protocols, particularly from the system administration

Figure 9.9   Schematic of Sebek honeynet

point of view, will help design better intrusion detection systems. Computational statistics and data mining technologies have played an important role in IDS. Honeypot technologies provide a different way to counter intrusions by luring intruders to attack certain deliberately created sacrificial assets. Understanding operating systems and network protocols is the key to construct honeypots.

9.8   Exercises

9.8.1   Discussions

9.1.   Discuss why it is possible to automatically detect intrusion activities.

9.2.   Describe your experience of using an IDS/IPS system (if you have used one).

9.3.   Some users comment that setting up an IDS/IPS system is not an easy task. Why do you think it is the case?

9.4.   If an IDS system produces a lot of false positive alarms or a lot of false positive alarms, what do you suppose it happens and how would you suggest to fix it?

9.5.   Is it possible for an IDS system to produce a lot of false positive and false negative alarms in a short period of time and why?

9.6.   Describe your experience of deploying a honeypot product (if you have used one).

9.8.2   Homework

9.1.   Similar to Fig. 9.2, draw a block diagram of a NBD IDS.

9.2. Download and install Snort from `http://www.snort.org`. Describe how to use Snort as a NBD tool.

9.3. Similar to Fig. 9.2, draw a block diagram of a HBD IDS.

9.4. Any device inside a LAN can be a target for intrusion. Network taps can be placed at any point in the LAN. Suppose that the command console is installed on a bastion host in a DMZ. Draw a block diagram of such an IDS system.

9.5. One may distribute the command console components (part or all of them) across several target systems to form a distributed IDS. Draw a block diagram of a distributed IDS that provides an overview of the system and detailed information between components.

9.6. Explain why the four listed activities in the subsection of "Single-Event Signatures" are harmful.

9.7. Explain why "a user would often use the same document he used in a previous login" is a multievent signature. Why is it an acceptable signature?

9.8. Explain why the example given in the subsection of "Multihost Signatures" is a multihost signature and why it is suspicious.

9.9. Assume that an IDS system detects the following payload signatures. Which signatures are acceptable and which signatures are not acceptable? Justify your answers.

```
so.com -> de.com ETHER TYPE=0800 (IP), SIZE=68 bytes
so.com -> de.com IP D=129.63.8.1 S=129.63.8.12 LEN=54,ID=44340
so.com -> de.com TCP D=21 S=28613 ACK=2132480783
        SEQ=1358787809 LEN=14 WIN=61320
so.com -> de.com FTP C PORT=28113 SITE exec cat /etc/hosts\r\n
so.com -> de.com FTP C PORT=28113 SITE exec cat /etc/services
\r\n
```

9.10. Given an example of an unacceptable payload signature not mentioned in the textbook. Justify why it is unacceptable.

9.11. Given an example of an unacceptable header signature not mentioned in the textbook. Justify why it is unacceptable.

9.12. Give an example of an unacceptable single-event signature not mentioned in the textbook. Justify why it is unacceptable.

9.13. Give an example of an unacceptable multievent signature not mentioned in the textbook. Justify why it is unacceptable.

9.14. Give an example of an unacceptable multihost signature not mentioned in the textbook. Justify why it is unacceptable.

9.15. Give an example of an unacceptable compound signature not mentioned in the textbook. Justify why it is unacceptable.

*9.16. Ask your system administrator for a system log during the past 2 weeks and analyze whether there are intrusion activities. Write a short paper of about 4000 words to describe your methods and results.

9.17. If an IDS has a high false positives, what would be a possible reason? Can you suggest a way to reduce false positive detections.

9.18. Assuming a HBD IDS is used to monitor system files. Under what situations can false positives occur? How can you reduce false positive detections?

*9.19. The following are commercial NBD IDS products and their vendors. Select four products from this list, and write a paper of about 4000 words to describe these products.
1. BlackICE by Network ICE
2. Dragon by Network Security Wizards
3. NFR by Network Flight Recorder
4. NetRanger by Cisco Systems
5. NetProwler by Axent Technologies
6. eTrustID by Computer Associates

*9.20. The following are commercial HBD IDS products and their vendors. Select four products from this List, and write a paper of about 4000 words to describe these products.
1. Computer Misuse Detection System (CMDS) by ODS Networks
2. Kane Security Monitor (KSM) by ODS Networks
3. SecurreCom8001 by ODS Networks
4. Intruder Alert (ITA) by Axent Technologies
5. PSAudit by Pentasafe
6. OperationsManager by Mission Critical

*9.21. The following are commercial hybrid (i.e., with both network-based and host-based detection) IDS products and their vendors. Write a paper of about 4000 words to describe these products.
1. Centrax by CyberSafe Corporation
2. CyberCop by Network Associates
3. RealSecure by Internet Security Systems (ISS)

9.22. Visit `http://honeytrap.mwcollect.org/attacks` and describe 10 most recent attacks trapped by Honeytrap.

9.23. A target browsing behavior may or may not be threatening, which is a point of interest of conducting behavioral data forensics. Give an example of analyzing browsing behaviors to help identify intrusion activities.

9.24. If a certain user is browsing mission-critical files from directory to directory and from host to host, then it is a suspicious activity. This is a point of interest of conducting behavioral data forensics. Given an example of analyzing critical file browsing trend to help identify intrusion activities.

9.25. "Recently, we launched a new service and were asked to create a nonsensical name for the DNS entry," said a reader, "so that if a hacker enumerated our DNS they would be unable to identify its worth. The company operates dozens of these servers with nonsensical names, and it is really hard to figure out where services are."

Do you think that this measure would be useful to counter intruders? Justify your answers.

9.26. Suppose that you have a spare older computer that you wish to use it to set up a honeypot. Download and install Honeytrap from `http://www.mwcollect.org` to make this computer a honeypot.

*9.27. On a Linux system, describe how to detect a connection attempt to an unbound TCP port and how to take over the network server to handle the TCP connection.

9.27. "I have setup an IDS system to support a firewall system," a reader told us. "The IDS was a part of a business security appliance, and I configured the IDS using the appliance Web UI. There is a lot more regarding working with and setting up IDS. It gets very complicated and time consuming in a large organization. This process requires extensive knowledge working with additional third part tools and application. Troubleshooting can then become overbearing." Why do you think setting up an IDS was difficult?

9.28. "My experience in managing network-based IDS/IPSs is that this appliance is very difficult to setup correctly based on organizational security policies," said a user. "Most default configurations do not fully comply with specific policies out-of-the-box and the effort to get the settings correct is a long process, requiring continuous management." Can you suggest a way to improve this process?

9.29. Conduct a survey on the following open-source IDS products: Snort, Suricata, Bro, Kismet, OSSEC, and Samhain. Explain what each of these products does and how to set them up. (Hint: you may consult Joe Schreiber's blog on `http://www.alienvault.com/blogs/security-essentials/open -source-intrusion-detection-tools-a-quick-overview`.

*9.31. *Node subversion* is a serious security threat in ad-hoc WSNs (see Exercise 6.9.2 for a description of WSNs). Attackers may be able to penetrate legitimate sensor nodes using reverse-engineering techniques, or replace them with attackers' own malicious sensor nodes. Thus, how to detect subverted sensor nodes is an important issue. Design an efficient intrusion detection mechanism that is suitable for WSNs and justify your designs.

9.32. "I once developed a new way of doing a meeting registration," a reader told us, "and my boss was thrilled since it was so much easier to maintain with this configurable technique. The first meeting came and went as a big success. A few months later when I got in one morning there were almost 100 error messages where it was clear that people were trying to hack into our server using SQL injection (I noted that the querystring parameter were part of the error messages). I looking up the IP addresses, and it turns out they were coming from a technical school in a foreign country! Since

that page wasn't actively in use, I took it offline, and did not put it back online until I had come up with a security model that matched all the rest of the site."

Why do you think the attacks came so late? Explain what the attackers were trying to achieve. Justify your answer.

9.33. Once a subverted node is detected in an ad-hoc WSN, the good nodes or the base station may want to take it out of the network by not receiving its data or not communicating with it. However, malicious nodes may also be able to impersonate good nodes to take out good nodes out of service. To deal with this issue, researchers have proposed to use suicide nodes to take out subverted nodes. That is, once a node A detects a certain node M is malicious, the node sends a signed suicide note $E_K(A, M)$ to other nodes. After verifying the signature of the suicide note, the other nodes will de-associate with both nodes A and M. Thus, node A sacrifices itself for the common good. Discuss the pros and cons of this approach.

10

The Art of Anti-Malicious Software

Malicious software, coded intentionally by malicious programmers, is used to inflict damage to other people's computers, including hardware resources, files, system programs, and application programs; steal other people's data; or exploit other people's computer resources. Malicious software sneaks into an internal host through software loopholes or improper system configurations; or by luring unvigilant users to copy or download it to their computers. Ignorance and negligence of computer users are a major factor contributing to malicious software being wide spread. Business travelers, a.k.a. road-warriors, who use public access points and other untrustworthy networks also present enormous risks to corporate networks.

It is evident that firewalls and IDS/IPS alone are not sufficient to stop malicious software from entering internal computers. New methods are needed to detect, block, and remove malicious software. For this purpose, we need to understand how malicious software is structured, how it lives, and how it disseminates. In every art, there are a few principles and many techniques. The art of anti-malicious software is no exception. To learn this art, we will need to understand common kinds of malicious software, which include viruses, worms, trojan horses, spyware, and zombieware. Malicious codes that exploit software flaws and configuration loopholes in Web systems are also common.

This chapter is focused on virus defense, Web security, and DDoS defense methodologies.

10.1 Viruses

A computer virus is a piece of code hiding in a program that can automatically copy itself or embed a mutation of itself in other programs. A computer worm, on the other hand, is a standalone program that can automatically replicate itself to other host computers through networks. Worms may be viewed, roughly, as network viruses.

The program that contains a viral code is referred to as a *host* program. A host program of a virus is sometimes referred to as an *infected program*. A program cleared of all viruses is referred to as an *uninfected program* or a *healthy program*. It is also called a *disinfected program* if the program was once infected with a virus but now the virus has been removed.

When the host program is not executed, the virus code hiding in it can do nothing. Only when the host program is executed will the virus code be activated and do something. Computer viruses are a special kind of parasitism.

Recent development of computing technology has allowed viruses to live in files typically not viewed as executable programs. These files include PDF files, office documents, and images, to name just a few. Viral codes contained in these files may be activated by the application programs that process them. Thus, we also use *host files* and *infected files* to denote files that contain viral codes.

10.1.1 Virus Types

Composing viral code is a competitive and wicked sport played by malicious programmers. Since the first creation of a computer virus in the 1980s, numerous types of viruses have been created and spread on almost all common platforms.

Viruses are specific to particular types of file systems, file formats, and operating systems. A virus that can infect one type of file system, for example, may not work in a different type of file system. Viruses are also specific to particular types of architecture, CPU, languages, macros, scripts, debuggers, and every other form of programming or system environment. While a typical virus that works in one environment may not work in a different environment, there are also viruses that may work in multiple environments. Viruses of this kind are typically written as platform-independent code. When such a virus gets in a particular environment, its platform-independent code will be translated to the local format and infect the local host files.

10.1.1.1 Classification Based on Host Programs

The following are common types of viruses according to the types of host files they would infect or where they live.

Boot Virus
These types of viruses infect the boot program of a host computer that resides in the boot sector. A boot virus uses the computer's boot sequence to activate itself. Once activated, it may modify the operating system to intercept disk access and infect other disks. It may also infect an updatable BIOS of a PC computer. For example, Elk Cloner and Cascade are boot viruses that could infect, respectively, Apple II computers and PC computers.

Boot viruses typically copy the original boot sector to another location. Thus, if two boot viruses store the original data in the same location and hit the same machine, then the second infection will replace the original boot sector copied to a new location with the new code in the first infection, causing the original boot sector to be lost permanently. The Stoned Empire Monkey virus, for example, is a virus of this kind.

File-System Virus
These types of viruses infect the file system of a host computer. A file is typically stored on a disk as a group of clusters, where each cluster is stored in contiguous sectors and different

clusters are stored in noncontiguous sectors. Thus, the file system maintains a table of pointers, where each pointer points to the first cluster of a file. A file-system virus may overwrite table entries and spread itself through file systems. For example, DIR-II is such a virus that infects the File Allocation Table (FAT) file system of Microsoft's DOS operating systems. The NTFS stream viruses and NTFS compression viruses have been created to infect Microsoft's New Technology File System.

File-Format Virus

These types of viruses infect individual files. For example, a COM virus infects binary files with the `.com` extension. An EXE virus infects executable files with the `.exe` extension. A DLL virus infects Dynamic Link Library (DLL) files with the `.dll` extension. An ELF virus infects executable files in UNIX, where ELF stands for executable and linking format. Device-driver viruses have also been created to infect driver files of Windows XP. Win32 and Win64 are the most recent types of viruses targeted, respectively, at 32-bit Windows and 64-bit Windows operating systems.

Macro Virus

These types of viruses infect documents that contain macro codes. In particular, macro viruses have been created to infect Microsoft Office documents that allow users to include macro codes to enhance processing capabilities. These documents include Word, Excel, PowerPoint, and Visio documents. For example, macros may be added in a Word document to check spelling automatically when the document is closed. Thousands of macro viruses have been created. The WM/DMV virus and the XM/Larous virus, created in the mid-1990s, were the first known macro viruses that infected, respectively, Word documents and Excel spreadsheets. Macro viruses have also been created for different language versions (e.g., Chinese, Japanese, and Russian) of the Microsoft Office programs.

Script Virus

These types of viruses infect script files, which include UNIX scripts, Visual Basic scripts (VBScript), Java scripts (JScript), and batch files. Script viruses typically replicate themselves in the form of email attachments, office documents, and Web documents. Thus, they are also classified as worms. For example, LoveLetter was a VBScript virus that spread rapidly in 2000 through email.

Registry Virus

These types of viruses infect the Microsoft Windows registry, where the Windows registry is a database storing settings and options for Windows operating systems and most nonsystem software. For example, the Happy99.exe virus infected the Registry by inserting a new key in the Registry when it is executed.

Memory-Resident Virus

These types of viruses stay in the main memory of the infected computer, infecting any program that is loaded in the main memory for execution. The Black Ice virus, for example, is a memory-resident virus.
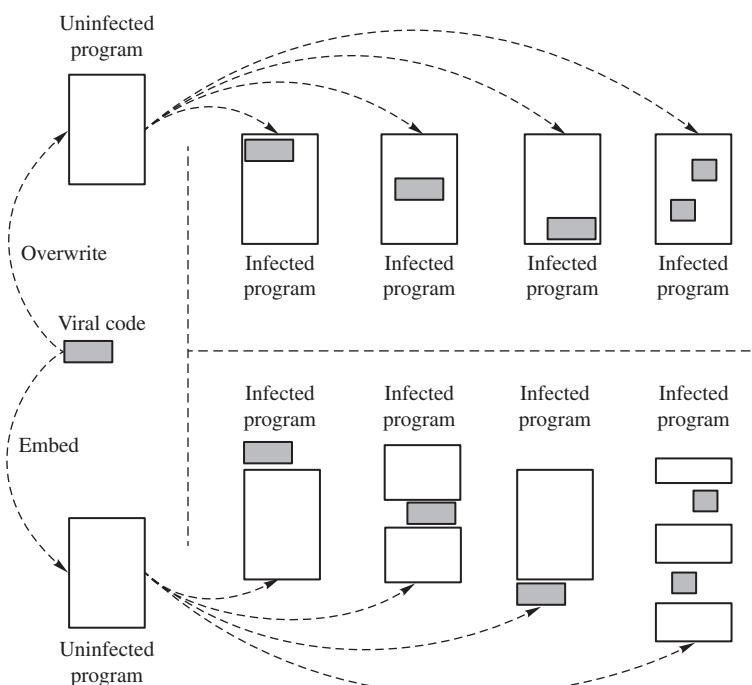
10.1.1.2 Classification Based on Embedded Forms

We may also classify viruses according to the forms they appear in when they infect their host programs. For example, there are *stealth viruses*, *polymorphic viruses*, and *metamorphic viruses*. Stealth viruses try to hide themselves without being detected. Compressing an uninfected program before embedding the viral code is a stealth virus. Polymorphic viruses may change instruction orderings or encrypt viral codes with different keys, so that the same viral code will appear in different forms. Metamorphic viruses can be rewritten automatically during transmission.

10.1.2 Virus Infection Schemes

A viral code may overwrite a segment of an existing program; or insert itself at the beginning, in the middle, or at the end of an uninfected host program. It may also break itself into several segments and insert a different segment in a different location of the uninfected host program. Figure 10.1 shows a schematic of where a viral code may be inserted in a program.

The viral code may place a goto statement at the entry point of the host program (i.e., at the first statement to be executed in the host program) to jump to the viral code and place a goto statement at the end of the viral code to jump back to the first statement of the host program. If a viral code is fragmented, then at the end of each fragment is a goto statement that jumps

Figure 10.1 Schematic of virus infection techniques

to the first statement of the next viral code fragment. Thus, when an infected host program is executed, the viral code will be executed before the original program of the host is executed.

Because viruses have the same access rights as the host files they live in, viruses can do much damage. For example, a virus may modify or delete certain system files or configurations.

Similarly to biological viruses, the life cycle of a computer virus is divided into the periods of latency, infection, and breakout. A virus in the latency period simply stays in the infected host file and does nothing. When the infected host file is executed, the virus may decide, according to the instructions of the viral code and the environment, to replicate itself or a mutation of itself to other healthy hosts (infection), do damage to the system (breakout), or both.

10.1.3 Virus Structures

The structure of a virus typically consists of four subroutines: infect, infection-condition, break-out, and breakout-condition. The infect subroutine searches for possible host programs and checks whether they are infected. If not, the subroutine embeds a viral code in it. The infection-condition subroutine checks whether certain conditions are satisfied for the purpose of launching the infect subroutine. A viral code may use the infection-condition to sit and wait for a particular event to occur. For example, it may wait for a certain notable date or a certain sequence of keystrokes. The break-out subroutine is responsible for carrying out the actual damage work, including modifying and deleting certain files or system configurations. The breakout-condition checks whether certain conditions are met for the purpose of executing the break-out subroutine. The following algorithm is an example showing the virus structure:

```
1.    program V := {
2.         12345;
3.         goto main;
4.         subroutine infect := {
5.             loop:
6.             P := get-random-host-program;
7.             if (the second line of P = 12345;)
8.                 then goto loop
9.                 else insert lines 1-27 in front of P;
10.        }
11.        subroutine break-out := {
12.            modify selected files;
13.            delete selected files;
14.            ...
15.        }
16.        subroutine infection-condition := {
17.            return true if certain conditions are satisfied;
18.        }
19.        subroutine breakout-condition := {
20.            return true if certain conditions are satisfied;
21.        }
22.    main: main-program := {
23.            if infection-condition then infect;
24.            if breakout-condition then break-out;
25.            goto next;
```

26.   }
27.   next:
28.   the original host program ...
29.   }

In this example, the viral code V contains an infect subroutine, which does the following:

1. Search at random for a host program P in the system and check whether the second line of P is equal to "12345"; a label indicating whether P is infected.
2. If not, that is, if P has not been infected, then insert the viral code in front of P to infect P.
3. If yes, search at random for another program P, and repeat the same procedure.

The first statement of V directs the infected host program to execute the viral code. After the viral code is executed, it is directed to execute the original host program.

10.1.4   Compressor Viruses

Although an infected host file still uses the same name as that of the original host file, the length of the infected host file will be longer than the length of the original host file because of the addition of the viral code. This can be detected simply by checking the size of the host file against the size of its original program. To avoid being detected, virus writers may write a viral code that compresses the healthy host before copying itself into it. In particular, when it finds a healthy host file P, the viral code will first compress it to produce P' to provide sufficient space for adding the viral code in it. If after the viral code is added, the length of the infected host file is still shorter than that of the original host file, the viral code may simply add a few dummy instructions or symbols to fix the length. When the infected program is executed, the viral code will need to decompress P' back to P before it executes P (see Fig. 10.2).

The following is an example of a host-compression virus.

1.   **program** CV := {
2.   012345;
3.   **goto** main;
4.   **subroutine** infect := {
5.   loop:
6.   P := get-random-host-program;
7.   **if** (the second line of P = 012345;)
8.   **then goto** loop
9.   **else** = {
10.   compress P to become P'
11.   insert viral code in front of P';
12.   }
13.   **subroutine** break-out := {
14.   modify selected files;
15.   delete selected files;
16.   ...
17.   }
18.   **subroutine** infection-condition := {
19.   **return** true **if** certain conditions are satisfied;

```
20.            }
21.            subroutine breakout-condition := {
22.                return true if certain conditions are satisfied;
23.            }
24.    main: main-program := {
25.                if infection-condition then infect;
26.                if breakout-condition then break-out;
27.                decompress P′ back to P;
28.                execute P;
29.    }
```
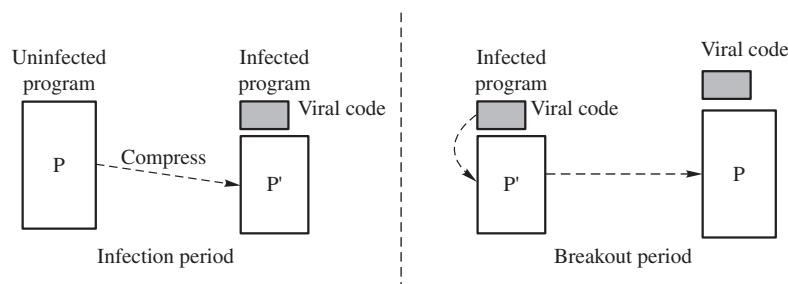
There are other ways to hide the extra size of the viral code to an infected host file. This can be done, for example, by storing code in the NTFS alternate file stream, because native Windows applications do not display its size. The virus may also install a root-kit to intercept system calls to hide the real file sizes.

10.1.5　Virus Disseminations

Once a virus is in a computer system, it may infect other programs or files within the system. A virus may enter a host computer through portable storage devices, such as floppy disks, CDs, and flash memory sticks; through program downloads; or through email attachments.

The use of email attachments to disseminate viruses is common in recent years. This is because most email systems will automatically open an email attachment when it is selected and will automatically execute it if it is an executable file. Thus, if an email attachment is an infected executable file or a document that contains infected macros, executing it allows the viral code to enter the host system. For example, The Zafi virus that appeared around 2004 Christmas was a virus spread through email attachments. Zafi is also called the Christmas virus. It contains an SMTP engine that can search the address books of different users in the infected host computer. It then sends a Christmas greeting message to these addresses with a Zafi virus attachment. Moreover, the Zafi virus can even detect whether the infected host computer has a virus scan installed and will attempt to replace the virus scan software with viral software.

To avoid getting infected from email attachments, users should be cautious not to open email attachments, particulary attachments sent from a stranger.
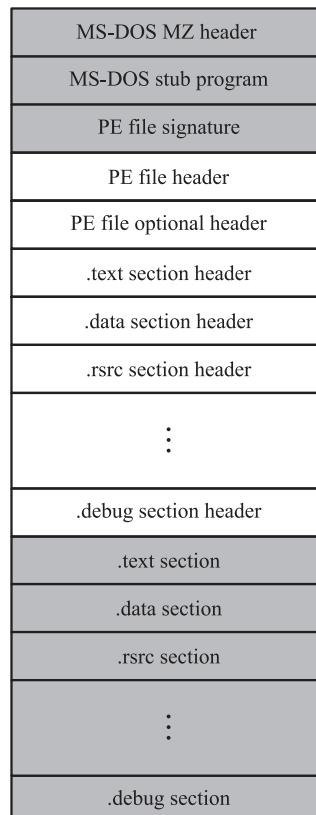
Figure 10.2　Schematic of host-compression virus

10.1.6   Win32 Virus Infection Dissection

This section uses Win32 viruses to dissect virus infection techniques. Most of the Win32 viruses have one thing in common. That is, they exploit Microsoft's Portable Executable (PE) format to infect other programs. Indeed, thousands upon thousands of Win32 viruses have been reported that use PE to do their tricks.

10.1.6.1   The PE Format

The PE format originated from the Common Object File Format (COFF) in UNIX. It modifies COFF to present executables, object code, and DLLs in Microsoft Windows operating systems. It allows the Windows operating system to map an executable image efficiently and reliably from disk to the memory space. In particular, a PE file consists of headers, sections, and other information (see Fig. 10.3). Note that on top of the PE format is the MS-DOS MZ header, where MZ, represented by hexadecimal 4D 5A, stands for Mark Zbikowski, one of the early developers of DOS. The PE sections contain the modules of code, data, resources, import

| MS-DOS MZ header |
| MS-DOS stub program |
| PE file signature |
| PE file header |
| PE file optional header |
| .text section header |
| .data section header |
| .rsrc section header |
| ⋮ |
| .debug section header |
| .text section |
| .data section |
| .rsrc section |
| ⋮ |
| .debug section |

Figure 10.3   Schematic of the PE file format

tables, and export tables, where the .text section holds program code, the .data section holds the global variables, and the .rsrc section holds the resources. The PE headers provide crucial information of the executable image.

To save storage space, different sections in one PE file are stored on disk contiguously. When loaded in the memory, however, each section needs to be aligned to a page boundary, so that different sections may be given different levels of memory protections. For example, the .text section should be mapped to executable and read-only memory, while the .data section should be mapped to nonexecutable, readable, and writable memory. The dynamic linker is responsible for mapping each section and setting an appropriate access permission to each page according to the information contained in the corresponding header. Thus, PE headers are natural targets of exploitations by Win32 viruses.

In a PE file, there is a special area to hold the Import Address Table (IAT), which will store at run time the actual locations of the Windows API functions to be used by the PE file so that the code can jump to these locations to run them. IAT is needed because the program at compile time does not know in advance where the DLLs it will need are located in memory. These DLLs will be loaded when the PE file is loaded. Thus, IAT is also a natural target of exploitation by Win32 viruses.

10.1.6.2   PE Headers and Virus Exploitations

A PE header is a structure consisting of a number of fields. The following fields, in particular, are exploited by most Win32 viruses:

WORD AddressOfEntryPoint
This field stores the address of the entry point of the PE file, which typically points to the .text section. That is, this is the address where the execution begins. Most Win32 viruses modify this field to point to the viral code.

DWORD ImageBase
This double-word field stores the address of the PE image. Most Win32 viruses use it to calculate the actual address of certain objects.

DWORD SectionAlignment
This double-word field stores the value for section alignment, where each section in the PE file is to be mapped into memory starting at a virtual address divisible by this value. Most Win32 viruses use it to calculate the actual address of the viral code.

10.1.7   Virus Creation Toolkits

Viruses are typically written in an assembly language. Thus, writing viruses requires special training. To make it possible for beginners to create viruses without knowing how to write assembly code, elite virus writers have produced virus creation toolkits for amateurs. The Next Generation Virus Creation Kit (NGVCK), for example, was a popular virus creation toolkit. NGVCK is a Visual Basic application for generating 32-bit viruses infecting Windows PE files.

NGVCK generates metamorphic viruses. It generates a piece of viral code that does the same thing with each different virus, but with an almost completely different structure each time.

10.2   Worms

A worm is typically a standalone application program that can replicate itself to a different computer on a network. This is similar to a worm that creeps its way from one place to another. A worm executes itself automatically on a remote computer with or without extra help from a user. Some worms, however, may need a host file for spreading. Thus, worms may be viewed as a special kind of viruses. In other words, a virus that replicates itself primarily through networks is a worm.

Worm Structures

A worm typically consists of a *target locator* subroutine and an *infection propagator* subroutine. The target locator subroutine is used to find new targets, and the infection propagator subroutine is used to transfer itself to a new computer.

10.2.1   Common Worm Types

Mass mailers and *rabbits* are the two most common types of computer worms.

   Mass mailers are worms that reproduce themselves to other computers through emails. It is customary to attach "@mm" at the end of the name of the worm to indicate that it is a mass-mailer worm. For example, the VBScript LoveLetter worm is sometimes denoted by VBS/LoveLetter.A@mm.
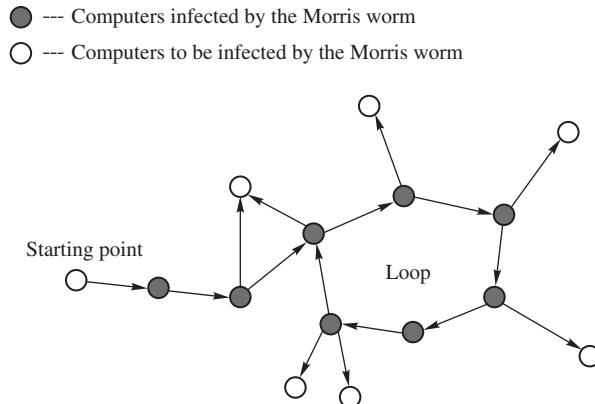
   Rabbits are worms that can massively replicate themselves to take over the entire memory, causing a system to crash. This behavior resembles, in a way, biological rabbits for their high breeding rates. Rabbits are often hidden in a file directory, or use normal file names to disguise themselves.

10.2.2   The Morris Worm

The Morris worm, created in 1988 by Robert Morris, a computer science graduate student at Cornell University, was one of the earliest worms. It exploited implementation flaws of UNIX utilities `sendmail`, `finger`, and `rsh/rexec` to replicate itself to other machines. The UNIX operating system at that time allowed users to create system files (e.g., `$ HOME/.rhosts`) to log on to another networked computer without typing user passwords. Such files make it easy for the Morris worm to spread across the Internet. Even if the infected user directory does not contain such documents, the Morris worm can still obtain other users' email addresses through the mail box contained in the infected user directory. The Morris worm then uses the dictionary attack to obtain user passwords. Early UNIX operating systems did not have any security guideline to help users select passwords, which helped make the dictionary attack successful.

   The objective of the Morris worm is to infect other networked computer as quickly as possible without leaving any trace. Thus, the Morris worm was not intended to harm infected computers. However, because it spread fast, the Morris worm had produced, unintentionally, the effect of denial of service. Figure 10.4 depicts the spread of the Morris worm.

   For his crime, Robert Morris was sentenced to 3-year probation, plus a fine of $10,050 and 400 hours of community service.

Figure 10.4 Spread schematics of the Morris worm

10.2.2.1 Exploiting Buffer Overflow in the finger Utility

To use the finger utility to propagate itself to a new host after obtaining a new email address from the host it resides in, the Morris worm exploited a buffer overflow loophole in the implementation of the finger server program (a.k.a. daemon). The loophole is that the finger daemon used a library function gets() on a 512-byte buffer, but it did not check bounds. The Morris worm sent a crafted 536-byte string to the new host to overrun the 512-byte buffer. In particular, the 536-byte string contained 28-byte of VAX assembly code, a.k.a. *shellcode*, and used buffer overrun to get the new host to run the shellcode contained within it.

10.2.3 The Melissa Worm

The Melissa worm was a macro virus created in 1999 by David L. Smith. It was arguably the first widely publicized worm targeted at Microsoft products. The Melissa worm replicated itself through emails. When the user opens an email attachment that contains the Melissa virus and if the user has Microsoft Outlook installed, the viral code will search 50 email addresses stored within Outlook and send an email to each of these addresses with a virus attachment. The email message looks like the following:

```
From: <the infected sender>
Subject: Important message from <the infected sender>
To: <The 50 chosen recipients>
Attachment: LIST.DOC
Body:
Here is that document you asked for ...
don't show anyone else ;-)
```

The attachment LIST.DOC is the host of the macro virus. This scheme repeated recursively, and so it spread extremely fast and created a huge amount of email traffic, which quickly jammed the networks. For example, if the Melissa worm spreads itself to exactly 50 recipients

(some recipients may have already received the same message), then after it spreads for n times, 50^n messages will be sent and a large number of users will be infected.

David L. Smith confessed to his crime, and he was sentenced to a 20-month jail time and a fine of $5000 in 2002.

10.2.4 The Code Red Worm

The Code Red worm, released in July 2001, infected about 300,000 computers within the first 24 hours of its release. Similarly to the Morris worm that exploited buffer overflow loopholes in UNIX utilities, the Code Red worm exploited a buffer overflow loophole in Microsoft's Internet Information Services (IIS). IIS is run on Windows 2000 or Windows NT servers. The Code Red worm arrived at the Web server computer as a `GET /default.ida` request as follows (with 224 N's):

```
GET /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801
%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3
%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=aHTTP/1.0 ...
```

where the ellipsis `...` represents headers and a request body that contains the main worm code, `%uXXXX` represents one unicode character for a total of 44-byte unicode encoding, and `%00=a` is an invalid unicode encoding.

This request started the worm code execution, and the request would exist only in the main memory of the Web server. If the system time was before the 20th of the month, the worm started to infect new systems. If the time was between the 20th and the 27th of the month, the worm started a DoS attack on the Website of the White House: `www.whitehouse.gov`. The worm propagated to other IIS Web server computers as follows: it generated at random an IP address and checked whether port 80 was open on this address.

When processing the 224 N's in the `GET` request, IIS overwrites the buffer allocated to `GET`, which caused a certain C runtime library function to throw an exception. The exception handling would cause the exception frame be called, which was

```
%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3
```

representing four addresses (in hexadecimal): `68589090`, `7801cbd3`, `90909090`, and `00c38190`. In particular, the address `7801cbd3` was in the memory image for a certain C runtime library function, which would eventually transfer the control to the main worm code.

10.2.5 The Conficker Worm

The Conficker worm was the work of a clever combination of several advanced malware techniques targeting at the Windows operating system. It uses dictionary attacks on administrator passwords and software flaws to propagate. The Conficker intends to create a botnet

by infecting millions of computers. However, the intension of the author(s) of the Conficker remains a mystery. The Conficker worm has infected millions of computers worldwide.

The Conficker comes with five variants called Conficker A, Conficker B, Conficker B++, Conficker C, and Conficker E. Conficker A was detected in November 2008 infecting computers that did not install a Microsoft patch released only weeks earlier. One month later, Conficker B added new distribution mechanisms including USB memory stick. Conficker A and Conficker B could infect no more than 250 pseudo-random domains a day, and Conficker C would generate 50,000 pseudo-random domains per day from over 116 domains all over the world.

Removing a Conficker worm from an infected computer is nontrivial, and the Conficker is still affecting people at the time of writing this section in July 2014. To counter the Conficker, a group of experts formed a working group, called the Conficker Working Group (CWG), with a mission to figure out how to remediate infected computers and eliminate the threat of the botnet. The following are CWG recommendations published in early 2011:

1. Focus on the larger overall threat environment and develop a strategy for dealing with that global issue, versus the "whack-a-mole" approach of battling one incident after another.
2. Establish the mindset of a "long-term battle" at the outset to help manage burn-out and fatigue.
3. Work to expand the size, skills, technological advantage and communications networks of cybersecurity defenders to match the growing threat.
4. Identify resources (monetary and otherwise) used for cybersecurity efforts and work toward an allocation model that is effective at the strategic level.

10.2.6   *Other Worms Targeted at Microsoft Products*

Shortly after the Code Red worm was released, the Code Red II worm and the W32.Nimda worm were released. The SQL slammer worm, the W32.Sobig.F@mm worm, the W32.Welchia worm, the W32.Mydoom@mm worm, the Storm worm, and the P2P.Palevo.DP worm were released subsequently.

Similarly to the Code Red worm, the Code Red II and the W32.Nimda worms were released in 2001, which were both targeted at IIS, exploiting the same buffer-overflow loophole in IIS. The Code Red II worm placed a backdoor in the Code Red worm, allowing the attackers to directly control the infected computer. The W32.Nimda worm is similar to the Code Red II worm, but it can spread itself through different channels, including emails, Web browsing, and backdoors. It not only can modify Web documents, including .htm, .html, .asp documents, and certain executable files, but also can replicate itself into several files with different file names in the infected machine.

The SQL slammer worm, found in 2003, uses a buffer overflow loophole in Microsoft's SQL server to infect SQL server machines. It is a small program, but it spread rapidly. The W32.Sobig.F@mm worm, found in 2003, uses open proxy servers to send a massive number of junk emails with forged sender information.

The W32.Welchia worm, found in 2003, exploits multiple vulnerabilities in Microsoft products to download a particular patch from Microsoft's Windows Update Website, install it, and then restart the computer.

The W32.Mydoom@mm worm, found in 2004, is a worm with a backdoor. It makes infected machines send out a huge number of junk mails. For example, within 36 hours, it had about 100 million messages sent.

The storm worm, found in 2007, infects PCs and makes them attack other PCs. For example, a YouTube storm worm found in August 2007 invited users to see themselves in a video, but the included link would direct visitors to a Website that downloads malicious code to their PCs.

The P2P.Palevo.DP worm spreads via an IM spam message that tricks the user into saving an executable file that looks like a .jpg file. If the user opens the file, the malicious code is executed, which creates hidden files in the Windows folder and modifies certain registry key to point to these hidden files to bypass the Windows firewall.

10.2.7   Email Attachments

Most email systems allow users to include attachments of various formats. Moreover, email systems can open an attachment using an appropriate application program or execute it if it is an executable file. This feature provides convenience to users, but it also provides a platform for spreading worms.

Certain types of email attachments are safe to open, for they do not contain executable codes or macros. We refer to such attachments as *safe attachments*. Other types of email attachments contain executable codes or macros. The receivers should be cautious whether or not to open them. Some types of attachments may be opened if they are from trustworthy senders. We refer to such attachments as *to-be-cautious attachments*. Other types of attachments are perilous and should not be opened at all. We refer to such attachments as *perilous attachments*.

The extension of the file name in an email attachment can be used to help determine which category the attachment belongs to.

10.2.7.1   Safe Attachments

Listed in Table 10.1 are extension names of common safe attachments.

10.2.7.2   To-Be-Cautious Attachments

Listed in Table 10.2 are extension names of common to-be-cautious attachments.

10.2.7.3   Perilous Attachments

Listed in Table 10.3 are extension names of common perilous attachments.

10.2.7.4   Remarks

The classification of email attachments presented in this case may change in the future because of new development of technologies. Certain types of attachments that are considered safe today may no longer be safe tomorrow when new exploitations are discovered for embedding viruses. Likewise, certain types of to-be-cautious attachments or perilous attachments may be upgraded to a safer category because known loopholes for virus infections have been fixed. In the meantime, new types of file extensions may be generated, and so additional members may

Table 10.1 Extension names of common safe attachments

Extension	File type	Comment
.ai	Graphics	Adobe Illustrator's graphics file
.art	Graphics	America Online's graphics file
.avi	Video	Microsoft's Audio-Video Interleaved file
.bmp	Video	Microsoft's Bitmap file
.cgm	Drawing	2D Computer Graphics Metafile file
.dxf	Same as above	AutoCAD Drawing Exchange Format file
.dwg	Same as above	AutoCAD data file
.eps	Graphics	Encapsulated PostScript file
.gif	Same as above	CompuServe's Graphics Interchange Format file
.jpe	Same as above	Joint Photographic Experts Group file
.jpg	Same as above	Same as above
.jpeg	Same as above	Same as above
.mid	Audio	Musical Instrument Digital Interface file
.midi	Audio	Same as above
.mov	Video	Apple's Quicktime Movie file
.mp2	Audio	MP2 file
.mp3	Audio	MP3 file
.mpg	Video	Moving Picture Experts Group file
.mpeg	Video	Same as above
.pcx	Graphics	Microsoft Windows' Paintbrush file
.pdf	Document	Adobe's Portable Document Format file
.rle	Graphics	Run Length Encoded file
.rm	Audio/video	Real Media file
.ram	Audio/video	Same as above
.rtf	Document	Microsoft's Rich Test Format file
.sdr	Drawing	SmartDraw file
.tif	Graphics	Image File Format file
.tiff	Same as above	Same as above
.ttf	Font file	TrueType font file
.txt	Text	Microsoft's text file
.wav	Audio	IBM and Microsoft's audio format file
.wma	Audio	Microsoft Windows Media Audio file
.wri	Text	Microsoft Windows Write file

be added. On the other hand, certain types of attachments may also be removed from these tables when they become obsolete.

10.3 Trojans

A Trojan is a program that appears to do something, but it also contains a piece of code that does something else. This piece of code, called a *warrior* code, is similar to a viral code, and so some people would refer to a Trojan horse as a Trojan virus. However, unlike a viral code that can replicate itself automatically to other programs or systems, the warrior code will remain in the same program that is also written by the attacker. In general, a Trojan horse will be

Table 10.2   Extension names of common to-be-cautious attachments

Extension	File type	Comment
.asp	Web document	May contain malicious codes or cookies
.doc	Word document	May contain macro virus
.dot	Word document template	May contain macro virus
.eml	email	Attachment itself is an email message, be cautious about the attachments of it
.htm	Web document	May contain malicious codes or cookies
.html	Same as above	Same as above
.lnk	File pointers	Linked files may contain malicious codes
.rar	Compressed file	Okay to uncompress, but be careful of the contents in the uncompressed files
.sea	Same as above	Same as above
.sit	Same as above	Same as above
.tex	T_EX document	May contain macros
.url	Web links	May contain malicious Web pages
.uue	Compressed file	Okay to uncompress, but be careful of the contents in the uncompressed files
.wk1	Lotus document	May contain macro viruses
.wk3	Same as above	Same as above
.wk4	Same as above	Same as above
.wks	Same as above	Same as above
.xls	Spreadsheet	Same as above
.zip	Compressed file	Okay to uncompress, but be careful of the contents in the uncompressed files

Table 10.3   Extension names of common perilous attachments

extension	file type	comment
.pif	Executable	Program Information File; contain information for Windows to run non-Windows applications; may contain SirCam and other viruses
.exe	Same as above	Microsoft applications
.com	Same as above	MS-DOS COM files
.vbs	Same as above	Microsoft Visual Basic Script
.vb	Same as above	Microsoft Visual Basic applications
.bat	Text	MS-DOS batch file of comments and programs
.bin	Executable	Macintosh applications
.reg	Text	Windows registry that can change setups
.js	Script	JavaScript
.jse	Script	Same as above
.scr	Screen saver	May be a disguised program
.xlm	Executable	Microsoft Excel Macros
.wmz	Media skin	Windows Media Compressed Skin File; may be used to spread viruses
.hta	HTML application	Executable contained in a Web page
.ocx	ActiveX control	Used to execute other programs
.wsf	Script	Windows Script Files
.wmf	Graphics	May be used to spread viruses

passively waiting for someone to bring it into his machine. The attacker, however, may try to lure users to do so. Trojan horse is considered the simplest kind of malicious software.

There are special kinds of Trojan horses that drop other Trojan horses into compromised computers. These Trojan horses are referred to as *Trojan droppers*.

Trojan horses may inflict the following damages to the compromised computers:

1. Install backdoors and Zombieware to prepare for a DDoS attack. Trojan horses can also be used to install email programs to send out junk emails.
2. Install spyware.
3. Look for users' bank account numbers and private information.
4. Install viruses or other forms of malicious codes to other machines.
5. Modify or delete user files.

10.3.1 Ransomware

Ransomware is a kind of malware that inflicts serious damages on the infected systems in such a way that the damages are very hard to recover, even by experts, although the ransomware itself may be easy to remove from the systems. The attackers hold the secrets to recover the damages and would offer to fix the infected systems for a fee. Ransomware typically gets in a system as a Trojan or as an email attachment. CryptoLocker, for example, is a ransomware targeted at computers running the Windows operating system. It uses RSA PKC to encrypt certain types of files on the computers and the mounted drives. The attackers hold the private keys that are strong enough to make the encryptions difficult to break, although it is easy to remove CryptoLocker from the infected systems. Once activated, the CryptoLocker malware posts an offer on the infected computer to decrypt the data for a fee, with Bitcoin payments or prepaid vouchers.

10.4 Malware Defense

For convenience, we use malware to denote viruses, worms, and Trojans. Prevention and restoration are common approaches to defending and countering malware attacks.

Prevention
Prevention mechanisms block malware from getting into a healthy system. Prevention may be achieved using the following measures:

1. Install security patches in time.
2. Do not download software from untrusted Websites. In general, you should only download software from reputable vendors and their distributors, or other sources you trust. When in doubt, search the Internet and determine whether a particular vendor or a Website could be trusted.
3. Do not open to-be-cautious email attachments from unknown senders.
4. Do not open perilous email attachments.
5. Avoid using BitTorrent and other peer-to-peer applications to share files (see Section 10.6 for more information on peer-to-peer security).

Restoration

Restoration mechanisms disinfect infected systems. Restoration may be achieved using the following measures:

1. Scan files using a malware scanner; quarantine or remove infected files when they are found. A malware scanner is a program that can detect known malware programs, quarantine them, and remove them.
2. Keep a backup of the system files and user files, which can be used to restore the system when it is attacked by malware.

10.4.1   Standard Scanning Methods

Malware scanners scan each file to look for malware. Files to be scanned are executable files, office documents, email attachments, instance messages, downloaded programs, and other files that are possible to become hosts to malware. For convenience, we refer to these files as *hostable files*. Basic scanning, heuristic scanning, ICV scanning, and behavior monitoring are standard scanning methods. A malware scanner may implement some or all of these scanning methods, which means that the computing resources used by various scanners may vary considerably depending on the method or combination of methods used.

Basic Scanning

Basic scanning looks for signatures of known malware in hostable files, including structures, formats, patterns, and other characteristics. Basic scanning may also check whether the size of system files has been altered to detect infections.

Heuristic Scanning

Heuristic scanning looks for suspicious code fragments in executable files on the basis of certain heuristics. For example, heuristic scanning may use certain heuristics to search for encryption keys embedded in infected files, which are used by polymorphic viruses.

ICV Scanning

ICV scanning computes the integrity check value of each uninfected executable file using a fixed HMAC algorithm (or other message code authentication algorithm) and a fixed encryption key. A ICV value is appended to the end of uninfected executable files. This ICV value is a mark that indicates that the file is uninfected because no viral code would know the encryption key, and so it cannot change the ICV. Thus, if such a file is infected with a virus, its ICV value will be different from the original ICV, which can be used to detect viruses.

Behavior Monitoring

Behavior monitoring looks at and evaluates the behavior of executing programs. If a program in execution is actively searching for other executable programs it is not supposed to, then this program is likely to have been infected.

10.4.2   Anti-Malicious-Software Products

This section introduces several widely used AMS software products. These products are constantly updated to include new found malware. Users are strongly recommended to apply

multiple AMS products on the same computer, as a given AMS product may not be able to capture a particular malware program.

McAfee VirusScan

McAfee is an antivirus software product widely used by users in large organizations and by home users. It uses basic scanning to detect known viruses and uses heuristic scanning to detect new viruses. McAfee VirusScan can be obtained from `www.mcafee.com`.

Norton AntiVirus

The basic functionalities of Norton AntiVirus are similar to those of McAfee VirusScan. In addition, Norton AntiVirus can also detect and remove spyware, as well as perform preinstallation virus checks. Norton AntiVirus can be obtained from `www.symantec.com`.

Avast! AntiVirus

Avast! AntiVirus is free of charge to home users. Other than lacking certain special features, Avast! AntiVirus is a good antivirus product. It can be obtained from `www.avast.com`.

Webroot SecureAnywhere

Webroot SecureAnywhere is a light-weight, easy-to-use product. It can be obtained from `www.webroot.com`.

Malwarebytes Anti-Malware

Malwarebytes Anti-Malware is a powerful AMS product, which offers free and paid versions for downloads from `www.malwarebytes.com`.

Other Antivirus Products

Other antivirus software products include PC-cillin, Panda, eTrust EZ Antivirus, AVG Anti-Virus, and Clam. They can be obtained from the following Websites:
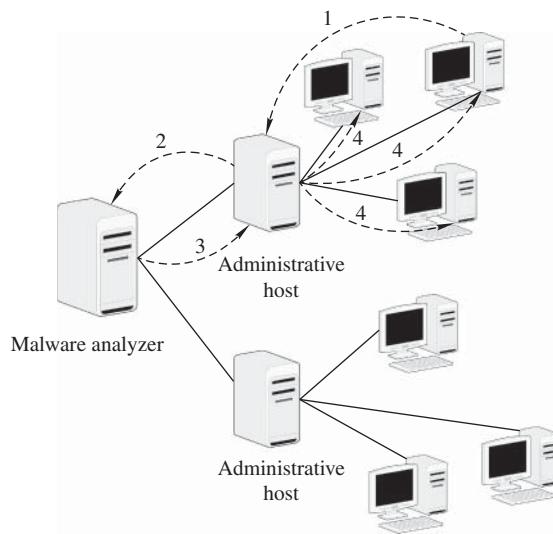
1. PC-cillin: `www.trendmicro.com`
2. Panda: `www.pandasoftware.com`
3. EZ Antivirus: `www3.ca.com`
4. AVG: `http://free.grisoft.com/`
5. ClamAV: `http://www.clamav.net/`

AVG Anti-Virus and ClamAV are free antivirus tools, where ClamAV has a Windows version and a Linux version.

10.4.3 Malware Emulator

Malware emulator provides an isolated hardware and software emulation environment to actually run suspicious programs. Doing so helps to identify malware without spreading it.

Users can setup an emulation environment in each host computer or in each LAN, so that users can run suspicious programs under tight controls. Doing so, however, may also incur high computation overhead. IBM proposed in 1997 the concept of *digital immune system* (DIS) to balance between detection effectiveness and efficiency. The basic idea of DIS is to

Figure 10.5 Schematic of digital immune system

set up a special computer, called the *malware analyzer*, in the internal network to provide a well-protected and isolated emulation environment. Each host computer inside each subnetwork in the internal network will use standard malware scanner to detect and remove malware. If the host computer finds that a certain program is suspicious, yet it passes malware scan, then the host computer will forward this suspected program to the malware analyzer through its administrative host. The malware analyzer runs the suspected program, determines whether it is infected, and emails the result back to the administrative host. The administrative host then sends the result to every host computer in the subnetwork. In other words, DIS execution has four phases (see Fig. 10.5).

1. An internal host in a subnetwork discovers a suspicious program and forwards it to the administrative host.
2. The administrative host encrypts the suspected program so that the program cannot execute during transmission. It then forwards the encrypted suspect to the malware analyzer.
3. The malware analyzer decrypts the encrypted suspect, sets up an enumerator to run the suspected program, and checks whether it contains malware code. For example, it checks whether the suspected program runs normally without suspicious behaviors. The malware analyzer then sends the result back to the administrative host.
4. The administrative host then forwards this result to each host computer in the subnetwork.

10.5 Hoaxes

Hoaxes are shams with the sole purpose to trick users to do something they would normally not do. Computer hoaxes often appear in the form of email messages, urging the recipients to do something and hoping that the recipients will do as asked out of their kindness, curiosity,

or greediness. For example, a hoax message may present a desperate plea to help a sick child, to help stop a virus, to help send a chain mail, to invite you for a free vacation in a fancy place, or to help a wealthy man's widow (or daughter) in an African tribe to get the money she is entitled to from an unheard bank with a promise of giving you a large monetary lump sum in return for your help. Indeed, almost any topic may be used in a hoax message.

Most of the hoaxes have been designed in order to take your money. You may be asked to deposit a check under the guise of securing a transaction for a large lump sum from which you will get a percentage; you may be asked to make a donation to a charity; you may receive a document with an official look stating that a huge amount of money will be given to you, provided that you send in a small check for processing the necessary paperwork.

There also are virus hoaxes. For example, the infamous "You've Got Virus!" hoax was once spread widely, claiming that your system might have been infected with the "WORST VIRUS EVER," telling you that this virus had just been discovered, and urging you to "FORWARD THIS TO EVERYONE YOU KNOW!!!". It also asked you to remove the "virus" in your system at a certain location. The "virus" in this hoax was a legitimate Java program `jdbgmgr.exe` or `sulfnbk.exe`. Many users were tricked and actually removed these "viruses." Note that this hoax was sometimes referred to as the "Teddy Bear virus hoax" for the icon of `jdbgmgr.exe` was a teddy bear.

The countermeasure of hoaxes is to ignore them. That is, do not do what the hoax message pleads you to do. There is no free lunch. If you think something is too good to be true, then most likely it is.

If you are not sure whether a virus is a hoax, you should ask your system administrator or check up on it at relevant Websites, such as McAfee at `http://vil.mcafee.com/hoax.asp`, Vmyths.com at `http://vmyths.com/`, and Sophos.com at `http://www.sophos.com/search/`.
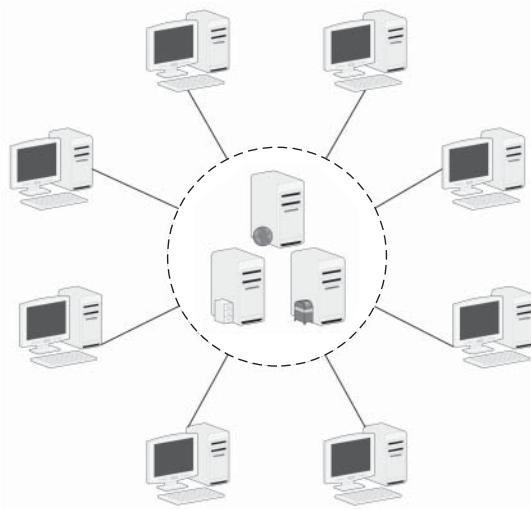
10.6   Peer-to-Peer Security

While most network applications are client-server applications, some are peer-to-peer (P2P) applications. P2P protocols include BitTorrent, eMule, Napster, Skype, and Gnutella, which have been widely used for users to distribute and share music, games, videos, and other types of files. The client-server model (see Fig. 10.6) has a star topology where a small number of servers provide services to a large number clients.
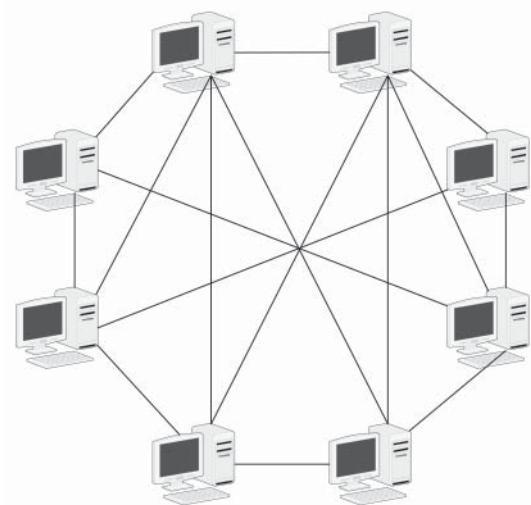
P2P networks are ad hoc networks, where each computer acts both as a client and as a server. When a user downloads a file to his computer using a P2P application, he becomes a client of several other computers that have parts (maybe different parts) of the file the user needs and feed these parts to the user. In the meantime, the user's computer also becomes a server to other users who request the same file and feeds the parts it has downloaded to these users. Figure 10.7 shows a topology of the P2P model.

10.6.1   P2P Security Vulnerabilities

P2P applications are often used to download and share music (e.g., MP3) and video (e.g., AVI) files. This gives rise to two security concerns. The first concern is copyright infringement, for users may download copyright-protected materials without paying the copyright owners. The

Figure 10.6   Schematic of a client-server topology

Figure 10.7   Schematic of a P2P topology

second concern is that P2P applications may consume too much network bandwidth and local disk storage, causing denial of service unintentionally or intentionally.

The biggest security concern, however, is that when you use a P2P application, it will open a specific port on your computer to share files with any user unknown to you without any form of authentication, making your computer vulnerable to Trojan horses, viruses, and other forms of malicious software.

10.6.2 P2P Security Measures

Firstly, users must install official versions of the P2P software. That is, do not download P2P software from an untrusted site to avoid downloading malicious software.

Secondly, keep in mind that the file downloaded from unknown computers using P2P applications may not be what it says it is. Thus, the user should scan the downloaded file using a good antivirus product before opening it. This could prevent Trojan horses, viruses, worms, and spyware from executing before they are discovered and removed.

Thirdly, disallow users to use P2P software in company computers without special permission. This will help companies avoid copyright violation lawsuits, prevent denial of service because of consumption of network resources, and other security attacks.

10.6.3 Instant Messaging

Instant messaging (IM) is a widely used communication tool over the Internet. It allows users to send instant messages to each other or talk to each other via voice of IP (VoIP) in real time. The precursor of IM was a simple application program allowing several users logged on to the same UNIX machine from different terminals to talk to each other simultaneously. It was later extended to local area networks and the Internet. Some IMs are client-server applications that use server computers to relay messages, and some are P2P applications.

Common IMs include Yahoo!Messenger, Skype, Google Talk, QQ, and WeChat. The use of IM is a convenient way to stay in touch with friends and business associates, but users should be aware of the following risks:
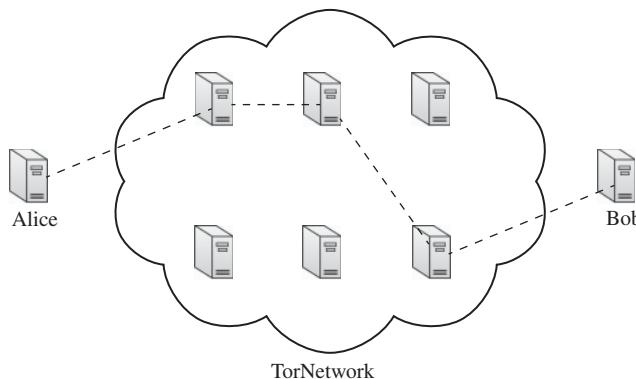
1. Instant messages are often transmitted in plaintext, and so they are subject to eavesdropping.
2. IM system may not check viruses or Trojan horses. Thus, IM may be used by attackers to transmit viruses and Trojan horses.
3. If end systems are not configured properly, attackers may be able to compromise these machines through IM.

Users may want to set up ACL rules to control packets passing through the IM ports.

10.6.4 Anonymous Networks

Suppose that Alice wishes to browse a Website stored on Bob's Web server. She could just use a standard Web browser to look at this Web page. However, Alice wants to do this anonymously. In other words, Alice does not want Bob to know that she is looking at a Webpage on his Web server. Alice can achieve this task by using the Tor protocol.

Tor is an overlay network that consists of a collection of volunteer proxy servers, called *onion routers* running a special piece of software that allows for the routing of a TCP message to a destination. Moreover the onion routers move traffic from Alice to Bob in an anonymous manner. Alice participates in this process by selecting onion routers, three in practice, to form a *circuit* (see Fig. 10.8). This circuit is used to anonymously route traffic to Bob. In addition, Alice must refresh this circuit after Alice transmits a certain number of messages. The

Figure 10.8   The selection of a circuit in the Tor network

communication across the circuit is encrypted via a set of symmetric keys shared between Alice and a given onion router in the circuit.

Alice sends a message to Bob through the circuit by passing the message encrypted with all of the keys shared with the onion routers across the circuit. Each individual onion router in the circuit will decrypt the message it receives with the key it shares with Alice.

For Alice (A) to send a message m to Bob (B) using onion routers O_1, O_2, and O_3, and shared keys K_{AO_1}, K_{AO_2}, and K_{AO_3}, respectively, Alice and the routers cooperate as follows:

1. $A \rightarrow O_1$: $E_{K_{AO_1}}(E_{K_{AO_2}}(E_{K_{AO_3}}(m)))$.
2. $O_1 \rightarrow O_2$: $E_{K_{AO_2}}(E_{K_{AO_3}}(m))$.
3. $O_2 \rightarrow O_3$: $E_{K_{AO_3}}(m)$
4. $O_3 \rightarrow B$: m.

Onion routers O_1 and O_3 are given special names in the circuit called the *entrance router* and the *exit router*, respectively.

A reply from B to A in Tor is symmetric to the communication between A and B. The only difference is that each onion router O_i will perform an encryption of the response at each step.

In practice, Tor is implemented as a proxy for TCP connections. This means that Tor is not a perfect protocol. In particular, if it is used in conjunction with an out-of-band protocol (a protocol that can bypass the proxy), the identity of a user can be leaked. For example, any UDP transmissions will bypass Tor and leak the source of the request.

10.7   Web Security

The World Wide Web, also referred to as the Web (or the small case web by some authors), is a client-server application program based on TCP connections. It provides software tools for users to obtain and share information. These tools, however, may contain design flaws or implementation loopholes that can be used by attackers to harm users. This section

introduces basic types of Web documents, software tools, possible exploitations, and countermeasures.

10.7.1 Basic Types of Web Documents

Web documents, also called Web pages, are written using the Hypertext Markup Language (HTML). They are ASCII text files, where the text is enclosed in HTML tags, describing how the documents should be displayed. Web documents may or may not contain executable codes. Web documents can be classified into three categories. They are *static documents*, *dynamic documents*, and *active documents*. A static document is a Web document that does not contain executable codes. A dynamic document and an active document both contain executable codes. The difference is that the executable codes contained in dynamic documents are executed on the server computer, while the executable codes contained in active documents are executed on the client computers.

Static Documents
When the client requests a static document from a Web server, the client browser will download the document to the client computer and display it locally. Static documents are safe to download.

Dynamic Documents
The executable codes contained in a dynamic document are typically written as Common Gateway Interface (CGI) scripts. CGI defines a series of system variables used to obtain values from the the server computers. CGI programs may also be written in other languages such as C and Visual Basic (VB), where the CGI variables and command lines will be executed.

When the client asks for a dynamic document from a Web server, the server computer runs the executable codes contained in the document, substitutes the CGI variables contained in the document with new values, and downloads the resulting document to the client computer for display.

Dynamic documents also include Java Server Pages (JSP), Active Server Pages (ASP), and Hypertext Preprocessor (PHP). JSP is a Java technology that enables a Web server to generate dynamically, upon request from a Web client, HTML, XML, or other types of documents by embedding certain predefined actions and Java code in a Web page. ASP is Microsoft's server-side scripting tool. ASP pages may be written in VBScript or other scripting languages. PHP is a general-purpose scripting language.

Active Documents
The executable codes contained in an active documents are typically written as JavaScripts or Java applets. When the client asks for an active document from a Web server, the server computer will download the entire document to the client machine. The client machine then runs the executable codes contained in the document.

Java applets are Java programs represented in the platform-independent Java byte code format for the Java Virtual Machine (JVM), which will be translated into machine code of the local machine and run at the client computer. A JavaScript file contains a sequence of instructions

written in the JavaScript language. JavaScript is embedded in an HTML document in between the `<script>` and `</script>` tags. The following is a simple JavaScript document:

```
<html>
<head>
<title>
    Sample JavaScript HTML file
</title>
<script language="JavaScript">
    document.writeln("Sample JavaScript HTML file");
</script>
</head>
<body>
.
.
.
</body>
</html>
```

10.7.2   Security of Web Documents

In general, static documents are safe, for they do not contain executable codes. Dynamic documents and active documents contain executable codes, and so they may be subject to security attacks. Because dynamic documents are executed on the server computer and active documents are executed on the client computer, the server computer may be attacked through loopholes contained in dynamic documents and in the Web server program, and the client computer may be attacked through loopholes contained in active documents and in the Web browser program.

The following are common security measures to protect server-side computers:

1. Update the Web server program to the newest version.
2. Manage rigorously the CGI programs and the directory that stores CGI programs.
3. Allow only designated persons to post CGI programs at the Web server.

Downloaded JavaScript and Java applets run on the client computer are restricted in well-protected memory space. However, attackers may be able to exploit loopholes in browser software to create malicious JavaScript or Java applets to harm users. For example, older browsers could be exploited to allow malicious JavaScript to change the user's default Web page to a different Web page designated by the attacker, read system files, or read the email addresses stored in the user's mail box.

The following are common security measures to protect client-side computers:

1. Install browser patches in time.
2. Disable JavaScript of the browser so that JavaScript cannot be run on the client computer.
3. Disable Java applets of the browser so that Java applets cannot be run on the client computer.

10.7.3 ActiveX

ActiveX is a set of technologies allowing software components to interact with each other, where different components may be written in different languages. In particular, ActiveX combines the Object Linking and Embedding (OLE) technology and the Component Object Model (COM) technology into one platform. ActiveX is commonly used to develop interactive applications for the Internet Explorer (IE) Web browser, allowing users to open Microsoft Office applications from IE, but it can also be used to develop other applications.

The OLE Technology

The OLE technology allows different applications to transfer and share information. It allows an object (e.g., an image file) to be linked to a compound file (e.g., a Word document, an Excel spreadsheet, or a PowerPoint document) or be embedded in a compound document. The difference between linking and embedding is that, when changes are made to the contents of an original object linked to a compound document, the changes will be seen automatically in the compound document. However, such changes will not be seen if the object is embedded (copied) in the compound document, unless the changes are made specifically to the embedded object.

The COM Technology

The COM technology allows programs to reuse software components and existing Windows programs to add new functionalities. COM components are commonly written in C++, but they can also be written using other languages. The COM technology allows a program to unplug a COM component at runtime without recompiling the program.

ActiveX Controls

During the last 10 years, many programs have been made "active." ActiveX controls, in particular, have been widely used to develop plugins for IE. An ActiveX control is a self-registering COM object that may be embedded in an HTML document.

ActiveX controls may be activated using the <OBJECT> tags as follows:

```
<OBJECT ID="ax_example"
CLASSID="clsid:431BD693-4A33-3B46-AA7CD285CA13"
CODEBASE="http://www.ABC.com/ax_controls/"
    WIDTH=80 HEIGHT=30>
<PARAM NAME=_version VALUE="2">
</OBJECT>
```

In this example, ax_example is the name of the ActiveX control, stored under http://www.ABC.com/ax_controls with the unique hexadecimal serial number. When the client asks for the HTML document from a Web server that contains this piece of code, ax_example will be downloaded automatically to the client computer, compiled into native machine code, and loaded in the client computer's memory.

Because an ActiveX control is just similar to any executable machine code running on the client computer, it can do anything once it is downloaded into the local machine. Thus, similarly to Trojan horses, ActiveX controls could cause serious security problems.

To prevent malicious ActiveX controls from entering the client computer, users may want to disable ActiveX downloads or only allow ActiveX downloads from trusted Websites. On the other hand, ActiveX controls can be authenticated, and so users may only download authenticated ActiveX controls.

10.7.4   Cookies

Cookies are text strings of information indicating Web browsing states, which are typically used to relate a disjoint sequence of connections to a seemingly continuous connection. A Web browser is a stateless client program that establishes a new connection with a Web server for each URL request, even if the current request and the previous requests are from the same Web server. In other words, a Web browser does the following for each request: it establishes a new TCP connection to the Web server, downloads the document requested, and closes the connection. For example, to visit a password-protected Web page, the user must first type in his user name and password. After he is authenticated, he may choose to visit subsequent pages. Different, unrelated TCP connections will be established for visiting these subsequent pages. It is cumbersome to require the user to type in his password each time he visits a subsequent page. To solve this problem, the Web browser generates, after the user types in his password, a cookie to store the user information and passes it to the user's browser. When the user requests a subsequent document, the browser passes the cookie along with the user's request to the Web server. The Web server checks the information contained in the cookie. If the information is acceptable, the Web server will grant the user's request.

Web browsers often store cookies in a directory for future use. When it connects to a Web server the next time, the Web browser will search for an appropriate cookie and send it to the Web server.

Some cookies are short and some are long. A short cookie may contain just an identification number for the Web server to uniquely identify a user. A long cookie may contain user name, the IP address of the user's computer, the operating system used by the computer, and other information (e.g., a travel itinerary). It is evident that cookies of this sort would raise serious privacy concerns.

The Web server uses the Set-Cookie header to encapsulate a cookie and send it to the user's browser. For example, the Web server may generate a cookie to identify a user and send the following to the browser:

```
Set-Cookie: USER_NAME=John Doe; path=/;
            expires=Monday, September 10, 2007, 16:59
```

The browser uses the Cookie header to encapsulate a cookie stored in the local machine and send back to the Web server as follows:

```
Cookie: USER_NAME=John Doe
```

Web servers may use the Set-Cookie header and the Cookie header to keep track which pages the user has visited and sell this information. Thus, reputable Web servers will often take substantial measures to ensure that cookies cannot be used for malicious purposes. Users may also remove cookies stored in their computers every now and then.

10.7.5   Spyware

Spyware is malicious software. It is typically installed as a plugin module in the user's Web browser without the user's informed consent. Spyware monitors the user's browsing activities and collects personal information that the user does not want others to know. It can also modify system settings to do something against the user's will. In particular, spyware may perform one or more of the followings:

1. Collect information, including the user's surfing habits, favorite Websites, online shopping lists, financial information, and credit card numbers; and send the information to the attacker's computer.
2. Monitor the user's Web surfing activities and pop up a corresponding advertisement window.
3. Modify the default settings of the user's browser and redirect the user to a certain Web page.

Spyware does not replicate itself. Thus, spyware is more like a Trojan horse. The difference is that spyware may be specifically designed to run under Web browsers, and it may not be a standalone application. To lure the user to download and install spyware, the attacker may bundle a spyware program in an attractive music, game, or system management package. The attacker may advertise that the spyware is a system management tool that can help increase the user computer's surfing speed, or an antispyware tool that can detect and remove spyware.
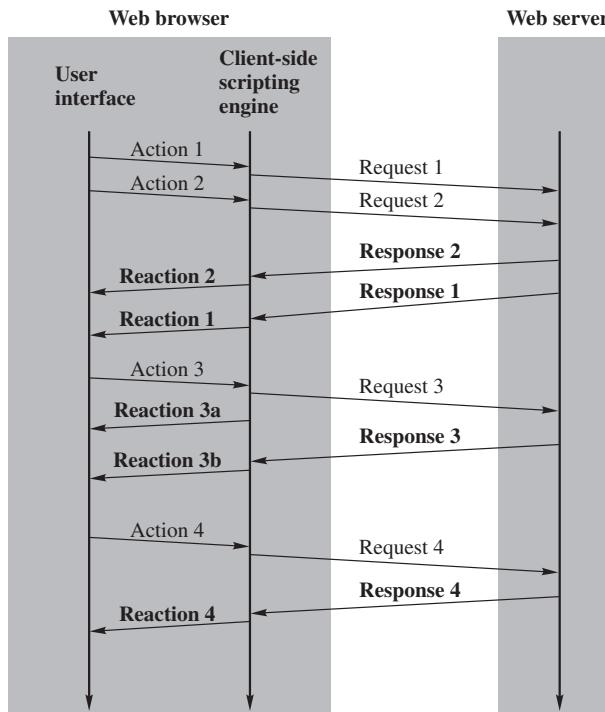
The following are common countermeasures of spyware:

1. Set up a firewall to prevent attackers from embedding spyware.
2. Install software patches in time.
3. Install antispyware software. Windows Defender, for example, is Microsoft's antispyware product that can detect, quarantine, or remove spyware in Windows operating systems.

10.7.6   AJAX Security

Traditional Web applications operate in the "click-and-wait" manner. That is, the user clicks a link or a submit button on his browser to request a page from the Web server, and then waits for the Web server to download the requested page to the user's local host. This synchronous communication of request-and-response sequence confines the interactions between the Web browser and the Web server and makes surfers feel abrupt. *Asynchronous JavaScript and XML* (AJAX) is a recently deployed technology that supports highly interactive Web applications, where asynchronous program calls can be made to the Web server without causing a full refresh of a Web page at the client side, providing smoother, faster, and seemingly continuous page updates. Google Maps, for instance, is an AJAX application. A combination of several existing technologies, AJAX plays an important part in Web 2.0, the perceived second generation of Web technologies.

AJAX achieves asynchronous interactions through a client-side JavaScript engine and server-side XML pages, where XML (acronym of Extensible Markup Language) is an extension of HTML that allows users to define their own HTML tags. Other scripting languages may also be used at the client side in place of JavaScript, and JavaScript Object

Figure 10.9   Schematic of asynchronous interactions between a Web browser and a Web server in AJAX

Notation (JSON) may be used in place of XML with less overhead. The client-side JavaScript code connects to a Web server through the XMLHttpRequest (XHR) object of JavaScript, which is triggered asynchronously by user keystrokes, timers, or other events. Figure 10.9 shows a schematic of AJAX interactions between a Web client (browser) and a Web server. In particular, the client-side scripting engine, in response to an action issued by the user, initiates calls to the Web server and updates the document at the user's browser asynchronously.

AJAX applications face the same security problems as traditional Web applications. Most of the known security attacks on AJAX applications are due to implementation flaws. Cross-site scripting, for example, is one of them. It tricks an unsuspecting user (e.g., using phishing email) to visit a malicious Web page. A mere visit to that page could cause the user's computer to download malicious JavaScript code and execute it.

For another example, we note that AJAX makes silent calls to Web servers without the consent of the user, and it replays cookies for each call. This mechanism, if not implemented properly, presents a security risk. Suppose that Alice logs on to a password-protected Website. Her browser is given a session cookie by the Web server after she is authenticated. Suppose that Alice is coaxed to browse a malicious page without logging out first and the malicious page contains AJAX code, which makes silent calls to the password-protected Website by replaying the session cookie she obtained earlier from the same Website.

10.7.7   Safe Web Surfing

In addition to using security tools such as Windows Defender, forming good Web surfing habits is essential to surfing the Web safely. The following are some of these good surfing habits.

1. Download software only from trusted Websites. If you are not sure whether a downloaded software contains malicious components, you should ask your system administrator or search the Web for information.
2. When a window pops up, do not click any button displayed on the popup window (including the cancel button), for these buttons may be traps. To remove a popup window, you should use other system methods (e.g., use the Windows Task Manager) or click the red X that appears at the corner of the window.
3. Read the privacy statements, the license statements, and the security warning statements of the downloaded software to find out the risks you are taking by installing and running the downloaded software.
4. When you visit a password-protected site, do not visit other sites with different addresses from the password-protected site.
5. Do not visit suspicious Websites.

10.8   Distributed Denial-of-Service Attacks
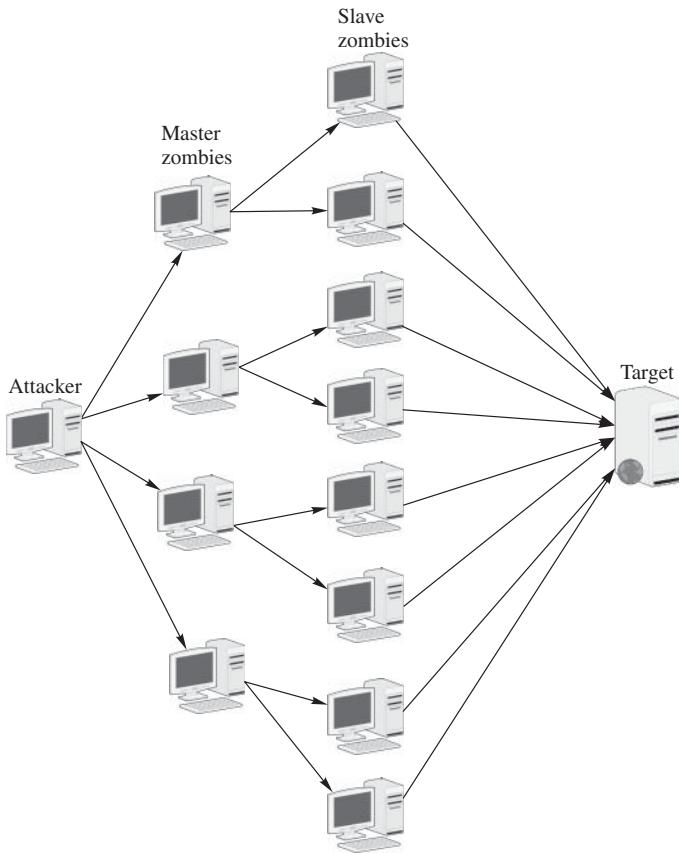
We have introduced in Section 1.2.9 the basic ideas of DDoS attacks. In general, to launch a DDoS attack, the attacker must first search for a large number of computers and lure their users to download zombie software. These zombies will then issue at the same time a large number of service requests to a selected computer to use up its computing resources. There are two types of DDoS attacks, namely, the *master-slave DDoS attack* and the *master-slave-reflector DDoS attack*.

10.8.1   Master-Slave DDoS Attacks

To prevent being traced when executing a DDoS attack, the attacker may divide zombies into master zombies and slave zombies. In particular, the attacker first obtains a host of zombies, called the *master zombies*. Each master zombie then obtains a host of its own zombies, called the *slave zombies* as if the master zombie were the attacker. To launch a master-slave attack to a particular target, the attacker issues an attack command to each of its master zombies. Each master zombie will then relay the attack command to its slave zombies. All the slave zombies will then attack the selected target at the same time. Figure 10.10 shows a schematic of master-slave DDoS attacks.

10.8.2   Master-Slave-Reflector DDoS Attacks

Attackers may also have master zombies order their slave zombies to launch a Smurf type of attack to a selected target. For example, each slave zombie sends a large number of crafted

Figure 10.10 Schematic of master-slave DDoS attack
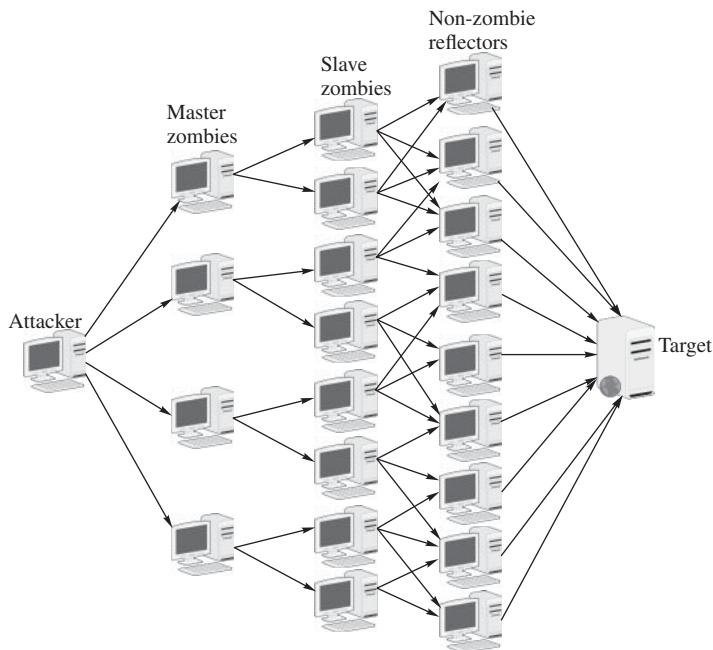
`ping` packets to the non-zombie computers it can find, where the source address in the crafted packet is the IP address of the target. The non-zombie computers used by slave zombies are referred to as *reflectors*. Figure 10.11 shows a schematic of master-slave-reflector DDoS attack.

10.8.3 DDoS Attacks Countermeasures

To be successful, a large-scale DDoS attack depends on several conditions. Firstly, it must use good zombie software. Secondly, it must have a large number of exploitable computers on the Internet that can be turned to zombie machines. Thirdly, the attacker must be able to find these computers.

Thus, to counter DDoS attacks, one should try to eliminate the last two conditions from the equation. That is, reduce the number of vulnerable computers and make it hard for the attackers find a vulnerable computer.

The first objective can be achieved by improving security management of networked computers so that the computers cannot be turned to zombies easily. Moreover, when a DDoS

Figure 10.11   Schematic of master-slave-reflector DDoS attack

attack comes, the target computer may still be able to detect it and help trace the source of attacks. These include the following measures:

1. Set up a backup system such that when the active servers suffer from DDoS attacks, the backup system can be activated to provide normal services.
2. Distribute resources appropriately and modify communication protocols to reduce the possibilities of becoming a DDoS victim.
3. Construct a DDoS monitoring and responding system that can detect DDoS attacks on the basis of DDoS behaviors and respond to DDoS attacks when they happen. For example, remove DDoS attack packets.
4. Keep a complete system log to help trace the sources of DDoS attacks.

The second objective can by achieved by the following measures:

1. Close all unnecessary ports to defy IP scans.
2. Automatically disconnect (e.g., shutdown the computer) the network connection of the user's host computer when it is no longer in use. This will help reduce the possibility of the computer to be used as a zombie in a DDoS attack, although it may have been invaded by zombieware.
3. Detect and remove zombieware.

10.9   Closing Remarks

Malicious software programs have become the major security foes of computer security. They enter networked computers through system loopholes and through a user's lack of care. Malicious software may modify or delete user files, collect information, or simply turn a computer to a background server for the attacker, stealing the user's computing, network, and power resources. Thus, in addition to having a good firewall, the user should also install a good anti-malicious software product (including antivirus software, antispyware software, and antispam software) and form good surfing habits.

10.10   Exercises

10.10.1   Discussions

10.1. Describe your experiences of being attacked by malware.

10.2. Describe your experiences in dealing with malware attacks.

10.3. If you have a Trojan in your system, how do you think it gets there?

10.4. What security measures should you take when surfing the Web?

10.5. Discuss the general tactics an attacker would often use to lure you to their malware.

10.6. Discuss your experiences in dealing with DoS attacks.

10.10.2   Homework

10.1. Table 10.4 lists common worms and their port numbers. Search relevant references and describe what each of these worms will do.

10.2. Construct ACL rules to block the worms listed in Exercise 10.1 from entering the internal network.

10.3. Table 10.5 lists common Trojan horses and their port numbers. Search relevant references and describe what each of these Trojan horses will do.

10.4. Construct ACL rules to block the Trojan horses from entering the internal network listed in Exercise 10.3.

10.5. What are the common methods to spread viruses?

10.6. Describe heuristic scanning for finding viruses. The textbook provides an example of heuristic scanning. Please provide a different example of heuristic scanning.

10.7. "A few years back I had a test laptop at work that was infected with the w32.blaster malware. I decided to try out the Symantec removal tool to remove it

Table 10.4 Common worms and their port numbers

Port	Protocol layer	Name
445	TCP	Zotob
1080	TCP	MyDoom.B
2041	TCP	W32.korgo
2745	TCP	Bagle.C
3067	TCP	W32.korgo
3127	TCP	MyDoom.A
3128	TCP	MyDoom.B
5554	TCP	Sasser–FTP server
8080	TCP	MyDoom.B
8998	UDP	Sobig.F
9898	TCP	Dabber
9996	TCP	Sasser–remote shell
10080	TCP	MyDoom.B

Table 10.5 Common Trojan horse and their port numbers

Port	Protocol layer	Name
1243	TCP	SubSeven
1349	UDP	Back Orifice DLL
1999	TCP	SubSeven
2583	TCP and UDP	WinCrash
6711	TCP	SubSeven
6776	TCP	SubSeven
8787	TCP and UDP	Back Orifice 2000
12345	TCP	NetBus
12346	TCP	NetBus Pro
27374	UDP	SubSeven
54320	TCP and UDP	Back Orifice 2000
54321	TCP and UDP	Back Orifice 2000
57341	TCP and UDP	NetRaider

(www.symantec.com/security_response/writeup.jsp?docid=
2003-081119-5051-99)) and it seemed to work. But I ended up wiping the system anyway and reloading the OS."

(a) Conduct a research on the w32.blaster malware.

(b) What are the pros and cons of reloading the OS as a measure to clean malware?

10.8. Visit http://www.avast.com/eng/download-avast-home.html, download avast! 4 Home free of charge for home users, and install it on your computer. Then run it to scan your system. According to what you see, explain the mechanism of this antivirus product.

10.9. Describe the similarities and differences between McAfee VirusScan, Norton AntiVirus, and Avast! AntiVirus. For example, do they perform preinstallation scanning? Do they support heuristic scanning? Do they provide system security levels?

10.10. Search from the relevant Websites and list the new viruses and worms that have occurred in the last 2 weeks.

*10.11. Search for relevant references and describe in details how the Code Red worm used the buffer-overflow attack to run the main worm code.

10.12. "The Conficker worm attack in 2009 affected some of the standalone systems I managed through the financial regulator's Website," a reader told us. "I was able to clean up that using the McAfee "stinger" fix released for the Conficker malware."

Conduct a research on the Internet and explain what the Conficker worm is and how it is disseminated.

10.13. "In 2009, we were infected by the Conficker worm in my organization. PCs that were previously infected by previous versions of the Conficker worm were triggered by the updated version of the worm, which were undetected. It was a nightmare. The worm disabled system AV products and utilizes scheduled tasks for reinfection. The worm also infected other systems via USB drives and password guessing. The worm thrived in environments with Windows OS shares, weak passwords, lack of software updates, and the unrestricted Autorun functionality for removable media. We had to go to roughly 400 PCs and removed each instance on a case by case basis. As a result, although we did not disable Autoruns, we did change the password security policy."

Suggest counter measures to prevent the Conficker worm from getting your system. Justify your solutions.

10.14. "I have had my home systems attacked many times through the years, even once with the Lsass malware," a reader said. "At that time a certified Microsoft technician helped me get rid of it. Since then my machines have been attacked by overly aggressive adware and spyware with strange popups and redirection of my homepage. Once I even had my machine infiltrated by a 'performance improving' and 'antivirus program' advertised on local TV, which acted much more like spyware. It took me several days to get rid of this memory hogging software. I had to eliminate or change Registry keys, eliminate certain suspicious, duplicated system files, rename malicious files before deletion and do some other 'out of the box' things before I could get rid of it. In recent years I have found it far preferable to run untrustworthy downloaded software on an OS installed on a VM like VMWare, to check what it will do to system files and to the Registry."

(a) Conduct a research on the Lsass malware mentioned by this reader.
(b) Share your experience of removing malware where you had to use some out-of-the-box methods.

10.15. "A few years ago I had to deal with a virus that lodged itself somehow on the boot process," a reader told us. "Apparently even Avast! that I had installed on my computer wasn't able to detect this virus. I suspected one of my kids downloaded

free games, which were probably where the virus came from. After a few attempts I had to use an anti-root-kit to get rid of the virus. This incidence has made me aware that security issues can strike anyone on a personal level. As they say, many things begin at home and this experience certainly taught me a good lesson to be more security conscious."

Have your experienced similar situations? Give one or more examples.

10.16. "My friend's Windows XP computer was once locked by the FBI Trojan and he was instructed by a popup notice to pay $200 or $300 to get it unlocked. I tried to clean the computer by hand like deleting the entries in registry (Windows/Current Version/ Run or RunOnce) and tried the Windows restore in Windows XP, but none of these measures worked. Finally I was able to use Microsoft Security Essentials (MSE) to remove it. But I needed to run MSE a few times to clean the computer completely." A reader of the first edition shared this story. Can you explain what happened to this person's computer?

10.17. "About two and a half years ago (i.e., sometime in 2012)," said a reader of the first edition, "I downloaded a desktop background from a site where there were lots of backgrounds available. It came with some sort of spy/malware/adware code attached to it. I rendered the computer unusable and I had to call Information Services to remove it. It was something that they claimed not to have seen before, and it took them about 45 minutes to clear my computer. I don't download such images anymore."

Conduct a search on the Internet and make an educational guess on what happened to this reader's computer.

10.18. "I find a lot of people that I help in my spare time encounter viruses and worms constantly," a reader to us. "There are so many different types out there that can cause a range of different issues. I find when trying to deal with these infections there is no one piece of software that can fix all of these types of infections. You have to build a whole toolbox of different types of software that can help you detect, quarantine, and successfully clean the infections. I have found sometime it is best to run multiple piece of software to clean and ensure that the virus or worm has been removed. Sometimes you cannot be successful and the only way to fix these types of issues is to reformat and start fresh. I tell all my customers to have a virus scan, antispyware software, and make sure that they keep their OS patches up to date."

In addition to the suggestions mentioned here, can you add a few more measures that we should take to protect our systems?

10.19. Use examples to explain why a system administrator should not use a user account with super-user password to browse the Web.

10.20. Use examples to explain why a system administrator should not use an user account with super-user password to send or receive email.

10.21. A zero-day attack is an attack that exploits a previously unknown flaw in an application and a patch has not been released. "Our company was subject to a zero-day attack a few months ago (i.e., in late 2013)," a reader told us. "That was the first

time our IT had heard of such attacks. The attackers had found a hole in the OS we were using before the developers did or had the chance to patch it." Can you suggest a good preventative measure to deal with zero-day attacks? Justify your answer.

10.22. A reader of the first edition shared the following story with us: "I remember that my computer has been infected a few times from the Websites I visited. I often search for solutions to my programming issues over the Internet. The sites I have frequented are hosted by very knowledgable people. A few years back, before really knowing the trusted sites, I often wound up in places that seemed legitimate, but after clicking a link inside of a post once, immediately my computer stopped responding and crazy stuff started filling my screen. Knowing what was happening, I shut down my computer (the old method of holding the power key down for five seconds), then started it back up in safe mode. With another computer next to me to guide me, I went through the process of restoring a backup from Windows of a few days earlier and all was good. I installed a new virus scan later, and the scan verified that there was no infection, and I was all good. Since then, I have tried to stick with official Microsoft sites, ExpertsExchange, and a few other sites that provide 90% of what I'm looking for. I think now, with modern updated malware protection, I feel pretty safe to check out some unknown sites if they seem to have a potential answer that none of the others do."

Do you agree with the reader's current practice? Justify your answer.

10.23. In Section 10.7.6, we introduced two security vulnerabilities in AJAX. There are more security problems in AJAX. Search the relevant literature and describe two additional security vulnerabilities in AJAX.

*10.24. Web 2.0 is a term coined in 2004 for the purpose of representing the second generation of Web technologies. The first generation of Web technologies is denoted by Web 1.0. There is a clear distinction between Web 2.0 and Web 1.0 technologies. Table 10.6 lists a few Web 1.0 technologies and the corresponding Web 2.0 technologies.

The list can go on. While Web 2.0 has the same types of security issues as Web 1.0, it has added several new dimensions. Search the relevant literature and describe five security issues in Web 2.0.

Table 10.6   Web 1.0 versus Web 2.0

Web 1.0 technology	Web 2.0 technology
Personal Web pages blogs	Blogs
Akamai	BitTorrent
mp3.com	Napster
DoubleClick	Google AdSense
Britannica Online	Wikipedia
Content management systems	Wikis

10.25. Visit `http://www.microsoft.com/downloads`, download Windows Defender free of charge, and install it on your computer. Then run it to scan for spyware. According to what you see, explain the mechanism of this antispyware product.

10.26. In Windows operating systems, cookies for IE are stored on the C drive under the Documents and Settings directory. First find your user name, and then open the Cookies directory. Select at random a cookie file and open it. Explain what you see, and answer the following questions.

(a) If cookies are transmitted to the Web servers in plaintext, list and describe the potential security threats the clients may face.

(b) If users are allowed to modify cookies stored on local computers, list and describe the potential security threats the Web servers may face.

10.27. When Alice surfs the Web, she will leave certain information at the Web servers because her computer needs to establish TCP connections with the Web server computers, including her user name, her computer's IP address, and the type and version of the operating system on her computer. To surf the Web anonymously, one way to do so is to use a Web proxy server to relay connections between Alice and the Web servers. Anonymizer is such a system. Download Anonymizer from `http://www.anonymizer.com` to surf the Web anonymously.

*10.28. Using a Web proxy server to surf the Web, although the actual Web server does not know the client's information (all requests will come from the proxy server), the proxy server still knows it. This means that the user has to trust the proxy server. Can one surf the Web anonymously without using any proxy server? Freenet is a protocol that allows the user to surf and publish articles in the Web anonymously. Conduct a research on Freenet and write a paper of about 4000 words describing its usage, architecture, and algorithms.

**10.29. NGVCK generates metamorphic viruses. In particular, it generates a piece of viral code that does the same thing of the same virus, but with an almost completely different structure each time. This would defy basic scanning. It has been suggested that using the hidden Markov model (HMM) can effectively detect metamorphic viruses.

HMM models a system statistically using a Markov process with unknown parameters. It provides a formal method to determine hidden parameters from observable ones.

Describe how you may use HMM to detect metamorphic viruses.

10.30. "Recently (that is, in early 2014) we encountered at work a potential threat originating from the Amazon cloud," a reader reported. "One of our business units at my company alerted us that their Web server was spitting out massive amounts of Internal server errors. We were thinking at first that the Web server could not handle the standard HTTP traffic and there might be a problem with the code. Looking at the logs, however, we noted a pattern where the traffic would flood in at a certain time in the evening. But the amount was barely noticeable by our firewalls or

load balancers, for the source IPs were coming from different networks owned by Amazon. As a precaution we denied all networks owned by Amazon from accessing this external IP we hosted. It has been two months since we instrumented this rule of denying Amazon traffic. The performance problems went away as soon as we blocked this traffic."

Can you help the reader to pinpoint what went wrong in their Web server?

10.31. One of the readers of the first edition shared with us the following incident:

"I had once experienced a DoS incident (not a malicious attack) that was resulted from a configuration error in the automatic update policy of an application. The application was attempting to download external updates for over 500 locally installed workstations at the same time. This configuration error caused extended communication outages for mission critical systems already constrained with satellite bandwidth. Primary measures to counter such incidents we took included active network monitoring, firewall log checking, and continuous administrator training so that changes in regular network traffic can be identified sooner. Also, adherence to a detailed configuration management plan is important to reduce network intrusions."

(a) Discuss the measures they took were reasonable ones.

(b) Can you think of other or better measures? Justify your answer.

Appendix A

7-bit ASCII code

	000	001	010	011	100	101	110	111
0000	nul	soh	stx	etx	eot	enq	ack	bel
0001	bs	ht	nl	vt	np	cr	so	si
0010	dle	dcl	dc2	dc3	dc4	nak	syn	etb
0011	can	em	sub	esc	fs	gs	rs	us
0100	space	!	"	#	$	%	&	'
0101	(	)	*	+	,	-	.	/
0110	0	1	2	3	4	5	6	7
0111	8	9	:	;	<	=	>	?
1000	@	A	B	C	D	E	F	G
1001	H	I	J	K	L	M	N	O
1010	P	Q	R	S	T	U	V	W
1011	X	Y	Z	[	\	]	^	_
1100	`	a	b	c	d	e	f	g
1101	h	i	j	k	l	m	n	o
1110	p	q	r	s	t	u	v	w
1111	x	y	z	{	|	}	~~	del

Row numbers represent the 4-bit prefix, and the column numbers represent the 3-bit suffix. The first 32 ASCII codes and the last ASCII code are control codes, which are not displayable.

Appendix B

SHA-512 Constants (in Hexadecimal)

i	K_i	i	K_i	i	K_i
0	428a2f98d728ae22	1	7137449123ef65cd	2	b5c0fbcfec4d3b2f
3	e9b5dba58189dbbc	4	3956c25bf348b538	5	59f111f1b605d019
6	923f82a4af194f9b	7	ab1c5ed5da6d8118	8	d807aa98a3030242
9	12835b0145706fbe	10	243185be4ee4b28c	11	550c7dc3d5ffb4e2
12	72be5d74f27b896f	13	80deb1fe3b1696b1	14	9bdc06a725c71235
15	c19bf174cf692694	16	e49b69c19ef14ad2	17	efbe4786384f25e3
18	0fc19dc68b8cd5b5	19	240ca1cc77ac9c65	20	2de92c6f592b0275
21	4a7484aa6ea6e483	22	5cb0a9dcbd41fbd4	23	76f988da831153b5
24	983e5152ee66dfab	25	a831c66d2db43210	26	b00327c898fb213f
27	bf597fc7beef0ee4	28	c6e00bf33da88fc2	29	d5a79147930aa725
30	06ca6351e003826f	31	142929670a0e6e70	32	27b70a8546d22ffc
33	2e1b21385c26c926	34	4d2c6dfc5ac42aed	35	53380d139d95b3df
36	650a73548baf63de	37	766a0abb3c77b2a8	38	81c2c92e47edaee6
39	92722c851482353b	40	a2bfe8a14cf10364	41	a81a664bbc423001
42	c24b8b70d0f89791	43	c76c51a30654be30	44	d192e819d6ef5218
45	d69906245565a910	46	f40e35855771202a	47	106aa07032bbd1b8
48	19a4c116b8d2d0c8	49	1e376c085141ab53	50	2748774cdf8eeb99
51	34b0bcb5e19b48a8	52	391c0cb3c5c95a63	53	4ed8aa4ae3418acb
54	5b9cca4f7763e373	55	682e6ff3d6b2b8a3	56	748f82ee5defb2fc
57	78a5636f43172f60	58	84c87814a1f0ab72	59	8cc702081a6439ec
60	90befffa23631e28	61	a4506cebde82bde9	62	bef9a3f7b2c67915
63	c67178f2e372532b	64	ca273eceea26619c	65	d186b8c721c0c207
66	eada7dd6cde0eb1e	67	f57d4f7fee6ed178	68	06f067aa72176fba
69	0a637dc5a2c898a6	70	113f9804bef90dae	71	1b710b35131c471b
72	28db77f523047d84	73	32caab7b40c72493	74	3c9ebe0a15c9bebc
75	431d67c49c100d4c	76	4cc5d4becb3e42b6	77	597f299cfc657e2a
78	5fcb6fab3ad6faec	79	6c44198c4a475817		

Appendix C
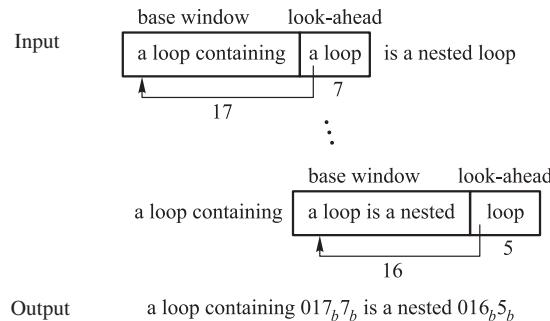
Data Compression Using ZIP

ZIP is a simple matching algorithm using two sliding windows, called the *base window* and the *look-ahead window*. These two windows are placed side-by-side on the data file, where the look-ahead window goes ahead of the base window. ZIP scans the entire file by sliding these two windows and encoding data on the fly. In particular, ZIP finds the longest prefix s of the data string contained in the look-ahead window that also appears in the base window. This string in the look-ahead window (if found) is a copy of s in the base window, and so it can be uniquely identified by two attributes: (1) the distance between the location of the first character of s in the base window and the location of the first character in the look-ahead window and (2) the length of s. If the space needed to hold the values of these two attributes is smaller than the space needed to hold s, we obtain a saving of space.

To implement this idea, we will need to distinguish the binary values of the two attributes from normal encodings of characters. Suppose that the data file is encoded using the 8-bit ASCII code set. If the first bit is used as a parity bit, then it could be either 0 or 1. The first bit of the binary string representing the two attributes can also be either 0 or 1. Thus, to make a distinction, we add an extra bit of 1 in front of each ASCII code to yield a 9-bit extended ASCII code and add an extra bit of 0 in front of the binary string representing the two attributes. This simple encoding uniquely identifies the original data file.

In particular, let w_1 denote the number of characters the base window can hold, where $2^{d-1} < w_1 \leq 2^d$ for some $d \geq 1$. Let w_2 denote the number of characters the look-ahead window can hold, where $2^{l-1} < w_2 \leq 2^l$ for some l with $1 \leq l \leq d$. This produces a $(d + l + 1)$-bit binary encoding for s, where the first bit is 0 (used as an indicator), the next d bits represent the distance, and the last l bits represent the length. For convenience, we call this $(d + l + 1)$-bit code a *location code*.

A location code is easily distinguishable from any 9-bit extended ASCII code because a location code has a fixed length and an indicator 0 different from the indicator in a 9-bit extended ASCII code. In other words, given a compressed file using this encoding method, it can be uniquely and easily "uncompressed" back to its original ASCII format. The proof is left to the reader (see the Exercise). Thus, as long as $d + l + 1 < 8k$, where $k = |s|$, ZIP may save space. ZIP then shifts both of the base window and the look-ahead window to the right

Figure C.1   A demonstration of a ZIP process

$\max\{1, k\}$ times and repeats the same procedure until the look-ahead window is shifted out of the data file.

For example, let $w_1 = 18$ and $w_2 = 7$. Then $d = 5$ and $l = 3$. Let us consider the following text string:

<div align="center">"a loop containing a loop is a nested loop"</div>

Denote by n_b, the binary representation of positive integer n. Running ZIP on this character string produces the following output (see Figure C.1):

<div align="center">"a loop containing $017_b 7_b$ is a nested $016_b 5_b$",</div>

where each letter and space in the output string is encoded by a 9-bit extended ASCII code. For clarity, we do not spell out the location code in binary. The length of the "compressed" string in binary is therefore equal to $18 \times 9 + 9 + 11 \times 9 + 9 = 279$ bits, and the length of the original character string encoded in the 8-bit ASCII code set is equal to $41 \times 8 = 328$ bits. Thus, ZIP has compressed the original data string to a shorter binary string.

To decode a compressed file, ZIP scans it from the beginning, removes the leading 1 from each 9-bit extended code, and replaces each 9-bit code with leading 0 by the corresponding character substring using the distance and length attributes.

Exercise

Show why the 10-bit code defined in this appendix is easily distinguishable from the extended 9-bit ASCII code. That is, given a compressed file using this encoding method, show that it can be uniquely and easily "uncompressed" back to its original ASCII format.

Appendix D

Base64 Encoding

Base64 encoding represents a 6-bit binary string using a printable character (see Table D.1), where a 6-bit value of 0–25 represents an upper-case letter A–Z correspondingly; a 6-bit value of 26–51 represents a lower-case letter a–z correspondingly; a 6-bit value of 52–61 represents a digit 0–9 correspondingly; and the last two 6-bit values of 62 and 63 represent "+" and "/", respectively. Transmitted in ASCII format, this means that every 6-bit string is replaced with an 8-bit string.

In addition, Base64 encoding uses character "=" as a special indicator. Using Base64 encoding, a binary string is converted to a character string as follows:

Case 1: The binary data consists of only one byte. Pad it at the end with 16 0's to extend it to a 24-bit string. This 24-bit string is then converted to a Base64 string of four characters, with "==" being the last two characters. This indicates that only the first two characters are to be decoded, and the suffix 0000 is discarded.

Case 2: The binary data consists of only two bytes. Pad it at the end with eight 0's to extend it to a 24-bit string. This 24-bit string is then converted to a Base64 string of four characters, with "=" being the last character. This indicates that only the first three characters are to be decoded, and the suffix 00 is discarded.

Case 3: The binary data consists of at least three bytes. Place the first three bytes of the binary data into a 24-bit buffer, where the first byte is placed in the most significant eight bits of the buffer, the second byte is placed in the middle, and the third byte in the least significant eight bits. This 24-bit string is then converted to a Base64 string of four characters. Repeat this process until there is no byte left, there is one byte left, or there are two bytes left. The conversion is completed if there is no byte left. If there is one byte left, apply Case 1 to this byte to complete the conversion. If there are two bytes left, apply Case 2 to these two bytes to complete the conversion.

Given in Table D.2 are several examples of Base64 conversions.

Decoding a Base64 string back to the original binary data is straightforward and is left to the reader (see Exercise).

The Base64 encoding was first used in the Privacy-enhanced Electronic Mail (PEM) protocol for transferring electronic data.

Table D.1 Base64 encoding

6-bit value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
character encoding	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
6-bit value	20	21	22	23	24	25	26	27	28	29	31	31	32	33	34	35	36	37	38	39
character encoding	U	V	W	Z	Y	Z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
6-bit value	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
character encoding	o	p	q	r	s	t	u	v	w	x	y	z	0	1	2	3	4	5	6	7
6-bit value	60	61	62	63																
character encoding	8	9	+	/																

Table D.2 Examples of Base64 conversions, where boldface bits are padding bits

Binary string	10110011 (one byte)
24-bit buffer	101100 11**0000 000000 000000** (padding of two bytes)
Base64 conversion	sw==
binary string	10110011 00000101
24-bit buffer	101100 110000 0101**00 000000** (padding of one byte)
Base64 conversion	swU=
binary string	10110011 00000101 01100010
24-bit buffer	101100 110000 010101 100010 (no padding)
Base64 conversion	swVi

Exercise

Describe how to decode Base64 strings back to their original binary strings.

Appendix E

Cracking WEP Keys Using WEPCrack

This appendix describes an experiment to crack a WEP-protected WLAN using WEPCrack, an open-source WEP cracking tool. WEPCrack implements the RC4 weak-key attack introduced in 2001 by Fluhrer, Mantin, and Shamir. It is written in the Perl language. Stephen Brinton designed and implemented the experiments.

E.1   System Setup

The experiment uses three computers and one WEP-enabled Linksys wireless router as an AP. One computer serves as an Apache Web server, which is connected to the router via an Ethernet cable. The second computer is a WEP-enabled wireless laptop PC connected to the router. The router and the laptop computer share a 104-bit secret WEP key K. This computer continuously requests Web pages from the Web server for the purpose of generating a large number of frames. The third computer is also a laptop PC equipped with a WEP-enabled wireless network interface card (NIC) that can monitor network traffic. This computer runs WEPCrack to crack the WEP key K. Figure E.1 shows the system setup of this experiment.
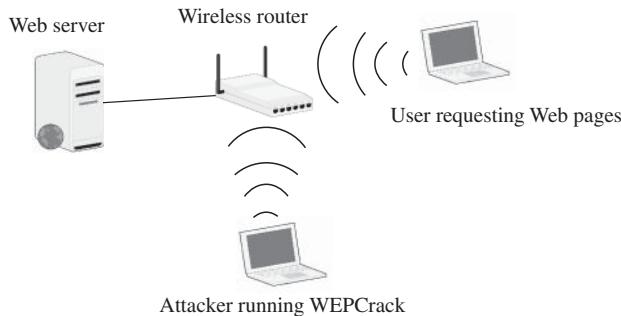
The experiment uses the following AP and wireless NICs:

AP

The AP used in the experiment was a WEP-enabled Linksys Wireless-B Broadband Router.

User's Network Card

Device:	Belkin F5D7010 54g Wireless Network card
Driver:	ndiswrapper (Belkin: bcmwl5.inf)
Vendor:	Broadman

Figure E.1   WEPCrack experiment system setup

Attacker's Network Card

Device:	AR5212 802.11 abg (Netgate)
Device Name:	ath0
Driver:	ath_pci
Vendor:	Atheros Communications, Inc.

E.2   Experiment Details

WEPCrack cracks WEP keys by first collecting weak initialization vectors. After sufficient information about weak initialization vectors is obtained, WEP Crack deduces from it the WEP key used in the WLAN. It may take a number of hours to collect information. After that, the actual cracking part may take only a few minutes.

Step 1: Initial Setup

Select a 104-bit WEP key for both the AP (the router) and the STA (the laptop computer that will continuously request Web pages from the Web server). In the experiment, the WEP key is chosen as a 13-byte binary string

$$K = 96 \quad 6 \quad 91 \quad 24 \quad 207 \quad 211 \quad 39 \quad 92 \quad 158 \quad 7 \quad 240 \quad 37 \quad 234.$$

Start the Apache server using the `#rcapache2 start` command. The STA starts the requester program `requester.c` (see Section E.3) using

```
./requester 172.16.1.1 80 GET /
```

where `172.16.1.1` is the IP address of the Apache server. This produces continuous request and sending of a web page over the wireless connection.

Step 2: Attacker Setup

The attacker's laptop runs Linux. First run the `ifconfig ath0 up` command to enable the laptop's NIC. Then run the `iwconfig ath0 scan` command to search for the AP within range and collect its MAC address, channel, and `essid` information. The `iwconfig ath0 scan` command returns the following output:

```
ath0   Scan completed
       Cell 01 -- Address: 00:11:F5:1D:98:04
       ESSID: "Gates"
       Mode: Master
       Frequency: 2.442 GHz (Channel 7)
       Quality = 43/94 Signal level = -52 dBm
       Noise level = -95 dBm
       Encryption Key: on
       Bit Rate: ...
```

Finally, configure the NIC using the following commands:

```
ifconfig ath0 down
iwconfig ath0 channel 11
iwconfig ath0 ap 00:06:25:F3:CD:89
iwconfig ath0 essid ResearchAP
iwconfig ath0 mode monitor
ifconfig ath0 up
```

Step 3: Collecting Weak Initialization Vectors

Start `Wireshark` and open the capture window to capture wireless frames. Then run the WEPCrack program `pcap-getIV.pl` using the following command: `./pcap-getIV.pl -i ath0`. This may take several hours to run to collect sufficient information. This program produces a log file named IVFile.log, which contains weak initialization vectors and encrypted outputs. They will be used to help reveal the WEP key.

Step 4: Cracking

Run WEPCrack.pl on IVFile.log to deduce the WEB-kep. After only a few minutes of execution, WEPCrack arrived at the correct encryption key shown as follows, where $ is the Linux prompt:

```
$./WEPCrack.pl
Keysize = 13 [104 bits]
96 6 91 24 207 211 39 92 158 7 240 37 234
```

E.3   Sample Code

The STA executes the following program, written by Stephen Brinton, to keep requesting Web pages.

```
requester.h

/***********************************************
 Header name: requester.h
 ***********************************************/

 #include <sys/types.h>
 #include <sys/socket.h>
 #include <netinet/in.h>
 #include <arpa/inet.h>
 #include <netdb.h>
 #include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 #include <fcntl.h>
 #include <sys/stat.h>
 #include <unistd.h>

 // Maximum Sizes
 #define BUFSIZE 1024
 #define HOST_NAME_SIZE 256
 #define COMMAND_NAME_SIZE 3
 #define FILENAME_SIZE 256
 #define PORTNUMBER_SIZE 4

 #define QLEN 128

requester.c

/***********************************************************
 Filename: requester.c
 Designer: Stephen Brinton - UML
 Overview: This program will continuously request and print
          Web pages Usage:

          client host portnumber command filename

 Example:  ./requester www.cnn.com GET index.html
 Function: make_socket() - makes a socket connection
 ***********************************************************/

 #include "requester.h"

 int main(int argc, char *argv[])
 {
    int sd;               // socket descriptor ID
```

```
    int n;              // number of characters to/from socket
    char msg[BUFSIZE]; // buffer used to hold socket message
    char host[HOST_NAME_SIZE];    // host address
    char command[COMMAND_NAME_SIZE]; // command - GET or PUT
    char filename[FILENAME_SIZE];    // filename to GET/PUT
    char port_number[PORTNUMBER_SIZE];
    // store portnumber from command line arguments

    int portnumber;  // portnumber to GET/PUT

    // **** GATHER THE ARGUMENTS FROM THE COMMAND LINE ****
    if (argc != 5)        // check if there are 5 arguments
    {                     // print error message otherwise
        fprintf(stderr, "Error - Usage:
                client host port_number command filename\n");
        exit(1);
    }
    {
        sprintf(host,argv[1]);
        sprintf(port_number,argv[2]);
        portnumber = atoi(port_number);
        sprintf(command,argv[3]);
        sprintf(filename,argv[4]);
    }

    while(1)
    {
        if (strcmp("GET",command)!=0 && strcmp("PUT",command)!=0)
        {
            fprintf(stderr, "Error - Invalid command entered:
                    %s (Must be either PUT or GET)\n", command);
                    exit(1);
        }
        // setup command to be sent through socket to host
        if (strcmp("GET",command)==0)  // Process the GET command
        {
            sprintf(msg, "GET %s HTTP/1.0\r\nHost: %s\r\n\r\n",
                    filename,host);
            if ((sd = make_socket(portnumber, host))== -1)
            {
                exit(1);
            };
            write(sd,msg,strlen(msg));
        }
        else // PUT command
        {
            FILE* fptr;
            int fd;
            int bytes_read;
            struct stat file_info;
            char* buffer;
```

```
        size_t length;

        if ((fptr = fopen(filename, "rb")) == NULL)
        {
            fprintf(stderr, "Error - File Not Found\n");
            close (sd);
            exit(1);
        }
        fd = fileno(fptr);
        fstat(fd, &file_info);
        length = file_info.st_size;
        if (!S_ISREG (file_info.st_mode))
        {
            fprintf(stderr, "Error - File is not regular\n");
            close (fd);
            close(sd);
            exit(1);
        }
        sprintf(msg, "PUT /%s HTTP/1.0\r\nHost:
                %s\r\nContent-type:
                text/plain\r\nContent-length:
                %d\r\n\r\n",filename,host,length);
        if ((buffer=(char*)malloc(length+strlen(msg)))==NULL)
        {
            fprintf(stderr, "Error - Insufficient
                    memory available to send file\n");
            close (fd);
            exit(1);
        }
        memcpy(buffer, msg, strlen(msg));
        bytes_read=fread(buffer+strlen(msg),1,length,fptr);
        close (fd);
        if ((sd = make_socket(portnumber, host))== -1)
        {
            free(buffer);
            exit(1);
        };
        write(sd,buffer,bytes_read+strlen(msg));
    }

    // **** READ AND DISPLAY MESSAGES FROM SOCKET ****
    // read from socket and keep doing it until nothing
    // remains in socket
    n = recv(sd,msg,sizeof(msg),0);
    while (n>0)
    {
        write(1,msg,n);
        n = recv(sd,msg,sizeof(msg),0);
    }
    close(sd);
}               // **** CLOSE CONNECTION ****
```

```c
        return(0);
 }


/***************************************************************
 Function name: make_socket
 Overview: This function setups a socket to be used by this client
 **************************************************************/

 int make_socket(int portnumber, char* host)
 {
    struct hostent *ptrh;           // pointer used by gethostbyname
    struct sockaddr_in sad;
    int sd;                         // socket descriptor ID
    // **** PREPARE THE ADDRESS TO BE USED IN MAKING THE CONNECTION
    memset ((char *)&sad, 0,sizeof(sad));
    sad.sin_family = AF_INET;
    sad.sin_port = htons((u_short)portnumber);
    ptrh = gethostbyname(host);
    if (((char *)ptrh) == NULL)
    {
        fprintf(stderr,"Error-Invalid host entered: %s\n",host);
        return -1;
    }
    memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);

    // **** MAKE THE SOCKET ****
    sd = socket(PF_INET, SOCK_STREAM, 0);
    if (sd < 0)
    {
        fprintf(stderr, "Error - Socket creation failed\n");
        return -1;
    }
    // **** CONNECT TO SERVER ****
    if (connect(sd, (struct sockaddr *)&sad, sizeof(sad))<0)
    {
        fprintf(stderr, "Error - Connect failed\n");
        return -1;
    }
    return sd;
 }
```

Appendix F

Acronyms

ACK	Acknowledgement
ACL	Access Control List
AES	Advanced Encryption Standard
AH	Authentication Header
AJAX	Asynchronous JavaScript and XML
ALG	Application-Level Gateway; Application-Layer Gateway
AMS	Anti-Malicious Software
ANSI	American National Standard Institute
AP	Access Point
ARP	Address Resolution Protocol
AS	Authentication Server
ASCII	American Standard Code for Information Interchange
ASIC	Application-Specific Integrated Circuit
ASP	Active Server Page
AVI	Audio-Video Interleaved
AWS	Amazon Web Services
BTC	Bitcoin unit
CA	Certificate Authority
CBC	Cipher-Block-Chaining Mode
CBC-MAC	Cipher-Block Chaining Message Authentication Code
CCMP	Counter Mode-CBC MAC Protocol
CEO	Chief Executive Officer
CERT	Computer Emergency Response Team (USA)
CGI	Common Gateway Interface
CIA	Central Intelligence Agency (USA)
CIFS	Common Internet File System

CHF	Cryptographic Hash Function
CLG	Circuit-Level Gateway
COFF	Common Object File Format
COM	Component Object Model
CPU	Central Processing Unit
CQA1	Chosen-Query Attack
CAQ2	Adaptive Chosen-Query Attack
CRC	Cyclic Redundancy Check
CTR	Center
CWG	Conficker Working Group
DAC	Data Authentication Code
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
DiF	Distributed Firewall
DIS	Digital Immune System
DLL	Dynamic Link Library
DMVPN	Dynamic Multipoint VPN
DMZ	Demilitarized Zone
DDoS	Distributed Denial of Service
DHCP	Dynamic Host Configuration Protocol
DHBS	Double-Homed Bastion System
DoS	Denial of Service
DPF	Dynamic Packet Filter
DSL	Digital Subscriber Line
DZ	Demilitarized Zone
EAPoL	Extensible Authentication Protocol over LAN
EBCDIC	Extended Binary Coded Decimal Interchange Code
EC2	Amazon Elastic Compute Cloud
ECB	Electronic-Codebook Mode
ECC	Elliptic-Curve Cryptography
ECDH	Elliptic-Curve DiffieHellman
EFF	Electronic Frontier Foundation
ELF	Executable and Linking Format
ESP	Encapsulating Security Payload
ESSID	Extended Service Set IDentifier
FAT	File Allocation Table
FBI	Federal Bureau of Investigation (USA)
FCS	Feistel Cipher Scheme
FTP	File Transfer Protocol
GB	*Guojia Biaozhun* (National Standards, China)
GCHQ	British Government Communications Headquarters
GMK	Group Master Key
GUI	Graphical User Interface

HBC	Honest-but-Curious
HIHAT	High Interaction Honeypot Analysis Toolkit
HMAC	Keyed-Hash Message Authentication Code
HBD	Host-Based Detection
HMM	Hidden Markov Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IAT	Import Address Table
IaaS	Infrastructure-as-a-Service
IBM	International Business Machines Corporation (USA)
ICMP	Internet Control Message Protocol
ICV	Integrity Check Value
IDEA	International Data Encryption Algorithm
IDES	Intrusion Detection Expert System
IDP	Intrusion Detection Policy
IDS	Intrusion Detection System
IE	Internet Explorer
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers (USA)
IETF	The Internet Engineering Task Force
IIS	Internet Information Services
IKE	Internet Key Exchange
IM	Instant Messaging
IMAP	Internet Mail Access Protocol
IP	Internet Protocol
IPS	Intrusion Prevention System
IPsec	IP Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISAKMP	Internet Security Association and Key Management Protocol
ISO	International Standardization Organization;
	International Organization for Standardization
ISP	Internet Service Provider
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
JSP	Java Server Page
KDC	Key Distribution Center
KDP	Key Determination Protocol
KGA	Key Generation Algorithm
KSA	Key Scheduling Algorithm
LAN	Local Area Network
LFSR	Linear Feedback Shift Registers

MAC	Media Access Control
MAC	Message Authentication Code
MBSA	Microsoft Baseline Security Analyzer
MIC	Message Integrity Code
MIDI	Musical Instrument Data Interface
MKPKC	Multiple-Key Public-Key Cryptography
MPDU	MAC Protocol Data Unit
MSDU	MAC Service Data Unit
NAT	Network Address Translation
NBD	Network-Based Detection
NBS	National Bureau of Standards (USA)
NESSIE	New European Schemes for Signatures, Integrity, and Encryption
NetBIOS	Network Basic Input and Output System
NFS	Network File System;
	National Science Foundation (USA)
NGVCK	Next Generation Virus Creation Kit
NIC	Network Interface Card
NIDS	Network-based Intrusion Detection System
NIST	National Institute of Standards and Technology (USA)
NSA	National Security Agency (USA)
NTFS	New Technology File System
OAuth	Open Authentication protocol
OCB	Offset-Codebook Mode
OFB	Output-Feedback Mode
OLE	Object Linking and Embedding
ORAM	Oblivious Random Access Machine
OSI	Open System Interconnection
PAN	Personal Area Network
PaaS	Platform-as-a-Service
PAT	Port Address Translation
PDA	Personal Digital Assistant
PE	Portable Executable
PEM	Privacy-enhanced Electronic Mail
PGP	Pretty Good Privacy
PHP	Hypertext Preprocessor
PHT	Pseudo Hadamard Transform
PID	Process Identifier
PKA	Public-Key Authority
PKC	Public-Key Cryptography; Public-Key Cryptosystem
PKI	Public-Key Infrastructure
PKIX	X.509 Public-Key Infrastructure
PMK	Pairwise Master Key
POP	Post Office Protocol

POP3	Post Office Protocol version 3
POW	Proof-of-Work
PRE	Proxy Re-Encryption
PRNG	Pseudo-Random Number Generator
PTK	Pairwise Transient Key
P2P	Peer-to-Peer
RADIUS	Remote Authentication Dial-In User Service
RAM	Random Access Memory
	Random Access Machine
REST	Representational State Transfer
RSN	Robust Security Network
RSN IE	Robust Security Network Information Element
RSNA	Robust Security Network Association
SaaS	Software-as-a-Service
SA	Security Association
SAD	Security Association Database
SANS	SysAdmin, Audit, Network, and Security Institute (USA)
SAS	Security Association Selector
SCP	Secure Copy Protocol
SET	Secure Electronic Transaction
SFTP	Secure File Transfer Protocol
SHA	Secure Hash Algorithm
SHBC	Semi-Honest-but-Curious
SHBS	Single-Homed Bastion System
SIV	System Integrity Verifier
SKKE	Symmetric-Key Key Establishment
SLA	Service-Level Agreement
S/MIME	Secure/Multipurpose Internet Mail Extension
SMTP	Simple Mail Transfer Protocol
SOHO	Small Office and Home Office
SPD	Security Policy Database
SPI	Security Parameters Index; Stateful Packet Inspection
SPF	Stateful Packet Filtering
SRES	Singed Response
SSE	Searchable Symmetric Encryption
SSH	Secure Shell
SSL	Secure Sockets Layer
SSP	Secure Simple Pairing
STaaS	Storage-as-a-Service
STA	Station (wireless endpoint)
SYN	Synchronization
TCP	Transmission Control Protocol
TCPv4	Transmission Control Protocol version 4

TCPv6	Transmission Control Protocol version 6
Telnet	Teletype network
TGS	Ticket-Granting Server
TKIP	Temporal Key Integrity Protocol
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
TOS	Trusted Operating System
TPM	Trusted Platform Module
TSC	TKIP Sequence Counter
TTL	Time-to-Live value
TTP	Trusted Third Party
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
Unicode	Unification Code
VB	Visual Basic
VBS	Visual Basic Script
VoIP	Voice of IP
VPN	Virtual Private Network
WAP	Wireless Access Point
WEP	Wired Equivalent Privacy
Wi-Fi	Wireless Fidelity
WKDC	Wireless Key Distribution Center
WLAN	Wireless Local Area Network
WN	Wireless Node
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access version 2
WPAN	Wireless Personal Area Network
WPKI	Wireless Public-Key Infrastructure
WSN	Wireless Sensor Network
XML	Extensible Markup Language

Further Readings

Adams C and Farrell S (1999) Internet X.509 Public Key Infrastructure: Certificate Management Protocols. RFC 2510.

Adida B (2008) Helios: web-based open-audit voting. In Proceedings of the 17th Conference on Security Symposium, pp. 335–348.

Agrawal M, Kayal N, and Saxena N (2004) PRIMES is in P. Annals of Mathematics 160(2):781–793.

Allen J (2001) The CERT Guide to System and Network Security Practices. Addison-Wesley, Massacusetts.

Arkin O and Yarochkin F (2002) Xprobe v2.0: A "Fuzzy" Approach to Remote Active Operating System Fingerprinting, `http://www.xprobe2.org+`.

Ateniese G, Burns R, Curtmola R, Herring J, Kissner L, Peterson, Z, and Song D (2007) Provable data possession at untrusted stores. In Proceedings of the 14th ACM Conference on Computer and Communication Security. ACM, New York.

Bace R (2000) Intrusion Detection. Macmillan Technical Publishing, Indiana.

Bace R and Mell P (2001) Intrusion Detection Systems. NIST Special Publication 800–31. `http://www.csrc.nist.gov/publications/nistpubs/800-31/sp800-31.pdf`.

Baronti P, Pillai P, Chook V W, Chessa S, Gotta A, and Hu Y F (2007) Wireless sensor networks: a survey on the state of the art and the 802.15. 4 and ZigBee standards. Computer communications 30(7):1655–1695.

Barreto P and Rijmen V (2003) The WHIRLPOOL Hashing Function.

Barrett D, Silverman R, and Byrnes R (2005) SSH: The Secure Shell (The Definitive Guide). 2nd ed. O'Reilly, California.

Barta M, Bonnell J, Enfield A, Esposito D, Francis B, Harrison R, Homer A, Jakab S, Li S, Murphy S, and Ullman C (1997) Professional IE4 Programming, Wrox Press.

Bass S (2007) Top 25 Web Hoaxes and Pranks. PC World. `http://www.pcworld.com/printable/article/id,131340/printable.html+`.

Bellovin S (1999) Distributed firewalls. login: (the USENIX magazine), pp. 39–47.

Benaloh J (2006) Simple verifiable elections. In Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop. pp. 1–5.

Bertoni G, Daemen J, Peeters M, and Van Assche G (2011) The Keccak Reference.

Biham E and Shamir A (1993) A Differential Cryptoanalysis of the Data Encryption Standard. Sringer-Verlag, New York.

Blaze M, Bleumer G, and Strauss M (1998) Divertible protocols and atomic proxy cryptography. In Proceedings of Advances in Cryptology—EUROCRYPT'98. pp. 127–144.

Bluetooth Special Interest Group (2006) Simple pairing whitepaper. Version V10r00.

Bluetooth Special Interest Group (2007) Bluetooth Protocol Architecture.

Bluetooth (2007) Bluetooth Specification Version 2.1 + EDR. Volumes 0–4.

Blum L, Blum M, and Shub M (1986) A simple unpredictable pseudo-random number generator. SIAM Journal on Computing 15:364–383.

Boneh D, Di Crescenzo G, Ostrovsky R, and Persiano G (2004) Public key encryption with keyword search. In Proceedings of EUROCRYPT 2004.

Borisov N, Goldberg I, and Wagner D (2001) Intercepting mobile communications: the insecurity of 802.11. In Proceedings of the 7th Annual International Conference on Mobile Computing and Networking.

Brassard G and Bratley P (1996) Fundamentals of Algorithmics. Prentice Hall, New Jersey.

Campbell P, Calvert B, and Boswell S (2003) Security Guide to Network Security Fundamentals. 2nd ed. Thompson Course Technology, Massachusetts.

Campbell K and Wiener M (1992) Proof that DES is not a group. In Proceedings of Crypto'92, pp. 518–526. Springer-Verlag, Berlin.

Cappé O, Moulines E, and Rydén T (2005) Inference in Hidden Markov Models. Springer-Verlag.

CERT Advisory (2001) "Code Red" worm exploiting buffer overflow in IIS indexing service DLL. CA-2001-19. http://www.cert.org/advisories/CA-2001-19.html+.

CERT Advisory (2001) Nimda worm. CA-2001-26. http://www.cert.org/advisories/CA-2001-26.html+.

CERT Incident Note (2001) "Code Red II:" another worm exploiting buffer overflow in IIS indexing service DLL. IN-2001-09. http://www.cert.org/incident_notes/IN-2001-09.html+.

CERT Incident Note (2003) W32/Sobig.F worm. IN-2003-03. http://www.cert.org/incident_notes/IN-2003-03.html+.

Chai Q and Gong G (2012) Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In Proceedings of the IEEE International Conference on Communications (ICC'12).

Chandra P (2005) Bulletproof Wireless Security: GSM, UMTS, 802.11, and Ad Hoc Security. Elsevier, Paris.

Chase M and Kamara S (2010) Structured encryption and controlled disclosure. In Proceedings of Advances in Cryptology-ASIACRYPT 2010, pp. 577–594. Springer-Verlag, Berlin Heidelberg.

Chaum D (1983) Blind signatures for untraceable payments. In Proceedings of CRYPTO'82, pp. 199–203. Plenum Press, New York.

Chaum D and van Antwerpen H (1989) Undeniable signatures. In Proceedings of Advances in Cryptology (CRYPTO'89), pp. 212–216.

Chaum D, Fiat A, and Naor M (1990) Untraceable electronic cash. In Proceedings of CRYPTO'88, Lecture Notes in Computer Science, vol. 403, pp. 319–327. Springer-Verlag, Berlin.

Chaum D and Pedersen T P (1992) Wallet database with observers. In Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology. pp. 82–105.

Cheswick W, Bellovin S, and Rubin A (2003) Firewalls and Internet Security, Repelling the Wily Hacker. 2nd ed. Addison-Wesley, Massachusetts.

Ciampa M (2005) Security Guide to Network Security Fundamentals. 2nd ed. Thompson Course technology, Massachusetts.

CNSS (2003) National Policy on the Use of the Advanced Encryption Standard (AES) to Protect Security Systems and National Security Information. CNSS Policy No. 15 Fact Sheet No. 1. http://www.cnss.gov/Assets/pdf/cnssp_15_fs.pdf+.

Cohen F (1994) A Short Course on Computer Viruses. John Wiley & Sons, New Jersey.

Cole E (2002) Hackers Beware. New Riders, Indiana.

Comer D (2006) Network Systems Design using Network Processors: Intel IXP 2xxx version. Prentice Hall, New Jersey.

Conficker Working Group (2011) Lessons Learned.

Coppersmith D (1994) The Data Encryption Standard (DES) and its strength against attacks. IBM Journal of Research and Development 38:243–250.

Courtois N and Pieprzyk J (2002) Cryptanalysis of block ciphers with overdefined systems of equations. In Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT), Lecture Notes in Computer Science, vol. 2501, pp. 267–287. Springer-Verlag, Berlin.

Crume J (2000) Inside Internet Security: What Hackers Don't Want You to Know. Addison-Wesley, New Jersey.

Daemen J and Rijmen V (1999) AES Proposal: The Rijndael Block Cipher. `http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf+`.

Dawson E and Nielsen L (1996) Automated cryptanalysis of XOR plaintext strings. Cryptologia 2:165–181.

Denning D (1987) An intrusion detection model. IEEE Transactions on Software Engineering 13(2):222–232.

Desmdt Y and Frankel Y (1998) Threshold cryptosystems. In Proceedings of AUSCRYPT '92. pp. 1–14.

Deswarte Y, Quisquater J-J, and Saidane A (2003) Remote integrity checking. In Proceedings of Conference on Integrity and Internal Control in Information Systems.

Diffie W and Hellman M (1976) New directions in cryptography. IEEE Transactions in Information Theory 22:644–654.

Dingledine R, Matthewson N, and Syverson P (2004) Tor: the second-generation onion router. In Proceedings of the 13th Conference on USENIX Security Symposium.

Doraswamy N and Harkings D (1999) IPSec the New Security Standard for the Internet, Intranet, and Virtual Private Networks. Prentice Hall, New Jersey.

Dornan A (2002) The Essential Guide to Wireless Communications Applications. Prentice-Hall, New Jersey.

Dwork C and Naor M (1992) Pricing via processing or combatting Junk Mail. Advances in Cryptology—CRYPTO' 92, vol. 740:139–147.

Easttom C (2006) Network Defense and Countermeasures: Principles and Practices. Pearson Prentice Hall, New Jersey.

Edney J and Arbaugh W (2004) Real 802.11 Security: Wi-Fi Protected Access and 802.11i. Addison-Wesley, Boston.

Electronic Frontier Foundation (1999) Distributed.Net and EFF DES Cracker put the final nail into the Data Encryption Standard's coffin. `http://www.eff.org/Privacy/Crypto/Crypto_misc/DESCracker+`.

Elgamal T (1985) A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4):469–472.

Even S, Goldreich O, and Lempel A (1985) A randomized protocol for signing contracts. Communications of the ACM 28(6):637–647.

Filho D L G and Baretto P S L M (2006) Demonstarting Data Possession and Uncheatable Data Transfer. In IACR ePrint archive, 2006. Report 2006/150, `http://eprint.iacr.org/2006/150`.

FIPS 171 (1995) American National Standard Financial Institution Key Management (Wholesale), National Institute of Standards and Technology.

FIPS 180-1 (1995) Secure Hash Standard. Federal Information Processing Standards Publication 180-1, National Institute of Standards and Technology.

FIPS 46-3 (1999) Data Encryption Standard (DES). Federal Information Processing Standards Publication 46-3 (Reaffirmed), National Institute of Standards and Technology.

FIPS 186-2 (2000) Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2, National Institute of Standards and Technology.

FIPS FIPS-197 (2001) Announcing the Advanced Encryption Standard. FIPS Special Publication 197, National Institute of Standards and Technology.

FIPS 180-2 (2002) Secure Hash Standards. Federal Information Processing Standards Publication 180-2, National Institute of Standards and Technology.

FIPS FIPS-198 (2002) The keyed-hash message authentication code (HMAC). FIPS Special Publication 198, National Institute of Standards and Technology.

FIPS FIPS-202 (2014) SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Federal Information Processing Standards Publication 202, National Institute of Standards and Technology.

Fluhrer S, Mantin I, and Shamir A (2001) Weaknesses in the key scheduling algorithm of RC4. In Proceedings of the 8th Annual International Workshop on Selected Areas in Cryptography, Lecture Notes in Computer Science, vol. 2259, pp. 1–24. Springer-Verlag, London.

Forouzan B (2008) Cryptography and Network Security. McGraw-Hill, New York.

Fredkin E (1960) Trie Memory. Communications of the ACM 3(9):490–499.

Gerkis A and Purcell J (2006) A Survey of Wireless Mesh Networking Security Technology and Threats. SANS Institute.

Goldreich O, Micali S, and Wigderson A (1987) How to play ANY mental game. In Proceedings of the 19th Annual ACM Conference on Theory of Computing, pp. 218–229.

Goldreich O and Ostrovsky R (1996) Software protection and simulation on oblivious RAMs. Journal of the ACM 43(3):431–473.

Hammer-Lahav E (2010) The OAuth 1.0 Protocol. RFC 5849.

Hardt D (2012) The OAuth 2.0 Authroization Framework. RFC 6749.

Harkens D and Carrel D (1998) The Internet Key Exchange (IKE). RFC 2409.

Harley D, Slade R, and Gattiker U (2001) Viruses Revealed. McGraw-Hill, New York.

Hayre J and Kelath J (2006) AJAX Security Basics. `http://www.securityfocus.com/infocus/1868/1+`.

He C and Mitchell J (2004) Analysis of the 802.11i 4-way handshake. In Proceedings of the 3rd ACM Workshop on Wireless Security, pp. 43–50. ACM Press, New York.

Housley R, Ford W, Polk W, and Solo D (2002) Internet X.509 Public Key Infrastructure: Certificate and CRL Profile. RFC 3280.

Howlett T (2005) Open Source Security Tools: A Practical Guide to Security Applications. Prentice-Hall, New Jersey.

Hua L-K (1987) Introduction to Number Theory. Translated from Chinese by P. Shiu. Springer-Verlag, Berlin.

ISO 11889-1 (2009) Information Technology—Trusted Platform Module Part 1: Overview, International Standards Organization.

ISO 11889-2 (2009) Information Technology—Trusted Platform Module Part 2: Design Principles, International Standards Organization.

Javitz H and Valdes A (1991) The SRI IDES statistical anomaly detector. In Proceedings of the IEEE Symposium in Security and Privacy, IEEE Computer Society Press, pp. 316–326.

Kamara S and Lauter K (2010) Cryptographic cloud storage. In Proceedings of Financial Cryptography and Data Security, pp. 136–149.

Karro J and Wang J (1998) Protecting web servers from security holes in server-side includes. In Proceedings of Annual Computer Security Application Conference (ACSAC'98), pp. 103–111. IEEE Computer Society Press, Washington, DC.

Karygiannis T and Owens L (2002) Wireless Network Security: 802.11, Bluetooth, and Handheld Devices. National Institute of Standards and Technology, Special Publication 800-48.

Kissel Z and Wang J (2014) A note on verifiable privacy-preserving Tries. In Proceedings of the 7th IEEE International Conference on Cloud Computing.

Knightley P (1986) The Second Oldest Profession, Spies and Spying in the Twentieth Century. Penguin Books, New York.

Knuth D (1998) The Art of Computer Programming, Seminumerical Algorithms, vol. 2. 3rd ed. Addison-Wesley, Massachusetts.

Koblas D and Koblas M (1992) SOCKS. In Proceedings of the 3rd Usenix Security Symposium. pp. 77–83.

Kobliz N (1998) Algebraic Aspects of Cryptography. Springer-Verlag, Berlin.

LAN/MAN Committee (2004) IEEE Standard for Information technology: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 6: Medium Access Control (MAC)Security Enhancements.

Linn, J (1987) Privacy Enhancement for Internet Electronic Mail: Part I: Message Encipherment and Authentication Procedures. RFC 989.

Linn, J (1993) Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. RFC 1421.

Massey J (1993) SAFER K-64: a byte-oriented block-ciphering algorithm. In Proceedings of Fast Software Encryption, pp. 1–17.

Massey J, Khachatrian G, and Kuregian M (1998) SAFER+. In Proceedings of the 1st Advanced Encryption Standard Candidate Conference. National Institute of Standards and Technology.

McKean C (2001) Peer-to-Peer Security and Intel's Peer-to-Peer Trusted Library. SANS Security Essentials, GSEC Practical Assignment, Version 1.2e.

Mell P and Grance T (2011) The NIST Definition of Cloud Computing. NIST Special Publication 800-145.

Merkle R (1979) Secrecy, Authentication, and Public Key Systems. PhD thesis, Standford University.

Miller, G (1976) Riemann's hypothesis and tests for primality. Journal of Computer and System Sciences 13(3):300–317.

Mirkovic J and Relher P (2004) A taxonomy of DDoS attack and DDoS defense mechanisms. ACM SIGCOMM Computer Communications Review 34(2):39–53.

Moore T, Clulow J, Anderson R, and Nagaraja S (2007) New strategies for revocation in Ad-Hoc networks. In Proceedings of the 4th European Workshop on Security and Privacy in Ad hoc and Sensor Networks, Lecture Notes in Computer Science, vol. 4572, pp. 232–246. Springer-Verlag, Berlin.

Nakamoto S (2009) Bitcoin: A Peer-to-Peer Electronic Cashs System. `https://bitcoin.org/bitcoin.pdf+`.

Neuman B-C and Ts'o T (1994) Kerberos: an authentication service for computer networks. IEEE Communications 32(9):33–38.

Northcutt S (1999) Network Intrusion Detection, An Analysit's Handmonograph. New Riders, Indiana.

Oppliger R (1999) Security Technologies for the World Wide Web. Artech House, Massachusetts.

Ostrovsky R (1992) Software Protection and Simulation on Oblivious RAMs. PhD thesis, Massachusetts Institute of Technology.

PC Magazine (2007) Ten most common passwords. `http://www.pcmag.com+verb+/article2/0,1759,2113976,00.asp+`.

Pedersen T P (1991) A threshold cryptosystem without a trusted party. In Proceedings of EUROCRYPT '91. pp. 522–526.

Peterson L and Davie B (2006) Computer Networks A Systems Approach. 3rd ed. Elsevier, Paris.

Pfleeger C and Pfleeger S (2006) Security in Computing. 4th ed. Prentice-Hall, New Jersey.

Pietrek M (1994) Peering Inside the PE: A Tour of the Win32 Portable Executable File Format. MSDN Magazine. `http://msdn2.microsoft.com/en-us/library/ms809762.aspx+`.

Pietrek M (2002) An In-Depth Look into the Win32 Portable Executable File Format. MSDN Magazine. Part I: `http://msdn.microsoft.com/msdnmag/issues/02/02/PE/default.aspx+`. Part II: `http://msdn.microsoft.com/msdnmag/issues/02/03/PE2/default.aspx+`.

Proctor P (2001) The Practical Intrusion Detection Handmonograph. Prentice-Hall, New Jersey.

Provos N (2004) A virtual honeypot framework. In Proceedings of the 13th USENIX Security Symposium. pp. 1–14.

Rabin, M (1980) Probabilistic algorithm for testing primality. Journal of Number Theory 12(1):128–138.

Ramachandran V and Ahmad M (2007) Cafe latte with a free topping of cracked WEP: retrieving WEP keys from road-warriors. In Proceedings of ToorCon.

Ranum M (1992) A network firewall. In Proceedings of the 1st World Conference on Systems Administration and Security.

Rescorla E (2001) SSL and TLS: Designing and Building Secure Systems. Addison-Wesley, Massachusetts.

Rivest R-L (1992) The RC4 encryption algorithm. RSA Data Security.

Rivest R-L (1995) The RC5 encryption algorithm. Dr. Dobb's Journal 20:146–148.

Rivest R-L, Shamir A, and Adleman L-M (1978) A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21:120–126.

Rogaway P, Bellare M, and Black J (2003) OCB: a block-cipher mode of operation for efficient authenticated encryption. ACM Transactions on Information and System Security 6(3):365–403.

Rubin A (2001) White-Hat Security Arsenal, Tackling the Threats. Addison-Wesley, Massachusetts.

Salomaa A (1990) Public-Key Cryptography. Springer-Verlag, Berlin.

Scarfone K, Souppaya M, and Hoffma P (2011) Guide to Security for Full Virtualization Technologies. NIST Special Publication 800-125.

Schneier B (1996) Applied Cryptography. 2nd ed. John Wiley & Sons, New York.

Schneier B (2000) Secrets and Lies, Digital Security in a Networked World. John Wiley & Sons, New York.

Seaminatha T and Elden C (2003) Wireless Security and Privacy. Addison-Wesley, Massachusetts.

Shaked Y and Wool A (2005) Cracking the bluetooth PIN. In Proceedings of the 3rd USENIX/ACM Conference Mobile Systems, Applications, and Services (MobiSys), pp. 39–50.

Shamir A (1979) How to share a secret. Communications of the ACM 22(11):612–613.

Shor P-W (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing 26:1481–1509.

Skoudis E (2002) Counter Hack, A Step-by-Step Guide to Computer Attacks and Effective Defenses. Prentice-Hall, New Jersey.

Song D X, Wagner D, and Perrig A (2000) Practical techniques for searches on encrypted data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, pp. 44–55.

Stallings W (2006) Cryptography and Network Security: Principles and Practice. 4th ed. Prentice-Hall, New Jersey.

Steiner J, Neuman C, and Schiller J (1988) Kerberos: an authentication service for open network systems (Version 4). In Proceedings of the Winter 1988 Usenix Conference.

Suehring S and Ziegler R (2006) Linux Firewalls. 3rd ed. Novell Press, Indiana.

Szor P (2005) The Art of Computer Virus Research and Defense. Addison-Wesley, New Jersey.

Tang Y, Gu D, Ding N, and Lu H (2012) Phrase search over encrypted data with symmetric encryption scheme. In Proceedings of the 32nd International Conference on Distributed Computing Systems Workshops, pp. 471–480.

Thomas S (2000) SSL and TLS Essentials Securing the Web. John Wiley & Sons, New York.

Tibbs R and Oakes E (2006) Firewalls and VPNs Principles and Practices. Pearson Prentice Hall, New Jersey.

Trappe W and Washington L (2006) Introduction to Cryptography with Coding Theory. 2nd ed. Prentice-Hall, New Jersey.

Viega J and McGraw G (2002) Building Secure Software. Addison-Wesley, Massachusetts.

Voice of America (1999) Navajo Code Talkers. `http://www.voanews.com/++special english/archive/2002-02/a-2002-02-01-26-1.cfm+`.

Wack J, Cutler K, and Pole J (2002) Guidelines on Firewalls and Firewall Policy. NIST Special Publication SP 800-41.

Walker J (2002) 802.11 security series part II: the Temporal Key Integrity protocol (TKIP). Intel Cooperation. `http://cache-www.intel.com/cd/00/00/01/77/17769_80211_part2 .pdf+`.

Wang X, Yin Y, and Yu H (2005) Finding collisions in the full SHA1. In Proceedings of CRYPTO'05, Lecture Notes in Computer Science, vol. 3621, pp. 17–36. Springer-Verlag, Berlin.

WEPCrack an 802.11 key breaker. `http://wepcrack.sourceforge.net`.

Whiteman M-E and Mattord H-J (2005) Principles of Information Security. 2nd ed. Thomson Course Technology, Massachusetts.

Yao A (1982) Protocols for secure computations. In Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science (FOCS'82), pp. 160–164.

Ylönen T (2006) The Secure Shell (SSH) Protocol Architecture. RFC 4251.

Ylönen T (2006) The Secure Shell (SSH) Authentication Protocol. RFC 4252.

ZigBee Alliance (2008) ZigBee Specification. `http://zigbee.org/Specifications.aspx+`.

Ziv J and Lempel A (1977) A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23:337–343.

Index

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley's ebook EULA.