Cryptography and Network Security

Principles and Practice

Fifth Edition

WILLIAM STALLINGS

NETWORK SECURITY ESSENTIALS, FOURTH EDITION

A tutorial and survey on network security technology. The book covers important network security tools and applications, including S/MIME, IP Security, Kerberos, SSL/TLS, SET, and X509v3. In addition, methods for countering hackers and viruses are explored.

COMPUTER SECURITY (with Lawrie Brown)

A comprehensive treatment of computer security technology, including algorithms, protocols, and applications. Covers cryptography, authentication, access control, database security, intrusion detection and prevention, malicious software, denial of service, firewalls, software security, physical security, human factors, auditing, legal and ethical aspects, and trusted systems. **Received the 2008 Text and Academic Authors Association (TAA) award for the best Computer Science and Engineering Textbook of the year.** ISBN 0-13-600424-5

WIRELESS COMMUNICATIONS AND NETWORKS, Second Edition

A comprehensive state-of-the art survey. Covers fundamental wireless communications topics, including antennas and propagation, signal encoding techniques, spread spectrum, and error correction techniques. Examines satellite, cellular, wireless local loop networks and wireless LANs, including Bluetooth and 802.11. Covers Mobile IP and WAP. ISBN 0-13-191835-4

HIGH-SPEED NETWORKS AND INTERNETS, SECOND EDITION

A state-of-the art survey of high-speed networks. Topics covered include TCP congestion control, ATM traffic management, Internet traffic management, differentiated and integrated services, Internet routing protocols and multicast routing protocols, resource reservation and RSVP, and lossless and lossy compression. Examines important topic of self-similar data traffic.
ISBN 0-13-03221-0

Purchasing this textbook now grants the reader six months of access to this online material. See the access card bound into the front of this book for details.

INSTRUCTIONAL SUPPORT MATERIALS

To support instructors, the following materials are provided:

- **Solutions Manual:** Solutions to end-of-chapter Review Questions and Problems.
- **Projects Manual:** Suggested project assignments for all of the project categories listed below.
- **PowerPoint Slides:** A set of slides covering all chapters, suitable for use in lecturing.
- **PDF Files:** Reproductions of all figures and tables from the book.
- **Test Bank:** A chapter-by-chapter set of questions.

All of these support materials are available at the Instructor Resource Center (IRC) for this textbook, which can be reached via pearsonhighered.com/stallings or by clicking on the button labeled "Book Info and More Instructor Resources" at this book's Web Site WilliamStallings.com/Crypto/Crypto5e.html. To gain access to the IRC, please contact your local Prentice Hall sales representative via pearsonhighered.com/educator/replocator/requestSalesRepPage.html or call Prentice Hall Faculty Services at 1-800-526-0485.

INTERNET SERVICES FOR INSTRUCTORS AND STUDENTS

There is a Web site for this book that provides support for students and instructors. The site includes links to other relevant sites, transparency masters of figures and tables in the book in PDF (Adobe Acrobat) format, and PowerPoint slides. The Web page is at **WilliamStallings.com/Crypto/Crypto5e.html**. For more information, see Chapter 0.

New to this edition is a set of homework problems with solutions available at this Web site. Students can enhance their understanding of the material by working out the solutions to these problems and then checking their answers.

An Internet mailing list has been set up so that instructors using this book can exchange information, suggestions, and questions with each other and with the author. As soon as typos or other errors are discovered, an errata list for this book will be available at WilliamStallings.com. In addition, the Computer Science Student Resource site at **WilliamStallings.com/StudentSupport.html** provides documents, information, and useful links for computer science students and professionals.

PROJECTS AND OTHER STUDENT EXERCISES

For many instructors, an important component of a cryptography or security course is a project or set of projects by which the student gets hands-on experience to reinforce concepts from the text. This book provides an unparalleled degree of support, including a projects component in the course. The IRC not only includes guidance on how to assign and structure

- **Chapter 4 — Basic Concepts In Number Theory And Finite Fields:** Euclidean and extended Euclidean algorithms, polynomial arithmetic, and GF(24).
- **Chapter 5 — Advanced Encryption Standard:** Exercise based on SAES.
- **Chapter 6 — Pseudorandom Number Generation And Stream Ciphers:** Blum Blum Shub, linear congruential generator, and ANSI X9.17 PRNG.
- **Chapter 8 — Number Theory:** Euler's Totient function, Miller Rabin, factoring, modular exponentiation, discrete logarithm, and Chinese remainder theorem.
- **Chapter 9 — Public-Key Cryptography And RSA:** RSA encrypt/decrypt and signing.
- **Chapter 10 — Other Public-Key Cryptosystems:** Diffie-Hellman, elliptic curve
- **Chapter 11 — Cryptographic Hash Functions:** Number-theoretic hash function.
- **Chapter 13 — Digital Signatures:** DSA.

WHAT'S NEW IN THE FIFTH EDITION

The changes for this new edition of *Cryptography and Network Security* are more substantial and comprehensive than those for any previous revision.

In the three years since the fourth edition of this book was published, the field has seen continued innovations and improvements. In this new edition, I try to capture these changes while maintaining a broad and comprehensive coverage of the entire field. To begin this process of revision, the fourth edition was extensively reviewed by a number of professors who teach the subject. In addition, a number of professionals working in the field reviewed individual chapters. The result is that, in many places, the narrative has been clarified and tightened, and illustrations have been improved. Also, a large number of new "field-tested" problems have been added.

One obvious change to the book is a revision in the organization, which makes for a clearer presentation of related topics. There is a new Part Three, which pulls together all of the material on cryptographic algorithms for data integrity, including cryptographic hash functions, message authentication codes, and digital signatures. The material on key management and exchange, previously distributed in several places in the book, is now organized in a single chapter, as is the material on user authentication.

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

- **Euclidean and extended Euclidean algorithms (revised):** These algorithms are important for numerous cryptographic functions and algorithms. The material on the Euclidean and extended Euclidean algorithms for integers and for polynomials has been completely rewritten to provide a clearer and more systematic treatment.
- **Advanced Encryption Standard (revised):** AES has emerged as the dominant symmetric encryption algorithm, used in a wide variety of applications. Accordingly, this edition has dramatically expanded the resources for learning about and understanding this important standard. The chapter on AES has been revised and expanded, with additional illustrations and a detailed example, to clarify the presentation. Examples and assignments using Sage have been added. And the book now includes an AES cryptography lab, which enables the student to gain hands-on experience with AES cipher internals and modes of use. The lab makes use of an AES calculator applet, available at this book's Web site, that can encrypt or decrypt test data values using the AES block cipher.

With each new edition it is a struggle to maintain a reasonable page count while adding new material. In part, this objective is realized by eliminating obsolete material and tightening the narrative. For this edition, chapters and appendices that are of less general interest have been moved online as individual PDF files. This has allowed an expansion of material without the corresponding increase in size and price.

ACKNOWLEDGEMENTS

This new edition has benefited from review by a number of people who gave generously of their time and expertise. The following people reviewed all or a large part of the manuscript: Marius Zimand (Towson State University), Shambhu Upadhyaya (University of Buffalo), Nan Zhang (George Washington University), Dongwan Shin (New Mexico Tech), Michael Kain (Drexel University), William Bard (University of Texas), David Arnold (Baylor University), Edward Allen (Wake Forest University), Michael Goodrich (UC-Irvine), Xunhua Wang (James Madison University), Xianyang Li (Illinois Institute of Technology), and Paul Jenkins (Brigham Young University).

Thanks also to the many people who provided detailed technical reviews of one or more chapters: Martin Bealby, Martin Hlavac (Department of Algebra, Charles University in Prague, Czech Republic), Martin Rublik (BSP Consulting and University of Economics in Bratislava), Rafael Lara (President of Venezuela's Association for Information Security and Cryptography Research), Amitabh Saxena, and Michael Spratte (Hewlett-Packard Company). I would especially like to thank Nikhil Bhargava (IIT Delhi) for providing detailed reviews of various chapters of the book.

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

Nikhil Bhargava (IIT Delhi) developed the set of online homework problems and solutions. Dan Shumow of Microsoft and the University of Washington developed all of the Sage examples and assignments in Appendices B and C. Professor Sreekanth Malladi of Dakota State University developed the hacking exercises. Lawrie Brown of the Australian Defence Force Academy provided the AES/DES block cipher projects and the security assessment assignments.

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

Finally, I would like to thank the many people responsible for the publication of the book, all of whom did their usual excellent job. This includes my editor Tracy Dunkelberger, her assistant Melinda Hagerty, and production manager Rose Kernan. Also, Jake Warde of Warde Publishers managed the reviews.

With all this assistance, little remains for which I can take full credit. However, I am proud to say that, with no help whatsoever, I selected all of the quotations.

This page intentionally left blank

CHAPTER 0

READER'S GUIDE

Other Web Sites

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

Newsgroups and Forums

A number of USENET newsgroups are devoted to some aspect of cryptography or network security. As with virtually all USENET groups, there is a high noise-to-signal ratio, but it is worth experimenting to see if any meet your needs. The most relevant are as follows:

- **sci.crypt.research:** The best group to follow. This is a moderated newsgroup that deals with research topics; postings must have some relationship to the technical aspects of cryptology.
- **sci.crypt:** A general discussion of cryptology and related topics.
- **sci.crypt.random-numbers:** A discussion of cryptographic-strength random number generators.
- **alt.security:** A general discussion of security topics.
- **comp.security.misc:** A general discussion of computer security topics.
- **comp.security.firewalls:** A discussion of firewall products and technology.
- **comp.security.announce:** News and announcements from CERT.
- **comp.risks:** A discussion of risks to the public from computers and users.
- **comp.virus:** A moderated discussion of computer viruses.

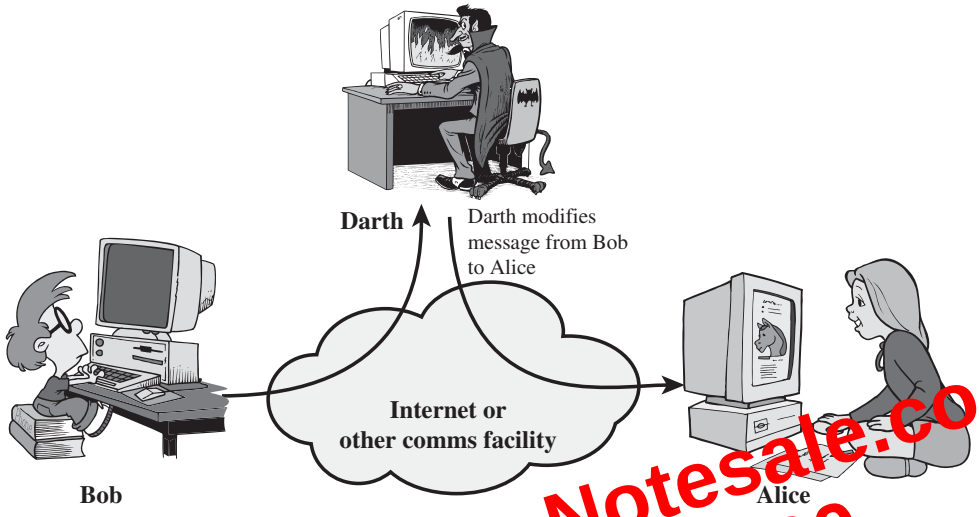
In addition, there are a number of forums dealing with cryptography available on the Internet. Among the most worthwhile are

- **Security and Cryptography forum:** Sponsored by DevShed. Discusses issues related to coding, server applications, network protection, data protection, firewalls, ciphers, and the like.
- **Cryptography forum:** On Topix. Fairly good focus on technical issues.
- **Security forums:** On WindowsSecurity.com. Broad range of forums, including cryptographic theory, cryptographic software, firewalls, and malware.

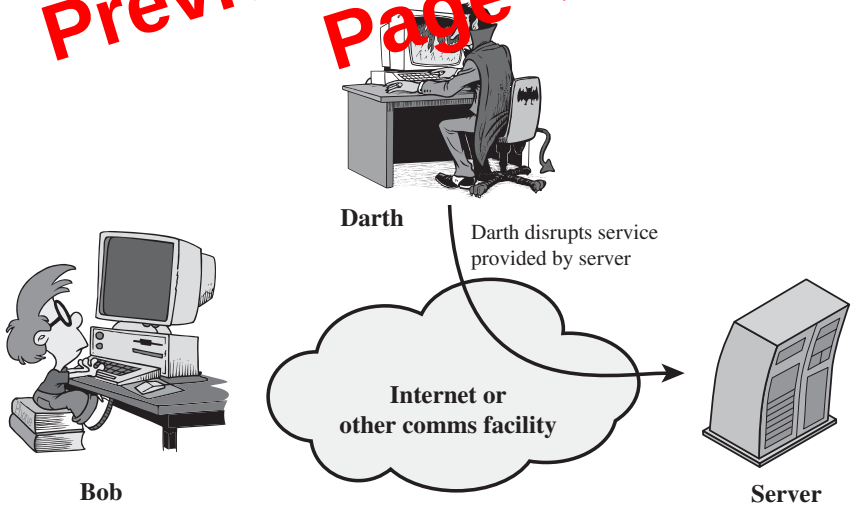
Links to these forums are provided at this book's Web site.

0.4 STANDARDS

Many of the security techniques and applications described in this book have been specified as standards. Additionally, standards have been developed to cover management practices and the overall architecture of security mechanisms



(c) Modification of messages



(d) Denial of service

Figure 1.3 Active attacks

1.4 SECURITY SERVICES

X.800 defines a security service as a service that is provided by a protocol layer of communicating open systems and that ensures adequate security of the systems or of data transfers. Perhaps a clearer definition is found in RFC 2828, which provides the following definition: a processing or communication service that is provided by

a system to give a specific kind of protection to system resources; security services implement security policies and are implemented by security mechanisms.

X.800 divides these services into five categories and fourteen specific services (Table 1.2). We look at each category in turn.[5]

Table 1.2 Security Services (X.800)

AUTHENTICATION

The assurance that the communicating entity is the one that it claims to be.

Peer Entity Authentication
Used in association with a logical connection to provide confidence in the identity of the entities connected.

Data-Origin Authentication
In a connectionless transfer, provides assurance that the source of received data is as claimed.

ACCESS CONTROL

The prevention of unauthorized use of a resource (i.e., this service controls who can have access to a resource, under what conditions access can occur, and what those accessing the resource are allowed to do).

DATA CONFIDENTIALITY

The protection of data from unauthorized disclosure.

Connection Confidentiality
The protection of all user data on a connection.

Connectionless Confidentiality
The protection of all user data in a single data block

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

Traffic-Flow Confidentiality
The protection of the information that might be derived from observation of traffic flows.

DATA INTEGRITY

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

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

Connection Integrity without Recovery
As above, but provides only detection without recovery.

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

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

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

NONREPUDIATION

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

Nonrepudiation, Origin
Proof that the message was sent by the specified party.

Nonrepudiation, Destination
Proof that the message was received by the specified party.

[5]There is no universal agreement about many of the terms used in the security literature. For example, the term *integrity* is sometimes used to refer to all aspects of information security. The term *authentication* is sometimes used to refer both to verification of identity and to the various functions listed under integrity in this chapter. Our usage here agrees with both X.800 and RFC 2828.

Table 1.4 Relationship Between Security Services and Mechanisms

	Mechanism							
Service	**Encipherment**	**Digital Signature**	**Access Control**	**Data Integrity**	**Authentication Exchange**	**Traffic Padding**	**Routing Control**	**Notarization**
Peer Entity Authentication	Y	Y			Y			
Data Origin Authentication	Y	Y						
Access Control			Y					
Confidentiality	Y						Y	
Traffic Flow Confidentiality	Y					Y	Y	
Data Integrity	Y	Y		Y				
Nonrepudiation		Y		Y				Y
Availability				Y	Y			

message without any knowledge of the enciphering details fall into the area of **cryptanalysis**. Cryptanalysis is what the layperson calls "breaking the code." The areas of cryptography and cryptanalysis together are called **cryptology**.

2.1 SYMMETRIC CIPHER MODEL

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

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

There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The

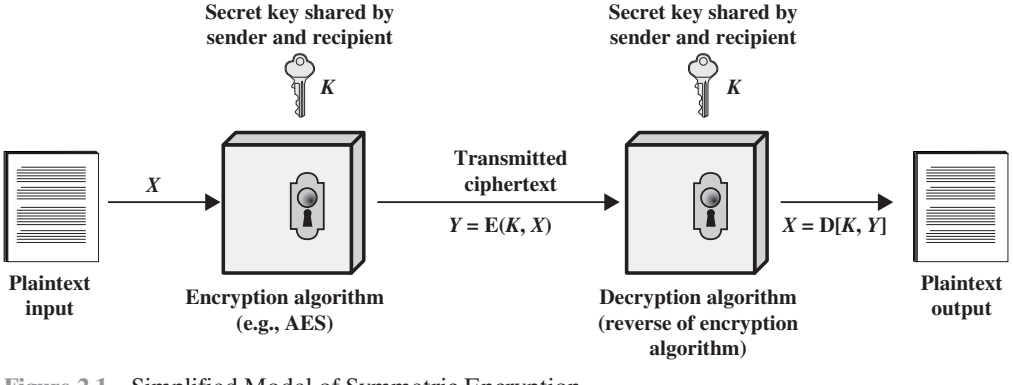


Figure 2.1 Simplified Model of Symmetric Encryption

The ciphertext-only attack is the easiest to defend against because the opponent has the least amount of information to work with. In many cases, however, the analyst has more information. The analyst may be able to capture one or more plaintext messages as well as their encryptions. Or the analyst may know that certain plaintext patterns will appear in a message. For example, a file that is encoded in the Postscript format always begins with the same pattern, or there may be a standardized header or banner to an electronic funds transfer message, and so on. All these are examples of *known plaintext*. With this knowledge, the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed.

Closely related to the known-plaintext attack is what might be referred to as a probable-word attack. If the opponent is working with the encryption of some general prose message, he or she may have little knowledge of what is in the message. However, if the opponent is after some very specific information, then parts of the message may be known. For example, if an entire accounting file is being transmitted, the opponent may know the placement of certain keywords in the header of the file. As another example, the source code for a program developed by Corporation X might include a copyright statement in some standardized position.

If the analyst is able somehow to get the source system to insert into the system a message chosen by the analyst, then a *chosen-plaintext* attack is possible. An example of this strategy is differential cryptanalysis, explored in Chapter 3. In general, if the analyst is able to choose the messages to encrypt, the analyst may deliberately pick patterns that can be expected to reveal the structure of the key.

Table 2.1 lists two other types of attack: chosen ciphertext and chosen text. These are less commonly employed as cryptanalytic techniques but are nevertheless possible avenues of attack.

Only relatively weak algorithms fail to withstand a ciphertext-only attack. Generally, an encryption algorithm is designed to withstand a known-plaintext attack.

Two more definitions are worthy of note. An encryption scheme is **unconditionally secure** if the ciphertext generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext is available. That is, no matter how much time an opponent has, it is impossible for him or her to decrypt the ciphertext simply because the required information is not there. With the exception of a scheme known as the one-time pad (described later in this chapter), there is no encryption algorithm that is unconditionally secure. Therefore, all that the users of an encryption algorithm can strive for is an algorithm that meets one or both of the following criteria:

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

An encryption scheme is said to be **computationally secure** if either of the foregoing two criteria are met. Unfortunately, it is very difficult to estimate the amount of effort required to cryptanalyze ciphertext successfully.

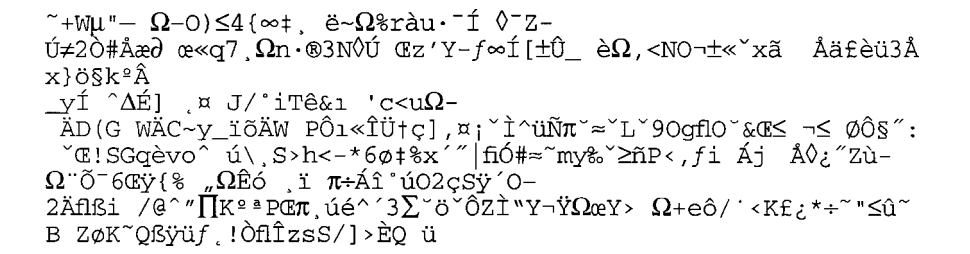


Figure 2.4 Sample of Compressed Text

The third characteristic is also significant. If the language of the plaintext is unknown, then plaintext output may not be recognizable. Furthermore, the input may be abbreviated or compressed in some fashion, making recognition difficult. For example, Figure 2.4 shows a portion of a text file compressed using an algorithm called ZIP. If this file is then encrypted with a simple substitution cipher (expanded to include more than just 26 alphabetic characters), then the plaintext may not be recognized when it is uncovered in the brute-force cryptanalysis.

Monoalphabetic Ciphers

With only 25 possible keys, the Caesar cipher is far from secure. A dramatic increase in the key space can be achieved by allowing an arbitrary substitution. Before proceeding, we define the term *permutation*. A **permutation** of a finite set of elements S is an ordered sequence of all the elements of S, with each element appearing exactly once. For example, if $S = \{a, b, c\}$, there are six permutations of S:

abc, acb, bac, bca, cab, cba

In general, there are $n!$ permutations of a set of n elements, because the first element can be chosen in one of n ways, the second in $n - 1$ ways, the third in $n - 2$ ways, and so on.

Recall the assignment for the Caesar cipher:

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

If, instead, the "cipher" line can be any permutation of the 26 alphabetic characters, then there are 26! or greater than 4×10^{26} possible keys. This is 10 orders of magnitude greater than the key space for DES and would seem to eliminate brute-force techniques for cryptanalysis. Such an approach is referred to as a **monoalphabetic substitution cipher**, because a single cipher alphabet (mapping from plain alphabet to cipher alphabet) is used per message.

There is, however, another line of attack. If the cryptanalyst knows the nature of the plaintext (e.g., noncompressed English text), then the analyst can exploit the regularities of the language. To see how such a cryptanalysis might

Friedrich Gauss believed that he had devised an unbreakable cipher using homophones. However, even with homophones, each element of plaintext affects only one element of ciphertext, and multiple-letter patterns (e.g., digram frequencies) still survive in the ciphertext, making cryptanalysis relatively straightforward.

Two principal methods are used in substitution ciphers to lessen the extent to which the structure of the plaintext survives in the ciphertext: One approach is to encrypt multiple letters of plaintext, and the other is to use multiple cipher alphabets. We briefly examine each.

Playfair Cipher

The best-known multiple-letter encryption cipher is the Playfair, which treats digrams in the plaintext as single units and translates these units into ciphertext digrams.[3]

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

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

In this case, the keyword is *monarchy*. The matrix is constructed by filling in the letters of the keyword (minus duplicates) from left to right and from top to bottom, and then filling in the remainder of the matrix with the remaining letters in alphabetic order. The letters I and J count as one letter. Plaintext is encrypted two letters at a time, according to the following rules:

1. Repeating plaintext letters that are in the same pair are separated with a filler letter, such as x, so that balloon would be treated as ba lx lo on.
2. Two plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row circularly following the last. For example, ar is encrypted as RM.
3. Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, mu is encrypted as CM.
4. Otherwise, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter. Thus, hs becomes BP and ea becomes IM (or JM, as the encipherer wishes).

The Playfair cipher is a great advance over simple monoalphabetic ciphers. For one thing, whereas there are only 26 letters, there are 26 × 26 = 676 digrams, so

[3]This cipher was actually invented by British scientist Sir Charles Wheatstone in 1854, but it bears the name of his friend Baron Playfair of St. Andrews, who championed the cipher at the British foreign office.
[4]The book provides an absorbing account of a probable-word attack.

that identification of individual digrams is more difficult. Furthermore, the relative frequencies of individual letters exhibit a much greater range than that of digrams, making frequency analysis much more difficult. For these reasons, the Playfair cipher was for a long time considered unbreakable. It was used as the standard field system by the British Army in World War I and still enjoyed considerable use by the U.S. Army and other Allied forces during World War II.

Despite this level of confidence in its security, the Playfair cipher is relatively easy to break, because it still leaves much of the structure of the plaintext language intact. A few hundred letters of ciphertext are generally sufficient.

One way of revealing the effectiveness of the Playfair and other ciphers is shown in Figure 2.6, based on [SIMM93]. The line labeled *plaintext* plots the frequency distribution of the more than 70,000 alphabetic characters in the *Encyclopaedia Britannica* article on cryptology.[5] This is also the frequency distribution of any monoalphabetic substitution cipher, because the frequency values for individual letters are the same, just with different letters substituted for the original letters. The plot was developed in the following way: The number of occurrences of each letter in the text was counted and divided by the number of occurrences of the letter e (the most frequently used letter). As a result, e has a relative frequency of 1, t of about 0.76, and so on. The points on the horizontal axis correspond to the letters in order of decreasing frequency.

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

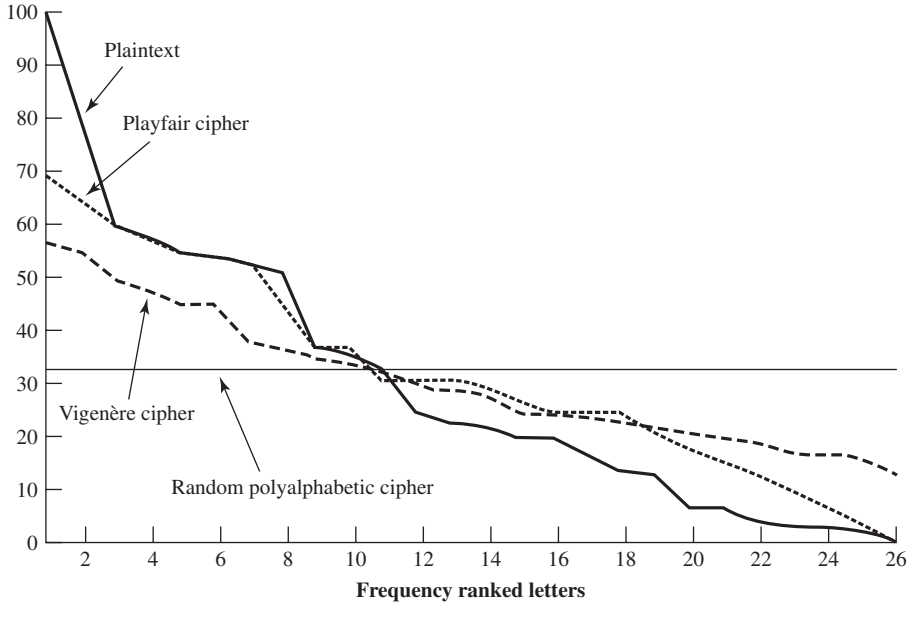


Figure 2.6 Relative Frequency of Occurrence of Letters

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

computed as $[\mathbf{A}^{-1}]_{ij} = (\det \mathbf{A})^{-1}(-1)^{i+j}(\mathrm{D}_{ji})$, where (D_{ji}) is the subdeterminant formed by deleting the jth row and the ith column of $\mathbf{A}$, det($\mathbf{A}$) is the determinant of $\mathbf{A}$, and $(\det \mathbf{A})^{-1}$ is the multiplicative inverse of $(\det \mathbf{A})$ mod 26.

Continuing our example,

$$\det \begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix} = (5 \times 3) - (8 \times 17) = -121 \bmod 26 = 9$$

We can show that $9^{-1} \bmod 26 = 3$, because $9 \times 3 = 27 \bmod 26 = 1$ (see Chapter 4 or Appendix E). Therefore, we compute the inverse of $\mathbf{A}$ as

$$\mathbf{A} = \begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix}$$

$$\mathbf{A}^{-1} \bmod 26 = 3\begin{pmatrix} 3 & -8 \\ -17 & 5 \end{pmatrix} = 3\begin{pmatrix} 3 & 18 \\ 9 & 5 \end{pmatrix} = \begin{pmatrix} 9 & 54 \\ 27 & 15 \end{pmatrix} = \begin{pmatrix} 9 & 2 \\ 1 & 15 \end{pmatrix}$$

The Hill Algorithm This encryption algorithm takes m successive plaintext letters and substitutes for them m ciphertext letters. The substitution is determined by m linear equations in which each character is assigned a numerical value (a = 0, b = 1, ... z = 25). For $m = 3$, the system can be described as

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

This can be expressed in terms of row vectors and matrices:[7]

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

or

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

where $\mathbf{C}$ and $\mathbf{P}$ are row vectors of length 3 representing the plaintext and ciphertext, and $\mathbf{K}$ is a 3×3 matrix representing the encryption key. Operations are performed mod 26.

For example, consider the plaintext "paymoremoney" and use the encryption key

$$\mathbf{K} = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}$$

[7]Some cryptography books express the plaintext and ciphertext as column vectors, so that the column vector is placed after the matrix rather than the row vector placed before the matrix. Sage uses row vectors, so we adopt that convention.

so

$$\mathbf{K} = \begin{pmatrix} 25 & 22 \\ 1 & 23 \end{pmatrix} \begin{pmatrix} 7 & 2 \\ 17 & 25 \end{pmatrix} = \begin{pmatrix} 549 & 600 \\ 398 & 577 \end{pmatrix} \bmod 26 = \begin{pmatrix} 3 & 2 \\ 8 & 5 \end{pmatrix}$$

This result is verified by testing the remaining plaintext–ciphertext pairs.

Polyalphabetic Ciphers

Another way to improve on the simple monoalphabetic technique is to use different monoalphabetic substitutions as one proceeds through the plaintext message. The general name for this approach is **polyalphabetic substitution cipher**. All these techniques have the following features in common:

1. A set of related monoalphabetic substitution rules is used.
2. A key determines which particular rule is chosen for a given transformation.

Vigenère Cipher The best known, and one of the simplest, polyalphabetic ciphers is the Vigenère cipher. In this scheme, the set of related monoalphabetic substitution rules consists of the 26 Caesar ciphers with shifts of 0 through 25. Each cipher is denoted by a key letter, which is the ciphertext letter that substitutes for the plaintext letter a. Thus, a Caesar cipher with a shift of 3 is denoted by the key value d.

We can express the Vigenère cipher in the following manner. Assume a sequence of plaintext letters $P = p_0, p_1, p_2, \ldots, p_{n-1}$ and a key consisting of the sequence of letters $K = k_0, k_1, k_2, \ldots, k_{m-1}$, where typically $m < n$. The sequence of ciphertext letters $C = C_0, C_1, C_2, \ldots, C_{n-1}$ is calculated as follows:

$$\begin{aligned} C = C_0, C_1, C_2, \ldots, C_{n-1} &= \mathrm{E}(K, P) = \mathrm{E}[(k_0, k_1, k_2, \ldots, k_{m-1}), (p_0, p_1, p_2, \ldots, p_{n-1})] \\ &= (p_0 + k_0) \bmod 26, (p_1 + k_1) \bmod 26, \ldots, (p_{m-1} + k_{m-1}) \bmod 26, \\ &\quad (p_m + k_0) \bmod 26, (p_{m+1} + k_1) \bmod 26, \ldots, (p_{2m-1} + k_{m-1}) \bmod 26, \ldots \end{aligned}$$

Thus, the first letter of the key is added to the first letter of the plaintext, mod 26, the second letters are added, and so on through the first m letters of the plaintext. For the next m letters of the plaintext, the key letters are repeated. This process continues until all of the plaintext sequence is encrypted. A general equation of the encryption process is

$$C_i = (p_i + k_{i \bmod m}) \bmod 26 \tag{2.3}$$

Compare this with Equation (2.1) for the Caesar cipher. In essence, each plaintext character is encrypted with a different Caesar cipher, depending on the corresponding key character. Similarly, decryption is a generalization of Equation (2.2):

$$p_i = (C_i - k_{i \bmod m}) \bmod 26 \tag{2.4}$$

To encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword. For example, if the keyword is *deceptive*, the message "we are discovered save yourself" is encrypted as

which has a somewhat regular structure. But after the second transposition, we have

```
17 09 05 27 24 16 12 07 10 02 22 20 03 25
15 13 04 23 19 14 11 01 26 21 18 08 06 28
```

This is a much less structured permutation and is much more difficult to cryptanalyze.

2.4 ROTOR MACHINES

The example just given suggests that multiple stages of encryption can produce an algorithm that is significantly more difficult to cryptanalyze. This is as true of substitution ciphers as it is of transposition ciphers. Before the introduction of DES, the most important application of the principle of multiple stages of encryption was a class of systems known as rotor machines.[9]

The basic principle of the rotor machine is illustrated in Figure 2.8. The machine consists of a set of independently rotating cylinders through which electrical pulses can flow. Each cylinder has 26 input pins and 26 output pins, with internal wiring that connects each input pin to a unique output pin. For simplicity, only three of the internal connections in each cylinder are shown.

If we associate each input and output pin with a letter of the alphabet, then a single cylinder defines a monoalphabetic substitution. For example, in Figure 2.8, if an operator depresses the key for the letter A, an electric signal is applied to the first pin of the first cylinder and flows through the internal connection to the twenty-fifth output pin.

Consider a machine with a single cylinder. After each input key is depressed, the cylinder rotates one position, so that the internal connections are shifted accordingly. Thus, a different monoalphabetic substitution cipher is defined. After 26 letters of plaintext, the cylinder would be back to the initial position. Thus, we have a polyalphabetic substitution algorithm with a period of 26.

A single-cylinder system is trivial and does not present a formidable cryptanalytic task. The power of the rotor machine is in the use of multiple cylinders, in which the output pins of one cylinder are connected to the input pins of the next. Figure 2.8 shows a three-cylinder system. The left half of the figure shows a position in which the input from the operator to the first pin (plaintext letter a) is routed through the three cylinders to appear at the output of the second pin (ciphertext letter B).

With multiple cylinders, the one closest to the operator input rotates one pin position with each keystroke. The right half of Figure 2.8 shows the system's configuration after a single keystroke. For every complete rotation of the inner cylinder, the middle cylinder rotates one pin position. Finally, for every complete rotation of the middle cylinder, the outer cylinder rotates one pin position. This is the same type of operation seen with an odometer. The result is that there are $26 \times 26 \times 26 = 17{,}576$ different substitution alphabets used before the system

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

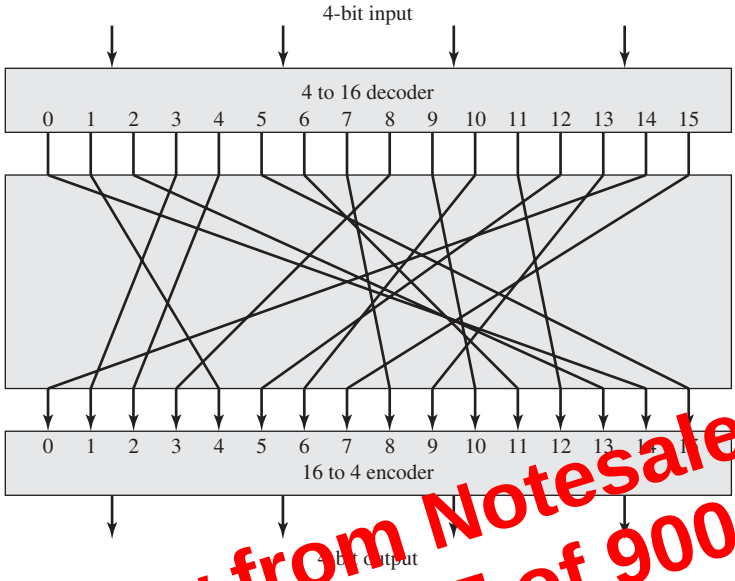


Figure 3.2 General n-bit n-bit Block Substitution (shown with $n = 4$)

defined by a tabulation, as shown in Table 3.1. This is the most general form of block cipher and can be used to define any reversible mapping between plaintext and ciphertext. Feistel refers to this as the *ideal block cipher*, because it allows for the maximum number of possible encryption mappings from the plaintext block [FEIS75].

Table 3.1 Encryption and Decryption Tables for Substitution Cipher of Figure 3.2

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

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

But there is a practical problem with the ideal block cipher. If a small block size, such as $n = 4$, is used, then the system is equivalent to a classical substitution cipher. Such systems, as we have seen, are vulnerable to a statistical analysis of the plaintext. This weakness is not inherent in the use of a substitution cipher but rather results from the use of a small block size. If n is sufficiently large and an arbitrary reversible substitution between plaintext and ciphertext is allowed, then the statistical characteristics of the source plaintext are masked to such an extent that this type of cryptanalysis is infeasible.

An arbitrary reversible substitution cipher (the ideal block cipher) for a large block size is not practical, however, from an implementation and performance point of view. For such a transformation, the mapping itself constitutes the key. Consider again Table 3.1, which defines one particular reversible mapping from plaintext to ciphertext for $n = 4$. The mapping can be defined by the entries in the second column, which show the value of the ciphertext for each plaintext block. This, in essence, is the key that determines the specific mapping from among all possible mappings. In this case, using this straightforward method of defining the key, the required key length is (4 bits) $\times$ (16 rows) $=$ 64 bits. In general, for an n-bit ideal block cipher, the length of the key defined in this fashion is $n \times 2^n$ bits. For a 64-bit block, which is a desirable length to thwart statistical attacks, the required key length is $64 \times 2^{64} = 2^{70} \approx 10^{21}$ bits.

In considering these difficulties, Feistel points out that what is needed is an approximation to the ideal block cipher system for large n, built up out of components that are easily realizable [FEIS75]. But before turning to Feistel's approach, let us make one other observation. We could use the general block substitution cipher but, to make its implementation tractable, confine ourselves to a subset of the $2^n!$ possible reversible mappings. For example, suppose we define the mapping in terms of a set of linear equations. In the case of $n = 4$, we have

$$
\begin{aligned}
y_1 &= k_{11}x_1 + k_{12}x_2 + k_{13}x_3 + k_{14}x_4 \\
y_2 &= k_{21}x_1 + k_{22}x_2 + k_{23}x_3 + k_{24}x_4 \\
y_3 &= k_{31}x_1 + k_{32}x_2 + k_{33}x_3 + k_{34}x_4 \\
y_4 &= k_{41}x_1 + k_{42}x_2 + k_{43}x_3 + k_{44}x_4
\end{aligned}
$$

where the x_i are the four binary digits of the plaintext block, the y_i are the four binary digits of the ciphertext block, the k_{ij} are the binary coefficients, and arithmetic is mod 2. The key size is just n^2, in this case 16 bits. The danger with this kind of formulation is that it may be vulnerable to cryptanalysis by an attacker that is aware of the structure of the algorithm. In this example, what we have is essentially the Hill cipher discussed in Chapter 2, applied to binary data rather than characters. As we saw in Chapter 2, a simple linear system such as this is quite vulnerable.

The Feistel Cipher

Feistel proposed [FEIS73] that we can approximate the ideal block cipher by utilizing the concept of a product cipher, which is the execution of two or more simple ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers. The essence of the approach is to develop a block

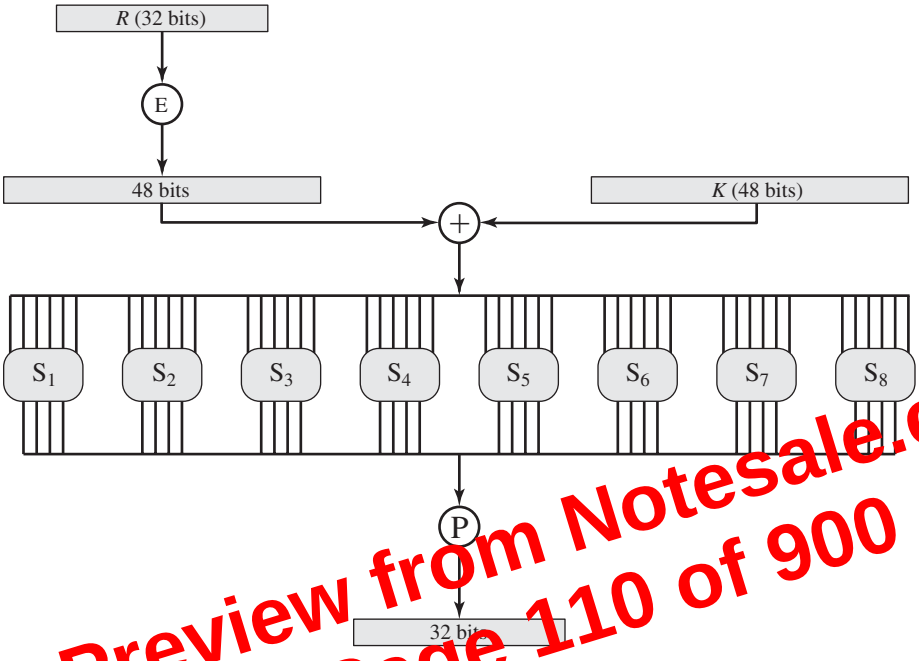


Figure 3.7 Calculation of F(R, K)

The outer two bits of each group select one of four possible substitutions (one row of an S-box). Then a 4-bit output value is substituted for the particular 4-bit input (the middle four input bits). The 32-bit output from the eight S-boxes is then permuted, so that on the next round, the output from each S-box immediately affects as many others as possible.

Key Generation Returning to Figures 3.5 and 3.6, we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated by the lack of shading in Table 3.4a. The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 3.4b). The resulting 56-bit key is then treated as two 28-bit quantities, labeled C_0 and D_0. At each round, C_{i-1} and D_{i-1} are separately subjected to a circular left shift or (rotation) of 1 or 2 bits, as governed by Table 3.4d. These shifted values serve as input to the next round. They also serve as input to the part labeled Permuted Choice Two (Table 3.4c), which produces a 48-bit output that serves as input to the function $F(R_{i-1}, K_i)$.

DES Decryption

As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed.

Results

Table 3.5 shows the progression of the algorithm. The first row shows the 32-bit values of the left and right halves of data after the initial permutation. The next 16 rows show the results after each round. Also shown is the value of the 48-bit subkey generated for each round. Note that $L_i = R_{i-1}$. The final row shows the left- and right-hand values after the inverse initial permutation. These two values combined form the ciphertext.

The Avalanche Effect

A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the ciphertext. This is referred to as the avalanche effect. If the change were small, this might provide a way to reduce the size of the plaintext or key space to be searched.

Using the example from Table 3.5, Table 3.6 shows the result when the fourth bit of the plaintext is changed so that the plaintext is `1244cbace2ed286420`. The second column of the table shows the intermediate 64-bit values at the end of each round for the two plaintexts. The third column shows the number of bits that differ between the two intermediate values. The table shows that, after just three rounds, 18 bits differ between the two blocks. On completion, the two ciphertexts differ in 32 bit positions.

Table 3.5 DES Example

Round	K_i	L_i	R_i
IP		5a005a00	3cf03c0f
1	1e030f03080d2930	3cf03c0f	bad22845
2	0a31293432242318	bad22845	99e9b723
3	23072318201d0c1d	99e9b723	0bae3b9e
4	05261d3824311a20	0bae3b9e	42415649
5	3325340136002c25	42415649	18b3fa41
6	123a2d0d04262a1c	18b3fa41	9616fe23
7	021f120b1c130611	9616fe23	67117cf2
8	1c10372a2832002b	67117cf2	c11bfc09
9	04292a380c341f03	c11bfc09	887fbc6c
10	2703212607280403	887fbc6c	600f7e8b
11	2826390c31261504	600f7e8b	f596506e
12	12071c241a0a0f08	f596506e	738538b8
13	300935393c0d100b	738538b8	c6a62c4e
14	311e09231321182a	c6a62c4e	56b0bd75
15	283d3e0227072528	56b0bd75	75e8fd8f
16	2921080b13143025	75e8fd8f	25896490
IP^{-1}		da02ce3a	89ecac3b

Note: DES subkeys are shown as eight 6-bit values in hex format

Table 3.6 Avalanche Effect in DES: Change in Plaintext

Round		δ
	02468aceeca86420 12468aceeca86420	1
1	3cf03c0fbad22845 3cf03c0fbad32845	1
2	bad2284599e9b723 bad3284539a9b7a3	5
3	99e9b7230bae3b9e 39a9b7a3171cb8b3	18
4	0bae3b9e42415649 171cb8b3ccaca55e	34
5	4241564918b3fa41 ccaca55ed16c3653	37
6	18b3fa419616fe23 d16c3653cf402c68	33
7	9616fe2367117cf2 cf402c682b2cefbc	32
8	67117cf2c11bfc09 [illegible]	33

Round		δ
9	c11bfc09887fbc6c 99f911532eed7d94	32
10	887fbc6c600f7e8b 2eed7d94d0f23094	34
11	600f7e8bf596506e d0f23094455da9c4	37
12	f596506e738538b8 455da9c47f6e3cf3	31
13	738538b8c6a62c4e 7f6e3cf34bc1a8d9	29
14	c6a62c4e56b[illegible]	33
15	[illegible] 1e07d4091ce[illegible]	31
16	75e8fd8[illegible] [illegible]6dc[illegible]5e5f59	32
IP^{-1}	da02ce3a89ecac3b 057cde97d7683f2a	32

Table 3.7 shows a similar test using the original plaintext of with two keys that differ in only the fourth bit position: the original key, `0f1571c947d9e859`, and the altered key, `1f1571c947d9e859`. Again, the results show that about half of the bits in the ciphertext differ and that the avalanche effect is pronounced after just a few rounds.

Table 3.7 Avalanche Effect in DES: Change in Key

Round		δ
	02468aceeca86420 02468aceeca86420	0
1	3cf03c0fbad22845 3cf03c0f9ad628c5	3
2	bad2284599e9b723 9ad628c59939136b	11
3	99e9b7230bae3b9e 9939136b768067b7	25
4	0bae3b9e42415649 768067b75a8807c5	29
5	4241564918b3fa41 5a8807c5488dbe94	26
6	18b3fa419616fe23 488dbe94aba7fe53	26
7	9616fe2367117cf2 aba7fe53177d21e4	27
8	67117cf2c11bfc09 177d21e4548f1de4	32

Round		δ
9	c11bfc09887fbc6c 548f1de471f64dfd	34
10	887fbc6c600f7e8b 71f64dfd4279876c	36
11	600f7e8bf596506e 4279876c399fdc0d	32
12	f596506e738538b8 399fdc0d6d208dbb	28
13	738538b8c6a62c4e 6d208dbbb9bdeeaa	33
14	c6a62c4e56b0bd75 b9bdeeaad2c3a56f	30
15	56b0bd7575e8fd8f d2c3a56f2765c1fb	33
16	75e8fd8f25896490 2765c1fb01263dc4	30
IP^{-1}	da02ce3a89ecac3b ee92b50606b62b0b	30

of the input differences. Overall, after three rounds, the probability that the output difference is as shown is equal to $0.25 \times 1 \times 0.25 = 0.0625$.

Linear Cryptanalysis

A more recent development is linear cryptanalysis, described in [MATS93]. This attack is based on finding linear approximations to describe the transformations performed in DES. This method can find a DES key given 2^{43} known plaintexts, as compared to 2^{47} chosen plaintexts for differential cryptanalysis. Although this is a minor improvement, because it may be easier to acquire known plaintext rather than chosen plaintext, it still leaves linear cryptanalysis infeasible as an attack on DES. So far, little work has been done by other groups to validate the linear cryptanalytic approach.

We now give a brief summary of the principle on which linear cryptanalysis is based. For a cipher with n-bit plaintext and ciphertext blocks and an m-bit key, let the plaintext block be labeled P[1], ... P[n], the ciphertext block C[1], ... C[n], and the key K[1], ..., K[m]. Then define

$$A[i, j, \ldots, k] = A[i] \oplus A[j] \oplus \ldots \oplus A[k]$$

The objective of linear cryptanalysis is to find an effective *linear* equation of the form:

$$P[\alpha_1, \alpha_2, \ldots, \alpha_a] \oplus C[\beta_1, \beta_2 \ldots, \beta_b] = K[\gamma_1, \gamma_2, \ldots, \gamma_c]$$

(where $x = 0$ or 1; $1 \leq a$; $b \leq n$; $c \leq m$; and where the α, β, and γ terms represent fixed, unique bit locations) that holds with probability $p \neq 0.5$. The further p is from 0.5, the more effective the equation. Once a proposed relation is determined, the procedure is to compute the results of the left-hand side of the preceding equation for a large number of plaintext–ciphertext pairs. If the result is 0 more than half the time, assume $K[\gamma_1, \gamma_2, \ldots, \gamma_c] = 0$. If it is 1 most of the time, assume $K[\gamma_1, \gamma_2, \ldots, \gamma_c] = 1$. This gives us a linear equation on the key bits. Try to get more such relations so that we can solve for the key bits. Because we are dealing with linear equations, the problem can be approached one round of the cipher at a time, with the results combined.

3.6 BLOCK CIPHER DESIGN PRINCIPLES

Although much progress has been made in designing block ciphers that are cryptographically strong, the basic principles have not changed all that much since the work of Feistel and the DES design team in the early 1970s. It is useful to begin this discussion by looking at the published design criteria used in the DES effort. Then we look at three critical aspects of block cipher design: the number of rounds, design of the function F, and key scheduling.

DES Design Criteria

The criteria used in the design of DES, as reported in [COPP94], focused on the design of the S-boxes and on the P function that takes the output of the S-boxes (Figure 3.7). The criteria for the S-boxes are as follows.

1. No output bit of any S-box should be too close a linear function of the input bits. Specifically, if we select any output bit and any subset of the six input bits, the fraction of inputs for which this output bit equals the XOR of these input bits should not be close to 0 or 1, but rather should be near 1/2.
2. Each row of an S-box (determined by a fixed value of the leftmost and rightmost input bits) should include all 16 possible output bit combinations.
3. If two inputs to an S-box differ in exactly one bit, the outputs must differ in at least two bits.
4. If two inputs to an S-box differ in the two middle bits exactly, the outputs must differ in at least two bits.
5. If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same.
6. For any nonzero 6-bit difference between inputs, no more than eight of the 32 pairs of inputs exhibiting that difference may result in the same output difference.
7. This is a criterion similar to the previous one, but for the case of three S-boxes.

Coppersmith pointed out that the first criterion in the preceding list was needed because the S-boxes are the only nonlinear part of DES. If the S-boxes were linear (i.e., each output bit is a linear combination of the input bits), the entire algorithm would be linear and easily broken. We have seen this phenomenon with the Hill cipher, which is linear. The remaining criteria were primarily aimed at thwarting differential cryptanalysis and at providing good confusion properties.

The criteria for the permutation P are as follows.

1. The four output bits from each S-box at round i are distributed so that two of them affect (provide input for) "middle bits" of round $(i + 1)$ and the other two affect end bits. The two middle bits of input to an S-box are not shared with adjacent S-boxes. The end bits are the two left-hand bits and the two right-hand bits, which are shared with adjacent S-boxes.
2. The four output bits from each S-box affect six different S-boxes on the next round, and no two affect the same S-box.
3. For two S-boxes j, k, if an output bit from S_j affects a middle bit of S_k on the next round, then an output bit from S_k cannot affect a middle bit of S_j. This implies that, for $j = k$, an output bit from S_j must not affect a middle bit of S_j.

These criteria are intended to increase the diffusion of the algorithm.

Number of Rounds

The cryptographic strength of a Feistel cipher derives from three aspects of the design: the number of rounds, the function F, and the key schedule algorithm. Let us look first at the choice of the number of rounds.

4.3 MODULAR ARITHMETIC

The Modulus

If a is an integer and n is a positive integer, we define $a \bmod n$ to be the remainder when a is divided by n. The integer n is called the **modulus**. Thus, for any integer a, we can rewrite Equation (4.1) as follows:

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

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

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

Two integers a and b are said to be **congruent modulo** n, if $(a \bmod n) = (b \bmod n)$. This is written as $a \equiv b \pmod{n}$.[2]

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

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

Properties of Congruences

Congruences have the following properties:

1. $a \equiv b \pmod{n}$ if $n|(a - b)$.
2. $a \equiv b \pmod{n}$ implies $b \equiv a \pmod{n}$.
3. $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$ imply $a \equiv c \pmod{n}$.

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

$23 \equiv 8 \pmod{5}$	because	$23 - 8 = 15 = 5 \times 3$
$-11 \equiv 5 \pmod{8}$	because	$-11 - 5 = -16 = 8 \times (-2)$
$81 \equiv 0 \pmod{27}$	because	$81 - 0 = 81 = 27 \times 3$

The remaining points are as easily proved.

Modular Arithmetic Operations

Note that, by definition (Figure 4.1), the (mod n) operator maps all integers into the set of integers $\{0, 1, \ldots, (n - 1)\}$. This suggests the question: Can we perform arithmetic

[2]We have just used the operator *mod* in two different ways: first as a **binary operator** that produces a remainder, as in the expression $a \bmod b$; second as a **congruence relation** that shows the equivalence of two integers, as in the expression $a \equiv b \pmod{n}$. See Appendix 4A for a discussion.

But we have already assumed that $r_i = ax_i + by_i$. Therefore,

$$x_i = x_{i-2} - q_i x_{i-1} \quad \text{and} \quad y_i = y_{i-2} - q_i y_{i-1}$$

We now summarize the calculations:

Extended Euclidean Algorithm			
Calculate	**Which satisfies**	**Calculate**	**Which satisfies**
$r_{-1} = a$		$x_{-1} = 1; y_{-1} = 0$	$a = ax_{-1} + by_{-1}$
$r_0 = b$		$x_0 = 0; y_0 = 1$	$b = ax_0 + by_0$
$r_1 = a \bmod b$ $q_1 = \lfloor a/b \rfloor$	$a = q_1 b + r_1$	$x_1 = x_{-1} - q_1 x_0 = 1$ $y_1 = y_{-1} - q_1 y_0 = -q_1$	$r_1 = ax_1 + by_1$
$r_2 = b \bmod r_1$ $q_2 = \lfloor b/r_1 \rfloor$	$b = q_2 r_1 + r_2$	$x_2 = x_0 - q_2 x_1$ $y_2 = y_0 - q_2 y_1$	$r_2 = ax_2 + by_2$
$r_3 = r_1 \bmod r_2$ $q_3 = \lfloor r_1/r_2 \rfloor$	$r_1 = q_3 r_2 + r_3$	$x_3 = x_1 - q_3 x_2$ $y_3 = y_1 - q_3 y_2$	$r_3 = ax_3 + by_3$
•	•	•	•
•	•	•	•
•	•	•	•
$r_n = r_{n-2} \bmod r_{n-1}$ $q_n = \lfloor r_{n-2}/r_{n-1} \rfloor$	$r_{n-2} = q_n r_{n-1} + r_n$	$x_n = x_{n-2} - q_n x_{n-1}$ $y_n = y_{n-2} - q_n y_{n-1}$	$r_n = ax_n + by_n$
$r_{n+1} = r_{n-1} \bmod r_n = 0$ $q_{n+1} = \lfloor r_{n-1}/r_{n-2} \rfloor$	$r_{n-1} = q_{n+1} r_n + 0$		$d = \gcd(a, b) = r_n$ $x = x_n; y = y_n$

We need to make several additional comments here. In each row, we calculate a new remainder r_i based on the remainders of the previous two rows, namely r_{i-1} and r_{i-2}. To start the algorithm, we need values for r_0 and r_{-1}, which are just a and b. It is then straightforward to determine the required values for x_{-1}, y_{-1}, x_0, and y_0.

We know from the original Euclidean algorithm that the process ends with a remainder of zero and that the greatest common divisor of a and b is $d = \gcd(a, b) = r_n$. But we also have determined that $d = r_n = ax_n + by_n$. Therefore, in Equation (4.7), $x = x_n$ and $y = y_n$.

As an example, let us use $a = 1759$ and $b = 550$ and solve for $1759x + 550y = \gcd(1759, 550)$. The results are shown in Table 4.4. Thus, we have $1759 \times (-111) + 550 \times 355 = -195249 + 195250 = 1$.

Table 4.4 Extended Euclidean Algorithm Example

i	r_i	q_i	x_i	y_i
–1	1759		1	0
0	550		0	1
1	109	3	1	–3
2	5	5	–5	16
3	4	21	106	–339
4	1	1	–111	355
5	0	4		

Result: $d = 1$; $x = -111$; $y = 355$

- Polynomial arithmetic in which the coefficients are in GF(p), and the polynomials are defined modulo a polynomial $m(x)$ whose highest power is some integer n.

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

Ordinary Polynomial Arithmetic

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

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

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

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

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

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

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

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

then addition is defined as

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

and multiplication is defined as

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

where

$$c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_{k-1} b_1 + a_k b_0$$

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

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

$$q(x)g(x) + r(x) = (x + 2)(x^2 - x + 1) + x = (x^3 + x^2 - x + 2) + x$$
$$= x^3 + x^2 + 2 = f(x)$$

For our purposes, polynomials over GF(2) are of most interest. Recall from Section 4.5 that in GF(2), addition is equivalent to the XOR operation, and multiplication is equivalent to the logical AND operation. Further, addition and subtraction are equivalent mod 2: $1 + 1 = 1 - 1 = 0; 1 + 0 = 1 - 0 = 1; 0 + 1 = 0 - 1 = 1$.

Figure 4.4 shows an example of polynomial arithmetic over GF(2). For $f(x) = (x^7 + x^5 + x^4 + x^3 + x + 1)$ and $g(x) = (x^3 + x + 1)$, the figure shows $f(x) + g(x); f(x) - g(x); f(x) \times g(x)$; and $f(x)/g(x)$. Note that $g(x) \times q(x) + r(x) = f(x)$.

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

The polynomial[9] $f(x) = x^4 + 1$ over GF(2) is reducible, because

$$x^4 + 1 = (x + 1)(x^3 + x^2 + x + 1).$$

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

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

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

[9] In the remainder of this chapter, unless otherwise noted, all examples are of polynomials over GF(2).

Summary

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

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

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

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

4.7 FINITE FIELDS OF THE FORM GF(2^n)

Earlier in this chapter, we mentioned that the order of a finite field must be of the form p^n, where p is a prime and n is a positive integer. In Section 4.5, we looked at the special case of finite fields with order p. We found that, using modular arithmetic in Z_p, all of the axioms for a field (Figure 4.2) are satisfied. For polynomials over p^n, with $n > 1$, operations modulo p^n do not produce a field. In this section, we show what structure satisfies the axioms for a field in a set with p^n elements and concentrate on GF(2^n).

Motivation

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

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

c. Show that if $1 \leq A, B \leq 2^N$, then Stein's algorithm takes at most $4N$ steps to find $\gcd(m, n)$. Thus, Stein's algorithm works in roughly the same number of steps as the Euclidean algorithm.

d. Demonstrate that Stein's algorithm does indeed return $\gcd(A, B)$.

4.19 Using the extended Euclidean algorithm, find the multiplicative inverse of

a. 1234 mod 4321

b. 24140 mod 40902

c. 550 mod 1769

4.20 Develop a set of tables similar to Table 4.5 for GF(5).

4.21 Demonstrate that the set of polynomials whose coefficients form a field is a ring.

4.22 Demonstrate whether each of these statements is true or false for polynomials over a field.

a. The product of monic polynomials is monic.

b. The product of polynomials of degrees m and n has degree $m + n$.

c. The sum of polynomials of degrees m and n has degree $\max[m, n]$.

4.23 For polynomial arithmetic with coefficients in Z_{10}, perform the following calculations.

a. $(7x + 2) - (x^2 + 5)$

b. $(6x^2 + x + 3) \times (5x^2 + 2)$

4.24 Determine which of the following are reducible over GF(2)

a. $x^3 + 1$

b. $x^3 + x^2 + 1$

c. $x^4 + 1$ (be careful)

4.25 Determine the gcd of the following pairs of polynomials.

a. $x^3 + x + 1$ and $x^2 + x + 1$ over GF(2)

b. $x^3 - x + 1$ and $x^2 + 1$ over GF(3)

c. $x^5 + x^4 + x^3 - x^2 - x + 1$ and $x^3 + x^2 + x + 1$ over GF(3)

d. $x^5 + 88x^4 + 73x^3 + 83x^2 + 51x + 67$ and $x^3 + 97x^2 + 40x + 38$ over GF(101)

4.26 Develop a set of tables similar to Table 4.7 for GF(4) with $m(x) = x^2 + x + 1$.

4.27 Determine the multiplicative inverse of $x^3 + x + 1$ in $GF(2^4)$ with $m(x) = x^4 + x + 1$.

4.28 Develop a table similar to Table 4.9 for $GF(2^4)$ with $m(x) = x^4 + x + 1$.

Programming Problems

4.29 Write a simple four-function calculator in $GF(2^4)$. You may use table lookups for the multiplicative inverses.

4.30 Write a simple four-function calculator in $GF(2^8)$. You should compute the multiplicative inverses on the fly.

APPENDIX 4A THE MEANING OF MOD

The operator mod is used in this book and in the literature in two different ways: as a binary operator and as a congruence relation. This appendix explains the distinction and precisely defines the notation used in this book regarding parentheses. This notation is common but, unfortunately, not universal.

where all of the variables are integers. Two conventions are used. The congruence sign is $\equiv$. The modulus for the relation is defined by placing the mod operator followed by the modulus in parentheses.

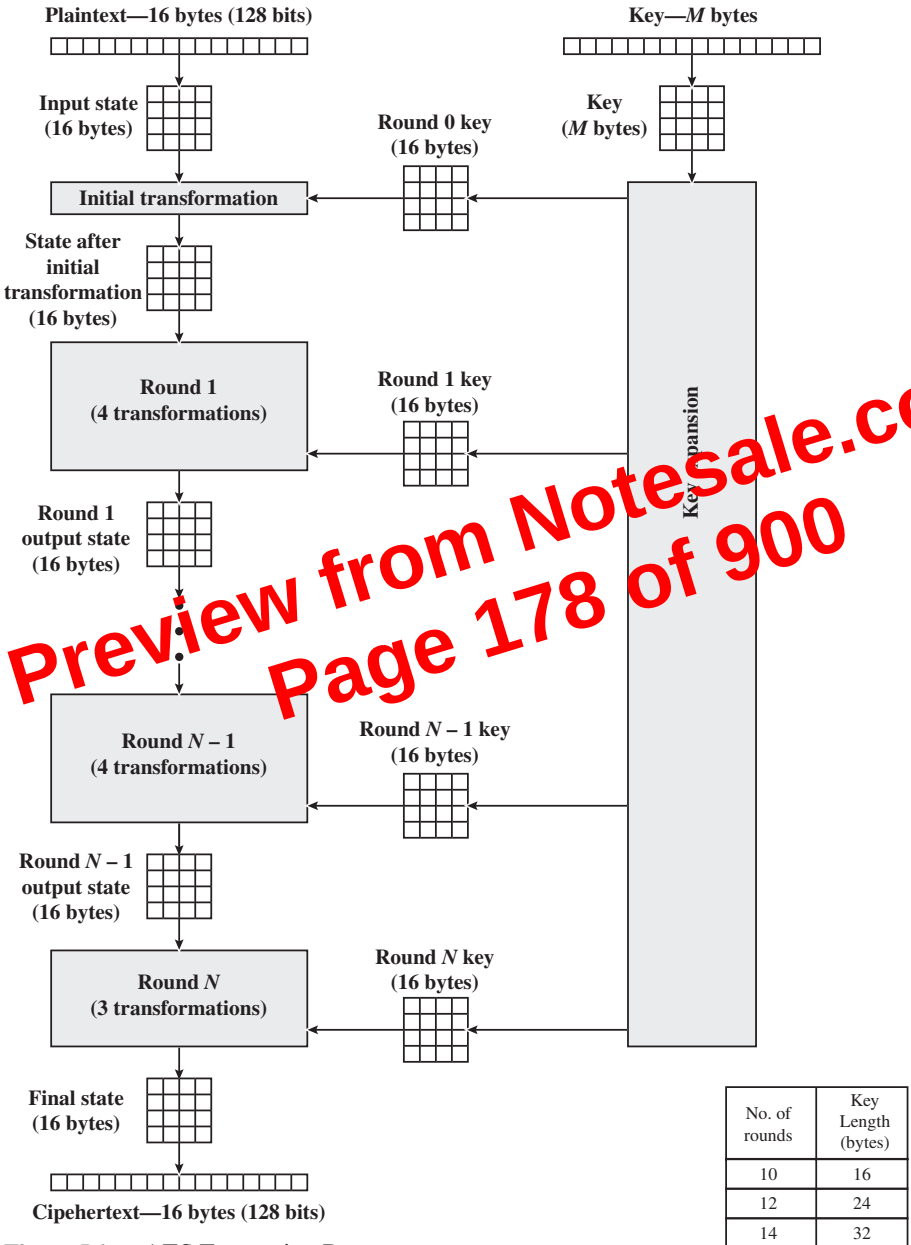
The congruence relation is used to define **residue classes**. Those numbers that have the same remainder r when divided by m form a residue class (mod m). There are m residue classes (mod m). For a given remainder r, the residue class to which it belongs consists of the numbers

$$r, r \pm m, r \pm 2m, \ldots$$

According to our definition, the congruence

$$a \equiv b \pmod{m}$$

signifies that the numbers a and b differ by a multiple of m. Consequently, the congruence can also be expressed in the terms that a and b belong to the same residue class (mod m).



No. of rounds	Key Length (bytes)
10	16
12	24
14	32

Figure 5.1 AES Encryption Process

which can be considered Round 0. Each transformation takes one or more 4×4 matrices as input and produces a 4×4 matrix as output. Figure 5.1 shows that the output of each round is a 4×4 matrix, with the output of the final round being the ciphertext. Also, the key expansion function generates $N + 1$ round keys, each of which is a distinct 4×4 matrix. Each round key serve as one of the inputs to the AddRoundKey transformation in each round.

of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

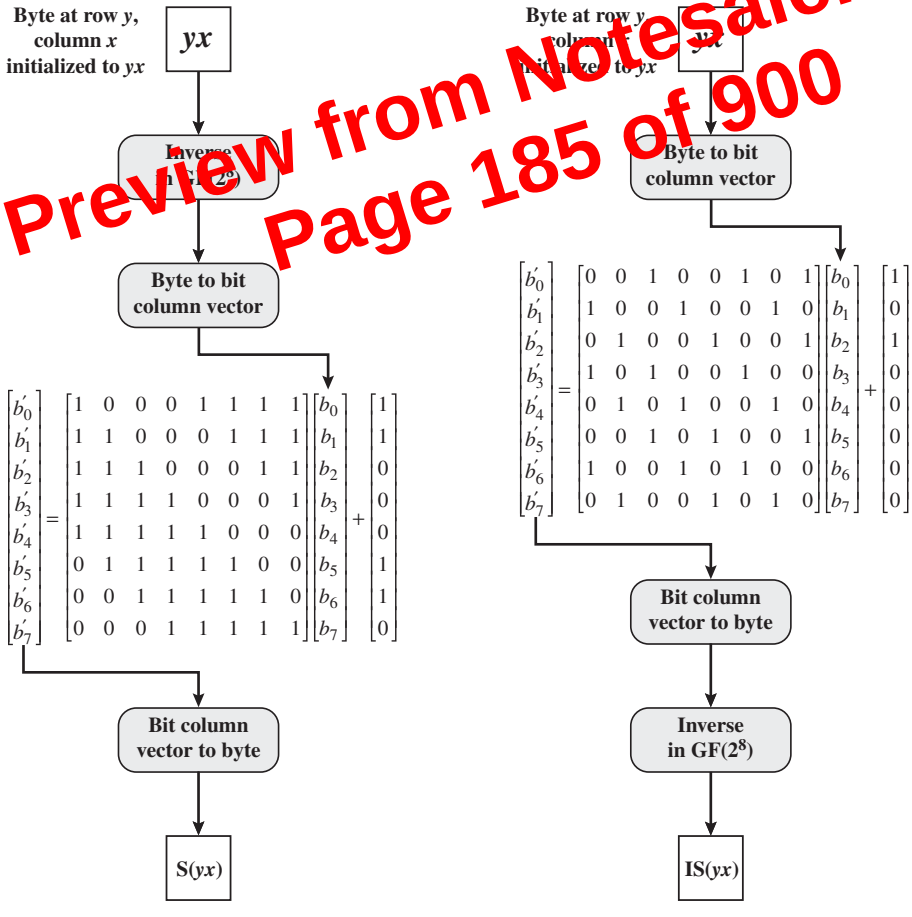
Here is an example of the SubBytes transformation:

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

The S-box is constructed in the following fashion (Figure 5.6a).



(a) Calculation of byte at row *y*, column *x* of S-box

(a) Calculation of byte at row *y*, column *x* of IS-box

Figure 5.6 Constuction of S-Box and IS-Box

1. Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02}, . . . , {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row y, column x is $\{yx\}$.
2. Map each byte in the S-box to its multiplicative inverse in the finite field GF(2^8); the value {00} is mapped to itself.
3. Consider that each byte in the S-box consists of 8 bits labeled (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0). Apply the following transformation to each bit of each byte in the S-box:

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \qquad \textbf{(5.1)}$$

where c_i is the ith bit of byte c with the value {63}; that is, $(c_7c_6c_5c_4c_3c_2c_1c_0) = (01100011)$. The prime ($'$) indicates that the variable is to be updated by the value on the right. The AES standard depicts this transformation in matrix form as follows.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \qquad \textbf{(5.2)}$$

Equation (5.2) has to be interpreted carefully. In ordinary matrix multiplication,[4] each element in the product matrix is the sum of products of the elements of one row and one column. In this case, each element in the product matrix is the bitwise XOR of products of elements of one row and one column. Furthermore, the final addition shown in Equation (5.2) is a bitwise XOR. Recall from Section 4.7 that the bitwise XOR is addition in GF(2^8).

As an example, consider the input value {95}. The multiplicative inverse in GF(2^8) is $\{95\}^{-1} = \{8A\}$, which is 10001010 in binary. Using Equation (5.2),

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

[4]For a brief review of the rules of matrix and vector multiplication, refer to Appendix E.

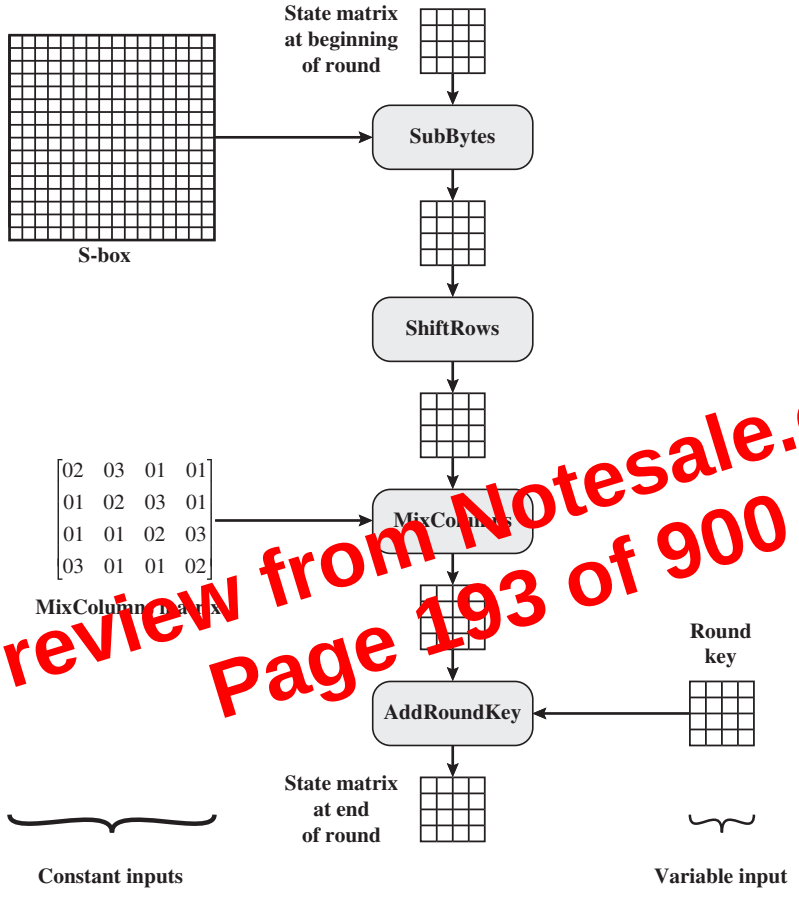


Figure 5.8 Inputs for Single AES Round

5.4 AES KEY EXPANSION

Key Expansion Algorithm

The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a four-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The pseudocode on the next page describes the expansion.

The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time. Each added word **w**[i] depends on the immediately preceding word, **w**[i − 1], and the word four positions back, w[i − 4]. In three out of four cases, a simple XOR is used. For a word whose position in the **w** array is a multiple of 4, a more complex function is used. Figure 5.9 illustrates the generation of the expanded key, using the symbol g to represent that complex function. The function g consists of the following subfunctions.

1. RotWord performs a one-byte circular left shift on a word. This means that an input word $[B_0, B_1, B_2, B_3]$ is transformed into $[B_1, B_2, B_3, B_0]$.
2. SubWord performs a byte substitution on each byte of its input word, using the S-box (Table 5.2a).
3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j].

The round constant is a word in which the three rightmost bytes are always 0. Thus, the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round and is defined as Rcon[j] = (RC[j], 0, 0, 0), with RC[1] = 1, RC[j] = 2 • RC[j−1] and with multiplication defined over the field GF(2^8). The values of RC[j] in hexadecimal are

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

For example, suppose that the round key for round 8 is

EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

Then the first 4 bytes (first column) of the round key for round 9 are calculated as follows:

i (decimal)	temp	After RotWord	After SubWord	Rcon (9)	After XOR with Rcon	w[i−4]	w[i] = temp ⊕ w[i−4]
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

Rationale

The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks. The inclusion of a round-dependent round constant eliminates the symmetry, or similarity, between the ways in which round keys are generated in different rounds. The specific criteria that were used are [DAEM99]

- Knowledge of a part of the cipher key or round key does not enable calculation of many other round-key bits.
- An invertible transformation [i.e., knowledge of any *Nk* consecutive words of the expanded key enables regeneration the entire expanded key (*Nk* = key size in words)].
- Speed on a wide range of processors.
- Usage of round constants to eliminate symmetries.
- Diffusion of cipher key differences into the round keys; that is, each key bit affects many round key bits.
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only.
- Simplicity of description.

32-Bit Processor The implementation described in the preceding subsection uses only 8-bit operations. For a 32-bit processor, a more efficient implementation can be achieved if operations are defined on 32-bit words. To show this, we first define the four transformations of a round in algebraic form. Suppose we begin with a **State** matrix consisting of elements $a_{i,j}$ and a round-key matrix consisting of elements $k_{i,j}$. Then the transformations can be expressed as follows.

SubBytes	$b_{i,j} = \mathrm{S}[a_{i,j}]$
ShiftRows	$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-1} \\ b_{2,j-2} \\ b_{3,j-3} \end{bmatrix}$
MixColumns	$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$
AddRoundKey	$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$

In the ShiftRows equation, the column indices are taken mod 4. We can combine all of these expressions into a single equation:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} \mathrm{S}[a_{0,j}] \\ \mathrm{S}[a_{1,j-1}] \\ \mathrm{S}[a_{2,j-2}] \\ \mathrm{S}[a_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

$$= \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot \mathrm{S}[a_{0,j}] \right) \oplus \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot \mathrm{S}[a_{1,j-1}] \right) \oplus \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot \mathrm{S}[a_{2,j-2}] \right)$$

$$\oplus \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot \mathrm{S}[a_{3,j-3}] \right) \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

In the second equation, we are expressing the matrix multiplication as a linear combination of vectors. We define four 256-word (1024-byte) tables as follows.

nibbles in that column. The transformation can be defined by the following matrix multiplication on **State** (Figure 5.14):

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} \\ s'_{1,0} & s'_{1,1} \end{bmatrix}$$

Performing the matrix multiplication, we get

$$\begin{aligned} S'_{0,0} &= S_{0,0} \oplus (4 \bullet S_{1,0}) \\ S'_{1,0} &= (4 \bullet S_{0,0}) \oplus S_{1,0} \\ S'_{0,1} &= S_{0,1} \oplus (4 \bullet S_{1,1}) \\ S'_{1,1} &= (4 \bullet S_{0,1}) \oplus S_{1,1} \end{aligned}$$

Where arithmetic is performed in GF(2^4), and the symbol • refers to multiplication in GF(2^4). Appendix I provides the addition and multiplication tables. The following is an example.

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 6 & 4 \\ C & 0 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 7 & 3 \end{bmatrix}$$

The inverse mix column function is defined as:

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} \\ s'_{1,0} & s'_{1,1} \end{bmatrix}$$

We demonstrate that we have indeed defined the inverse in the following fashion.

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix}$$

The preceding matrix multiplication makes use of the following results in GF(2^4): $9 + (2 \bullet 4) = 9 + 8 = 1$ and $(9 \bullet 4) + 2 = 2 + 2 = 0$. These operations can be verified using the arithmetic tables in Appendix I or by polynomial arithmetic.

The mix column function is the most difficult to visualize. Accordingly, we provide an additional perspective on it in Appendix I.

Key Expansion For key expansion, the 16 bits of the initial key are grouped into a row of two 8-bit words. Figure 5.15 shows the expansion into six words, by the calculation of four new words from the initial two words. The algorithm is

$$\begin{aligned} w_2 &= w_0 \oplus g(w_1) = w_0 \oplus \text{Rcon}(1) \oplus \text{SubNib}(\text{RotNib}(w_1)) \\ w_3 &= w_2 \oplus w_1 \\ w_4 &= w_2 \oplus g(w_3) = w_2 \oplus \text{Rcon}(2) \oplus \text{SubNib}(\text{RotNib}(w_3)) \\ w_5 &= w_4 \oplus w_3 \end{aligned}$$

Rcon is a round constant, defined as follows: RC[i] = x^{i+2}, so that RC[1] = x^3 = 1000 and RC[2] = $x^4 \bmod (x^4 + x + 1) = x + 1$ = 0011. RC[i] forms the leftmost nibble of a byte, with the rightmost nibble being all zeros. Thus, Rcon(1) = 10000000 and Rcon(2) = 00110000.

For example, suppose the key is 2D55 = 0010 1101 0101 0101 = $w_0 w_1$. Then

Reduction to a Single Stage Suppose it were true for DES, for all 56-bit key values, that given any two keys K_1 and K_2, it would be possible to find a key K_3 such that

$$\mathrm{E}(K_2, \mathrm{E}(K_1, P)) = \mathrm{E}(K_3, P) \tag{6.1}$$

If this were the case, then double encryption, and indeed any number of stages of multiple encryption with DES, would be useless because the result would be equivalent to a single encryption with a single 56-bit key.

On the face of it, it does not appear that Equation (6.1) is likely to hold. Consider that encryption with DES is a mapping of 64-bit blocks to 64-bit blocks. In fact, the mapping can be viewed as a permutation. That is, if we consider all 2^{64} possible input blocks, DES encryption with a specific key will map each block into a unique 64-bit block. Otherwise, if, say, two given input blocks mapped to the same output block, then decryption to recover the original plaintext would be impossible. With 2^{64} possible inputs, how many different mappings are there that generate a permutation of the input blocks? The value is easily seen to be

$$(2^{64})! = 10^{3473800000000000000000} \; (> 10^{10^{20}})$$

On the other hand, DES defines one mapping for each different key, for a total number of mappings:

$$2^{56} < 10^{17}$$

Therefore, it is reasonable to assume that if DES is used twice with different keys, it will produce one of the many mappings that are not defined by a single application of DES. Although there was much supporting evidence for this assumption, it was not until 1992 that the assumption was proven [CAMP92].

Meet-in-the-Middle Attack Thus, the use of double DES results in a mapping that is not equivalent to a single DES encryption. But there is a way to attack this scheme, one that does not depend on any particular property of DES but that will work against any block encryption cipher.

The algorithm, known as a **meet-in-the-middle attack**, was first described in [DIFF77]. It is based on the observation that, if we have

$$C = \mathrm{E}(K_2, \mathrm{E}(K_1, P))$$

then (see Figure 6.1a)

$$X = \mathrm{E}(K_1, P) = \mathrm{D}(K_2, C)$$

Given a known pair, (P, C), the attack proceeds as follows. First, encrypt P for all 2^{56} possible values of K_1. Store these results in a table and then sort the table by the values of X. Next, decrypt C using all 2^{56} possible values of K_2. As each decryption is produced, check the result against the table for a match. If a match occurs, then test the two resulting keys against a new known plaintext–ciphertext pair. If the two keys produce the correct ciphertext, accept them as the correct keys.

For any given plaintext P, there are 2^{64} possible ciphertext values that could be produced by double DES. Double DES uses, in effect, a 112-bit key, so that

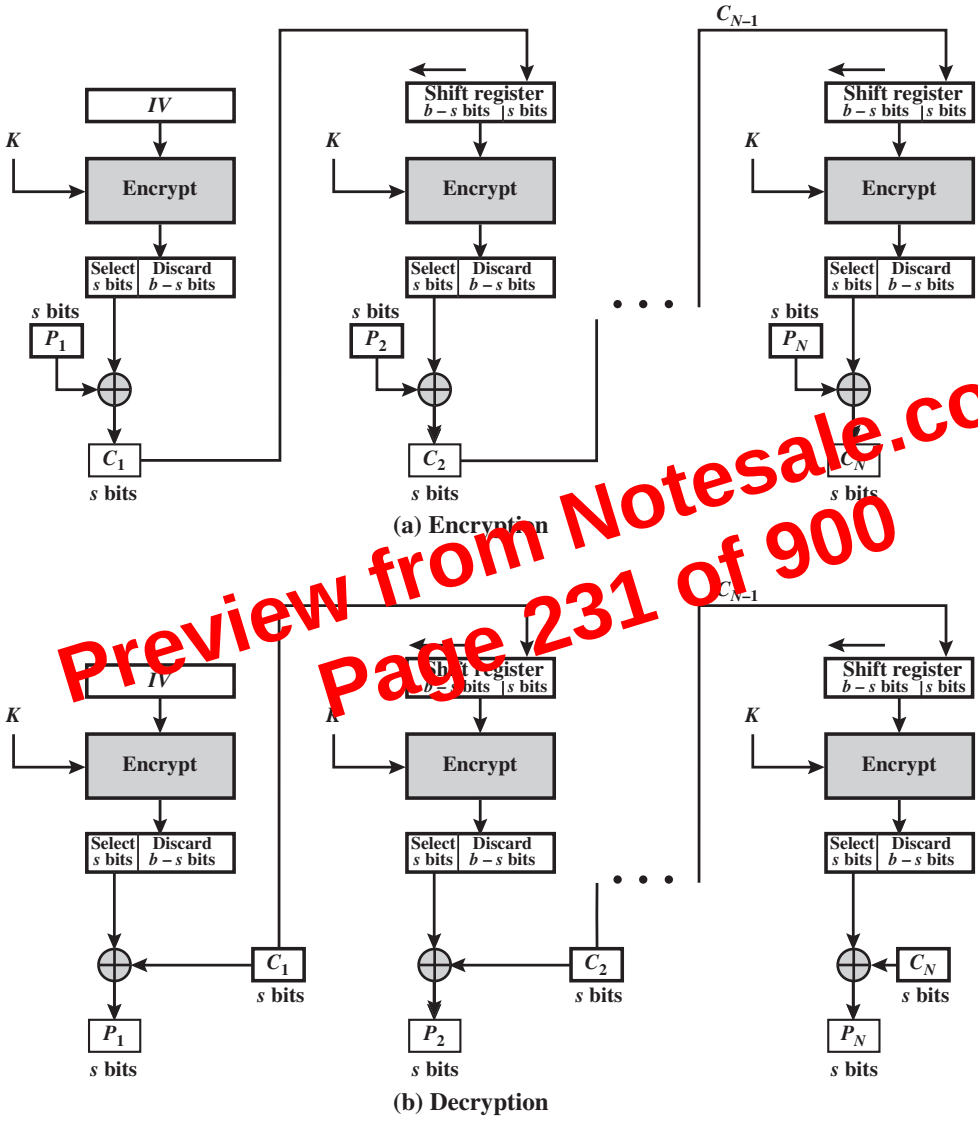


Figure 6.5 *s*-bit Cipher Feedback (CFB) Mode

CFB	$I_1 = IV$ $I_j = \text{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1} \quad j = 2, \ldots, N$ $O_j = \text{E}(K, I_j) \quad j = 1, \ldots, N$ $C_j = P_j \oplus \text{MSB}_s(O_j) \quad j = 1, \ldots, N$	$I_1 = IV$ $I_j = \text{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1} \quad j = 2, \ldots, N$ $O_j = \text{E}(K, I_j) \quad j = 1, \ldots, N$ $P_j = C_j \oplus \text{MSB}_s(O_j) \quad j = 1, \ldots, N$

Although CFB can be viewed as a stream cipher, it does not conform to the typical construction of a stream cipher. In a typical stream cipher, the cipher takes as input some initial value and a key and generates a stream of bits, which is then XORed with the plaintext bits (see Figure 3.1). In the case of CFB, the stream of bits that is XORed with the plaintext also depends on the plaintext.

CHAPTER 7

PSEUDORANDOM NUMBER GENERATION AND STREAM CIPHERS

The comparatively late rise of the theory of probability shows how hard it is to grasp, and the many paradoxes show clearly that we, as humans, lack a well grounded intuition in this matter.

In probability theory there is a great deal of art in setting up the model, in solving the problem, and in applying the results back to the real world actions that will follow.

—*The Art of Probability*, Richard Hamming

KEY POINTS

- A capability with application to a number of cryptographic functions is random or pseudorandom number generation. The principle requirement for this capability is that the generated number stream be unpredictable.
- A stream cipher is a symmetric encryption algorithm in which ciphertext output is produced bit-by-bit or byte-by-byte from a stream of plaintext input. The most widely used such cipher is RC4.

An important cryptographic function is cryptographically strong pseudorandom number generation. Pseudorandom number generators (PRNGs) are used in a variety of cryptographic and security applications. We begin the chapter with a look at the basic principles of PRNGs and contrast these with true random number generators (TRNGs).[1] Next, we look at some common PRNGs, including PRNGs based on the use of a symmetric block cipher.

The chapter then moves on to the topic of symmetric stream ciphers, which are based on the use of a PRNG. The chapter next examines the most important stream cipher, RC4. Finally, we examine TRNGs.

7.1 PRINCIPLES OF PSEUDORANDOM NUMBER GENERATION

Random numbers play an important role in the use of encryption for various network security applications. In this section, we provide a brief overview of the use of random numbers in cryptography and network security and then focus on the principles of pseudorandom number generation.

The Use of Random Numbers

A number of network security algorithms and protocols based on cryptography make use of random binary numbers. For example,

[1]A note on terminology. Some standards documents, notably NIST and ANSI, refer to a TRNG as a nondeterministic random number generator (NRNG) and a PRNG as a deterministic random number generator (DRNG).

The CTR algorithm for PRNG can be summarized as follows.

```
while (len (temp) < requested_number_of_bits) do
      V = (V + 1) mod 2^128.
      output_block = E(Key, V)
      temp = temp || ouput_block
```

The OFB algorithm can be summarized as follows.

```
while (len (temp) < requested_number_of_bits) do
      V = E(Key, V)
      temp = temp || V
```

To get some idea of the performance of these two PRNGs, consider the following short experiment. A random bit sequence of 256 bits was obtained from random.org, which uses three radios tuned between stations to pick up atmospheric noise. These 256 bits form the seed, allocated as

Key:	cfb0ef3108d49ec4562c5810b0a9af60
V:	4c89af0617bb728ed1e2ea8ba27f5a4

The total number of one bits in the 256-bit seed is 124, or a fraction of 0.48, which is reassuringly close to the ideal of 0.5.

For the OFB PRNG, Table 7.2 shows the first eight output blocks (1024 bits) with two rough measures of security. The second column shows the fraction of one bits in each 128-bit block. This corresponds to one of the NIST tests. The results indicate that the output is split roughly equally between zero and one bits. The third column shows the fraction of bits that match between adjacent blocks. If this number differs substantially from 0.5, that suggests a correlation between blocks, which could be a security weakness. The results suggest no correlation.

Table 7.2 Example Results for PRNG Using OFB

Output Block	Fraction of One Bits	Fraction of Bits that Match with Preceding Block
1786f4c7ff6e291dbdfdd90ec3453176	0.57	—
5e17b22b14677a4d66890f87565eae64	0.51	0.52
fd18284ac82251dfb3aa62c326cd46cc	0.47	0.54
c8e545198a758ef5dd86b41946389bd5	0.50	0.44
fe7bae0e23019542962e2c52d215a2e3	0.47	0.48
14fdf5ec99469598ae0379472803accd	0.49	0.52
6aeca972e5a3ef17bd1a1b775fc8b929	0.57	0.48
f7e97badf359d128f00d9b4ae323db64	0.55	0.45

- **Keys:** The generator makes use of three triple DES encryption modules. All three make use of the same pair of 56-bit keys, which must be kept secret and are used only for pseudorandom number generation.
- **Output:** The output consists of a 64-bit pseudorandom number and a 64-bit seed value.

Let us define the following quantities.

DT_i	Date/time value at the beginning of ith generation stage
V_i	Seed value at the beginning of ith generation stage
R_i	Pseudorandom number produced by the ith generation stage
K_1, K_2	DES keys used for each stage

Then

$$R_i = \text{EDE}([K_1, K_2], [V_i \oplus \text{EDE}([K_1, K_2], DT_i)])$$
$$V_{i+1} = \text{EDE}([K_1, K_2], [R_i \oplus \text{EDE}([K_1, K_2], DT_i)])$$

where EDE($[K_1, K_2], X$) refers to the sequence encrypt-decrypt-encrypt using two-key triple DES to encrypt X.

Several factors contribute to the cryptographic strength of this method. The technique involves a 112-bit key and three EDE encryptions for a total of nine DES encryptions. The scheme is driven by two pseudorandom inputs, the date and time value, and a seed produced by the generator that is distinct from the pseudorandom number produced by the generator. Thus, the amount of material that must be compromised by an opponent is overwhelming. Even if a pseudorandom number R_i were compromised, it would be impossible to deduce the V_{i+1} from the R_i, because an additional EDE operation is used to produce the V_{i+1}.

7.4 STREAM CIPHERS

A typical stream cipher encrypts plaintext one byte at a time, although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time. Figure 7.5 is a representative diagram of stream cipher structure. In this structure, a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. The output of the generator, called a **keystream**, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation. For example, if the next byte generated by the generator is 01101100 and the next plaintext byte is 11001100, then the resulting ciphertext byte is

```
  11001100  plaintext
⊕ 01101100  key stream
  10100000  ciphertext
```

RUKH08 Rukhin, A., et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.* NIST SP 800-22, August 2008.

SIMM92 Simmons, G., ed. *Contemporary Cryptology: The Science of Information Integrity.* Piscataway, NJ: IEEE Press, 1992.

ZENG91 Zeng, K.; Yang, C.; Wei, D.; and Rao, T. "Pseudorandom Bit Generators in Stream-Cipher Cryptography." *Computer*, February 1991.

Recommended Web Sites:

- **NIST Random Number Generation Technical Working Group:** Contains documents and tests developed by NIST that related to PRNGs for cryptographic applications. Also has useful set of links.
- **NIST Random Number Generation Cryptographic Toolkit:** Another useful NIST site with documents and links.
- **LavaRnd:** LavaRnd is an open source project that uses a chaotic source to generate truly random numbers. The site also has background information on random numbers in general.
- **Quantum Random Numbers:** You can access quantum random numbers on the fly here.
- **RandomNumber.org:** Another source of random numbers.
- **A Million Random Digits:** Compiled by the RAND Corporation.

7.8 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

backward unpredictability	pseudorandom function (PRF)	stream cipher
Blum, Blum, Shub generator	pseudorandom number generator (PRNG)	skew
deskewing	randomness	true random number generator (TRNG)
entropy source	RC4	unpredictability
forward unpredictability	seed	
keystream		
linear congruential generator		

Review Questions

7.1 What is the difference between statistical randomness and unpredictability?
7.2 List important design considerations for a stream cipher.
7.3 Why is it not desirable to reuse a stream cipher key?
7.4 What primitive operations are used in RC4?

More generally, we can say that the highest possible exponent to which a number can belong (mod n) is $\phi(n)$. If a number is of this order, it is referred to as a **primitive root** of n. The importance of this notion is that if a is a primitive root of n, then its powers

$$a, a^2, \ldots, a^{\phi(n)}$$

are distinct (mod n) and are all relatively prime to n. In particular, for a prime number p, if a is a primitive root of p, then

$$a, a^2, \ldots, a^{p-1}$$

are distinct (mod p). For the prime number 19, its primitive roots are 2, 3, 10, 13, 14, and 15.

Not all integers have primitive roots. In fact, the only integers with primitive roots are those of the form 2, 4, p^α, and $2p^\alpha$, where p is any odd prime and α is a positive integer. The proof is not simple but can be found in many number theory books, including [ORE76].

Logarithms for Modular Arithmetic

With ordinary positive real numbers, the logarithm function is the inverse of exponentiation. An analogous function exists for modular arithmetic.

Let us briefly review the properties of ordinary logarithms. The logarithm of a number is defined to be the power to which some positive base (except 1) must be raised in order to equal the number. That is, for base x and for a value y,

$$y = x^{\log_x(y)}$$

The properties of logarithms include

$$\log_x(1) = 0$$

$$\log_x(x) = 1$$

$$\log_x(yz) = log_x(y) + \log_x(z) \quad \textbf{(8.11)}$$

$$\log_x(y^r) = r \times \log_x(y) \quad \textbf{(8.12)}$$

Consider a primitive root a for some prime number p (the argument can be developed for nonprimes as well). Then we know that the powers of a from 1 through $(p - 1)$ produce each integer from 1 through $(p - 1)$ exactly once. We also know that any integer b satisfies

$$b \equiv r \,(\text{mod } p) \quad \text{for some } r, \text{ where } 0 \leq r \leq (p - 1)$$

by the definition of modular arithmetic. It follows that for any integer b and a primitive root a of prime number p, we can find a unique exponent i such that

$$b \equiv a^i (\text{mod } p) \quad \text{where } 0 \leq i \leq (p - 1)$$

Every Egyptian received two names, which were known respectively as the true name and the good name, or the great name and the little name; and while the good or little name was made public, the true or great name appears to have been carefully concealed.

—*The Golden Bough,* Sir James George Frazer

KEY POINTS

- Asymmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the different keys—one a public key and one a private key. It is also known as public-key encryption.
- Asymmetric encryption transforms plaintext into ciphertext using a one of two keys and an encryption algorithm. Using the paired key and a decryption algorithm, the plaintext is recovered from the ciphertext.
- Asymmetric encryption can be used for confidentiality, authentication, or both.
- The most widely used public-key cryptosystem is RSA. The difficulty of attacking RSA is based on the difficulty of finding the prime factors of a composite number.

The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography. From its earliest beginnings to modern times, virtually all cryptographic systems have been based on the elementary tools of substitution and permutation. After millennia of working with algorithms that could be calculated by hand, a major advance in symmetric cryptography occurred with the development of the rotor encryption/decryption machine. The electromechanical rotor enabled the development of fiendishly complex cipher systems. With the availability of computers, even more complex systems were devised, the most prominent of which was the Lucifer effort at IBM that culminated in the Data Encryption Standard (DES). But both rotor machines and DES, although representing significant advances, still relied on the bread-and-butter tools of substitution and permutation.

Public-key cryptography provides a radical departure from all that has gone before. For one thing, public-key algorithms are based on mathematical functions rather than on substitution and permutation. More important, public-key cryptography is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication, as we shall see.

Before proceeding, we should mention several common misconceptions concerning public-key encryption. One such misconception is that public-key encryption is more secure from cryptanalysis than is symmetric encryption. In fact, the security of any encryption scheme depends on the length of the key and the computational work

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
- **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

The essential steps are the following.

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As Figure 9.1a suggests, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a user's private key remains protected and secret, incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

Table 9.2 summarizes some of the important aspects of symmetric and public-key encryption. To discriminate between the two, we refer to the key used in symmetric encryption as a **secret key**. The two keys used for asymmetric encryption are referred to as the **public key** and the **private key**.[2] Invariably, the private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with symmetric encryption.

Let us take a closer look at the essential elements of a public-key encryption scheme, using Figure 9.2 (compare with Figure 2.2). There is some source A that

[2]The following notation is used consistently throughout. A secret key is represented by K_m, where m is some modifier; for example, K_a is a secret key owned by user A. A public key is represented by PU_a, for user A, and the corresponding private key is PR_a. Encryption of plaintext X can be performed with a secret key, a public key, or a private key, denoted by $E(K_a, X)$, $E(PU_a, X)$, and $E(PR_a, X)$, respectively. Similarly, decryption of ciphertext C can be performed with a secret key, a public key, or a private key, denoted by $D(K_a, X)$, $D(PU_a, X)$, and $D(PR_a, X)$, respectively.

such tests with many different randomly chosen values for a, then we can have high confidence that n is, in fact, prime.

In summary, the procedure for picking a prime number is as follows.

1. Pick an odd integer n at random (e.g., using a pseudorandom number generator).
2. Pick an integer $a < n$ at random.
3. Perform the probabilistic primality test, such as Miller-Rabin, with a as a parameter. If n fails the test, reject the value n and go to step 1.
4. If n has passed a sufficient number of tests, accept n; otherwise, go to step 2.

This is a somewhat tedious procedure. However, remember that this process is performed relatively infrequently: only when a new pair (PU, PR) is needed.

It is worth noting how many numbers are likely to be rejected before a prime number is found. A result from number theory, known as the prime number theorem, states that the primes near N are spaced on the average one every $\ln(N)$ integers. Thus, on average, one would have to test on the order of $\ln(N)$ integers before a prime is found. Actually, because all even integers can be immediately rejected, the correct figure is $\ln(N)/2$. For example, if a prime on the order of magnitude of 2^{200} were sought, then about $\ln(2^{200})/2 = 70$ trials would be needed to find a prime.

Having determined prime numbers p and q, the process of key generation is completed by selecting a value of e and calculating d or, alternatively, selecting a value of d and calculating e. Assuming the former, then we need to select an e such that $\gcd(\phi(n), e) = 1$ and then calculate $d \equiv e^{-1} \pmod{\phi(n)}$. Fortunately, there is a single algorithm that will, at the same time, calculate the greatest common divisor of two integers and, if the gcd is 1, determine the inverse of one of the integers modulo the other. The algorithm, referred to as the extended Euclid's algorithm, is explained in Chapter 4. Thus, the procedure is to generate a series of random numbers, testing each against $\phi(n)$ until a number relatively prime to $\phi(n)$ is found. Again, we can ask the question: How many random numbers must we test to find a usable number, that is, a number relatively prime to $\phi(n)$? It can be shown easily that the probability that two random numbers are relatively prime is about 0.6; thus, very few tests would be needed to find a suitable integer (see Problem 8.2).

The Security of RSA

Four possible approaches to attacking the RSA algorithm are

- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks:** These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

The defense against the brute-force approach is the same for RSA as for other cryptosystems, namely, to use a large key space. Thus, the larger the number of bits in d, the better. However, because the calculations involved, both in key generation

block (DB). Next, a random seed is generated and passed through another hash function, called the mask generating function (MGF). The resulting hash value is bit-by-bit XORed with DB to produce a maskedDB. The maskedDB is in turn passed through the MGF to form a hash that is XORed with the seed to produce the masked seed. The concatenation of the maskedseed and the maskedDB forms the encoded message EM. Note that the EM includes the padded message, masked by the seed, and the seed, masked by the maskedDB. The EM is then encrypted using RSA.

9.3 RECOMMENDED READING AND WEB SITE

The recommended treatments of encryption listed in Chapter 3 cover public-key as well as symmetric encryption.

[DIFF88] describes in detail the several attempts to devise secure two-key cryptographic algorithms and the gradual evolution of a variety of protocols based on them. [CORM04] provides a concise but complete and readable summary of all of the algorithms relevant to the verification, computation, and cryptanalysis of RSA. [BONE99] discusses various cryptanalytic attacks on RSA. A more recent discussion is [SHAM03].

BONE99 Boneh, D. "Twenty Years of Attacks on the RSA Cryptosystem." *Notices of the American Mathematical Society*, February 1999.

CORM04 Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2004.

DIFF88 Diffie, W. "The First Ten Years of Public-Key Cryptography." *Proceedings of the IEEE*, May 1988.

SHAM03 Shamir, A., and Tromer, E. "On the Cost of Factoring RSA-1024." *CryptoBytes*, Summer 2003. http://www.rsasecurity.com/rsalabs

Recommended Web Site:

RSA Laboratories: Extensive collection of technical material on RSA and other topics in cryptography.

9.4 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

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

"I am afraid, Watson, that your proposal isn't without flaws and at least it needs some additional conditions to be satisfied by F. Let's consider, for instance, the RSA encryption function, that is $F(M) = M^K \bmod N$, K is secret. This function is believed to be one-way, but I wouldn't recommend its use, for example, on the sequence $M = 2, 3, 4, 5, 6, \ldots$."

"But why, Holmes?" Dr. Watson apparently didn't understand. "Why do you think that the resulting sequence $2^K \bmod N, 3^K \bmod N, 4^K \bmod N, \ldots$ is not appropriate for one-time pad encryption if K is kept secret?"

"Because it is—at least partially—predictable, dear Watson, even if K is kept secret. You have said that the cryptanalyst is assumed to know F and the general nature of the sequence. Now let's assume that he will obtain somehow a short segment of the output sequence. In crypto circles this assumption is generally considered to be a viable one. And for this output sequence, knowledge of just the first two elements will allow him to predict quite a lot of the next elements of the sequence, even if not all of them, thus this sequence can't be considered to be cryptographically strong. And with the knowledge of a longer segment he could predict even more of the next elements of the sequence. Look, knowing the general nature of the sequence and its first two elements $2^K \bmod N$ and $3^K \bmod N$, you can easily compute its following elements."

Show how this can be done.

9.12 Show how RSA can be represented by matrices M1, M2, and M3 of Problem 9.1.

9.13 Consider the following scheme:

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

Is this scheme equivalent to RSA? Show why or why not.

9.14 Consider the following scheme by which B encrypts a message for A.

1. A chooses two large primes P and Q that are also relatively prime to $(P - 1)$ and $(Q - 1)$.
2. A publishes $N = PQ$ as its public key.
3. A calculates P' and Q' such that $PP' \equiv 1 \pmod{Q - 1}$ and $QQ' \equiv 1 \pmod{P - 1}$.
4. B encrypts message M as $C = M^N \bmod N$.
5. A finds M by solving $M \equiv C^{P'} \pmod{Q}$ and $M \equiv C^{Q'} \pmod{P}$.

a. Explain how this scheme works.
b. How does it differ from RSA?
c. Is there any particular advantage to RSA compared to this scheme?
d. Show how this scheme can be represented by matrices M1, M2, and M3 of Problem 9.1.

9.15 "This is a very interesting case, Watson," Holmes said. "The young man loves a girl, and she loves him too. However, her father is a strange fellow who insists that his would-be son-in-law must design a simple and secure protocol for an appropriate public-key cryptosystem he could use in his company's computer network. The young man came up with the following protocol for communication between two parties. For example, user A wishing to send message M to user B: (messages exchanged are in the format sender's name, text, receiver's name)"

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

"You can see that the protocol is really simple. But the girl's father claims that the young man has not satisfied his call for a simple protocol, because the proposal contains a certain redundancy and can be further simplified to the following:"

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

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.

The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. Briefly, we can define the discrete logarithm in the following way. Recall from Chapter 8 that a primitive root of a prime number p as one whose powers modulo p generate all the integers from 1 to $p - 1$. That is, if a is a primitive root of the prime number p, then the numbers

$$a \bmod p, a^2 \bmod p, \ldots, a^{p-1} \bmod p$$

are distinct and consist of the integers from 1 through $p - 1$ in some permutation.

For any integer b and a primitive root a of prime number p, we can find a unique exponent i such that

$$b \equiv a^i \pmod{p} \qquad \text{where } 0 \leq i \leq (p - 1)$$

The exponent i is referred to as the **discrete logarithm** of b for the base a mod p. We express this value as $\text{dlog}_{a,p}(b)$. See Chapter 8 for an extended discussion of discrete logarithms.

The Algorithm

Figure 10.1 summarizes the Diffie-Hellman key exchange algorithm. For this scheme, there are two publicly known numbers: a prime number q and an integer α that is a primitive root of q. Suppose the users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{X_A} \bmod q$. Similarly, user B independently selects a random integer $X_B < q$ and computes $Y_B = \alpha^{X_B} \bmod q$. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as $K = (Y_B)^{X_A} \bmod q$ and user B computes the key as $K = (Y_A)^{X_B} \bmod q$. These two calculations produce identical results:

$$\begin{aligned}
K &= (Y_B)^{X_A} \bmod q \\
&= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\
&= (\alpha^{X_B})^{X_A} \bmod q && \text{by the rules of modular arithmetic} \\
&= \alpha^{X_B X_A} \bmod q \\
&= (\alpha^{X_A})^{X_B} \bmod q \\
&= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}$$

The result is that the two sides have exchanged a secret value. Furthermore, because X_A and X_B are private, an adversary only has the following ingredients to work with: q, α, Y_A, and Y_B. Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute

$$X_B = \text{dlog}_{\alpha,q}(Y_B)$$

The adversary can then calculate the key K in the same manner as user B calculates it.

Man-in-the-Middle Attack

The protocol depicted in Figure 10.2 is insecure against a man-in-the-middle attack. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows.

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_B} \bmod q$.
5. Bob transmits Y_B to Alice.
6. Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)^{X_{D1}} \bmod q$.
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key $K1$ and Alice and Darth share secret key $K2$. All future communication between Bob and Alice is compromised in the following way.

1. Alice sends an encrypted message M: E($K2$, M) .
2. Darth intercepts the encrypted message and decrypts it to recover M.
3. Darth sends Bob $E(K1, M)$ or E($K1$, M') , where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates; these topics are explored in Chapters 13 and 14.

10.2 ELGAMAL CRYPTOGRAPHIC SYSTEM

In 1984, T. Elgamal announced a public-key scheme based on discrete logarithms, closely related to the Diffie-Hellman technique [ELGA84, ELGA85]. The ElGamal[2] cryptosystem is used in some form in a number of standards including the digital signature standard (DSS), which is covered in Chapter 13, and the S/MIME e-mail standard (Chapter 18).

[2]For no apparent reason, everyone calls this the ElGamal system although Mr. Elgamal's last name does not have a capital letter G.

system challenges RSA: elliptic curve cryptography (ECC). ECC is showing up in standardization efforts, including the IEEE P1363 Standard for Public-Key Cryptography.

The principal attraction of ECC, compared to RSA, is that it appears to offer equal security for a far smaller key size, thereby reducing processing overhead. On the other hand, although the theory of ECC has been around for some time, it is only recently that products have begun to appear and that there has been sustained cryptanalytic interest in probing for weaknesses. Accordingly, the confidence level in ECC is not yet as high as that in RSA.

ECC is fundamentally more difficult to explain than either RSA or Diffie-Hellman, and a full mathematical description is beyond the scope of this book. This section and the next give some background on elliptic curves and ECC. We begin with a brief review of the concept of abelian group. Next, we examine the concept of elliptic curves defined over the real numbers. This is followed by a look at elliptic curves defined over finite fields. Finally, we are able to examine elliptic curve ciphers.

The reader may wish to review the material on finite fields in Chapter 4 before proceeding.

Abelian Groups

Recall from Chapter 4 that an **abelian group** G, sometimes denoted by $\{G, \bullet\}$, is a set of elements with a binary operation, denoted by $\bullet$, that associates to each ordered pair (a, b) of elements in G an element $(a \bullet b)$ in G, such that the following axioms are obeyed:[3]

(A1) Closure:	If a and b belong to G, then $a \bullet b$ is also in G.
(A2) Associative:	$a \bullet (b \bullet c) = (a \bullet b) \bullet c$ for all a, b, c in G.
(A3) Identity element:	There is an element e in G such that $a \bullet \text{e} = \text{e} \bullet a = a$ for all a in G.
(A4) Inverse element:	For each a in G there is an element a' in G such that $a \bullet a' = a' \bullet a = \text{e}$.
(A5) Commutative:	$a \bullet b = b \bullet a$ for all a, b in G.

A number of public-key ciphers are based on the use of an abelian group. For example, Diffie-Hellman key exchange involves multiplying pairs of nonzero integers modulo a prime number q. Keys are generated by exponentiation over the group, with exponentiation defined as repeated multiplication. For example, $a^k \bmod q = \underbrace{(a \times a \times \ldots \times a)}_{k \text{ times}} \bmod q$. To attack Diffie-Hellman, the attacker must determine k given a and a^k; this is the discrete logarithm problem.

[3]The operator $\bullet$ is generic and can refer to addition, multiplication, or some other mathematical operation.

over GF(2^m), the variables and coefficients all take on values in GF(2^m) and in calculations are performed over GF(2^m). [FERN99] points out that prime curves are best for software applications, because the extended bit-fiddling operations needed by binary curves are not required; and that binary curves are best for hardware applications, where it takes remarkably few logic gates to create a powerful, fast cryptosystem. We examine these two families in this section and the next.

There is no obvious geometric interpretation of elliptic curve arithmetic over finite fields. The algebraic interpretation used for elliptic curve arithmetic over real numbers does readily carry over, and this is the approach we take.

For elliptic curves over Z_p, as with real numbers, we limit ourselves to equations of the form of Equation (10.1), but in this case with coefficients and variables limited to Z_p:

$$y^2 \bmod p = (x^3 + ax + b) \bmod p \tag{10.5}$$

For example, Equation (10.5) is satisfied for $a = 1$, $b = 1$, $x = 9$, $y = 7$, $\alpha = 1$ $p = 23$:

$$7^2 \bmod 23 = (9^3 + 9 + 1) \bmod 23$$
$$49 \bmod 23 = 739 \bmod 23$$
$$3 = 3$$

Now consider the set $E_p(a, b)$ consisting of all pairs of integers (x, y) that satisfy Equation (10.5), together with a point at infinity O. The coefficients a and b and the variables x and y are all elements of Z_p.

For example, let $p = 23$ and consider the elliptic curve $y^2 = x^3 + x + 1$. In this case, $a = b = 1$. Note that this equation is the same as that of Figure 10.4b. The figure shows a continuous curve with all of the real points that satisfy the equation. For the set $E_{23}(1, 1)$, we are only interested in the nonnegative integers in the quadrant from $(0, 0)$ through $(p - 1, p - 1)$ that satisfy the equation mod p. Table 10.1 lists the points (other than O) that are part of $E_{23}(1, 1)$. Figure 10.5 plots the points of $E_{23}(1, 1)$; note that the points, with one exception, are symmetric about $y = 11.5$.

Table 10.1 Points on the Elliptic Curve E_{23} (1,1)

(0, 1)	(6, 4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)

10.16 The following is a first attempt at an elliptic curve signature scheme. We have a global elliptic curve, prime p, and "generator" G. Alice picks a private signing key X_A and forms the public verifying key $Y_A = X_A G$. To sign a message M:

- Alice picks a value k.
- Alice sends Bob M, k and the signature $S = M - kX_A G$.
- Bob verifies that $M = S + kY_A$.

a. Show that this scheme works. That is, show that the verification process produces an equality if the signature is valid.

b. Show that the scheme is unacceptable by describing a simple technique for forging a user's signature on an arbitrary message.

10.17 Here is an improved version of the scheme given in the previous problem. As before, we have a global elliptic curve, prime p, and "generator" G. Alice picks a private signing key X_A and forms the public verifying key $Y_A = X_A G$. To sign a message M:

- Bob picks a value k.
- Bob sends Alice $C_1 = kG$.
- Alice sends Bob M and the signature $S = M - X_A C_1$.
- Bob verifies that $M = S + kY_A$.

a. Show that this scheme works. That is, show that the verification process produces an equality if the signature is valid.

b. Show that forging a message in this scheme is as hard as breaking (ElGamal) elliptic curve cryptography. (Or find an easier way to forge a message?)

c. This scheme has an extra "pass" compared to other cryptosystems and signature schemes we have looked at. What are some drawbacks to this?

appends the resulting hash value to M. Because B possesses S, it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

d. Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.

When confidentiality is not required, method (b) has an advantage over methods (a) and (d), which encrypts the entire message, in that less computation is required. Nevertheless, there has been growing interest in techniques that avoid encryption (Figure 11.2c). Several reasons for this interest are pointed out in [TSUD92].

- Encryption software is relatively slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
- Encryption hardware costs are not negligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.
- Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
- Encryption algorithms may be covered by patents, and there is a cost associated with licensing their use.

More commonly, message authentication is achieved using a **message authentication code (MAC)**, also known as a **keyed hash function**. Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties. A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC. This can then be transmitted with or stored with the protected message. If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the stored MAC value. An attacker who alters the message will be unable to alter the MAC value without knowledge of the secret key. Note that the verifying party also knows who the sending party is because no one else knows the secret key.

Note that the combination of hashing and encryption results in an overall function that is, in fact, a MAC (Figure 11.2b). That is, $E(K, H(M))$ is a function of a variable-length message M and a secret key K, and it produces a fixed-size output that is secure against an opponent who does not know the secret key. In practice, specific MAC algorithms are designed that are generally more efficient than an encryption algorithm.

We discuss MACs in Chapter 12.

Digital Signatures

Another important application, which is similar to the message authentication application, is the **digital signature**. The operation of the digital signature is similar to that of the MAC. In the case of the digital signature, the hash value of a message is encrypted with a user's private key. Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature. In this

of the chaining variable is the hash value. Often, $b > n$; hence the term *compression*. The hash function can be summarized as

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

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

The motivation for this iterative structure stems from the observation by Merkle [MERK89] and Damgard [DAMG89] that if the compression function is collision resistant, then so is the resultant iterated hash function.[1] Therefore, the structure can be used to produce a secure hash function to operate on a message of any length. The problem of designing a secure hash function reduces to that of designing a collision-resistant compression function that operates on inputs of some fixed size.

Cryptanalysis of hash functions focuses on the internal structure of f and is based on attempts to find efficient techniques for producing collisions for a single execution of f. Once that is done, the attack must take into account the fixed value of IV. The attack on f depends on exploiting its internal structure. Typically, as with symmetric block ciphers, f consists of a series of rounds of processing, so that the attack involves analysis of the pattern of bit changes from round to round.

Keep in mind that for any hash function there must exist collisions, because we are mapping a message of length at least equal to twice the block size b (because we must append a length field) into a hash code of length n, where $b \geq n$. What is required is that it is computationally infeasible to find collisions.

The attacks that have been mounted on hash functions are rather complex and beyond our scope here. For the interested reader, [DOBB96] and [BELL97] are recommended.

11.4 HASH FUNCTIONS BASED ON CIPHER BLOCK CHAINING

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

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

This is similar to the CBC technique, but in this case, there is no secret key. As with any hash code, this scheme is subject to the birthday attack, and if the encryption algorithm is DES and only a 64-bit hash code is produced, then the system is vulnerable.

[1] The converse is not necessarily true.

These values are stored in **big-endian** format, which is the most significant byte of a word in the low-address (leftmost) byte position. These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

Step 4 **Process message in 1024-bit (128-word) blocks.** The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure 11.8. The logic is illustrated in Figure 11.9.

Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer. At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1}. Each round t makes use of a 64-bit value W_t, derived from the current 1024-bit block being processed (M_i). These values are derived using a message schedule described subsequently. Each round also makes use of an additive constant K_t, where $0 \leq t \leq 79$ indicates one of the 80 rounds. These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers. The constants provide a "randomized" set of 64-bit patterns, which should eliminate any regularities in the input data. Table 11.4 shows these constants in hexadecimal format (from left to right).

The output of the eightieth round is added to the input to the first round (H_{i-1}) to produce H_i. The addition is done independently for each of the eight

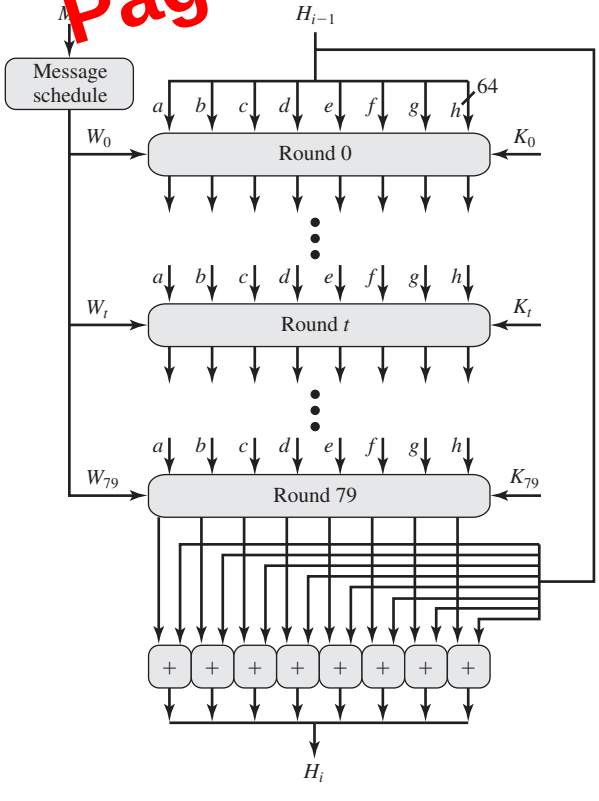


Figure 11.9 SHA-512 Processing of a Single 1024-Bit Block

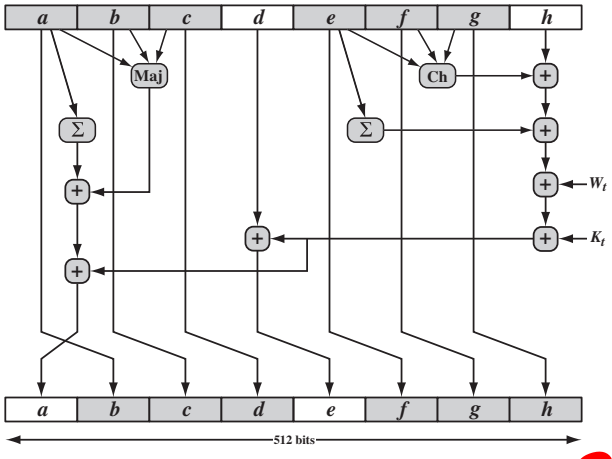


Figure 11.10 Elementary SHA-512 Operation (single round)

It remains to indicate how the 64-bit word values W_t are derived from the 1024-bit message. Figure 11.11 illustrates the mapping. The first 16 values of W_t are taken directly from the 16 words of the current block. The remaining values are defined as

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

where

$$\begin{aligned}
\sigma_0^{512}(x) &= \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x) \\
\sigma_1^{512}(x) &= \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x) \\
\text{ROTR}^n(x) &= \text{circular right shift (rotation) of the 64-bit argument } x \text{ by } n \text{ bits} \\
\text{SHR}^n(x) &= \text{left shift of the 64-bit argument } x \text{ by } n \text{ bits with padding by zeros on the right} \\
+ &= \text{addition modulo } 2^{64}
\end{aligned}$$

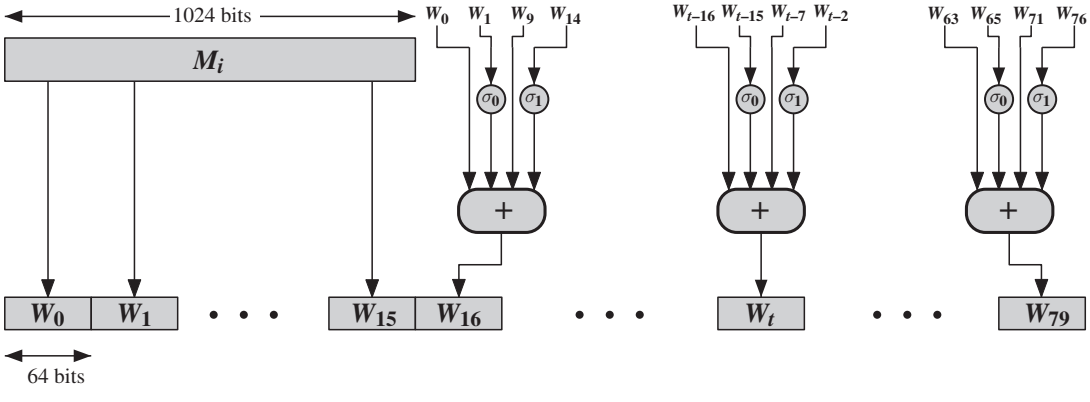


Figure 11.11 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

there are N possible messages with $N >> 2^n$. Furthermore, with a k-bit key, there are 2^k possible keys.

For example, suppose that we are using 100-bit messages and a 10-bit MAC. Then, there are a total of 2^{100} different messages but only 2^{10} different MACs. So, on average, each MAC value is generated by a total of $2^{100}/2^{10} = 2^{90}$ different messages. If a 5-bit key is used, then there are $2^5 = 32$ different mappings from the set of messages to the set of MAC values.

It turns out that, because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption.

The process depicted in Figure 12.4a provides authentication but not confidentiality, because the message as a whole is transmitted in the clear. Confidentiality can be provided by performing message encryption either after (Figure 12.4b) or before (Figure 12.4c) the MAC algorithm. In both these cases, two separate keys are needed, each of which is shared by the sender and the receiver. In the first case, the MAC is calculated with the message as input and is then concatenated to the message. The entire block is then encrypted. In the second case, the message is encrypted first. Then the MAC is calculated using the resulting ciphertext and is concatenated to the ciphertext to form the transmitted block. Typically, it is preferable to tie the authentication directly to the plaintext, so the method of Figure 12.4b is used.

Because symmetric encryption will provide authentication and because it is widely used with readily available products, why not simply use this instead of a separate message authentication code? [DAVI89] suggests three situations in which a message authentication code is used.

1. There are a number of applications in which the same message is broadcast to a number of destinations. Examples are notification to users that the network is now unavailable or an alarm signal in a military control center. It is cheaper and more reliable to have only one destination responsible for monitoring authenticity. Thus, the message must be broadcast in plaintext with an associated message authentication code. The responsible system has the secret key and performs authentication. If a violation occurs, the other destination systems are alerted by a general alarm.
2. Another possible scenario is an exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages. Authentication is carried out on a selective basis, messages being chosen at random for checking.
3. Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having to decrypt it every time, which would be wasteful of processor resources. However, if a message authentication code were attached to the program, it could be checked whenever assurance was required of the integrity of the program.

Three other rationales may be added.

4. For some applications, it may not be of concern to keep messages secret, but it is important to authenticate messages. An example is the Simple Network Management Protocol Version 3 (SNMPv3), which separates the functions of

way of knowing which is the correct key. On average, a total of $2^k/2^n = 2^{(k-n)}$ keys will produce a match. Thus, the opponent must iterate the attack.

- **Round 1**

  Given: $M_1, T_1 = \text{MAC}(K, M_1)$
  Compute $T_i = \text{MAC}(K_i, M_1)$ for all 2^k keys
  Number of matches $\approx 2^{(k-n)}$

- **Round 2**

  Given: $M_2, T_2 = \text{MAC}(K, M_2)$
  Compute $T_i = \text{MAC}(K_i, M_2)$ for the $2^{(k-n)}$ keys resulting from Round 1
  Number of matches $\approx 2^{(k-2\times n)}$

And so on. On average, α rounds will be needed if $k = \alpha \times n$. For example, if an 80-bit key is used and the tab is 32 bits, then the first round will produce about 2^{48} possible keys. The second round will narrow the possible keys to about 2^{16} possibilities. The third round should produce only a single key, which must be the one used by the sender.

If the key length is less than or equal to the tag length, then it is likely that a first round will produce a single match. It is possible that more than one key will produce such a match, in which case the opponent would need to perform the same test on a new (message, tag) pair.

Thus, a brute-force attempt to discover the authentication key is no less effort and may be more effort than that required to discover a decryption key of the same length. However, other attacks that do not require the discovery of the key are possible.

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

$$\Delta(M) = X_1 \oplus X_2 \oplus \cdots \oplus X_m$$
$$\text{MAC}(K, M) = \text{E}(K, \Delta(M))$$

where $\oplus$ is the exclusive-OR (XOR) operation and the encryption algorithm is DES in electronic codebook mode. Thus, the key length is 56 bits, and the tag length is 64 bits. If an opponent observes $\{M \| \text{MAC}(K, M)\}$, a brute-force attempt to determine K will require at least 2^{56} encryptions. But the opponent can attack the system by replacing X_1 through X_{m-1} with any desired values Y_1 through Y_{m-1} and replacing X_m with Y_m, where Y_m is calculated as

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

The opponent can now concatenate the new message, which consists of Y_1 through Y_m, using the original tag to form a message that will be accepted as authentic by the receiver. With this tactic, any message of length $64 \times (m - 1)$ bits can be fraudulently inserted.

Thus, in assessing the security of a MAC function, we need to consider the types of attacks that may be mounted against it. With that in mind, let us state the requirements for the function. Assume that an opponent knows the MAC function

but does not know K. Then the MAC function should satisfy the following requirements.

1. If an opponent observes M and MAC(K, M), it should be computationally infeasible for the opponent to construct a message M' such that
$$\text{MAC}(K, M') = \text{MAC}(K, M)$$
2. MAC(K, M) should be uniformly distributed in the sense that for randomly chosen messages, M and M', the probability that MAC(K, M) = MAC(K, M') is 2^{-n}, where n is the number of bits in the tag.
3. Let M' be equal to some known transformation on M. That is, $M' = \text{f}(M)$. For example, f may involve inverting one or more specific bits. In that case,
$$\Pr[\text{MAC}(K, M) = \text{MAC}(K, M')] = 2^{-n}$$

The first requirement speaks to the earlier example, in which an opponent is able to construct a new message to match a given tag, even though the opponent does not know and does not learn the key. The second requirement deals with the need to thwart a brute-force attack based on chosen plaintext. That is, we assume that the opponent does not know K but does have access to the MAC function and can present messages for MAC generation. Then the opponent could try various messages until finding one that matches a given tag. If the MAC function exhibits uniform distribution, then a brute-force method would require, on average, $2^{(n-1)}$ attempts before finding a message that fits a given tag.

The final requirement dictates that the authentication algorithm should not be weaker with respect to certain parts or bits of the message than others. If this were not the case, then an opponent who had M and MAC(K, M) could attempt variations on M at the known "weak spots" with a likelihood of early success at producing a new message that matched the old tags.

12.4 SECURITY OF MACS

Just as with encryption algorithms and hash functions, we can group attacks on MACs into two categories: brute-force attacks and cryptanalysis.

Brute-Force Attacks

A brute-force attack on a MAC is a more difficult undertaking than a brute-force attack on a hash function because it requires known message-tag pairs. Let us see why this is so. To attack a hash code, we can proceed in the following way. Given a fixed message x with n-bit hash code $h = \text{H}(x)$, a brute-force method of finding a collision is to pick a random bit string y and check if $\text{H}(y) = \text{H}(x)$. The attacker can do this repeatedly off line. Whether an off-line attack can be used on a MAC algorithm depends on the relative size of the key and the tag.

To proceed, we need to state the desired security property of a MAC algorithm, which can be expressed as follows.

- **Computation resistance:** Given one or more text-MAC pairs [x_i, MAC(K, x_i)], it is computationally infeasible to compute any text-MAC pair [x, MAC(K, x)] for any new input $x \neq x_i$.

a MAC derived from a cryptographic hash function. The motivations for this interest are

1. Cryptographic hash functions such as MD5 and SHA generally execute faster in software than symmetric block ciphers such as DES.
2. Library code for cryptographic hash functions is widely available.

With the development of AES and the more widespread availability of code for encryption algorithms, these considerations are less significant, but hash-based MACs continue to be widely used.

A hash function such as SHA was not designed for use as a MAC and cannot be used directly for that purpose, because it does not rely on a secret key. There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm. The approach that has received the most support is HMAC [BELL96a, BELL96b]. HMAC has been issued as RFC 2104, has been chosen as the mandatory-to-implement MAC for IP security, and is used in other Internet protocols, such as SSL. HMAC has also been issued as a NIST standard (FIPS 198).

HMAC Design Objectives

RFC 2104 lists the following design objectives for HMAC.

- To use, without modifications, available hash functions. In particular, to use hash functions that perform well in software and for which code is freely and widely available.
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

The first two objectives are important to the acceptability of HMAC. HMAC treats the hash function as a "black box." This has two benefits. First, an existing implementation of a hash function can be used as a module in implementing HMAC. In this way, the bulk of the HMAC code is prepackaged and ready to use without modification. Second, if it is ever desired to replace a given hash function in an HMAC implementation, all that is required is to remove the existing hash function module and drop in the new module. This could be done if a faster hash function were desired. More important, if the security of the embedded hash function were compromised, the security of HMAC could be retained simply by replacing the embedded hash function with a more secure one (e.g., replacing SHA-2 with SHA-3).

The last design objective in the preceding list is, in fact, the main advantage of HMAC over other proposed hash-based schemes. HMAC can be proven secure

In the second attack, the attacker is looking for two messages M and M' that produce the same hash: $H(M) = H(M')$. This is the birthday attack discussed in Chapter 11. We have shown that this requires a level of effort of $2^{n/2}$ for a hash length of n. On this basis, the security of MD5 is called into question, because a level of effort of 2^{64} looks feasible with today's technology. Does this mean that a 128-bit hash function such as MD5 is unsuitable for HMAC? The answer is no, because of the following argument. To attack MD5, the attacker can choose any set of messages and work on these off line on a dedicated computing facility to find a collision. Because the attacker knows the hash algorithm and the default IV, the attacker can generate the hash code for each of the messages that the attacker generates. However, when attacking HMAC, the attacker cannot generate message/code pairs off line because the attacker does not know K. Therefore, the attacker must observe a sequence of messages generated by HMAC under the same key and perform the attack on these known messages. For a hash code length of 128 bits, this requires 2^{64} observed blocks (2^{72} bits) generated using the same key. On a 1-Gbps link, one would need to observe a continuous stream of messages with no change in key for about 150,000 years in order to succeed. Thus, if speed is a concern, it is fully acceptable to use MD5 rather than SHA-1 as the embedded hash function for HMAC.

12.6 MACs BASED ON BLOCK CIPHERS: DAA AND CMAC

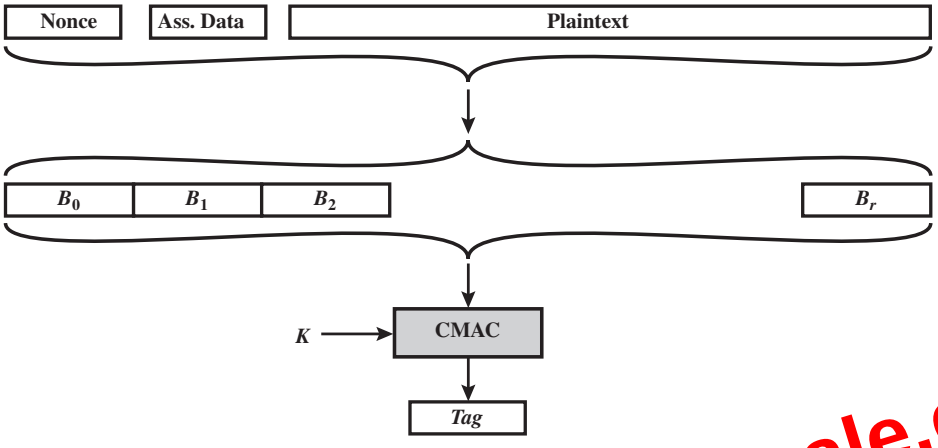
In this section, we look at two MACs that are based on the use of a block cipher mode of operation. We begin with an older algorithm, the Data Authentication Algorithm (DAA), which is now obsolete. Then we examine CMAC, which is designed to overcome the deficiencies of DAA.

Data Authentication Algorithm

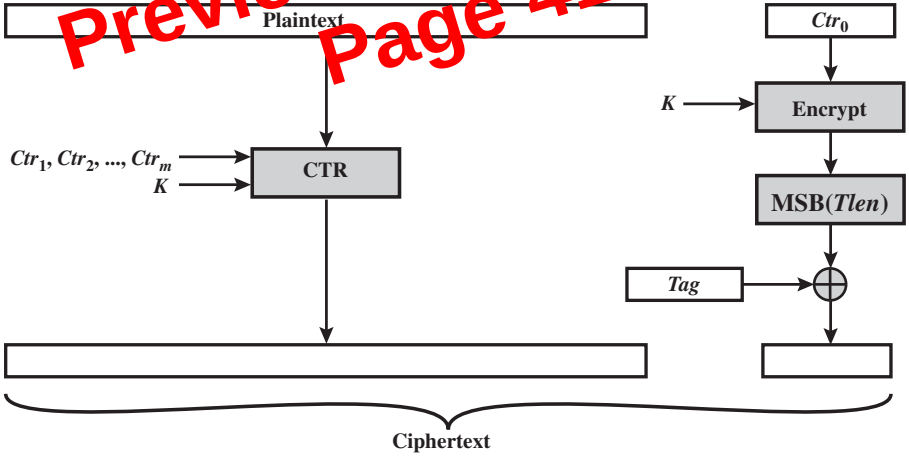
The **Data Authentication Algorithm** (DAA), based on DES, has been one of the most widely used MACs for a number of years. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17). However, as we discuss subsequently, security weaknesses in this algorithm have been discovered, and it is being replaced by newer and stronger algorithms.

The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES (Figure 6.4) with an initialization vector of zero. The data (e.g., message, record, file, or program) to be authenticated are grouped into contiguous 64-bit blocks: $D_1, D_2, \ldots, D_N$. If necessary, the final block is padded on the right with zeroes to form a full 64-bit block. Using the DES encryption algorithm E and a secret key K, a data authentication code (DAC) is calculated as follows (Figure 12.7).

$$
\begin{aligned}
O_1 &= \mathrm{E}(K, D) \\
O_2 &= \mathrm{E}(K, [D_2 \oplus O_1]) \\
O_3 &= \mathrm{E}(K, [D_3 \oplus O_2]) \\
&\bullet \\
&\bullet \\
&\bullet \\
O_N &= \mathrm{E}(K, [D_N \oplus O_{N-1}])
\end{aligned}
$$



(a) Authentication



(b) Encryption

Figure 12.9 Counter with Cipher Block Chaining-Message Authentication Code (CCM)

3. For $i = 1$ to r, do $Y_i = \mathrm{E}(K, (B_i \oplus Y_{i-1}))$.
4. Set $T = \mathrm{MSB}_{Tlen}(Y_r)$.
5. Apply the counter generation function to generate the counter blocks $Ctr_0, Ctr_1, \ldots, Ctr_m$, where $m = \lceil Plen/128 \rceil$.
6. For $j = 0$ to m, do $S_j = \mathrm{E}(K, Ctr_j)$.
7. Set $S = S_1 \| S_2 \| \cdots \| S_m$.
8. Return $C = (P \oplus \mathrm{MSB}_{Plen}(S)) \| (T \oplus \mathrm{MSB}_{Tlen}(S_0))$.

We noted in Chapters 7 and 10 that, because an encryption algorithm produces an apparently random output, it can serve as the basis of a (PRNG). Similarly, a hash function or MAC produces apparently random output and can be used to build a PRNG. Both ISO standard 18031 (*Random Bit Generation*) and NIST SP 800-90 (*Recommendation for Random Number Generation Using Deterministic Random Bit Generators*) define an approach for random number generation using a cryptographic hash function. SP 800-90 also defines a random number generator based on HMAC. We look at these two approaches in turn.

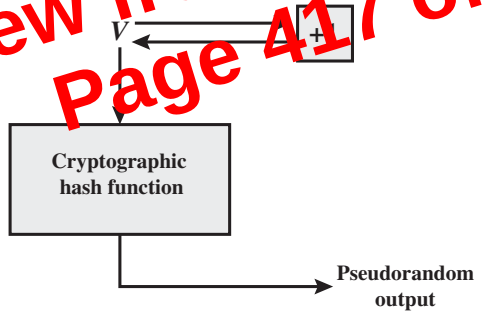
PRNG Based on Hash Function

Figure 12.12a shows the basic strategy for a hash-based PRNG specified in SP 800-90 and ISO 18031. The algorithm takes as input:

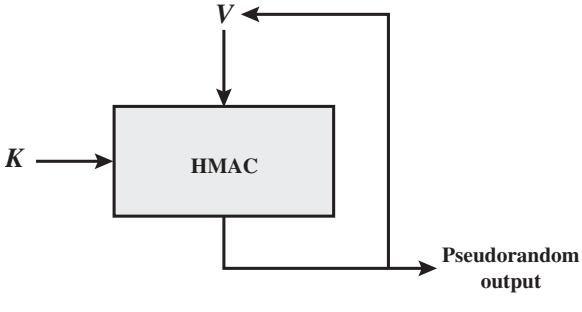
V = seed

seedlen = bit length of $V \geq k + 64$, where k is a desired security level expressed in bits

n = desired number of output bits



(a) PRNG using cryptographic hash function



(b) PRNG using HMAC

Figure 12.12 Basic Structure of Hash-Based PRNGs (SP 800-90)

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

Finish the explanation.

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

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

13.4 The DSS document includes a recommended algorithm for testing a number for primality.

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

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

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

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

Algorithm 1	Algorithm 2

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

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

14.1 SYMMETRIC KEY DISTRIBUTION USING SYMMETRIC ENCRYPTION

For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Furthermore, frequent key changes are usually desirable to limit the amount of data compromised if an attacker learns the key. Therefore, the strength of any cryptographic system rests with the *key distribution technique*, a term that refers to the means of delivering a key to two parties who wish to exchange data without allowing others to see the key. For two parties A and B, key distribution can be achieved in a number of ways, as follows:

1. A can select a key and physically deliver it to B.
2. A third party can select the key and physically deliver it to A and B.
3. If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.
4. If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B.

Options 1 and 2 call for manual delivery of a key. For link encryption, this is a reasonable requirement, because each link encryption device is going to be exchanging data only with its partner on the other end of the link. However, for **end-to-end encryption** over a network, manual delivery is awkward. In a distributed system, any given host or terminal may need to engage in exchanges with many other hosts and terminals over time. Thus, each device needs a number of keys supplied dynamically. The problem is especially difficult in a wide-area distributed system.

The scale of the problem depends on the number of communicating pairs that must be supported. If end-to-end encryption is done at a network or IP level, then a key is needed for each pair of hosts on the network that wish to communicate. Thus, if there are N hosts, the number of required keys is $[N(N - 1)]/2$. If encryption is done at the application level, then a key is needed for every pair of users or processes that require communication. Thus, a network may have hundreds of hosts but thousands of users and processes. Figure 14.1 illustrates the magnitude of the key distribution task for end-to-end encryption.[1] A network using node-level encryption with 1000 nodes would conceivably need to distribute as many as half a million keys. If that same network supported 10,000 applications, then as many as 50 million keys may be required for application-level encryption.

Returning to our list, option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys will be revealed. Furthermore, the initial distribution of potentially millions of keys still must be made.

[1]Note that this figure uses a log-log scale, so that a linear graph indicates exponential growth. A basic review of log scales is in the math refresher document at the Computer Science Student Resource Site at WilliamStallings.com/StudentSupport.html.

"Hmm, please describe the communication protocol used on the network." Holmes opened his eyes, thus proving that he had followed Lestrade's talk with an attention that contrasted with his sleepy look.

"Well, the protocol is as follows. Each node N of the network has been assigned a unique secret key K_n. This key is used to secure communication between the node and a trusted server. That is, all the keys are stored also on the server. User A, wishing to send a secret message M to user B, initiates the following protocol:

1. A generates a random number R and sends to the server his name A, destination B, and $E(K_a, R)$.
2. Server responds by sending $E(K_b, R)$ to A.
3. A sends $E(R, M)$ together with $E(K_b, R)$ to B.
4. B knows K_b, thus decrypts $E(K_b, R)$, to get R and will subsequently use R to decrypt $E(R, M)$ to get M.

You see that a random key is generated every time a message has to be sent. I admit the man could intercept messages sent between the top-secret trusted nodes, but I see no way he could decrypt them."

"Well, I think you have your man, Lestrade. The protocol isn't secure because the server doesn't authenticate users who send him a request. Apparently designers of the protocol have believed that sending $E(K_x, R)$ implicitly authenticates user X as the sender, as only X (and the server) knows K_x. But you know that $E(K_x, R)$ can be intercepted and later replayed. Once you understand where the hole is, you will be able to obtain enough evidence by monitoring the man's use of the computer he has accessed. Most likely he works as follows. After intercepting $E(K_a, R)$ and $E(R, M)$ (see steps 1 and 3 of the protocol), the man, let's denote him as Z, will continue by pretending to be A and ...

Finish the sentence for Holmes.

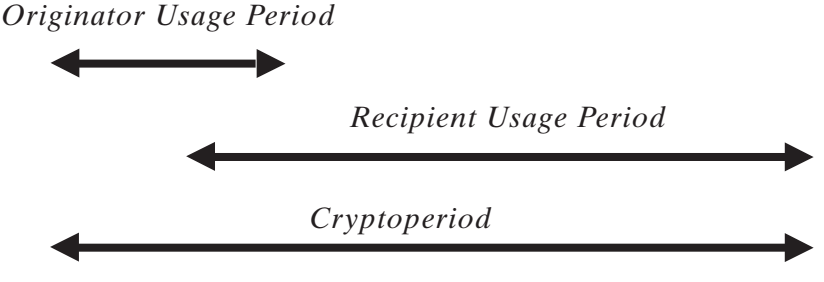
14.3 The 1988 version of X.509 lists properties that RSA keys must satisfy to be secure given current knowledge about the difficulty of factoring large numbers. The discussion concludes with a constraint on the public exponent and the modulus n:

> It must be ensured that $e > \log_2(n)$ to prevent attack by taking the eth root mod n to disclose the plaintext.

Although the constraint is correct, the reason given for requiring it is incorrect. What is wrong with the reason given and what is the correct reason?

14.4 Find at least one intermediate certification authority's certificate and one trusted root certification authority's certificate on your computer (e.g. in the browser). Print screenshots of both the general and details tab for each certificate.

14.5 NIST defines the term cryptoperiod as the time span during which a specific key is authorized for use or in which the keys for a given system or application may remain in effect. One document on key management uses the following time diagram for a shared secret key.



Explain the overlap by giving an example application in which the originator's usage period for the shared secret key begins before the recipient's usage period and also ends before the recipients usage period.

14.6 Consider the following protocol, designed to let A and B decide on a fresh, shared session key K'_{AB}. We assume that they already share a long-term key K_{AB}.

1. $A \rightarrow B$:A, N_A.
2. $B \rightarrow A$:E(K_{AB}, [N_A, K'_{AB}])
3. $A \rightarrow B$:E(K'_{AB}, N_A)

a. We first try to understand the protocol designer's reasoning:

—Why would A and B believe after the protocol ran that they share K'_{AB} with the other party?

—Why would they believe that this shared key is fresh?

In both cases, you should explain both the reasons of both A and B, so your answer should complete the sentences

A believes that she shares K'_{AB} with B since...

B believes that he shares K'_{AB} with A since...

A believes that K'_{AB} is fresh since...

B believes that K'_{AB} is fresh since...

b. Assume now that A starts a run of this protocol with C. However, the connection is intercepted by the adversary C. Show how C can start a new run of the protocol using reflection, causing A to believe that she has agreed on a fresh key with B (in spite of the fact that she has only been communicating with C). Thus, in particular, the belief in a) is false.

c. Propose a modification of the protocol that prevents this attack.

14.7 What are the core components of a PKI? Briefly describe each component.

14.8 Explain the problems with key management and how it affects symmetric cryptography.

Note: The remaining problems deal with the a cryptographic product developed by IBM, which is briefly described in a document at this book's Web site (IBMCrypto.pdf). Try these problems after reviewing the document.

14.9 What is the effect of adding the instruction EMKi

$$\text{EMK}_i\text{: } X \rightarrow \text{E}(KMH_i, X)\ i = 0, 1$$

14.10 Suppose N different systems use the IBM Cryptographic Subsystem with host master keys KMH[i](i = 1, 2, ... N). Devise a method for communicating between systems without requiring the system to either share a common host master key or to divulge their individual host master keys. *Hint*: each system needs three variants of its host master key.

14.11 The principal objective of the IBM Cryptographic Subsystem is to protect transmissions between a terminal and the processing system. Devise a procedure, perhaps adding instructions, which will allow the processor to generate a session key KS and distribute it to Terminal i and Terminal j without having to store a key-equivalent variable in the host.

CHAPTER 15

USER AUTHENTICATION

The process of verifying an identity claimed by or for a system entity. An authentication process consists of two steps:

- **Identification step:** Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.)
- **Verification step:** Presenting or generating authentication information that corroborates the binding between the entity and the identifier.

obtain or guess Alice's password, then the combination of Alice's user ID and password enables administrators to set up Alice's access permissions and audit her activity. Because Alice's ID is not secret, system users can send her e-mail, but because her password is secret, no one can pretend to be Alice.

In essence, identification is the means by which a user provides a claimed identity to the system; user authentication is the means of establishing the validity of the claim. Note that user authentication is distinct from message authentication. As defined in Chapter 12, message authentication is a procedure that allows communicating parties to verify that the contents of a received message have not been altered and that the source is authentic. This chapter is concerned solely with user authentication.

There are four general means of authenticating a user's identity, which can be used alone or in combination:

- **Something the individual knows:** Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions.
- **Something the individual possesses:** Examples include cryptographic keys, electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a *token*.
- **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina, and face.
- **Something the individual does (dynamic biometrics):** Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

All of these methods, properly implemented and used, can provide secure user authentication. However, each method has problems. An adversary may be able to guess or steal a password. Similarly, an adversary may be able to forge or steal a token. A user may forget a password or lose a token. Furthermore, there is a significant administrative overhead for managing password and token information on systems and securing such information on systems. With respect to biometric authenticators, there are a variety of problems, including dealing with false positives and false negatives, user acceptance, cost, and convenience. For network-based user authentication, the most important methods involve cryptographic keys and something the individual knows, such as a password.

Mutual Authentication

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. This topic was examined in Chapter 14. There, the focus was key distribution. We return to this topic here to consider the wider implications of authentication.

Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness. To prevent masquerade and to prevent compromise of session keys, essential identification and session-key information must be communicated in encrypted form. This requires the prior existence of secret or public keys that can be used for this purpose. The second issue, timeliness, is important because of the threat of message replays. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

[GONG93] lists the following examples of **replay attacks**:

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

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

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

It can be argued (e.g., [LAM92a]) that the timestamp approach should not be used for connection-oriented applications because of the inherent difficulties with this technique. First, some sort of protocol is needed to maintain synchronization among the various processor clocks. This protocol must be both fault tolerant, to

Motivation

If a set of users is provided with dedicated personal computers that have no network connections, then a user's resources and files can be protected by physically securing each personal computer. When these users instead are served by a centralized time-sharing system, the time-sharing operating system must provide the security. The operating system can enforce access-control policies based on user identity and use the logon procedure to identify users.

Today, neither of these scenarios is typical. More common is a distributed architecture consisting of dedicated user workstations (clients) and distributed or centralized servers. In this environment, three approaches to security can be envisioned.

1. Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).
2. Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.
3. Require the user to prove his or her identity for each service invoked. Also require that servers prove their identity to clients.

In a small, closed environment in which all systems are owned and operated by a single organization, the first or perhaps the second strategy may suffice.[6] But in a more open environment in which network connections to other machines are supported, the third approach is needed to protect user information and resources housed at the server. Kerberos supports this third approach. Kerberos assumes a distributed client/server architecture and employs one or more Kerberos servers to provide an authentication service.

The first published report on Kerberos [STEI88] listed the following requirements.

- **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- **Reliable:** For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture with one system able to back up another.
- **Transparent:** Ideally, the user should not be aware that authentication is taking place beyond the requirement to enter a password.
- **Scalable:** The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture.

To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78], which was discussed in Section 15.2.

---

[6]However, even a closed environment faces the threat of attack by a disgruntled employee.

may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

With these ground rules in place, we can describe the mechanism as follows (Figure 15.2): A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm). The client can then apply to the remote TGS for service-granting ticket for the desired server in the realm of the remote TGS.

The details of the exchanges illustrated in Figure 15.2 are as follows (compare Table 15.1).

(1) C → AS: $ID_c \| ID_{tgs} \| TS_1$

(2) AS → C: $E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

(3) C → TGS: $ID_{tgsrem} \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C: $E(K_{c,tgs}, [K_{c,\,tgsrem} \| ID_{tgsrem} \| TS_4 \| Ticket_{tgsrem}])$

(5) C → TGS_{rem}: $ID_{vrem} \| Ticket_{tgsrem} \| Authenticator_c$

(6) TGS_{rem} → C: $E(K_{c,tgsrem}, [K_{c,\,vrem} \| ID_{vrem} \| TS_6 \| Ticket_{vrem}])$

(7) C → V_{rem}: $Ticket_{vrem} \| Authenticator_c$

The ticket presented to the remote server (V_{rem}) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are N realms, then there must be $N(N - 1)/2$ secure key exchanges so that each Kerberos realm can interoperate with all other Kerberos realms.

Kerberos Version 5

Kerberos version 5 is specified in RFC 4120 and provides a number of improvements over version 4 [KOHL94]. To begin, we provide an overview of the changes from version 4 to version 5 and then look at the version 5 protocol.

Differences between Versions 4 and 5 Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies. Let us briefly summarize the improvements in each area.[8]

[8]The following discussion follows the presentation in [KOHL94].

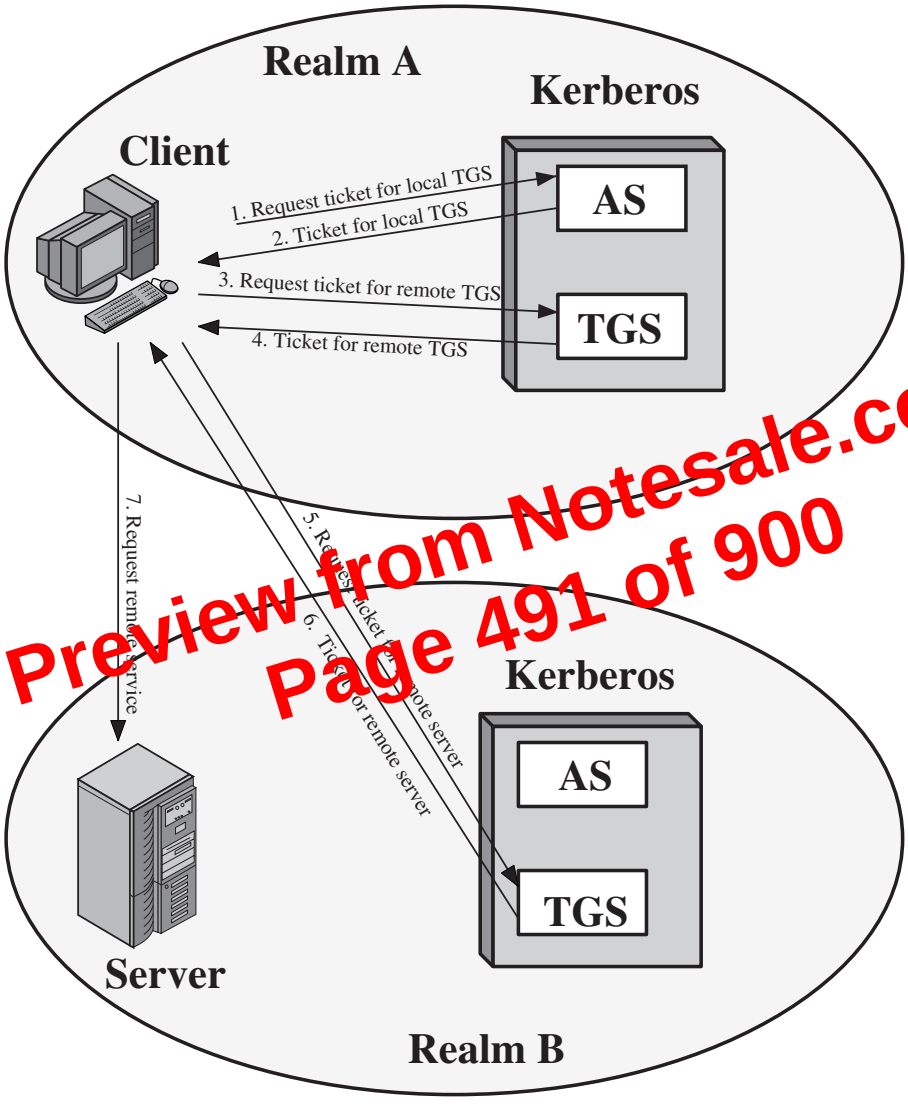


Figure 15.2 Request for Service in Another Realm

Kerberos version 4 was developed for use within the Project Athena environment and, accordingly, did not fully address the need to be of general purpose. This led to the following **environmental shortcomings**.

1. **Encryption system dependence:** Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of concern. In version 5, ciphertext is tagged with an encryption-type identifier so that any encryption technique may be used. Encryption keys are tagged with a type and a length, allowing the same key to be used in different

15.4 REMOTE USER AUTHENTICATION USING ASYMMETRIC ENCRYPTION

Mutual Authentication

In Chapter 14, we presented one approach to the use of public-key encryption for the purpose of session-key distribution (Figure 14.8). This protocol assumes that each of the two parties is in possession of the current public key of the other. It may not be practical to require this assumption.

A protocol using timestamps is provided in [DENN81]:

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

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

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

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

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

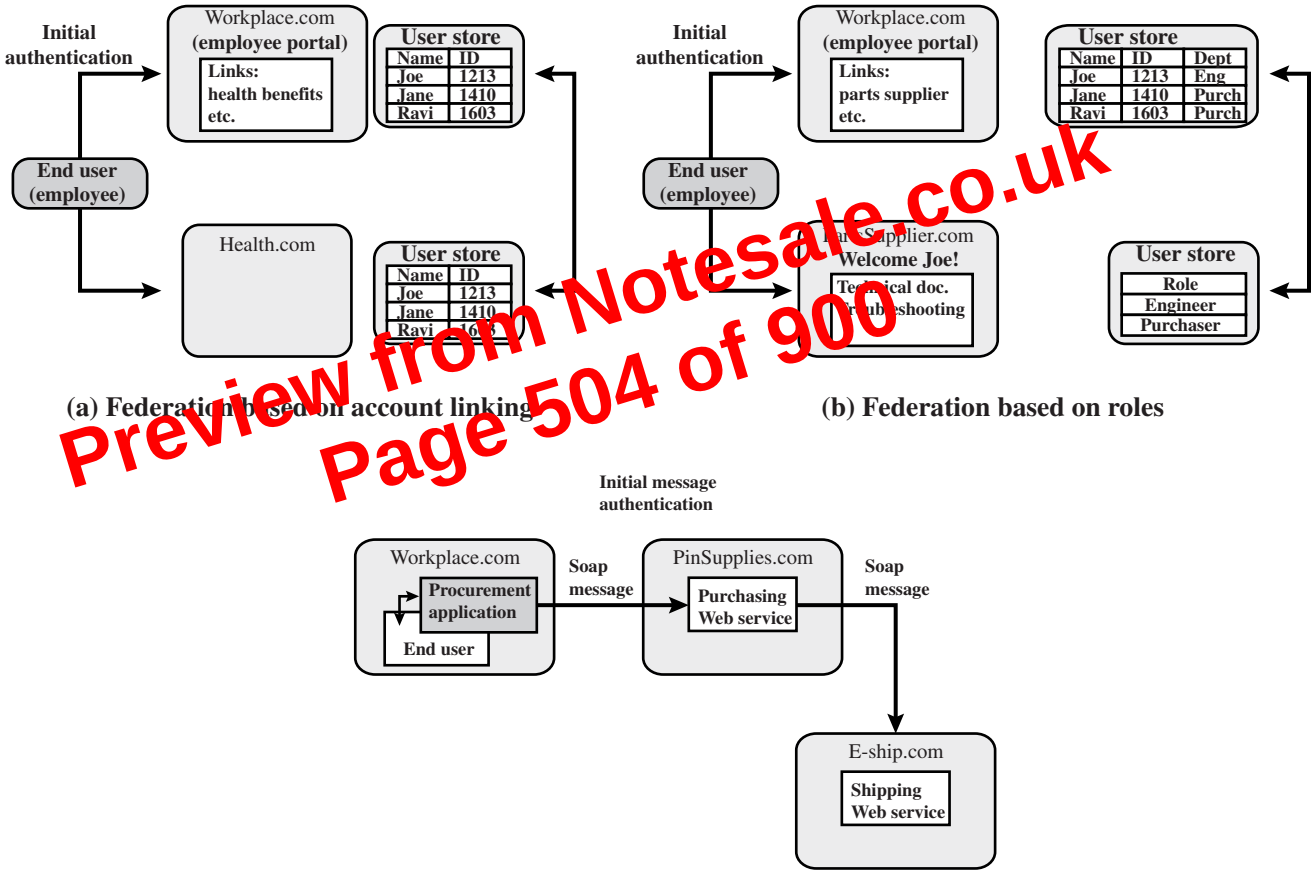
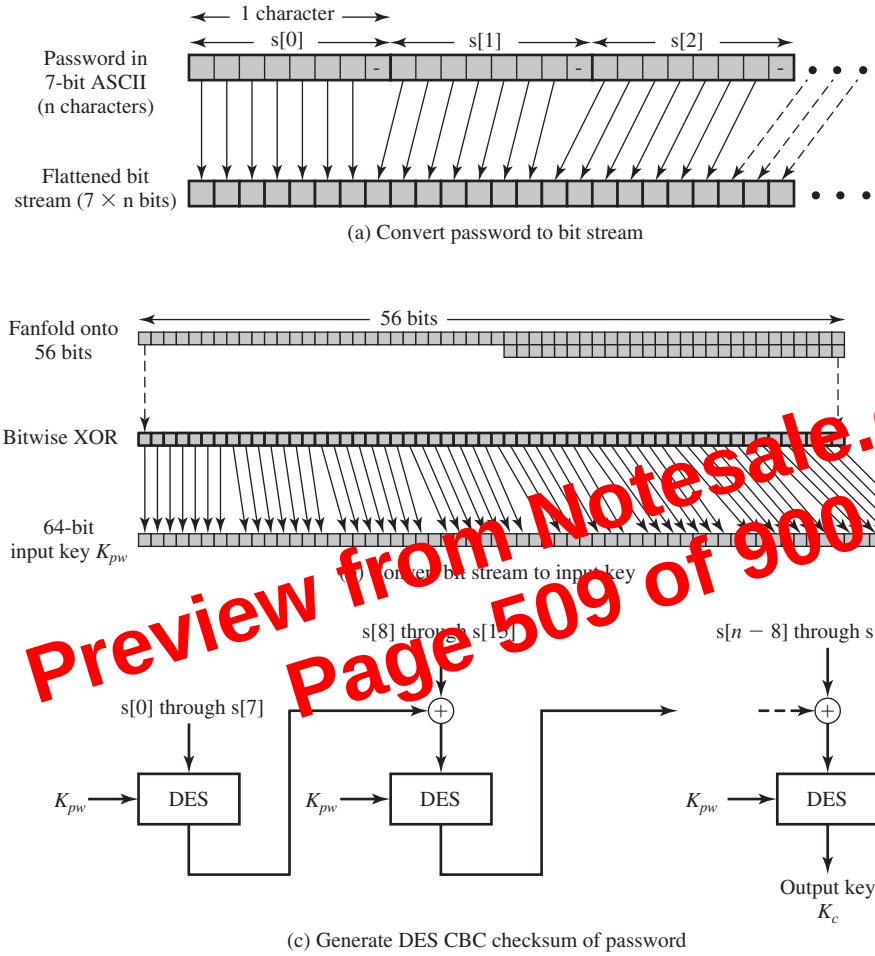


Figure 15.5 Federated Identity Scenarios



(a) Convert password to bit stream

(c) Generate DES CBC checksum of password

Figure 15.6 Generation of Encryption Key from Password

Next, the bit string is compacted to 56 bits by aligning the bits in "fanfold" fashion and performing a bitwise XOR. For example, if the bit string is of length 59, then

$$\begin{aligned} b[55] &= b[55] \oplus b[56] \\ b[54] &= b[54] \oplus b[57] \\ b[53] &= b[53] \oplus b[58] \end{aligned}$$

This creates a 56-bit DES key. To conform to the expected 64-bit key format, the string is treated as a sequence of eight 7-bit blocks and is mapped into eight 8-bit blocks to form an input key K_{pw}.

Finally, the original password is encrypted using the cipher block chaining (CBC) mode of DES with key K_{pw}. The last 64-bit block returned from this process, known as the CBC checksum, is the output key associated with this password.

The entire algorithm can be viewed as a hash function that maps an arbitrary password into a 64-bit hash code.

16.3 TRANSPORT LAYER SECURITY

TLS is an IETF standardization initiative whose goal is to produce an Internet standard version of SSL. TLS is defined as a Proposed Internet Standard in RFC 5246. RFC 5246 is very similar to SSLv3. In this section, we highlight the differences.

Version Number

The TLS Record Format is the same as that of the SSL Record Format (Figure 16.4), and the fields in the header have the same meanings. The one difference is in version values. For the current version of TLS, the major version is 3 and the minor version is 3.

Message Authentication Code

There are two differences between the SSLv3 and TLS MAC schemes: the actual algorithm and the scope of the MAC calculation. TLS makes use of the HMAC algorithm defined in RFC 2104. Recall from Chapter 12 that HMAC is defined as

$$\text{HMAC}_K(M) = \text{H}[(K^+ \oplus \text{opad}) \| \text{H}[(K^+ \oplus \text{ipad}) \| M]]$$

where

H	=	embedded hash function (for TLS, either MD5 or SHA-1)
M	=	message input to HMAC
K^+	=	secret key padded with zeros on the left so that the result is equal to the block length of the hash code (for MD5 and SHA-1, block length = 512 bits)
ipad	=	00110110 (36 in hexadecimal) repeated 64 times (512 bits)
opad	=	01011100 (5C in hexadecimal) repeated 64 times (512 bits)

SSLv3 uses the same algorithm, except that the padding bytes are concatenated with the secret key rather than being XORed with the secret key padded to the block length. The level of security should be about the same in both cases.

For TLS, the MAC calculation encompasses the fields indicated in the following expression:

```
MAC(MAC_write_secret, seq_num || TLSCompressed.type ||
    TLSCompressed.version || TLSCompressed.length ||
    TLSCompressed.fragment)
```

The MAC calculation covers all of the fields covered by the SSLv3 calculation, plus the field `TLSCompressed.version`, which is the version of the protocol being employed.

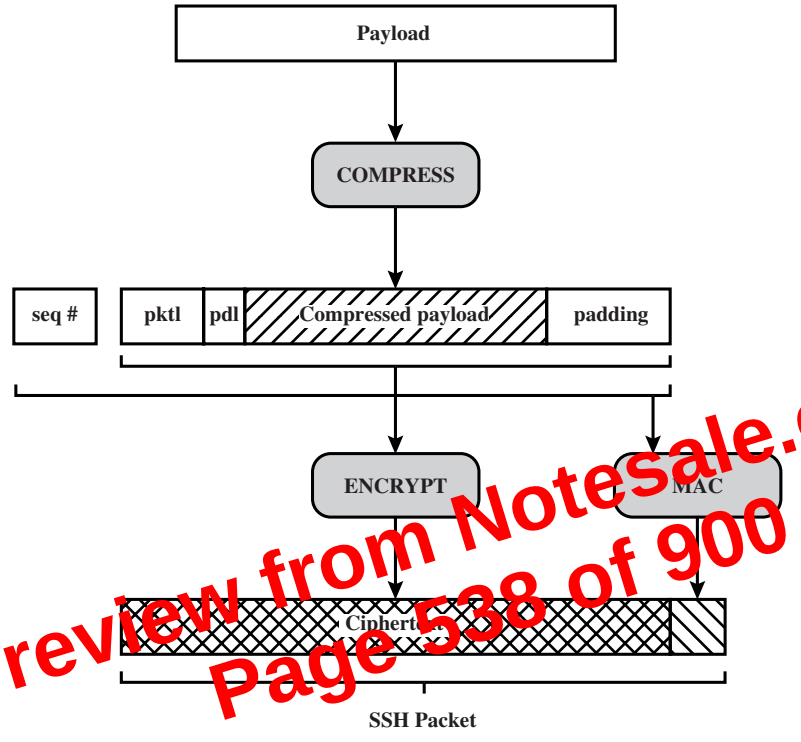


Figure 16.10 SSH Transport Layer Protocol Packet Formation

The next step is **key exchange**. The specification allows for alternative methods of key exchange, but at present, only two versions of Diffie-Hellman key exchange are specified. Both versions are defined in RFC 2409 and require only one packet in each direction. The following steps are involved in the exchange. In this, C is the client; S is the server; p is a large safe prime; g is a generator for a subgroup of GF(p); q is the order of the subgroup; `V_S` is S's identification string; `V_C` is C's identification string; `K_S` is S's public host key; `I_C` is C's `SSH_MSG_KEXINIT` message and `I_S` is S's `SSH_MSG_KEXINIT` message that have been exchanged before this part begins. The values of p, g, and q are known to both client and server as a result of the algorithm selection negotiation. The hash function `hash()` is also decided during algorithm negotiation.

1. C generates a random number $x(1 < x < q)$ and computes $e = g^x \bmod p$. C sends e to S.
2. S generates a random number $y(0 < y < q)$ and computes $f = g^y \bmod p$. S receives e. It computes $K = e^y \bmod p$, $H = \text{hash}(\text{V\_C} \| \text{V\_S} \| \text{I\_C} \| \text{I\_S} \| \text{K\_S} \| e \| f \| \text{K})$, and signature s on H with its private host key. S sends $(\text{K\_S} \| f \| s)$ to C. The signing operation may involve a second hashing operation.

16.6 RECOMMENDED READING AND WEB SITES

[RESC01] is a good detailed treatment of SSL and TLS. [BARR05] provides a thorough treatment of SSH. The original version (SSH-1) of SSH was introduced in [YLON96].

BARR05 Barrett, D.; Silverman, R.; and Byrnes, R. *SSH The Secure Shell: The Definitive Guide.* Sebastopol, CA: O'Reilly, 2005.

RESC01 Rescorla, E. *SSL and TLS: Designing and Building Secure Systems.* Reading, MA: Addison-Wesley, 2001.

YLON96 Ylonen, T. "SSH - Secure Login Connections over the Internet." *Proceedings, Sixth USENIX Security Symposium*, July 1996.

Recommended Web Sites

- **Transport Layer Security Charter:** Latest RFCs and Internet drafts for TLS.
- **OpenSSL Project:** Project to develop open-source SSL and TLS software. Site includes documents and links.

16.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

Alert protocol	HTTPS (HTTP over SSL)	Secure Socket Layer (SSL)
Change Cipher Spec protocol	Master Secret	Transport Layer Security (TLS)
Handshake protocol	Secure Shell (SSH)	

Review Questions

16.1 What are the advantages of each of the three approaches shown in Figure 16.1?
16.2 What protocols comprise SSL?
16.3 What is the difference between an SSL connection and an SSL session?
16.4 List and briefly define the parameters that define an SSL session state.
16.5 List and briefly define the parameters that define an SSL session connection.
16.6 What services are provided by the SSL Record Protocol?
16.7 What steps are involved in the SSL Record Protocol transmission?
16.8 What is the purpose of HTTPS?
16.9 For what applications is SSH useful?
16.10 List and briefly define the SSH protocols.

IEEE 802.11 Services

IEEE 802.11 defines nine services that need to be provided by the wireless LAN to achieve functionality equivalent to that which is inherent to wired LANs. Table 17.2 lists the services and indicates two ways of categorizing them.

1. The service provider can be either the station or the DS. Station services are implemented in every 802.11 station, including AP stations. Distribution services are provided between BSSs; these services may be implemented in an AP or in another special-purpose device attached to the distribution system.
2. Three of the services are used to control IEEE 802.11 LAN access and confidentiality. Six of the services are used to support delivery of MSDUs between stations. If the MSDU is too large to be transmitted in a single MPDU, it may be fragmented and transmitted in a series of MPDUs.

Following the IEEE 802.11 document, we next discuss the services in an order designed to clarify the operation of an IEEE 802.11 ESS network. **MSDU delivery**, which is the basic service, already has been mentioned. Services related to security are introduced in Section 17.2.

Distribution of Messages within a DS The two services involved with the distribution of messages within a DS are distribution and integration. **Distribution** is the primary service used by stations to exchange MPDUs when the MPDUs must traverse the DS to get from a station in one BSS to a station in another BSS. For example, suppose a frame is to be sent from station 2 (STA 2) to station 7 (STA 7) in Figure 17.3. The frame is sent from STA 2 to AP 1, which is the AP for this BSS. The AP gives the frame to the DS, which has the job of directing the frame to the AP associated with STA 7 in the target BSS. AP 2 receives the frame and forwards it to STA 7. How the message is transported through the DS is beyond the scope of the IEEE 802.11 standard.

If the two stations that are communicating are within the same BSS, then the distribution service logically goes through the single AP of that BSS.

Table 17.2 IEEE 802.11 Services

Service	Provider	Used to support
Association	Distribution system	MSDU delivery
Authentication	Station	LAN access and security
Deauthentication	Station	LAN access and security
Disassociation	Distribution system	MSDU delivery
Distribution	Distribution system	MSDU delivery
Integration	Distribution system	MSDU delivery
MSDU delivery	Station	MSDU delivery
Privacy	Station	LAN access and security
Reassociation	Distribution system	MSDU delivery

- **Support for navigation among cards and decks:** WML includes provisions for event handling, which is used for navigation or executing scripts.

In an HTML-based Web browser, a user navigates by clicking on links. At a WML-capable mobile device, a user interacts with cards, moving forward and back through the deck.

WAP Architecture

Figure 17.13, from the WAP architecture document, illustrates the overall stack architecture implemented in a WAP client. In essence, this is a five-layer model. Each layer provides a set of functions and/or services to other services and applications through a set of well-defined interfaces. Each of the layers of the architecture is accessible by the layers above, as well as by other services and applications. Many of the services in the stack may be provided by more than one protocol. For example, either HTTP or WSP may provide the Hypermedia Transfer service.

Common to all five layers are sets of services that are accessible by multiple layers. These common services fall into two categories: security services and service discovery.

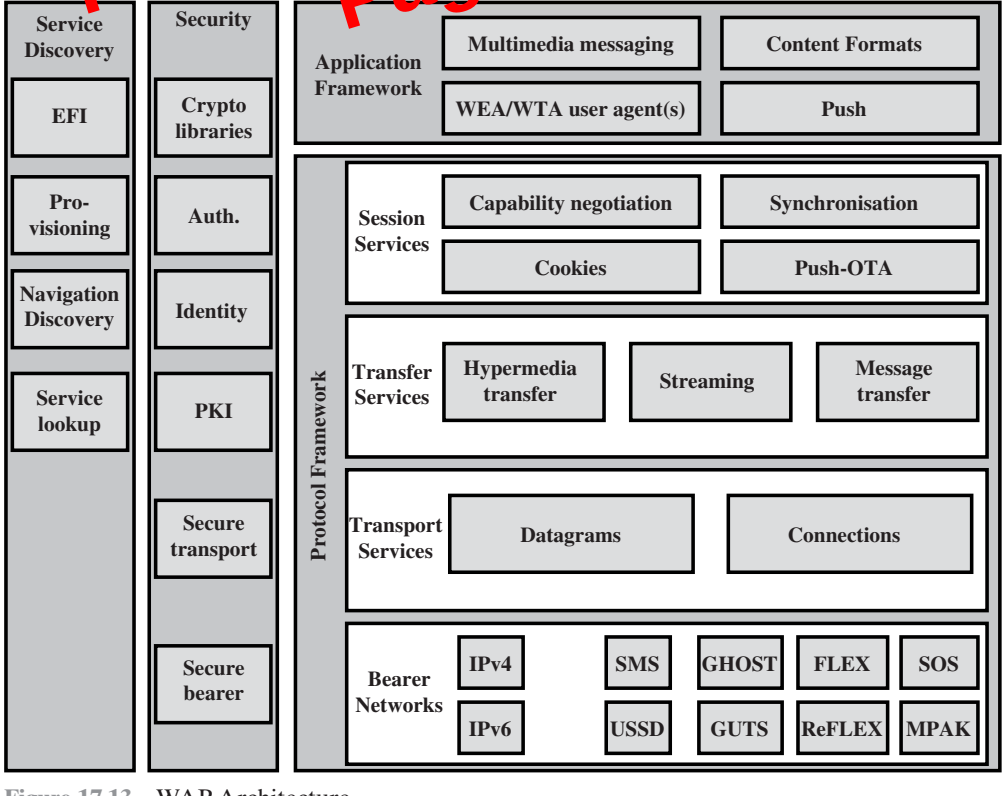


Figure 17.13 WAP Architecture

that are intended to ease the task of developing applications and devices supported by WAP. The major elements of the WAE model (Figure 17.13) are

- **WAE user agents:** Software that executes in the user's wireless device and that provides specific functionality (e.g., display content) to the end user.
- **Wireless telephony applications (WTA):** A collection of telephony-specific extensions for call and feature control mechanisms that provide authors advanced mobile network services. Using WTA, applications developers can use the microbrowser to originate telephone calls and to respond to events from the telephone network.
- **Standard content encoding:** Defined to allow a WAE user agent (e.g., a browser) to conveniently navigate Web content. On the server side are content generators. These are applications (or services) on origin servers (e.g., CGI scripts) that produce standard content formats in response to requests from user agents in the mobile terminal. WAE does not specify any standard content generators but expects that there will be a variety available running on typical HTTP origin servers commonly used in WWW today.
- **Push:** A service to receive such transmissions from the server, i.e., transmissions that are not in response to a Web client request but are sent on the initiative of the server. This service is supported by the Push-OTA (Push Over The Air) session service.
- **Multimedia messaging:** Provides for the transfer and processing of multimedia messages, such as e-mail and instant messages, to WAP devices.

WAP Protocol Architecture

The WAP architecture illustrated in Figure 17.13 dictates a collection of services at each level and provides interface specifications at the boundary between each pair of layers. Because several of the services in the WAP stack can be provided using different protocols based on the circumstances, there are more than one possible stack configurations. Figure 17.14 depicts a common protocol stack configuration in which a WAP client device connects to a Web server via a WAP gateway. This configuration is common with devices that implement version 1 of the WAP specification but is also used in version 2 devices (WAP2) if the bearer network does not support TCP/IP.

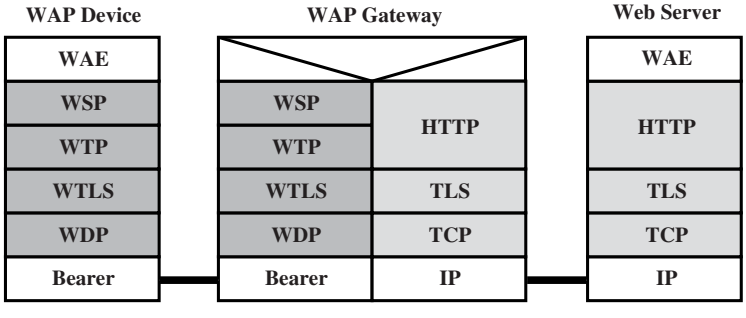


Figure 17.14 WTP 1.x Gateway

- **Connection end:** Whether this entity is considered a client or a server in this secure session.
- **Bulk cipher algorithm:** Includes the key size of this algorithm, how much of that key is secret, whether it is a block or stream cipher, and the block size of the cipher (if appropriate).
- **MAC algorithm:** Includes the size of the key used for MAC calculation and the size of the hash which is returned by the MAC algorithm.
- **Compression algorithm:** Includes all information the algorithm requires to do compression.
- **Master secret:** A 20-byte secret shared between the client and server.
- **Client random:** A 16-byte value provided by the client.
- **Server random:** A 16-byte value provided by the server.
- **Sequence number mode:** Which scheme is used to communicate sequence numbers in this secure connection.
- **Key refresh:** Defines how often some connection state parameters (encryption key, MAC secret, and IV) are updated. New keys are calculated at every $n = 2^{key\_refresh}$ messages, that is, when the sequence number is $0, 2^n, 3^n$, etc.

WTLS Protocol Architecture

WTLS is not a single protocol but rather two layers of protocols, as illustrated in Figure 17.15. The WTLS Record Protocol provides basic security services to various higher-layer protocols. In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of WTLS. Three higher-layer protocols are defined as part of WTLS: the Handshake Protocol, The Change Cipher Spec Protocol, and the Alert Protocol. These WTLS-specific protocols are used in the management of WTLS exchanges and are examined subsequently in this section.

WTLS Record Protocol The WTLS Record Protocol takes user data from the next higher layer (WTP, WTLS Handshake Protocol, WTLS Alert Protocol, and WTLS Change Cipher Spec Protocol) and encapsulates these data in a PDU. The following steps occur (Figure 17.16).

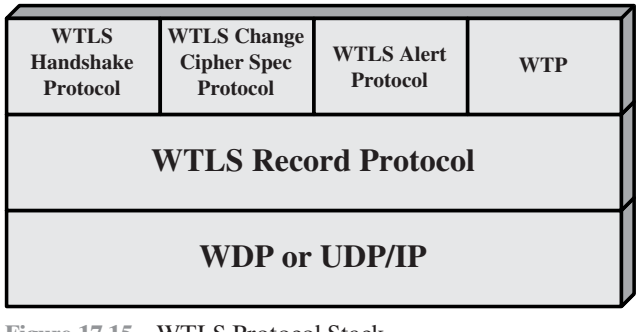
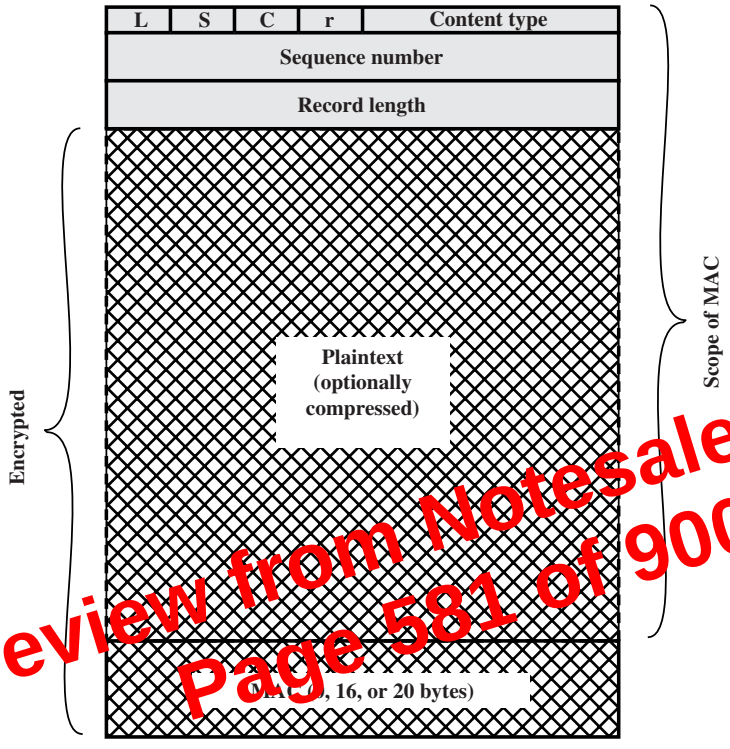


Figure 17.15 WTLS Protocol Stack



r = reserved
C = cipher spec indicator
S = sequence number field indicator
L = record length field indicator
MAC = message authentication code

Figure 17.17 WTLS Record Format

part of HMAC, and cryptographic attributes, such as MAC code size. There are two states associated with each session. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created.

The Change Cipher Spec Protocol is one of the three WTLS-specific protocols that use the WTLS Record Protocol, and it is the simplest. This protocol consists of a single message, which consists of a single byte with the value 1. The sole purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection. Thus, when the Change Cipher Spec message arrives, the sender of the message sets the current write state to the pending state and the receiver sets the current read state to the pending state.

Alert Protocol The Alert Protocol is used to convey WTLS-related alerts to the peer entity. As with other applications that use WTLS, alert messages are compressed and encrypted, as specified by the current state.

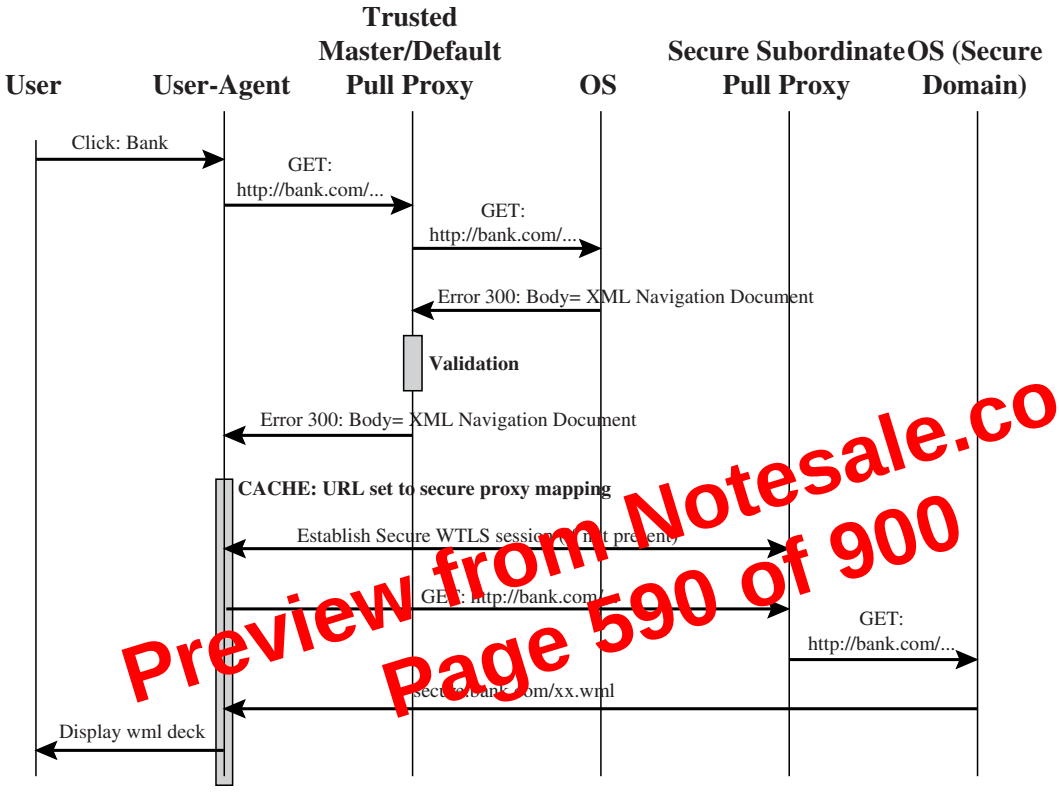


Figure 17.22 WAP Transport Layer End-to-End Security Example

17.6 RECOMMENDED READING AND WEB SITES

The IEEE 802.11 and WiFi specifications are covered in more detail in [STAL07]. A good book-length treatment is [ROSH04]. [FRAN07] is an excellent, detailed treatment of IEEE 802.11i. [CHEN05] provides an overview of IEEE 802.11i.

CHEN05 Chen, J.; Jiang, M.; and Liu, Y. "Wireless LAN Security and IEEE 802.11i." *IEEE Wireless Communications*, February 2005.

FRAN07 Frankel, S.; Eydt, B.; Owens, L.; and Scarfone, K. *Establishing Wireless Robust Security Networks: A Guide to IEEE 802.11i.* NIST Special Publication SP 800-97, February 2007.

ROSH04 Roshan, P., and Leary, J. *802.11 Wireless LAN Fundamentals*. Indianapolis: Cisco Press, 2004.

STAL07 Stallings, W. *Data and Computer Communications, Eighth Edition.* Upper Saddle River, NJ: Prentice Hall, 2007.

Despite the refusal of VADM Poindexter and LtCol North to appear, the Board's access to other sources of information filled much of this gap. The FBI provided documents taken from the files of the National Security Advisor and relevant NSC staff members, including messages from the PROF system between VADM Poindexter and LtCol North. The PROF messages were conversations by computer, written at the time events occurred and presumed by the writers to be protected from disclosure. In this sense, they provide a first-hand, contemporaneous account of events.

—The Tower Commission Report to President Reagan on the Iran-Contra Affair, 1987

KEY POINTS

- PGP is an open-source, freely available software package for e-mail security. It provides authentication through the use of digital signature, confidentiality through the use of symmetric block encryption, compression using the ZIP algorithm, and e-mail compatibility using the radix-64 encoding scheme.
- PGP incorporates tools for developing a public-key trust model and public-key certificate management.
- S/MIME is an Internet standard approach to e-mail security that incorporates the same functionality as PGP.
- DKIM is a specification used by e-mail providers for cryptographically signing e-mail messages on behalf of the source domain.

In virtually all distributed environments, electronic mail is the most heavily used network-based application. Users expect to be able to, and do, send e-mail to others who are connected directly or indirectly to the Internet, regardless of host operating system or communications suite. With the explosively growing reliance on e-mail, there grows a demand for authentication and confidentiality services. Two schemes stand out as approaches that enjoy widespread use: Pretty Good Privacy (PGP) and S/MIME. Both are examined in this chapter. The chapter closes with a discussion of DomainKeys Identified Mail.

18.1 PRETTY GOOD PRIVACY

PGP is a remarkable phenomenon. Largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. In essence, Zimmermann has done the following:

- **User ID:** Typically, this will be the user's e-mail address (e.g., stallings@acm.org). However, the user may choose to associate a different name with each pair (e.g., Stallings, WStallings, WilliamStallings, etc.) or to reuse the same User ID more than once.

The private-key ring can be indexed by either User ID or Key ID; later we will see the need for both means of indexing.

Although it is intended that the private-key ring be stored only on the machine of the user that created and owns the key pairs and that it be accessible only to that user, it makes sense to make the value of the private key as secure as possible. Accordingly, the private key itself is not stored in the key ring. Rather, this key is encrypted using CAST-128 (or IDEA or 3DES). The procedure is as follows:

1. The user selects a passphrase to be used for encrypting private keys.
2. When the system generates a new public/private key pair using RSA, it asks the user for the passphrase. Using SHA-1, a 160-bit hash code is generated from the passphrase, and the passphrase is discarded.
3. The system encrypts the private key using CAST-128 with the 128 bits of the hash code as the key. The hash code is then discarded, and the encrypted private key is stored in the private-key ring.

Subsequently, when a user accesses the private-key ring to retrieve a private key, he or she must supply the passphrase. PGP will retrieve the encrypted private key, generate the hash code of the passphrase, and decrypt the encrypted private key using CAST-128 with the hash code.

This is a very compact and effective scheme. As in any system based on passwords, the security of this system depends on the security of the password. To avoid the temptation to write it down, the user should use a passphrase that is not easily guessed but that is easily remembered.

Figure 18.4 also shows the general structure of a **public-key ring.** This data structure is used to store public keys of other users that are known to this user. For the moment, let us ignore some fields shown in the figure and describe the following fields.

- **Timestamp:** The date/time when this entry was generated.
- **Key ID:** The least significant 64 bits of the public key for this entry.
- **Public Key:** The public key for this entry.
- **User ID:** Identifies the owner of this key. Multiple user IDs may be associated with a single public key.

The public-key ring can be indexed by either User ID or Key ID; we will see the need for both means of indexing later.

We are now in a position to show how these key rings are used in message transmission and reception. For simplicity, we ignore compression and radix-64 conversion in the following discussion. First consider message transmission (Figure 18.5) and assume that the message is to be both signed and encrypted. The sending PGP entity performs the following steps.

Here is a simple example of a multipart message containing two parts—both consisting of simple text (taken from RFC 2046).

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Sample message
MIME-Version: 1.0
Content-type: multipart/mixed; boundary="simple
boundary"

This is the preamble. It is to be ignored, though it
is a handy place for mail composers to include an
explanatory note to non-MIME conformant readers.
—simple boundary

This is implicitly typed plain ASCII text. It does NOT
end with a linebreak.
—simple boundary
Content-type: text/plain; charset=us-ascii

This is explicitly typed plain ASCII text. It DOES end
with a linebreak.

—simple boundary—
This is the epilogue. It is also to be ignored.
```


There are four subtypes of the multipart type, all of which have the same overall syntax. The **multipart/mixed subtype** is used when there are multiple independent body parts that need to be bundled in a particular order. For the **multipart/parallel subtype,** the order of the parts is not significant. If the recipient's system is appropriate, the multiple parts can be presented in parallel. For example, a picture or text part could be accompanied by a voice commentary that is played while the picture or text is displayed.

For the **multipart/alternative subtype,** the various parts are different representations of the same information. The following is an example:

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary=boundary42

    —boundary42

Content-Type: text/plain; charset=us-ascii

    ...plain text version of message goes here....
```

```
—boundary42
Content-Type: text/enriched

    .... RFC 1896 text/enriched version of same message
goes here ...

—boundary42—
```

In this subtype, the body parts are ordered in terms of increasing preference. For this example, if the recipient system is capable of displaying the message in the text/enriched format, this is done; otherwise, the plain text format is used.

The **multipart/digest subtype** is used when each of the body parts is interpreted as an RFC 5322 message with headers. This subtype enables the construction of a message whose parts are individual messages. For example, the moderator of a group might collect e-mail messages from participants, bundle these messages, and send them out in one encapsulating MIME message.

The **message type** provides a number of important capabilities in MIME. The **message/rfc822 subtype** indicates that the body is an entire message, including header and body. Despite the name of this subtype, the encapsulated message may be not only a simple RFC 5322 message but also any MIME message.

The **message/partial subtype** enables fragmentation of a large message into a number of parts, which must be reassembled at the destination. For this subtype, three parameters are specified in the Content-Type: Message/Partial field: an *id* common to all fragments of the same message, a *sequence number* unique to each fragment, and the *total* number of fragments.

The **message/external-body subtype** indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data. As with the other message types, the message/external-body subtype has an outer header and an encapsulated message with its own header. The only necessary field in the outer header is the Content-Type field, which identifies this as a message/external-body subtype. The inner header is the message header for the encapsulated message. The Content-Type field in the outer header must include an access-type parameter, which indicates the method of access, such as FTP (file transfer protocol).

The **application type** refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application.

MIME Transfer Encodings The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

The MIME standard defines two methods of encoding data. The Content-Transfer-Encoding field can actually take on six values, as listed in Table 18.4. However, three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data. For SMTP transfer, it is safe to use the 7bit form. The 8bit and binary forms may be usable in other mail transport contexts. Another Content-Transfer-Encoding value is x-token,

Digital IDs can also contain other user-supplied information, including

- Address
- E-mail address
- Basic registration information (country, zip code, age, and gender)

VeriSign provides three levels, or classes, of security for public-key certificates, as summarized in Table 18.8. A user requests a certificate online at VeriSign's Web site or other participating Web sites. Class 1 and Class 2 requests are processed on line, and in most cases take only a few seconds to approve. Briefly, the following procedures are used.

- For Class 1 Digital IDs, VeriSign confirms the user's e-mail address by sending a PIN and Digital ID pick-up information to the e-mail address provided in the application.
- For Class 2 Digital IDs, VeriSign verifies the information in the application through an automated comparison with consumer database in addition to

Table 18.8 Verisign Public-Key Certificate Classes

	Class 1	Class 2	Class 3
Summary of Confirmation of Identity	Automated unambiguous name and e-mail address search.	Same as Class 1, plus automated enrollment information check and automated address check.	Same as Class 1, plus personal presence and ID documents plus Class 2 automated ID check for individuals; business records (or filings) for organizations.
IA Private Key Protection	PCA: trustworthy hardware; CA: trustworthy software or trustworthy hardware.	PCA and CA: trustworthy hardware.	PCA and CA: trustworthy hardware.
Certificate Applicant and Subscriber Private Key Protection	Encryption software (PIN protected) recommended but not required.	Encryption software (PIN protected) required.	Encryption software (PIN protected) required; hardware token recommended but not required.
Applications Implemented or Contemplated by Users	Web-browsing and certain e-mail usage.	Individual and intra- and inter-company e-mail, online subscriptions, password replacement, and software validation.	E-banking, corp. database access, personal banking, membership-based online services, content integrity services, e-commerce server, software validation; authentication of LRAAs; and strong encryption for certain servers.

IA = Issuing Authority
CA = Certification Authority
PCA = VeriSign public primary certification authority
PIN = Personal Identification Number
LRAA = Local Registration Authority Administrator

performing all of the checking associated with a Class 1 Digital ID. Finally, confirmation is sent to the specified postal address alerting the user that a Digital ID has been issued in his or her name.

- For Class 3 Digital IDs, VeriSign requires a higher level of identity assurance. An individual must prove his or her identity by providing notarized credentials or applying in person.

Enhanced Security Services

As of this writing, three enhanced security services have been proposed in an Internet draft. The details of these may change, and additional services may be added. The three services are

- **Signed receipts:** A signed receipt may be requested in a `SignedData` object. Returning a signed receipt provides proof of delivery to the originator of a message and allows the originator to demonstrate to a third party that the recipient received the message. In essence, the recipient signs the entire original message plus the original (sender's) signature and appends the new signature to form a new S/MIME message.
- **Security labels:** A security label may be included in the authenticated attributes of a `SignedData` object. A security label is a set of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation. The labels may be used for access control, by indicating which users are permitted access to an object. Other uses include priority (secret, confidential, restricted, and so on) or role based, describing which kind of people can see the information (e.g., patient's health-care team, medical billing agents, etc.).
- **Secure mailing lists:** When a user sends a message to multiple recipients, a certain amount of per-recipient processing is required, including the use of each recipient's public key. The user can be relieved of this work by employing the services of an S/MIME Mail List Agent (MLA). An MLA can take a single incoming message, perform the recipient-specific encryption for each recipient, and forward the message. The originator of a message need only send the message to the MLA with encryption performed using the MLA's public key.

18.3 DOMAINKEYS IDENTIFIED MAIL

DomainKeys Identified Mail (DKIM) is a specification for cryptographically signing e-mail messages, permitting a signing domain to claim responsibility for a message in the mail stream. Message recipients (or agents acting in their behalf) can verify the signature by querying the signer's domain directly to retrieve the appropriate public key and thereby can confirm that the message was attested to by a party in possession of the private key for the signing domain. DKIM is a proposed Internet Standard (RFC 4871: *DomainKeys Identified Mail (DKIM) Signatures*). DKIM has been widely adopted by a range of e-mail providers, including corporations, government agencies, gmail, yahoo, and many Internet Service Providers (ISPs).

2. At the next level are professional senders of bulk spam mail. These attackers often operate as commercial enterprises and send messages on behalf of third parties. They employ more comprehensive tools for attack, including Mail Transfer Agents (MTAs) and registered domains and networks of compromised computers (zombies) to send messages and (in some cases) to harvest addresses to which to send.
3. The most sophisticated and financially motivated senders of messages are those who stand to receive substantial financial benefit, such as from an e-mail-based fraud scheme. These attackers can be expected to employ all of the above mechanisms and additionally may attack the Internet infrastructure itself, including DNS cache-poisoning attacks and IP routing attacks.

Capabilities RFC 4686 lists the following as capabilities that an attacker might have.

1. Submit messages to MTAs and Message Submission Agents (MSAs) at multiple locations in the Internet.
2. Construct arbitrary Message Header fields, including those claiming to be mailing lists, resenders, and other mail agents.
3. Sign messages on behalf of domains under their control.
4. Generate substantial numbers of either unsigned or apparently signed messages that might be used to attempt a denial-of-service attack.
5. Resend messages that may have been previously signed by the domain.
6. Transmit messages using any envelope information desired.
7. Act as an authorized submitter for messages from a compromised computer.
8. Manipulation of IP routing. This could be used to submit messages from specific IP addresses or difficult-to-trace addresses, or to cause diversion of messages to a specific domain.
9. Limited influence over portions of DNS using mechanisms such as cache poisoning. This might be used to influence message routing or to falsify advertisements of DNS-based keys or signing practices.
10. Access to significant computing resources, for example, through the conscription of worm-infected "zombie" computers. This could allow the "bad actor" to perform various types of brute-force attacks.
11. Ability to eavesdrop on existing traffic, perhaps from a wireless network.

Location DKIM focuses primarily on attackers located outside of the administrative units of the claimed originator and the recipient. These administrative units frequently correspond to the protected portions of the network adjacent to the originator and recipient. It is in this area that the trust relationships required for authenticated message submission do not exist and do not scale adequately to be practical. Conversely, within these administrative units, there are other mechanisms (such as authenticated message submission) that are easier to deploy and more likely to be used than DKIM. External "bad actors" are usually attempting to exploit the "any-to-any" nature of e-mail that motivates most recipient MTAs to accept messages from anywhere for delivery to their local domain. They may generate messages without

Before a message is signed, a process known as canonicalization is performed on both the header and body of the RFC 5322 message. Canonicalization is necessary to deal with the possibility of minor changes in the message made en route, including character encoding, treatment of trailing white space in message lines, and the "folding" and "unfolding" of header lines. The intent of canonicalization is to make a minimal transformation of the message (for the purpose of signing; the message itself is not changed, so the canonicalization must be performed again by the verifier) that will give it its best chance of producing the same canonical value at the receiving end. DKIM defines two header canonicalization algorithms ("simple" and "relaxed") and two for the body (with the same names). The simple algorithm tolerates almost no modification, while the relaxed tolerates common modifications.

The signature includes a number of fields. Each field begins with a tag consisting of a tag code followed by an equals sign and ends with a semicolon. The fields include the following:

- **v** = DKIM version.
- **a** = Algorithm used to generate the signature; must be either rsa-sha1 or rsa-sha256.
- **c** = Canonicalization method used on the header and the body.
- **d** = A domain name used as an identifier to refer to the identity of a responsible person or organization. In DKIM, this identifier is called the Signing Domain IDentifier (SDID). In our example, this field indicates that the sender is using a gmail address.
- **s** = In order that different keys may be used in different circumstances for the same signing domain (allowing expiration of old keys, separate departmental signing, or the like), DKIM defines a selector (a name associated with a key), which is used by the verifier to retrieve the proper key during signature verification.
- **h** = Signed Header fields. A colon-separated list of header field names that identify the header fields presented to the signing algorithm. Note that in our example above, the signature covers the domainkey-signature field. This refers to an older algorithm (since replaced by DKIM) that is still in use.
- **bh** = The hash of the canonicalized body part of the message. This provides additional information for diagnosing signature verification failures.
- **b** = The signature data in base64 format; this is the encrypted hash code.

18.4 RECOMMENDED READING AND WEB SITES

[LEIB07] provides an overview of DKIM.

LEIB07 Leiba, B., and Fenton, J. "DomainKeys Identified Mail (DKIM): Using Digital Signatures for Domain Verification." *Proceedings of Fourth Conference on E-mail and Anti-Spam (CEAS 07)*, 2007.

Recommended Web Sites:

- **PGP Home Page:** PGP Web site by PGP Corp., the leading PGP commercial vendor.
- **International PGP Home Page:** Designed to promote worldwide use of PGP. Contains documents and links of interest.
- **PGP Charter:** Latest RFCs and Internet drafts for Open Specification PGP.
- **S/MIME Charter:** Latest RFCs and Internet drafts for S/MIME.
- **DKIM:** Website hosted by Mutual Internet Practices Association, this site contains a wide range of documents and information related to DKIM.
- **DKIM Charter:** Latest RFCs and Internet drafts for DKIM.

18.5 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

detached signature	Multipurpose Internet Mail Extensions (MIME)	session key
DomainKeys Identified Mail (DKIM)	Pretty Good Privacy (PGP)	S/MIME
electronic mail	radix 64	trust
		ZIP

Review Questions

18.1 What are the five principal services provided by PGP?
18.2 What is the utility of a detached signature?
18.3 Why does PGP generate a signature before applying compression?
18.4 What is R64 conversion?
18.5 Why is R64 conversion useful for an e-mail application?
18.6 How does PGP use the concept of trust?
18.7 What is RFC 5322?
18.8 What is MIME?
18.9 What is S/MIME?
18.10 What is DKIM?

Problems

18.1 PGP makes use of the cipher feedback (CFB) mode of CAST-128, whereas most symmetric encryption applications (other than key encryption) use the cipher block chaining (CBC) mode. We have

$$\text{CBC: } C_i = \text{E}(K, [C_{i-1} \oplus \text{P}_i]); \quad \text{P}_i = C_{i-1} \oplus \text{D}(K, C_i)$$

$$\text{CFB: } C_i = \text{P}_i \oplus \text{E}(K, C_{i-1}); \quad \text{P}_i = C_i \oplus \text{E}(K, C_{i-1})$$

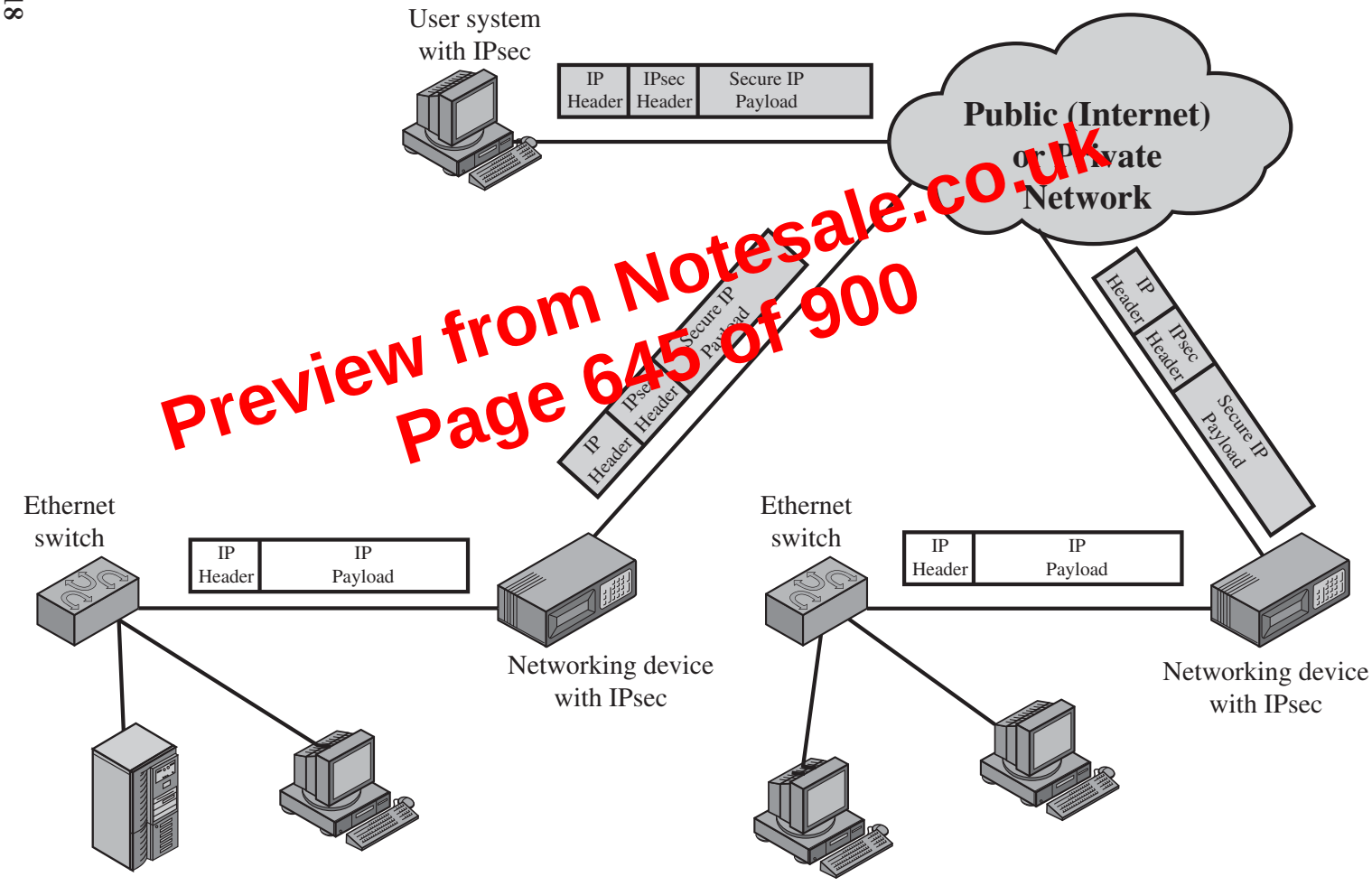


Figure 19.1 An IP Security Scenario

Transport Mode Transport mode provides protection primarily for upper-layer protocols. That is, transport mode protection extends to the payload of an IP packet.[1] Examples include a TCP or UDP segment or an ICMP packet, all of which operate directly above IP in a host protocol stack. Typically, transport mode is used for end-to-end communication between two hosts (e.g., a client and a server, or two workstations). When a host runs AH or ESP over IPv4, the payload is the data that normally follow the IP header. For IPv6, the payload is the data that normally follow both the IP header and any IPv6 extensions headers that are present, with the possible exception of the destination options header, which may be included in the protection.

ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header. AH in transport mode authenticates the IP payload and selected portions of the IP header.

Tunnel Mode Tunnel mode provides protection to the entire IP packet. To achieve this, after the AH or ESP fields are added to the IP packet, the entire packet plus security fields is treated as the payload of new outer IP packet with a new outer IP header. The entire original, inner, packet travels through a tunnel from one point of an IP network to another; no routers along the way are able to examine the inner IP header. Because the original packet is encapsulated, the new, larger packet may have totally different source and destination addresses, adding to the security. Tunnel mode is used when one or both ends of a security association (SA) are a security gateway, such as a firewall or router that implements IPsec. With tunnel mode, a number of hosts on networks behind firewalls may engage in secure communications without implementing IPsec. The unprotected packets generated by such hosts are tunneled through external networks by tunnel mode SAs set up by the IPsec software in the firewall or secure router at the boundary of the local network.

Here is an example of how tunnel mode IPsec operates. Host A on a network generates an IP packet with the destination address of host B on another network. This packet is routed from the originating host to a firewall or secure router at the boundary of A's network. The firewall filters all outgoing packets to determine the need for IPsec processing. If this packet from A to B requires IPsec, the firewall performs IPsec processing and encapsulates the packet with an outer IP header. The source IP address of this outer IP packet is this firewall, and the destination address may be a firewall that forms the boundary to B's local network. This packet is now routed to B's firewall, with intermediate routers examining only the outer IP header. At B's firewall, the outer IP header is stripped off, and the inner packet is delivered to B.

ESP in tunnel mode encrypts and optionally authenticates the entire inner IP packet, including the inner IP header. AH in tunnel mode authenticates the entire inner IP packet and selected portions of the outer IP header.

Table 19.1 summarizes transport and tunnel mode functionality.

[1]In this chapter, the term *IP packet* refers to either an IPv4 datagram or an IPv6 packet.

Table 19.1 Tunnel Mode and Transport Mode Functionality

	Transport Mode SA	Tunnel Mode SA
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts entire inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts entire inner IP packet. Authenticates inner IP packet.

19.2 IP SECURITY POLICY

Fundamental to the operation of IPsec is the concept of a security policy applied to each IP packet that transits from a source to a destination. IPsec policy is determined primarily by the interaction of two databases, the **security association database (SAD)** and the **security policy database (SPD)**. This section provides an overview of these two databases and then summarizes their use during IPsec operation. Figure 19.2 illustrates the relevant relationships.

Security Associations

A key concept that appears in both the authentication and confidentiality mechanisms for IP is the security association (SA). An association is a one-way logical connection between a sender and a receiver that affords security services to the traffic carried on it. If a peer relationship is needed for two-way secure exchange, then two security associations are required. Security services are afforded to an SA for the use of AH or ESP, but not both.

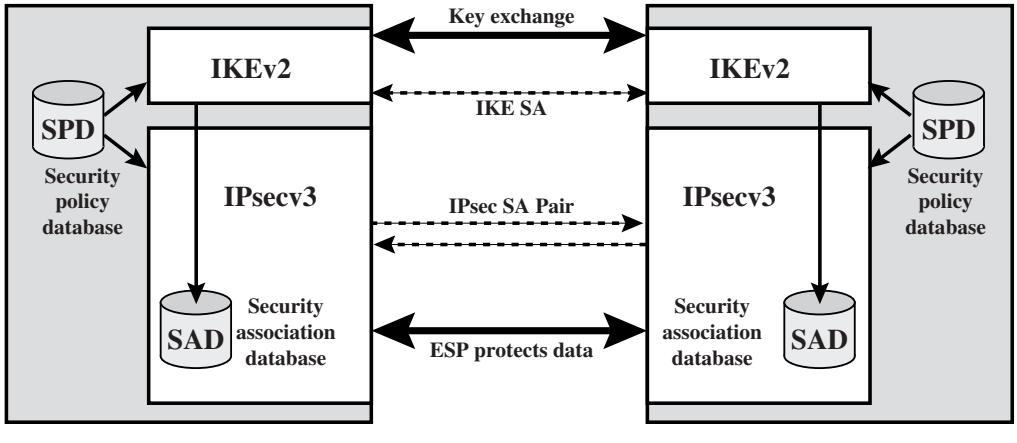


Figure 19.2 IPsec Architecture

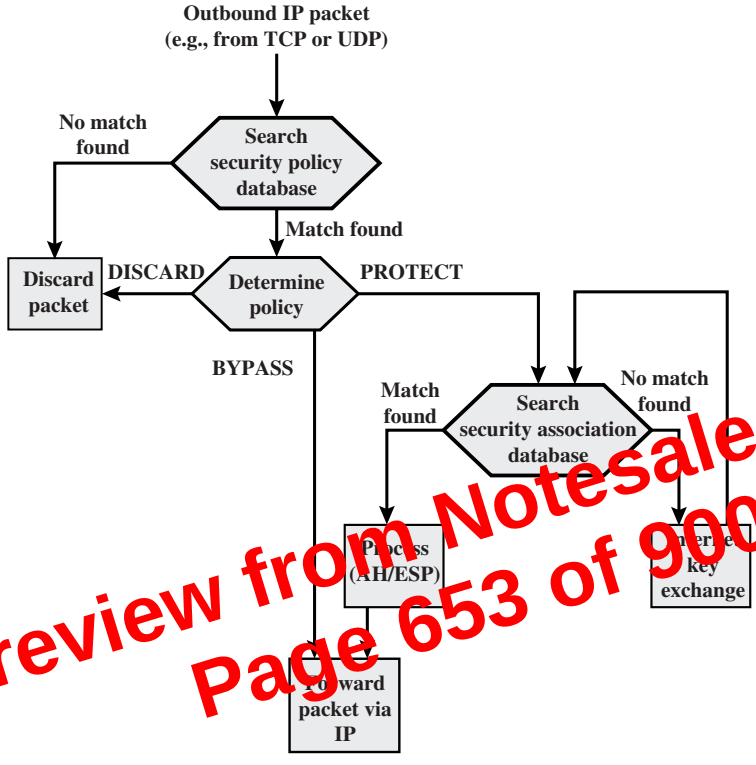


Figure 19.3 Processing Model for Outbound Packets

3. If a match is found, further processing is determined by the first matching entry in the SPD. If the policy for this packet is DISCARD, then the packet is discarded. If the policy is BYPASS, then there is no further IPsec processing; the packet is forwarded to the network for transmission.
4. If the policy is PROTECT, then a search is made of the SAD for a matching entry. If no entry is found, then IKE is invoked to create an SA with the appropriate keys and an entry is made in the SA.
5. The matching entry in the SAD determines the processing for this packet. Either encryption, authentication, or both can be performed, and either transport or tunnel mode can be used. The packet is then forwarded to the network for transmission.

Inbound Packets Figure 19.4 highlights the main elements of IPsec processing for inbound traffic. An incoming IP packet triggers the IPsec processing. The following steps occur:

1. IPsec determines whether this is an unsecured IP packet or one that has ESP or AH headers/trailers, by examining the IP Protocol field (IPv4) or Next Header field (IPv6).

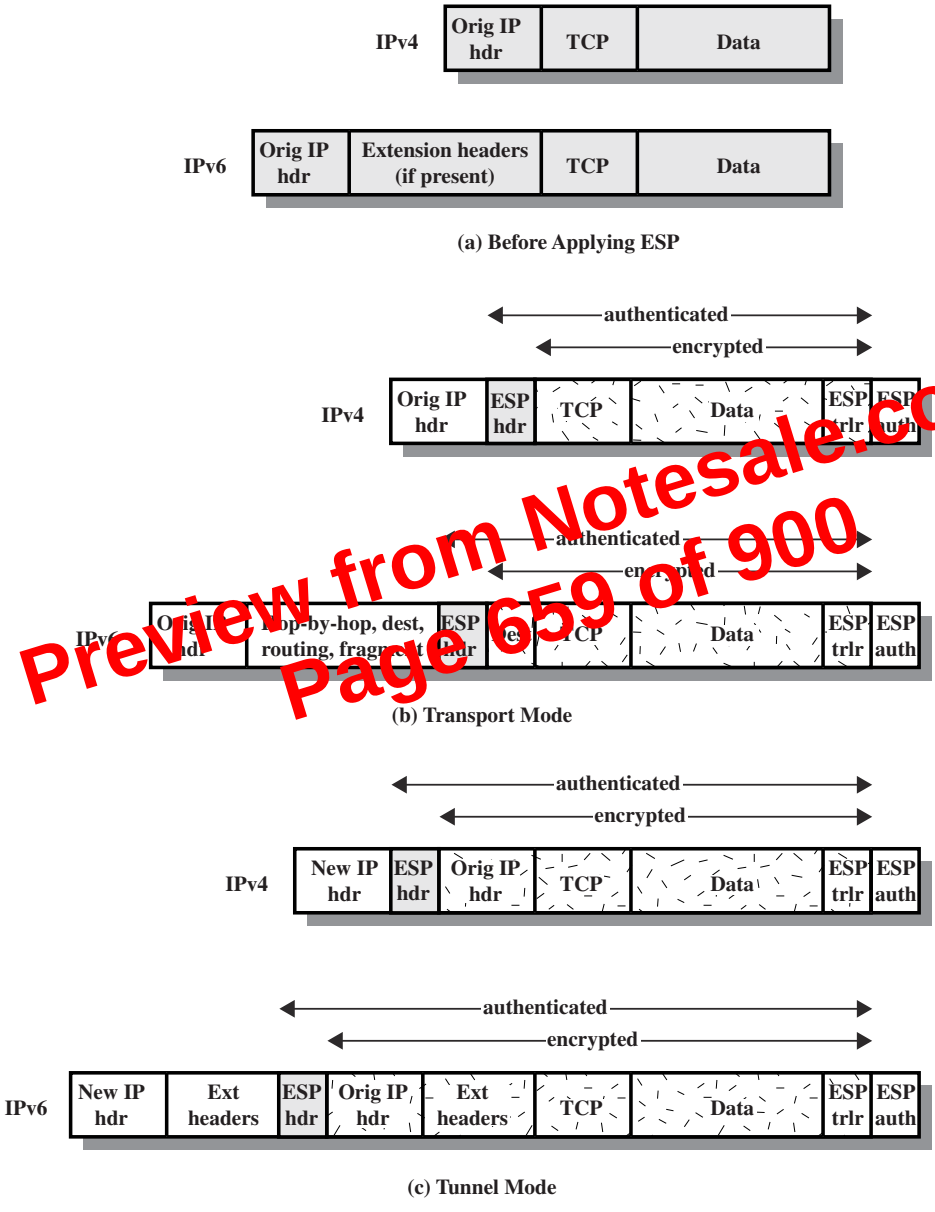


Figure 19.8 Scope of ESP Encryption and Authentication

For this mode using IPv4, the ESP header is inserted into the IP packet immediately prior to the transport-layer header (e.g., TCP, UDP, ICMP), and an ESP trailer (Padding, Pad Length, and Next Header fields) is placed after the IP packet. If authentication is selected, the ESP Authentication Data field is added after the ESP trailer. The entire transport-level segment plus the ESP trailer are encrypted. Authentication covers all of the ciphertext plus the ESP header.

- **Symmetric-key encryption:** A key derived by some out-of-band mechanism can be used to authenticate the exchange by symmetric encryption of exchange parameters.

IKEv2 Exchanges The IKEv2 protocol involves the exchange of messages in pairs. The first two pairs of exchanges are referred to as the **initial exchanges** (Figure 19.11a). In the first exchange, the two peers exchange information concerning cryptographic algorithms and other security parameters they are willing to use along with nonces and Diffie-Hellman (DH) values. The result of this exchange is to set up a special SA called the IKE SA (see Figure 19.2). This SA defines parameters for a secure channel between the peers over which subsequent message exchanges take place. Thus, all subsequent IKE message exchanges are protected by encryption and message authentication. In the second exchange, the two parties authenticate one another and set up a first IPsec SA to be placed in the SADB and used for protecting ordinary (i.e. non-IKE) communications between the peers. Thus, four messages are needed to establish the first SA for general use.

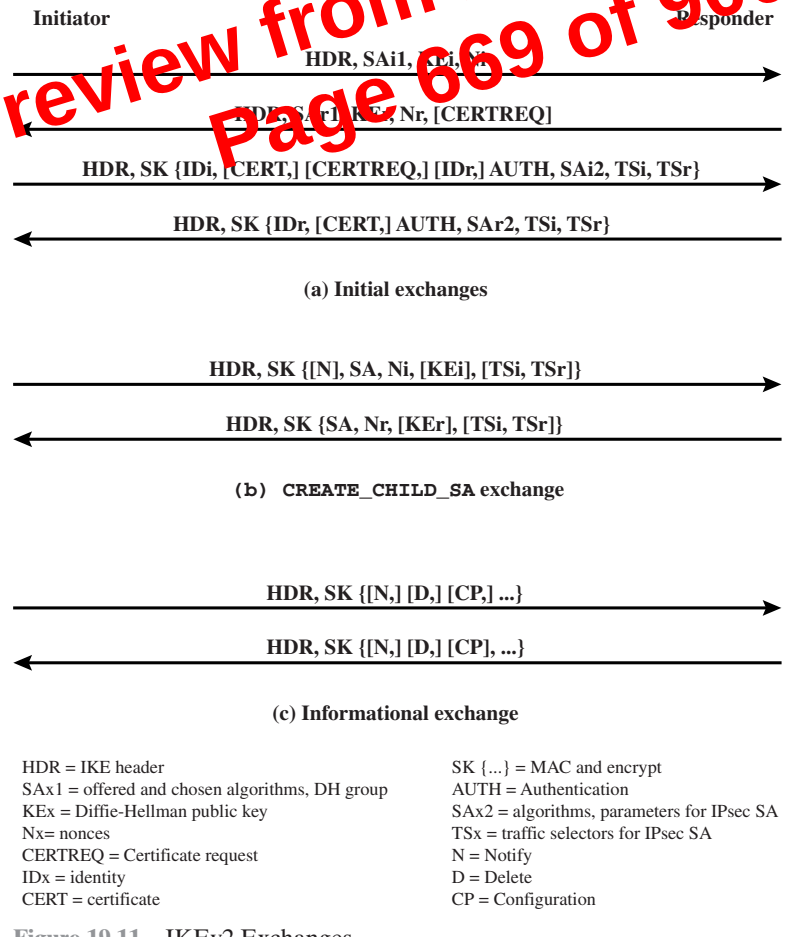


Figure 19.11 IKEv2 Exchanges

The **CREATE_CHILD_SA exchange** can be used to establish further SAs for protecting traffic. The **informational exchange** is used to exchange management information, IKEv2 error messages, and other notifications.

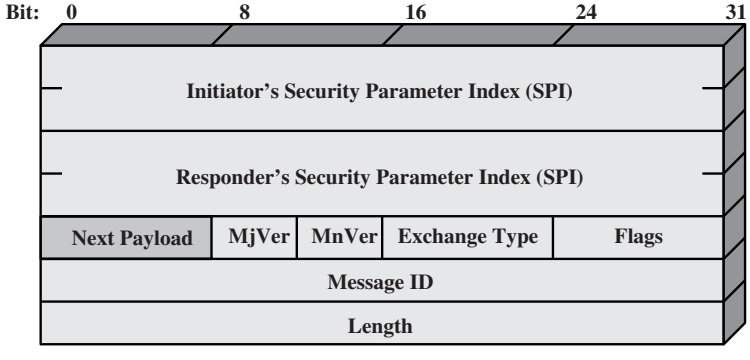
Header and Payload Formats

IKE defines procedures and packet formats to establish, negotiate, modify, and delete security associations. As part of SA establishment, IKE defines payloads for exchanging key generation and authentication data. These payload formats provide a consistent framework independent of the specific key exchange protocol, encryption algorithm, and authentication mechanism.

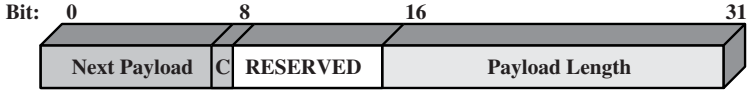
IKE Header Format An IKE message consists of an IKE header followed by one or more payloads. All of this is carried in a transport protocol. The specification dictates that implementations must support the use of UDP for the transport protocol.

Figure 19.12a shows the header format for an IKE message. It consists of the following fields.

- **Initiator SPI (64 bits):** A value chosen by the initiator to identify a unique IKE security association (SA).
- **Responder SPI (64 bits):** A value chosen by the responder to identify a unique IKE SA.
- **Next Payload (8 bits):** Indicates the type of the first payload in the message; payloads are discussed in the next subsection.
- **Major Version (4 bits):** Indicates major version of IKE in use.
- **Minor Version (4 bits):** Indicates minor version in use.



(a) IKE header



(b) Generic Payload header

Figure 19.12 IKE Formats

A.6 PROGRAMMING PROJECTS

The programming project is a useful pedagogical tool. There are several attractive features of stand-alone programming projects that are not part of an existing security facility:

1. The instructor can choose from a wide variety of cryptography and network security concepts to assign projects.
2. The projects can be programmed by the students on any available computer and in any appropriate language; they are platform and language independent.
3. The instructor need not download, install, and configure any particular infrastructure for stand-alone projects.

There is also flexibility in the size of projects. Larger projects give students more a sense of achievement, but students with less ability or fewer organizational skills can be left behind. Larger projects usually elicit more overall effort from the best students. Smaller projects can have a higher concepts-to-code ratio and, because more of them can be assigned, more opportunity exists to address a variety of different areas.

Again, as with research projects, the students should first submit a proposal. The student handout should include the same elements listed in the preceding section. The IRC includes a set of twelve possible programming projects.

The following individuals have supplied the research and programming projects suggested in the IRC: Henning Schulzrinne of Columbia University; Cetin Kaya Koc of Oregon State University; and David M. Balenson of Trusted Information Systems and George Washington University.

A.7 PRACTICAL SECURITY ASSESSMENTS

Examining the current infrastructure and practices of an existing organization is one of the best ways of developing skills in assessing its security posture. The IRC contains a list of such activities. Students, working either individually or in small groups, select a suitable small- to medium-sized organization. They then interview some key personnel in that organization in order to conduct a suitable selection of security risk assessment and review tasks as it relates to the organization's IT infrastructure and practices. As a result, they can then recommend suitable changes, which can improve the organization's IT security. These activities help students develop an appreciation of current security practices and the skills needed to review these and recommend changes.

Lawrie Brown of the Australian Defence Force Academy developed these projects.

A.8 WRITING ASSIGNMENTS

Writing assignments can have a powerful multiplier effect in the learning process in a technical discipline such as cryptography and network security. Adherents of the Writing Across the Curriculum (WAC) movement (http://wac.colostate.edu/) report

```
P10_data = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6];

P8_data = [6, 3, 7, 4, 8, 5, 10, 9];

LS1_data = [2, 3, 4, 5, 1];

LS2_data = [3, 4, 5, 1, 2];

IP_data = [2, 6, 3, 1, 4, 8, 5, 7];

IPinv_data = [4, 1, 3, 5, 7, 2, 8, 6];

EP_data = [4, 1, 2, 3, 2, 3, 4, 1];

P4_data = [2, 4, 3, 1];

SW_data = [5, 6, 7, 8, 1, 2, 3, 4];

#
# SDES lookup tables
#

S0_data = [[1, 0, 3, 2],
           [3, 2, 1, 0],
           [0, 2, 1, 3],
           [3, 1, 3, 2]];

S1_data = [[0, 1, 2, 3],
           [2, 0, 1, 3],
           [3, 0, 1, 0],
           [2, 1, 0, 3]];

def ApplyPermutation(X, permutation):
    r"""
    This function takes a permutation list (list of
    bit positions.)
    And outputs a bit list with the bits taken from X.
    """
    # permute the list X
    l = len(permutation);
    return [X[permutation[j]-1] for j in xrange(l)];

def ApplySBox(X, SBox):
    r"""
    This function Applies the SDES SBox (by table
    look up
    """
    r = 2*X[0] + X[3];
    c = 2*X[1] + X[2];
    o = SBox[r][c];
    return [o & 2, o & 1];
```

```
        temp_block1 = EP(right_block);

        temp_block2 = XorBlock(temp_block1, K);

        left_temp_block2 = LeftHalfBits(temp_block2);
        right_temp_block2  =  RightHalfBits(temp_block2);

        S0_out = S0(left_temp_block2);
        S1_out = S1(right_temp_block2);

        temp_block3 = concatenate(S0_out, S1_out);

        temp_block4 = P4(temp_block3)

        temp_block5 = XorBlock(temp_block4, left_block);

        output_block = concatenate(temp_block5,
        right_block)

        return output_block;

    def SDESEncrypt(plaintext_block, K):
        r"""
        Performs a single SDES plaintext block encryption.
        (Given plaintext and key as bit lists.)
        """
        (K1, K2) = SDESKeySchedule(K);

        temp_block1 = IP(plaintext_block);

        temp_block2 = f_K(temp_block1, K1);

        temp_block3 = SW(temp_block2);

        temp_block4 = f_K(temp_block3, K2);

        output_block = IPinv(temp_block4);

        return output_block;
```


B.4 CHAPTER 4: BASIC CONCEPTS IN NUMBER THEORY AND FINITE FIELDS

Example 1: The Euclidean algorithm for the greatest common divisor

```
def EUCLID(a,b):
    r"""
    The Euclidean algorithm for finding the gcd of a and b.
    This algorithm assumes that a > b => 0

    INPUT:

        a - positive integer

        b - nonnegative integer less than a
```

the second and third elements are coefficients u,v such that gcd(a,b) = u* a + v* b. This can be called as:

```
sage: xgcd(17,31)
(1, 11, -6)
sage: xgcd(10, 115)
(5, -11, 1)
```

This can also be called as a method on Integer objects

```
sage: x = 300
sage: x.xgcd(36)
(12, 1, -8)
```

Example 6: Sage includes robust support for working with finite fields and performing finite field arithmetic. To initialize a finite field with prime order, use the GF command passing the order as the parameter.

```
sage: F = GF(2)
sage: F
Finite Field of size 2
sage: F = GF(37)
sage: F
Finite Field of size 37
```

```
sage: p = 95131
sage: K = GF(p)
sage: K
Finite Field of size 95131
```

To initialize a field with a prime power order use the GF command with the following syntax (to keep track of the primitive element of the extension field.)

```
sage: F.<a> = GF(128)
sage: F
Finite Field in a of size 2^7
```

To do arithmetic in finite fields use the following syntax:

```
sage: K = GF(37)
sage: a = K(3)
sage: b = K(18)
sage: a - b
22
sage: a + b
21
sage: a * b
17
sage: a/b
31
```

```
sage: a^-1
25
sage: 1/a
25
```

To do arithmetic in a finite field with a prime power order, specify elements using the primitive element:

```
sage: F.<a> = GF(128)
sage: b = a^2 + 1
sage: c = a^5 + a^3 + 1
sage: b - c
a^5 + a^3 + a^2
sage: b + c
a^5 + a^3 + a^2
sage: b*c
a^3 + a^2 + a
sage: b/c
a^5 + a^3 + a^2 + a
sage: b^-1
a^5 + a^3 + a
sage: 1/b
a^5 + a^3 + a
```


Example 7: With Sage you can create rings of polynomials over finite fields and do arithmetic with them. To create polynomial rings over finite fields do the following:

```
sage: R.<x> = GF(2)[]
sage: R

Univariate Polynomial Ring in x over Finite Field of
size 2 (using NTL)
sage: R.<x> = GF(101)[]

sage: R
sage: R.<x> = F[]
sage: R
Univariate Polynomial Ring in x over Finite Field in
a of size 2^7
```

After initializing a polynomial ring, you can then just perform arithmetic as you would expect:

```
sage: R.<x> = GF(2)[]
sage: f = x^3 + x + 1
sage: g = x^5 + x
sage: f + g
x^5 + x^3 + 1
sage: f*g
x^8 + x^6 + x^5 + x^4 + x^2 + x
```

```
def SAES_ToStateMatrix(block):
    r"""
    Converts a bit list into an SAES state matrix.
    """
    B = block;

    # form the plaintext block into a matrix of GF(2^n)
    elements
    S00 = L(V([B[0], B[1], B[2], B[3]]));
    S01 = L(V([B[4], B[5], B[6], B[7]]));
    S10 = L(V([B[8], B[9], B[10], B[11]]));
    S11 = L(V([B[12], B[13], B[14], B[15]]));

    state_matrix = Matrix(L, [[S00,S01],[S10,S11]]);

    return state_matrix;

def SAES_FromStateMatrix(State Matrix):
    r"""
    Converts an SAES State Matrix to a bit list.
    """
    output = [];
    # convert State Matrix back into bit list
    for r in xrange(2):
        for c in xrange(2):
            v = V(State Matrix[r,c]);
            for j in xrange(4):
                output.append(Integer(v[j]));

    return output;

def SAES_AddRoundKey(state_matrix, K):
    r"""
    Adds a round key to an SAES state matrix.
    """

    K_matrix = SAES_ToStateMatrix(K);

    next_state_matrix = K_matrix + state_matrix;

    return next_state_matrix;

def SAES_MixColumns(state_matrix):
    r"""
    Performs the Mix Columns operation.
    """
    next_state_matrix = MixColumns_matrix*state_matrix;
    return next_state_matrix;
```

```
    # way to generate primes, because we do not know how the
    # internal sage random_prime function works.

    p = 3;
    while (p < 2^(bitlen-1)) or (3 != (p % 4)):
        p = random_prime(2^bitlen);

    q = 3;
    while (q < 2^(bitlen-1)) or (3 != (q % 4)):
        q = random_prime(2^bitlen);

    N = p*q;

    X = (seed^2 % N)

    state = [N, X]

    return state;

def BlumBlumShub_Generate(num_bits, state):
    r"""
     Blum-Blum-Shub random number generation function.

     INPUT:

        num_bits - the number of bits (iterations) to
        generate with this RNG.

        state - an internal state of the BBS-RNG (a
        list [N, X].)

     OUTPUT:

        random_bits - a num_bits length list of random
        bits.

     """

     random_bits = [];

     N = state[0]
     X = state[1]

     for j in xrange(num_bits):

        X = X^2 % N
        random_bits.append(X % 2)

     # update the internal state
     state[1] = X;

     return random_bits;
```

Example 2: Linear Congruential RNG

```
def LinearCongruential_Initialize(a, c, m, X0):
```

Note that this output is printed (x : y: z). This is a minor technical consideration because Sage stores points in what is known as "projective coordinates." The precise meaning is not important, because for non-infinite points the value z will always be 1 and the first two values in a coordinate will be the x and y coordinates, exactly as you would expect. This representation is useful because it allows the point at infinity to be specified as a point with the z coordinate equal to 0:

```
sage: E(0)
(0 : 1 : 0)
```

This shows how you can recognize a point at infinity as well as specify it. If you want to get the x and y coordinates out of a point on the curve, you can do so as follows:

```
sage: P = E.random_point(); P
(62 : 38 : 1)
sage: (x,y) = P.xy(); (x,y)
(62, 38)
```

You can specify a point on the curve by casting an ordered pair to the curve as:

```
sage: P = E((62,-38)); P
(62 : 65 : 1)
```

Now that you can find the generators on a curve and specify points you can experiment with these points and do arithmetic as well. Continuing to use E as the curve instantiated in the previous example, we can set G1 and G2 to the generators:

```
sage: (G1, G2) = E.gens()
sage: P = E.random_point(); P
(49 : 29 : 1)
```

You can compute the sum of two points as in the following examples:

```
sage: G1 + G2 + P
(69 : 96 : 1)
sage: G1 + P
(40 : 62 : 1)
sage: P + P + G2
(84 : 25 : 1)
```

You can compute the inverse of a point using the unary minus (–) operator:

```
sage: -P
(49 : 72 : 1)
sage: -G1
(7 : 59 : 1)
```

You can also compute repeated point addition (adding a point to itself many times) with the * operator:

```
sage: 13*G1
```

However, in practice most curves that are used have a prime order:

```
sage: # Calculate the domain parameters
sage: F = GF(101)
sage: E = EllipticCurve(F, [0, 0, 0, 25, 7])
sage: G = E((97,34))
sage: q = E.order()
sage: # Alice computes a secret values x in 2...q-1
sage: x = randint(2,q-1)
sage: # Alice computes a public value X = x*G
sage: X = x*G
sage: # Bob computes a secret value y in 2...q-1
sage: y = randint(2,q-1)
sage: # Bob computes a public value Y = y*G
sage: Y = y*G
sage: # Alice computes the shared secret value
sage: x*Y
(23 : 15 : 1)
sage: # Bob computes the shared secret value
sage: y*X
(23 : 15 : 1)
```


B.10 CHAPTER 11: CRYPTOGRAPHIC HASH FUNCTIONS

Example 1: The following is an example of the MASH hash function in Sage. MASH is a function based on the use of modular arithmetic. It involves use of an RSA-like modulus M, whose bit length affects the security. M should be difficult to factor, and for M of unknown factorization, the security is based in part on the difficulty of extracting modular roots. M also determines the block size for processing messages. In essence, MASH is defined as:

$$H_i = ((x_i \oplus H_{i-1})^2 \text{OR}\, H_{i-1}) \,(\text{mod}\, M)$$

where

A = 0xFF00...00

H_{i-1} = the largest prime less than M

x_i = the ith digit of the base M expansion of input n. That is, we express n as a number of base M. Thus:

$$n = x_0 + x_1 M + x_2 M^2 + \ldots$$

The following is an example of the MASH hash function in Sage

```
#
# This function generates a mash modulus
# takes a bit length, and returns a Mash
# modulus l or l-1 bits long (if n is odd)
```

REFERENCES

In matters of this kind everyone feels he is justified in writing and publishing the first thing that comes into his head when he picks up a pen, and thinks his own idea as axiomatic as the fact that two and two make four. If critics would go to the trouble of thinking about the subject for years on end and testing each conclusion against the actual history of war, as I have done, they would undoubtedly be more careful of what they wrote.

—**On War** *Carl von Clausewitz*

ABBREVIATIONS

ACM Association for Computing Machinery
IBM International Business Machines Corporation
IEEE Institute of Electrical and Electronics Engineers

ACM04 The Association for Computing Machinery. *USACM Policy Brief: Digital Millennium Copyright Act (DMCA).* February 6, 2004. acm.org/usacm/Issues/DMCA.htm

ADAM90 Adams, C., and Tavares, S. "Generating and Counting Binary Bent Sequences." *IEEE Transactions on Information Theory*, 1990.

AGRA02 Agrawal, M.; Kayal, N.; and Saxena, N. "PRIMES is in P." *IIT Kanpur, Preprint*, August 2002. http://www.cse.iitk.ac.in/news/primality.pdf

ANDR04 Andrews, M., and Whittaker, J. "Computer Security." *IEEE Security and Privacy*, September/October 2004.

AKL83 Akl, S. "Digital Signatures: A Tutorial Survey." *Computer*, February 1983.

ALVA90 Alvare, A. "How Crackers Crack Passwords or What Passwords to Avoid." *Proceedings, UNIX Security Workshop II*, August 1990.

ANDE80 Anderson, J. *Computer Security Threat Monitoring and Surveillance.* Fort Washington, PA: James P. Anderson Co., April 1980.

ANDE93 Anderson, R., et al. "Using the New ACM Code of Ethics in Decision Making." *Communications of the ACM*, February 1993.

ANTE06 Ante, S., and Grow, B. "Meet the Hackers." *Business Week*, May 29, 2006.

ASHL01 Ashley, P.; Hinton, H.; and Vandenwauver, M. "Wired versus Wireless Security: The Internet, WAP and iMode for E-Commerce." *Proceedings, Annual Computer Security Applications Conference*, 2001.

AUDI04 Audin, G. "Next-Gen Firewalls: What to Expect." *Business Communications Review*, June 2004.

AXEL00 Axelsson, S. "The Base-Rate Fallacy and the Difficulty of Intrusion Detection." *ACM Transactions and Information and System Security*, August 2000.

AYCO06 Aycock, J. *Computer Viruses and Malware.* New York: Springer, 2006.

BACE00 Bace, R. *Intrusion Detection.* Indianapolis, IN: Macmillan Technical Publishing, 2000.

BARK91 Barker, W. *Introduction to the Analysis of the Data Encryption Standard (DES).* Laguna Hills, CA: Aegean Park Press, 1991.

BARK07a Barker, E., and Kelsey, J. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators.* NIST SP 800-90, March 2007.

BARK07b Barker, E., et al. *Recommendation for Key Management — Part 1: General.* NIST SP800-57, March 2007.

BARK07c Barker, E., et al. *Recommendation for Key Management — Part 2: Best Practices for Key Management Organization.* NIST SP800-57, March 2007.

BARK08 Barker, E., et al. *Recommendation for Key Management — Part* 3: *Specific Key Management Guidance.* NIST SP800-57, August 2008.

BARR05 Barrett, D.; Silverman, R.; and Byrnes, R. *SSH The Secure Shell: The Definitive Guide.* Sebastopol, CA; O'Reilly, 2005.

DAWS96 Dawson, E., and Nielsen, L. "Automated Cryptoanalysis of XOR Plaintext Strings." *Cryptologia*, April 1996.

DENN81 Denning, D. "Timestamps in Key Distribution Protocols." *Communications of the ACM*, August 1981.

DENN82 Denning, D. *Cryptography and Data Security.* Reading, MA: Addison-Wesley, 1982.

DENN87 Denning, D. "An Intrusion-Detection Model." *IEEE Transactions on Software Engineering,* February 1987.

DESK92 Deskins, W. *Abstract Algebra.* New York: Dover, 1992.

DIFF76a Diffie, W., and Hellman, M. "New Directions in Cryptography." *Proceedings of the AFIPS National Computer Conference*, June 1976.

DIFF76b Diffie, W., and Hellman, M. "Multiuser Cryptographic Techniques." *IEEE Transactions on Information Theory*, November 1976.

DIFF77 Diffie, W., and Hellman, M. "Exhaustive Cryptanalysis of the NBS Data Encryption Standard." *Computer*, June 1977.

DIFF79 Diffie, W., and Hellman, M. "Privacy and Authentication: An Introduction to Cryptography." *Proceedings of the IEEE*, March 1979.

DIFF88 Diffie, W. "The First Ten Years of Public-Key Cryptography." *Proceedings of the IEEE*, May 1988.

DOBB96 Dobbertin, H. "The Status of MD5 After a Recent Attack." *CryptoBytes*, Summer 1996.

DOJ00 U.S. Department of Justice. *The Electronic Frontier: The Challenge of Unlawful Conduct Involving the Use of the Internet.* March 2000. usdoj.gov/criminal/cybercrime/unlawful.htm

EAST05 Eastlake, D.; Schiller, J.; and Crocker, S. *Randomness Requirements for Security.* RFC 4086, June 2005.

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

ELGA84 Elgamal, T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." Proceedings, Crypto 84, 1984.

ELGA85 Elgamal, T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." *IEEE Transactions on Information Theory*, July 1985.

ELLI70 Ellis, J. *The Possibility of Secure Non-Secret Digital Encryption.* CESG Report, January 1970.

ELLI99 Ellis, J. "The History of Non-Secret Encryption." *Cryptologia*, July 1999.

ENGE80 Enger, N., and Howerton, P. *Computer Security.* New York: Amacom, 1980.

ENGE99 Enge, A. *Elliptic Curves and Their Applications to Cryptography.* Norwell, MA: Kluwer Academic Publishers, 1999.

FEIS73 Feistel, H. "Cryptography and Computer Privacy." *Scientific American*, May 1973.

FEIS75 Feistel, H.; Notz, W.; and Smith, J. "Some Cryptographic Techniques for Machine-to-Machine Data Communications." *Proceedings of the IEEE*, November 1975.

FERN99 Fernandes, A. "Elliptic Curve Cryptography." *Dr. Dobb's Journal*, December 1999.

FLUH00 Fluhrer, S., and McGrew, D. "Statistical Analysis of the Alleged RC4 Key Stream Generator." *Proceedings, Fast Software Encryption 2000*, 2000.

FLUH01 Fluhrer, S.; Mantin, I.; and Shamir, A. "Weakness in the Key Scheduling Algorithm of RC4." *Proceedings, Workshop in Selected Areas of Cryptography,*2001.

FORR97 Forrest, S.; Hofmeyr, S.; and Somayaji, A. "Computer Immunology." *Communications of the ACM*, October 1997.

FRAN05 Frankel, S., et al. *Guide to IPsec VPNs.* NIST SP 800-77, 2005.

FRAN07 Frankel, S.; Eydt, B.; Owens, L.; and Scarfone, K. *Establishing Wireless Robust Security Networks: A Guide to IEEE 802.11i.* NIST Special Publication SP 800-97, February 2007.

FRAN09 Frankel, S., and Krishnan, S. *IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap.* draft-ietf-ipsecme-roadmap-01.txt, March 6, 2009.

FRAS97 Fraser, B. *Site Security Handbook.* RFC 2196, September 1997.

FUMY93 Fumy, S., and Landrock, P. "Principles of Key Management." *IEEE Journal on Selected Areas in Communications*, June 1993.

GARD72 Gardner, M. *Codes, Ciphers, and Secret Writing.* New York: Dover, 1972.

GARD77 Gardner, M. "A New Kind of Cipher That Would Take Millions of Years to Break." *Scientific American*, August 1977.

GARF02 Garfinkel, S., and Spafford, G. *Web Security, Privacy & Commerce.* Sebastapol, CA: O'Reilly, 2002.

GARR01 Garrett, P. *Making, Breaking Codes: An Introduction to Cryptology.* Upper Saddle River, NJ: Prentice Hall, 2001.

GAUD00 Gaudin, S. "The Omega Files." *Network World*, June 26, 2000.

GIBB00 Gibbs, J. "The Digital Millennium Copyright Act." *ACM Ubiquity*, August 2000.

GILB03 Gilbert, H. and Handschuh, H. "Security Analysis of SHA-256 and Sisters." "*Proceedings, CRYPTO '03*, 2003; published by Springer-Verlag.

GOLD88 Goldwasser, S.; Micali, S.; and Rivest, R. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks." *SIAM Journal on Computing*, April 1988.

GONG92 Gong, L. "A Security Risk of Depending on Synchronized Clocks." *Operating Systems Review*, January 1992.

GONG93 Gong, L. "Variations on the Themes of Message Freshness and Replay." *Proceedings, IEEE Computer Security Foundations Workshop*, June 1993.

GOTT99 Gotterbarn, D. " How the New Software Engineering Code of Ethics Affects You." *IEEE Software*, November/ December 1999.

GRAH94 Graham, R.; Knuth, D.; and Patashnik, O. *Concrete Mathematics: A Foundation for Computer Science.* Reading, MA: Addison-Wesley, 1994.

GRAN04 Grance, T.; Kent, K.; and Kim, B. *Computer Security Incident Handling Guide.* NIST Special Publication SP 800-61, January 2004.

GUTM02 Gutmann, P. "PKI: It's Not Dead, Just Resting." *Computer*, August 2002.

GUTT06 Gutterman, Z.; Pinkas, B.; and Reinman, T. "Analysis of the Linux Random Number Generator." *Proceedings, 2006 IEEE Symposium on Security and Privacy*, 2006.

HAMM91 Hamming, R. *The Art of Probability for Scientists and Engineers.* Reading, MA: Addison-Wesley, 1991.

HANK04 Hankerson, D.; Menezes, A.; and Vanstone, S. *Guide to Elliptic Curve Cryptography.* New York: Springer, 2004.

HARR90 Harrington, S., and McCollum, R. "Lessons from Corporate America Applied to Training in Computer Ethics." *Proceedings of the ACM Conference on Computers and the Quality of Life (SIGCAS and SIGCAPH)*, September 1990.

HEBE92 Heberlein, L.; Mukherjee, B.; and Levitt, K. "Internetwork Security Monitor: An Intrusion-Detection System for Large-Scale Networks." *Proceedings, 15th National Computer Security Conference,* October 1992.

HEGL06 Hegland, A., et al. "A Survey of Key Management in Ad Hoc Networks." *IEEE Communications Surveys & Tutorials*. 3rd Quarter 2006.

HELD96 Held, G. *Data and Image Compression: Tools and Techniques.* New York: Wiley, 1996.

HELL79 Hellman, M. "The Mathematics of Public-Key Cryptography." *Scientific American*, August 1970.

HEVI99 Hevia, A., and Kiwi, M. "Strength of Two Data Encryption Standard Implementations Under Timing Attacks." *ACM Transactions on Information and System Security*, November 1999.

HERS75 Herstein, I. *Topics in Algebra.* New York: Wiley, 1975.

HEYS95 Heys, H., and Tavares, S. "Avalanche Characteristics of Substitution-Permutation Encryption Networks." *IEEE Transactions on Computers*, September 1995.

HEYS02 Heys, H. "A Tutorial on Linear and Differential Cryptanalysis." *Cryptologia*, July 2002.

HONE01 The Honeynet Project. *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community.* Reading, MA: Addison-Wesley, 2001.

HORO71 Horowitz, E. "Modular Arithmetic and Finite Field Theory: A Tutorial." *Proceedings of the Second ACM Symposium and Symbolic and Algebraic Manipulation*, March 1971.

HUIT98 Huitema, C. *IPv6: The New Internet Protocol.* Upper Saddle River, NJ: Prentice Hall, 1998.

HYPP06 Hypponen, M. "Malware Goes Mobile." *Scientific American*, November 2006.

IANN06 Iannella, R. "Digital Rights Management." In Bidgoli, H., editor. *Handbook of Information Security.* New York: Wiley, 2006.

B

C

PART 6: SYSTEM SECURITY

CHAPTER 20

INTRUDERS

interviews with a number of password crackers, [ALVA90] reports the following techniques for learning passwords:

1. Try default passwords used with standard accounts that are shipped with the system. Many administrators do not bother to change these defaults.
2. Exhaustively try all short passwords (those of one to three characters).
3. Try words in the system's online dictionary or a list of likely passwords. Examples of the latter are readily available on hacker bulletin boards.
4. Collect information about users, such as their full names, the names of their spouse and children, pictures in their office, and books in their office that are related to hobbies.
5. Try users' phone numbers, Social Security numbers, and room numbers.
6. Try all legitimate license plate numbers for this state.
7. Use a Trojan horse (described in Chapter 21) to bypass restrictions on access.
8. Tap the line between a remote user and the host system.

The first six methods are various ways of guessing a password. If an intruder has to verify the guess by attempting to log in, it is a tedious and easily countered means of attack. For example, a system can simply reject any login after three password attempts, thus requiring the intruder to reconnect to the host to try again. Under these circumstances, it is not practical to try more than a handful of passwords. However, the intruder is unlikely to try such crude methods. For example, if an intruder can gain access with a low level of privileges to an encrypted password file, then the strategy would be to capture that file and then use the encryption mechanism of that particular system at leisure until a valid password that provided greater privileges was discovered.

Guessing attacks are feasible, and indeed highly effective, when a large number of guesses can be attempted automatically and each guess verified, without the guessing process being detectable. Later in this chapter, we have much to say about thwarting guessing attacks.

The seventh method of attack listed earlier, the Trojan horse, can be particularly difficult to counter. An example of a program that bypasses access controls was cited in [ALVA90]. A low-privilege user produced a game program and invited the system operator to use it in his or her spare time. The program did indeed play a game, but in the background it also contained code to copy the password file, which was unencrypted but access protected, into the user's file. Because the game was running under the operator's high-privilege mode, it was able to gain access to the password file.

The eighth attack listed, line tapping, is a matter of physical security.

Other intrusion techniques do not require learning a password. Intruders can get access to a system by exploiting attacks such as buffer overflows on a program that runs with certain privileges. Privilege escalation can be done this way as well.

We turn now to a discussion of the two principal countermeasures: detection and prevention. Detection is concerned with learning of an attack, either before or after its success. Prevention is a challenging security goal and an uphill battle at all times. The difficulty stems from the fact that the defender must attempt to thwart all possible attacks, whereas the attacker is free to try to find the weakest link in the defense chain and attack at that point.

Table 20.2 Measures That May Be Used for Intrusion Detection

Measure	Model	Type of Intrusion Detected
Login and Session Activity		
Login frequency by day and time	Mean and standard deviation	Intruders may be likely to log in during off-hours.
Frequency of login at different locations	Mean and standard deviation	Intruders may log in from a location that a particular user rarely or never uses.
Time since last login	Operational	Break-in on a "dead" account.
Elapsed time per session	Mean and standard deviation	Significant deviations might indicate masquerader.
Quantity of output to location	Mean and standard deviation	Excessive amounts of data transmitted to remote locations could signify leakage of sensitive data.
Session resource utilization	Mean and standard deviation	Unusual processor or I/O levels could signal an intruder.
Password failures at login	Operational	Attempted break-in by password guessing.
Failures to login from specified terminals	Operational	Attempted break-in.
Command or Program Execution Activity		
Execution frequency	Mean and standard deviation	May detect intruders, who are likely to use different commands, or a successful penetration by a legitimate user, who has gained access to privileged commands.
Program resource utilization	Mean and standard deviation	An abnormal value might suggest injection of a virus or Trojan horse, which performs side-effects that increase I/O or processor utilization.
Execution denials	Operational model	May detect penetration attempt by individual user who seeks higher privileges.
File Access Activity		
Read, write, create, delete frequency	Mean and standard deviation	Abnormalities for read and write access for individual users may signify masquerading or browsing.
Records read, written	Mean and standard deviation	Abnormality could signify an attempt to obtain sensitive data by inference and aggregation.
Failure count for read, write, create, delete	Operational	May detect users who persistently attempt to access unauthorized files.

system administrators and security analysts to collect a suite of known penetration scenarios and key events that threaten the security of the target system.

A simple example of the type of rules that can be used is found in NIDX, an early system that used heuristic rules that can be used to assign degrees of suspicion to activities [BAUE88]. Example heuristics are the following:

1. Users should not read files in other users' personal directories.
2. Users must not write other users' files.

The Base-Rate Fallacy

To be of practical use, an intrusion detection system should detect a substantial percentage of intrusions while keeping the false alarm rate at an acceptable level. If only a modest percentage of actual intrusions are detected, the system provides a false sense of security. On the other hand, if the system frequently triggers an alert when there is no intrusion (a false alarm), then either system managers will begin to ignore the alarms, or much time will be wasted analyzing the false alarms.

Unfortunately, because of the nature of the probabilities involved, it is very difficult to meet the standard of high rate of detections with a low rate of false alarms. In general, if the actual numbers of intrusions is low compared to the number of legitimate uses of a system, then the false alarm rate will be high unless the test is extremely discriminating. A study of existing intrusion detection systems, reported in [AXEL00], indicated that current systems have not overcome the problem of the base-rate fallacy. See Appendix 20A for a brief background on the mathematics of this problem.

Distributed Intrusion Detection

Until recently, work on intrusion detection systems focused on single-system stand-alone facilities. The typical organization, however, needs to defend a distributed collection of hosts supported by a LAN or internetwork. Although it is possible to mount a defense by using stand-alone intrusion detection systems on each host, a more effective defense can be achieved by coordination and cooperation among intrusion detection systems across the network.

Porras points out the following major issues in the design of a distributed intrusion detection system [PORR92]:

- A distributed intrusion detection system may need to deal with different audit record formats. In a heterogeneous environment, different systems will employ different native audit collection systems and, if using intrusion detection, may employ different formats for security-related audit records.
- One or more nodes in the network will serve as collection and analysis points for the data from the systems on the network. Thus, either raw audit data or summary data must be transmitted across the network. Therefore, there is a requirement to assure the integrity and confidentiality of these data. Integrity is required to prevent an intruder from masking his or her activities by altering the transmitted audit information. Confidentiality is required because the transmitted audit information could be valuable.
- Either a centralized or decentralized architecture can be used. With a centralized architecture, there is a single central point of collection and analysis of all audit data. This eases the task of correlating incoming reports but creates a potential bottleneck and single point of failure. With a decentralized architecture, there are more than one analysis centers, but these must coordinate their activities and exchange information.

as frightening, is shown in Table 20.5. In all, nearly one-fourth of the passwords were guessed. The following strategy was used:

1. Try the user's name, initials, account name, and other relevant personal information. In all, 130 different permutations for each user were tried.
2. Try words from various dictionaries. The author compiled a dictionary of over 60,000 words, including the online dictionary on the system itself, and various other lists as shown.

Table 20.5 Passwords Cracked from a Sample Set of 13,797 Accounts [KLEI90]

Type of Password	Search Size	Number of Matches	Percentage of Passwords Matched	Cost/Benefit Ratio[a]
User/account name	130	368	2.7%	2.830
Character sequences	866	22	0.2%	0.025
Numbers	427	9	0.1%	0.021
Chinese	392	56	0.4%	0.143
Place names	628	82	0.6%	0.131
Common names	2239	548	4.0%	0.245
Female names	4280	161	1.2%	0.038
Male names	2866	140	1.0%	0.049
Uncommon names	4955	130	0.9%	0.026
Myths & legends	1246	66	0.5%	0.053
Shakespearean	473	11	0.1%	0.023
Sports terms	238	32	0.2%	0.134
Science fiction	691	59	0.4%	0.085
Movies and actors	99	12	0.1%	0.121
Cartoons	92	9	0.1%	0.098
Famous people	290	55	0.4%	0.190
Phrases and patterns	933	253	1.8%	0.271
Surnames	33	9	0.1%	0.273
Biology	58	1	0.0%	0.017
System dictionary	19683	1027	7.4%	0.052
Machine names	9018	132	1.0%	0.015
Mnemonics	14	2	0.0%	0.143
King James bible	7525	83	0.6%	0.011
Miscellaneous words	3212	54	0.4%	0.017
Yiddish words	56	0	0.0%	0.000
Asteroids	2407	19	0.1%	0.007
TOTAL	62727	3340	24.2%	0.053

[a]Computed as the number of matches divided by the search size. The more words that needed to be tested for a match, the lower the cost/benefit ratio.

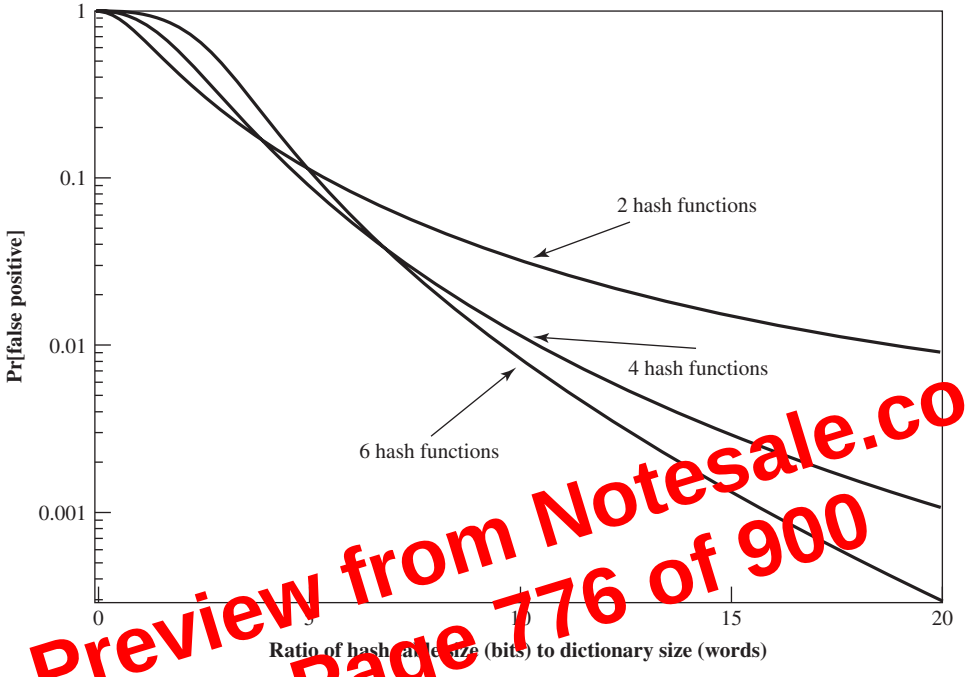


Figure 20.6 Performance of Bloom Filter

20.4 RECOMMENDED READING AND WEB SITES

Two thorough treatments of intrusion detection are [BACE00] and [PROC01]. A more concise but very worthwhile treatment is [SCAR07]. Two short but useful survey articles on the subject are [KENT00] and [MCHU00]. [NING04] surveys recent advances in intrusion detection techniques. [HONE01] is the definitive account on honeypots and provides a detailed analysis of the tools and methods of hackers.

BACE00 Bace, R. *Intrusion Detection.* Indianapolis, IN: Macmillan Technical Publishing, 2000.

HONE01 The Honeynet Project. *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community.* Reading, MA: Addison-Wesley, 2001.

KENT00 Kent, S. "On the Trail of Intrusions into Information Systems." *IEEE Spectrum*, December 2000.

MCHU00 McHugh, J.; Christie, A.; and Allen, J. "The Role of Intrusion Detection Systems." *IEEE Software*, September/October 2000.

NING04 Ning, P., et al. "Techniques and Tools for Analyzing Intrusion Alerts." *ACM Transactions on Information and System Security*, May 2004.

PROC01 Proctor, P., *The Practical Intrusion Detection Handbook.* Upper Saddle River, NJ: Prentice Hall, 2001.

SCAR07 Scarfone, K., and Mell, P. *Guide to Intrusion Detection and Prevention Systems.* NIST Special Publication SP 800-94, February 2007.

What is the concept of defense: The parrying of a blow. What is its characteristic feature: Awaiting the blow.

—*On War,* Carl Von Clausewitz

KEY POINTS

- Malicious software is software that is intentionally included or inserted in a system for a harmful purpose.
- A virus is a piece of software that can "infect" other programs by modifying them; the modification includes a copy of the virus program, which can then go on to infect other programs.
- A worm is a program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. In addition to propagation, the worm usually performs some unwanted function.
- A denial of service (DoS) attack is an attempt to prevent legitimate users of a service from using that service.
- A distributed denial of service attack is launched from multiple coordinated sources.

Perhaps the most sophisticated types of threats to computer systems are presented by programs that exploit vulnerabilities in computing systems. Such threats are referred to as **malicious software**, or **malware**. In this context, we are concerned with threats to application programs as well as utility programs, such as editors and compilers, and kernel-level programs.

This chapter examines malicious software, with a special emphasis on viruses and worms. The chapter begins with a survey of various types of malware, with a more detailed look at the nature of viruses and worms. We then turn to distributed denial-of-service attacks. Throughout, the discussion presents both threats and countermeasures.

21.1 TYPES OF MALICIOUS SOFTWARE

The terminology in this area presents problems because of a lack of universal agreement on all of the terms and because some of the categories overlap. Table 21.1 is a useful guide.

Malicious software can be divided into two categories: those that need a host program, and those that are independent. The former, referred to as **parasitic**, are essentially fragments of programs that cannot exist independently of some actual application program, utility, or system program. Viruses, logic bombs,

workers [GAUD00]. Ultimately, Lloyd was sentenced to 41 months in prison and ordered to pay $2 million in restitution.

Trojan Horses

A Trojan horse[1] is a useful, or apparently useful, program or command procedure containing hidden code that, when invoked, performs some unwanted or harmful function.

Trojan horse programs can be used to accomplish functions indirectly that an unauthorized user could not accomplish directly. For example, to gain access to the files of another user on a shared system, a user could create a Trojan horse program that, when executed, changes the invoking user's file permissions so that the files are readable by any user. The author could then induce users to run the program by placing it in a common directory and naming it such that it appears to be a useful utility program or application. An example is a program that ostensibly produces a listing of the user's files in a desirable format. After another user has run the program, the author of the program can then access the information in the user's files. An example of a Trojan horse program that would be difficult to detect is a compiler that has been modified to insert additional code into certain programs as they are compiled, such as a system login program [THOM84]. The code creates a backdoor in the login program that permits the author to log on to the system using a special password. This Trojan horse can never be discovered by reading the source code of the login program.

Another common motivation for the Trojan horse is data destruction. The program appears to be performing a useful function (e.g., a calculator program), but it may also be quietly deleting the user's files. For example, a CBS executive was victimized by a Trojan horse that destroyed all information contained in his computer's memory [TIME90]. The Trojan horse was implanted in a graphics routine offered on an electronic bulletin board system.

Trojan horses fit into one of three models:

- Continuing to perform the function of the original program and additionally performing a separate malicious activity
- Continuing to perform the function of the original program but modifying the function to perform malicious activity (e.g., a Trojan horse version of a login program that collects passwords) or to disguise other malicious activity (e.g., a Trojan horse version of a process listing program that does not display certain processes that are malicious)
- Performing a malicious function that completely replaces the function of the original program

---

[1]In Greek mythology, the Trojan horse was used by the Greeks during their siege of Troy. Epeios constructed a giant hollow wooden horse in which thirty of the most valiant Greek heroes concealed themselves. The rest of the Greeks burned their encampment and pretended to sail away but actually hid nearby. The Trojans, convinced the horse was a gift and the siege over, dragged the horse into the city. That night, the Greeks emerged from the horse and opened the city gates to the Greek army. A bloodbath ensued, resulting in the destruction of Troy and the death or enslavement of all its citizens.

```
      program CV :=

{goto main;
      01234567;

      subroutine infect-executable :=
            {loop:
                  file := get-random-executable-file;
            if (first-line-of-file = 01234567) then goto loop;
         (1)   compress file;
         (2)   prepend CV to file;
         }

main:   main-program :=
            {if ask-permission then infect-executable;
         (3)   uncompress rest-of-file;
         (4)   run uncompressed file;}
         }
```

Figure 21.2 Logic for a Compression Virus

phase of the program is reasonably rapid, a user is unlikely to notice any difference between the execution of an infected and an uninfected program.

A virus such as this one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file so that both the infected and uninfected versions are of identical length. Figure 21.2 [COHE94] shows in general terms the logic required. The key lines in this virus are numbered, and Figure 21.3 [COHE94] illustrates the operation. We assume that program P1 is infected with the virus CV. When this program is invoked, control passes to its virus, which performs the following steps:

1. For each uninfected file P_2 that is found, the virus first compresses that file to produce P'_2, which is shorter than the original program by the size of the virus.
2. A copy of the virus is prepended to the compressed program.
3. The compressed version of the original infected program, P'_1, is uncompressed.
4. The uncompressed original program is executed.

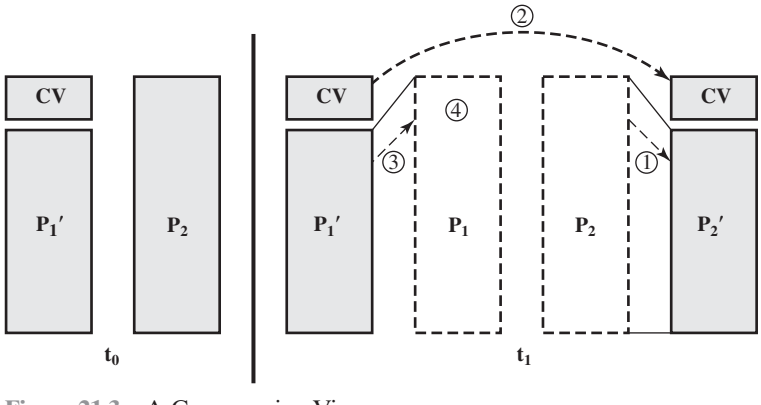


Figure 21.3 A Compression Virus

- **Stealth virus:** A form of virus explicitly designed to hide itself from detection by antivirus software. Thus, the entire virus, not just a payload is hidden.
- **Polymorphic virus:** A virus that mutates with every infection, making detection by the "signature" of the virus impossible.
- **Metamorphic virus:** As with a polymorphic virus, a metamorphic virus mutates with every infection. The difference is that a metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses may change their behavior as well as their appearance.

One example of a **stealth virus** was discussed earlier: a virus that uses compression so that the infected program is exactly the same length as an uninfected version. Far more sophisticated techniques are possible. For example, a virus can place intercept logic in disk I/O routines, so that when there is an attempt to read suspected portions of the disk using these routines, the virus will present back the original, uninfected program. Thus, *stealth* is not a term that applies to a virus as such but, rather, refers to a technique used by a virus to evade detection.

A **polymorphic virus** creates copies during replication that are functionally equivalent but have distinctly different bit patterns. As with a stealth virus, the purpose is to defeat programs that scan for viruses. In this case, the "signature" of the virus will vary with each copy. To achieve this variation, the virus may randomly insert superfluous instructions or interchange the order of independent instructions. A more effective approach is to use encryption. The strategy of the encryption virus is followed. The portion of the virus that is responsible for generating keys and performing encryption/decryption is referred to as the *mutation engine*. The mutation engine itself is altered with each use.

Virus Kits

Another weapon in the virus writers' armory is the virus-creation toolkit. Such a toolkit enables a relative novice to quickly create a number of different viruses. Although viruses created with toolkits tend to be less sophisticated than viruses designed from scratch, the sheer number of new viruses that can be generated using a toolkit creates a problem for antivirus schemes.

Macro Viruses

In the mid-1990s, macro viruses became by far the most prevalent type of virus. Macro viruses are particularly threatening for a number of reasons:

1. A macro virus is platform independent. Many macro viruses infect Microsoft Word documents or other Microsoft Office documents. Any hardware platform and operating system that supports these applications can be infected.
2. Macro viruses infect documents, not executable portions of code. Most of the information introduced onto a computer system is in the form of a document rather than a program.
3. Macro viruses are easily spread. A very common method is by electronic mail.
4. Because macro viruses infect user documents rather than system programs, traditional file system access controls are of limited use in preventing their spread.

Macro viruses take advantage of a feature found in Word and other office applications such as Microsoft Excel, namely the macro. In essence, a macro is an executable program embedded in a word processing document or other type of file. Typically, users employ macros to automate repetitive tasks and thereby save keystrokes. The macro language is usually some form of the Basic programming language. A user might define a sequence of keystrokes in a macro and set it up so that the macro is invoked when a function key or special short combination of keys is input.

Successive releases of MS Office products provide increased protection against macro viruses. For example, Microsoft offers an optional Macro Virus Protection tool that detects suspicious Word files and alerts the customer to the potential risk of opening a file with macros. Various antivirus product vendors have also developed tools to detect and correct macro viruses. As in other types of viruses, the arms race continues in the field of macro viruses, but they no longer are the predominant virus threat.

E-Mail Viruses

A more recent development in malicious software is the e-mail virus. The first rapidly spreading e-mail viruses, such as Melissa, made use of a Microsoft Word macro embedded in an attachment. If the recipient opens the e-mail attachment, the Word macro is activated. Then:

1. The e-mail virus sends itself to everyone on the mailing list in the user's e-mail package.
2. The virus does local damage on the user's system.

In 1999, a more powerful version of the e-mail virus appeared. This newer version can be activated merely by opening an e-mail that contains the virus rather than opening an attachment. The virus uses the Visual Basic scripting language supported by the e-mail package.

Thus we see a new generation of malware that arrives via e-mail and uses e-mail software features to replicate itself across the Internet. The virus propagates itself as soon as it is activated (either by opening an e-mail attachment or by opening the e-mail) to all of the e-mail addresses known to the infected host. As a result, whereas viruses used to take months or years to propagate, they now do so in hours. This makes it very difficult for antivirus software to respond before much damage is done. Ultimately, a greater degree of security must be built into Internet utility and application software on PCs to counter the growing threat.

21.3 VIRUS COUNTERMEASURES

Antivirus Approaches

The ideal solution to the threat of viruses is prevention: Do not allow a virus to get into the system in the first place, or block the ability of a virus to modify any files containing executable code or macros. This goal is, in general, impossible to achieve,

An example of a mobile phone worm is CommWarrior, which was launched in 2005. This worm replicates by means of Bluetooth to other phones in the receiving area. It also sends itself as an MMS file to numbers in the phone's address book and in automatic replies to incoming text messages and MMS messages. In addition, it copies itself to the removable memory card and inserts itself into the program installation files on the phone.

Worm Countermeasures

There is considerable overlap in techniques for dealing with viruses and worms. Once a worm is resident on a machine, antivirus software can be used to detect it. In addition, because worm propagation generates considerable network activity, network activity and usage monitoring can form the basis of a worm defense.

To begin, let us consider the requirements for an effective worm countermeasure scheme:

- **Generality:** The approach taken should be able to handle a wide variety of worm attacks, including polymorphic worms.
- **Timeliness:** The approach should respond quickly so as to limit the number of infected systems and the number of generated transmissions from infected systems.
- **Resiliency:** The approach should be resistant to evasion techniques employed by attackers to evade worm countermeasures.
- **Minimal denial-of-service costs:** The approach should result in minimal reduction in capacity or service due to the actions of the countermeasure software. That is, in an attempt to contain worm propagation, the countermeasure should not significantly disrupt normal operation.
- **Transparency:** The countermeasure software and devices should not require modification to existing (legacy) OSs, application software, and hardware.
- **Global and local coverage:** The approach should be able to deal with attack sources both from outside and inside the enterprise network.

No existing worm countermeasure scheme appears to satisfy all these requirements. Thus, administrators typically need to use multiple approaches in defending against worm attacks.

Countermeasure Approaches Following [JHI07], we list six classes of worm defense:

A. **Signature-based worm scan filtering:** This type of approach generates a worm signature, which is then used to prevent worm scans from entering/leaving a network/host. Typically, this approach involves identifying suspicious flows and generating a worm signature. This approach is vulnerable to the use of polymorphic worms: Either the detection software misses the worm or, if it is sufficiently sophisticated to deal with polymorphic worms, the scheme may take a long time to react. [NEWS05] is an example of this approach.

B. **Filter-based worm containment:** This approach is similar to class A but focuses on worm content rather than a scan signature. The filter checks a message to

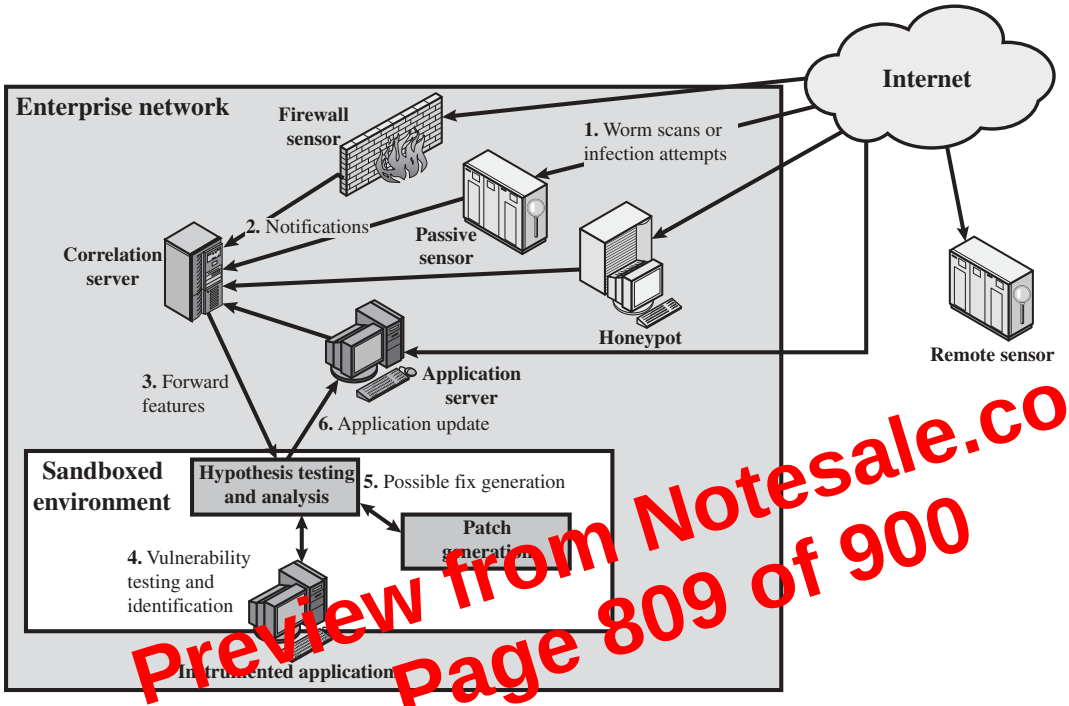


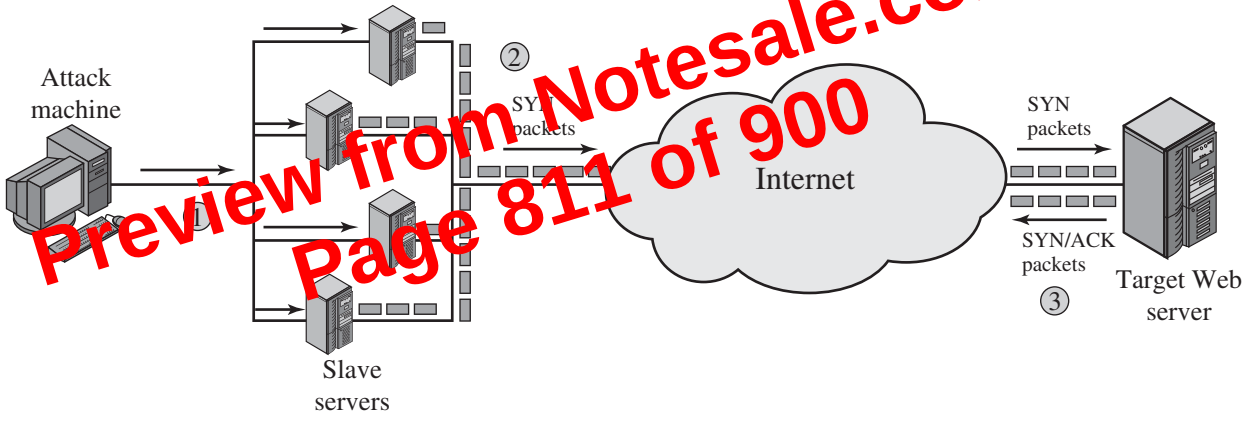
Figure 21.8 Placement of Worm Monitors

The success of such an automated patching system depends on maintaining a current list of potential attacks and developing general tools for patching software to counter such attacks. Examples of approaches are as follows:

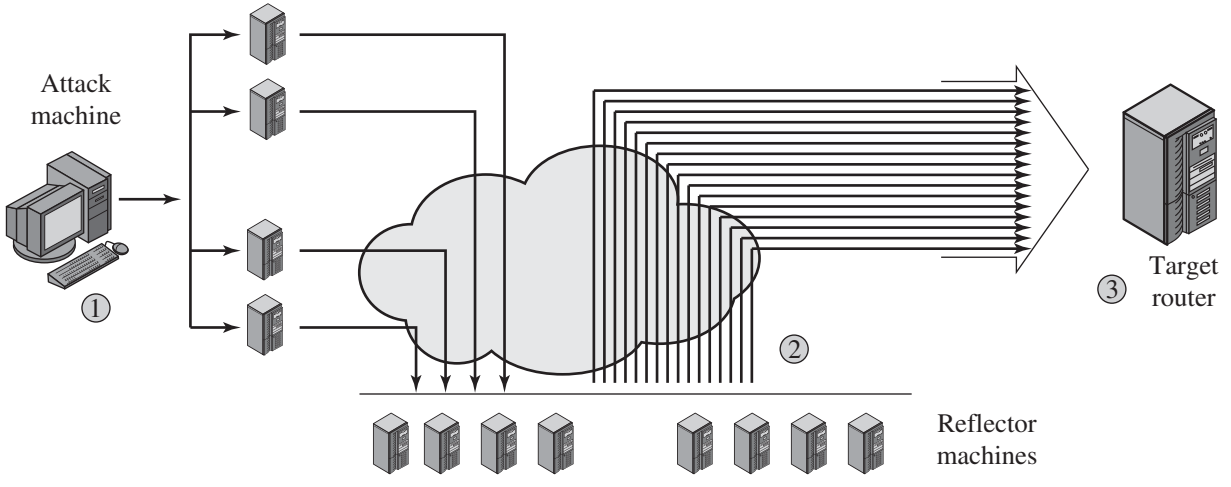
- Increasing the size of buffers
- Using minor code-randomization techniques [BHAT03] so that the infection no longer works because the code to be attacked is no longer in the same form and location
- Adding filters to the application that enable it to recognize and ignore an attack

21.5 DISTRIBUTED DENIAL OF SERVICE ATTACKS

Distributed denial of service (DDoS) attacks present a significant security threat to corporations, and the threat appears to be growing [VIJA02]. In one study, covering a three-week period in 2001, investigators observed more than 12,000 attacks against more than 5000 distinct targets, ranging from well-known ecommerce companies such as Amazon and Hotmail to small foreign ISPs and dial-up connections [MOOR01]. DDoS attacks make computer systems inaccessible by flooding servers, networks, or even end user systems with useless traffic so that legitimate users can no longer gain access to those resources. In a typical DDoS attack, a large number of



(a) Distributed SYN flood attack



(a) Distributed ICMP attack

Figure 21.9 Examples of Simple DDoS Attacks

(client) application is a number between 1024 and 65535. The numbers less than 1024 are the "well-known" port numbers and are assigned permanently to particular applications (e.g., 25 for server SMTP). The numbers between 1024 and 65535 are generated dynamically and have temporary significance only for the lifetime of a TCP connection.

A simple packet filtering firewall must permit inbound network traffic on all these high-numbered ports for TCP-based traffic to occur. This creates a vulnerability that can be exploited by unauthorized users.

A stateful inspection packet firewall tightens up the rules for TCP traffic by creating a directory of outbound TCP connections, as shown in Table 22.2. There is an entry for each currently established connection. The packet filter will now allow incoming traffic to high-numbered ports only for those packets that fit the profile of one of the entries in this directory.

A stateful packet inspection firewall reviews the same packet information as a packet filtering firewall, but also records information about TCP connections (Figure 22.1c). Some stateful firewalls also keep track of TCP sequence numbers to prevent attacks that depend on the sequence number, such as session hijacking. Some even inspect limited amounts of application data for some well-known protocols like FTP, IM and SIPS commands, in order to identify and track related connections.

Application-Level Gateway

An application-level gateway, also called an **application proxy,** acts as a relay of application-level traffic (Figure 22.1d). The user contacts the gateway using a TCP/IP application, such as Telnet or FTP, and the gateway asks the user for the name of the remote host to be accessed. When the user responds and provides a valid user ID and authentication information, the gateway contacts the application on the remote host and relays TCP segments containing the application data between the two endpoints. If the gateway does not implement the proxy code for a specific application, the service is not supported and cannot be forwarded across the firewall. Further, the gateway can be configured to support only specific features of

Table 22.2 Example Stateful Firewall Connection State Table [WACK02]

Source Address	Source Port	Destination Address	Destination Port	Connection State
192.168.1.100	1030	210.22.88.29	80	Established
192.168.1.102	1031	216.32.42.123	80	Established
192.168.1.101	1033	173.66.32.122	25	Established
192.168.1.106	1035	177.231.32.12	79	Established
223.43.21.231	1990	192.168.1.6	80	Established
2122.22.123.32	2112	192.168.1.6	80	Established
210.922.212.18	3321	192.168.1.6	80	Established
24.102.32.23	1025	192.168.1.6	80	Established
223.21.22.12	1046	192.168.1.6	80	Established

- Each proxy maintains detailed audit information by logging all traffic, each connection, and the duration of each connection. The audit log is an essential tool for discovering and terminating intruder attacks.
- Each proxy module is a very small software package specifically designed for network security. Because of its relative simplicity, it is easier to check such modules for security flaws. For example, a typical UNIX mail application may contain over 20,000 lines of code, while a mail proxy may contain fewer than 1000.
- Each proxy is independent of other proxies on the bastion host. If there is a problem with the operation of any proxy, or if a future vulnerability is discovered, it can be uninstalled without affecting the operation of the other proxy applications. Also, if the user population requires support for a new service, the network administrator can easily install the required proxy on the bastion host.
- A proxy generally performs no disk access other than to read its initial configuration file. Hence, the portions of the file system containing executable code can be made read only. This makes it difficult for an intruder to install Trojan horse sniffers or other dangerous files on the bastion host.
- Each proxy runs as a nonprivileged user in a private and secured directory on the bastion host.

Host-Based Firewalls

A host-based firewall is a software module used to secure an individual host. Such modules are available in many operating systems or can be provided as an add-on package. Like conventional stand-alone firewalls, host-resident firewalls filter and restrict the flow of packets. A common location for such firewalls is a server. There are several advantages to the use of a server-based or workstation-based firewall:

- Filtering rules can be tailored to the host environment. Specific corporate security policies for servers can be implemented, with different filters for servers used for different application.
- Protection is provided independent of topology. Thus both internal and external attacks must pass through the firewall.
- Used in conjunction with stand-alone firewalls, the host-based firewall provides an additional layer of protection. A new type of server can be added to the network, with its own firewall, without the necessity of altering the network firewall configuration.

Personal Firewall

A personal firewall controls the traffic between a personal computer or workstation on one side and the Internet or enterprise network on the other side. Personal firewall functionality can be used in the home environment and on corporate intranets. Typically, the personal firewall is a software module on the personal computer. In a

3. Multiple internal firewalls can be used to protect portions of the internal network from each other. For example, firewalls can be configured so that internal servers are protected from internal workstations and vice versa. A common practice is to place the DMZ on a different network interface on the external firewall from that used to access the internal networks.

Virtual Private Networks

In today's distributed computing environment, the **virtual private network (VPN)** offers an attractive solution to network managers. In essence, a VPN consists of a set of computers that interconnect by means of a relatively unsecure network and that make use of encryption and special protocols to provide security. At each corporate site, workstations, servers, and databases are linked by one or more local area networks (LANs). The Internet or some other public network can be used to interconnect sites, providing a cost savings over the use of a private network and offloading the wide area network management task to the public network provider. That same public network provides an access path for telecommuters and other mobile employees to log on to corporate systems from remote sites.

But the manager faces a fundamental requirement: security. Use of a public network exposes corporate traffic to eavesdropping and provides an entry point for unauthorized users. To counter this problem, a VPN is needed. In essence, a VPN uses encryption and authentication in the lower protocol layers to provide a secure connection through an otherwise insecure network, typically the Internet. VPNs are generally cheaper than real private networks using private lines but rely on having the same encryption and authentication system at both ends. The encryption may be performed by firewall software or possibly by routers. The most common protocol mechanism used for this purpose is at the IP level and is known as IPsec.

An organization maintains LANs at dispersed locations. A logical means of implementing an IPsec is in a firewall, as shown in Figure 22.4, which essentially repeats Figure 19.1. If IPsec is implemented in a separate box behind (internal to) the firewall, then VPN traffic passing through the firewall in both directions is encrypted. In this case, the firewall is unable to perform its filtering function or other security functions, such as access control, logging, or scanning for viruses. IPsec could be implemented in the boundary router, outside the firewall. However, this device is likely to be less secure than the firewall and thus less desirable as an IPsec platform.

Distributed Firewalls

A distributed firewall configuration involves stand-alone firewall devices plus host-based firewalls working together under a central administrative control. Figure 22.5 suggests a distributed firewall configuration. Administrators can configure host-resident firewalls on hundreds of servers and workstations as well as configure personal firewalls on local and remote user systems. Tools let the network administrator set policies and monitor security across the entire network. These firewalls protect against internal attacks and provide protection tailored to specific machines and applications. Stand-alone firewalls provide global protection, including internal firewalls and an external firewall, as discussed previously.

22.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

application-level gateway	firewall	personal firewall
bastion host	host-based firewall	proxy
circuit-level gateway	IP address spoofing	stateful inspection firewall
distributed firewalls	IP security (IPsec)	tiny fragment attack
DMZ	packet filtering firewall	virtual private network (VPN)

Review Questions

22.1 List three design goals for a firewall.

22.2 List four techniques used by firewalls to control access and enforce a security policy.

22.3 What information is used by a typical packet filtering firewall?

22.4 What are some weaknesses of a packet filtering firewall?

22.5 What is the difference between a packet filtering firewall and a stateful inspection firewall?

22.6 What is an application-level gateway?

22.7 What is a circuit-level gateway?

22.8 What are the differences among the firewalls of Figure 22.1?

22.9 What are the common characteristics of a bastion host?

22.10 Why is it useful to have host-based firewalls?

22.11 What is a DMZ network and what types of systems would you expect to find on such networks?

22.12 What is the difference between an internal and an external firewall?

Problems

22.1 As was mentioned in Section 22.3, one approach to defeating the tiny fragment attack is to enforce a minimum length of the transport header that must be contained in the first fragment of an IP packet. If the first fragment is rejected, all subsequent fragments can be rejected. However, the nature of IP is such that fragments may arrive out of order. Thus, an intermediate fragment may pass through the filter before the initial fragment is rejected. How can this situation be handled?

22.2 In an IPv4 packet, the size of the payload in the first fragment, in octets, is equal to Total Length – (4 × IHL). If this value is less than the required minimum (8 octets for TCP), then this fragment and the entire packet are rejected. Suggest an alternative method of achieving the same result using only the Fragment Offset field.

22.3 RFC 791, the IPv4 protocol specification, describes a reassembly algorithm that results in new fragments overwriting any overlapped portions of previously received fragments. Given such a reassembly implementation, an attacker could construct a series of packets in which the lowest (zero-offset) fragment would contain innocuous data (and thereby be passed by administrative packet filters), and in which some subsequent packet having a non-zero offset would overlap TCP header information (destination port, for instance) and cause it to be modified. The second packet would be passed through most filter implementations because it does not have a zero fragment offset. Suggest a method that could be used by a packet filter to counter this attack.

Table 22.3 Sample Packet Filter Firewall Ruleset

	Source Address	Source Port	Dest Address	Dest Port	Action
1	Any	Any	192.168.1.0	> 1023	Allow
2	192.168.1.1	Any	Any	Any	Deny
3	Any	Any	192.168.1.1	Any	Deny
4	192.168.1.0	Any	Any	Any	Allow
5	Any	Any	192.168.1.2	SMTP	Allow
6	Any	Any	192.168.1.3	HTTP	Allow
7	Any	Any	Any	Any	Deny

22.4 Table 22.3 shows a sample of a packet filter firewall ruleset for an imaginary network of IP address that range from 192.168.1.0 to 192.168.1.254. Describe the effect of each rule.

22.5 SMTP (Simple Mail Transfer Protocol) is the standard protocol for transferring mail between hosts over TCP. A TCP connection is set up between a user client and a server program. The server listens on TCP port 25 for incoming connection requests. The user end of the connection is on a TCP port number above 1023. Suppose you wish to build a packet filter rule set allowing inbound and outbound SMTP traffic. You generate the following ruleset:

Rule	Direction	Src Addr	Dest Addr	Protocol	Dest Port	Action
A	In	External	Internal	TCP	25	Permit
B	Out	Internal	External	TCP	>1023	Permit
C	Out	Internal	External	TCP	25	Permit
D	In	External	Internal	TCP	>1023	Permit
E	Either	Any	Any	Any	Any	Deny

a. Describe the effect of each rule.

b. Your host in this example has IP address 172.16.1.1. Someone tries to send e-mail from a remote host with IP address 192.168.3.4. If successful, this generates an SMTP dialogue between the remote user and the SMTP server on your host consisting of SMTP commands and mail. Additionally, assume that a user on your host tries to send e-mail to the SMTP server on the remote system. Four typical packets for this scenario are as shown:

Packet	Direction	Src Addr	Dest Addr	Protocol	Dest Port	Action
1	In	192.168.3.4	172.16.1.1	TCP	25	?
2	Out	172.16.1.1	192.168.3.4	TCP	1234	?
3	Out	172.16.1.1	192.168.3.4	TCP	25	?
4	In	192.168.3.4	172.16.1.1	TCP	1357	?

Indicate which packets are permitted or denied and which rule is used in each case.

sensitivity tag (classification if you like) in its subject and for external e-mail to have the lowest sensitivity tag. Discuss how this measure could be implemented in a firewall and what components and architecture would be needed to do this.

22.10 You are given the following "informal firewall policy" details to be implemented using a firewall like that in Figure 22.3:

1. E-mail may be sent using SMTP in both directions through the firewall, but it must be relayed via the DMZ mail gateway that provides header sanitization and content filtering. External e-mail must be destined for the DMZ mail server.
2. Users inside may retrieve their e-mail from the DMZ mail gateway, using either POP3 or POP3S, and authenticate themselves.
3. Users outside may retrieve their e-mail from the DMZ mail gateway, but only if they use the secure POP3 protocol, and authenticate themselves
4. Web requests (both insecure and secure) are allowed from any internal user out through the firewall but must be relayed via the DMZ Web proxy, which provides content filtering (noting this is not possible for secure requests), and users must authenticate with the proxy for logging.
5. Web requests (both insecure and secure) are allowed from anywhere on the Internet to the DMZ Web server
6. DNS lookup requests by internal users are relayed via the DMZ DNS server, which queries to the Internet.
7. External DNS requests are provided by the DMZ DNS server.
8. Management and update of information on the DMZ servers is allowed using secure shell connections from relevant authorized internal users (may have different sets of users on each system as appropriate).
9. SNMP management requests are permitted from the internal management hosts to the firewalls, with the firewalls also allowed to send management traps (i.e., notification of some event occurring) to the management hosts

Design suitable packet filter rulesets (similar to those shown in Table 22.1) to be implemented on the "External Firewall" and the "Internal Firewall" to satisfy the aforementioned policy requirements.

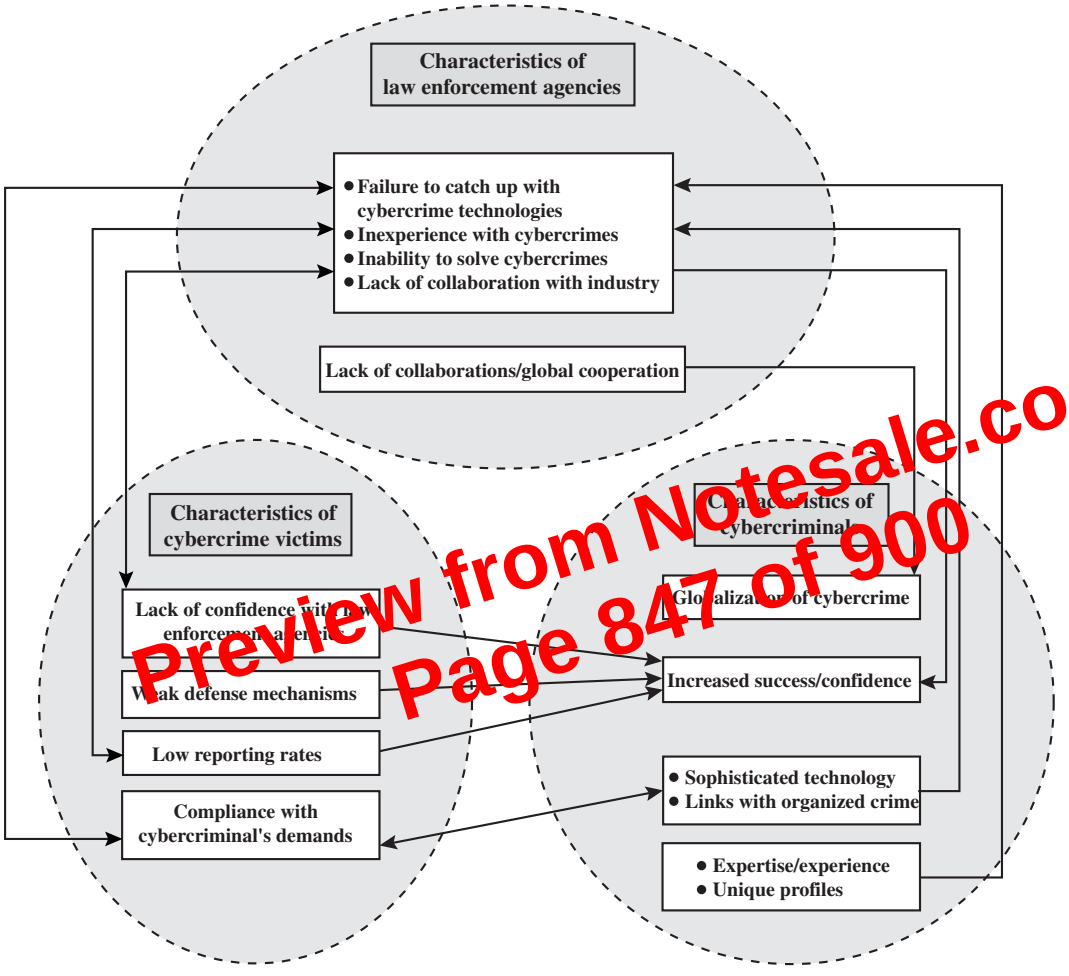


Figure 23.1 The Vicious Cycle of Cybercrime

rates and the reluctance to work with law enforcement on the part of victims feeds into the handicaps under which law enforcement works, completing the vicious cycle.

Working With Law Enforcement

Executive management and security administrators need to look upon law enforcement as another resource and tool, alongside technical, physical, and human-factor resources. The successful use of law enforcement depends much more on people skills than technical skills. Management needs to understand the criminal investigation process, the inputs that investigators need, and the ways in which the victim can contribute positively to the investigation.

DMCA, signed into law in 1998, is designed to implement World Intellectual Property Organization (WIPO) treaties, signed in 1996. In essence, DMCA strengthens the protection of copyrighted materials in digital format.

The DMCA encourages copyright owners to use technological measures to protect copyrighted works. These measures fall into two categories: measures that prevent access to the work and measures that prevent copying of the work. Further, the law prohibits attempts to bypass such measures. Specifically, the law states that "no person shall circumvent a technological measure that effectively controls access to a work protected under this title." Among other effects of this clause, it prohibits almost all unauthorized decryption of content. The law further prohibits the manufacture, release, or sale of products, services, and devices that can crack encryption designed to thwart either access to or copying of material unauthorized by the copyright holder. Both criminal and civil penalties apply to attempts to circumvent technological measures and to assist in such circumvention.

Certain actions are exempted from the provisions of the DMCA and other copyright laws, including the following:

- **Fair use:** This concept is not tightly defined. It is intended to permit others to perform, show, quote, copy, and otherwise distribute portions of the work for certain purposes. These purposes include review, comment, and discussion of copyrighted works.
- **Reverse engineering:** Reverse engineering of a software product is allowed if the user has the right to use a copy of the program and if the purpose of the reverse engineering is not to duplicate the functionality of the program but rather to achieve interoperability.
- **Encryption research:** "Good faith" encryption research is allowed. In essence, this exemption allows decryption attempts to advance the development of encryption technology.
- **Security testing:** This is the access of a computer or network for the good faith testing, investigating, or correcting a security flaw or vulnerability, with the authorization of the owner or operator.
- **Personal privacy:** It is generally permitted to bypass technological measures if that is the only reasonable way to prevent the access to result in the revealing or recording of personally identifying information.

Despite the exemptions built into the Act, there is considerable concern, especially in the research and academic communities, that the act inhibits legitimate security and encryption research. These parties feel that DMCA stifles innovation and academic freedom and is a threat to open source software development [ACM04].

Digital Rights Management

Digital Rights Management (DRM) refers to systems and procedures that ensure that holders of digital rights are clearly identified and receive the stipulated payment for their works. The systems and procedures may also impose further restrictions on the use of digital objects, such as inhibiting printing or prohibiting further distribution.

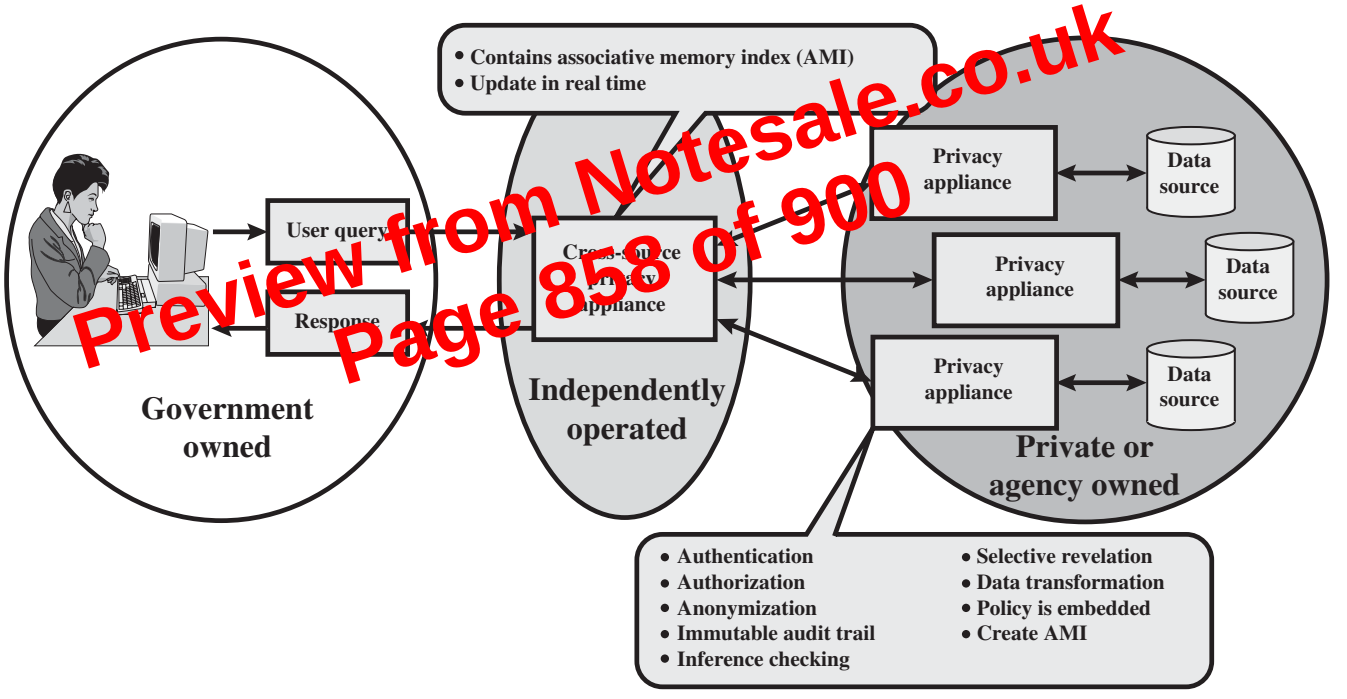


Figure 23.5 Privacy Appliance Concept

related to social power. Construct a table that shows for each theme and for each code, the relevant clause or clauses in the code that address the theme.

23.10 This book's Web site includes a copy of the ACM Code of Professional Conduct from 1982. Compare this Code with the 1997 ACM Code of Ethics and Professional Conduct (Figure 23.7).

a. Are there any elements in the 1982 Code not found in the 1997 Code? Propose a rationale for excluding these.

b. Are there any elements in the 1997 Code not found in the 1982 Code? Propose a rationale for adding these.

23.11 This book's Web site includes a copy of the IEEE Code of Ethics from 1979. Compare this Code with the 2006 IEEE Code of Ethics (Figure 23.8).

a. Are there any elements in the 1979 Code not found in the 2006 Code? Propose a rationale for excluding these.

b. Are there any elements in the 2006 Code not found in the 1979 Code? Propose a rationale for adding these.

23.12 This book's Web site includes a copy of the 1999 Software Engineering Code of Ethics and Professional Practice (Version 5.2) as recommended by an ACM/IEEE-CS Joint Task Force. Compare this Code with each of the three codes reproduced in this chapter (Figure 23.7 through 23.9). Comment on each case on the differences.

APPENDIX C

SAGE EXERCISES

By Dan Shumow
University of Washington

```
for A in foo:
  print A
```

prints the list of the elements in the list **foo**. Or

```
for j in range(10):
  print j
```

prints the integers from 1 to 10.

The most common exception to using a list or a tuple as an iterable object is iterating through a list of integers using the **xrange** function. For example:

```
for j in xrange(len(foo)):
  print 'j=', j, foo[j]
```

prints a list of the indices and the elements at that index in the array.

The **xrange** function allows loops to iterate through the integers **[0, 1, ..., (len(foo)-1)]** without instantiating the list as the function range would. The function xrange takes the parameters as the same as the range function does, the only difference is the output.

While loops have the syntax:

```
while <boolean expression> :
<tab> <block of code>
```

For example:

```
while ( x < 1):
  y = y + x
  x = x/2
```

With both while and for loops, the **break** keyword cause execution of the loop to stop, and **continue** causes control to begin executing at the next iteration of the loop.

Functions

Creating functions is very easy:

```
def <function-name>(< comma separated list of parame-
ters>):
<tab> <block of code>
```

Just like control statements, the body of the function must be delimited by indentation. The **return** statement specifies the value of the function to **return**. If

the function does not have a **return** statement, or the end of the body of the function is reached without hitting a **return** statement, then the function returns the value **None**. For example, the following function:

```
def f1(x, y):
  if (0 == x % 2):
    z = x^2 + x + 1
else:
    z = x-y
return y
```

For more information on Python programming, see http://www.Python.org/. At this time Sage uses Python 2.x, and not Python 3.0 or higher. This is not likely to change, but as the differences in the language are significant, if the examples here are not working, it may be worth checking out if the underlying version of Python that Sage uses has changed.

C.3 CHAPTER 2: CLASSICAL CRYPTOGRAPHY

2.1 Implement Sage functions that perform affine cipher encryption/decryption, given a key that consists of a pair of integers a, b, both in $\{1, 2, \ldots, 25\}$ with a not divisible by 2 or 13. The functions should work on strings, and leave any non-alphabetic characters unchanged. Show the operation of your functions on an example. See problem 2.1 in Chapter 2 for a definition of an affine cipher.

2.2 This question is to implement some functions useful to performing classical cipher attacks.

- **a.** Implement a Sage function that performs frequency attacks on a mono-alphabetic substitution ciphers. This function should take a ciphertext string, compute a histogram of the incidence of each letter (ignoring all non alphabet characters), and return a list of pairs (letter, incidence percentage) sorted by incidence percentage.
- **b.** Implement a Sage function that takes a partial mono-alphabetic substitution and a ciphertext and returns a potential plaintext. The partial mono-alphabetic substitution should be specified as follows: As a 26 character string where the character at position i is the substitution of ith character of the alphabet, OR an underscore '_' if the corresponding substitution is unknown. The potential plaintext should be the ciphertext with values specified by the mono-alphabetic substitution replaced by the lower-case plaintext. If the corresponding character is unknown (i.e. '_' in the monoalphabetic substitution cipher) print the cipher text as an uppercase character.)
- **c.** Use your functions from (a) and (b) to decrypt the following ciphertext: "ztmn pxtne cfa peqef kecnp cjt tmn zcwsenp ontmjsw ztnws tf wsvp xtfwvfefw, c feb fcwvtf, xtfxevqea vf gvoenwk, cfa aeavxcwea wt wse rntrtpvwvtf wscw cgg lef cne xnecwea eymcg."

2.3 Implement Sage functions to perform encryption/decryption with 2×2 Hill Cipher. The key should be an invertible Sage matrix over the integers mod 26.

d. Suppose you know that o[i] = 58246156843038996, and o[i + 1] = 64511473570997445, use the fact that you have two subsequent outputs to determine the possible internal states that could have generated these two outputs.

10.5 For all of the following questions show your Sage input/output.

a. Compute the order of the curve defined by $y^2 = x^3 + 7*x + 25$ over the finite field with 47 elements.

b. On the curve defined by $y^2 + x*y = x^3 + x$ over $GF(2^8)$ compute the inverse of the point (1,1).

c. On the curve defined by $y^2 + y = x^3 + x^2 + x + 1$ over the finite field with 701 elements, find a generator and show its order.

d. On the curve defined by $y^2 = x^3 + 4187*x + 3814$ over finite field of size 6421 compute the sum of the points (3711,373) and (4376,2463).

e. On the elliptic curve defined by $y^2 = x^3 + 3361*x + 6370$ over finite field of size 8461 compute 1001 times the point (1735,3464).

f. On the elliptic curve defined by $y^2 = x^3 + 8000*x + 1357$ over finite field of size 8191, let P1 = (1794, 1319) and P2 = (3514, 409), compute the sum of 13 times P1 plus 28 times P2.

10.6 In this problem use the domain parameters that is the elliptic curve defined by $y^2 = x^3 + 38871*x + 7063$ over the finite field with order 70177. The generator point G = (49359,30140) has order 70393. Show your Sage input/output.

a. Suppose you are Bob and Alice has sent the point (10117, 64081) compute an integer y the point Y and the shared secret.

b. Suppose that Alice chooses the secret value x = 2532 and Bob chooses the secret value y = 15276.

c. Perform a full simulated secret agreement between Alice and Bob.

10.7 The purpose of this question is to implement Sage functions to perform ECDH.

a. Write a function that takes a curve, and a base point on the curve and generates the secret value x and the public value X as per ECDH.

b. Write a function that takes a public value and a secret value and computes the shared secret.

c. Assume that your domain parameters are:
Elliptic Curve defined by $y^2 = x^3 + 26484*x + 15456$ over Finite Field of size 63709
q = 63839
G = (53819,6786)
Show your functions work by simulating an ECDH key exchange.

10.8 Recall that for cryptographic purposes, we use curves with prime order. The purpose of this question is to show why. Let E be the elliptic curve defined by $y^2 = x^3 + 7489*x + 12591$ over Finite Field of size 23431. This curve has order 23304. Let the base point be (20699, 19493).

a. Compute 10 random multiples of this base point. What do you notice?

b. Why is this bad? (*Hint:* What would happen if this was Alice or Bob's public point?)

This page intentionally left blank

trapdoor one-way function A function that is easily computed, and the calculation of its inverse is infeasible unless certain privileged information is known.

Trojan horse* A computer program that appears to have a useful function, but also has a hidden and potentially malicious function that evades security mechanisms, sometimes by exploiting legitimate authorizations of a system entity that invokes the program.

trusted system A computer and operating system that can be verified to implement a given security policy.

unconditionally secure Secure even against an opponent with unlimited time and unlimited computing resources.

virtual private network Consists of a set of computers that interconnect by means of a relatively unsecure network and that make use of encryption and special protocols to provide security.

virus Code embedded within a program that causes a copy of itself to be inserted in one or more other programs. In addition to propagation, the virus usually performs some unwanted function.

worm Program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. In addition to propagation, the worm usually performs some unwanted function.

zombie A program that secretly takes over another Internet-attached computer and then uses that computer to launch attacks that are difficult to trace to the zombie's creator.