

Cryptography and Network Security

Third Edition

About the Author



Atul Kahate has over 17 years of experience in Information Technology in India and abroad in various capacities. He currently works as Adjunct Professor in Computer Science in Pune University and Symbiosis International University. His last IT employment was as Consulting Practice Director at Oracle Financial Services Software Limited (earlier known as i-flex solutions limited). He has conducted several training programs/seminars in institutions such as IIT, Symbiosis, Pune University, and many other colleges.

A prolific writer, Kahate is also the author of 38 books on Computer Science, Science, Technology, Medicine, Economics, Cricket, Management, and History.

Books such as *Web Technologies*, *Cryptography and Network Security*, *Operating Systems*, *Data Communications and Networks*, *An Introduction to Database Management Systems* are used as texts in several universities in India and many other countries. Some of these have been translated into Chinese.

Atul Kahate has won prestigious awards such as Computer Society of India's award for contribution to IT literacy, Indradhanu's Yuvonmesh Puraskar, Indira Group's Excellence Award, Maharashtra Sahitya Parishad's "Granthakar Puraskar", and several others.

He has appeared on quite a few programmes on TV channels such as Doordarshan's Sahyadri channel, IBN Lokmat, Star Maaza, and Saam TV related to IT, education, and careers. He has also worked as official cricket scorer and statistician in several international cricket matches.

Besides these achievements, he has written over 4000 articles and various columns on IT, cricket, science, technology, history, medicine, economics, management, careers in popular newspapers/magazines such as *Loksatta*, *Sakal*, *Maharashtra Times*, *Lokmat*, *Lokprabha*, *Saptahik Sakal*, *Divya Marathi*, and others.

Cryptography and Network Security

Third Edition

Atul Kahate

Adjunct Professor

Pune University and Symbiosis International University

Author in Computer Science



McGraw Hill Education (India) Private Limited
NEW DELHI

McGraw Hill Education Offices

New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto



McGraw Hill Education (India) Private Limited

Published by McGraw Hill Education (India) Private Limited
P-24, Green Park Extension, New Delhi 110 016

Cryptography and Network Security, 3/e

Copyright © 2013, 2008, 2003, by McGraw Hill Education (India) Private Limited

No part of this publication may be reproduced or distributed in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise or stored in a database or retrieval system without the prior written permission of the publishers. The program listings (if any) may be entered, stored and executed in a computer system, but they may not be reproduced for publication.

This edition can be exported from India only by the publishers,
McGraw Hill Education (India) Private Limited,

ISBN 13: 978-1-25-902988-2

ISBN 10: 1-25-902988-3

Vice President and Managing Director: *Ajay Shukla*

Head—Higher Education (Publishing and Marketing): *Vibha Mahajan*

Publishing Manager (SEM & Tech. Ed.): *Shalini Jha*

Asst. Sponsoring Editor: *Smruti Snigdha*

Editorial Researcher: *Sourabh Maheshwari*

Manager—Production Systems: *Satinder S Baveja*

Asst. Manager—Editorial Services: *Sohini Mukherjee*

Sr. Production Manager: *P L Pandita*

Asst. General Manager (Marketing)—Higher Education: *Vijay Sarathi*

Sr. Product Specialist (SEM & Tech. Ed.): *Tina Jajoriya*

Sr. Graphic Designer (Cover): *Meenu Raghav*

General Manager—Production: *Rajender P Ghansela*

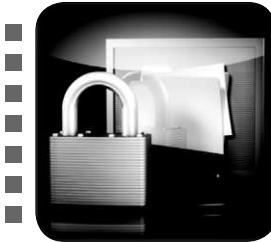
Manager—Production: *Reji Kumar*

Information contained in this work has been obtained by McGraw Hill Education (India), from sources believed to be reliable. However, neither McGraw Hill Education (India) nor its authors guarantee the accuracy or completeness of any information published herein, and neither McGraw Hill Education (India) nor its authors shall be responsible for any errors, omissions, or damages arising out of use of this information. This work is published with the understanding that McGraw Hill Education (India) and its authors are supplying information but are not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought.

Typeset at The Composers, 260, C.A. Apt., Paschim Vihar, New Delhi 110 063, and printed at
SDR Printers, A-28, West Jyoti Nagar, Loni Road, Shadara, Delhi 110 094

Cover: SDR

RYZCRRRLORQLLD



CONTENTS

<i>Preface</i>	<i>ix</i>
<i>Important Terms and Abbreviations</i>	<i>xiii</i>
1. Introduction to the Concepts of Security	1
1.1 Introduction 1	
1.2 The Need for Security 2	
1.3 Security Approaches 6	
1.4 Principles of Security 8	
1.5 Types of Attacks 12	
<i>Summary</i> 27	
<i>Key Terms and Concepts</i> 28	
<i>Practice Set</i> 29	
2. Cryptography Techniques	32
2.1 Introduction 32	
2.2 Plain Text and Cipher Text 33	
2.3 Substitution Techniques 36	
2.4 Transposition Techniques 47	
2.5 Encryption and Decryption 51	
2.6 Symmetric and Asymmetric Key Cryptography 53	
2.7 Steganography 64	
2.8 Key Range and Key Size 65	
2.9 Possible Types of Attacks 68	
<i>Case Study: Denial of Service (DOS) Attacks</i> 72	
<i>Summary</i> 74	
<i>Key Terms and Concepts</i> 75	
<i>Practice Set</i> 76	
3. Computer-based Symmetric Key Cryptographic Algorithms	80
3.1 Introduction 80	
3.2 Algorithm Types and Modes 80	
3.3 An Overview of Symmetric-Key Cryptography 92	
3.4 Data Encryption Standard (DES) 94	

3.5 International Data Encryption Algorithm (IDEA)	108
3.6 RC4	116
3.7 RC5	118
3.8 Blowfish	127
3.9 Advanced Encryption Standard (AES)	130
<i>Case Study: Secure Multiparty Calculation</i>	141
<i>Summary</i>	142
<i>Key Terms and Concepts</i>	144
<i>Practice Set</i>	145
4. Computer-based Asymmetric-Key Cryptography Algorithms	148
4.1 Introduction	148
4.2 Brief History of Asymmetric-Key Cryptography	148
4.3 An Overview of Asymmetric-Key Cryptography	149
4.4 The RSA Algorithm	151
4.5 ElGamal Cryptography	157
4.6 Symmetric- and Asymmetric-Key Cryptography	158
4.7 Digital Signatures	162
4.8 Knapsack Algorithm	193
4.9 ElGamal Digital Signature	194
4.10 Attacks on Digital Signatures	194
4.11 Problems with the Public-Key Exchange	195
<i>Case Study 1: Virtual Elections</i>	197
<i>Case Study 2: Contract Signing</i>	198
<i>Summary</i>	199
<i>Key Terms and Concepts</i>	200
<i>Practice Set</i>	200
5. Public Key Infrastructure (PKI)	204
5.1 Introduction	204
5.2 Digital Certificates	205
5.3 Private-Key Management	234
5.4 The PKIX Model	236
5.5 Public Key Cryptography Standards (PKCS)	238
5.6 XML, PKI and Security	244
<i>Case Study: Cross Site Scripting Vulnerability (CSSV)</i>	256
<i>Summary</i>	258
<i>Key Terms and Concepts</i>	259
<i>Practice Set</i>	260
6. Internet-Security Protocols	263
6.1 Introduction	263
6.2 Basic Concepts	263
6.3 Secure Socket Layer (SSL)	271
6.4 Transport Layer Security (TLS)	282
6.5 Secure Hyper Text Transfer Protocol (SHTTP)	282
6.6 Secure Electronic Transaction (SET)	283

6.7	SSL Versus SET	295
6.8	3-D Secure Protocol	296
6.9	Email Security	299
6.10	Wireless Application Protocol (WAP) Security	319
6.11	Security in GSM	322
6.12	Security in 3G	324
6.13	IEEE 802.11 Security	327
6.14	Link Security Versus Network Security	331
<i>Case Study 1: Secure Inter-branch Payment Transactions</i> 331		
<i>Case Study 2: Cookies and Privacy</i> 335		
<i>Summary</i>		336
<i>Key Terms and Concepts</i>		338
<i>Practice Set</i>		339
7.	User-Authentication Mechanisms	342
7.1	Introduction	342
7.2	Authentication Basics	342
7.3	Passwords	343
7.4	Authentication Tokens	356
7.5	Certificate-based Authentication	366
7.6	Biometric Authentication	372
7.7	Kerberos	374
7.8	Key Distribution Center (KDC)	380
7.9	Security Handshake Pitfalls	381
7.10	Single Sign On (SSO) Approaches	390
7.11	Attacks on Authentication Schemes	391
<i>Case Study: Single Sign On (SSO)</i> 392		
<i>Summary</i>		395
<i>Key Terms and Concepts</i>		396
<i>Practice Set</i>		397
8.	Practical Implementations of Cryptography/Security	400
8.1	Introduction	400
8.2	Cryptographic Solutions using Java	401
8.3	Cryptographic Solutions Using Microsoft .NET Framework	408
8.4	Cryptographic Toolkits	410
8.5	Web Services Security	411
8.6	Cloud Security	413
<i>Summary</i>		414
<i>Key Terms and Concepts</i>		415
<i>Practice Set</i>		416
9.	Network Security, Firewalls, and Virtual Private Networks (VPN)	418
9.1	Introduction	418
9.2	Brief Introduction to TCP/IP	418
9.3	Firewalls	423
9.4	IP Security	440

9.5 Virtual Private Networks (VPN)	458
9.6 Intrusion	461
<i>Case Study 1: IP Spoofing Attacks</i>	464
<i>Case Study 2: Creating a VPN</i>	466
<i>Summary</i>	467
<i>Key Terms and Concepts</i>	468
<i>Practice Set</i>	469
Appendices	472
A. Mathematical Background	472
B. Number Systems	481
C. Information Theory	486
D. Real-life Tools	488
E. Web Resources	489
F. A Brief Introduction to ASN, BER, DER	492
References	497
Index	499



PREFACE

This book has already been used by thousands of students, teachers, and IT professionals in its past edition. There is no change in the intended audience for this book. It is aimed at the same audience in the given order. The book can be used for any graduate/postgraduate course involving computer security/cryptography as a subject. It aims to explain the key concepts in cryptography to anyone who has basic understanding in computer science and networking concepts. No other assumptions are made. The new edition is updated to cover certain topics in the syllabi which were found to be covered inadequately in the earlier editions.

Computer and network security is one of the most crucial areas today. With so many attacks happening on all kinds of computer systems and networks, it is imperative that the subject be understood by students who are going to be the IT professionals of the future. Consequently, topics such as Cloud security, and Web services security have been added to this edition. The main focus of the book is to explain every topic in a very lucid fashion with plenty of diagrams. All technical terms are explained in detail.

■ SALIENT FEATURES ■

- Uses a bottom-up approach: *Cryptography* → *Network Security* → *Case Studies*
- Inclusion of new topics: *IEEE 802.11Security*, *Elgamal Cryptography*, *Cloud Security and Web Services Security*
- Improved treatment of *Ciphers*, *Digital Signatures*, *SHA-3 Algorithm*
- Practical orientation of the subject to help students for real-life implementation of the subject through integrated case studies
- Refreshed pedagogy includes
 - 150 Design/Programming Exercises
 - 160 Exercises
 - 170 Multiple-Choice Questions
 - 530 Illustrations
 - 10 Case Studies

■ CHAPTER ORGANIZATION ■

The organization of the book is as follows:

Chapter 1 introduces the basic concepts of security. It discusses the need for security, the principles of security and the various types of attacks on computer systems and networks. We discuss both the theoretical concepts behind all these aspects, as well as the practical issues and examples of each one of them. This will cement our understanding of security. Without understanding why security is required, and what is under threat, there is no point in trying to understand how to make computer systems and networks secure. A new section on wireless network attacks has been included. Some obsolete material on cookies and ActiveX controls has been deleted.

Chapter 2 introduces the concept of cryptography, the fundamental building block of computer security. Cryptography is achieved by using various algorithms. All these algorithms are based on either substitution of plain text with some cipher text, or by using certain transposition techniques, or a combination of both. The chapter then introduces the important terms of encryption and decryption. Playfair cipher and Hill cipher are covered in detail. The Diffie-Hellman Key Exchange coverage is expanded, and types of attacks are covered in detail.

Chapter 3 discusses the various issues involved in computer-based symmetric-key cryptography. We discuss stream and block cipher and the various chaining modes. We also discuss the chief symmetric-key cryptographic algorithms in great detail, such as DES, IDEA, RC5 and Blowfish. The Feistel cipher is covered in detail. Discussions related to the security of DES and attacks on the algorithm are expanded. Similarly, the security issues pertaining to AES are also covered.

Chapter 4 examines the concepts, issues and trends in asymmetric-key cryptography. We go through the history of asymmetric-key cryptography. Later, we discuss the major asymmetric-key cryptographic algorithms, such as RSA, MD5, SHA, and HMAC. We introduce several key terms, such as message digests and digital signatures in this chapter. We also study how best we can combine symmetric-key cryptography with asymmetric-key cryptography. Security issues pertaining to RSA algorithm are included. The ElGamal Cryptography and ElGamal Digital Signature schemes are covered. SHA-3 algorithm is introduced. Issues pertaining to RSA digital signature are covered.

Chapter 5 talks about the upcoming popular technology of Public Key Infrastructure (PKI). Here, we discuss what we mean by digital certificates, how they can be created, distributed, maintained and used. We discuss the role of Certification Authorities (CA) and Registration Authorities (RA). We also introduce the Public Key Cryptography Standards (PKCS). Some obsolete topics such as roaming digital certificates and attribute certificates are removed.

Chapter 6 deals with the important security protocols for the Internet. These protocols include SSL, SHTTP, TSP, SET and 3D-Secure. We also discuss how electronic money works, what are the dangers involved therein and how best we can make use of it. An extensive coverage of email security is provided with a detailed discussion of the key email security protocols, such as PGP, PEM and S/MIME. We also discuss wireless security here. The obsolete SET protocol is reduced. Discussion on 3-D Secure is expanded. Electronic money is completely removed. DomainKeys Identified Mail (DKIM) is covered. Security in IEEE 802.11 (WiFi) is discussed in detail.

Chapter 7 tells us how to authenticate a user. There are various ways to do this. The chapter examines each one of them in significantly great detail and addresses their pros and cons. We discuss password-

based authentication, authentication based on something derived from the password, authentication tokens, certificate-based authentication, and biometrics. We also study the popular Kerberos protocol. Discussion of biometric techniques is expanded. Attacks on authentication schemes are covered.

Chapter 8 deals with the practical issues involved in cryptography. Currently, the three main ways to achieve this is to use the cryptographic mechanisms provided by Sun (in the Java programming language), Microsoft, and third-party toolkits. We discuss each of these approaches. Operating systems security and database security are removed. Web services security and cloud security are added.

Chapter 9 is concerned with network-layer security. Here, we examine firewalls, their types and configurations. Then we go on to IP security, and conclude our discussion with Virtual Private Networks (VPN).

Each chapter has an introduction that explains the scope of coverage and a chapter summary at the end. There are multiple-choice and detailed questions to verify the student's understanding. Several case studies are included at appropriate places to give a practical flavor to the subject. Every difficult concept is explained using a diagram. Unnecessary mathematics is avoided wherever possible.

■ ONLINE LEARNING CENTER ■

The OLC for this book can be accessed at <https://www.mhhe.com/kahate/cns3> and contains the following material:

- **For the Student**

- Additional programming exercises of varying levels of difficulty
- Cryptography Demos with DES and AES Demo Applets
- Web References (Updated with latest links)
- Real-Life Case Studies

- **For the Instructor**

- Solutions to exercises (Updated with the new programming exercises solutions)
- Sample Question Papers
- List of Additional Material added to the text
- Web References (Interesting Links)

■ ACKNOWLEDGEMENTS ■

I would like to thank all my family members, colleagues, and friends for their help. Hundreds of students and professors have appreciated the previous editions of the book, which makes the efforts of coming up with a new edition very enjoyable. More specifically, I would like to thank my ex-students Swapnil Pandittrao and Pranav Sorte, who have helped me with the third edition. Mr Nikhil Bhalla pointed out a few errors in the earlier edition, which now stand corrected.

A sincere note of appreciation is due to all TMH members—Shalini Jha, Smruti Snigdha, Sourabh Maheshwari, Satinder Singh, Sohini Mukherjee and P L Pandita who helped me during various stages of the publication process.

I would also like to thank all those reviewers who took out time to review the book and gave useful comments. Their names are given as follows:

Vrutika Shah	<i>LEADS Institute of Technology and Engineering, Ahmedabad, Gujarat</i>
Metul Patel	<i>Shree Swami Atmanandan College of Engineering, Ahmedabad, Gujarat</i>
Amitab Nag	<i>Academy of Technology, Kolkata</i>
Subhajit Chatterjee	<i>Calcutta Institute of Engineering and Management, Kolkata</i>
Garimella Rama Murthy	<i>International Institute of Information Technology (IIIT), Hyderabad</i>

Feedback

Readers are welcome to send any feedback/comments on my Website www.atulkahate.com (in the Testimonials section) or via email at akahate@gmail.com.

Atul Kahate

Publisher's Note

Do you have any further request or a suggestion? We are always open to new ideas (the best ones come from you!). You may send your comments to tmh.csefeedback@gmail.com

Piracy-related issues may also be reported!



IMPORTANT TERMS AND ABBREVIATIONS

1-factor authentication	Authentication mechanism, which involves the party to be authenticated concerned with only one factor (e.g. <i>know</i> something).
2-factor authentication	Authentication mechanism, which involves the party to be authenticated concerned with two factors (e.g. <i>know</i> something and <i>have</i> something).
3-D Secure	Payment mechanism developed by Visa for Web-based transactions.
Active attack	Form of attack on security where the attacker makes attempts to change the contents of the message.
Algorithm mode	Defines the details of a cryptographic algorithm.
Algorithm type	Defines how much plain text should be encrypted/decrypted at a time.
Application gateway	Type of firewall that filters packets at the application layer of TCP/IP stack. Same as <i>Bastion host</i> or <i>Proxy server</i> .
Asymmetric Key Cryptography	Cryptographic technique where a key pair is used for encryption and decryption operations.
Authentication	Principle of security, which identifies a user or a computer system, so that it can be trusted.
Authentication token	Small piece of hardware used in 2-factor authentication mechanisms.
Authority Revocation List (ARL)	List of revoked Certification Authorities (CA).
Avalanche effect	The principle that determines minor changes to plaintext result into what sort of changes to the resulting ciphertext in an encryption algorithm.
Availability	Principle of security, which ensures that a resource/computer system is available to the authorized users.

Bastion host	Type of firewall that filters packets at the application layer of TCP/IP stack. Same as <i>Application gateway</i> or <i>Proxy server</i> .
Behaviour-blocking software	Software that integrates with the operating system of the computer and keeps a watch on virus-like behavior in real time.
Behavioural techniques	Biometric authentication techniques that depend on the behavioural characteristics of a human being.
Bell-LaPadula model	A highly trustworthy computer system is designed as a collection of objects and subjects. Objects are passive repositories or destinations for data, such as files, disks, printers, etc. Subjects are active entities, such as users, processes, or threads operating on behalf of those users.
Biometric authentication	Authentication mechanism that depends on the biological characteristics of a user.
Block cipher	Encrypts/decrypts a group of characters at a time.
Bucket brigade attack	A form of attack in which the attacker intercepts the communication between two parties, and fools them to believe that they are communicating with each other, whereas they actually communicate with the attacker. Same as <i>man-in-the-middle attack</i> .
Book Cipher	Cryptographic technique involving the key selected randomly from a page in a book.
Brute-force attack	Form of attack wherein the attacker tries all possible combinations of the key one after the other in quick succession.
Caesar Cipher	Cryptographic technique wherein each plain text character is replaced with an alphabet three places down the line.
Cardholder	Customer, who shops online on the Web, and makes payments for the same using a credit/debit card.
Certificate directory	Pre-specified area containing the list of digital certificates.
Certificate Management Protocol (CMP)	Protocol used in the requesting of a digital certificate.
Certificate Revocation List (CRL)	List of revoked digital certificates. It is an offline certificate checking mechanism.
Certificate Signing Request (CSR)	Format used by a user to request for a digital certificate from a CA/RA.
Certificate-based authentication	Authentication mechanism wherein the user needs to produce her digital certificate, and also has to provide a proof of possessing that certificate.
Certification Authority (CA)	Authority that can issue digital certificates to users after proper authentication checks.

Certification Authority hierarchy	Hierarchy that allows multiple CAs to operate, thereby taking load off a single CA.
Chain of trust	Mechanism whereby a trust is established from the current CA up to the root CA.
Chaining mode	Technique of adding complexity to the cipher text, making it harder to crack.
Challenge/response token	Type of authentication token.
Chosen cipher text attack	Type of attack where the attacker knows the cipher text to be decrypted, the encryption algorithm that was used to produce this cipher text, and the corresponding plain text block. The attacker's job is to discover the key used for encryption.
Chosen-message attack	A trick where the attacker makes the user believe that she signed a message using RSA, which she did not.
Chosen plain text attack	Here, the attacker selects a plain text block, and tries to look for the encryption of the same in the cipher text. Here, the attacker is able to choose the messages to encrypt. Based on this, the attacker intentionally picks patterns of cipher text that result in obtaining more information about the key.
Chosen text attack	This is essentially a combination of <i>chosen plain text attack</i> and <i>chosen cipher text attack</i> .
Cipher Block Chaining (CBC)	Mechanism of chaining.
Cipher Feedback (CFB)	Mechanism of chaining.
Cipher text	Result of encryption on a plain text message.
Cipher text only attack	In this type of attack, the attacker does not have any clue about the plain text. She/he has some or all of the cipher text.
Circuit gateway	Form of application gateway, which creates a connection between itself and the remote host/server.
Clear text	Message in an understandable/readable form, same as <i>Plain text</i> .
Collision	If two messages yield the same message digest, there is a collision.
Completeness effect	A principle that demands that every ciphertext bit should depend on more than one plaintext bits.
Confidentiality	Principle of security, which ensures that only the sender and the recipient of a message come to know about the contents of that message.
Confusion	Performing substitution during encryption.
Counter (mode)	In this algorithm mode, a counter and plain text block are encrypted together, after which the counter is incremented.

Cross-certification	Technology wherein CAs from different domains/locations sign each other's certificates, for ease of operation.
Cryptanalysis	Process of analyzing cipher text.
Cryptanalyst	Person who performs cryptanalysis.
Cryptographic toolkit	Software that provides cryptographic algorithms/operations for use in applications.
Cryptography	Art of codifying messages, so that they become unreadable.
Cryptology	Combination of cryptography and cryptanalysis.
Cycling attack	An attack where the attacker believes that plain text was converted into cipher text using some permutation, which the attacker tries on the cipher text to obtain the original plain text.
Data Encryption Standard (DES)	IBM's popular algorithm for symmetric key encryption, uses 56-bit keys, not used widely of late.
Decryption	Process of transforming cipher text back into plain text—opposite of <i>Encryption</i> .
Demilitarized Zone (DMZ)	Firewall configuration that allows an organization to securely host its public servers and also protect its internal network at the same time.
Denial Of Service (DOS) attack	An attempt by an attacker to disallow authorized users from accessing a resource/computer system.
Dictionary attack	Attack wherein the attacker tries all the possible words from the dictionary (e.g. as a password).
Differential cryptanalysis	Method of cryptanalysis that looks at pairs of cipher text whose plain texts have particular differences.
Diffusion	Performing transposition during encryption.
Digital cash	Computer file representing the equivalent of real cash. Bank debits the user's real bank account and issues digital cash, instead. Same as <i>electronic cash</i> .
Digital certificate	Computer file similar to a paper-based passport, links a user to a particular public key, and also provides other information about the user.
Digital envelope	Technique wherein the original message is encrypted with a one-time session key, which itself is encrypted with the intended recipient's public key.
Digital Signature Algorithm (DSA)	Asymmetric key algorithm for performing digital signatures.
Digital Signature Standard (DSS)	Standard specifying how digital signature should be done.
DNS spoofing	See <i>Pharming</i> .

DomainKeys Identified Mail (DKIM)	An Internet email scheme where the user's email system digitally signs an email message to confirm that it originated from there.
Double DES	Modified version of DES, involves 128-bit keys.
Dual signature	Mechanism used in the Secure Electronic Transaction (SET) protocol whereby the payment details are hidden from the merchant, and the purchase details are hidden from the payment gateway.
Dynamic packet filter	Type of packet filter, which keeps learning from the current status of the network.
ElGamal	A set of schemes for encryption and digital signature.
Electronic Code Book (ECB)	Mechanism of chaining.
Electronic money	See <i>Electronic cash</i> .
Encryption	Process of transforming plain text into cipher text—opposite of <i>Decryption</i> .
Fabrication	False message created by an attacker to distort the attention of the authorized users.
Factorization attack	Factorizing a number into its two prime factors is very difficult if the number is large. An attacker would still attempt it to break the security of the RSA algorithm, which is based on this principle.
Feistel Cipher	A cryptographic technique that uses substitution and transposition alternatively to produce cipher text.
Firewall	Special type of router, which can perform security checks and allows rule-based filtering.
Hash	<i>Finger print</i> of a message, same as <i>Message digest</i> . Identifies a message uniquely.
Hill Cipher	Hill cipher works on multiple letters at the same time. Hence, it is a type of polygraphic substitution cipher.
HMAC	Similar to a message digest, HMAC also involves encryption.
Homophonic Substitution Cipher	Technique of encryption in which one plain text character is replaced with one cipher text character, at a time. The cipher text character is not fixed.
Integrity	Principle of security, which specifies that the contents of a message must not be altered during its transmission from the sender to the receiver.
Interception	Process of an attacker getting hold of a message in transit, before it reaches the intended recipient.
International Data Encryption Algorithm (IDEA)	A symmetric key encryption algorithm, developed in 1990's.

Internet Security Association and Key Management Protocol (ISAKMP)	Protocol used in IPSec for key management. Also called as Oakley.
Interruption	Attacker creating a situation where the availability of a system is in danger. Same as <i>Masquerade</i> .
IP Security (IPSec)	Protocol to encrypt messages at the network layer.
Issuer	Bank/financial institution that facilitates a cardholder to make credit card payments on the Internet.
Jamming attack	A Denial-of-Service attack on wireless networks that introduces unnecessary wireless frames.
Java Cryptography Architecture (JCA)	Java's cryptography mechanism, in the form of APIs.
Java Cryptography Extensions (JCE)	Java's cryptography mechanism, in the form of APIs.
Kerberos	Single Sign On (SSO) mechanism, that allows a user to have a single user id and password to access multiple resources/systems.
Key	The secret information in a cryptographic operation.
Key Distribution Center (KDC)	A central <i>authority</i> dealing with keys for individual computers (nodes) in a computer network.
Key-only attack	Only using a genuine user's public key, the attacker attempts an attack.
Key wrapping	See <i>Digital envelope</i> .
Known plaintext attack	In this case, the attacker knows about some pairs of plain text and corresponding cipher text for those pairs. Using this information, the attacker tries to find other pairs, and therefore, know more and more of the plain text.
Lightweight Directory Access Protocol (LDAP)	Protocol that allows easy storage and retrieval of information at/from a central place.
Linear cryptanalysis	An attack based on linear approximations.
Low decryption exponent attack	If the decryption key value used in RSA is very small, the attacker can guess it better.
Lucifer	One symmetric key encryption algorithm.
Man-in-the-middle attack	A form of attack in which the attacker intercepts the communication between two parties, and fools them to believe that they are communicating with each other, whereas they actually communicate with the attacker. Same as <i>bucket brigade attack</i> .
Masquerade	Attacker creating a situation where the availability of a system is in danger. Same as <i>Interruption</i> .
MD5	Message digest algorithm, now seems vulnerable to attacks.
Message Authentication Code (MAC)	See <i>HMAC</i> .

Message digest	<i>Finger print</i> of a message, same as <i>Hash</i> . Identifies a message uniquely.
Microsoft Cryptography Application Programming Interface (MS-CAPI)	Microsoft's cryptography mechanism, in the form of APIs.
Modification	Attack on a message where its contents are changed.
Mono-alphabetic Cipher	Technique of encryption in which one plain text character is replaced with one cipher text character, at a time.
Multi-factor authentication	Authentication mechanism, which involves the party to be authenticated concerned with multiple factors (e.g. <i>know</i> something, <i>be</i> something and <i>have</i> something).
Mutual authentication	In mutual authentication, A and B both authenticate each other.
Network level attack	Security attacks attempted at the network/hardware level.
Non-repudiation	Provision whereby the sender of a message cannot refuse having sent it, later on, in the case of a dispute.
One-Time Pad	Considered very secure, this method involves the usage of a key, which is used only once and then discarded forever.
One-time password	Technology that authenticates user based on passwords that are generated dynamically, used once, and then destroyed.
One-way authentication	In this scheme, if there are two users A and B, B authenticates A, but A does not authenticate B.
Online Certificate Status Protocol (OCSP)	Online protocol to check the status of a digital certificate.
Output Feedback (OFB)	Mode of chaining.
Packet filter	Firewall that filters individual packets based on rules. Works at the network layer.
Passive attack	Form of attack on security where the attacker does not make an attempt to change the contents of the message.
Password	Authentication mechanism that requires a user to enter a secret piece of information (i.e. the password) when challenged.
Password policy	Statement outlining the structure, rules and mechanisms of passwords, in an organization.
Person-in-the-middle attack	A form of wireless attack, where the attacker plays a role that is quite different from the real identity of the attacker.
Pharming	Modifying the Domain Name System (DNS) so as to direct genuine URLs to false IP addresses of attackers.
Phishing	Technique used by attackers to fool innocent users into providing confidential/personal information.

Physiological techniques	Biometric authentication techniques that depends on the physical characteristics of a human being.
Plain text	Message in an understandable/readable form, same as <i>Clear text</i> .
Playfair Cipher	A cryptographic technique that is used for manual encryption of data. This scheme was invented by Charles Wheatstone in 1854.
Polygram Substitution Cipher	Technique of encryption where one block of plain text is replaced with another, at a time.
Pretty Good Privacy (PGP)	Protocol for secure email communications, developed by Phil Zimmerman.
Privacy Enhanced Mail (PEM)	Protocol for secure email communications, developed by Internet Architecture Board (IAB).
Proof Of Possession (POP)	Establishing the proof that a user possesses the private key corresponding to the public key, as specified in the user's digital certificate.
Proxy server	Type of firewall that filters packets at the application layer of TCP/IP stack. Same as <i>Application gateway</i> or <i>Bastion host</i> .
Pseudocollision	Specific case of collision in the MD5 algorithm.
Psuedo-random number	Random number generated using computers.
Public Key Cryptography Standards (PKCS)	Standards developed by RSA Security Inc for the Public Key Infrastructure (PKI) technology.
Public Key Infrastructure (PKI)	Technology for implementing asymmetric key cryptography, with the help of message digests, digital signatures, encryption and digital certificates.
Public Key Infrastructure X.509 (PKIX)	Model to implement PKI.
Rail Fence Technique	Example of transposition technique.
RC5	Symmetric key block encryption algorithm, involving variable length keys.
Reference monitor	Central entity, which is responsible for all the decisions related to access control of computer systems.
Registration Authority (RA)	Agency that takes some of the jobs of a Certification Authority (CA) on itself, and helps the CA in many ways.
Replay attack	Attack on a system wherein the attacker gets hold of a message, and attempts to re-send it, hoping that the receiver does not detect this as a message sent twice.
Revealed decryption exponent attack	If the attacker can guess the decryption key in RSA, it is called with this name.
Roaming certificate	Digital certificate, which can be carried along as users move from one computer/location to another.

RSA algorithm	Asymmetric key algorithm, widely used for encryption and digital signatures.
Running Key Cipher	Technique where some portion of text from a book is used as the key.
Secure Electronic Transaction (SET)	Protocol developed jointly by MasterCard, Visa and many other companies for secure credit card payments on the Internet.
Secure MIME (S/MIME)	Protocol that adds security to the basic Multipurpose Internet Mail Extensions (MIME) protocol.
Secure Socket Layer (SSL)	Protocol developed by Netscape Communications for secure exchange of information between a Web browser and a Web server over the Internet.
Self-signed certificate	Digital certificate, wherein the subject name and the issuer name are the same, and is signed by the issuer (which is also the subject). Usually the case only with CA certificates.
SHA	Message digest algorithm, now preferred as the standard algorithm of choice.
Short message attack	With the assumption that the attacker knows some small part of the plain text, the attack involves comparing this plain text with small blocks of cipher text to find relationship between the two.
Simple Certificate Validation protocol (SCVP)	Enhancement of the basic Online Certificate Status Protocol (OCSP). Allows checks other than only the status of the certificate, unlike OCSP.
Simple Columnar Transposition Technique	Variation of the basic transposition technique such as Rail Fence Technique.
Simple Columnar Transposition Technique with multiple rounds	Variation of Simple Columnar Transposition Technique.
Single Sign On (SSO)	Technology providing the users a single user id and password to access multiple systems/applications.
Stream cipher	Technique of encrypting one bit at a time.
Substitution Cipher	Cryptographic technique involving the replacement of plain text characters with other characters.
Symmetric Key Cryptography	Cryptographic technique where the same key is used for encryption and decryption operations.
Time Stamping Authority (TSA)	Notary-like authority, which can vouch for the availability/creation of a digital document at a particular point of time.
Time Stamping Protocol (TSP)	Protocol using which a Time Stamping Authority (TSP) vouches for the availability/creation of a digital document at a particular point of time.

Time-based token	Type of authentication token.
Traffic analysis	Mechanism whereby an attacker examines the packets moving across a network, and uses this information to launch an attack.
Transport Layer Security (TLS)	Protocol similar to SSL.
Transposition Cipher	Cryptographic technique involving the re-arrangement of plain text characters in some other form.
Triple DES	Modified version of DES, involves 128-bit or 168-bit keys.
Trojan horse	Small program that does not attempt to delete anything on the user's disk, but instead, replicates itself on the computer/networks.
Trusted system	Computer system that can be trusted to a certain extent in terms of implementing the designated security policy.
Unconcealed message attack	In some very rare cases, encrypting plain text gives cipher text that is the same as the original plain text. Since the plain text can thus not be hidden, it is called with this name.
Vernam Cipher	See <i>One-time pad</i> .
Virtual Private Network (VPN)	Technology that makes use of the existing Internet as a private network, using cryptographic techniques.
Virus	Small program that causes harm to a user's computer and performs destructive activities.
Wireless Equivalent Privacy (WEP)	A weak algorithm that attempts to provide encryption-based security in a wireless network.
WiFi Protected Access (WPA)	A wireless security scheme that overcomes the drawbacks of WEP and provides for authentication, encryption, and message integrity.
Wireless Transport Layer Security (WTLS)	Layer in WAP for facilitating secure communications between a client and a server.
Worm	Small program, which does not damage a computer/network, but consumes resources, slowing it down considerably.
WS-Security	Set of standards for protecting Web Services.
X.500	Standard name for the LDAP technology.
X.509	Format for digital certificate contents and structure.
XML digital signatures	Technology that allows signing of specific portions of a message.



INTRODUCTION TO THE CONCEPTS OF SECURITY

■ 1.1 INTRODUCTION ■

This is a book on network and Internet security. As such, before we embark on our journey of understanding the various concepts and technical issues related to security (i.e. trying to understand *how* to protect), it is essential to know *what* we are trying to protect. What are the various dangers when we use computers, computer networks, and the biggest network of them all, the Internet? What are the likely pitfalls? What can happen if we do not set up the right security policies, framework and technology implementations? This chapter attempts to provide answers to these basic questions.

We start with a discussion of the basic question: Why is security required in the first place? People sometimes say that security is like statistics: what it reveals is trivial, what it conceals is vital! In other words, the right security infrastructure opens up just enough doors that are mandatory. It protects everything else. We discuss a few real-life incidents that should prove beyond doubt that security cannot simply be compromised. Especially these days, when serious business and other types of transactions are being conducted over the Internet to such a large extent, that inadequate or improper security mechanisms can bring the whole business down, or play havoc with people's lives!

We then discuss the key principles of security. These principles help us identify the various areas, which are crucial while determining the security threats and possible solutions to tackle them. Since electronic documents and messages are now becoming equivalent to the paper documents in terms of their legal validity and binding, we examine the various implications in this regard.

This is followed by a discussion on the types of attacks. There are certain theoretical concepts associated with attacks, and there is a practical side to it as well. We shall discuss all these aspects.

Finally, we discuss the outline and scope of the rest of the book. This will pave the way for further discussions of network and Internet security concepts.

■ 1.2 THE NEED FOR SECURITY ■

1.2.1 Basic Concepts

Most previous computer applications had *no*, or at best, *very little* security. This continued for a number of years until the importance of data was truly realized. Until then, computer data was considered to be useful, but not something to be protected. When computer applications were developed to handle financial and personal data, the real need for security was felt like never before. People realized that data on computers is an extremely important aspect of modern life. Therefore, various areas in security began to gain prominence. Two typical examples of such security mechanisms were as follows:

- Provide a user identification and password to every user, and use that information to authenticate a user.
- Encode information stored in the databases in some fashion, so that it is not visible to users who do not have the right permission.

Organizations employed their own mechanisms in order to provide for these kinds of basic security mechanisms. As technology improved, the communication infrastructure became extremely mature, and newer applications began to be developed for various user demands and needs. Soon, people realized the basic security measures were not quite enough.

Furthermore, the Internet took the world by storm. There were many examples of what could happen if there was insufficient security built in applications developed for the Internet. Figure 1.1 shows such an example of what can happen when you use your credit card for making purchases over the Internet. From the user's computer, the user details such as user id, order details such as order id and item id,

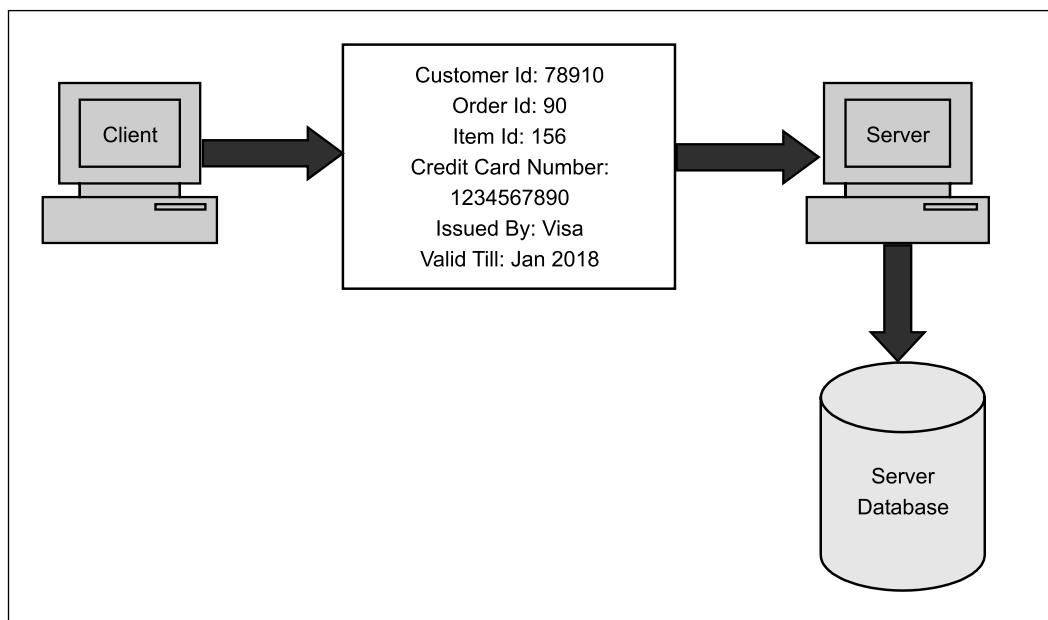


Fig. 1.1 Example of information traveling from a client to a server over the Internet

and payment details such as credit-card information travel across the Internet to the server (i.e. to the merchant's computer). The merchant's server stores these details in its database.

There are various security holes here. First of all, an intruder can capture the credit-card details as they travel from the client to the server. If we somehow protect this transit from an intruder's attack, it still does not solve our problem. Once the merchant computer receives the credit-card details and validates them so as to process the order and later obtain payments, the merchant computer stores the credit-card details into its database. Now, an attacker can simply succeed in accessing this database, and therefore gain access to all the credit-card numbers stored therein! One Russian attacker (called 'Maxim') actually managed to intrude into a merchant Internet site and obtained 300,000 credit-card numbers from its database. He then attempted extortion by demanding protection money (\$100,000) from the merchant. The merchant refused to oblige. Following this, the attacker published about 25,000 of the credit-card numbers on the Internet! Some banks reissued all the credit cards at a cost of \$20 per card, and others forewarned their customers about unusual entries in their statements.

Such attacks could obviously lead to great losses—both in terms of finance and goodwill. Generally, it takes \$20 to replace a credit card. Therefore, if a bank has to replace 3,00,000 such cards, the total cost of such an attack is about \$6 million! How helpful would it have been, if the merchant in the example just discussed had employed proper security measures!

Of course, this was just one example. Several such cases have been reported in the last few months, and the need for proper security is being felt increasingly with every such attack. In another example of security attack, in 1999, a Swedish hacker broke into Microsoft's Hotmail Web site, and created a mirror site. This site allowed anyone to enter any Hotmail user's email id, and read his/her emails!

In 1999, two independent surveys were conducted to invite people's opinions about the losses that occur due to successful attacks on security. One survey pegged the losses figuring at an average of \$256,296 per incident, and the other one's average was \$759,380 per incident. Next year, this figure rose to \$972,857!

1.2.2 Modern Nature of Attacks

If we attempt to demystify technology, we would realize that computer-based systems are not all that different from what happens in the real world. Changes in computer-based systems are mainly due to the speed at which things happen and the accuracy that we get, as compared to the traditional world.

We can highlight a few salient features of the modern nature of attacks, as follows:

1. Automating Attacks

The speed of computers make several attacks worthwhile for miscreants. For example, in the real world, let's suppose someone manages to create a machine that can produce counterfeit coins. Would that bother authorities? It certainly would. However, producing so many coins on a mass scale may not be that much economical compared to the return on that investment! How many such coins would the attacker be able to get into the market so rapidly? But, the scenario is quite different with computers. They are quite efficient and happy in doing routine, mundane, repetitive tasks. For example, they would excel in somehow stealing a very low amount (say half a dollar or 20 rupees) from a million bank accounts in a matter of a few minutes. This would give the attacker a half million dollars possibly without any major complaints! This is shown in Fig. 1.2.

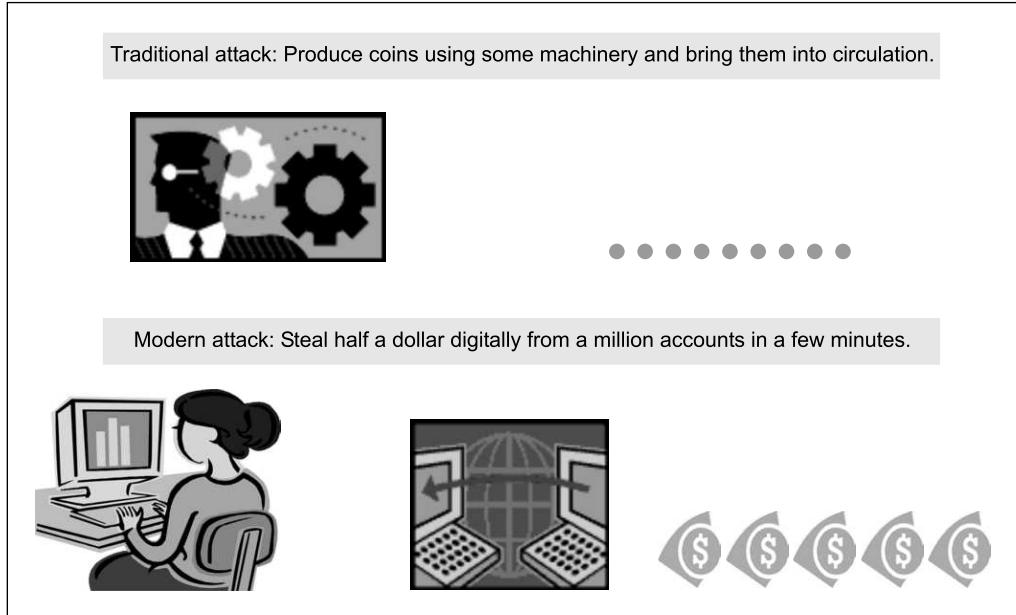


Fig. 1.2 The changing nature of attacks due to automation

The morale of the story is:

Humans dislike mundane and repetitive tasks. Automating them can cause financial destruction or a security nuisance quite rapidly.

2. Privacy Concerns

Collecting information about people and later (mis)using it is turning out to be a huge problem these days. The so-called *data mining* applications gather, process, and tabulate all sorts of details about individuals. People can then illegally sell this information. For example, companies like Experian (formerly TRW), TransUnion, and Equifax maintain credit history of individuals in the USA. Similar trends are seen in the rest of the world. These companies have volumes of information about a majority of citizens of that country. These companies can collect, collate, polish, and format all sorts of information to whosoever is ready to pay for that data! Examples of information that can come out of this are: which store the person buys more from, which restaurant he/she eats in, where he/she goes for vacations frequently, and so on! Every company (e.g. shopkeepers, banks, airlines, insurers) are collecting and processing a mind-boggling amount of information about us, without us realizing when and how it is going to be used.

3. Distance Does not Matter

Thieves would earlier attack banks, because banks had money. Banks do not have money today! Money is in digital form inside computers, and moves around by using computer networks. Therefore, a modern thief would perhaps not like to wear a mask and attempt a robbery! Instead, it is far easier and cheaper to attempt an attack on the computer systems of the bank while sitting at home! It may be far

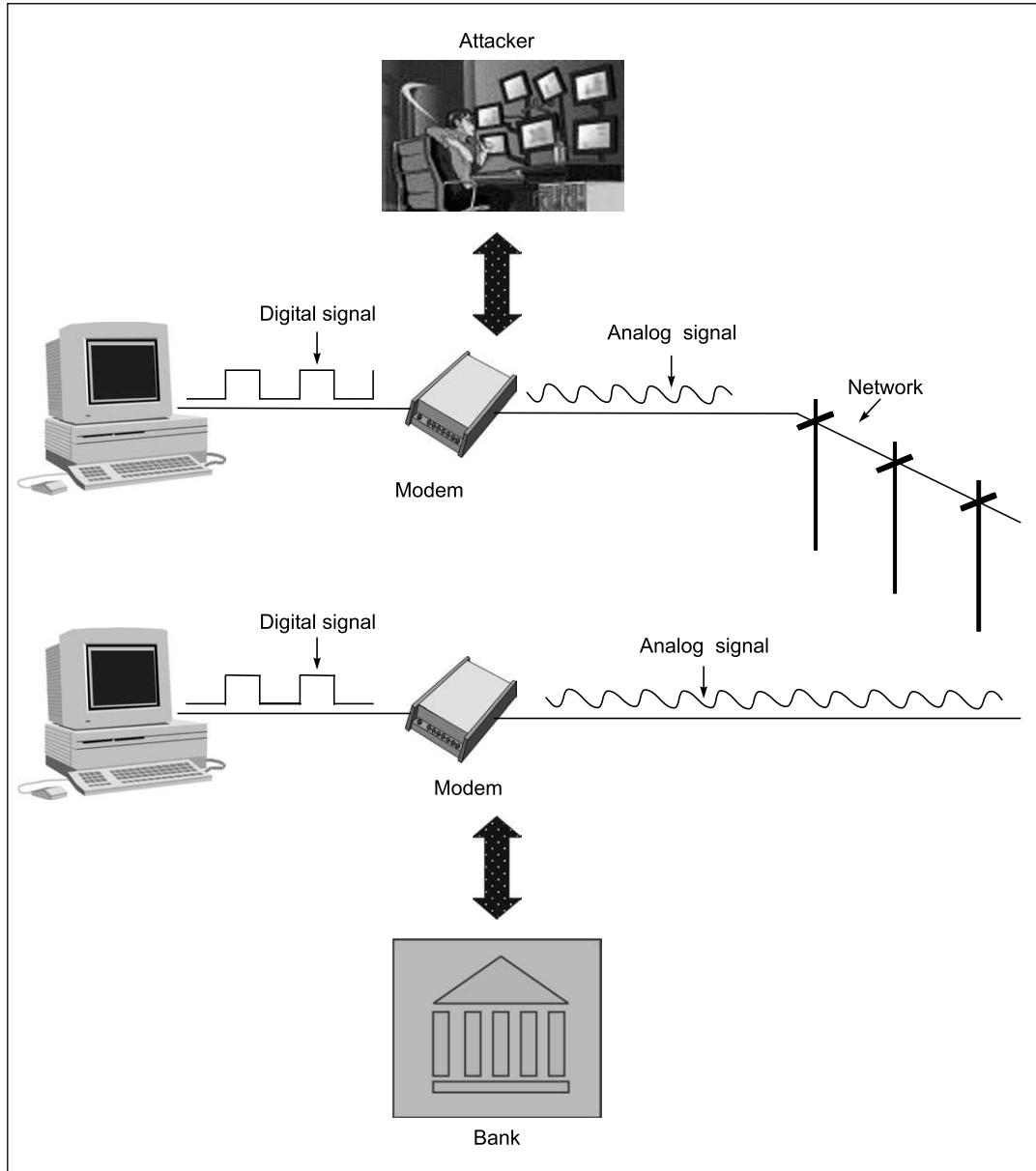


Fig. 1.3 Attacks can now be launched from a distance

more prudent for the attacker to break into the bank's servers, or steal credit card/ATM information from the comforts of his/her home or place of work. This is illustrated in Fig. 1.3.

In 1995, a Russian hacker broke into Citibank's computers remotely, stealing \$12 million. Although the attacker was traced, it was very difficult to get him extradited for the court case.

■ 1.3 SECURITY APPROACHES ■

1.3.1 Trusted Systems

A **trusted system** is a computer system that can be trusted to a specified extent to enforce a specified security policy.

Trusted systems were initially of primary interest to the military. However, these days, they have spanned across various areas, most prominently in the banking and financial community, but the concept never caught on. Trusted systems often use the term **reference monitor**. This is an entity that is at the logical heart of the computer system. It is mainly responsible for all the decisions related to access controls. Naturally, following are the expectations from the reference monitor:

- (a) It should be tamper-proof.
- (b) It should always be invoked.
- (c) It should be small enough so that it can be tested independently.

In their 1983 *Orange Book* (also called the *Trusted Computer System Evaluation Criteria (TCSEC)*), the National Security Agency (NSA) of the US Government defined a set of *evaluation classes*. These described the features and assurances that the user could expect from a trusted system.

The highest levels of assurance were provided by significant efforts directed towards reduction of the size of the trusted computing base, or TCB. In this context, TCB was defined as a combination of hardware, software, and firmware responsible for enforcing the system's security policy. The lower the TCB, the higher the assurance. However, this raises an inherent problem (quite similar to the decisions related to the designing of operating systems). If we make the TCB as small as possible, the surrounding hardware, software, and firmware are likely to be quite big!

The mathematical foundation for trusted systems was provided by two relatively independent yet interrelated works. In the year 1974, David Bell and Leonard LaPadula of MITRE devised a technique called the **Bell-LaPadula model**. In this model, a highly trustworthy computer system is designed as a collection of objects and subjects. Objects are passive repositories or destinations for data, such as files, disks, printers, etc. Subjects are active entities, such as users, processes, or threads operating on behalf of those users. Subjects cause information to flow among objects.

Around the same time, Dorothy Denning at Purdue University was preparing for her doctorate. It dealt with *lattice-based information flows* in computer systems. A mathematical *lattice* is a partially ordered set, in which the relationship between any two vertices either *dominates*, *is dominated by* or *neither*. She devised a generalized notion of *labels*—similar to the full security markings on classified military documents. Examples of this are TOP SECRET.

Later, Bell and LaPadula integrated Denning's theory into their MITRE technical report, which was titled *Secure Computer System: Unified Exposition and Multics Interpretation*. Here, *labels* attached to *objects* represented the sensitivity of data contained within the *object*. Interestingly, the Bell–LaPadula model talks only about *confidentiality* or *secrecy* of information. It does not talk about the problem of *integrity* of information.

1.3.2 Security Models

An organization can take several approaches to implement its security model. Let us summarize these approaches.

1. No Security

In this simplest case, the approach could be a decision to implement no security at all.

2. Security through Obscurity

In this model, a system is secure simply because nobody knows about its existence and contents. This approach cannot work for too long, as there are many ways an attacker can come to know about it.

3. Host Security

In this scheme, the security for each host is enforced individually. This is a very safe approach, but the trouble is that it cannot scale well. The complexity and diversity of modern sites/organizations makes the task even harder.

4. Network Security

Host security is tough to achieve as organizations grow and become more diverse. In this technique, the focus is to control network access to various hosts and their services, rather than individual host security. This is a very efficient and scalable model.

1.3.3 Security-Management Practices

Good **security-management practices** always talk of a **security policy** being in place. Putting a security policy in place is actually quite tough. A good security policy and its proper implementation go a long way in ensuring adequate security-management practices. A good security policy generally takes care of four key aspects, as follows.

- **Affordability** How much money and effort does this security implementation cost?
- **Functionality** What is the mechanism of providing security?
- **Cultural Issues** Does the policy complement the people's expectations, working style and beliefs?
- **Legality** Does the policy meet the legal requirements?

Once a security policy is in place, the following points should be ensured.

- (a) Explanation of the policy to all concerned.
- (b) Outline everybody's responsibilities.
- (c) Use simple language in all communications.
- (d) Accountability should be established.
- (e) Provide for exceptions and periodic reviews.

■ 1.4 PRINCIPLES OF SECURITY ■

Having discussed some of the attacks that have occurred in real life, let us now classify the principles related to security. This will help us understand the attacks better, and also help us in thinking about the possible solutions to tackle them. We shall take an example to understand these concepts.

Let us assume that a person A wants to send a check worth \$100 to another person B. Normally, what are the factors that A and B will think of, in such a case? A will write the check for \$100, put it inside an envelope, and send it to B.

- A will like to ensure that no one except B gets the envelope, and even if someone else gets it, he/she does not come to know about the details of the check. This is the principle of **confidentiality**.
- A and B will further like to make sure that no one can tamper with the contents of the check (such as its amount, date, signature, name of the payee, etc.). This is the principle of **integrity**.
- B would like to be assured that the check has indeed come from A, and not from someone else posing as A (as it could be a fake check in that case). This is the principle of **authentication**.
- What will happen tomorrow if B deposits the check in his/her account, the money is transferred from A's account to B's account, and then A refuses having written/sent the check? The court of law will use A's signature to disallow A to refute this claim, and settle the dispute. This is the principle of **non-repudiation**.

These are the four chief principles of security. There are two more: **access control** and **availability**, which are not related to a particular message, but are linked to the overall system as a whole.

We shall discuss all these security principles in the next few sections.

1.4.1 Confidentiality

The principle of *confidentiality* specifies that only the sender and the intended recipient(s) should be able to access the contents of a message. Confidentiality gets compromised if an unauthorized person is able to access a message. An example of compromising the confidentiality of a message is shown in Fig. 1.4. Here, the user of computer A sends a message to the user of computer B. (Actually, from here

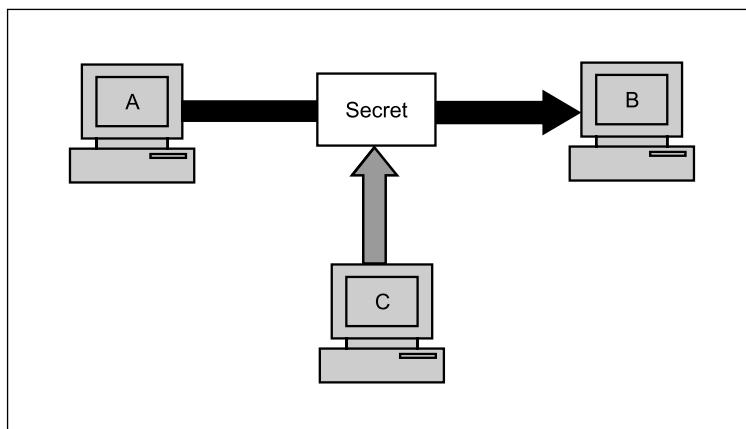


Fig. 1.4 Loss of confidentiality

onwards, we shall use the term **A** to mean the *user A*, **B** to mean *user B*, etc., although we shall just show the computers of users A, B, etc.). Another user C gets access to this message, which is not desired, and therefore defeats the purpose of confidentiality. An example of this could be a confidential email message sent by A to B, which is accessed by C without the permission or knowledge of A and B. This type of attack is called **interception**.

Interception causes loss of message confidentiality.

1.4.2 Authentication

Authentication mechanisms help establish **proof of identities**. The authentication process ensures that the origin of an electronic message or document is correctly identified. For instance, suppose that user C sends an electronic document over the Internet to user B. However, the trouble is that user C had posed as user A when he/she sent this document to user B. How would user B know that the message has come from user C, who is posing as user A? A real-life example of this could be the case of a user C, posing as user A, sending a funds transfer request (from A's account to C's account) to bank B. The bank might happily transfer the funds from A's account to C's account—after all, it would think that user A has requested for the funds transfer! This concept is shown in Fig. 1.5. This type of attack is called **fabrication**.

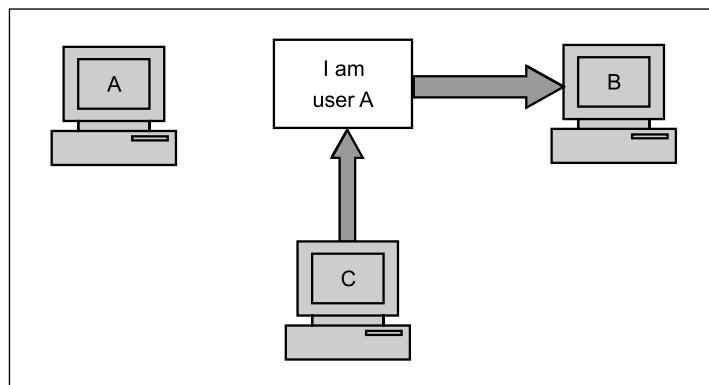


Fig. 1.5 Absence of authentication

Fabrication is possible in absence of proper authentication mechanisms.

1.4.3 Integrity

When the contents of a message are changed after the sender sends it, but before it reaches the intended recipient, we say that the *integrity* of the message is lost. For example, suppose you write a check for \$100 to pay for goods bought from the US. However, when you see your next account statement, you are startled to see that the check resulted in a payment of \$1000! This is the case for loss of message integrity. Conceptually, this is shown in Fig. 1.6. Here, user C tampers with a message originally sent by user A, which is actually destined for user B. User C somehow manages to access it, change its contents, and send the changed message to user B. User B has no way of knowing that the contents of

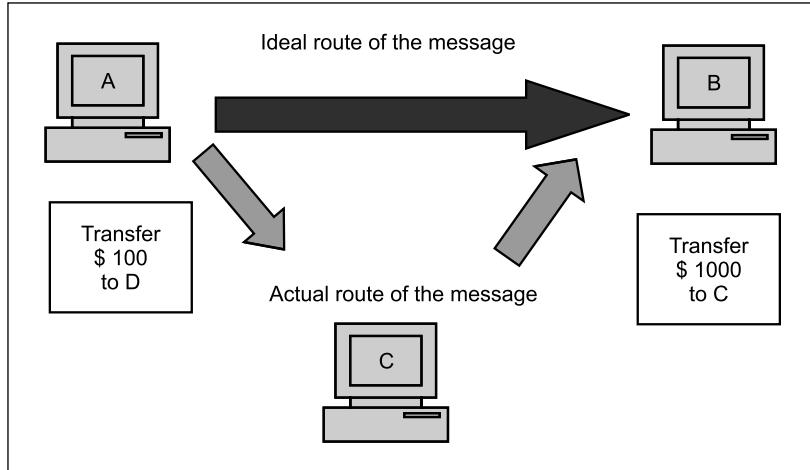


Fig. 1.6 Loss of integrity

the message were changed after user A had sent it. User A also does not know about this change. This type of attack is called **modification**.

Modification causes loss of message integrity.

1.4.4 Non-repudiation

There are situations where a user sends a message, and later on refuses that she had sent that message. For instance, user A could send a funds transfer request to bank B over the Internet. After the bank performs the funds transfer as per A's instructions, A could claim that he/she never sent the funds transfer instruction to the bank! Thus, A repudiates, or denies, his/her funds transfer instruction. The principle of *non-repudiation* defeats such possibilities of denying something after having done it. This is shown in Fig. 1.7.

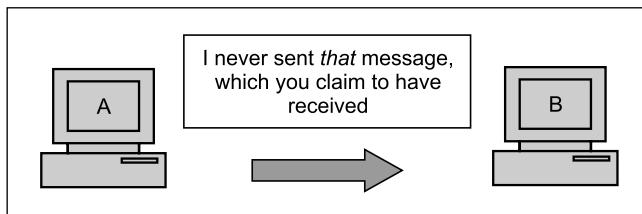


Fig. 1.7 Establishing non-repudiation

Non-repudiation does not allow the sender of a message to refute the claim of not sending that message.

1.4.5 Access Control

The principle of *access control* determines *who* should be able to access *what*. For instance, we should be able to specify that user A can view the records in a database, but cannot update them. However, user B might be allowed to make updates as well. An access-control mechanism can be set up to ensure this. Access control is broadly related to two areas: *role management* and *rule management*. Role management concentrates on the user side (which user can do what), whereas rule management focuses on the resources side (which resource is accessible, and under what circumstances). Based on the decisions taken here, an access-control matrix is prepared, which lists the users against a list of items they can access (e.g. it can say that user A can write to file X, but can only update files Y and Z). An **Access Control List (ACL)** is a subset of an access-control matrix.

Access control specifies and controls who can access what.

1.4.6 Availability

The principle of *availability* states that resources (i.e. information) should be available to authorized parties at all times. For example, due to the intentional actions of another unauthorized user C, an authorized user A may not be able to contact a server computer B, as shown in Fig. 1.8. This would defeat the principle of availability. Such an attack is called **interruption**.

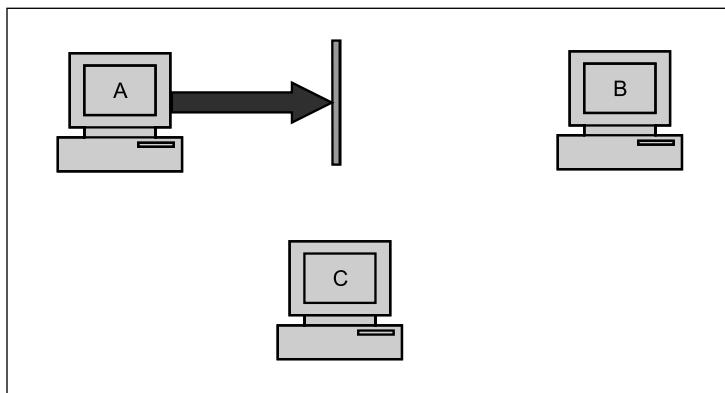


Fig. 1.8 Attack on availability

Interruption puts the availability of resources in danger.

We may be aware of the traditional OSI standard for Network Model (titled OSI Network Model 7498-1), which describes the seven layers of the networking technology (application, presentation, session, transport, network, data link, and physical). A very less known standard on similar lines is the **OSI standard for Security Model** (titled OSI Security Model 7498-2). This also defines seven layers of security in the form of

- Authentication
- Access control

- Non-repudiation
- Data integrity
- Confidentiality
- Assurance or availability
- Notarization or signature

We shall be touching upon most of these topics in this book.

Having discussed the various principles of security, let us now discuss the various types of attacks that are possible, from a technical perspective.

1.4.7 Ethical and Legal Issues

Many ethical issues (and legal issues) in computer security systems seem to be in the area of the individual's right to privacy versus the greater good of a larger entity (e.g. a company, society, etc.) Some examples are tracking how employees use computers for crowd surveillance, managing customer profiles, tracking a person's travel with a passport, so as to spam their cell phone with text-message advertisements), and so on. A key concept in resolving this issue is to find out a person's expectation of privacy.

Classically, the ethical issues in security systems are classified into the following four categories:

Privacy This deals with the right of an individual to control personal information.

Accuracy This talks about the responsibility for the authenticity, fidelity, and accuracy of information.

Property Here, we find out the owner of the information. We also talk about who controls access.

Accessibility This deals with the issue of what information does an organization have the right to collect? And in that situation, it also expects to know what the measures are, which will safeguard against any unforeseen eventualities.

Privacy is the protection of personal or sensitive information. Individual privacy is the desire to be *left alone* as an extension of our *personal space* and may or may not be supported by local regulations or laws. Privacy is subjective. Different people have different ideas of what privacy is and how much privacy they will trade for safety or convenience.

When dealing with legal issues, we need to remember that there is a hierarchy of regulatory bodies that govern the legality of information security. We can roughly classify them as follows.

- **International**, e.g. International Cybercrime Treaty
- **Federal**, e.g. FERPA, GLB, HIPAA, DMCA, Teach Act, Patriot Act, Sarbanes-Oxley Act, etc.
- **State**, e.g. UCITA, SB 1386, etc.
- **Organization**, e.g. computer use policy

■ 1.5 TYPES OF ATTACKS ■

We shall classify attacks with respect to two views: the common person's view and a technologist's view.

1.5.1 Attacks: A General View

From a common person's point of view, we can classify attacks into three categories, as shown in Fig. 1.9.

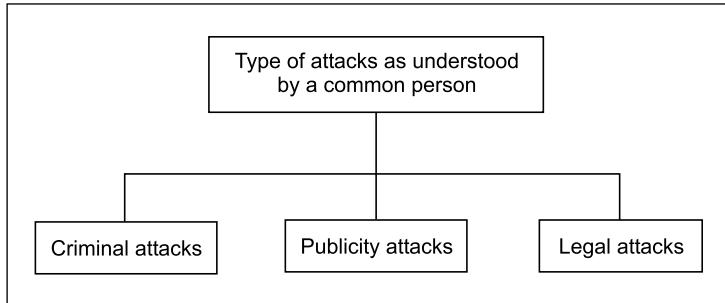


Fig. 1.9 Classification of attacks as understood in general terms

Let us now discuss these attacks.

1. Criminal Attacks

Criminal attacks are the simplest to understand. Here, the sole aim of the attackers is to maximize financial gain by attacking computer systems. Table 1.1 lists some forms of criminal attacks.

2. Publicity Attacks

Publicity attacks occur because the attackers want to see their names appear on television news channels and newspapers. History suggests that these types of attackers are usually not hardcore criminals. They are people such as students in universities or employees in large organizations, who seek publicity by adopting a novel approach of attacking computer systems.

One form of publicity attacks is to damage (or deface) the Web pages of a site by attacking it. One of the most famous of such attacks occurred on the *US Department of Justice's* Web site in 1996. The *New York Times* home page was also infamously defaced two years later.

3. Legal Attacks

This form of attack is quite novel and unique. Here, the attacker tries to make the judge or the jury doubtful about the security of a computer system. This works as follows. The attacker attacks the computer system, and the attacked party (say a bank or an organization) manages to take the attacker to the court. While the case is being fought, the attacker tries to convince the judge and the jury that there is inherent weakness in the computer system and that she has done nothing wrongful. The aim of the attacker is to exploit the weakness of the judge and the jury in technological matters.

For example, an attacker may sue a bank for performing an online transaction, which he/she never wanted to perform. In court, the attacker could innocently say something like: *The bank's Web site asked me to enter a password and that is all that I provided; I do not know what happened thereafter.* A judge is unwittingly likely to sympathize with the attacker!

Table 1.1 Types of criminal attacks

Attack	Description
Fraud	Modern fraud attacks concentrate on manipulating some aspects of electronic currency, credit cards, electronic stock certificates, checks, letters of credit, purchase orders, ATMs, etc.
Scams	Scams come in various forms, some of the most common ones being sale of services, auctions, multilevel marketing schemes, general merchandise, and business opportunities, etc. People are enticed to send money in return of great returns, but end up losing their money. A very common example is the <i>Nigeria scam</i> , where an email from Nigeria (and other African countries) entices people to deposit money into a bank account with a promise of hefty gains. Whosoever gets caught in this scam loses money heavily.
Destruction	Some sort of grudge is the motive behind such attacks. For example, unhappy employees attack their own organization, whereas terrorists strike at much bigger levels. For example, in the year 2000, there was an attack against popular Internet sites such as Yahoo!, CNN, eBay, Buy.com, Amazon.com, and e*Trade where authorized users of these sites failed to log in or access these sites.
Identity theft	This is best understood with a quote from Bruce Schneier: <i>Why steal from someone when you can just become that person?</i> In other words, an attacker does not steal anything from a legitimate user—he/she becomes that legitimate user! For example, it is much easier to get the password of someone else's bank account, or to actually be able to get a credit card on someone else's name. Then that privilege can be misused until it gets detected.
Intellectual property theft	Intellectual property theft ranges from stealing companies' trade secrets, databases, digital music and videos, electronic documents and books, software, and so on.
Brand theft	It is quite easy to set up fake Web sites that look like real Web sites. How would a common user know if he/she is visiting the HDFC Bank site or an attacker's site? Innocent users end up providing their secrets and personal details on these fake sites to the attackers. The attackers use these details to then access the real site, causing an <i>identity theft</i> .

1.5.2 Attacks: A Technical View

From a technical point of view, we can classify the types of attacks on computers and network systems into two categories for better understanding: (a) Theoretical concepts behind these attacks, and (b) Practical approaches used by the attackers. Let us discuss these one by one.

1. Theoretical Concepts

As we discussed earlier, the principles of security face threat from various attacks. These attacks are generally classified into four categories, as mentioned earlier. These are the following:

Interception It has been discussed in the context of *confidentiality* earlier. It means that an unauthorized party has gained access to a resource. The party can be a person, program, or computer-based system. Examples of interception are copying of data or programs, and listening to network traffic.

Fabrication It has been discussed in the context of *authentication* earlier. This involves the creation of illegal objects on a computer system. For example, the attacker may add fake records to a database.

Modification It has been discussed in the context of *integrity* earlier. Here, the attacker may modify the values in a database.

Interruption It has been discussed in the context of *availability* earlier. Here, the resource becomes unavailable, lost, or unusable. Examples of interruption are causing problems to a hardware device, erasing program, data, or operating-system components.

These attacks are further grouped into two types: **passive attacks** and **active attacks**, as shown in Fig. 1.10.

Let us discuss these two types of attacks now.

(a) Passive Attacks *Passive attacks* are those wherein the attacker indulges in eavesdropping or monitoring of data transmission. In other words, the attacker aims to obtain information that is in transit. The term *passive* indicates that the attacker does not attempt to perform any modifications to the data. In fact, this is also why passive attacks are harder to detect. Thus, the general approach to deal with passive attacks is to think about prevention, rather than detection or corrective actions.

Passive attacks do not involve any modifications to the contents of an original message.

Figure 1.11 shows further classification of passive attacks into two sub-categories. These categories are, namely **release of message contents** and **traffic analysis**.

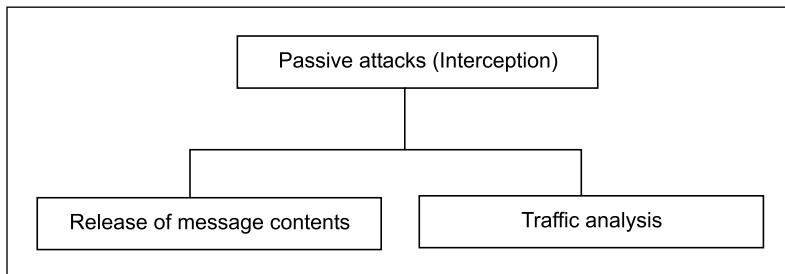


Fig. 1.11 Passive attacks

Release of message contents is quite simple to understand. When you send a confidential email message to your friend, you desire that only he/she be able to access it. Otherwise, the contents of the message are released against our wishes to someone else. Using certain security mechanisms, we can prevent the *release of message contents*. For example, we can encode messages using a code language, so that only the desired parties understand the contents of a message, because only they know the code language. However, if many such messages are passing through, a passive attacker could try to figure out similarities between them to come up with some sort of pattern that provides her some clues regarding the communication that is taking place. Such attempts of analyzing (encoded) messages to come up with likely patterns are the work of the *traffic-analysis* attack.

(b) Active Attacks Unlike *passive attacks*, the *active attacks* are based on the modification of the original message in some manner, or in the creation of a false message. These attacks cannot be prevented easily. However, they can be detected with some effort, and attempts can be made to recover from them. These attacks can be in the form of interruption, modification and fabrication.

In active attacks, the contents of the original message are modified in some way.

- Trying to pose as another entity involves **masquerade** attacks.
- Modification attacks can be classified further into **replay attacks** and **alteration of messages**.
- Fabrication causes **Denial Of Service (DOS)** attacks.

This classification is shown in Fig. 1.12.

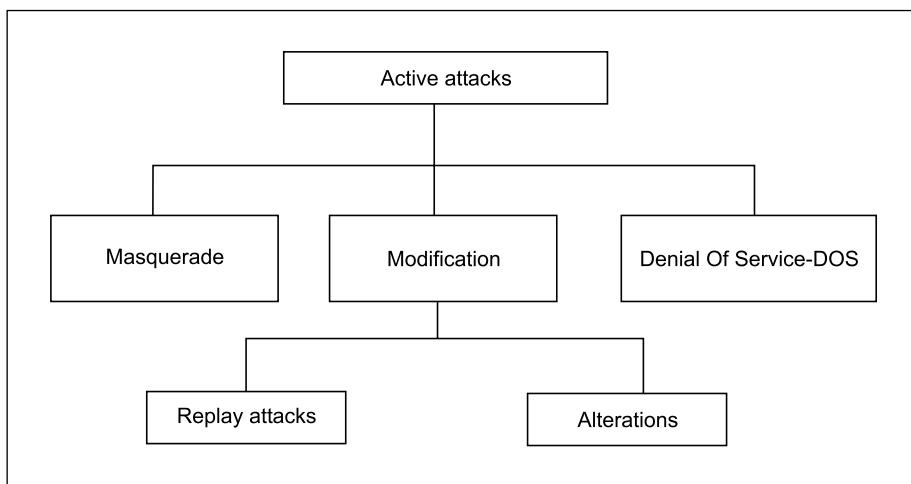


Fig. 1.12 Active attacks

Masquerade is caused when an unauthorized entity pretends to be another entity. As we have seen, user C might pose as user A and send a message to user B. User B might be led to believe that the message indeed came from user A. In masquerade attacks, an entity poses as another entity. In masquerade attacks, usually some other forms of active attacks are also embedded. As an instance, the attack may involve capturing the user's authentication sequence (e.g. user ID and password). Later, those details can be replayed to gain illegal access to the computer system.

In a *replay attack*, a user captures a sequence of events, or some data units, and re-sends them. For instance, suppose user A wants to transfer some amount to user C's bank account. Both users A and C have accounts with bank B. User A might send an electronic message to bank B, requesting for the funds transfer. User C could capture this message, and send a second copy of the same to bank B. Bank B would have no idea that this is an unauthorized message, and would treat this as a second, and *different*, funds transfer request from user A. Therefore, user C would get the benefit of the funds transfer twice: once authorized, once through a replay attack.

Alteration of messages involves some change to the original message. For instance, suppose user A sends an electronic message *Transfer \$1000 to D's account* to bank B. User C might capture this, and change it to

Transfer \$10000 to C's account. Note that both the beneficiary and the amount have been changed—instead, only one of these could have also caused alteration of the message.

Denial Of Service (DOS) attacks make an attempt to prevent legitimate users from accessing some services, which they are eligible for. For instance, an unauthorized user might send too many login requests to a server using random user ids in quick succession, so as to flood the network and deny other legitimate users to use the network facilities.

1.5.3 The Practical Side of Attacks

The attacks discussed earlier can come in a number of forms in real life. They can be classified into two broad categories: application-level attacks and network-level attacks, as shown in Fig. 1.13.

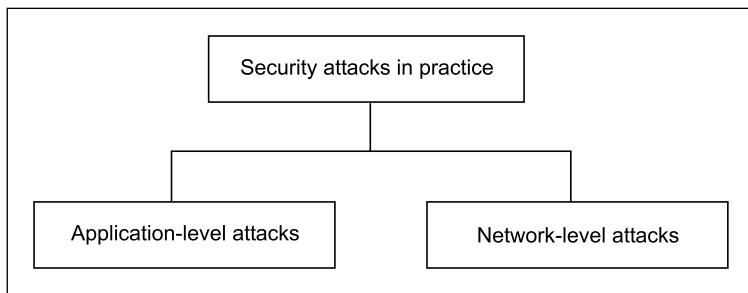


Fig. 1.13 Practical side of attacks

Let us discuss these now.

1. Application-level Attacks

These attacks happen at an application level in the sense that the attacker attempts to access, modify, or prevent access to information of a particular application, or the application itself. Examples of this are trying to obtain someone's credit-card information on the Internet, or changing the contents of a message to change the amount in a transaction, etc.

2. Network-level Attacks

These attacks generally aim at reducing the capabilities of a network by a number of possible means. These attacks generally make an attempt to either slow down, or completely bring to halt, a computer network. Note that this automatically can lead to application-level attacks, because once someone is able to gain access to a network, usually he/she is able to access/modify at least some sensitive information, causing havoc.

These two types of attacks can be attempted by using various mechanisms, as discussed next. We will not classify these attacks into the above two categories, since they can span across application as well as network levels.

Security attacks can happen at the application level or the network level.

1.5.4 Programs that Attack

Let us now discuss a few programs that attack computer systems to cause some damage or to create confusion.

1. Virus

One can launch an application-level attack or a network level attack using a **virus**. In simple terms, a virus is a piece of program code that attaches itself to legitimate program code, and runs when the legitimate program runs. It can then infect other programs in that computer, or programs that are in other computers but on the same network. This is shown in Fig. 1.14. In this example, after deleting all the files from the current user's computer, the virus self-propagates by sending its code to all users whose email addresses are stored in the current user's address book.

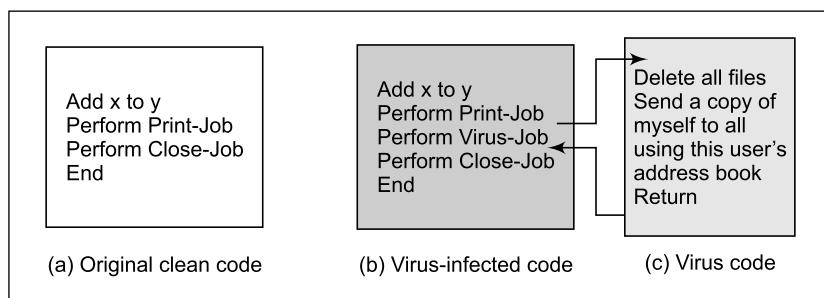


Fig. 1.14 Virus

Viruses can also be triggered by specific events (e.g. a virus could automatically execute at 12 p.m. every day). Usually viruses cause damage to computer and network systems to the extent that they can be repaired, assuming that the organization deploys good backup and recovery procedures.

A virus is a computer program that attaches itself to another legitimate program, and causes damage to the computer system or to the network.

During its lifetime, a virus goes through four phases:

(a) Dormant Phase Here, the virus is idle. It gets activated based on a certain action or event (e.g. the user typing a certain key or a certain date or time is reached, etc). This is an optional phase.

(b) Propagation Phase In this phase, a virus copies itself, and each copy starts creating more copies of itself, thus propagating the virus.

(c) Triggering Phase A dormant virus moves into this phase when the action/event for which it was waiting is initiated.

(d) Execution Phase This is the actual work of the virus, which could be harmless (display some message on the screen) or destructive (delete a file on the disk).

Viruses can be classified into the following categories:

(a) Parasitic Virus This is the most common form of virus. Such a virus attaches itself to executable files and keeps replicating. Whenever the infected file is executed, the virus looks for other executable files to attach itself and spread.

(b) Memory-resident Virus This type of virus first attaches itself to an area of the main memory and then infects every executable program that is executed.

(c) Boot sector Virus This type of virus infects the master boot record of the disk and spreads on the disk when the operating system starts booting the computer.

(d) Stealth Virus This virus has intelligence built in, which prevents anti-virus software programs from detecting it.

(e) Polymorphic Virus A virus that keeps changing its signature (i.e. identity) on every execution, making it very difficult to detect.

(f) Metamorphic Virus In addition to changing its signature like a polymorphic virus, this type of virus keeps rewriting itself every time, making its detection even harder.

There is another popular category of viruses, called the **macro virus**. This virus affects specific application software, such as Microsoft Word or Microsoft Excel. They affect the documents created by users, and spread quite easily since such documents are very commonly exchanged over email. There is a feature called *macro* in these application-software programs, which allows users to write small, useful, utility programs within the documents. Viruses attack these macros, and hence the name *macro virus*.

2. Worm

Similar in concept to a virus, a **worm** is actually different in implementation. A virus modifies a program (i.e. it attaches itself to the program under attack). A worm, however, does not modify a program. Instead, it replicates itself again and again. This is shown in Fig. 1.15. The replication grows so much that ultimately the computer or the network on which the worm resides, becomes very slow, ultimately coming to a halt. Thus, the basic purpose of a worm attack is different from that of a virus. A worm attack attempts to make the computer or the network under attack unusable by eating all its resources.

A worm does not perform any destructive actions, and instead, only consumes system resources to bring it down.

3. Trojan Horse

A Trojan horse is a hidden piece of code, like a virus. However, the purpose of a Trojan horse is different. Whereas the main purpose of a virus is to make some sort of modifications to the target computer or network, a Trojan horse attempts to reveal confidential information to an attacker. The name (Trojan horse) comes from the epic poem *Iliad*. The story says that Greek soldiers hid inside a large hollow horse, which was pulled into the city of Troy by its citizens, unaware of its *contents*. Once the Greek soldiers entered the city of Troy, they opened the gates for the rest of the Greek soldiers.

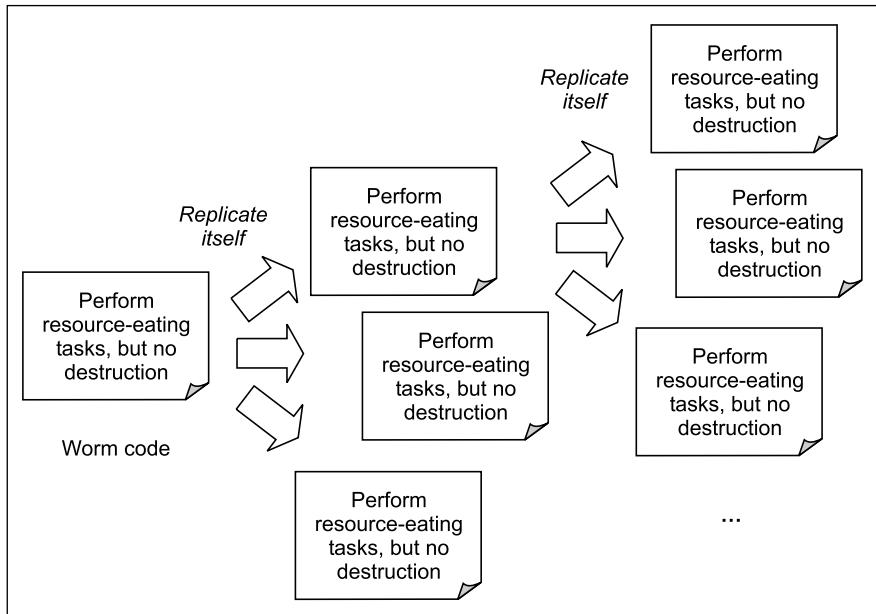


Fig. 1.15 Worm

In a similar fashion, a Trojan horse could silently sit in the code for a *Login* screen by attaching itself to it. When the user enters the user id and password, the Trojan horse could capture these details, and send this information to the attacker without the knowledge of the user who had entered the id and password. The attacker can then merrily misuse the user id and password to gain access to the system. This is shown in Fig. 1.16.

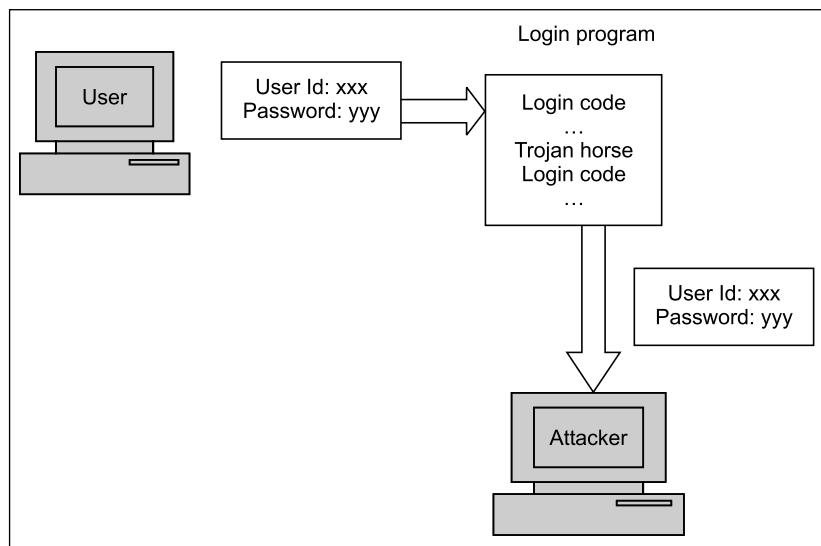


Fig. 1.16 Trojan horse

A Trojan horse allows an attacker to obtain some confidential information about a computer or a network.

1.5.5 Dealing with Viruses

Preventing viruses is the best option. However, in today's world, it is almost impossible to achieve cent per cent security given that the world is connected to the Internet all the time. We have to accept that viruses will attack, and we would need to find ways to deal with them. Hence, we can attempt to detect, identify, and remove viruses. This is shown in Fig. 1.17.

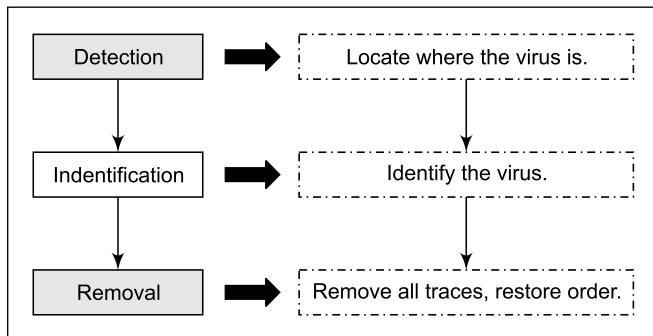


Fig. 1.17 Virus-elimination steps

Detection of viruses involves locating the virus, having known that a virus has attacked. Then we need to *identify* the specific virus that has attacked. Finally, we need to *remove* it. For this, we need to remove all traces of the virus and restore the affected programs/files to their original states. This is done by anti-virus software.

Anti-virus software is classified into four generations, as depicted in Fig. 1.18.

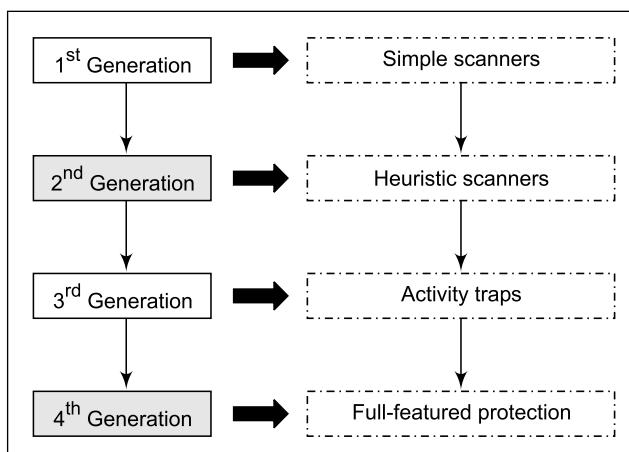


Fig. 1.18 Generations of Anti-virus software

Let us summarize the key characteristics of the four generations of anti-virus software.

1. First Generation

These anti-virus software programs were called *simple scanners*. They needed a virus signature to identify a virus. A variation of such programs kept a watch on the length of programs and looked for changes so as to possibly identify a virus attack.

2. Second Generation

These anti-virus software programs did not rely on simple virus signatures. Rather, they used heuristic rules to look for possible virus attacks. The idea was to look for code blocks that were commonly associated with viruses. For example, such a program could look for an encryption key used by a virus, find it, decrypt and remove the virus, and clean the code. Another variation of these anti-virus programs used to store some identification about the file (e.g. a message digest, which we shall study later) are also notorious for detecting changes in the contents of the file.

3. Third Generation

These anti-virus software programs were memory resident. They watched for viruses based on actions, rather than their structure. Thus, it is not necessary to maintain a large database of virus signatures. Instead, the focus is to keep watch on a small number of suspect actions.

4. Fourth Generation

These anti-virus software programs package many anti-virus techniques together (e.g. scanners, activity monitoring). They also contain access control features, thus thwarting the attempts of viruses to infect files.

There is a category of software called **behavior-blocking software**, which integrates with the operating system of the computer and keeps a watch on virus-like behavior in real time. Whenever such an action is detected, this software blocks it, preventing damages. The actions under watch can be

- Opening, viewing, modifying, deleting files
- Network communications
- Modification of settings such as start-up scripts
- Attempts to format disks
- Modification of executable files
- Scripting of email and instant messaging to send executable content to others

The main advantage of such software programs is that they are more into *virus prevention* than *virus detection*. In other words, they stop viruses before they can do any damage, rather than detecting them after an attack.

1.5.6 Specific Attacks

1. Sniffing and Spoofing

On the Internet, computers exchange messages with each other in the form of small groups of data, called packets. A packet, like a postal envelope contains the actual data to be sent, and the addressing information. Attackers target these packets, as they travel from the source computer to the destination computer over the Internet. These attacks take two main forms: (a) **Packet sniffing** (also called **snooping**), and (b) **Packet spoofing**. Since the protocol used in this communication is called Internet Protocol (IP), other names for these two attacks are (a) **IP sniffing**, and (b) **IP spoofing**. The meaning remains the same.

Let us discuss these two attacks.

(a) Packet Sniffing Packet sniffing is a passive attack on an ongoing conversation. An attacker need not *hijack* a conversation, but instead, can simply observe (i.e. *sniff*) packets as they pass by. Clearly, to prevent an attacker from sniffing packets, the information that is passing needs to be protected in some ways. This can be done at two levels: (i) The data that is traveling can be encoded in some ways, or (ii) The transmission link itself can be encoded. To read a packet, the attacker somehow needs to access it in the first place. The simplest way to do this is to control a computer via which the traffic goes through. Usually, this is a router. However, routers are highly protected resources. Therefore, an attacker might not be able to attack it, and instead, attack a less-protected computer on the same path.

(b) Packet Spoofing In this technique, an attacker sends packets with an incorrect source address. When this happens, the receiver (i.e. the party who receives these packets containing false addresses) would inadvertently send replies back to this forged address (called **spoofed address**), and not to the attacker. This can lead to three possible cases:

- (i) *The attacker can intercept the reply* If the attacker is between the destination and the forged source, the attacker can see the reply and use that information for *hijacking* attacks.
- (ii) *The attacker need not see the reply* If the attacker's intention was a Denial Of Service (DOS) attack, the attacker need not bother about the reply.
- (iii) *The attacker does not want the reply* The attacker could simply be *angry* with the host, so it may put that host's address as the forged source address and send the packet to the destination. The attacker does not want a reply from the destination, as it wants the host with the forged address to receive it and get confused.

2. Phishing

Phishing has become a big problem in recent times. In 2004, the estimated losses due to phishing were to the tune of USD 137 million, according to Tower Group. Attackers set up fake Web sites, which look like real Web sites. It is quite simple to do so, since creating Web pages involves relatively simple technologies such as HTML, JavaScript, CSS (Cascading Style Sheets), etc. Learning and using these technologies is quite simple. The attacker's modus operandi works as follows.

- The attacker decides to create his/her own Web site, which looks very identical to a real Web site. For example, the attacker can clone Citibank's Web site. The cloning is so clever that the human eye will not be able to distinguish between the real (Citibank's) and fake (attacker's) site.

- The attacker can use many techniques to attack the bank's customers. We illustrate the most common one below.

The attacker sends an email to the legitimate customers of the bank. The email itself appears to have come from the bank. For ensuring this, the attacker exploits the email system to suggest that the sender of the email is some bank official (e.g. accountmanager@citibank.com). This fake email warns the user that there has been some sort of attack on Citibank's computer systems and that the bank wants to issue new passwords to all its customers, or verify their existing PINs, etc. For this purpose, the customer is asked to visit a URL mentioned in the same email. This is conceptually shown in Fig. 1.19.

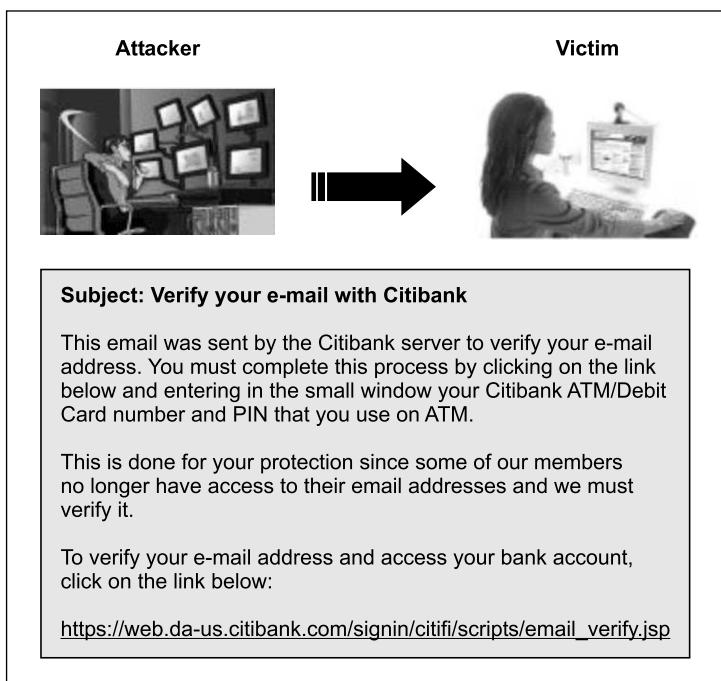


Fig. 1.19 Attacker sends a forged email to the innocent victim (customer)

- When the customer (i.e. the victim) innocently clicks on the URL specified in the email, he/she is taken to the attacker's site, and not the bank's original site. There, the customer is prompted to enter confidential information, such as his/her password or PIN. Since the attacker's fake site looks exactly like the original bank site, the customer provides this information. The attacker gladly accepts this information and displays a *Thank you* to the unsuspecting victim. In the meanwhile, the attacker now uses the victim's password or PIN to access the bank's real site and can perform any transaction as if he/she is the victim!

A real-life example of this kind of attack is reproduced below from the site <http://www.fraudwatchinternational.com>.

Figure 1.20 shows a fake email sent by an attacker to an authorized PayPal user.

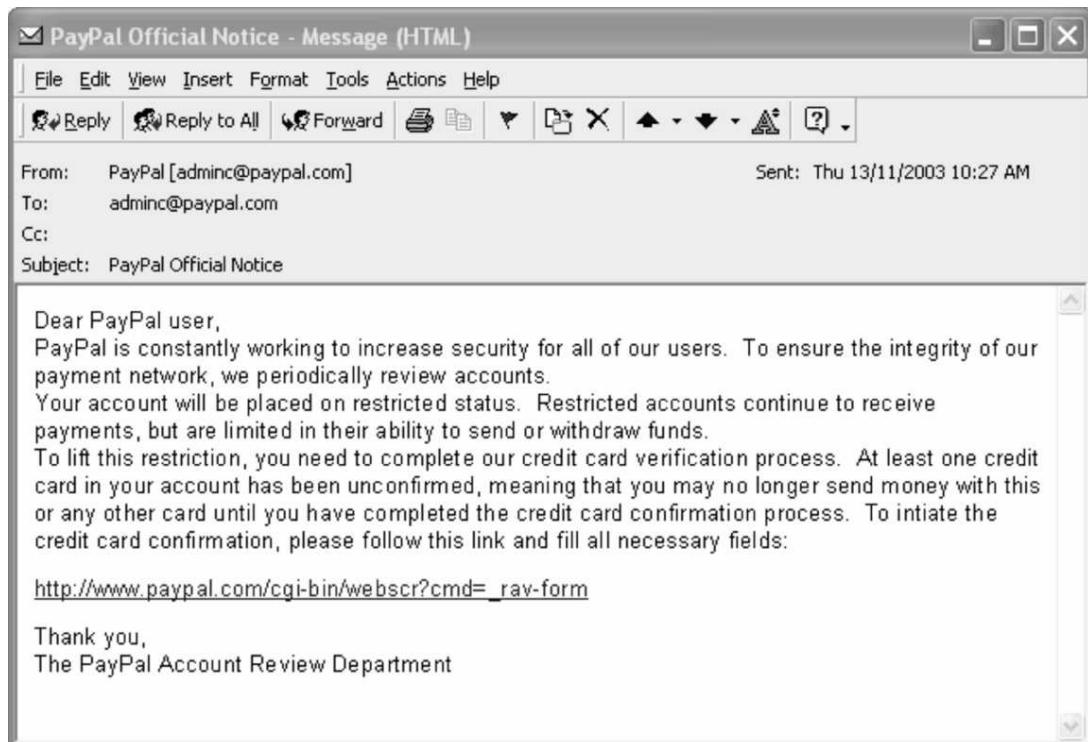


Fig. 1.20 Fake email from the attacker to a PayPal user

As we can see, the attacker is trying to fool the PayPal customer to verify his/her credit-card details. Quite clearly, the aim of the attacker is to access the credit-card information of the customer and then misuse it. Figure 1.21 shows the screen that appears when the user clicks on the URL specified in the fake email.

Once the user provides these details, the attacker's job is easy! He/she simply uses these credit-card details to make purchases on behalf of the cheated card holder!

3. **Pharming (DNS Spoofing)**

Another attack, known earlier as **DNS spoofing** or **DNS poisoning**, is now called **pharming** attack. As we know, using the **Domain Name System (DNS)**, people can identify Web sites with human-readable names (such as www.yahoo.com), and computers can continue to treat them as IP addresses (such as 120.10.81.67). For this, a special server computer called a DNS server maintains the mappings between domain names and the corresponding IP addresses. The DNS server could be located anywhere. Usually, it is with the Internet Service Provider (ISP) of the users. With this background, the DNS spoofing attack works as follows.

- Suppose that there is a merchant (Bob) whose site's domain name is www.bob.com, and the IP address is 100.10.10.20. Therefore, the DNS entry for Bob in all the DNS servers is maintained as follows:

www.bob.com 100.10.10.20

PayPal - Random Account Verification - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites Media Go Links SnagIt

Address http://123.456.789.123/paypal/ Sign Up | Log In | Help

Welcome | Send Money | Request Money | Merchant Tools | Auction Tools

Random Account Verification Secure Verification

Your credit/debit card information along with your personal information will be verified instantly.

All the data is protected by the industry standard SSL encryption. All information is required and is kept confidential in accordance with PayPal Privacy Policy.

*Your credit/debit card information is needed to verify your identity.

Account information

***All the fields are necessary.

Email :

Password :

Credit/debit card information

Card type : Visa Credit Debit

Issue Bank Name :

Card number :

Expiration date : mm/yyyy

CVV code : 3 last digits at the back of your card; next to signature

Name on card :

Billing address :

Country : USA

City :

State/Province :

Zip/Postal-code :

Telephone :

Verify you are the true holder of this card

Bank routing number :

Checking account number :

PIN-code : 4 Digit code used in ATM's

SSN : Social Security Number

MMN : Mother's Maiden Name

DOB : mm/dd/yyyy Date Of Birth

[About](#) | [Accounts](#) | [Fees](#) | [Privacy](#) | [Security Center](#) | [User Agreement](#) | [Developers](#) | [Referrals](#) | [Shops](#)
an eBay company
Copyright © 1999-2003 PayPal. All rights reserved.
[Information about FDIC pass-through insurance](#)

Fig. 1.21 Fake PayPal site asking for user's credit-card details

- The attacker (say, Trudy) manages to hack and replace the IP address of Bob with her own (say 100.20.20.20) in the DNS server maintained by the ISP of a user, say Alice. Therefore, the DNS server maintained by the ISP of Alice now has the following entry:

www.bob.com 100.20.20.20

Thus, the contents of the hypothetical DNS table maintained by the ISP would be changed. A hypothetical portion of this table (before and after the attack) is shown in Fig. 1.22.

DNS Name	IP Address	DNS Name	IP Address
www.amazon.com	161.20.10.16	www.amazon.com	161.20.10.16
www.yahoo.com	121.41.67.89	www.yahoo.com	121.41.67.89
www.bob.com	100.10.10.20	www.bob.com	100.20.20.20
...

Before the attack After the attack

Fig. 1.22 Effect of the DNS attack

- When Alice wants to communicate with Bob's site, her Web browser queries the DNS server maintained by her ISP for Bob's IP address, providing it the domain name (i.e. www.bob.com). Alice gets the replaced (i.e. Trudy's) IP address, which is 100.20.20.20.
- Now, Alice starts communicating with Trudy, believing that she is communicating with Bob!

Such attacks of DNS spoofing are quite common, and cause a lot of havoc. Even worse, the attacker (Trudy) does not have to listen to the conversation on the wire! She has to simply be able to hack the DNS server of the ISP and replace a single IP address with her own!

A protocol called **DNSSec (Secure DNS)** is being used to thwart such attacks. Unfortunately, it is not widely used.



Summary

- Network and Internet security has gained immense prominence in the last few years, as conducting business using these technologies have become very crucial.
- Automation of attacks, privacy concerns, and distance becoming immaterial are some of the key characteristics of modern attacks.
- The principles of any security mechanism are confidentiality, authentication, integrity, non-repudiation, access control, and availability.
- Confidentiality specifies that only the sender and the intended recipients should be able to access the contents of a message.
- Authentication identifies the user of a computer system, and builds a trust with the recipient of a message.

- Integrity of a message should be preserved as it travels from the sender to the recipient. It is compromised if the message is modified during transit.
- Non-repudiation ensures that the sender of a message cannot refute the fact of sending that message in case of disputes.
- Access control specifies what users can do with a network or Internet system.
- Availability ensures that computer and network resources are always available to the legitimate users.
- Attacks on a system can be classified into interception, fabrication, modification, and interruption.
- Common way of classifying attacks is to categorize them into criminal, publicity, and legal attacks.
- Attacks can also be classified into passive and active attacks.
- In passive attacks, the attacker does not modify the contents of a message.
- Active attacks involve modification of the contents of a message.
- Release of message contents and traffic analysis are types of passive attacks.
- Masquerade, replay attacks, alteration of messages and Denial Of Service (DOS) are types of active attacks.
- Another way to classify attacks is application-level attacks and network-level attacks.
- Viruses, worms, Trojan horses and Java applets, ActiveX controls can practically cause attacks on a computer system.
- Java offers a high amount of security in programming, if implemented correctly.
- Sniffing and spoofing cause packet-level attacks.
- Phishing is a new attack which attempts to fool legitimate users to provide their confidential information to fake sites.
- DNS spoofing or pharming attack involves changing the DNS entries so that users are redirected to an invalid site, while they keep thinking that they have connected to the right site.



Key Terms and Concepts

- | | |
|----------------------------------|-------------------------|
| ● Access Control List (ACL) | ● Active attack |
| ● ActiveX control | ● Alteration of message |
| ● Application-level attack | ● Attacker |
| ● Authentication | ● Availability |
| ● Behavior-blocking software | ● Confidentiality |
| ● Denial Of Service (DOS) attack | ● Fabrication |
| ● Identity theft | ● Integrity |
| ● Interception | ● Interruption |
| ● Java applet | ● Masquerade |
| ● Modification | ● Network-level attack |

- Non-repudiation
- Phishing
- Release of message contents
- Signed Java applet
- Trojan horse
- Worm
- Passive attack
- Pharming
- Replay attack
- Traffic analysis
- Virus



PRACTICE SET

■ Multiple-Choice Questions

1. The principle of _____ ensures that only the sender and the intended recipients have access to the contents of a message.
 - (a) confidentiality
 - (b) authentication
 - (c) integrity
 - (d) access control
2. If the recipient of a message has to be satisfied with the identify of the sender, the principle of _____ comes into picture.
 - (a) confidentiality
 - (b) authentication
 - (c) integrity
 - (d) access control
3. If we want to ensure the principle of _____, the contents of a message must not be modified while in transit.
 - (a) confidentiality
 - (b) authentication
 - (c) integrity
 - (d) access control
4. The principle of _____ ensures that the sender of a message cannot later claim that the message was never sent.
 - (a) access control
 - (b) authentication
 - (c) availability
 - (d) non-repudiation
5. The _____ attack is related to confidentiality.
 - (a) interception
 - (b) fabrication
 - (c) modification
 - (d) interruption
6. The _____ attack is related to authentication.
 - (a) interception
 - (b) fabrication
 - (c) modification
 - (d) interruption
7. The _____ attack is related to integrity.
 - (a) interception
 - (b) fabrication
 - (c) modification
 - (d) interruption
8. The _____ attack is related to availability.
 - (a) interception
 - (b) fabrication
 - (c) modification
 - (d) interruption
9. In _____ attacks, there is no modification to message contents.
 - (a) passive
 - (b) active
 - (c) both of the above
 - (d) none of the above

10. In _____ attacks, the message contents are modified.
- passive
 - active
 - both of the above
 - none of the above
11. Interruption attacks are also called _____ attacks.
- masquerade
 - alteration
 - denial of service
 - replay attacks
12. DOS attacks are caused by _____.
- authentication
 - alteration
 - fabrication
 - replay attacks
13. Virus is a computer _____.
- file
 - program
 - database
 - network
14. A worm _____ modify a program.
- does not
 - does
 - may or may not
 - may
15. A _____ replicates itself by creating its own copies, in order to bring the network to a halt.
- virus
 - worm
 - Trojan horse
 - bomb

■ Exercises

- Find out more examples of security attacks reported in the last few years.
- What are the key principles of security?
- Why is confidentiality an important principle of security? Think about ways of achieving the same. (*Hint:* Think about the ways in which children use a secret language).
- Discuss the reasons behind the significance of authentication. Find out the simple mechanisms of authentication. (*Hint:* What information do you provide when you use a free email service such as Yahoo or Hotmail?)
- In real life, how is message integrity ensured? (*Hint:* On what basis is a check honored or dishonored?)
- What is repudiation? How can it be prevented in real life? (*Hint:* Think what happens if you issue a check, and after the bank debits your account with the amount therein, you complain to the bank that you never issued that check).
- What is access control? How different is it from availability?
- Why are some attacks called passive? Why are other attacks called active?
- Discuss any one passive attack.
- What is ‘masquerade’? Which principle of security is breached because of that?
- What are ‘replay attacks’? Give an example of replay attacks.
- What is ‘denial of service’ attack?
- What is a ‘worm’? What is the significant difference between a ‘worm’ and a ‘virus’?
- Discuss the concepts of ‘phishing’ and ‘pharming’.
- Would message integrity on its own ensure that the contents of a message are not changed during transit? Does something more needs to be done?

■ Design/Programming Exercises

1. Write a C program that contains a string (char pointer) with a value ‘Hello World’. The program should XOR each character in this string with 0 and display the result. Repeat the exercise by an XOR operation with 1.
2. Write a C program that contains a string (char pointer) with a value ‘Hello World’. The program should AND, OR and XOR each character in this string with 127 and display the result. Why are these results different?
3. Study ‘phishing’ in more detail. Find out which popular bank sites have been phished and how.
4. Think about offering phishing-prevention techniques. Which ones of them would be most effective, and why?
5. Why is it easier to fall prey to ‘pharming’ than ‘phishing’? Explain in technical terms.
6. Often, it is said that a technology called SSL can prevent ‘phishing’ and ‘pharming’. Is it always true? Why?
7. Write a small viruslike program in plain English language that accepts a file name and changes every character in the file to an asterisk.
8. How is DNS secured? Are standard protocols available?
9. Study what is meant by Nigerian Fraud and how it can be prevented.
10. What is the online lottery scam? How does it work?
11. What tricks do attackers use to hack into online banking accounts?
12. Study what is meant by social engineering and how it works.
13. Who is Kevin Mitnick? Why is he well known?
14. What threats do attacks on social networking sites pose? How can those be prevented?
15. Which tools are popularly used by attackers to attack Web sites?



CRYPTOGRAPHY TECHNIQUES

■ 2.1 INTRODUCTION ■

This chapter introduces the basic concepts in **cryptography**. Although this word sounds fearful, we shall realize that it is very simple to understand. In fact, most terms in computer security have a very straightforward meaning. Many terms, for no reason, sound complicated. Our aim will be to demystify all such terms in relation to cryptography in this chapter. After we are through with this chapter, we shall be equipped to understand computer-based security solutions and issues that follow in later chapters.

Cryptography is the art of achieving security by encoding messages to make them non-readable.

Figure 2.1 shows the conceptual view of cryptography.

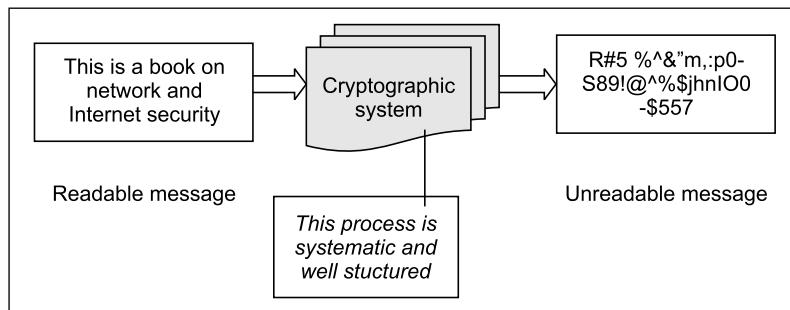


Fig. 2.1 Cryptographic system

Some more terms need to be introduced in this context.

Cryptanalysis is the technique of decoding messages from a non-readable format back to a readable format without knowing how they were initially converted from readable format to non-readable format.

In other words, it is like *breaking a code*. This concept is shown in Fig. 2.2.

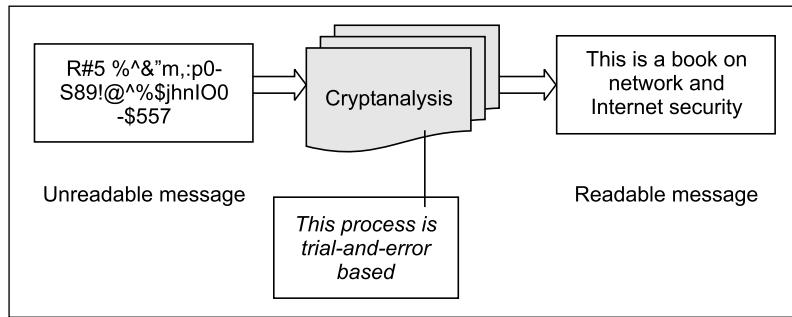


Fig. 2.2 Cryptanalysis

Cryptology is a combination of cryptography and cryptanalysis.

This concept is shown in Fig. 2.3.

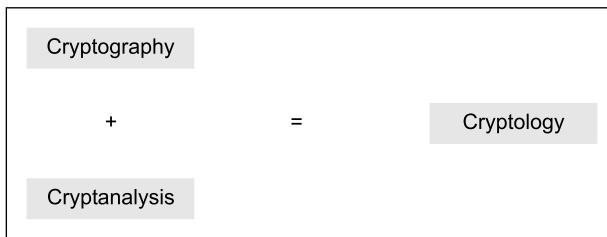


Fig. 2.3 Cryptography + Cryptanalysis = Cryptology

In the early days, cryptography was performed by using manual techniques. The basic framework of performing cryptography has remained more or less the same, of course, with a lot of improvements in the actual implementation. More importantly, computers now perform these cryptographic functions/algorithms, thus making the process a lot faster and secure. This chapter, however, discusses the basic methods of achieving cryptography without referring to computers.

The basic concepts in cryptography are introduced first. We then proceed to discuss how we can make messages illegible, and thus secure. This can be done in many ways. We discuss all these approaches in this chapter. Modern computer-based cryptography solutions have actually evolved based on these premises. This chapter touches upon all these cryptography algorithms. We also discuss the relative advantages and disadvantages of the various algorithms, as and when applicable.

Some cryptography algorithms are very trivial to understand, replicate, and therefore, crack. Some other cryptography algorithms are highly complicated, and therefore difficult to crack. The rest are somewhere in the middle. A detailed discussion of these is highly essential in cementing our concepts that we shall keep referring to when we actually discuss computer-based cryptography solutions in later chapters.

■ 2.2 PLAIN TEXT AND CIPHER TEXT ■

Any communication in the language that you and I speak—that is the human language—takes the form of **plain text** or **clear text**. That is, a message in plain text can be understood by anybody knowing the language as long as the message is not codified in any manner. For instance, when we speak with

our family members, friends or colleagues, we use plain text because we do not want to hide anything from them. Suppose I say “Hi Anita”, it is plain text because both Anita and I know its meaning and intention. More significantly, anybody in the same room would also get to hear these words, and would know that I am greeting Anita.

Notably, we also use plain text during electronic conversations. For instance, when we send an email to someone, we compose the email message using English, or these days another language. For instance, I can compose the email message as shown in Fig. 2.4.

Hi Amit
Hope you are doing fine. How about meeting at the train station this Friday at 5 p.m.? Please let me know if it is OK with you.
Regards,
Atul

Fig. 2.4 Example of a plain-text message

Now, not only Amit, but also any other person who reads this email would know what I have written. As before, this is simply because I am not using any codified language here. I have composed my email message using plain English. This is another example of plain text, albeit in written form.

Clear text, or plain text, signifies a message that can be understood by the sender, the recipient, and also by anyone else who gets access to that message.

In normal life, we do not bother much about the fact that someone could be overhearing us. In most cases, that makes little difference to us because the person overhearing us can do little damage by using the overheard information. After all, we do not reveal many secrets in our day-to-day lives!

However, there are situations where we are concerned about the secrecy of our conversations. For instance, suppose I wish to know my bank account’s balance and hence I call up my phone banker from my office. The phone banker would generally ask a secret question (e.g. What is your mother’s maiden name?) whose answer only I know. This is to ascertain that someone else is not posing as me. Now, when I give the answer to the secret question (e.g. Leela), I would generally speak in low voice, or better yet, initially call up from a phone that is isolated. This ensures that only the intended recipient (the phone banker) gets to know the correct answer.

On the same lines, suppose that my email to my friend Amit shown earlier is confidential for some reason. Therefore, I do not want anyone else to understand what I have written, even if he/she is able to access the email by using some means, before it reaches Amit. How do I ensure this? This is exactly the problem that small children face. Many times, they want to communicate in such a manner that their little secrets are hidden from the elderly. What do they do in order to achieve this? Usually, the simplest trick that they use is a code language. For instance, they replace each alphabet in their conversation with another character. As an example, they replace each alphabet with the alphabet that is actually three alphabets down the order. So, each A will be replaced by D, B will be replaced by E, C will be replaced by F, and so on. To complete the cycle, each W will be replaced by Z, each X will be replaced by A, each Y will be replaced by B and each Z will be replaced by C. We can summarize this scheme as shown in Fig. 2.5. The first row shows the original alphabets, and the second row shows what each original alphabet will be replaced with.

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
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

Fig. 2.5 A scheme for codifying messages (replacing each alphabet with an alphabet three places down the line)

Thus, using the scheme of replacing each alphabet with the one that is three places down the line, a message *I love you* shall become *L ORYH BRX* as shown in Fig. 2.6.

I		L	O	V	E		Y	O	U
L	O	R	Y	H		B	R	X	

Fig. 2.6 Codification using the alphabet-replacement scheme

Of course, there can be many variants of such a scheme. It is not necessary to replace each alphabet with the one that is three places down the order. It can be the one that is four, five or more places down the order. The point is, however, that each alphabet in the original message can be replaced by another to hide the original contents of the message. The codified message is called **cipher text**. Cipher means a code or a secret message.

When a plain-text message is codified using any suitable scheme, the resulting message is called cipher text.

Based on these concepts, let us put these terms into a diagrammatic representation, as shown in Fig. 2.7.

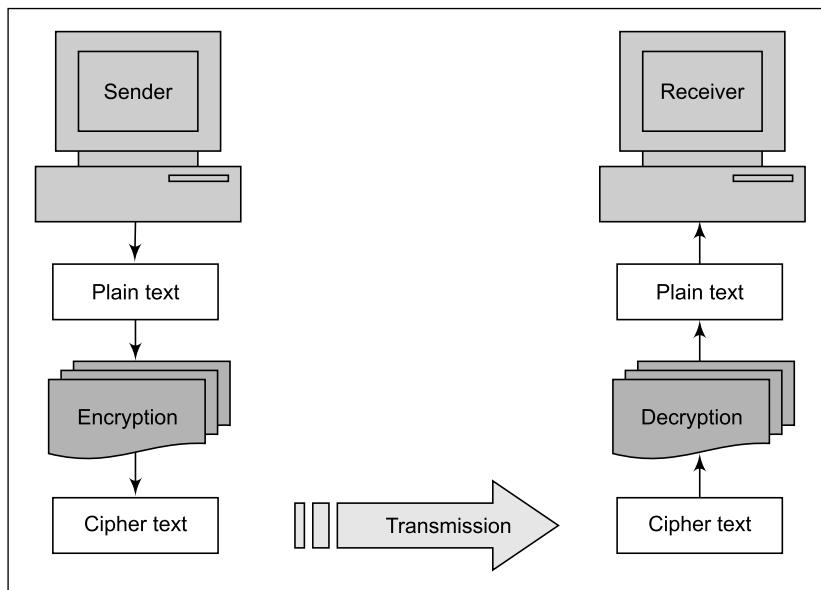


Fig. 2.7 Elements of a cryptographic operation

Let us now write our original email message and the resulting cipher text by using the alphabet-replacing scheme, as shown in Fig. 2.8. This will clarify the idea further.

As shown in Fig. 2.9, there are two primary ways in which a plain-text message can be codified to obtain the corresponding cipher text: **substitution** and **transposition**.

<p>Hi Amit,</p> <p>Hope you are doing fine. How about meeting at the train station this Friday at 5 p.m.? Please let me know if it is OK with you.</p> <p>Regards.</p> <p>Atul</p>	<p>KI Dplw,</p> <p>Krsh brx duh grlqj ilqh. Krz derxw phhwlijqj dw wkh wudlq vwdwlrq wklv lulgdb dw 5 sp? Sohdvh ohw ph nqrz li lw lv rn zlwk brx.</p> <p>Uhjdugv.</p> <p>Dwxo</p>
Plain-text message	Corresponding cipher-text message

Fig. 2.8 Example of a plain-text message being transformed into cipher text

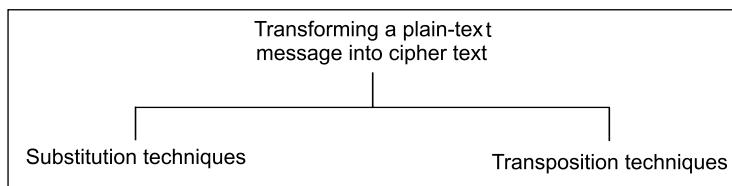


Fig. 2.9 Techniques for transforming plain text to cipher text

Let us discuss these two approaches now. Note that when the two approaches are used together, we call the technique **product cipher**.

■ 2.3 SUBSTITUTION TECHNIQUES ■

2.3.1 Caesar Cipher

The scheme explained earlier (of replacing an alphabet with the one three places down the order) was first proposed by Julius Caesar, and is termed **Caesar cipher**. It was the first example of substitution cipher. In the substitution-cipher technique, the characters of a plain-text message are replaced by other characters, numbers or symbols. The Caesar cipher is a special case of substitution technique wherein each alphabet in a message is replaced by an alphabet three places down the line. For instance, using the Caesar cipher, the plain-text ATUL will become cipher-text DWXO.

In the substitution-cipher technique, the characters of a plain-text message are replaced by other characters, numbers or symbols.

Clearly, the Caesar cipher is a very weak scheme of hiding plain-text messages. All that is required to break the Caesar cipher is to do the reverse of the Caesar cipher process—i.e. replace each alphabet in a cipher-text message produced by Caesar cipher with the alphabet that is three places up the line. Thus, to work backwards, take a cipher text produced by Caesar cipher, and replace each A with X, B with Y, C with Z, D with A, E with B and so on. The simple algorithm required to break the Caesar cipher can be summarized as shown in Fig. 2.10.

1. Read each alphabet in the cipher-text message, and search for it in the second row of the replacement table (i.e. the second row of the table).
2. When a match is found, replace that alphabet in the cipher-text message with the corresponding alphabet in the same column but the first row of the table (e.g. if the alphabet in cipher text is J, replace it with G).
3. Repeat the process for all alphabets in the cipher-text message.

Fig. 2.10 Algorithm to break Caesar cipher

The process shown above will reveal the original plain text. Thus, given a cipher-text message *L ORYH BRX*, it is easy to work backwards and obtain the plain text *I LOVE YOU* as shown in Fig. 2.11.

Cipher text	L		O	R	Y	H		B	R	X
Plain text	I		L	O	V	E		Y	O	U

Fig. 2.11 Example of breaking Caesar cipher

2.3.2 Modified Version of Caesar Cipher

The Caesar cipher is good in theory, but not so good in practice. Let us now try and complicate the Caesar cipher to make an attacker's task difficult. How can we generalize Caesar cipher a bit more? Let us assume that the cipher-text alphabets corresponding to the original plain-text alphabets may not necessarily be three places down the order, but instead, can be *any* places down the order. This can complicate matters a bit.

Thus, we are now saying that an alphabet A in plain text would not necessarily be replaced by D. It can be replaced by any valid alphabet, i.e. by E or by F or by G, and so on. Once the replacement scheme is decided, it would be constant and will be used for all other alphabets in that message. As we know, the English language contains 26 alphabets. Thus, an alphabet A can be replaced by any *other* alphabet in the English alphabet set, (i.e. B through Z). Of course, it does not make sense to replace an alphabet by itself (i.e. replacing A with A). Thus, for each alphabet, we have 25 possibilities of replacement. Hence, to break a message in the modified version of Caesar cipher, our earlier algorithm would not work. Let us write a new algorithm to break this version of the Caesar cipher, as shown in Fig. 2.12.

1. Let k be a number equal to 1.
2. Read the complete cipher text message.
3. Replace each alphabet in the cipher text message with an alphabet that is k positions down the order.
4. Increment k by 1.
5. If k is less than 26, then go to Step 2. Otherwise, stop the process.
6. The original text message corresponding to the cipher-text message is one of the 25 possibilities produced by the above steps.

Fig. 2.12 Algorithm to break the modified Caesar cipher

Let us take a cipher-text message produced by the modified Caesar cipher, and try breaking it to obtain the original plain-text message by applying the algorithm shown earlier. Since each alphabet in the plain-text can be potentially replaced by any other of the 25 alphabets, we have 25 possible plain-text messages to choose from. Thus, the output produced by the above algorithm to break a cipher-text message *KWUM PMZN* is shown in Fig. 2.13.

We can see that the cipher text shown in the first row of the figure needs 25 different attempts to break in, as depicted by the algorithm shown earlier. As it turns out, the 18th attempt reveals the correct

Cipher text	K	W	U	M		P	M	Z	M
Attempt Number (Value of k)									
1	L	X	V	N		Q	N	A	N
2	M	Y	W	O		R	O	B	O
3	N	Z	X	P		S	P	C	P
4	O	A	Y	Q		T	Q	D	Q
5	P	B	Z	R		U	R	E	R
6	Q	C	A	S		V	S	F	S
7	R	D	B	T		W	T	G	T
8	S	E	C	U		X	U	H	U
9	T	F	D	V		Y	V	I	V
10	U	G	E	W		Z	W	J	W
11	V	H	F	X		A	X	K	X
12	W	I	G	Y		B	Y	L	Y
13	X	J	H	Z		C	Z	M	Z
14	Y	K	I	A		D	A	N	A
15	Z	L	J	B		E	B	O	B
16	A	M	K	C		F	C	P	C
17	B	N	L	D		G	D	Q	D
18	C	O	M	E		H	E	R	E
19	D	P	N	F		I	F	S	F
20	E	Q	O	G		J	G	T	G
21	F	R	P	H		K	H	U	H
22	G	S	Q	I		L	I	V	I
23	H	T	R	J		M	J	W	J
24	I	U	S	K		N	K	X	K
25	J	V	T	L		O	L	Y	L

Fig. 2.13 Attempts to break modified Caesar-cipher text using multiple possibilities

plain text corresponding to the cipher text. Therefore, we can actually stop at this juncture. For the sake of completeness, however, we have shown all the 25 steps, which is, of course, the worst possible case.

A mechanism of encoding messages so that they can be sent securely is called cryptography. Let us take this opportunity to introduce a few terms used in cryptography. An attack on a cipher-text message, wherein the attacker attempts to use all possible permutations and combinations, is called a **brute-force attack**. The process of trying to break any cipher-text message to obtain the original plain-text message itself is called **cryptanalysis**, and the person attempting a cryptanalysis is called a **cryptanalyst**.

A cryptanalyst is a person who attempts to break a cipher-text message to obtain the original plain-text message. The process itself is called cryptanalysis.

As we have noticed, even the modified version of the Caesar cipher is not very secure. After all, the cryptanalyst needs to be aware of only the following points to break a cipher-text message using the brute-force attack, in this scheme:

1. Substitution technique was used to derive the cipher text from the original plain text.
2. There are only 25 possibilities to try out.
3. The language of the plain text was English.

A cryptanalyst attempting a brute-force attack tries all possibilities to derive the original plain-text message from a given cipher-text message.

Anyone armed with this knowledge can easily break a cipher text produced by the modified version of Caesar cipher. How can we make the modified Caesar cipher even tougher to crack?

2.3.3 Mono-alphabetic Cipher

The major weakness of the Caesar cipher is its predictability. Once we decide to replace an alphabet in a plain-text message with an alphabet that is k positions up or down the order, we replace all other alphabets in the plain-text message with the same technique. Thus, the cryptanalyst has to try out a maximum of 25 possible attacks, and he/she is assured of success.

Now imagine that rather than using a uniform scheme for all the alphabets in a given plain-text message, we decide to use random substitution. This means that in a given plain-text message, each A can be replaced by any other alphabet (B through Z), each B can also be replaced by any other random alphabet (A or C through Z), and so on. The crucial difference being, there is no relation between the replacement of B and replacement of A. That is, if we have decided to replace each A with D, we need not necessarily replace each B with E—we can replace each B with any other character!

To put it mathematically, we can now have any permutation or combination of the 26 alphabets, which means $(26 \times 25 \times 24 \times 23 \times \dots \times 2)$ or 4×10^{26} possibilities! This is extremely hard to crack. It might actually take years to try out these many combinations even with the most modern computers.

Mono-alphabetic ciphers pose a difficult problem for a cryptanalyst because it can be very difficult to crack, thanks to the high number of possible permutations and combinations.

There is only one hitch. If the cipher text created with this technique is short, the cryptanalyst can try different attacks based on his/her knowledge of the English language. As we know, some alphabets in the English language occur more frequently than others. Language analysts have found that given a single alphabet in cipher text, the probability that it is a P is 13.33%—the highest. After P comes Z, which is likely to occur 11.67%. The probability that the alphabet is C, K, L, N or R is almost 0—the lowest.

A cryptanalyst looks for patterns of alphabets in a cipher text, substitutes the various available alphabets in place of cipher-text alphabets, and then tries his/her attacks.

Apart from single-alphabet replacements, the cryptanalyst also looks for repeated patterns of words to try the attacks. For example, the cryptanalyst might look for two-alphabet cipher text patterns since the word *to* occurs very frequently in English. If the cryptanalyst finds that two alphabet combinations are found frequently in a cipher-text message, he/she might try and replace all of them with *to*, and then try and deduce the remaining alphabets/words. Next, the cryptanalyst might try to find repeating three-alphabet patterns and try and replace them with the word *the, and*, and so on.

2.3.4 Homophonic Substitution Cipher

The **homophonic substitution cipher** is very similar to mono-alphabetic cipher. Like a plain substitution cipher technique, we replace one alphabet with another in this scheme. However, the difference between the two techniques is that whereas the replacement alphabet set in case of the simple substitution techniques is fixed (e.g. replace A with D, B with E, etc.), in the case of homophonic substitution cipher, one plain-text alphabet can map to more than one cipher-text alphabet. For instance, A can be replaced by D, H, P, R; B can be replaced by E, I, Q, S, etc.

Homophonic substitution cipher also involves substitution of one plain-text character with a cipher-text character at a time, however the cipher-text character can be any one of the chosen set.

2.3.5 Polygram Substitution Cipher

In the **Polygram substitution cipher** technique, rather than replacing one plain-text alphabet with one cipher text alphabet at a time, a block of alphabets is replaced with another block. For instance, HELLO could be replaced by YUQQW, but HELL could be replaced by a totally different cipher text block TEUI, as shown in Fig. 2.14. This is true in spite of the first four characters of the two blocks of text (HELL) being the same. This shows that in the polygram substitution cipher, the replacement of plain text happens block by block, rather than character by character.

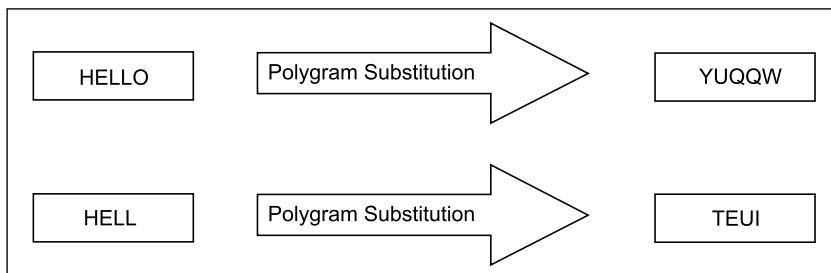


Fig. 2.14 Polygram substitution

Polygram substitution cipher technique replaces one block of plain text with another block of cipher text—it does not work on a character-by-character basis.

2.3.6 Polyalphabetic Substitution Cipher

Leon Battista invented the **Polyalphabetic substitution cipher** in 1568. This cipher has been broken many times, and yet it has been used extensively. The **Vigenère cipher** and the **Beaufort cipher** are examples of polyalphabetic substitution cipher.

This cipher uses multiple one-character keys. Each of the keys encrypts one plain-text character. The first key encrypts the first plain-text character; the second key encrypts the second plain-text character, and so on. After all the keys are used, they are recycled. Thus, if we have 30 one-letter keys, every 30th character in the plain text would be replaced with the same key. This number (in this case, 30) is called the **period** of the cipher.

The main features of polyalphabetic substitution cipher are the following:

- (a) It uses a set of related monoalphabetic substitution rules.
- (b) It uses a key that determines which rule is used for which transformation.

For example, let us discuss the Vigenère cipher, which is an example of this cipher. In this algorithm, 26 Caesar ciphers make up the mono-alphabetic substitution rules. There is a shifting mechanism, from a count of 0 to 25. For each plain-text letter, we have a corresponding substitution, which we call the *key letter*. For instance, the key value is *e* for a letter with shift as 3.

To understand this technique, we need to take a look at a table, which is formally known as Vigenère tableau. This table is shown in Fig. 2.15.

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

Fig. 2.15 Vigenère tableau

The logic for encryption is quite simple. For key letter *p* and plain-text letter *q*, the corresponding cipher-text letter is at the intersection of row titled *p* and column titled *q*. For this very particular case, the cipher text, therefore, would be *F*, based on the above table.

By now, it should be clear that for encrypting a plain-text message, we need a key whose length is equal to that of the plain-text message. Usually, a key that repeats itself is used.

2.3.7 Playfair Cipher

The **Playfair cipher**, also called **Playfair square**, is a cryptographic technique used for manual encryption of data. This scheme was invented by Charles Wheatstone in 1854. However, eventually the scheme came to be known by the name of Lord Playfair, who was Wheatstone's friend. Playfair made this scheme popular, and hence his name was used.

The Playfair cipher was used by the British army in World War I and by the Australians in World War II. This was possible because the Playfair cipher is quite fast to use and does not demand any special equipment to be used. It was used to protect important but not very critical information, so that by the time the cryptanalysts could break it, the value of the information was nullified anyway! In today's world, of course, Playfair cipher would be deemed as an outdated cryptographic algorithm, and rightly so. Playfair cipher now has only academic purpose, except in its usage in some crosswords that appear in several newspapers.

The Playfair encryption scheme uses two main processes, as shown in Fig. 2.16.

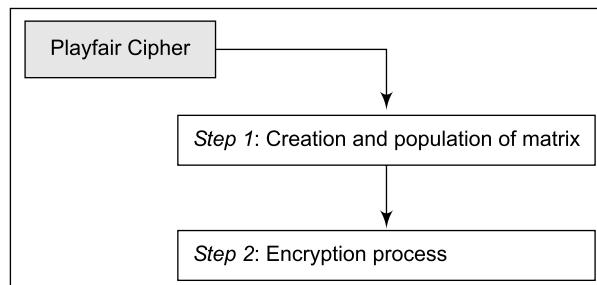


Fig. 2.16 Playfair Cipher steps

Step 1: Creation and Population of Matrix

The Playfair cipher makes use of a 5×5 matrix (table), which is used to store a *keyword* or *phrase* that becomes the *key* for encryption and decryption. The way this is entered into the 5×5 matrix is based on some simple rules, as shown in Fig. 2.17.

1. Enter the keyword in the matrix row-wise: left-to-right, and then top-to-bottom.
2. Drop duplicate letters.
3. Fill the remaining spaces in the matrix with the rest of the English alphabets (A-Z) that were not a part of our keyword. While doing so, combine I and J in the same cell of the table. In other words, if I or J is a part of the keyword, disregard both I and J while filling the remaining slots.

Fig. 2.17 Matrix creation and population

For example, suppose that our keyword is PLAYFAIR EXAMPLE. Then, the 5×5 matrix containing our keyword will look as shown in Fig. 2.18.

Explanation of this row-wise is as follows.

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

Fig. 2.18 Keyword matrix for our example

Row 1 The first row of the matrix is as shown in Fig. 2.19.

As we can see, this is simply the first five letters of our keyword (*PLAYF*). None of the alphabets is a duplicate so far. Therefore, we write these one after the other in the same row, as per our rule #1 defined earlier.

Row 2 The second row of the matrix is as shown in Fig. 2.20.

The first four cells are just the continuation of keyword (*I R E X*) from where we had left in the previous row. This is as per our rule #1 defined earlier. However, after this, we have a repetition of alphabet A, as shown in Fig. 2.21.

P	L	A	Y	F
---	---	---	---	---

Fig. 2.19 Keyword matrix (*first* row of mentioned example)

I	R	E	X	M
---	---	---	---	---

Fig. 2.20 Keyword matrix (*second* row of mentioned example)

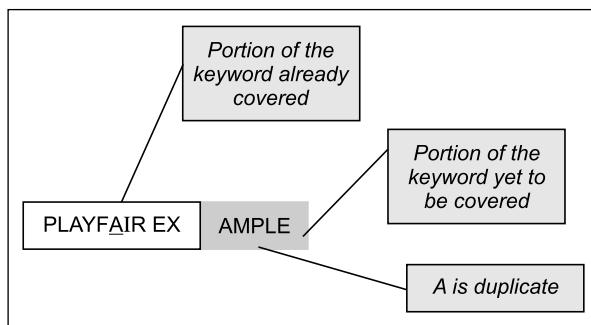


Fig. 2.21 Situation when a duplicate alphabet (A) is discovered

Therefore, we disregard the repeating A as per our rule #2 defined earlier, and choose the next alphabet, which is M. This gets loaded into the fifth cell of this row.

Row 3 The third row of the matrix is as shown in Fig. 2.22.

Let us now review what part of our keyword was covered in the first two rows.

B	C	D	G	H
---	---	---	---	---

Fig. 2.22 Keyword matrix (*third* row of mentioned example)

Our keyword was **PLAYFAIR EXAMPLE**

Our first two rows were **P L A Y F** and **I R E X M**.

Let us now change the font of the alphabets in our keyword which are covered anywhere in these two rows, in italics. Let us also indicate duplicates with an underscore. This would make our keyword look like this:

PLAYFAIR EXAMPE

As we can see, all of our keyword alphabets are covered now (because every alphabet is (a) either in italics, indicating that it is a part of our matrix, or is (b) underlined, indicating that it is a duplicate).

Therefore, there is nothing left to populate in our matrix from the third row onwards. Hence, we would now consider our rule #3 defined earlier. This rule suggests that we should fill the remaining part of the matrix with alphabets from A-Z that are yet unused. Based on this criterion, we see that B, C, D, G, and H will fit into the third row of our matrix.

Row 4 The fourth row of the matrix is as shown in Fig. 2.23.

Here, we simply apply our rule #3 straightforwardly to get the text of K N O Q S.

K	N	O	Q	S
---	---	---	---	---

Fig. 2.23 Keyword matrix (*fourth* row of mentioned example)

Row 5 The fifth row of the matrix is as shown in Fig. 2.24.

T	U	V	W	Z
---	---	---	---	---

Here, we simply apply our rule #3 straightaway to get the text of T U V W Z.

Fig. 2.24 Keyword matrix (fifth row of mentioned example)

Step 2: Encryption Process

The encryption process consists of five steps, as outlined in Fig. 2.25.

1. Before executing these steps, the plain-text message that we want to encrypt needs to be broken down into groups of two alphabets. For example, if our message is MY NAME IS ATUL, it becomes MY NAME IS AT UL. The encryption process works on this *broken-down* message.
2. If both alphabets are the same (or only one is left), add an X after the first alphabet. Encrypt the new pair and continue.
3. If both the alphabets in the pair appear in the same row of our matrix, replace them with alphabets to their immediate right respectively. If the original pair is on the right side of the row, then wrapping around to the left side of the row happens.
4. If both the alphabets in the pair appear in the same column of our matrix, replace them with alphabets immediately below them respectively. If the original pair is on the bottom side of the row, then wrapping around to the top side of the row happens.
5. If the alphabets are not in the same row or column, replace them with the alphabets in the same row respectively, but at the other pair of corners of the rectangle defined by the original pair. The order is quite significant here. The first encrypted alphabet of the pair is the one that is present on the same row as the first plain-text alphabet.

Fig. 2.25 Encryption process in Playfair cipher

Decryption process works in the opposite direction. We also need to remove the extra X alphabets that we had added in step #1 above, if any.

Let us now take a concrete example to illustrate the process of encrypting some text using a keyword. Our keyword is PLAYFAIR EXAMPLE and the original text is MY NAME IS ATUL. We know that the matrix for our keyword is as shown in Fig. 2.26. We have discussed this in detail earlier and need not repeat it.

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

Fig. 2.26 Keyword matrix for our example

Here is an explanation of the encryption process.

1. First we break the original text into pairs of two alphabets each. This means that our original text would now look like this:

MY NA ME IS AT UL

2. Now, we apply our Playfair cipher algorithm to this text. The first pair of alphabets is MY. Looking at the matrix, we see that the alphabets M and Y do not occur in the same row or column. Therefore, we need to apply Step #5 of our Playfair cipher encryption process. This means that we need to replace this text with the text diagonally opposite to it. In this case, this text is XF, which is our first cipher text block. This is shown in Fig. 2.27.

3. Our next text block to be encrypted is NA. Again, Step #5 will apply as depicted in Fig. 2.28.

As we can see, our second block of cipher text is OL.

4. We will now take a look at the third block of plain text, which is ME. This is shown in Fig. 2.29. We can see that the alphabets E and M making up this block are in the same (second) row. Therefore, based on our logic of Step #3, the cipher-text block would be IX.

5. We will now take a look at the fourth block of plain text, which is IS. This is shown in Fig. 2.30. We can see that we need to apply the logic of Step #5 to get the diagonal alphabets. Based on this, the cipher-text block would be MK.

6. We will now take a look at the fifth block of plain text, which is AT. This is shown in Fig. 2.31. We can see that we need to apply the logic of Step #5 to get the diagonal alphabets. Based on this, the cipher-text block would be PV.

7. We will now take a look at the sixth and last block of plain text, which is UL. This is shown in Fig. 2.32. We can see that the two alphabets U and L are in the same column. Therefore, we need to apply the logic of Step #4 to get the alphabets LR.

Thus, our plain-text blocks MY NA ME IS AT UL becomes XF OL IX MK PV LR.

We will not illustrate the decryption process. It is pretty straightforward. It would work in exactly the opposite steps. We leave it to the reader to verify this.

Just to get ourselves more familiar with the process further, we illustrate another example. For brevity, we have taken out the description of the various steps.

2.3.8 Hill Cipher

The **Hill cipher** works on multiple letters at the same time. Hence, it is a type of polygraphic substitution cipher. Lester Hill invented this in 1929. The Hill cipher has its roots in the matrix theory of mathematics. More specifically, we need to know how to compute the inverse of a matrix. This mathematics is explained in *Appendix A*. Interested readers are encouraged to refer to the mathematical theory there.

The way the Hill cipher works is as shown in Fig. 2.34.

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

Fig. 2.27 Alphabet Pair 1

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

Fig. 2.28 Alphabet Pair 2

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

Fig. 2.29 Alphabet Pair 3

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

Fig. 2.30 Alphabet Pair 4

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

Fig. 2.31 Alphabet Pair 5

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

Fig. 2.32 Alphabet Pair 6

Our keyword here is *Harsh*. The plain text to be encrypted is *My name is Jui Kahate. I am Harshu's sister.*

Based on this information, the keyword matrix is as follows:

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

Our plain-text message broken down into pairs of alphabets is:

MY NA ME IS IU IK AH AT EI AM HA RS HU 'S XS IS TE RX

Using Playfair cipher based on the above matrix, the resulting cipher text would be:

TS KB LF MH NO KL RA SP CL SK AR SB BO AB YR MH QF ER

The reader is encouraged to work out this example step by step.

Fig. 2.33 Practice example for Playfair cipher

1. Treat every letter in the plain-text message as a number, so that A = 0, B = 1, ..., Z = 25.
2. The plain-text message is organized as a matrix of numbers, based on the above conversion. For example, if our plain text is CAT. Based on the above step, we know that C = 2, A = 0, and T = 19. Therefore, our plain-text matrix would look as follows:

$$\begin{pmatrix} 2 \\ 0 \\ 19 \end{pmatrix}$$

3. Now, our plain-text matrix is multiplied by a matrix of randomly chosen keys. The key matrix consists of size $n \times n$, where n is the number of rows in our plain-text matrix. For example, we take the following key matrix:

$$\begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}$$

4. Now multiply the two matrices, as shown below:

$$\begin{pmatrix} 2 \\ 0 \\ 19 \end{pmatrix} \times \begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix} = \begin{pmatrix} 31 \\ 216 \\ 325 \end{pmatrix}$$

5. Now compute a mod 26 value of the above matrix. That is, take the remainder after dividing the above matrix values by 26. That is

$$\begin{pmatrix} 31 \\ 216 \\ 325 \end{pmatrix} \bmod 26 = \begin{pmatrix} 5 \\ 8 \\ 13 \end{pmatrix}$$

6. (This is because: $31 / 26 = 1$ with a remainder of 5: which goes in the above matrix, and so on).

7. Now, translating the numbers to alphabets, 5 = F, 8 = I, and 13 = N. Therefore, our cipher text is FIN.

8. For decryption, take the cipher-text matrix and multiply it by the inverse of our original key matrix (explained later). The inverse of our original key matrix is

$$\begin{pmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{pmatrix}$$

Fig. 2.34(a) Hill cipher example—Part 1/2

1. For decryption, take the cipher-text matrix and multiply it by the inverse of our original key matrix. The inverse of our original key matrix is

$$\begin{pmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{pmatrix} \times \begin{pmatrix} 5 \\ 8 \\ 13 \end{pmatrix} = \begin{pmatrix} 210 \\ 442 \\ 305 \end{pmatrix}$$

2. Now we need to take modulo 26 of this matrix, as follows.

$$\begin{pmatrix} 210 \\ 442 \\ 305 \end{pmatrix} \bmod 26 = \begin{pmatrix} 2 \\ 0 \\ 19 \end{pmatrix}$$

3. Thus, our plain-text matrix contains 2, 0, 19; which corresponds to 2 = C, 0 = A, and 19 = T. This gives us the original plain text back successfully.

Fig. 2.34(b) Hill cipher example—Part 2/2

The Hill cipher is vulnerable to the *known-plain-text attack*, which we are going to discuss later. This is because it is linear (i.e. it is possible to compute smaller factors of the matrices, work on them individually, and then join them back as and when they are ready).

■ 2.4 TRANSPOSITION TECHNIQUES ■

As we discussed, substitution techniques focus on substituting a plain-text alphabet with a cipher-text alphabet. Transposition techniques differ from substitution techniques in the way that they do not simply replace one alphabet with another, but they also perform some permutation over the plain text

2.4.1 Rail-Fence Technique

The **rail-fence technique** is an example of transposition. It uses a simple algorithm as shown in Fig. 2.35.

1. Write down the plain-text message as a sequence of diagonals.
2. Read the plain text written in Step 1 as a sequence of rows.

Fig. 2.35 Rail-fence technique

Let us illustrate the rail-fence technique with a simple example. Suppose that we have a plain-text message :*Come home tomorrow*. How would we transform that into a cipher-text message using the rail-fence technique? This is shown in Fig. 2.36.

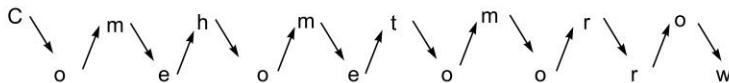
As the figure shows, the plain-text message ‘*Come home tomorrow*’ transforms into ‘*Cmhmtm-rooeoeoowr*’ with the help of rail-fence technique.

Rail-fence technique involves writing plain text as a sequence of diagonals and then reading it row by row to produce cipher text.

It should be quite clear that the rail-fence technique is quite simple for a cryptanalyst to break into. It has very little sophistication built in.

Original plain-text message: **Come home tomorrow**

- After we arrange the plain-text message as a sequence of diagonals, it would look as follows (write the first character on the first line i.e. C, then second character on the second line, i.e. o, then the third character on the first line, i.e. m, then the fourth character on the second line, i.e. e, and so on). This creates a zigzag sequence, as shown below.



- Now read the text row by row, and write it sequentially. Thus, we have:
Cmhmtmrroooeoeeoorw as the cipher text.

Fig. 2.36 Example of rail-fence technique

2.4.2 Simple Columnar Transposition Technique

1. Basic Technique

Variations of the basic transposition technique such as rail-fence technique exist. Such a scheme is shown in Fig. 2.37, which we shall call **simple columnar transposition technique**.

- Write the plain-text message row by row in a rectangle of a pre-defined size.
- Read the message column by column. However, it need not be in the order of columns 1, 2, 3, etc. It can be any random order such as 2, 3, 1, etc.
- The message thus obtained is the cipher-text message.

Fig. 2.37 Simple columnar transposition technique

Let us examine the simple columnar transposition technique with an example. Consider the same plain-text message '*Come home tomorrow*'. Let us understand how it can be transformed into cipher text using this technique. This is illustrated in Fig. 2.38.

Original plain-text message: **Come home tomorrow**

- Let us consider a rectangle with six columns. Therefore, when we write the message in the rectangle row by row (suppressing spaces), it would look as follows:

Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
C	o	m	e	h	o
m	e	t	o	m	o
r	r	o	w		

- Now, let us decide the order of columns as some random order, say 4, 6, 1, 2, 5 and 3. Then read the text in the order of these columns.
- The cipher text thus obtained would be **eowoocmroerhmmto**.

Fig. 2.38 Example of simple columnar transposition technique

The simple columnar transposition technique simply arranges the plain text as a sequence of rows of a rectangle that are read in columns randomly.

Like the rail-fence technique, the simple columnar transposition Technique is also quite simple to break into. It is a matter of trying out a few permutations and combinations of column orders to get hold of

the original plain text. To make matters complex for a cryptanalyst, we can modify the simple columnar transposition technique to add another twist: perform more than one round of transposition using the same technique.

2. Simple Columnar Transposition Technique with Multiple Rounds

To improve the basic simple columnar transposition technique, we can introduce more complexity. The idea is to use the same basic procedure as used by the simple columnar transposition technique, but to do it more than once. That adds considerably more complexity for the cryptanalyst.

The basic algorithm used in this technique is shown in Fig. 2.39.

1. Write the plain text message row by row in a rectangle of a pre-defined size.
2. Read the message column by column. However, it need not be in the order of columns 1, 2, 3 etc. It can be any random order such as 2, 3, 1, etc.
3. The message thus obtained is the cipher text message of round 1.
4. Repeat steps 1 to 3 as many times as desired.

Fig. 2.39 Simple columnar transposition technique with multiple rounds

As we can see, the only addition in this technique to the basic simple columnar transposition technique is step 4, which results in the execution of the basic algorithm on more than one occasion. Although this sounds trivial, in reality, it makes the cipher text far more complex as compared to the basic simple columnar transposition technique. Let us extend our earlier example to now have multiple rounds of transposition, as shown in Fig. 2.40.

Original plain-text message: **Come home tomorrow**

1. Let us consider a rectangle with six columns. Therefore, when we write the message in the rectangle row by row, it would look as follows:

Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
C	o	m	e	h	o
m	e	t	o	m	o
r	r	o	w		

2. Now, let us decide the order of columns as some random order, say 4, 6, 1, 2, 5, 3. Then read the text in the order of these columns.
3. The cipher text thus obtained would be **eowoocmroerhmmto** in round 1.
4. Let us perform Steps 1 through 3 once more. So, the tabular representation of the cipher text after round 1 is as follows:

Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
e	o	w	o	o	c
m	r	o	e	r	h
m	m	t	o		

5. Now, let us use the same order of columns, as before, that is 4, 6, 1, 2, 5, and 3. Then read the text in the order of these columns.
6. The cipher text thus obtained would be **oeochemnmormorwot** in round 2.
7. Continue like this if more number of iterations is desired, otherwise stop.

Fig. 2.40 Example of simple columnar transposition technique with multiple rounds

As the figure shows, multiple rounds or iterations add more complexity to the cipher text produced by the basic simple columnar transposition technique. The more the number of iterations, the more complex is the cipher text thus produced.

Cipher text produced by the simple columnar transposition technique with multiple rounds is much more complex to crack as compared to the basic technique.

2.4.3 Vernam Cipher (One-Time Pad)

The **Vernam cipher**, whose specific subset is called **one-time pad**, is implemented using a random set of non-repeating characters as the input cipher text. The most significant point here is that once an input cipher text for transposition is used, it is never used again for any other message (hence the name *one-time*). The length of the input cipher text is equal to the length of the original plain text. The algorithm used in the Vernam cipher is described in Fig. 2.41.

1. Treat each plain-text alphabet as a number in an increasing sequence, i.e. A = 0, B = 1, ... Z = 25.
2. Do the same for each character of the input cipher text.
3. Add each number corresponding to the plain-text alphabet to the corresponding input cipher-text alphabet number.
4. If the sum thus produced is greater than 26, subtract 26 from it.
5. Translate each number of the sum back to the corresponding alphabet. This gives the output cipher text.

Fig. 2.41 Algorithm for Vernam cipher

Let us apply the Vernam cipher algorithm to a plain-text message *HOW ARE YOU* using a one-time pad *NCBTZQARX* to produce a cipher-text message *UQXTRUYFR* as shown in Fig. 2.42.

1. Plain text	H 7	O 14	W 22	A 0	R 17	E 4	Y 24	O 14	U 20
+									
2. One-time pad	13 N	2 C	1 B	19 T	25 Z	16 Q	0 A	17 R	23 X
3. Initial Total	20	16	23	19	42	20	24	31	43
4. Subtract 26, if > 25	20	16	23	19	16	20	24	5	17
5. Cipher text	U	Q	X	T	Q	U	Y	F	R

Fig. 2.42 Example of Vernam cipher

It should be clear that since the one-time pad is discarded after a single use, this technique is highly secure and suitable for small plain-text message, but is clearly impractical for large messages. The Vernam Cipher was first implemented at AT&T with the help of a device called the **Vernam machine**.

Vernam Cipher uses a one-time pad, which is discarded after a single use, and therefore, is suitable only for short messages.

2.4.4 Book Cipher/Running-Key Cipher

The idea used in **book cipher**, also incorrectly called **running-key cipher**, is quite simple, and is similar in principle to the Vernam cipher. For producing cipher text, some portion of text from a book is used, which serves the purpose of a one-time pad. Thus, the characters from a book are used as one-time pad, and they are *added* to the input plain-text message similar to the way a one-time pad works.

■ 2.5 ENCRYPTION AND DECRYPTION ■

We have discussed the concepts of plain text and how it can be transformed into cipher text so that only the sender and the recipient can make any sense out of it. There are technical terms to describe these concepts, which we shall learn now. In technical terms, the process of encoding plain-text messages into cipher text messages is called **encryption**. Figure 2.43 illustrates the idea.

The reverse process of transforming cipher-text messages back to plain text messages is called **decryption**. Figure 2.44 illustrates the idea.

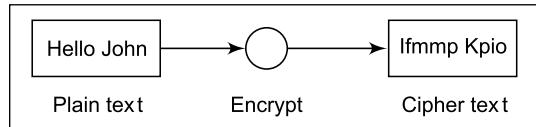


Fig. 2.43 Encryption

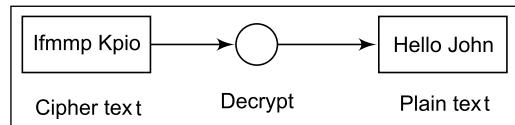


Fig. 2.44 Decryption

Decryption is exactly the opposite of encryption. Encryption transforms a plain-text message into cipher text, whereas decryption transforms a cipher text message back into plain text.

In computer-to-computer communications, the computer at the sender's end usually transforms a plain-text message into ciphertext by performing encryption. The encrypted cipher-text message is then sent to the receiver over a network (such as the Internet, although it can be any other network). The receiver's computer then takes the encrypted message, and performs the reverse of encryption, i.e. it performs the decryption process to obtain the original plain-text message. This is shown in Figure 2.45.

To encrypt a plain-text message, the sender (we shall henceforth treat the term *sender* to mean the *sender's computer*) performs encryption, i.e. applies the encryption algorithm. To decrypt a received encrypted message, the recipient performs decryption, i.e. applies the **decryption algorithm**. The algorithm is similar in concept to the algorithms we discussed earlier.

Clearly, the decryption algorithm must be the same as the **encryption algorithm**. Otherwise, decryption would be unable to retrieve the original message. For instance, if the sender uses the rail-fence technique for encryption and the receiver uses the simple columnar technique for decryption, the decryption would yield a totally incorrect plain text. Thus, the sender and the receiver must agree on a common algorithm for any meaningful communication can take place. The algorithm basically takes one text as input and produces another as the output.

The second aspect of performing encryption and decryption of messages is the **key**. What is a key? A key is something similar to the one-time pad used in the Vernam cipher. Anyone can use the Vernam cipher. However, as long as only the sender and the receiver know the one-time pad, no one except the sender and the receiver can do anything with the message.

*Every encryption and decryption process has two aspects: the **algorithm** and the **key** used for encryption and decryption.*

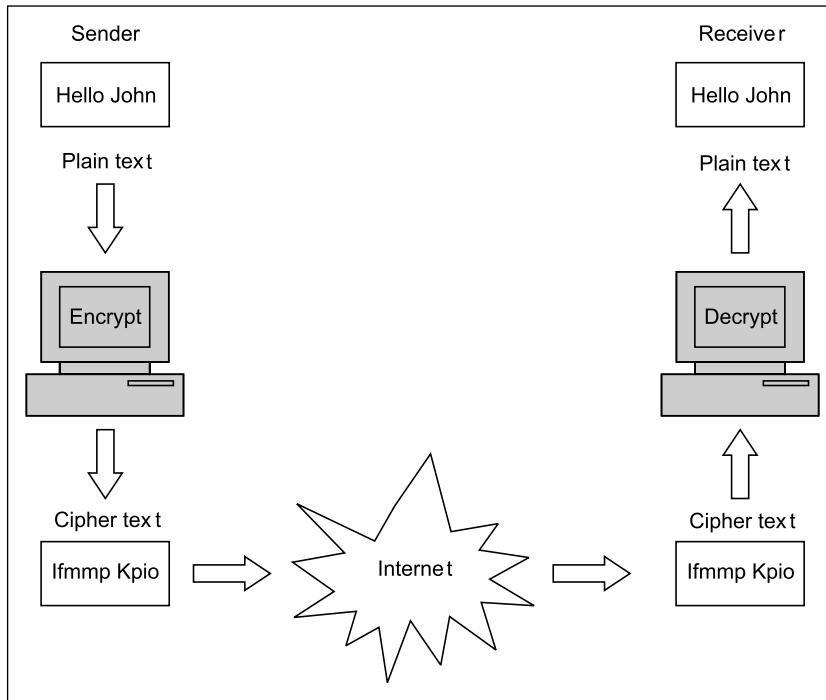


Fig. 2.45 Encryption and decryption in the real world

This is shown in Fig. 2.46.

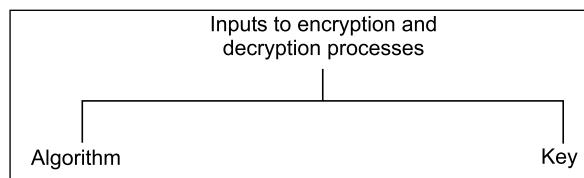


Fig. 2.46 Aspects of encryption and decryption

To understand this better, let us take the example of a combination lock which we use in real life. We need to remember the combination (which is a number, such as 871) needed to open up the lock. The facts that it is a combination lock and how to open it (*algorithm*) are pieces of public knowledge. However, the actual value of the key required for opening a specific lock (*key*), which is 871 in this case, is kept secret. The idea is illustrated in Fig. 2.47.

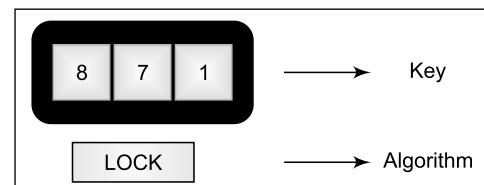


Fig. 2.47 Combination lock

Thus, as an example, the sender and receiver can safely agree to use Vernam cipher as the algorithm, and XYZ as the key, and be assured that no one else is able to get any access to their conversation. Others might know that the Vernam cipher is in use. However, they do not know that XYZ is the encryption/decryption key.

In general, the algorithm used for encryption and decryption processes is usually known to everybody. However, it is the key used for encryption and decryption that makes the process of cryptography secure.

Broadly, there are two cryptographic mechanisms, depending on what keys are used. If the *same* key is used for encryption and decryption, we call the mechanism **symmetric key cryptography**. However, if two *different* keys are used in a cryptographic mechanism, wherein one key is used for encryption, and *another, different* key is used for decryption; we call the mechanism **asymmetric key cryptography**. This is shown in Fig. 2.48.

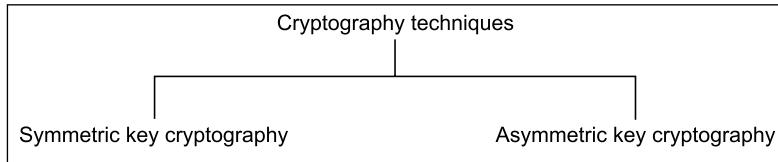


Fig. 2.48 Cryptography techniques

We shall study the basic concepts behind these two mechanisms now. We shall study the various computer-based cryptographic algorithms in each of these categories in great detail in subsequent chapters.

Symmetric key cryptography involves the usage of the same key for encryption and decryption. Asymmetric key cryptography involves the usage of one key for encryption, and another, different key for decryption.

■ 2.6 SYMMETRIC AND ASYMMETRIC KEY CRYPTOGRAPHY ■

2.6.1 Symmetric Key Cryptography and the Problem of Key Distribution

Before we discuss computer-based symmetric and asymmetric key cryptographic algorithms (in the next few chapters), we need to understand why we need two different types of cryptographic algorithms in the first place. To understand this, let us consider a simple problem statement.

Person A wants to send a highly confidential letter to another person B. A and B both reside in the same city, but are separated by a few miles, and for some reason, cannot meet each other.

Let us now see how we can tackle this problem. The simplest solution would appear to be that A puts the confidential letter in an envelope, seals it, and sends it by post. A hopes that no one opens it before it reaches B. This is shown in Fig. 2.49.

Clearly, this solution does not seem to be acceptable. What is the guarantee that an unscrupulous person does not obtain and open the envelope before it reaches B? Sending the envelope by registered post or courier might slightly improve the situation, but will not guarantee that the envelope does not get opened before it reaches B. After all, someone can open the envelope, read the confidential letter and re-seal the envelope!

Another option is to send the envelope via a hand-delivery mechanism. Here, A hands the envelope over to another person P, who personally hand-delivers the envelope to B. This seems to be a slightly better solution. However, it is still not foolproof.

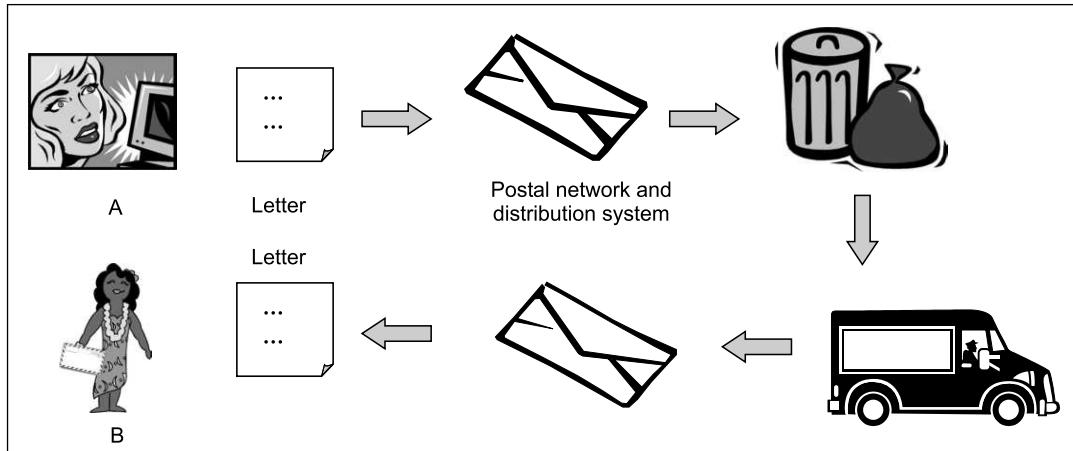


Fig. 2.49 Simplest channel to send a confidential letter

Consequently, A comes up with another idea. A now puts the envelope inside a box, seals that box with a highly secure lock, and sends the box to B (through the mechanism of post/courier/hand-delivery). Since the lock is highly secure, nobody can open the box while in transit, and therefore, open the envelope. Consequently, nobody will be able to read/access the highly confidential letter! The problem is resolved! If we think about it, we will realize that the problem indeed seems to be resolved. However, this solution has given birth to a new problem. How on earth can the intended recipient (B) now open the box, and therefore, the envelope? This solution has not only prevented unauthorized access to the letter, but also the authorized access. That is, even B would not be able to open the lock. This defeats the purpose of sending the letter in this manner, in the first place.

What if A also sends the key of the lock along with the box, so that B can open the lock, and get access to the envelope inside the box, and hence the letter? This seems absurd. If the key travels with the box, anybody who has access to the box in transit (e.g. P) can unlock and open the box.

Therefore, A now comes up with an improved plan. A decides that the locked box should travel to B as discussed (by post/courier/hand-delivery). However, she will not send the key used to lock the box along with the box. Instead, she will decide a place and a time to meet B in person, meet B at that time, and hand over the key personally to B. This will ensure that the key does not land up in the wrong hands, and that only B can access the confidential letter! This now seems to be a foolproof solution! Is it, really?

If A can meet B in person to hand over the key, she can as well hand the confidential letter to B in person! Why have all these additional worries and overheads? Remember that the whole problem started because A and B cannot, for some reason, meet in person!

As a result, we will observe that no solution is completely acceptable. Either it is not foolproof, or is not practically possible. This is the problem of **key distribution** or **key exchange**. Since the sender and the receiver will use the same key to lock and unlock, this is called *symmetric key operation* (when used in the context of cryptography, this operation is called **symmetric key cryptography**). Thus, we observe that the key distribution problem is inherently linked with the symmetric key operation.

Let us now imagine that not only A and B but also thousands of people want to send such confidential letters securely to each other. What would happen if they decide to go for symmetric key operation? If

we examine this approach more closely, we can see that it has one big drawback if the number of people that want to avail of its services is very large.

We will start with small numbers and then inspect this scheme for a larger number of participants. For instance, let us assume that A now wants to communicate with two persons, B and C, securely. Can A use the same kind of lock (i.e. a lock with the same properties, which can be opened with the same key) and key for sealing the box to be sent to B and C? Of course, this is not advisable at all! After all, if A uses the same kind of lock and key to seal the boxes addressed for B and C, what is the guarantee that B does not open the box intended for C, or vice versa (because B and C would also possess the same key as A)? Even if B and C live in the two extreme corners of the city, A cannot simply take such a chance! Therefore, no matter how secure the lock and key is, A must use a *different* lock-and-key pair for B and C. This means that A must buy two *different* locks and the corresponding two keys (i.e. one key per lock). This is shown in Fig. 2.50. It must also somehow send the respective lock-opening keys to B and C.

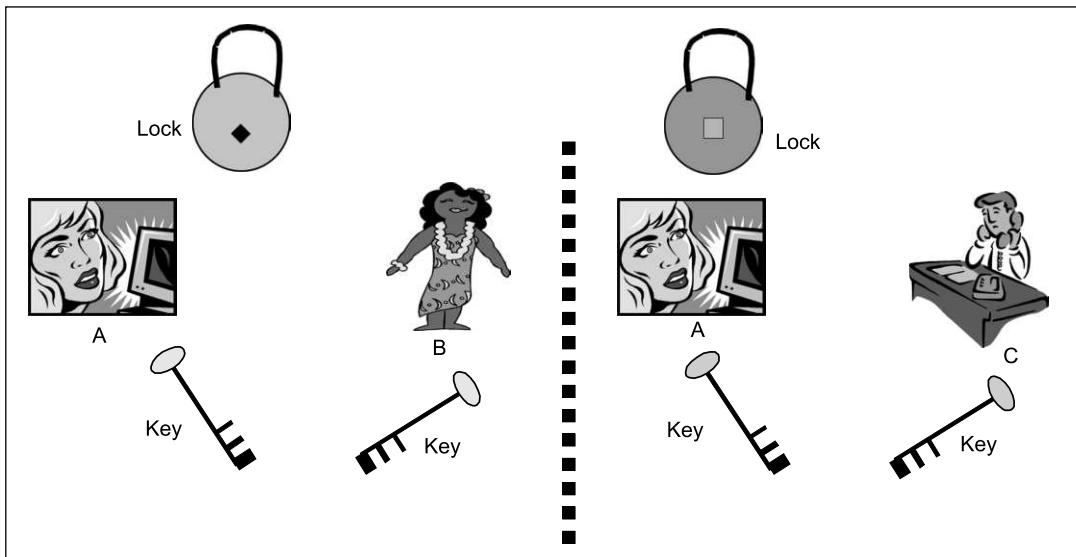


Fig. 2.50 Use of separate locks and keys per communication pair

Thus, we have the following situation:

- When A wanted to communicate only with B, we needed one lock-and-key pair (A-B).
- When A wants to communicate with B and C, we need two lock-and-key pairs (A-B and A-C). Thus, we need one lock-and-key pair per person with whom A wants to communicate. If B also wants to communicate with C, we have B-C as the third communicating pair, requiring its own lock-and-key pair. Thus, we would need three lock-and-key pairs to serve the needs of three communicating pairs.
- Let us consider the participation of a fourth person D. Let us also imagine that all of the four persons (A, B, C and D) want to be able to communicate with each other securely. Thus, we have six communicating pairs, namely A-B, A-C, A-D, B-C, B-D and C-D. Thus, we need six lock-and-key pairs, one per communicating pair, to serve the needs of four communicating pairs.

- If E is the fifth person joining this group, we have ten communicating pairs, namely A-B, A-C, A-D, A-E, B-C, B-D, B-E, C-D, C-E and D-E. Thus, we would need ten lock-and-key pairs to make secure communication between all these pairs possible.

Let us now tabulate these results as shown in Fig. 2.51 to see if any pattern emerges.

Parties involved	Number of lock-and-key pairs required
2 (A, B)	1 (A-B)
3 (A, B, C)	3 (A-B, A-C, B-C)
4 (A, B, C, D)	6 (A-B, A-C, A-D, B-C, B-D, C-D)
5 (A, B, C, D, E)	10 (A-B, A-C, A-D, A-E, B-C, B-D, B-E, C-D, C-E, D-E)

Fig. 2.51 Number of parties and the corresponding number of lock-and-key pairs

We can conclude the following:

- If the number of parties is 2, we need $2 * (2 - 1) / 2 = 2 * (1) / 2 = 1$ lock-and-key pair.
- If the number of parties is 3, we need $3 * (3 - 1) / 2 = 3 * (2) / 2 = 3$ lock-and-key pairs.
- If the number of parties is 4, we need $4 * (4 - 1) / 2 = 4 * (3) / 2 = 6$ lock-and-key pairs.
- If the number of parties is 5, we need $5 * (5 - 1) / 2 = 5 * (4) / 2 = 10$ lock-and-key pairs.

Therefore, can we see that, in general, for n persons, the number of lock-and-key pairs is $n * (n - 1)/2$? Now, if we have about 1,000 persons in this scheme, we will have $1000 * (1000 - 1)/2 = 1000 * (999)/2 = 99,9000/2 = 499,500$ lock-and-key pairs!

Moreover, we must keep in mind that a record of which lock-and-key pair was issued to which communicating pair must be maintained by somebody. Let us call this somebody as T. This is required because it is quite possible that some persons might lose the lock or key, or both. In such cases, T must ensure that the proper duplicate key is issued, or that the lock is replaced with an exact replica of the lock, or that a different lock and key pair is issued (for security reasons), depending on the situation. This is quite a bit of task! Also, who is T, after all? T must be highly trustworthy and accessible to everybody. This is because each communicating pair has to approach T to obtain the lock-and-key pair. This is quite a tedious and time-consuming process!

2.6.2 Diffie–Hellman Key-Exchange/Agreement Algorithm

1. Introduction

Whitefield Diffie and Martin Hellman devised an amazing solution to the problem of key agreement, or key exchange, in 1976. This solution is called the **Diffie–Hellman key-exchange/agreement algorithm**. The beauty of this scheme is that the two parties, who want to communicate securely, can agree on a symmetric key using this technique. This key can then be used for encryption/decryption. However, we must note that the Diffie–Hellman key exchange algorithm can be used only for key agreement, but not for encryption or decryption of messages. Once both the parties agree on the key to be used, they need to use other symmetric key-encryption algorithms (we shall discuss some of those subsequently) for actual encryption or decryption of messages.

Although the Diffie–Hellman key-exchange algorithm is based on mathematical principles, it is quite simple to understand. We shall first describe the steps in the algorithm, then illustrate its use with a simple example, and then discuss the mathematical basis for it.

2. Description of the Algorithm

Let us assume that Alice and Bob want to agree upon a key to be used for encrypting/decrypting messages that would be exchanged between them. Then, the Diffie–Hellman key-exchange algorithm works as shown in Fig. 2.52.

1. Firstly, Alice and Bob agree on two large prime numbers, n and g . These two integers need not be kept secret. Alice and Bob can use an insecure channel to agree on them.
2. Alice chooses another large random number x , and calculates A such that:

$$A = g^x \bmod n$$
3. Alice sends the number A to Bob.
4. Bob independently chooses another large random integer y and calculates B such that:

$$B = g^y \bmod n$$
5. Bob sends the number B to Alice.
6. Alice computes the secret key $K1$ as follows:

$$K1 = B^x \bmod n$$
7. Bob now computes the secret key $K2$ as follows:

$$K2 = A^y \bmod n$$

Fig. 2.52 Diffie–Hellman key-exchange algorithm

This is shown diagrammatically in Fig. 2.53.

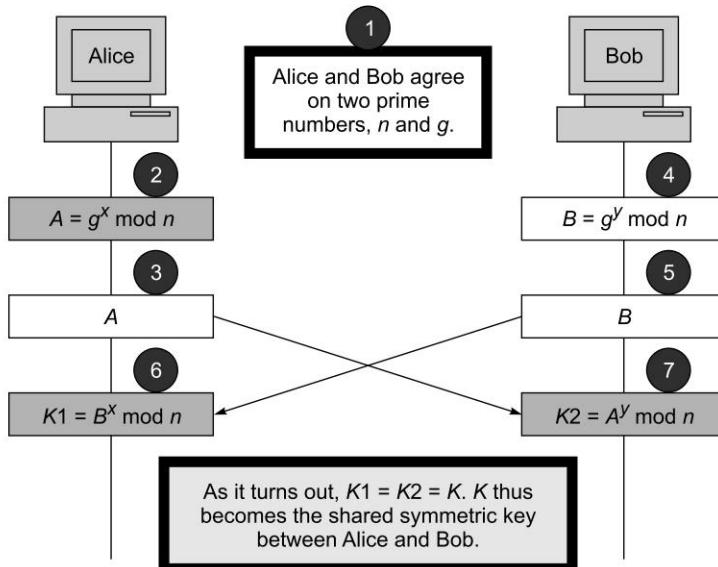


Fig. 2.53 Diffie–Hellman key-exchange

It might come as a surprise, but $K1$ is actually equal to $K2$! This means that $K1 = K2 = K$ is the symmetric key, which Alice and Bob must keep secret and can henceforth use for encrypting/decrypting their messages with. The mathematics behind this is quite interesting. We shall first prove it, and then examine it.

3. Example of the Algorithm

Let us take a small example to prove that the Diffie–Hellman works in practical situations. Of course, we shall use very small values for ease of understanding. In real life, these values are very large. The process of key agreement is shown in Fig. 2.54.

- Firstly, Alice and Bob agree on two large prime numbers, n and g . These two integers need not be kept secret. Alice and Bob can use an insecure channel to agree on them.

Let $n = 11, g = 7$.

- Alice chooses another large random number x , and calculates A such that:

$$A = g^x \bmod n$$

Let $x = 3$. Then, we have, $A = 7^3 \bmod 11 = 343 \bmod 11 = 2$.

- Alice sends the number A to Bob.

Alice sends 2 to Bob.

- Bob independently chooses another large random integer y and calculates B such that

$$B = g^y \bmod n$$

Let $y = 6$. Then, we have, $B = 7^6 \bmod 11 = 117649 \bmod 11 = 4$.

- Bob sends the number B to Alice.

Bob sends 4 to Alice.

- A now computes the secret key $K1$ as follows:

$$K1 = B^x \bmod n$$

We have, $K1 = 4^3 \bmod 11 = 64 \bmod 11 = 9$.

- B now computes the secret key $K2$ as follows:

$$K2 = A^y \bmod n$$

We have, $K2 = 2^6 \bmod 11 = 64 \bmod 11 = 9$.

Fig. 2.54 Example of Diffie–Hellman key exchange

Having taken a look at the actual proof of Diffie–Hellman key-exchange algorithm, let us now think about the mathematical theory behind it.

4. Mathematical Theory Behind the Algorithm

Let us first take a look at the technical (and quite complicated) description of the complexity of the algorithm:

The Diffie–Hellman key-exchange algorithm gets its security from the difficulty of calculating discrete logarithms in a finite field, as compared with the ease of calculating exponentiation in the same field.

Let us try to understand what this actually means, in simple terms.

(a) Firstly, take a look at what Alice does in step 6. Here, Alice computes:

$$K1 = B^x \bmod n.$$

What is B? From step 4, we have:

$$B = g^y \bmod n.$$

Therefore, if we substitute this value of B in step 6, we will have the following equation:

$$K1 = (g^y)^x \bmod n = g^{yx} \bmod n$$

(b) Now, take a look at what Bob does in step 7. Here, Bob computes:

$$K2 = A^y \bmod n.$$

What is A? From step 2, we have:

$$A = g^x \bmod n.$$

Therefore, if we substitute this value of A in step 7, we will have the following equation:

$$K2 = (g^x)^y \bmod n = g^{xy} \bmod n$$

Now, basic mathematics says that:

$$K^{yx} = K^{xy}$$

Therefore, in this case, we have $K1 = K2 = K$. This equation provides the proof.

An obvious question now is, if Alice and Bob can both calculate K independently, so can an attacker! What prevents this? The fact is, Alice and Bob exchange n , g , A and B . Based on these values, x (a value known only to Alice) and y (a value known only to Bob) cannot be calculated easily. Mathematically, the calculations to find out x and y are extremely complicated, if they are sufficiently large numbers. Consequently, an attacker cannot calculate x and y , and therefore, cannot derive K .

5. Working of the Diffie–Hellman Algorithm

The idea behind the Diffie–Hellman algorithm is quite simple but beautiful. Think about the final shared symmetric key between Alice and Bob to be made up of three parts: g , x , and y , as shown in Fig. 2.55.

The first part of the key (g) is a public number, known to everyone. The other two parts of the key (i.e. x and y) must be made available by Alice and Bob. Alice adds the second part (x), while Bob adds the third (y). When Alice receives the two-thirds completed key from Bob, she adds the one-third remaining part (i.e. x). This completes Alice's key. Similarly, when Bob receives the two-thirds completed key from Alice, he adds the one-third remaining part (i.e. y). This completes Bob's key.

Note that although Alice's key is made up using a sequence of $g-y-x$, and Bob's key is made up using a sequence of $g-x-y$, the two keys are the same because $g^{xy} = g^{yx}$.

Importantly, although the eventual two keys are the same, Alice cannot find Bob's part (i.e. y) because the computation is done using modulus n . Similarly, Bob cannot derive Alice's part (i.e. x).

6. Problems with the Algorithm

Can we now consider that the Diffie–Hellman key-exchange algorithm solves all our problems associated with key exchange? Unfortunately, not quite!

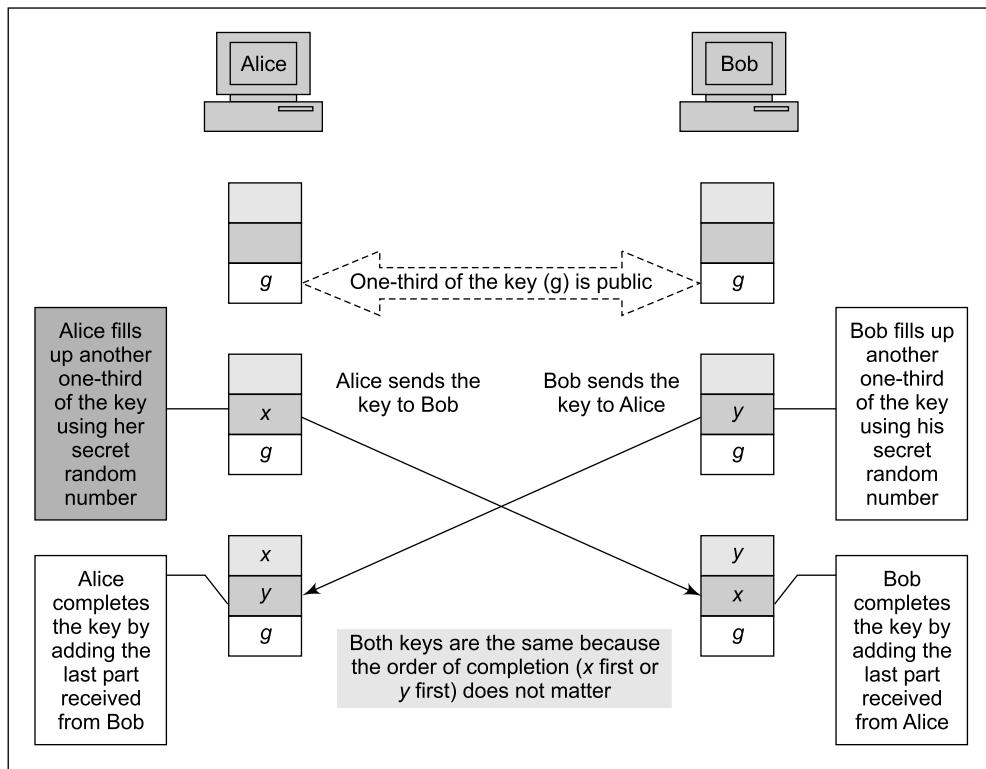


Fig. 2.55 Working of the Diffie–Hellman algorithm

The Diffie–Hellman key-exchange algorithm can fall prey to the **man-in-the-middle attack** (or to be politically correct, *woman-in-the-middle attack*), also called **bucket-brigade attack**. The name *bucket-brigade attack* comes from the way firefighters of yesteryears formed a line between the fire and water source, and passed full buckets towards the fire and the empty buckets back. The way this happens is as follows.

1. Alice wants to communicate with Bob securely, and therefore, she first wants to do a Diffie–Hellman key exchange with him. For this purpose, she sends the values of n and g to Bob, as usual. Let $n = 11$ and $g = 7$. (As usual, these values will form the basis of Alice's A and Bob's B , which will be used to calculate the symmetric key $K1 = K2 = K$.)
2. Alice does not realize that the attacker Tom is listening quietly to the conversation between her and Bob. Tom simply picks up the values of n and g , and also forwards them to Bob as they originally were (i.e. $n = 11$ and $g = 7$). This is shown in Fig. 2.56.
3. Now, let us assume that Alice, Tom and Bob select random numbers x and y as shown in Fig. 2.57.

Alice	Tom	Bob
$n = 11, g = 7$	$n = 11, g = 7$	$n = 11, g = 7$

Fig. 2.56 Man-in-the-middle attack—Part I

Alice	Tom	Bob
$x = 3$	$x = 8, y = 6$	$y = 9$

Fig. 2.57 Man-in-the-middle attack—Part II

4. One question at this stage could be: why does Tom select both x and y ? We shall answer that shortly. Now, based on these values, all the three persons calculate the values of A and B as shown in Fig. 2.58. Note that Alice and Bob calculate only A and B , respectively. However, Tom calculates both A and B . We shall revisit this shortly.

Alice	Tom	Bob
$A = g^x \bmod n$	$A = g^x \bmod n$	$B = g^y \bmod n$
$= 7^3 \bmod 11$	$= 7^8 \bmod 11$	$= 7^9 \bmod 11$
$= 343 \bmod 11$	$= 5764801 \bmod 11$	$= 40353607 \bmod 11$
$= 2$	$= 9$	$= 8$
	$B = g^y \bmod n$	
	$= 7^6 \bmod 11$	
	$= 117649 \bmod 11$	
	$= 4$	

Fig. 2.58 Man-in-the-middle attack—Part III

5. Now, the real drama begins, as shown in Fig. 2.59.

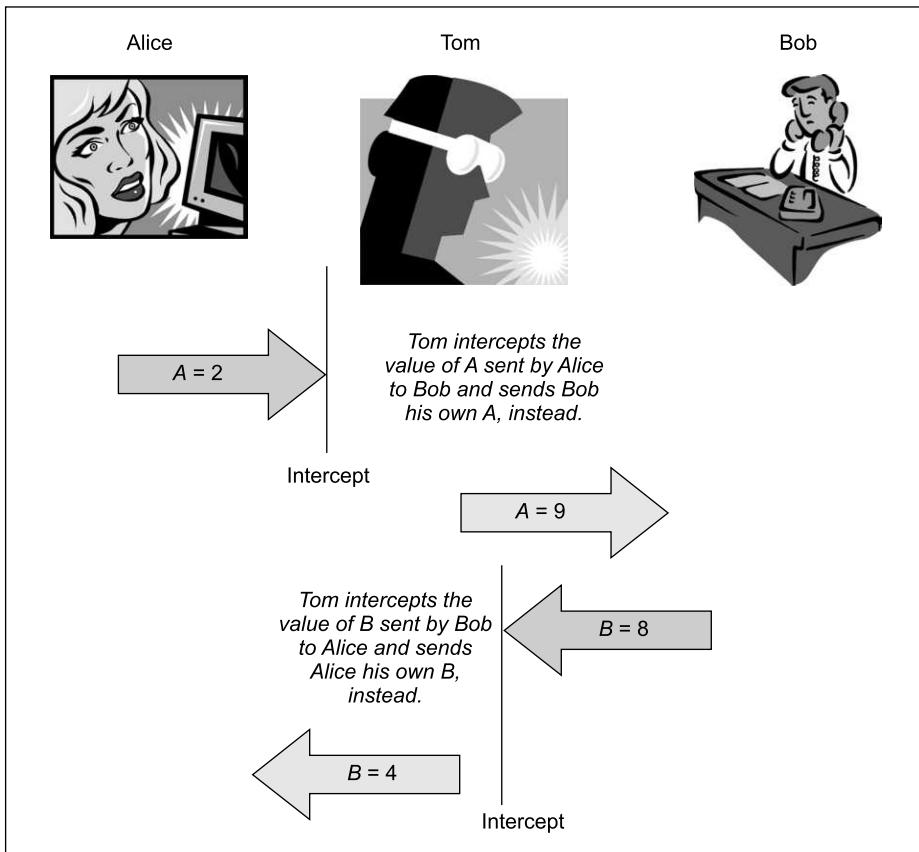


Fig. 2.59 Man-in-the-middle attack—Part IV

As shown in the figure, the following things happen:

- (a) Alice sends her A (i.e. 2) to Bob. Tom intercepts it, and instead, sends his A (i.e. 9) to Bob. Bob has no idea that Tom had hijacked Alice's A and has instead given his A to Bob.
- (b) In return, Bob sends his B (i.e. 8) to Alice. As before, Tom intercepts it, and instead, sends his B (i.e. 4) to Alice. Alice thinks that this B came from Bob. She has no idea that Tom had intercepted the transmission from Bob, and changed B .
- (c) Therefore, at this juncture, Alice, Tom and Bob have the values of A and B as shown in Fig. 2.60.

Alice	Tom	Bob
$A = 2, B = 4^*$	$A = 2, B = 8$	$A = 9^*, B = 8$
(Note: * indicates that these are the values after Tom hijacked and changed them.)		

Fig. 2.60 Man-in-the-middle attack—Part V

Based on these values, all the three persons now calculate their keys as shown in Fig. 2.61. We will notice that Alice calculates only K_1 , Bob calculates only K_2 , whereas Tom calculates both K_1 and K_2 . Why does Tom need to do this? We shall discuss that soon.

Alice		Tom		Bob	
K_1	$= B^x \bmod n$	K_1	$= B^x \bmod n$	K_2	$= A^y \bmod n$
	$= 4^3 \bmod 11$		$= 8^8 \bmod 11$		$= 9^9 \bmod 11$
	$= 64 \bmod 11$		$= 16777216 \bmod 11$		$= 387420489 \bmod 11$
	$= 9$		$= 5$		$= 5$
		K_2	$= A^y \bmod n$		
			$= 2^6 \bmod 11$		
			$= 64 \bmod 11$		
			$= 9$		

Fig. 2.61 Man-in-the-middle attack—Part VI

Let us now revisit the question as to why Tom needs two keys. This is because at one side, Tom wants to communicate with Alice securely using a shared symmetric key (9), and on the other hand, he wants to communicate with Bob securely using a *different* shared symmetric key (5). Only then can he receive messages from Alice, view/manipulate them and forward them to Bob, and vice versa. Unfortunately for Alice and Bob, both will (incorrectly) believe that they are directly communicating with each other. That is, Alice will feel that the key 9 is shared between her and Bob, whereas Bob will feel that the key 5 is shared between him and Alice. Actually, what is happening is, Tom is sharing the key 5 with Alice and 9 with Bob!

This is also the reason why Tom needed both sets of the secret variables x and y , as well as later on, the non-secret variables A and B .

As we can see, the *man-in-the-middle attack* can work against the Diffie–Hellman key-exchange algorithm, causing it to fail. This is plainly because the *man-in-the-middle* makes the actual communicators believe that they are talking to each other, whereas they are actually talking to the *man-in-the-middle*, who is talking to each of them!

This attack can be prevented if Alice and Bob authenticate each other before beginning to exchange information. This proves to Alice that Bob is indeed Bob, and not someone else (e.g. Tom) posing as Bob. Similarly, Bob can also get convinced that Alice is genuine as well.

However, not everything is lost. If we think deeply and try to come up with the scheme that will still work, an alternative emerges, namely **asymmetric key operation**.

2.6.3 Asymmetric Key Operation

In this scheme, A and B do not have to jointly approach T for a lock-and-key pair. Instead, B alone approaches T, obtains a lock and a key (K_1) that can seal the lock, and sends the lock and key K_1 to A. B tells A that A can use that lock and key to seal the box before sending the sealed box to B. How can B open the lock, then?

An interesting property of this scheme is that B possesses a *different but related* key (K_2), which is obtained by B from T along with the lock and key K_1 , only which can open the lock. It is guaranteed that no other key, and of course, including the one used by A (i.e. K_1) for locking, can open the lock. Since one key (K_1) is used for locking, and *another, different* key (K_2) is used for unlocking; we will call this scheme as *asymmetric key operation*. Also, T is clearly defined here as a **trusted third party**. T is certified as a highly trustworthy and efficient agency by the government.

This means that B possesses a **key pair** (i.e. two keys K_1 and K_2). One key (i.e. K_1) can be used for locking, and only the corresponding other key (i.e. K_2) from the key pair can be used for unlocking. Thus, B can send the lock and key K_1 to anybody (e.g. A) who wants to send anything securely to B. B would request the sender (e.g. A) to use that lock and key K_1 to seal the contents. B can then open the seal using the key K_2 . Since the key K_1 is meant for locking, and is available to the general public, we shall call K_1 **public key**. Note that K_1 need not be secret—in fact, it *should not be* secret! Thus, unlike what happens in the case of symmetric key operation, the (locking) key need not be guarded secretly now. The other key K_2 is meant for unlocking, and is strictly held secret/private by A. Therefore, we shall call it the **private key or secret key**.

This is shown in Fig. 2.62.

Note that if B wants to receive something securely from another person say C, B need *not* obtain a fresh lock-and-key pair. B can send *the same* lock-and-key (K_1) pair (or a copy of the lock and B's *public key* K_1 , in case A and C want to send something securely to B at the same time) to C. Thus, C will also use the same lock and B's *public key* K_1 to seal the contents before sending them to B. As before, B will use the corresponding *private key* K_2 to open the lock. Extending this concept a step further, if B wants to receive messages securely from 10,000 different persons, B can send the same lock-and-public key K_1 (or their copies) to each one of them! It need not have 10,000 unique locks and keys (unlike the symmetric key approach)! Always, B's public key K_1 will be used by the sender for locking, and B's private key K_2 will be used for unlocking by the receiver (i.e. B).

Clearly, this is an extremely convenient approach, as compared to symmetric key operation!

Let us now consider what happens if three persons A, B and C want to communicate with each other. That is, A, B and C must all be able to send/receive messages securely to/from each other. For this to be possible, all the three persons can obtain a lock-and-public key pair from the trusted third party (T). Whenever any one of them wants to receive a message securely from another person, he/she has to send her lock-and-public key to the sender. That is, when A wants to receive a message securely from B, A

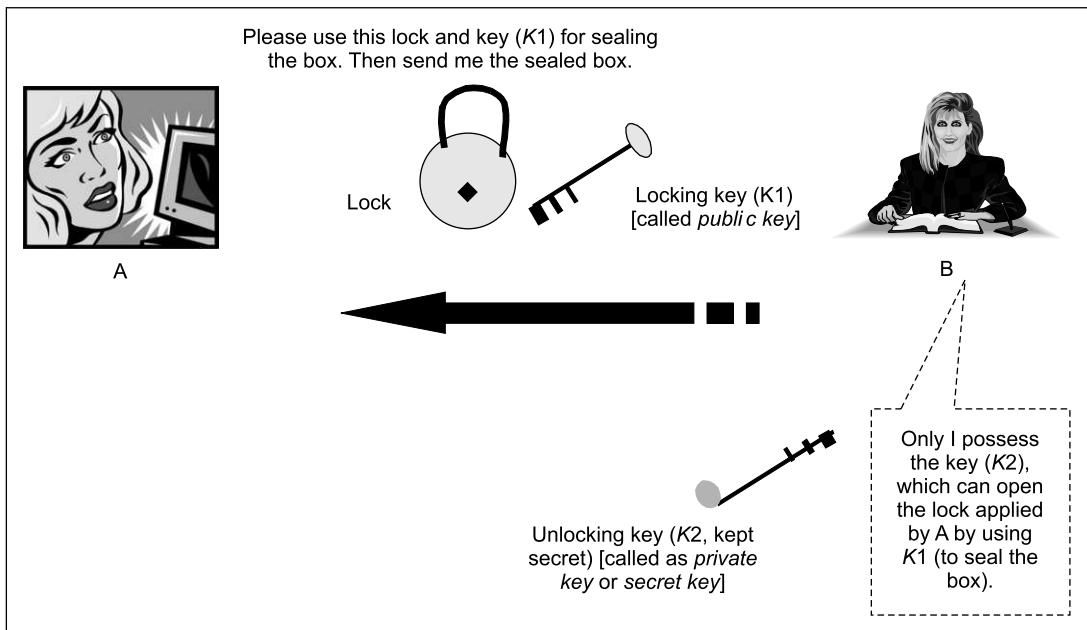


Fig. 2.62 Use of key pair

sends his/her lock and public key to B. B can use that to seal the message and send the sealed message to A. A can then use her private key to open the lock. Similarly, when B wants to receive a message securely from A, B sends his/her lock and public key to A, using which B can secure the message. Since only B has his/her own private key, he/she can open the lock and access the message.

Extending this basic idea, if 1,000 people want to be able to securely communicate with each other, only 1,000 locks, 1,000 public keys and the corresponding 1,000 private keys are required. This is in stark contrast to the symmetric key operation wherein for 1,000 participants, we needed 499,500 lock-and-key pairs (please refer to our earlier discussion).

Therefore, in general, when using asymmetric key operation, the recipient has to send the lock and his/her public key to the sender. The sender uses these to apply the lock and sends the sealed contents to the recipient. The recipient uses his/her private key to open the lock. Since only the recipient possesses the private key, all concerned are assured that only the intended recipient can open the lock.

■ 2.7 STEGANOGRAPHY ■

Steganography is a technique that facilitates hiding of a message that is to be kept secret inside other messages. This results in the concealment of the secret message itself! Historically, the sender used methods such as invisible ink, tiny pin punctures on specific characters, minute variations between handwritten characters, pencil marks on handwritten characters, etc.

Of late, people hide secret messages within graphic images. For instance, suppose that we have a secret message to send. We can take another image file and we can replace the last two rightmost bits of each byte of that image with (the next) two bits of our secret message. The resulting image would not look

too different, and yet carry a secret message inside! The receiver would perform the opposite trick: it would read the last two bits of each byte of the image file, and reconstruct the secret message. This concept is illustrated in Fig. 2.63.

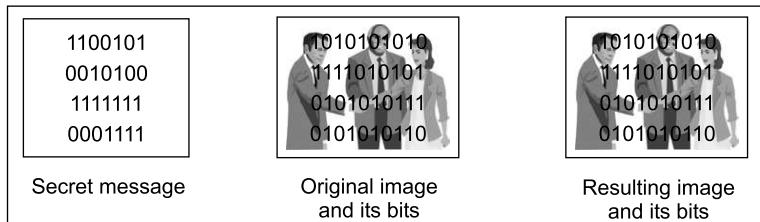


Fig. 2.63 Steganography example

■ 2.8 KEY RANGE AND KEY SIZE ■

We have already seen one way to classify attacks on information itself, or information sources (such as computers or networks). However, the encrypted messages can be attacked, too! Here, the cryptanalyst is armed with the following information:

- The encryption/decryption algorithm
- The encrypted message
- Knowledge about the key size (e.g. the value of the key is a number between 0 and 100 billion)

As we have seen previously, the encryption/decryption algorithm is usually not a secret—everybody knows about it. Also, one can access an encrypted message by various means (such as by listening to the flow of information over a network). Thus, only the actual value of the key remains a challenge for the attacker. If the key is found, the attacker can resolve the mystery by working backwards to the original plain-text message, as shown in Fig. 2.64. We shall consider the brute-force attack here, which works on the principle of trying every possible key in the **key range**, until you get the right key.

It usually takes a very small amount of time to try a key. The attacker can write a computer program that tries many such keys in one second. In the best case, the attacker finds the right key in the first attempt itself, and in the worst case, it is the 100 billionth attempt. However, the usual observation is that the key is found somewhere in between the possible range. Mathematics tells us that on an average, the key can be found after about half of the possible values in the key range are checked. Of course, this is just a guideline, and may or may not work in real practice for a given situation.

As Fig. 2.64 shows, the attacker has access to the cipher-text block and the encryption/decryption algorithm. He/she also knows the key range (a number between 0 and 100 billion). The attacker now starts trying every possible key, starting from 0. After every decryption, he/she looks at the generated *plain text* (actually, it is not truly plain text, but decrypted text, but we shall ignore this technical detail). If the attacker notices that the decryption has yielded unintelligent plain text, she continues the process with the next key in the sequence. Finally, she is able to find the right key with a value 90,171,451,191, which yields the plain text *To: Payroll*.

How does the attacker determine if the plain text, and therefore the key, are the right ones? This can be determined depending on the value of the plain text. If the plain text seems reasonable (i.e. very close

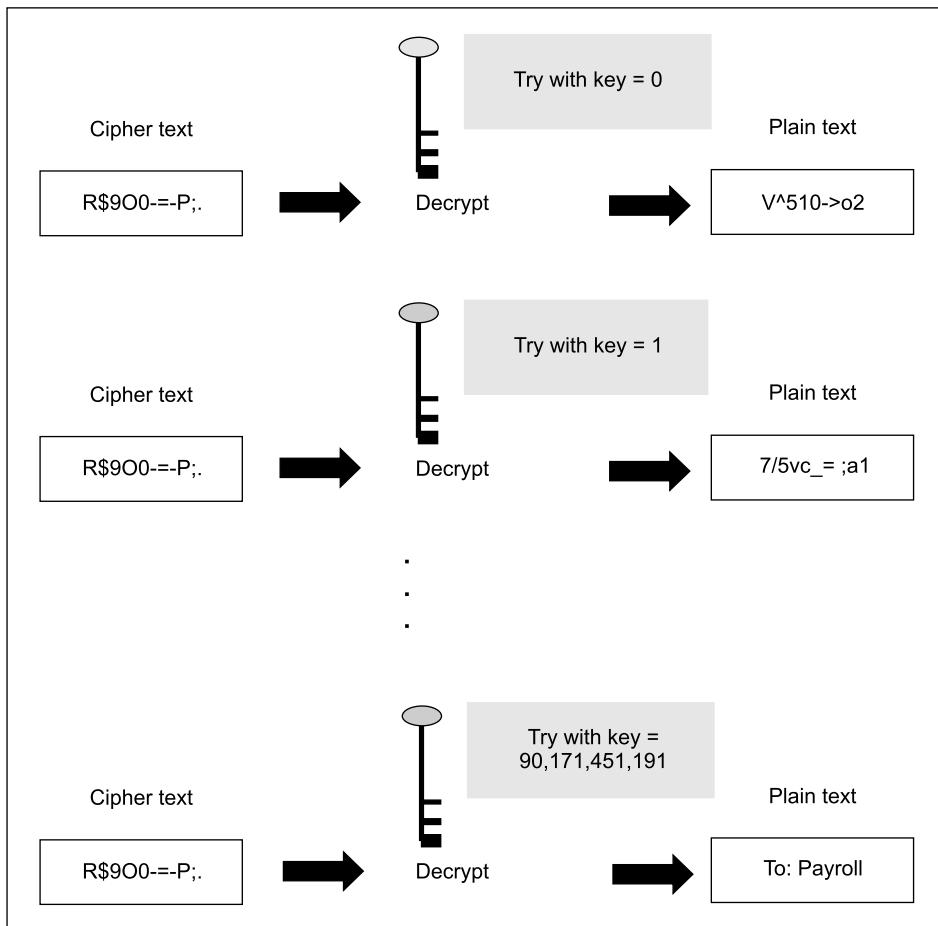


Fig. 2.64 Brute-force attack

to actual English words/sentences/numbers that make sense), it is highly probable that the plain text is indeed what corresponds to the cipher text.

This means that our key, and therefore, the plain-text message, are now cracked! How can we prevent an attacker from succeeding in such attempts? As we know, currently, our key range is 0 to 100 billion. Also, let us assume that the attacker took only 5 minutes to successfully crack our key. However, we want our message to remain secret for at least 5 years. This means that the attacker must spend at least 5 years in trying out every possible key, in order to obtain our original plain-text message. Therefore, the solution to our problem lies in expanding or increasing the *key range* to a size, which requires the attacker to work for more than 5 years in order to crack the key. Perhaps our key range should be from 0 to 100 billion billion billion.

In computer terms, the concept of *key range* leads us to the principle of **key size**. Just as we measure the value of stocks with a given index, gold in troy ounces, money in dollars, pounds or rupees; we measure the strength of a cryptographic key with *key size*. We measure key size in bits, and represent it using the binary number system. Thus, our key might be of 40 bits, 56 bits, 128 bits, and so on. In order to

protect ourselves against a brute-force attack, the key size should be such that the attacker cannot crack it within a specified amount of time. How long should it be? Let us study this.

At the simplest level, the key size can be just 1 bit. This means that the key can be either 0 or 1. If the key size is 2, the possible key values are 00, 01, 10, 11. Obviously, these examples are merely to understand the theory, and have no practical significance.

From a practical viewpoint, a 40-bit key takes about 3 hours to crack. However, a 41-bit key would take 6 hours, a 42-bit key takes 12 hours, and so on. This means that every additional bit doubles the amount of time required to crack the key. Why is this so? This works on the simple theory of binary numbers wherein every additional bit doubles the number of possible states of the number. This is shown in Fig. 2.65.

A 2-bit binary number has four possible states:
00
01
10
11
If we have one more bit to make it a 3-bit binary number, the number of possible states also doubles to eight, as follows:
000
001
010
011
100
101
110
111
In general, if an n -bit binary number has k possible states, an $n + 1$ bit binary number will have $2k$ possible states.

Fig. 2.65 Understanding key range

Thus, with every incremental bit, the attacker has to perform double the number of operations as compared to the previous key size. It is found that for a 56-bit key, it takes 1 second to search 1 percent of the key range. Taking this argument further, it takes about 1 minute to search about half of the key range (which is what is required, on an average, to crack a key). Using this as the basis, let us have a look at the similar values (time required for a search of 1 percent and 50 percent of the key space) for various key sizes. This is shown in Fig. 2.66.

Key size on bits	Time required to search 1 percent of the key space	Time required to search 50 percent of the key space
56	1 second	1 minute
57	2 seconds	2 minutes
58	4 seconds	4 minutes
64	4.2 minutes	4.2 hours
72	17.9 hours	44.8 days
80	190.9 days	31.4 years
90	535 years	321 centuries
128	146 billion millennia	8 trillion millennia

Fig. 2.65 Efforts required to break a key

We can represent the possible values in the key range using hexadecimal notation, and see visually how an increase in the key size increases the key range, and therefore, the complexity for an attacker. This is shown in Fig. 2.67.

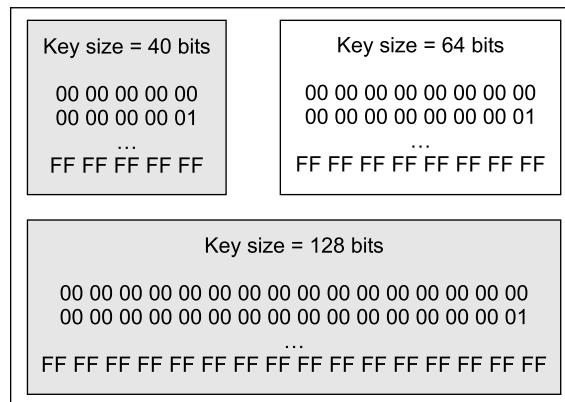


Fig. 2.67 Key sizes and ranges

Clearly, we can assume with reasonable confidence that a 128-bit key is quite safe (because 2^{128} means about 340,000,000,000,000,000,000,000,000,000,000,000 possible keys!), considering the capabilities of computers today. Obviously, as computing power and techniques improve, these numbers change. May be, in a few years time, a 128-bit key will be cracked. Then, we would need to rely on keys of size at least 256 bits, or 512 bits.

One may think that with the technological progress chasing key sizes so fast, how long can this go on and on? Today, 56-bit keys are not safe, tomorrow, 128-bit keys may not be sufficient, another day, 256-bit keys can be cracked, and so on! How far can we go? Well, it is argued that we may not have to look beyond 512-bit keys at any point of time in the future. That is, 512-bit keys will always be safe. What substantiates this argument?

Suppose that every atom in the universe is actually a computer. Then, we would have 2^{300} such computers in the world. Now, if each of these computers could check 2^{300} keys in one second, it would take 2^{162} millennia to search 1 percent of a 512-bit key! The Big Bang theory suggests that the amount of time that has passed since the universe came into existence is less than 2^{24} millennia, which is significantly less than the key search time. Thus, 512-bit keys will always be safe.

■ 2.9 POSSIBLE TYPES OF ATTACKS ■

Based on the discussion so far, when the sender of a message encrypts a plain-text message into its corresponding cipher text, there are five possibilities for an attack on this message, as shown in Fig. 2.68.

Let us discuss these attacks now.

1. Cipher-Text Only Attack

In this type of attack, the attacker does not have any clue about the plain text. She has some or all of the cipher text. (Interestingly, we should point out that if the attacker does not have an access even to the cipher text, there would be no need to encrypt the plain text to obtain cipher text in the first place!).

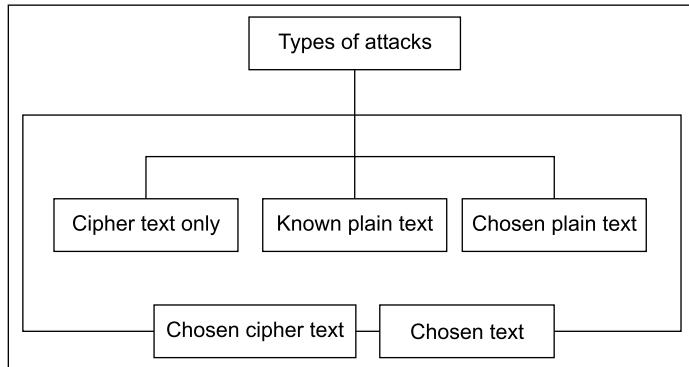


Fig. 2.68 Types of attacks

The attacker analyzes the cipher text at leisure to try and figure out the original plain text. Based on the frequency of letters (e.g. the alphabets e, i, a are very common in English, etc.) the attacker makes an attempt to guess the plain text. Obviously, the more cipher text available to the attacker, more are the chances of a successful attack. For instance, consider a very small cipher text block, RTQ. It is very difficult to guess the original plain text, given this block. There could be numerous possible plain texts, which yield this cipher text upon encryption. In contrast, if the cipher text size is bigger, the attacker can narrow down the various permutations and combinations to try and obtain the original plain text. The concept is shown in Fig. 2.69.

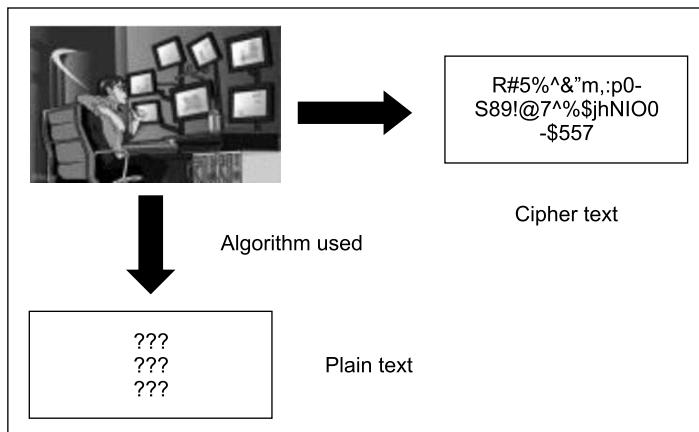


Fig. 2.69 Cipher text- only attack

We have mentioned that for this attack to succeed, the attacker should have access to sufficient amount of cipher text. The reason behind this is simple. For instance, if the attacker has cipher text available as ABC and knows that the encryption algorithm used was Monoalphabetic cipher, it is almost impossible to try and deduce the correct plain text. There are so many three-letter words in English that could correspond to this cipher text. Does this cipher text map to CAT, RAT, MAT, SHE, ARE, ...? This problem is shown in Fig. 2.70.

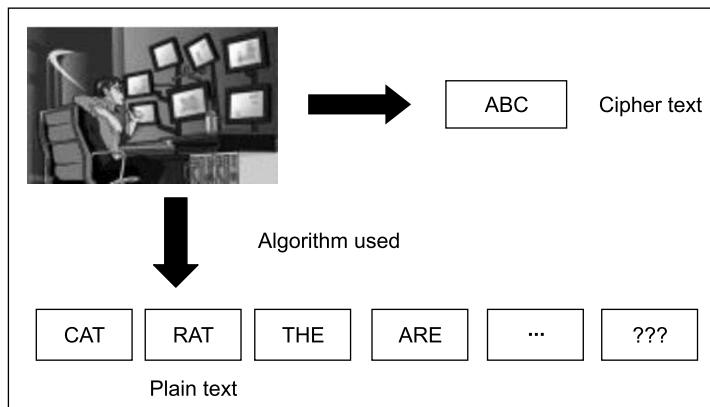


Fig. 2.70 Confusion for the attacker

2. Known Plain-Text Attack

In this case, the attacker knows about some pairs of plain text and corresponding cipher text for those pairs. Using this information, the attacker tries to find other pairs, and therefore, know more and more of the plain text. Examples of such *known plain texts* are company banners, file headers, etc., which are found commonly in all the documents of a particular company. How can the attacker obtain the plain text, in the first place? This can happen because plain-text information may become outdated over time, and hence, become public knowledge. Alternatively, it could be leaked inadvertently. This is shown in Fig. 2.71.

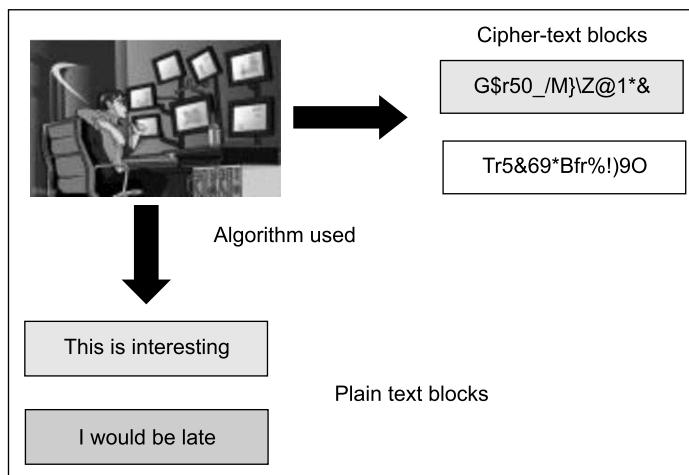


Fig. 2.71 Known plain text attack

3. Chosen Plain-Text Attack

Here, the attacker selects a plain-text block, and tries to look for the encryption of the same in the cipher text. Here, the attacker is able to choose the messages to encrypt. Based on this, the attacker intentionally picks patterns of cipher text that result in obtaining more information about the key.

How is this possible? For example, a telegraph company may offer a paid service where they encrypt people's messages and send them to the desired recipient. The telegraph company on the other side would decrypt the message and give the original message to the recipient. Therefore, it is quite possible for the attacker to choose some plain text, which she thinks is quite commonly used in secret messages. Therefore, the attacker chooses some such plain text and pays the telegraph company to encrypt it. The result of this is that the attacker now has access to some plain text that he/she had chosen, and its corresponding cipher text.

This concept is depicted in Fig. 2.72.

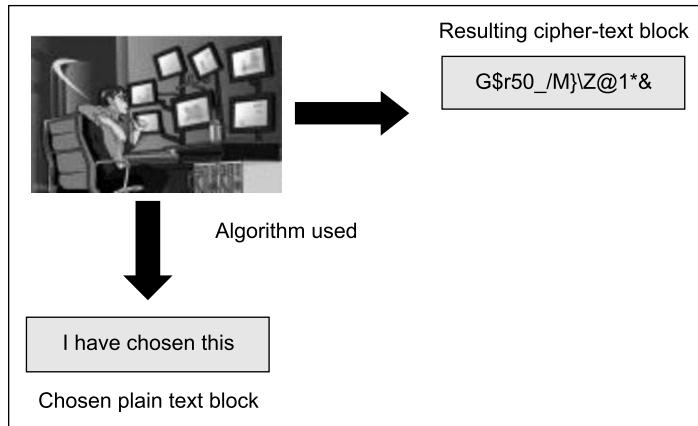


Fig. 2.72 Chosen plain-text attack

4. Chosen Cipher-Text Attack

In the **chosen cipher-text attack**, the attacker knows the cipher text to be decrypted, the encryption algorithm that was used to produce this cipher text, and the corresponding plain-text block. The attacker's job is to discover the key used for encryption. However, this type of attack is not very commonly used.

5. Chosen-Text Attack

The **chosen-text attack** is essentially a combination of *chosen plain-text attack* and *chosen cipher-text attack*. This is shown in Fig. 2.73.

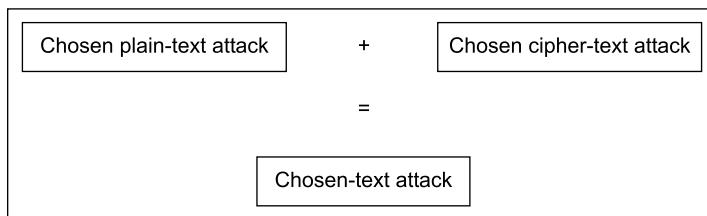


Fig. 2.73 Chosen text attack

Figure 2.74 summarizes the characteristics of these attacks.

Attack	Things known to the attacker	Things the attacker wants to find out
Cipher text only	<ul style="list-style-type: none"> • Cipher text of several messages, all of which are encrypted with the same encryption key • Algorithm used 	<ul style="list-style-type: none"> • Plain-text messages corresponding to these cipher text messages • Key used for encryption
Known cipher text	<ul style="list-style-type: none"> • Cipher text of several messages, all of which are encrypted with the same encryption key • Plain-text messages corresponding to the above cipher text messages • Algorithm used 	<ul style="list-style-type: none"> • Key used for encryption • Algorithm to decrypt cipher text with the same key
Chosen plain text	<ul style="list-style-type: none"> • Cipher text and associated plain-text messages • Chooses the plain-text to be encrypted 	<ul style="list-style-type: none"> • Key used for encryption • Algorithm to decrypt cipher text with the same key
Chosen cipher text	<ul style="list-style-type: none"> • Cipher text of several messages to be decrypted • Corresponding plain-text messages 	<ul style="list-style-type: none"> • Key used for encryption
Chosen text	<ul style="list-style-type: none"> • Some of the above 	<ul style="list-style-type: none"> • Some of the above

Fig. 2.74 Summary of types of attacks

■ CASE STUDY: DENIAL OF SERVICE (DOS) ATTACKS ■

Points for Classroom Discussions

1. *What is the Denial Of Service (DOS) attack?*
2. *Which of the DOS attacks – the one launched using TCP protocol or the one launched using the UDP protocol, more severe? Why?*
3. *What is the meaning of the term ‘service’ in DOS?*
4. *What can possibly prevent DOS attacks?*

The **Denial Of Service (DOS)** attack has gained a lot of attention in the last few years. Most notable DOS attacks were launched against famous Web sites such as Yahoo, Amazon, etc. In April 2000, a 15-year old Canadian called Mafiaboy launched DOS attacks, which caused a lot of concern regarding the security mechanisms of Web sites.

The basic purpose of a DOS attack is simply to flood/overhaul a network so as to deny the authentic users services of the network. A DOS attack can be launched in many ways. The end result is the flooding of a network or change in the configurations of routers on the network.

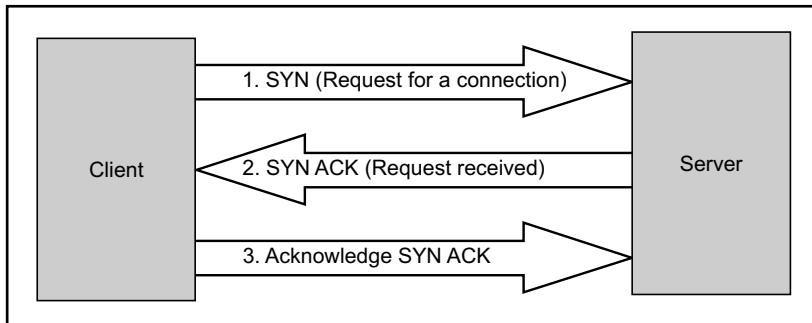
The reason it is not easy to detect a DOS attack is because there is nothing apparent to suggest that a user is launching a DOS attack and is actually not a legitimate user of the system. This is because in a DOS attack, the attacker simply goes on sending a flood of packets to the server/network being attacked. It is up to the server to detect that certain packets are from an attacker and not from a legitimate user and take an appropriate action. This is not an easy task. Failing this, the server would fall short of resources (memory, network connections, etc. and come to a grinding halt after a while).

A typical mechanism to launch a DOS attack is with the help of the SYN requests. On the Internet, a client and a server communicate using the TCP/IP protocol. This involves the creation of a TCP con-

nection between the client and the server, before they can exchange any data. The sequence of these interactions is as follows:

1. The client sends a SYN request to the server. A SYN (abbreviation of *synchronize*) request indicates to the server that the client is requesting for a TCP connection with it.
2. The server responds back to the client with an acknowledgement, which is technically called as SYN ACK.
3. The client is then expected to acknowledge the server's SYN ACK.

This is shown in the figure.



Only after all the three steps above are completed that a TCP connection between a client and a server is considered as established. At this juncture, they can start exchanging the actual application data.

An attacker interested in launching a DOS attack on a server performs Step 1. The server performs Step 2. However, the attacker does not perform Step 3. This means that the TCP connection is not complete. As a result, the server needs to keep the entry for the connection request from the client as *incomplete* and must wait for a response (i.e. Step 3) from the client. The client (i.e. the attacker) is not at all interested in executing step 3. Instead, she simply keeps quiet. Now, imagine that the client sends many such SYN requests to the same server and does not perform Step 3 in any of the requests. Clearly, a lot of incomplete SYN requests would be pending in the server's memory and if these are too many, the server could come to a halt!

Another mechanism to launch the SYN attack is, in Step 1, the attacker forges the source address. That is, the attacker puts the source address as the address of a non-existing client. Therefore, when the server executes Step 2, the SYN ACK never reaches any client at all, fooling the server!

In a more severe case, the attacker launches a Distributed DOS (DDOS) attack. Here, the attacker sends many SYN requests to the server from many physically different client computers. Thus, even if the server detects a DOS attack, it cannot do much by blocking SYN requests coming from a particular IP address – there are many such requests from a variety of (forged) clients!

Mafiaboy performed an attack, which was quite similar to what we have discussed here. He flooded the Web servers with packets sent one after the other in quick succession. This brought a few sites to halt for more than three hours. To stop these attacks, the site administrators started blocking packets coming from suspected source IP addresses. After a few hours, the attacks seemed to be nullified. What had actually happened was quite different. Mafiaboy had simply stopped his attacks! Perhaps he was too tired or bored!

How does one prevent DOS attacks? There are no clear answers here. The following mechanisms can be tried out:

1. Investigate the incoming packets and look for a particular pattern. If such a pattern emerges, then try blocking incoming packets from the concerned IP addresses. However, this is not easy as we have mentioned and this is even tougher to be performed in real time.
2. Another mechanism is to configure the services offered by a particular application so that it never accepts more than a particular number of requests in a specified time interval. Therefore, even if the attacker keeps sending more and more requests, the server would process them at its own pace and use that time to analyze/detect the attacks and take a corrective action.
3. Blocking a particular IP address, port number or a combination of such factors can also prevent a DOS attack. Of course, doing this in real time is not easy.
4. As a precaution, it is always good to have a backup of the firewall and the servers ready. If the main machine is compromised, it should be quickly brought down and the backup can take its place until a proper cleanup is performed.



Summary

- Cryptography is the technique of transforming plain text into cipher text by encoding plain-text messages.
- Cryptanalysis is the technique of decoding messages from a non-readable format back to readable format without knowing how they were initially converted from readable format to non-readable format.
- The person performing cryptanalysis is called a cryptanalysts.
- Normal language of communication uses plain text or clear text.
- A codified message produces cipher text.
- Cipher text cannot be understood by anyone who does not know the coding scheme used for producing the cipher text.
- Plain text can be transformed into cipher text using substitution or transposition techniques.
- In substitution cipher, the characters of plain text are replaced by other characters or symbols.
- Caesar cipher is the world's first well-known substitution cipher. It replaces each character in the plain-text message with a character that is three places down the line.
- Modified versions of Caesar cipher are available.
- An attack on cipher text wherein the attacker tries all possibilities is called brute-force attack.
- Mono-alphabetic cipher is a modified version of Caesar cipher, and is much harder to crack than Caesar cipher.
- Homophonic substitution cipher is very similar to mono-alphabetic cipher. However, it adds more complexity to it.
- In polygram substitution cipher, one block of text is replaced by another block.
- The Vigenère cipher and the Beaufort cipher are examples of polyalphabetic substitution ciphers.
- The Playfair cipher, also called Playfair square, is a cryptographic technique that is used for manual encryption of data.

- The Hill cipher works on multiple letters at the same time. Hence, it is a type of polygraphic substitution cipher.
- Transposition techniques involve permutations and combinations over the plain text to produce cipher text.
- In rail-fence technique, the plain text contents are written as sequence of diagonals, and are read as a sequence of rows.
- In simple columnar transposition technique, the plain-text contents are written as rows and read as columns, but not necessarily in the same order. It can also have multiple rounds.
- The Vernam cipher (also called one-time pad) uses random cipher text every time.
- Book cipher or running-key cipher uses some text from a book to produce cipher text.
- The process of encoding plain-text messages into cipher text is called encryption.
- The process of decoding cipher-text messages back into plain text is called decryption.
- Cryptography can be based on a single key (symmetric) or two keys (asymmetric).
- Attackers can launch attacks by using one of the following techniques: cipher text only, known plain text, chosen plain text, chosen cipher text, and chosen text.
- In the cipher-text only attack, the attacker does not have any clue about the plain text. He/she has some or all of the cipher text. The attacker analyzes the cipher text at leisure to try and figure out the original plain text.
- In the known-plain-text attack, the attacker knows about some pairs of plain text and corresponding cipher text for those pairs. Using this information, the attacker tries to find other pairs, and therefore, know more and more of the plain text.
- In the chosen-plain-text attack, the attacker selects a plain-text block, and tries to look for the encryption of the same in the cipher text. Here, the attacker is able to choose the messages to encrypt. Based on this, the attacker intentionally picks patterns of cipher text that result in obtaining more information about the key.
- In the chosen-cipher-text attack, the attacker knows the cipher text to be decrypted, the encryption algorithm that was used to produce this cipher text, and the corresponding plain-text block.
- The chosen-text attack is essentially a combination of *chosen-plain-text attack* and *chosen cipher text attack*.



Key Terms and Concepts

- Asymmetric key cryptography
- Brute-force attack
- Caesar cipher
- Chosen-plain-text attack
- Cipher text
- Clear text
- Cryptanalyst
- Book cipher
- Bucket brigade attack
- Chosen-cipher-text attack
- Chosen-text attack
- Cipher-text-Only attack
- Cryptanalysis
- Cryptography

- Cryptology
- Decryption algorithm
- Encryption algorithm
- Homophonic substitution cipher
- Known-plain-text attack
- Mono-alphabetic cipher
- Playfair cipher
- Polygram substitution cipher
- Running key cipher
- Simple columnar transposition technique with multiple rounds
- Symmetric key cryptography
- Vernam cipher
- Decryption
- Encryption
- Hill cipher
- Key
- Man-in-the-middle attack
- One-time pad
- Plain text
- Rail-fence technique
- Simple columnar transposition technique
- Substitution cipher
- Transposition cipher
- Vigenère cipher



PRACTICE SET

■ Multiple-Choice Questions

1. The language that we commonly use can be termed
 - (a) pure text
 - (b) simple text
 - (c) plain text
 - (d) normal text
2. The codified language can be termed
 - (a) clear text
 - (b) unclear text
 - (c) code text
 - (d) cipher text
3. In substitution cipher, the following happens.
 - (a) Characters are replaced by other characters.
 - (b) Rows are replaced by columns.
 - (c) Columns are replaced by rows.
 - (d) None of the above.
4. Transposition cipher involves
 - (a) replacement of blocks of text with other blocks
 - (b) replacement of characters of text with other characters
 - (c) strictly row-to-column replacement
 - (d) some permutation on the input text to produce cipher text
5. Caesar cipher is an example of
 - (a) substitution cipher
 - (b) transposition cipher
 - (c) substitution as well as transposition cipher
 - (d) none of the above
6. Vernam cipher is an example of
 - (a) substitution cipher
 - (b) transposition cipher
 - (c) substitution as well as transposition cipher
 - (d) none of the above
7. A cryptanalyst is a person who
 - (a) devises cryptography solutions
 - (b) attempts to break cryptography solutions

- (c) none of these
 (d) both of these
8. Homophonic substitution cipher is _____ to break as compared to mono-alphabetic cipher.
 (a) easier
 (b) the same
 (c) difficult
 (d) easier or same
9. The process of writing the text as diagonals and reading it as a sequence of rows is called
 (a) rail-fence technique
 (b) Caesar cipher
 (c) Mono-alphabetic cipher
 (d) Homophonic substitution cipher
10. The mechanism of writing text as rows and reading as columns is called as
 (a) Vernam cipher
 (b) Caesar cipher
 (c) Simple columnar transposition technique
 (d) Homophonic substitution cipher
11. Vernam cipher is also called
 (a) rail-fence technique
 (b) one-time pad
- (c) book cipher
 (d) running-key cipher
12. Book cipher is also called
 (a) rail-fence technique
 (b) one-time pad
 (c) mono-alphabetic cipher
 (d) running-key cipher
13. A polyalphabetic cipher uses many _____.
 (a) keys
 (b) transpositions
 (c) codes
 (d) monoalphabetic substitution rules
14. The matrix theory is used in the _____ technique.
 (a) Hill cipher
 (b) Monoalphabetic cipher
 (c) Playfair cipher
 (d) Vigenère Cipher
15. If the number of parties involved in a lock-key mechanism is 4, the number of keys needed is
 (a) 2
 (b) 4
 (c) 6
 (d) 8

■ Exercises

- What is plain text? What is cipher text? Give an example of transformation of plain text into cipher text.
- What are the two basic ways of transforming plain text into cipher text?
- What is the difference between substitution cipher and transposition cipher?
- Discuss the concept of Caesar cipher. What is the output of the following plain text: 'Hello there. My name is Atul.', if we use Caesar cipher to encode it?
- How can Caesar cipher be cracked?
- What is mono-alphabetic cipher? How is it different from Caesar cipher? Why is mono-alalphabetic cipher difficult to crack?
- Discuss homophonic substitution cipher with reference to mono-alphabetic cipher.
- What is the main feature of polygram substitution cipher?
- Discuss the algorithm for rail-fence technique. Assume a plain-text security is important, and generate the corresponding cipher text using rail-fence technique.
- How does simple columnar transposition technique work? Assume the same plain text (security is important) and generate the corresponding cipher text using this technique.
- What is the principle behind one-time pads? Why are they highly secure?
- How is book cipher different from one-time pad?

13. What is encryption? What is decryption? Draw a block diagram showing plain text, cipher text, encryption and decryption.
14. Distinguish between symmetric and asymmetric key cryptography.
15. Discuss the playfair cipher.

■ Design/Programming Exercises

1. Write a Java program to perform encryption and decryption using the following algorithms:
 - Caesar cipher
 - Rail-fence technique
 - Simple transposition technique
2. Alice meets Bob and says *Rjjy rj ts ymj xfggfym. bj bnqq inxhzxx ymj uqfs*. If she is using Caesar cipher, what does she want to convey?
3. What would be the transformation of a message ‘Happy birthday to you’ using rail-fence technique?
4. The following message was received by Bob: *hs yis ls. efistof n^TyymrieraseMr e ho ec^etose Dole^*. If the message is encrypted by using the simple transposition method, with the key as 24153, find the original plain text.
5. During the World War II, a German spy used a technique known as *null cipher*. Using this technique, the actual message is created from the first alphabet of each word in the message that is actually transmitted. Find out the hidden secret message if the transmitted message was *President's embargo ruling should have immediate notice. Grave situation affecting international law, statement foreshadows ruin of many neutrals. Yellow journals unifying national excitement immensely*.
6. Consider a scheme involving the replacement of alphabets as follows:

Original	A	B	C	...	X	Y	Z
Changed to	Z	Y	X	...	C	B	A

 If Alice sends a message *HSLDNVGSVNLMB*, what should Bob infer from this?
7. Diffie-Hellman Exercises:
 - (a) Alice and Bob want to establish a secret key using the Diffie-Hellman Key Exchange protocol. Assuming the values as $n = 11$, $g = 5$, $x = 2$ and $y = 3$, find out the values of A , B and the secret key (K_1 or K_2).
 - (b) Next time, they choose $n = 10$, $g = 3$, $x = 5$ and $y = 11$. Find out the values of A , B , K_1 and K_2 .
8. Encrypt the following message using mono-alphabetic substitution cipher with key = 4.
This is a book on Security
9. Decrypt the following message using mono-alphabetic substitution cipher with key = 4.
wigyyvmxc rixiv gsqiw jsv jvii
10. Encrypt the following plain-text bit pattern with the supplied key, using the XOR operation, and state the resulting cipher-text bit pattern.

Plain text	10011110100101010
Key	0100010111101101

11. Transform the cipher text generated in the above exercise back to the original plain text.

12. Consider a plain text message I AM A HACKER. Encrypt it with the help of the following algorithm:
 - (a) Replace each alphabet with its equivalent 7-bit ASCII code.
 - (b) Add a 0 bit as the leftmost bit to make each of the above bit patterns 8 positions long.
 - (c) Swap the first four bits with the last four bits for each alphabet.
 - (d) Write the hexadecimal equivalent of every four bits.
13. Write a C program to perform the task of the above exercise.
14. Implement the playfair cipher in the Java programming language.
Implement the Diffie–Hellman key-exchange mechanism using HTML and JavaScript. Consider the end user as one of the parties (*Alice*), and the JavaScript application as the other party (*Bob*).



COMPUTER-BASED SYMMETRIC-KEY CRYPTOGRAPHIC ALGORITHMS

■ 3.1 INTRODUCTION ■

We have discussed the basic concepts of cryptography, encryption and decryption. Computers perform encryption quite easily and fast. Symmetric-key cryptography has been quite popular over a number of years. Of late, it is losing the edge to asymmetric key cryptography. However, as we shall study later, most practical implementations use a combination of symmetric- and asymmetric-key cryptography.

In this chapter, we shall look at the various aspects of symmetric-key cryptography. We shall see the various types and modes of symmetric-key algorithms. We shall also examine why chaining is useful.

This chapter also discusses a number of symmetric-key algorithms in great detail. We go to the last bit of explanation, which should make the reader understand how these algorithms work. The algorithms discussed here are DES (and its variations), IDEA, RC4, RC5 and Blowfish. We also touch upon the Rijndael algorithm, which is approved by the US Government as the Advanced Encryption Standard (AES).

After reading this chapter, the reader should have a complete understanding of how symmetric-key encryption works. However, if the reader is not very keen in understanding the minute details of the algorithms, the relevant sections can be skipped without any loss of continuity.

■ 3.2 ALGORITHM TYPES AND MODES ■

Before we discuss real-life computer-based cryptography algorithms, let us discuss two key aspects of such algorithms: **algorithm types** and **algorithm modes**. An algorithm type defines what size of plain text should be encrypted in each step of the algorithm. The algorithm mode defines the details of the cryptographic algorithm, once the type is decided.

3.2.1 Algorithm Types

We have been talking about the transformation of plain-text messages into cipher-text messages. We have also seen the various methods for performing these transformations earlier. Regardless of the techniques used at a broad level, the generation of cipher text from plain text itself can be done in two basic ways, **stream ciphers** and **block ciphers**. This is shown in Fig. 3.1.

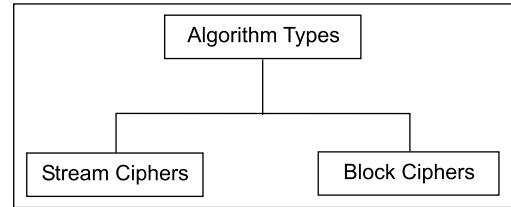


Fig. 3.1 Types of ciphers

1. Stream Ciphers

In **stream ciphers**, the plain text is encrypted one bit at a time. Suppose the original message (plain text) is *Pay 100* in ASCII (i.e. text format). When we convert these ASCII characters to their binary values, let us assume that it translates to 01011100 (hypothetically, just for simplicity; in reality, the binary text would be much larger as each text character takes seven bits). Let us also assume that we apply the *XOR* logic as the encryption algorithm. XOR is quite simple to understand as shown in Fig. 3.2. In simple terms, XOR produces an output of 1, only if one input is 0 and the other is 1. The output is 0 if both the inputs are 0 or if both the inputs are 1 (hence the name *exclusive*).

We can see the effect of XOR in Fig. 3.3.

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3.2 Functioning of XOR logic

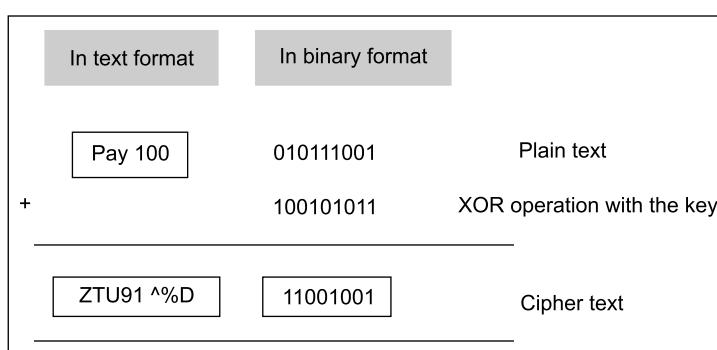


Fig. 3.3 Stream cipher

As a result of applying one bit of key for every respective bit of the original message, suppose the cipher text is generated as 11001001 in binary (ZTU91 ^% in text). Note that each bit of the plain text is encrypted one after the other. Thus, what is transmitted is 11001001 in binary, which even when translated back to ASCII would mean ZTU91 ^%. This makes no sense to an attacker, and thus protects the information.

Stream-cipher technique involves the encryption of one plain-text bit at a time. The decryption also happens one bit at a time.

Another interesting property of XOR is that when used twice, it produces the original data. For example, suppose we have two binary numbers $A = 101$ and $B = 110$. We now want to perform an XOR operation on A and B to produce a third number C , i.e.

$$C = A \text{ XOR } B$$

Thus, we will have

$$C = 101 \text{ XOR } 110 = 011$$

Now, if we perform C XOR A , we will get B . That is,

$$B = 011 \text{ XOR } 101 = 110$$

Similarly, if we perform C XOR B , we will get A . That is,

$$A = 011 \text{ XOR } 110 = 101$$

This reversibility of XOR operations has many implications in cryptographic algorithms, as we shall notice.

XOR is reversible—when used twice, it produces the original values. This is useful in cryptography.

2. Block Ciphers

In **block ciphers**, rather than encrypting one bit at a time, a block of bits is encrypted at one go. Suppose we have a plain text *FOUR_AND_FOUR* that needs to be encrypted. Using block cipher, *FOUR* could be encrypted first, followed by *_AND_* and finally *FOUR*. Thus, one block of characters gets encrypted at a time.

During decryption, each block would be translated back to the original form. In actual practice, the communication takes place only in bits. Therefore, *FOUR* actually means the binary equivalent of the ASCII characters *FOUR*. After any algorithm encrypts these, the resultant bits are converted back into their ASCII equivalents. Therefore, we get funny symbols such as *Vfa%*, etc. In actual practice, their binary equivalents are received, which are decrypted back into binary equivalent of ASCII *FOUR*. This is shown in Fig. 3.4.

An obvious problem with block ciphers is repeating text. For repeating text patterns, the same cipher is generated. Therefore, it could give a clue to the cryptanalyst regarding the original plain text. The cryptanalyst can look for repeating strings and try to break them. If he/she succeeds in doing so, there is a danger that a relatively large portion of the original message is broken into, and therefore, the entire message can then be revealed with more effort. Even if the cryptanalyst cannot guess the remaining words, suppose he/she changes all *debit* to *credit* and vice versa in a funds transfer message, it could cause havoc! To deal with this problem, block ciphers are used in **chaining mode**, as we shall study later. In this approach, the previous block of cipher text is mixed with the current block, so as to obscure the cipher text, thus avoiding repeated patterns of blocks with the same contents.

The block-cipher technique involves encryption of one block of text at a time. Decryption also takes one block of encrypted text at a time.

Practically, the blocks used in a block cipher generally contain 64 bits or more. As we have seen, stream ciphers encrypt one bit at a time. This can be very time-consuming and actually unnecessary in real

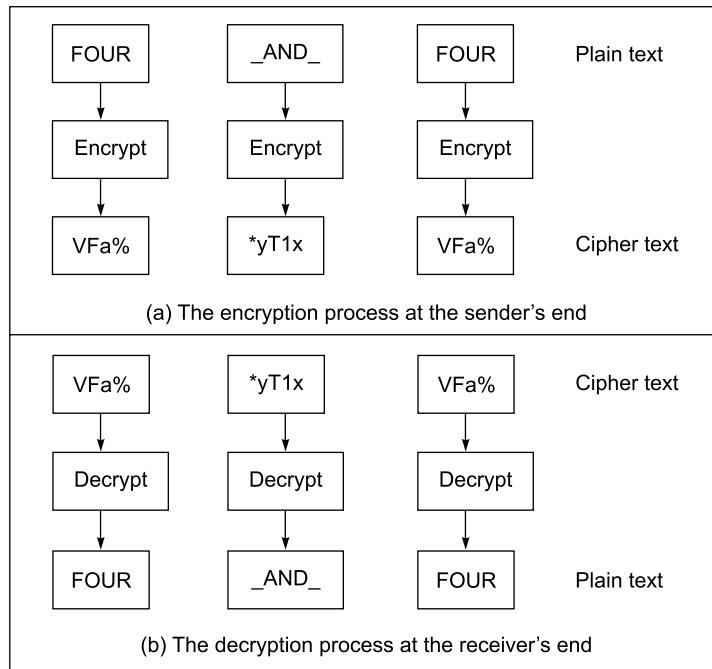


Fig. 3.4 Block cipher

life. That is why block ciphers are used more often in computer-based cryptographic algorithms as compared to stream ciphers. Consequently, we will focus our attention on block ciphers with reference to algorithm modes. However, as we shall see, two of the block-cipher algorithm modes can also be implemented as stream-cipher modes.

3. Group Structures

When discussing an algorithm, many times a question arises, as to whether it is a **group**. The elements of the group are the cipher-text blocks with each possible key. Grouping, thus, means how many times the plain text is scrambled in various ways to generate the cipher text.

4. Concepts of Confusion and Diffusion

Claude Shannon introduced the concepts of **confusion** and **diffusion**, which are significant from the perspective of the computer-based cryptographic techniques.

Confusion is a technique of ensuring that a cipher text gives no clue about the original plain text. This is to try and thwart the attempts of a cryptanalyst to look for patterns in the cipher text, so as to deduce the corresponding plain text. We already know how to achieve confusion: it is achieved by means of the substitution techniques discussed earlier.

Diffusion increases the redundancy of the plain text by spreading it across rows and columns. We have already seen that this can be achieved by using the transposition techniques (also called permutation techniques).

Stream cipher relies only on confusion. Block cipher uses both confusion and diffusion.

3.2.2 Algorithm Modes

An *algorithm mode* is a combination of a series of the basic algorithm steps on block cipher, and some kind of feedback from the previous step. We shall discuss it now, as it forms the basis for the computer-based security algorithms. There are four important algorithm modes, namely **Electronic Code Book (ECB)**, **Cipher Block Chaining (CBC)**, **Cipher Feedback (CFB)**, and **Output Feedback (OFB)**. This is shown in Fig. 3.5. The first two modes operate on block-cipher, whereas the latter two modes are block cipher modes, which can be used as if they are working on stream cipher.

Apart from this, we will also discuss a variation of the OFB mode, called **Counter (CTR)**.

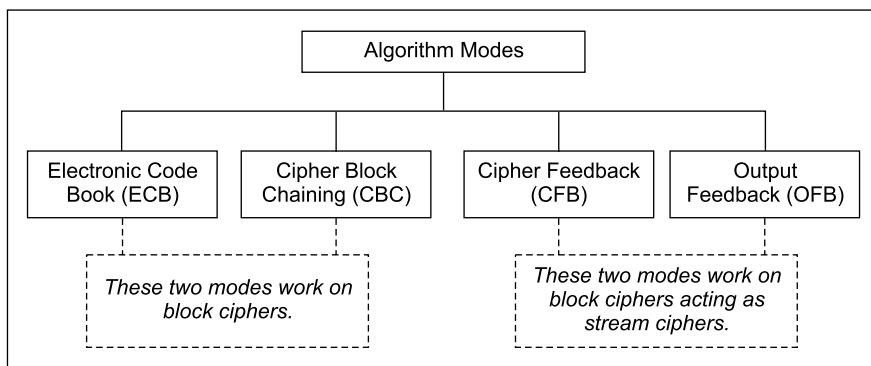


Fig. 3.5 Algorithm modes

We shall discuss the algorithm modes in brief now.

1. Electronic Code Book (ECB) Mode

Electronic Code Book (ECB) is the simplest mode of operation. Here, the incoming plain-text message is divided into blocks of 64 bits each. Each such block is then encrypted independently of the other blocks. For all blocks in a message, the same key is used for encryption. This process is shown in Fig. 3.6.

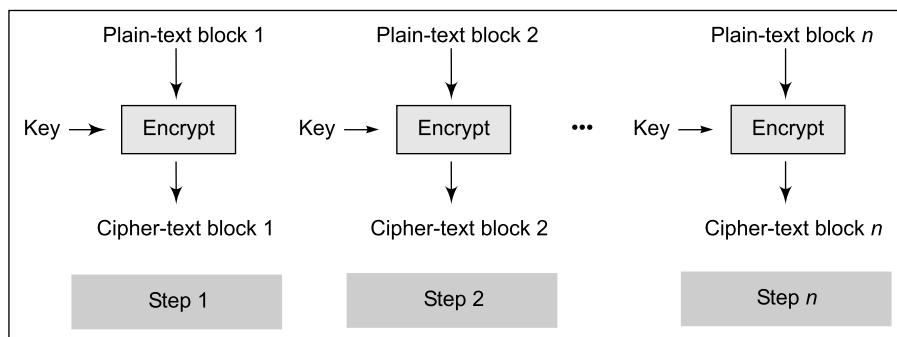


Fig. 3.6 ECB mode—the encryption process

At the receiver's end, the incoming data is divided into 64-bit blocks, and by using the same key as was used for encryption, each block is decrypted to produce the corresponding plain-text block. This process is shown in Fig. 3.7.

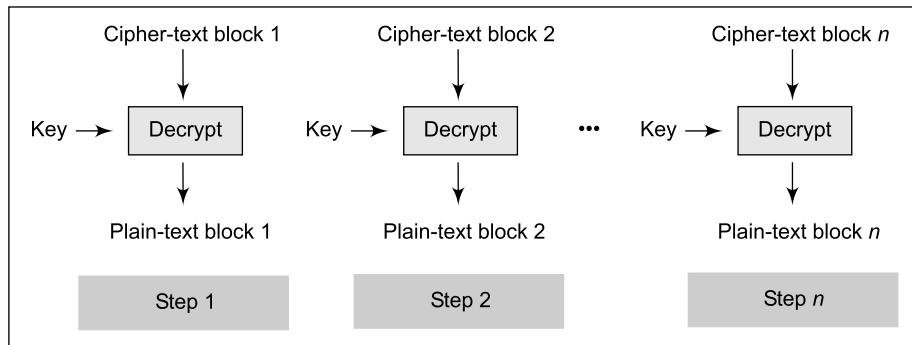


Fig. 3.7 ECB mode—the decryption process

In ECB, since a single key is used for encrypting all the blocks of a message, if a plain-text block repeats in the original message, the corresponding cipher-text block will also repeat in the encrypted message. Therefore, ECB is suitable only for encrypting small messages, where the scope for repeating the same plain-text blocks is quite less.

2. Cipher Block Chaining (CBC) Mode

We saw that in the case of ECB, within a given message (i.e. for a given key), a plain-text block always produces the same cipher-text block. Thus, if a block of plain text occurs more than once in the input, the corresponding cipher text block will also occur more than once in the output, thus providing some clues to a cryptanalyst. To overcome this problem, the **Cipher Block Chaining (CBC)** mode ensures that even if a block of plain text repeats in the input, these two (or more) identical plain-text blocks yield totally different cipher-text blocks in the output. For this, a feedback mechanism is used, as we shall learn now.

Chaining adds a feedback mechanism to a block cipher. In Cipher Block Chaining (CBC), the results of the encryption of the previous block are fed back into the encryption of the current block. That is, each block is used to modify the encryption of the next block. Thus, each block of cipher text is dependent on the corresponding current input plain-text block, as well as all the previous plain-text blocks.

The encryption process of CBC is depicted in Fig. 3.8 and described thereafter.

1. As shown in the figure, the first step receives two inputs: the first block of plain text and a random block of text, called **Initialization Vector (IV)**.
 - (a) The IV has no special meaning: it is simply used to make each message unique. Since the value of IV is randomly generated, the likelihood of it repeating in two different messages is quite rare. Consequently, IV helps in making the cipher-text somewhat unique, or at least quite different from all the other cipher texts in a different message. Interestingly, it is not mandatory to keep IV secret—it can be known to everybody. This seems slightly concerning and confusing. However, if we take a re-look at the operation of CBC, we will realize that IV is simply one of the two inputs to the first encryption step. The output of step 1 is

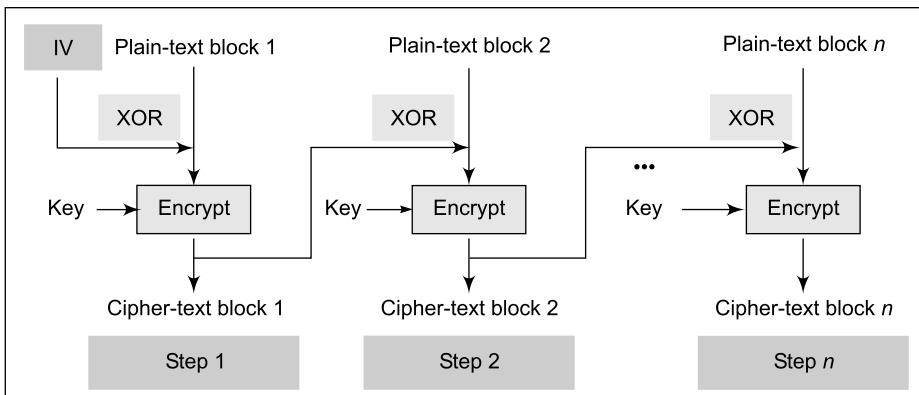


Fig. 3.8 CBC mode—the encryption process

cipher-text block 1, which is also one of the two inputs to the second encryption step. In other words, cipher-text block 1 is also an IV for step 2! Similarly, cipher-text block 2 is also an IV for step 3, and so on. Since all these cipher-text blocks will be sent to the receiver, we are anyway sending all IVs for step 2 onwards! Thus, there is no special reason why the IV for step 1 should be kept secret. It is the key used for encryption that needs to be kept secret. However, in practice, for maximum security, both the key and the IV are kept secret.

- (b) The first block of cipher text and IV are combined using XOR and then encrypted using a key to produce the first cipher-text block. The first cipher-text block is then provided as a *feedback* to the next plain-text block, as explained below.
2. In the second step, the second plain-text block is XORed with the output of step 1, i.e. the first cipher-text block. It is then encrypted with the same key, as used in step 1. This produces cipher-text block 2.
 3. In the third step, the third plain-text block is XORed with the output of step 2, i.e. the second cipher-text block. It is then encrypted with the same key, as used in step 1.
 4. This process continues for all the remaining plain-text blocks of the original message.

Remember that the IV is used only in the first plain text block. However, the same key is used for the encryption of all plain text blocks.

The decryption process works as follows.

1. The cipher-text block 1 is passed through the decryption algorithm using the same key, which was used during the encryption process for all the plain-text blocks. The output of this step is then XORed with the IV. This process yields plain-text block 1.
2. In step 2, the cipher-text block 2 is decrypted, and its output is XORed with cipher-text block 1, which yields plain-text block 2.
3. This process continues for all the Cipher text blocks in the encrypted message.

The decryption process is shown in Fig. 3.9.

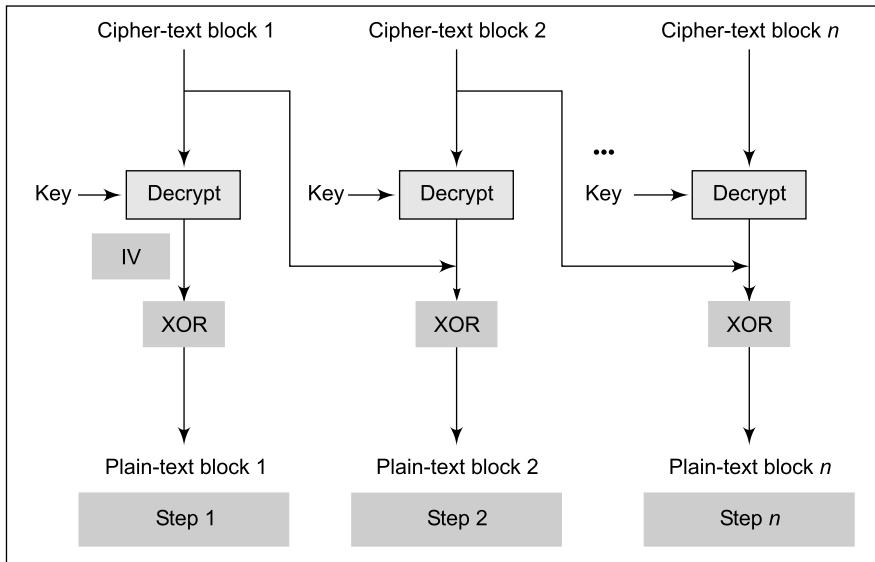


Fig. 3.9 CBC mode—the decryption process

Appendix A provides some more mathematical details about the CBC mode.

3. Cipher Feedback (CFB) Mode

Not all applications can work with blocks of data. Security is also required in applications that are character-oriented. For instance, an operator can be typing keystrokes at a terminal, which needs to be immediately transmitted across the communications link in a secure manner, i.e. by using encryption. In such situations, stream cipher must be used. The **Cipher Feedback (CFB)** mode is useful in such cases. In this mode, data is encrypted in units that are smaller (e.g. they could be of size 8 bits, i.e. the size of a character typed by an operator) than a defined block size (which is usually 64 bits).

Let us understand how CFB mode works, assuming that we are dealing with j bits at a time (as we have seen usually, but not always, $j = 8$). Since CFB is slightly more complicated as compared to the first two cryptography modes, we shall study CFB in a step-by-step fashion.

Step 1 Like CBC, a 64-bit Initialization Vector (IV) is used in the case of CFB mode. The IV is kept in a shift register. It is encrypted in the first step to produce a corresponding 64-bit IV cipher text. This is shown in Fig. 3.10.

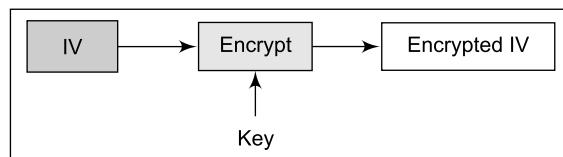


Fig. 3.10 CFB—Step 1

Step 2 Now, the leftmost (i.e. the most significant) j bits of the encrypted IV are XORed with the first j bits of the plain text. This produces the first portion of cipher text (say C) as shown in Fig. 3.11. C is then transmitted to the receiver.

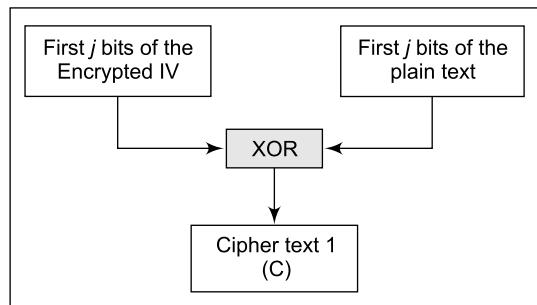


Fig. 3.11 CFB—Step 2

Step 3 Now, the bits of IV (i.e. the contents of the shift register containing IV) are shifted left by j positions. Thus, the rightmost j positions of the shift register now contain unpredictable data. These rightmost j positions are now filled with C . This is shown in Fig. 3.12.

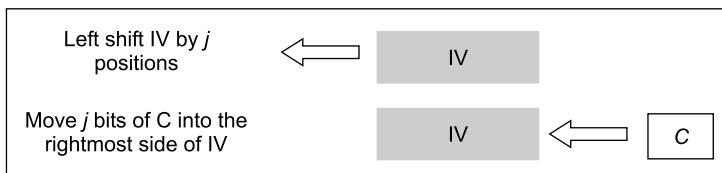


Fig. 3.12 CFB—Step 3

Step 4 Now, steps 1 through 3 continue until all the plain-text units are encrypted. That is, the following steps are repeated:

- IV is encrypted.
- The leftmost j bits resulting from this encryption process are XORed with the next j bits of the plain text.
- The resulting cipher-text portion (i.e. the next j bits of cipher text) is sent to the receiver.
- The shift register containing the IV is left-shifted by j bits.
- The j bits of the cipher text are inserted from right into the shift register containing the IV.

Figure 3.13 shows the overall conceptual view of the CFB mode.

At the receiver's end, the decryption process is pretty similar, with minor changes. We shall not discuss it.

4. Output Feedback (OFB) Mode

The **Output Feedback (OFB)** mode is extremely similar to the CFB. The only difference is that in the case of CFB, the cipher text is fed into the next stage of encryption process. But in the case of OFB,

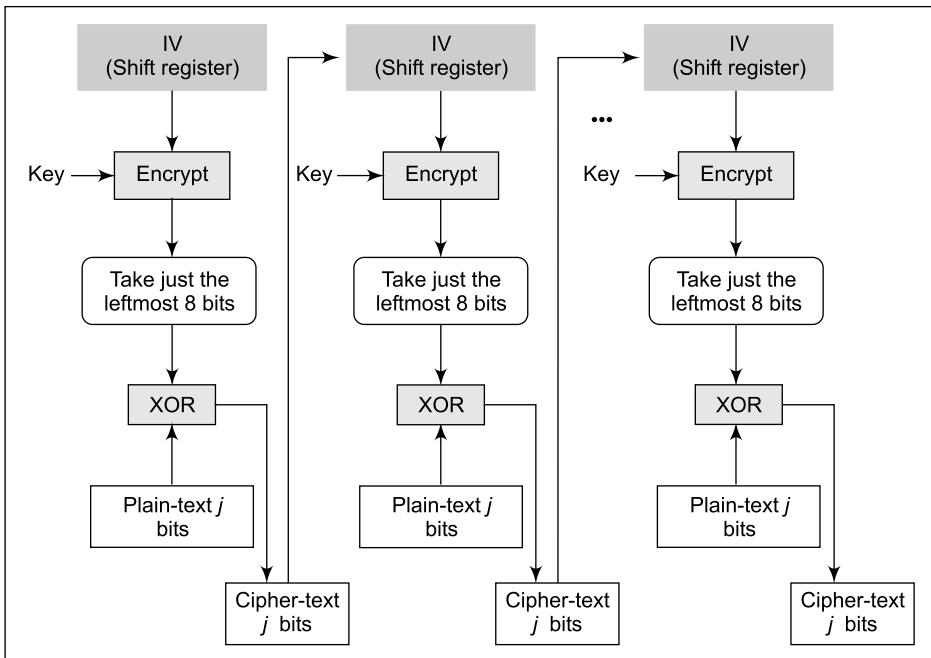


Fig. 3.13 CFB—the overall encryption process

the output of the IV encryption process is fed into the next stage of encryption process. Therefore, we shall not describe the details of OFB, and instead, shall simply draw the block diagram of the OFB process, as shown in Fig. 3.14. The same details as discussed in CFB apply here, except the change, as pointed out above.

Let us summarize the key advantage of the OFB mode. In simple terms, we can state that in this mode, if there are errors in individual bits, they remain errors in individual bits and do not corrupt the whole message. That is, bit errors do not get propagated. If a cipher-text bit C_i is in error, only the decrypted value corresponding to this bit, i.e. P_i is wrong. Other bits are not affected. Remember that in contrast to this, in the CFB mode, the cipher text bit C_i is fed back as input to the shift register, and would corrupt the other bits in the message!

The possible drawback with OFB is that an attacker can make necessary changes to the cipher text and the checksum of the message in a controlled fashion. This causes changes in the cipher text without it getting detected. In other words, the attacker changes both the cipher text and the checksum at the same time, hence there is no way to detect this change.

5. Counter (CTR) Mode

The **Counter (CTR)** mode is quite similar to the OFB mode, with one variation. It uses *sequence numbers* called *counters* as the inputs to the algorithm. After each block is encrypted, to fill the register, the next counter value is used. Usually, a constant is used as the initial counter value, and is incremented (usually by 1) for every iteration. The size of the counter block is the same as that of the plain-text block.

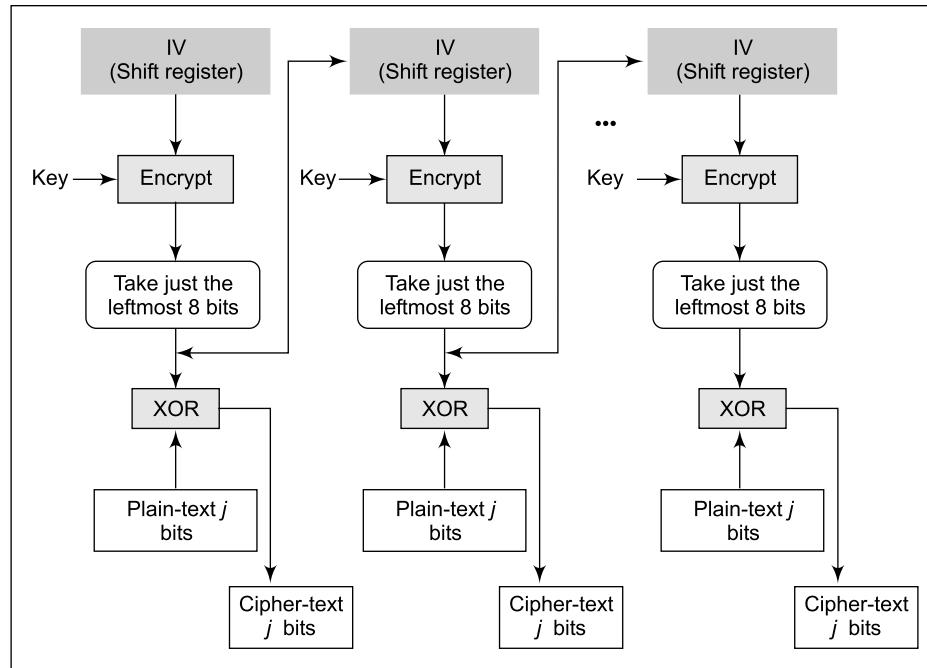


Fig. 3.14 OFB—the overall encryption process

For encryption, the counter is encrypted and then XORed with the plain text block to get the cipher text. No chaining process is used. On the other hand, for decryption, the same sequence of counters is used. Here, each encrypted counter is XORed with the corresponding cipher-text block to obtain the original plain-text block.

The overall operation of the *counter mode* is shown in Fig. 3.15 and Fig. 3.16.

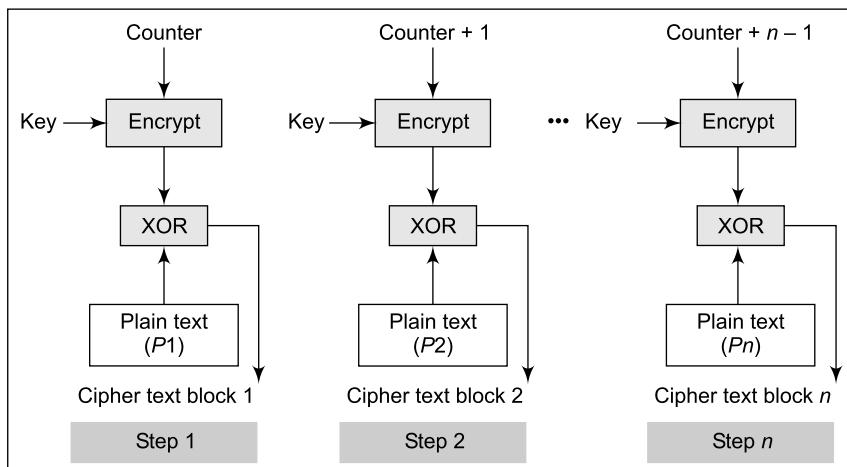


Fig. 3.15 Counter (CTR) mode: the encryption process

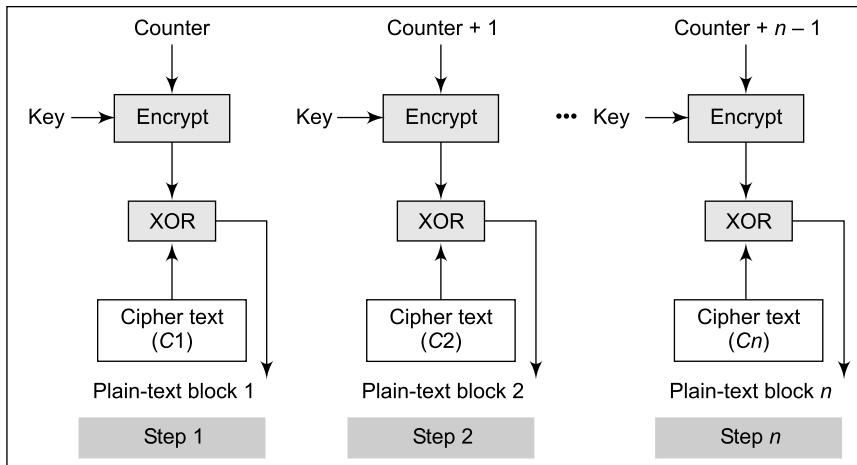


Fig. 3.16 Counter (CTR) mode: the decryption process

The encryption or decryption process in the counter can be done in parallel on multiple text blocks, since no chaining is involved. This can make the execution speed of the counter faster. Multiprocessing systems can take advantage of this feature to introduce features that help reduce the overall processing time. Pre-processing can be achieved to prepare the output of the encryption boxes that input to the XOR operations. The counter mode mandates implementation of the encryption process only, and not of the decryption process.

Table 3.1 Summarizes the key features of the various algorithm modes.

Table 3.1 Algorithm modes: details and usage

Algorithm mode	Details	Usage
Electronic Code Book (ECB)	The same key independently encrypts blocks of text, 64 bits at a time.	Transmitting a single value in a secure fashion (e.g. password or key used for encryption).
Cipher Block Chaining (CBC)	64 bits of cipher text from the previous step and 64 bits of plain text of the next step are XORed together.	Encrypting blocks of text Authentication.
Cipher Feedback (CFB)	K bits of randomized cipher text from the previous step and K bits of plain text of the next step are XORed together.	Transmitting encrypted stream of data Authentication.
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption step is the preceding DES output.	Transmitting encrypted stream of data.
Counter (CTR)	A counter and plain-text block are encrypted together, after which the counter is incremented.	Block-oriented transmissions Applications needing high speed.

Table 3.2 summarizes the key advantages and disadvantages of the various modes.

Table 3.2 Algorithm modes: advantages and problems

Feature	ECB	CBC	CFB	OFB/Counter
Security-related problems	Plain text patterns are not hidden. Input to the block cipher is the same as the plain text, and is not randomized. Plain text is easy to manipulate, blocks of text can be removed, repeated, or exchanged.	Plain-text blocks can be removed from the beginning and end of the message, and bits of the first block can be altered.	Plain-text blocks can be removed from the beginning and end of the message, and bits of the first block can be altered.	Plain text is easy to manipulate. Altering cipher text alters plain text directly.
Security-related advantages	The same key can be used for encrypting multiple messages.	XOR of plain text with previous cipher-text block hides the plain text. The same key can be used for encrypting multiple messages.	Plain-text patterns are hidden. The same key can be used for encrypting multiple messages, by using a different IV. Input to the block cipher is randomized.	Plain-text patterns are hidden. The same key can be used for encrypting multiple messages, by using a different IV. Input to the block cipher is randomized.
Problems related to effectiveness	Size of cipher text is more than the plain-text size by one padding block. Pre-processing is not possible.	Size of cipher text is more than the plain text size by one block. Pre-processing is not possible. Parallelism cannot be introduced in encryption.	Size of cipher text is the same as that of the plain-text size. Parallelism cannot be introduced in encryption.	Size of cipher text is the same as that of the plain-text size. Parallelism cannot be introduced (OFB only).

■ 3.3 AN OVERVIEW OF SYMMETRIC-KEY CRYPTOGRAPHY ■

Let us briefly review symmetric-key cryptography. Symmetric-key cryptography is referred to by various other terms, such as **secret-key cryptography** or **private-key cryptography**. In this scheme, only one key is used and the same key is used for both encryption and decryption of messages. Obviously, both the parties must agree upon the key before any transmission begins, and nobody else should know

about it. The example in Figure 3.17 shows how symmetric-key cryptography works. Basically, at the sender's end (*A*), the key transforms the plain-text message into a cipher-text form. At the receiver's end (*B*), the same key is used to decrypt the encrypted message, thus deriving the original message out of it.

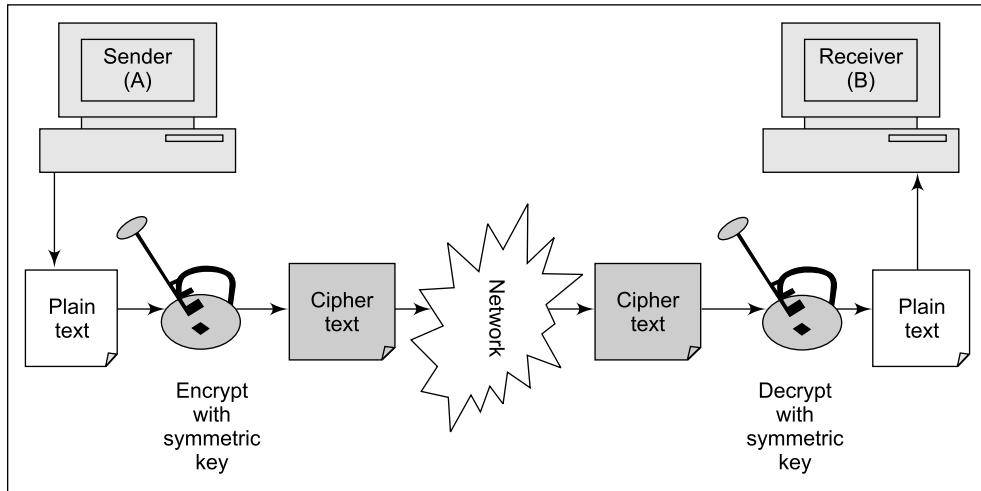


Fig. 3.17 Symmetric-key cryptography

As we have discussed, in practical situations, symmetric-key cryptography has a few problems. We shall revise them quickly now.

The first problem here is that of key agreement or key distribution. The problem is: in the first place, how do two parties agree on a key? One solution is that somebody from the sender's end physically visits the receiver and hands over the key. Another way is to courier a paper on which the key is written. Both are not exactly very convenient. A third way is to send the key over the network to *B* and ask for confirmation. But then, if an intruder gets the message, he/she can interpret all the subsequent ones!

The second problem is more serious. Since the same key is used for encryption and decryption, one key per communicating parties is required. Suppose *A* wants to securely communicate with *B* and also with *C*. Clearly, there must be one key for all communications between *A* and *B*; and there must be another, *distinct* key for all communications between *A* and *C*. The same key as used by *A* and *B* cannot be used for communications between *A* and *C*. Otherwise, there is a chance that *C* can interpret messages going between *A* and *B*, or *B* can do the same for messages going between *A* and *C*! Since the Internet has thousands of merchants selling products to hundreds of thousands of buyers, using this scheme would be impractical because every buyer-seller combination would need a separate key!

Regardless, because these drawbacks can be overcome using intelligent solutions as we shall see, and also because symmetric-key cryptography has several advantages as well, it is widely used in practice. However, we shall first discuss the most popular computer-based symmetric-key cryptographic algorithms, and think about solving the problems associated when we discuss asymmetric-key cryptography subsequently.

■ 3.4 DATA ENCRYPTION STANDARD (DES) ■

3.4.1 Background and History

The **Data Encryption Standard (DES)**, also called the Data Encryption Algorithm (DEA) by ANSI and DEA-1 by ISO, has been a cryptographic algorithm used for over two decades. Of late, DES has been found vulnerable against very powerful attacks, and therefore, the popularity of DES has been slightly on the decline. However, no book on security is complete without DES, as it has been a landmark in cryptographic algorithms. We shall also discuss DES at length to achieve two objectives: Firstly to learn about DES, but secondly and more importantly, to dissect and understand a real-life cryptographic algorithm. Using this philosophy, we shall then discuss some other cryptographic algorithms, but only at a conceptual level; because the in-depth discussion of DES would have already helped us understand in depth, how computer-based cryptographic algorithms work. DES is generally used in the ECB, CBC or the CFB mode.

The origins of DES go back to 1972, when in the US, the National Bureau of Standards (NBS), now known as the National Institute of Standards and Technology (NIST), embarked upon a project for protecting the data in computers and computer communications. They wanted to develop a single cryptographic algorithm. After two years, NBS realized that IBM's **Lucifer** could be considered a serious candidate, rather than developing a fresh algorithm from scratch. After a few discussions, in 1975, the NBS published the details of the algorithm. Towards the end of 1976, the US Federal Government decided to adopt this algorithm, and soon, it was renamed *Data Encryption Standard (DES)*. Soon, other bodies also recognized and adopted DES as a cryptographic algorithm.

3.4.2 How DES Works

1. Basic Principles

DES is a block cipher. It encrypts data in blocks of 64 bits each. That is, 64 bits of plain text goes as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits. The basic idea is shown in Fig. 3.18.

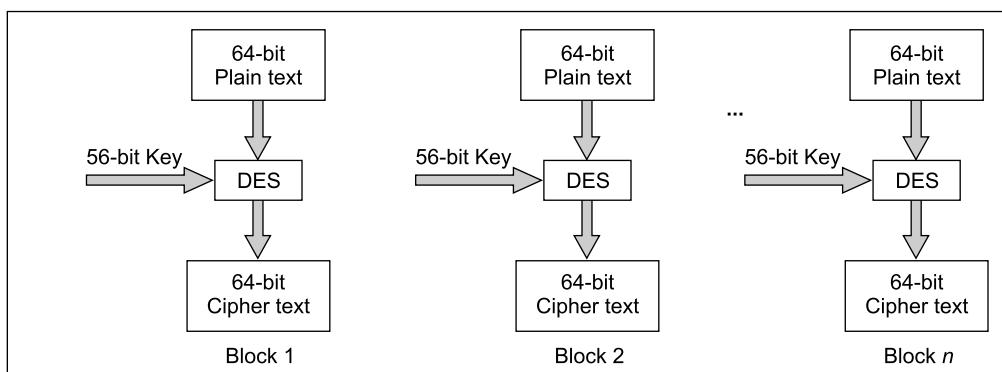


Fig. 3.18 Conceptual working of DES

We have mentioned that DES uses a 56-bit key. Actually, the initial key consists of 64 bits. However, before the DES process even starts, every eighth bit of the key is discarded to produce a 56-bit key. That is, bit positions 8, 16, 24, 32, 40, 48, 56 and 64 are discarded. This is shown in Fig. 3.19 with shaded bit positions indicating discarded bits. (Before discarding, these bits can be used for parity checking to ensure that the key does not contain any errors.)

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

Fig. 3.19 Discarding of every 8th bit of the original key (shaded bit positions are discarded)

Thus, the discarding of every 8th bit of the key produces a 56-bit key from the original 64-bit key, as shown in Fig. 3.20.

Simplistically, DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a **round**. Each *round* performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

1. In the first step, the 64-bit plain-text block is handed over to an **Initial Permutation (IP)** function.
2. The initial permutation is performed on plain text.
3. Next, the Initial Permutation (IP) produces two halves of the permuted block; say Left Plain Text (LPT) and Right Plain Text (RPT).
4. Now, each of LPT and RPT go through 16 *rounds* of encryption process, each with its own key.
5. In the end, LPT and RPT are rejoined, and a **Final Permutation (FP)** is performed on the combined block.
6. The result of this process produces 64-bit cipher text.

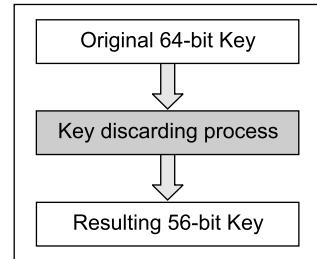


Fig. 3.20 Key-discarding process

This process is shown diagrammatically in Fig. 3.21.

Let us now understand each of these processes in detail.

2. Initial Permutation (IP)

As we have noted, the Initial Permutation (IP) happens only once, and it happens before the first round. It suggests how the transposition in IP should proceed, as shown in Fig. 3.22. For example, it says that the IP replaces the first bit of the original plain-text block with the 58th bit of the original plain-text block, the second bit with the 50th bit of the original plain text block, and so on. This is nothing but jugglery of bit positions of the original plain-text block.

The complete transposition table used by IP is shown in Fig. 3.23. This table (and all others in this chapter) should be read from left to right, top to bottom. For instance, we have noted that 58 in the first position indicates that the contents of the 58th bit in the original plain-text block will overwrite the contents of the 1st bit position, during IP. Similarly, 1 is shown at the 40th position in the table, which

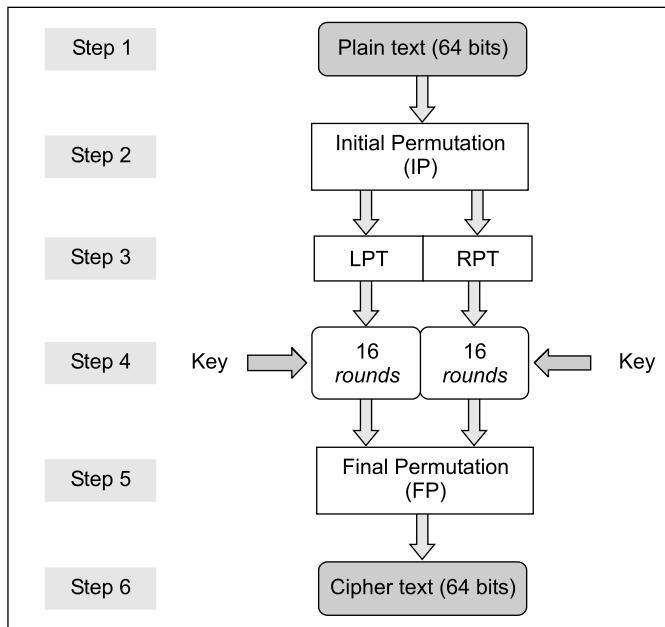


Fig. 3.21 Broad level steps in DES

Bit position in the plain-text block	To be overwritten with the contents of this bit position
1	58
2	50
3	42
..	..
64	7

Fig. 3.22 Idea of IP

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

Fig. 3.23 Initial Permutation (IP) table

means that the first bit will overwrite the 40th bit in the original plain-text block. The same rule applies for all other bit positions.

As we have noted, after IP is done, the resulting 64-bit permuted text block is divided into two half blocks. Each half block consists of 32 bits. We have called the left block as LPT and the right block as RPT. Now, 16 rounds are performed on these two blocks. We shall discuss this process now.

3. Rounds

Each of the 16 rounds, in turn, consists of the broad-level steps outlined in Fig. 3.24.

Let us discuss these details step-by-step.

Step 1: Key Transformation We have noted that the initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each round, a 56-bit key is available. From this 56-bit key, a different 48-bit **subkey** is generated during each *round* using a process called **key transformation**. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round. For example, if the *round* number is 1, 2, 9 or 16, the shift is done by only one position. For other rounds, the circular shift is done by two positions. The number of key bits shifted per round is shown in Fig. 3.25.

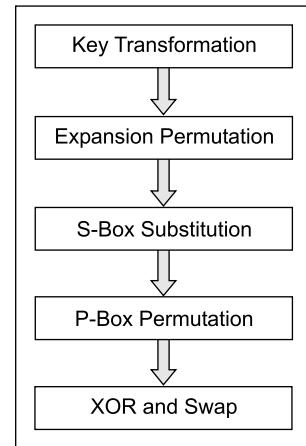


Fig. 3.24 Details of one *round* in DES

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of key bits shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	1	

Fig. 3.25 Number of key bits shifted per *round*

After an appropriate shift, 48 of the 56 bits are selected. For selecting 48 of the 56 bits, the table shown in Fig. 3.26 is used. For instance, after the shift, bit number 14 moves into the first position, bit number 17 moves into the second position, and so on. If we observe the table carefully, we will realize that it contains only 48 bit positions. Bit number 18 is discarded (we will not find it in the table), like 7 others, to reduce the 56-bit key to a 48-bit key. Since the key-transformation process involves permutation as well as selection of a 48-bit subset of the original 56-bit key, it is called **compression permutation**.

14	17	11	24	1	5	3	28	15	6	21	10				
23	19	12	4	26	8	16	7	27	20	13	2				
41	52	31	37	47	55	30	40	51	45	33	48				
44	49	39	56	34	53	46	42	50	36	29	32				

Fig. 3.26 Compression permutation

Because of this compression permutation technique, a different subset of key bits is used in each round. That makes DES more difficult to crack.

Step 2: Expansion Permutation Recall that after initial permutation, we had two 32-bit plain text areas, called Left Plain Text (LPT) and Right Plain Text (RPT). During **expansion permutation**, the RPT is expanded from 32 bits to 48 bits. Besides increasing the bit size from 32 to 48, the bits are permuted as well, hence the name *expansion permutation*. This happens as follows:

1. The 32-bit RPT is divided into 8 blocks, with each block consisting of 4 bits. This is shown in Fig. 3.27.

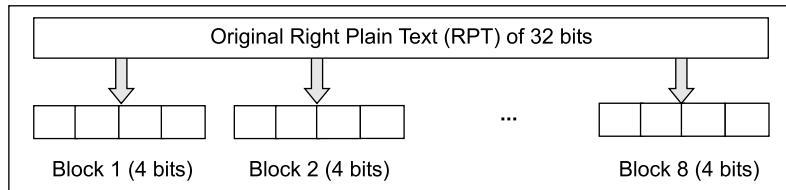


Fig. 3.27 Division of 32-bit RPT into eight 4-bit blocks

2. Next, each 4-bit block of the above step is then expanded to a corresponding 6-bit block. That is, per 4-bit block, 2 more bits are added. What are these two bits? They are actually the repeated first and the fourth bits of the 4-bit block. The second and the third bits are written down as they were in the input. This is shown in Fig. 3.28. Note that the first bit inputted is outputted to the second output position, and also repeats in output position 48. Similarly, the 32nd input bit is found in the 47th output position as well as in the first output position.

Clearly, this process results into expansion as well as permutation of the input bits while creating the output.

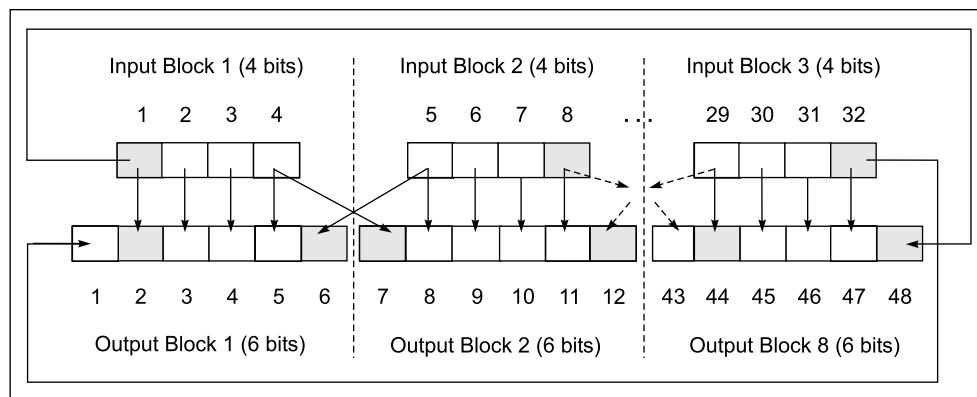


Fig. 3.28 RPT expansion permutation process

As we can see, the first input bit goes into the second and the 48th output positions. The second input bit goes into the third output position, and so on. Consequently, we will observe that the expansion permutation has actually used the table shown in Fig. 3.29.

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

Fig. 3.29 RPT expansion permutation table

As we have seen, firstly, the *key-transformation* process compresses the 56-bit key to 48 bits. Then, the *expansion permutation* process expands the 32-bit RPT to 48 bits. Now, the 48-bit key is XORED

with the 48-bit RPT, and the resulting output is given to the next step, which is the **S-box substitution** (which we shall discuss in the next section) as shown in Fig. 3.30.

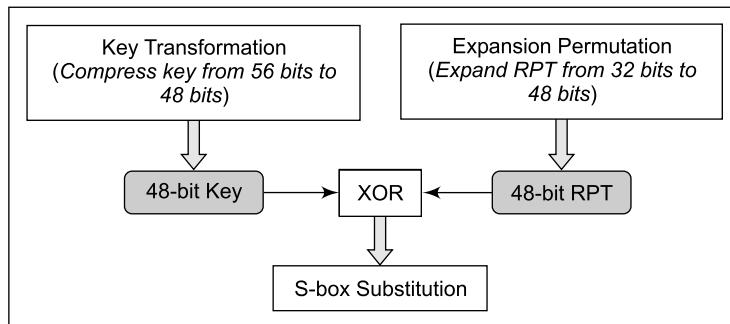


Fig. 3.30 Way to S-box substitution

Step 3: S-box Substitution **S-box substitution** is a process that accepts the 48-bit input from the XOR operation involving the compressed key and expanded RPT, and produces a 32-bit output using the substitution technique. The substitution is performed by eight **substitution boxes** (also called as **S-boxes**). Each of the eight S-boxes has a 6-bit input and a 4-bit output. The 48-bit input block is divided into 8 sub-blocks (each containing 6 bits), and each such sub-block is given to an S-box. The S-box transforms the 6-bit input into a 4-bit output, as shown in Fig. 3.31.

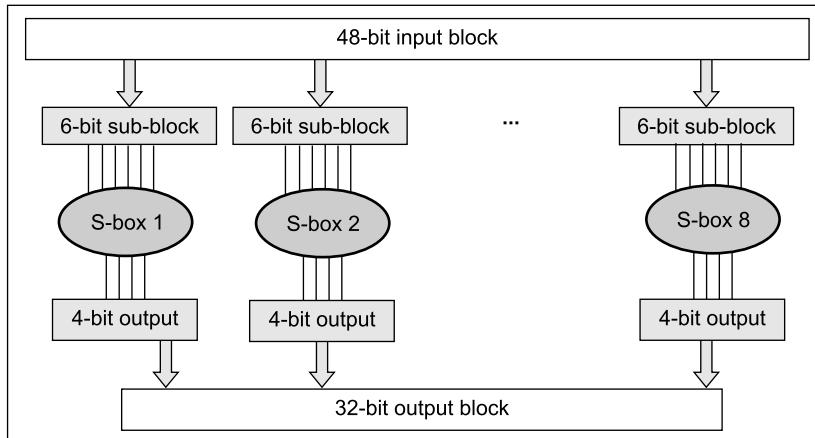


Fig. 3.31 S-box substitution

What is the logic used by S-box substitution for selecting only four of the six bits? We can conceptually think of every S-box as a table that has 4 rows (numbered 0 to 3) and 16 columns (numbered 0 to 15). Thus, we have 8 such tables, one for each S-box. At the intersection of every row and column, a 4-bit number (which will be the 4-bit output for that S-box) is present. This is shown in Fig. 3.32.

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Fig. 3.32(a) S-box 1

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Fig. 3.32(b) S-box 2

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Fig. 3.32(c) S-box 3

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Fig. 3.32(d) S-box 4

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Fig. 3.32(e) S-box 5

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Fig. 3.32(f) S-box 6

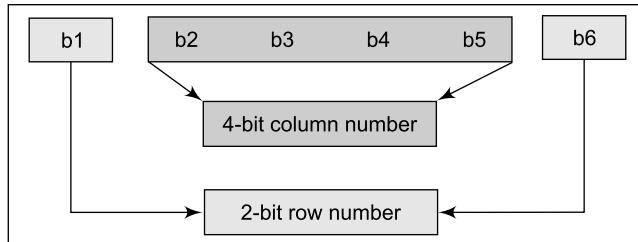
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Fig. 3.32(g) S-box 7

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Fig. 3.32(h) S-box 8

The 6-bit input indicates which row and column, and therefore, which intersection is to be selected, and thus, determines the 4-bit output. How is it done? Let us assume that the six bits of an S-box are indicated by b_1, b_2, b_3, b_4, b_5 , and b_6 . Now, bits b_1 and b_6 are combined to form a two-bit number. Two bits can store any decimal number between 0 (binary 00) and 3 (binary 11). This specifies the row number. The remaining four bits b_2, b_3, b_4, b_5 make up a four-bit number, which specifies the column number between decimal 0 (binary 0000) and 15 (binary 1111). Thus, the 6-bit input automatically selects the row number and column number for the selection of the output. This is shown in Fig. 3.33.

**Fig. 3.33** Selecting an entry in an S-box based on the 6-bit input

Let us take an example now. Suppose the bits 5 to 8 of the 48-bit input (i.e. the input to the second S-box) contain a value 101101 in binary. Therefore, using our earlier diagram, we have $(b_1, b_6) = 11$ in binary (i.e. 3 in decimal), and $(b_2, b_3, b_4, b_5) = 0110$ in binary (i.e. 6 in decimal). Thus, the output of S-box 2 at the intersection of row number 3 and column number 6 will be selected, which is 4. (Remember to count rows and columns from 0, not 1). This is shown in Fig. 3.34.

The output of each S-box is then combined to form a 32-bit block, which is given to the last stage of a *round*, the P-box permutation, as discussed next.

Step 4: P-box Permutation The output of S-box consists of 32 bits. These 32 bits are permuted using a **P-box**. This straightforward permutation mechanism involves simple permutation (i.e. replacement of each bit with another bit, as specified in the P-box table, without any expansion or compression). This is called **P-box permutation**. The P-box is shown in Fig. 3.35. For example, a 16 in the first block indicates that the bit at position 16 of the original input moves to the bit at position 1 in the

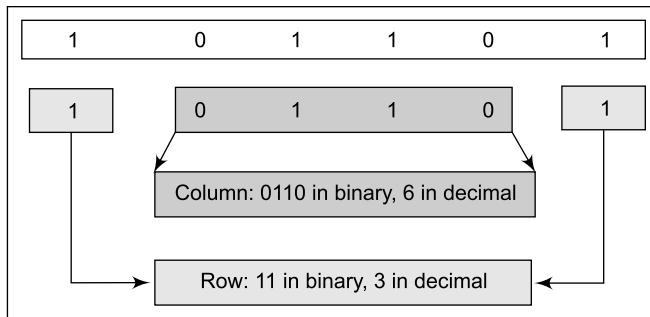


Fig. 3.34 Example of selection of S-box output based on the input

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Fig. 3.35 P-box permutation

output, and a 10 in the block number 16 indicates that the bit at the position 10 of the original input moves to bit at the position 16 in the output.

Step 5: XOR and Swap Note that we have been performing all these operations only on the 32-bit right half portion of the 64-bit original plain text (i.e. on the RPT). The left half portion (i.e. LPT) was untouched so far. At this juncture, the left half portion of the initial 64-bit plain text block (i.e. LPT) is XORed with the output produced by P-box permutation. The result of this XOR operation becomes the new right half (i.e. RPT). The old right half (i.e. RPT) becomes the new left half, in a process of swapping. This is shown in Fig. 3.36.

4. Final Permutation

At the end of the 16 rounds, the **final permutation** is performed (only once). This is a simple transposition based on Fig. 3.37. For instance, the 40th input bit takes the position of the 1st output bit, and so on.

The output of the final permutation is the 64-bit encrypted block.

5. DES Decryption

From the above discussion of DES, we might get a feeling that it is an extremely complicated encryption scheme, and therefore, the decryption using DES would employ a completely different approach. To most people's surprise, the same algorithm used for encryption in DES also works for decryption! The values of the various tables and the operations as well as their sequence are so carefully chosen that the algorithm is reversible. The only difference between the encryption and the decryption process is the reversal of key portions. If the original key K was divided into $K_1, K_2, K_3, \dots, K_{16}$ for the 16 encryption rounds, then for decryption, the key should be used as $K_{16}, K_{15}, K_{14}, \dots, K_1$.

6. Analyzing DES

(a) Use of S-boxes The tables used for substitution, i.e. the S-boxes, in DES are kept secret by IBM. IBM maintains that it took them over 17 person years to come up with the internal design of the

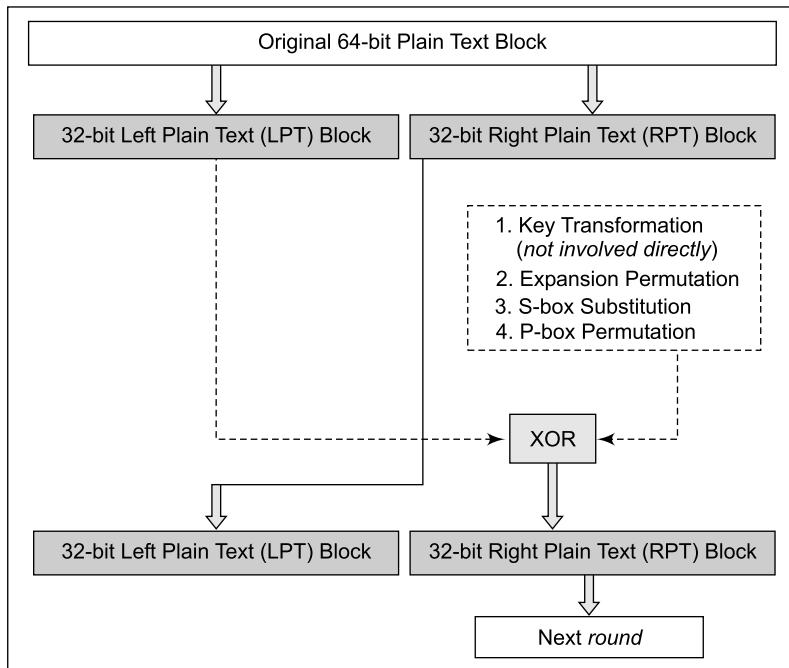


Fig. 3.36 XOR and swap

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

Fig. 3.37 Final permutation

S-boxes. Over the years, suspicion has grown that there is some vulnerability in this aspect of DES, intentional (so that the government agencies could secretly open encrypted messages) or otherwise. Several studies keep appearing, which suggest that there is some scope for attacks on DES via the S-boxes. However, no concrete example has emerged till date.

(b) Key Length We have mentioned earlier that any cryptographic system has two important aspects: the cryptographic algorithm and the key. The inner workings of the DES algorithm (which we have discussed earlier) are completely known to the general public. Therefore, the strength of DES lies only in the other aspect—its key, which must be secret.

As we know, DES uses 56-bit keys. (Interestingly, the original proposal was to make use of 112-bit keys.) Thus, there are 2^{56} possible keys (which is roughly equal to 7.2×10^{16} keys). Thus, it seems that a brute-force attack on DES is impractical. Even if we assume that to obtain the correct key, only half of the possible keys (i.e. the half of the **key space**) needs to be examined and tried out, a single computer performing one DES encryption per microsecond would require more than 1,000 years to break DES.

(c) Differential and Linear Cryptanalysis In 1990, Eli Biham and Adi Shamir introduced the concept of **differential cryptanalysis**. This method looks at pairs of cipher text whose plain texts have particular differences. The technique analyzes the progress of these differences as the plain texts travel through the various rounds of DES. The idea is to choose pairs of plain text with fixed differences. The two plain texts can be chosen at random, as long as they satisfy specific difference conditions (which can be as simple as XOR). Then, using the differences in the resulting cipher texts, assign different likelihood to different keys. As more and more cipher-text pairs are analyzed, the correct key emerges.

Invented by Mitsuru Matsui, the **linear cryptanalysis** attack is based on linear approximations. If we XOR some plain-text bits together, XOR some cipher-text bits together and then XOR the result, we will get a single bit, which is the XOR of some of the key bits.

The descriptions of these attacks are quite complex, and we will not discuss them here.

(d) Timing Attacks **Timing attacks** refer more to asymmetric-key cryptography. However, they can also apply to symmetric-key cryptography. The idea is simple: observe how long it takes for the cryptographic algorithm to decrypt different blocks of cipher text. The idea is to try and obtain either the plain text or the key used for encryption by observing these timings. In general, it would take different amounts of time to decrypt different sized cipher-text blocks.

3.4.3 Variations of DES

In spite of its strengths, it is generally felt that with the tremendous advances in computer hardware (higher processing speeds of gigahertz, higher memory availability at cheap prices, parallel processing capabilities, etc.), DES is susceptible to possible attacks. However, because DES is already proven to be a very competent algorithm, it would be wise to reuse DES by making it stronger by some means, rather than writing a new cryptographic algorithm. Writing a new algorithm is not easy, more so because it has to be tested sufficiently so as to be proved as a strong algorithm. Consequently, two main variations of DES have emerged, which are **double DES** and **triple DES**. Let us discuss them now.

1. Double DES

Double DES is quite simple to understand. Essentially, it does twice what DES normally does only once. Double DES uses two keys, say $K1$ and $K2$. It first performs DES on the original plain text using $K1$ to get the encrypted text. It again performs DES on the encrypted text, but this time with the other key, i.e. $K2$. The final output is the encryption of encrypted text (i.e. the original plain text encrypted twice with two different keys). This is shown in Fig. 3.38.

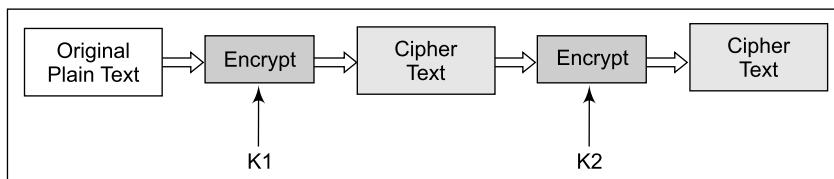


Fig. 3.38 Double DES encryption

Of course, there is no reason why double encryption cannot be applied to other cryptographic algorithms as well. However, in the case of DES, it is already quite popular, and therefore, we have dis-

cussed this in the context of DES. It should also be quite simple to imagine that the decryption process would work in exactly the reverse order, as shown in Fig. 3.39.

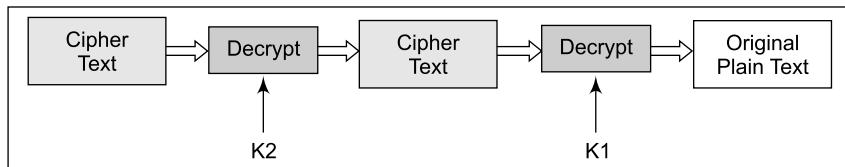


Fig. 3.39 Double DES decryption

The doubly encrypted cipher-text block is first decrypted using the key K_2 to produce the singly encrypted cipher text. This cipher-text block is then decrypted using the key K_1 to obtain the original plain-text block.

If we use a key of just 1 bit, there are two possible keys (0 and 1). If we use a 2-bit key, there are four possible key values (00, 01, 10 and 11). In general, if we use an n -bit key, the cryptanalyst has to perform 2^n operations to try out all the possible keys. If we use two different keys, each consisting of n bits, the cryptanalyst would need 2^{2n} attempts to crack the key. Therefore, on the face of it, we may think that since the cryptanalysis for the basic version of DES requires a search of 2^{56} keys, Double DES would require a key search of $(2^{56})^2$, i.e. 2^{112} keys. However, it is not quite true. Merkle and Hellman introduced the concept of the **meet-in-the-middle** attack. This attack involves encryption from one end, decryption from the other, and matching the results in the middle, hence the name *meet-in-the-middle* attack. Let us understand how it works.

Suppose that the cryptanalyst knows two basic pieces of information: P (a plain-text block), and C (the corresponding final cipher-text block) for a message. We know that the relations shown in Fig. 3.40 are true for P and C , if we are using double DES. The mathematical equivalents of these are also shown. The result of the first encryption is called T , and is denoted as $T = E_{K_1}(P)$ [i.e. encrypt the block P with key K_1]. After this encrypted block is encrypted with another key K_2 , we denote the result as $C = E_{K_2}(E_{K_1}(P))$ [i.e. encrypt the already encrypted block T , with a different key K_2 , and call the final cipher text as C].

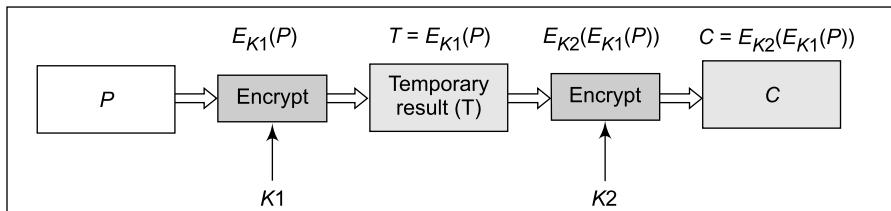


Fig. 3.40 Mathematical expression of double DES

Now, the aim of the cryptanalyst, who is armed with the knowledge of P and C , is to obtain the values of K_1 and K_2 . What would the cryptanalyst do?

Step 1 For all possible values (2^{56}) of key K_1 , the cryptanalyst would use a large table in the memory of the computer, and perform the following two steps:

1. The cryptanalyst would encrypt the plain-text block P by performing the first encryption operation, i.e. $E_{K1}(P)$. That is, it will calculate T .
2. The cryptanalyst would store the output of the operation $E_{K1}(P)$, i.e. the temporary cipher text (T), in the next available row of the table in the memory.

We show this process for the ease of understanding using a 2-bit key (actually, the cryptanalyst has to do this using a 64-bit key, which makes the task a lot harder). Refer to Fig. 3.41.

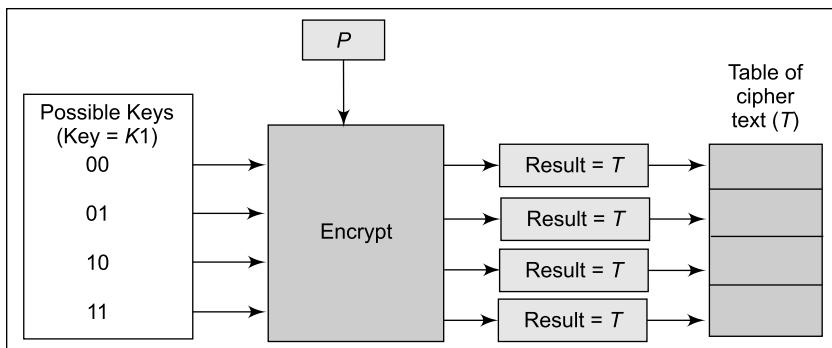


Fig. 3.41 Conceptual view of the cryptanalyst's *Encrypt* operation

Step 2 Thus, at the end of the above process, the cryptanalyst will have the table of cipher texts as shown in the figure. Next, the cryptanalyst will perform the reverse operation. That is, he/she will now decrypt the known cipher text C with all the possible values of $K2$ [i.e. perform $D_{K2}(C)$ for all possible values of $K2$]. In each case, the cryptanalyst will compare the resulting value with all the values in the table of cipher texts, which were computed earlier. This process (as before, for 2-bit key) is shown in Fig. 3.42.

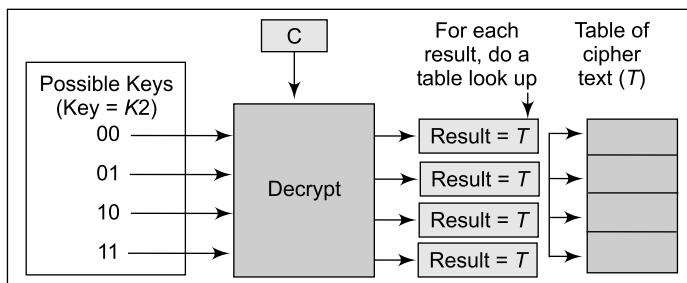


Fig. 3.42 Conceptual view of the cryptanalyst's *Decrypt* operation

To summarize:

- In the first step, the cryptanalyst was calculating the value of T from the left-hand side (i.e. encrypt P with $K1$ to find T). Thus, here $T = E_{K1}(P)$.
- In the second step, the cryptanalyst was finding the value of T from the right-hand side (i.e. decrypt C with $K2$ to find T). Thus, here $T = D_{K2}(C)$.

From the above two steps, we can actually conclude that the temporary result (T) can be obtained in two ways, either by encrypting P with $K1$, or by decrypting C with $K2$. This is because, we can write the following equations:

$$T = E_{K1}(P) = D_{K2}(C)$$

Now, if the cryptanalyst creates a table of $E_{K1}(P)$ (i.e. table of T) for all the possible values of $K1$, and then performs $D_{K2}(C)$ for all possible values of $K2$ (i.e. computes T), there is a chance that she gets the same T in both the operations. If the cryptanalyst is able to find the same T for both *encrypt with K1* and *decrypt with K2* operations, it means that the cryptanalyst knows not only P and C , but he/she has now also been able to find out the possible values of $K1$ and $K2$!

The cryptanalyst can now try this $K1$ and $K2$ pair on another known pair of P and C , and if he/she is able to get the same T by performing the $E_{K1}(P)$ and $D_{K2}(C)$ operations, he/she can then try to use $K1$ and $K2$ for the remaining blocks of the message.

Clearly, this attack is possible, but requires a lot of memory. For an algorithm that uses 64-bit plain-text blocks and 56-bit keys, we would need 2^{56} 64-bit blocks to store the table of T in memory (there is no point in storing it on the disk, as it would be too slow, and defeat the very purpose of the attack). This is equivalent to 10^{17} bytes, which is too high for the next few generations of computers!

2. Triple DES

Although the *meet-in-the-middle* attack on double DES is not quite practical yet, in cryptography, it is always better to take the minimum possible chances. Consequently, double DES seemed inadequate, paving way for **triple DES**. As we can imagine, Triple DES is *DES three times*. It comes in two kinds: one that uses three keys, and the other that uses two keys. We will study both, one by one.

(a) Triple DES with Three Keys The idea of triple DES with three keys is illustrated in Fig. 3.43. As we can see, the plain-text block P is first encrypted with a key $K1$, then encrypted with a second key $K2$, and finally with a third key, $K3$, where $K1$, $K2$ and $K3$ are all different from each other.

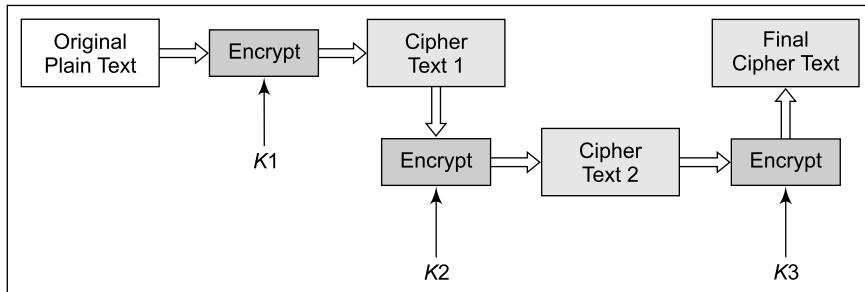


Fig. 3.43 Triple DES with three keys

Triple DES with three keys is used quite extensively in many products, including PGP and S/MIME. To decrypt the cipher text C and obtain the plain text P , we need to perform the operation $P = D_{K1}(D_{K2}(D_{K3}(C)))$.

(b) Triple DES with Two Keys Triple DES with three keys is highly secure. It can be denoted in the form of an equation as $C = E_{K3}(E_{K2}(E_{K1}(P)))$. However, triple DES with three keys also has the drawback of requiring $56 \times 3 = 168$ bits for the key, which can be slightly difficult to have in practical

situations. A workaround suggested by Tuchman uses just two keys for triple DES. Here, the algorithm works as follows:

1. Encrypt the plain text with key K_1 . Thus, we have $E_{K_1}(P)$.
2. Decrypt the output of step 1 above with key K_2 . Thus, we have $D_{K_2}(E_{K_1}(P))$.
3. Finally, encrypt the output of step 2 again with key K_1 . Thus, we have $E_{K_1}(D_{K_2}(E_{K_1}(P)))$.

This is shown in Fig. 3.44.

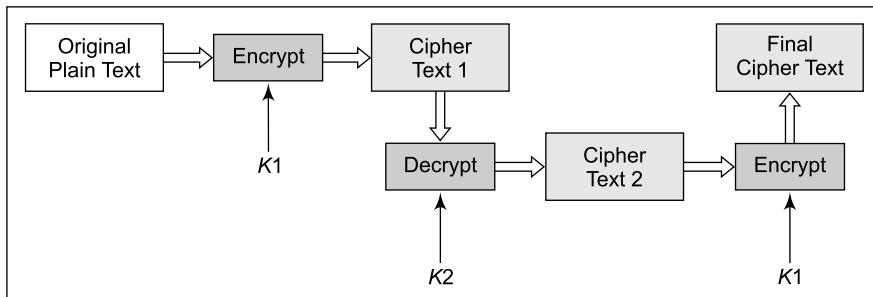


Fig. 3.44 Triple DES with two keys

To decrypt the cipher text C and obtain the original plain text P , we need to perform the operation $P = D_{K_1}(E_{K_2}(D_{K_1}(C)))$.

There is no special meaning attached to the second step of decryption. Its only significance is that it allows triple DES to work with two, rather than three keys. This is also called **Encrypt-Decrypt-Encrypt (EDE)** mode. Triple DES with two keys is not susceptible to the *meet-in-the-middle* attack, unlike double DES as K_1 and K_2 alternate here.

■ 3.5 INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA) ■

3.5.1 Background and History

The **International Data Encryption Algorithm (IDEA)** is perceived as one of the strongest cryptographic algorithms. It was launched in 1990, and underwent certain changes in names and capabilities as shown in Table 3.3.

Table 3.3 Progress of IDEA

Year	Name	Description
1990	Proposed Encryption Standard (PES).	Developed by Xuejia Lai and James Massey at the Swiss Federal Institute of Technology.
1991	Improved Proposed Encryption Standard (IPES).	Improvements in the algorithm as a result of cryptanalysts finding some areas of weakness.
1992	International Data Encryption Algorithm (IDEA).	No major changes, simply renamed.

Although it is quite *strong*, IDEA is not as popular as DES for two primary reasons. Firstly, it is patented unlike DES, and therefore, must be licensed before it can be used in commercial applications. Secondly, DES has a long history and track record as compared to IDEA. However one popular email privacy technology known as **Pretty Good Privacy (PGP)** is based on IDEA.

3.5.2 How IDEA Works

1. Basic Principles

Technically, IDEA is a block cipher. Like DES, it also works on 64-bit plain-text blocks. The key is longer, however, and consists of 128 bits. IDEA is reversible like DES, that is, the same algorithm is used for encryption and decryption. Also, IDEA uses both *diffusion* and *confusion* for encryption.

The working of IDEA can be visualized at a broad level as shown in Fig. 3.45. The 64-bit input plain-text block is divided into four portions of plain text (each of size 16 bits), say P_1 to P_4 . Thus, P_1 to P_4 are the inputs to the first *round* of the algorithm. There are eight such *rounds*. As we mentioned, the key consists of 128 bits. In each *round*, six subkeys are generated from the original key. Each of the subkeys consists of 16 bits. (Do not worry if you do not understand this. We shall discuss this in great detail soon). These six subkeys are applied to the four input blocks P_1 to P_4 . Thus, for the first *round*, we will have the six keys K_1 to K_6 . For the second *round*, we will have keys K_7 to K_{12} . Finally, for the eighth round, we will have keys K_{43} to K_{48} . The final step consists of an **output transformation**, which uses just four subkeys (K_{49} to K_{52}). The final output produced is the output produced by the output transformation step, which is four blocks of cipher text named C_1 to C_4 (each consisting of 16 bits). These are combined to form the final 64-bit cipher-text block.

2. Rounds

We have mentioned that there are 8 rounds in IDEA. Each round involves a series of operations on the four data blocks using six keys. At a broad level, these steps can be described as shown in Fig. 3.46. As we can see, these steps perform a lot of mathematical actions. There are multiplications, additions and XOR operations.

Note that we have put an asterisk in front of the words *add* and *multiply*, causing them to be shown as Add* and Multiply*, respectively. The reason behind this is that these are not mere additions and multiplications. Instead, these are addition modulo 2^{16} (i.e. addition modulo 65536) and multiplication modulo $2^{16} + 1$ (i.e. multiplication modulo 65537), respectively. [For those who are not conversant with modulo arithmetic, if a and b are two integers, then $a \bmod b$ is the remainder of the division a/b . Thus, for example, $5 \bmod 2$ is 1 (because the remainder of $5/2$ is 1), and $5 \bmod 3$ is 2 (because the remainder of $5/3$ is 2)].

Why is this required in IDEA, and what does this mean? Let us examine the case of the normal binary addition with an example. Suppose that we are in *round 2* of IDEA. Further, let us assume that $P_2 = 1111111100000000$ and $K_2 = 11111111000001$. First, simply add them (*and not add* them!*) and see what happens. The operation would look like as shown in Fig. 3.47.

As we can see, the normal addition will produce a number that consists of 17 bits (i.e. 1111111011000001). However, remember that we have only 16 bit positions available for the output of *round 2*. Therefore, we must reduce this number (which is 130753 in decimal) to a 16-bit number. For this, we take modulo

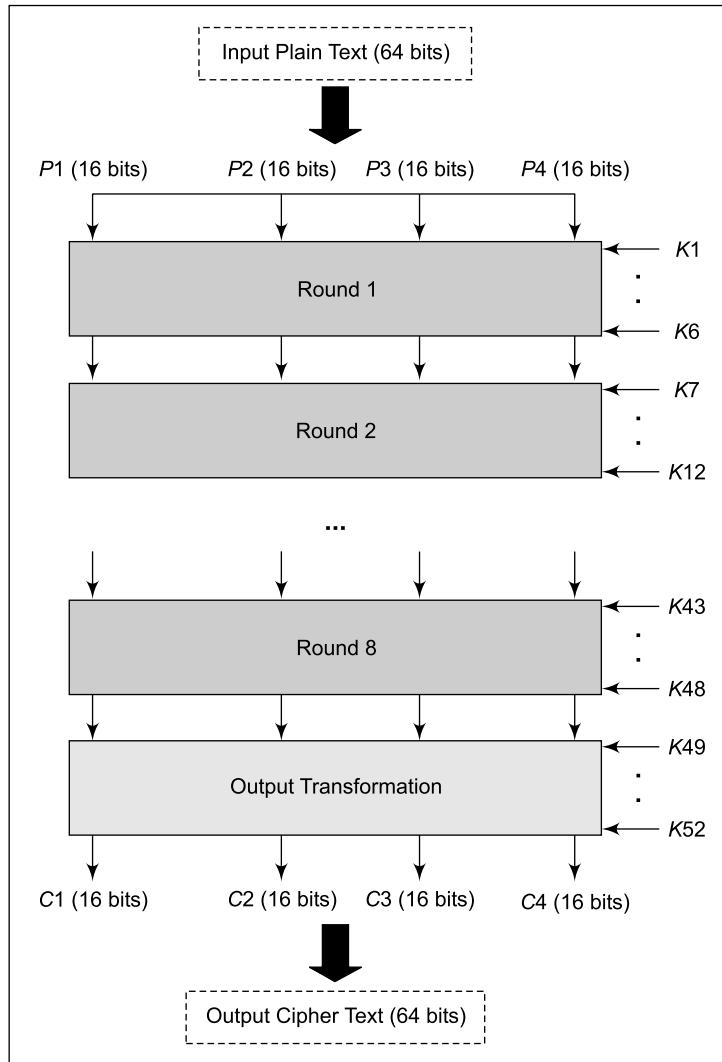
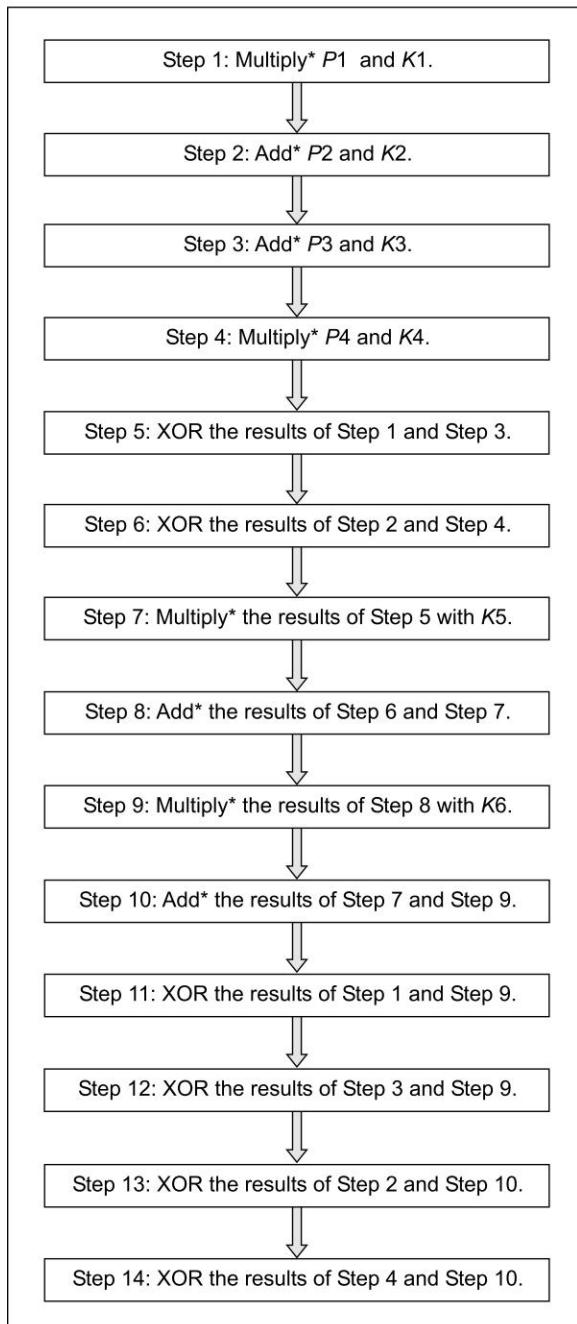
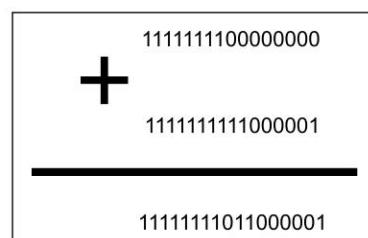


Fig. 3.45 Broad level steps in IDEA

65536 of this. $130753 \text{ modulo } 65536$ yields 65217, which is 111111011000001 in binary, and is a 16-bit number, which fits well in our scheme.

This should explain why modulo arithmetic is required in IDEA. It simply ensures that even if the result of an addition or multiplication of two 16-bit numbers contains more than 17 bits, we bring it back to 16 bits.

Let us now re-look at the details of one round in a more symbolic fashion, as shown in Fig. 3.48. It conveys the same meaning as the earlier diagram, but is more symbolic than verbose in nature. We have depicted the same steps as earlier. The input blocks are shown as P_1 to P_4 , the subkeys are denoted by K_1 to K_6 , and the output of this step is denoted by R_1 to R_4 (and not C_1 to C_4 , because this is not the

**Fig. 3.46** Details of one *round* in IDEA**Fig. 3.47** Binary addition of two 16-bit numbers

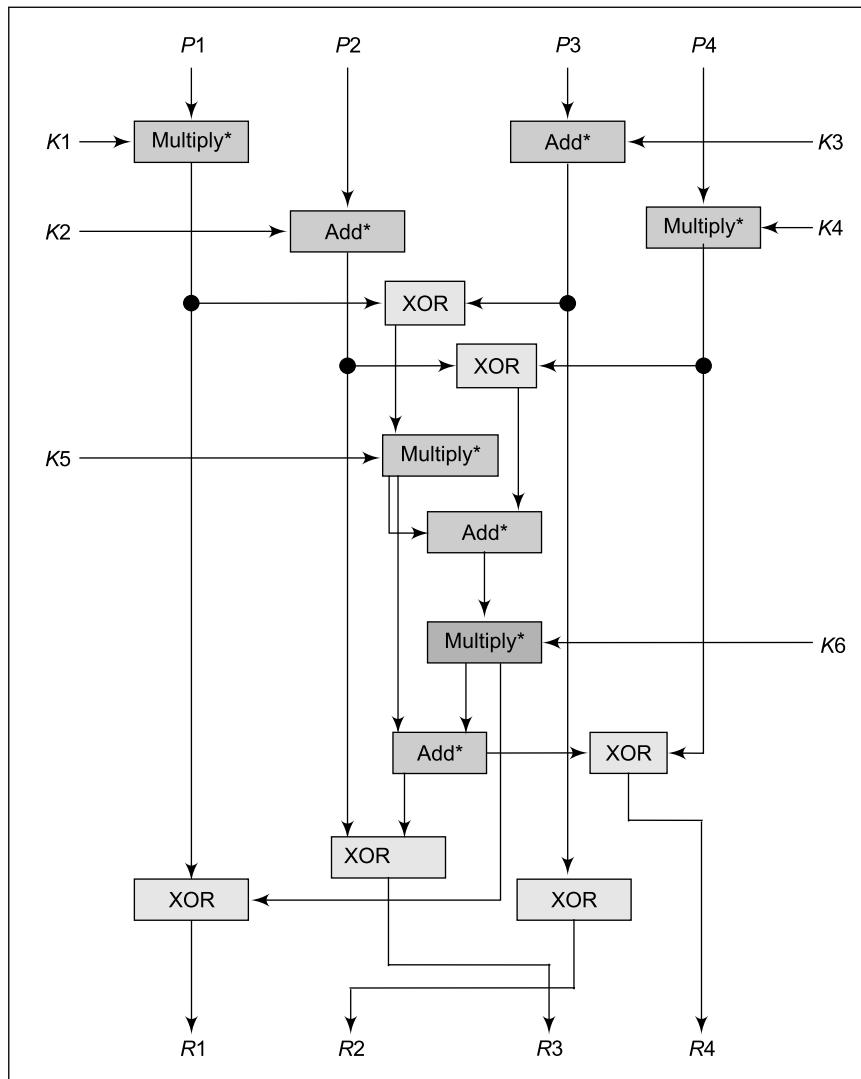


Fig. 3.48 One round of IDEA

final cipher text—it is an intermediate output, which will be processed in further *rounds* as well as in the *output transformation* step.

3. Subkey Generation for a Round

We have been talking about subkeys in our discussion quite frequently. As we mentioned, each of the eight *rounds* makes use of six subkeys (so, $8 \times 6 = 48$ subkeys are required for the *rounds*), and the final output transformation uses four subkeys (making a total of $48 + 4 = 52$ subkeys overall). From an input key of 128 bits, how are these 52 subkeys generated? Let us understand this with the explanation

for the first two rounds. Based on the understanding of the subkey generation process for the first two rounds, we will later tabulate the subkey generation for all the rounds.

(a) First Round We know that the initial key consists of 128 bits, from which 6 subkeys K_1 to K_6 are generated for the first *round*. Since K_1 to K_6 consist of 16 bits each, out of the original 128 bits, the first 96 bits (6 subkeys x 16 bits per subkey) are used for the first round. Thus, at the end of the first round, bits 97–128 of the original key are unused. This is shown in Fig. 3.49.

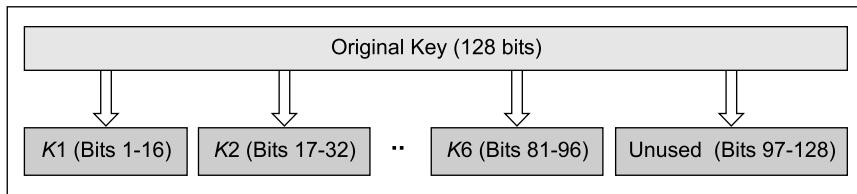


Fig. 3.49 Subkey generation for *round 1*

(b) Second Round As we can see, bits 1–96 are adequate to produce subkeys K_1 to K_6 for round 1. For the second round, we can utilize the 32 unused key bits at positions 97 to 128 (which would give us two subkeys, each of 16 bits). How do we now get the remaining 64 bits for the second round? For this, IDEA employs the technique of **key shifting**. At this stage, the original key is *shifted left circularly* by 25 bits. That is, the 26th bit of the original key moves to the first position, and becomes the first bit after the shift, and the 25th bit of the original key moves to the last position, and becomes the 128th bit after the shift. The whole process is shown in Fig. 3.50.

We can now imagine that the unused bits of the second *round* (i.e. bit positions 65–128) will firstly be used in *round 3*, and then a circular-left shift of 25 bits will be performed once again. From the resulting shifted key, we would extract the first 32 bits, which covers the shortfall in the key bits for this round. This process would go on for the remaining rounds on similar line. These procedures for all the 8 rounds can be tabulated as shown in Table 3.4.

4. Output Transformation

The **output transformation** is a one-time operation. It takes place at the end of the 8th *round*. The input to the output transformation is, of course, the output of the 8th *round*. This is, as usual, a 64-bit value divided into four sub-blocks (say R_1 to R_4 , each consisting of 16 bits). Also, four subkeys are applied here, and not six. We will describe the key-generation process for the output transformation later. For now, we shall assume that four 16-bit subkeys K_1 to K_4 are available to the output transformation. The process of the output transformation is described in Fig. 3.51.

This process can be shown diagrammatically as illustrated by Fig. 3.52. The output of this process is the final 64-bit cipher text, which is a combination of the four cipher-text sub-blocks C_1 to C_4 .

5. Subkey Generation for the Output Transformation

The process for the subkey generation for the output transformation is exactly similar to the subkey generation process for the eight *rounds*. Recall that at the end of the eighth and the final *round*, the key is exhausted and shifted. Therefore, in this round, the first 64 bits make up subkeys K_1 to K_4 , which are used as the four subkeys for this round.

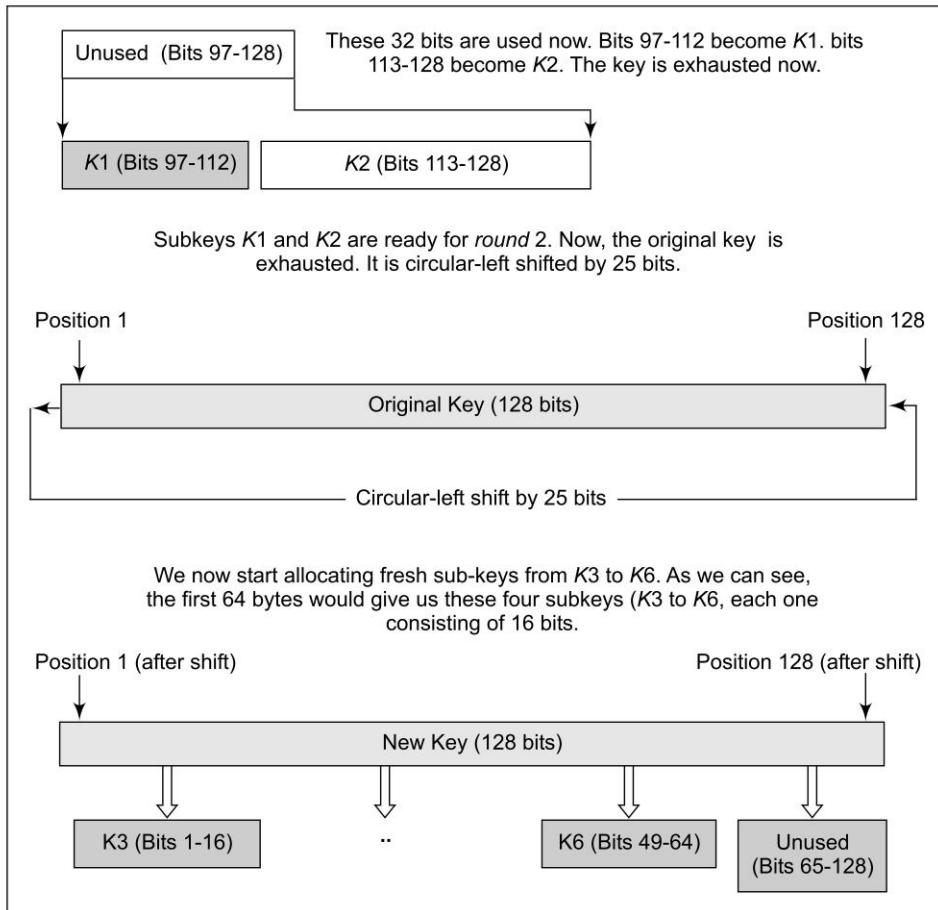


Fig. 3.50 Circular-left key shift and its use in subkey generation for *round 2*

6. IDEA Decryption

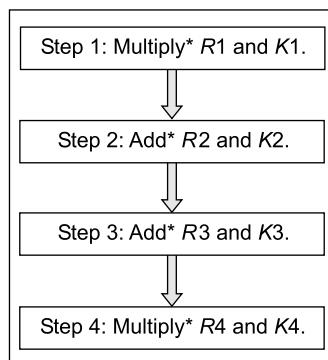
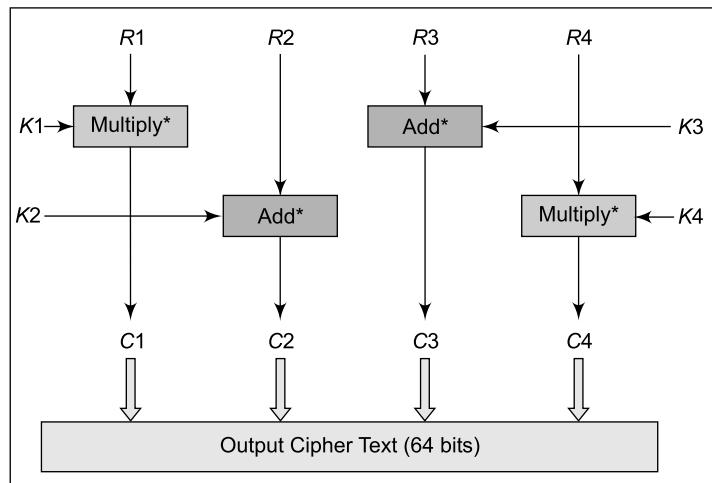
The decryption process is exactly the same as the encryption process. There are some alterations in the generation and pattern of subkeys. The decryption subkeys are actually an inverse of the encryption subkeys. We shall not discuss this any further, as the basic concepts remain the same.

7. The Strength of IDEA

IDEA uses a 128-bit key, which is double than the key size of DES. Thus, to break into IDEA, 2^{128} (i.e. 10^{38}) encryption operations would be required. As before, even if we assume that to obtain the correct key, only half of the possible keys (i.e. half of the *key space*) needs to be examined and tried out, a single computer performing one IDEA encryption per microsecond would require more than 54000000000000000000000000000000 years to break IDEA!

Table 3.4 Subkey generation process for each round

Round	Details of the subkey generation and use
1	Bit positions 1-96 of the initial 128-bit key would be used. This would give us 6 subkeys K_1 to K_6 for round 1. Key bits 97 to 128 are available for the next round.
2	Key bits 97 to 128 make up subkeys K_1 and K_2 for this round. A 25-bit shift on the original key happens, as explained. Post this shifting; the first 64 bits are used as subkeys K_3 to K_6 for this round. This leaves bits 65 to 128 unused for the next round.
3	Unused key bits 65 to 128 are used as subkeys K_1 to K_4 of this round. Upon key exhaustion, another 25-bit shift happens, and bits 1 to 32 of the shifted key are used as subkeys K_5 and K_6 . This leaves bits 33 to 128 unused for the next round.
4	Bits 33 to 128 are used for this round, which is perfectly adequate. No bits are unused at this stage. After this, the current key is again shifted.
5	This is similar to round 1. Bit positions 1-96 of the current 128-bit key would be used. This would give us 6 subkeys K_1 to K_6 for round 1. Key bits 97 to 128 are available for the next round.
6	Key bits 97 to 128 make up subkeys K_1 and K_2 for this round. A 25-bit shift on the original key happens, as explained. Post this shifting; the first 64 bits are used as subkeys K_3 to K_6 for this round. This leaves bits 65 to 128 unused for the next round.
7	Unused key bits 65 to 128 are used as subkeys K_1 to K_4 of this round. Upon key exhaustion, another 25-bit shift happens, and bits 1 to 32 of the shifted key are used as subkeys K_5 and K_6 . This leaves bits 33 to 128 unused for the next round.
8	Bits 33 to 128 are used for this round, which is perfectly adequate. No bits are unused at this stage. After this, the current key is again shifted for the output transformation round.

**Fig. 3.51** Details of the output transformation**Fig. 3.52** Output transformation process

■ 3.6 RC4 ■

3.6.1 Background

RC4 was designed by Ron Rivest of RSA Security in 1987. The official name for this algorithm is “Rivest Cipher 4”. However, because of its ease of reference, the acronym **RC4** has stuck.

RC4 is a stream cipher. This means that the encryption happens byte-by-byte. However, this can be changed to bit-by-bit encryption (or to a size other than a byte/bit).

RC4 was initially kept secret. However, in September 1994, a description of the algorithm was anonymously posted to the *Cypherpunks* mailing list. Thereafter, it was posted on the *sci.crypt* newsgroup, and from there to many other Internet sites. Because the algorithm is now well known, it is no longer a secret. However, we should note that there is a trademark associated with the name “RC4”.

An interesting remark says that “The current status seems to be that *unofficial* implementations are legal, but cannot use the RC4 name.”

RC4 has become part of some widely used encryption techniques and standards, including WEP and WPA for wireless cards and TLS. What has made its wide deployment possible is its speed and simplicity of design. Implementations in both software and hardware are possible. RC4 does not consume many resources.

3.6.2 Description

RC4 generates a pseudorandom stream of bits called **keystream**. This is combined with the plain text using XOR for encryption. Even decryption is performed in a similar manner.

Let us understand this in more detail now.

There is a variable length key consisting of 1 to 256 bytes (or 8 to 2048 bits). This key is used to initialize a 256-byte *state vector* with elements identified as $S[0]$, $S[1]$, ..., $S[255]$. To perform an encryption or decryption operation, one of these 256 bytes of S is selected, and processed. We will call the resulting output as k . After this, the entries in S are permuted once again.

Overall, there are two processes involved: (a) initialization of S , and (b) stream generation. We describe these below.

1. Initialization of S

This process consists of the following steps.

1. Choose a key (K) of length between 1 and 256 bytes.
2. Set the values in the state vector S equal to the values from 0 to 255 in an ascending order. In other words, we should have $S[0] = 0$, $S[1] = 1$, ..., $S[255] = 255$.
3. Create another temporary array T . If the length of the key K (termed as *keylen*) is 256 bytes, copy K into T as is. Otherwise, after copying K to T , whatever are the remaining positions in T are filled with the values of K again. At the end, T should be completely filled.

Thus, the following steps are executed:

for $i = 0$ to 255

```
// Copy the current value of i into the current position in the S array
S [i] = i;

// Now copy the contents of the current position of the K array into T. If K is exhausted, loop back
// to get the values of the K array from the unexhausted portion of K.

T [i] = K [i mod keylen];
```

A small Java program shown in Fig. 3.53 implements this logic. Here, we have considered that the K array contains 10 elements, i.e. $keylen$ is 10.

```
public class InitRC4 {

    public static void main (String [] args) {

        int[] S, T, K;
        S = new int [255];
        T = new int [255];
        K = new int [255];

        int i;
        int keylen = 10;

        for (i=0; i<200; i++)
            K [i] = i * 2;

        for (i=0; i<255; i++) {
            S [i] = i;
            T [i] = K [i % keylen];
        }
    }
}
```

Fig. 3.53 Java code for implementing *initialization of S* step

After this, the T array is used to produce initial permutation of S . For this purpose, a loop executes, iterating i from 0 to 255. In each case, the byte at the position $S [i]$ is swapped with another byte in the S array, as per an arrangement decided by $T [i]$. For this purpose, the following logic is used:

$j = 0;$

for $i = 0$ to 255

```
j = (j + S [i] + T [i]) mod 256;
swap (S [i], S [j]);
```

Note that this is just a permutation. The values of S are simply being rearranged, not changed. The corresponding Java portion is shown in Fig. 3.54.

```
// Initial permutation of S

int j = 0, temp;

for (i=0; i<255; i++) {
    j = (j + S [i] + T [i]) % 256;
    temp = S [i];
    S [i] = T [i];
    T [i] = temp;
}

for (i=0; i<255; i++) {
    System.out.println ("S [i] = " + S [i]);
}
```

Fig. 3.54 Java code for implementing *permutation of S* step

2. Stream Generation

Now that the S array is ready with the above initializations and permutations, the initial key array K is discarded. Now, we need to again loop for $i = 0$ to 255. In each step, we swap $S [i]$ with another byte in S , as per the mechanism decided by the implementation of S . Once we exhaust the 255 positions, we need to restart at $S [0]$.

The logic is as follows:

```
i = 0;
j = 0;
while (true)
{
    i = (i + 1) mod 256;
    j = (j + S [i]) mod 256;
    swap (S [i], S [j]);
    t = (S [i] + S [j]) mod 256;
    k = S [t];
}
```

After this, for encryption, k is XORed with the next byte of the plain text. For decryption, k is XORed with the next byte of the cipher text.

Some vulnerability has been found in RC4, hence it is not recommended for new applications.

■ 3.7 RC5 ■

3.7.1 Background and History

RC5 is a symmetric-key block-encryption algorithm developed by Ron Rivest. The main features of RC5 are that it is quite fast as it uses only the primitive computer operations (such as addition, XOR, shift, etc). It allows for a variable number of *rounds*, and a variable bit-size key to add to the flexibility. Different applications that demand varying security needs can set these values accordingly. Another important aspect is that RC5 requires less memory for execution, and is therefore, suitable not only

for desktop computers, but also for smart cards and other devices that have a small memory capacity. It has been incorporated into the RSA Data Security Incorporation's products such as BSAFE, JSafe and S/Mail.

3.7.2 How RC5 Works

1. Basic Principles

In RC5, the word size (i.e. input plain-text block size), number of rounds and number of 8-bit bytes (octets) of the key, all can be of variable length. These values can consist of the sizes as shown in Table 3.5. Of course, once decided, these values remain the same for a particular execution of the cryptographic algorithm. These are variable in the sense that before the execution of a particular instance of RC5, these values can be chosen from those allowed. This is unlike DES, for instance, where the block size must be of 64 bits and the key size must always be of 56 bits; or unlike IDEA, which uses 64-bit blocks and 128-bit keys.

Table 3.5 RC5 block, round and key details

Parameter	Allowed Values
Word size in bits (RC5 encrypts 2-word blocks at a time)	16, 32, 64
Number of rounds	0-255
Number of 8-bit bytes (octets) in the key	0-255

The following conclusions emerge from the table:

- The plain-text block size can be of 32, 64 or 128 bits (since 2-word blocks are used).
- The key length can be 0 to 2040 bits (since we have specified the allowed values for 8-bit keys).

The output resulting from RC5 is the cipher text, which has the same size as the input plain text. Since RC5 allows for variable values in the three parameters, as specified, a particular instance of the RC5 algorithm is denoted as $RC5-w/r/b$, where w = word size in bits, r = number of rounds, b = number of 8-bit bytes in the key. Thus, if we have $RC5-32/16/16$, it means that we are using RC5 with a block size of 64 bits (remember that RC5 uses 2-word blocks), 16 rounds of encryption, and 16 bytes (i.e. 128 bits) in the key. Rivest has suggested $RC5-32/12/16$ as the *minimum safety version*.

2. Principles of Operation

At first, RC5 appears to be complicated because of the notations used. However, it is actually quite simple to understand. Rather than getting into notations, we shall first illustrate the working of RC5 using Fig. 3.55. As shown in the figure, there is one initial operation consisting of two steps (shown shaded), then a number of *rounds*. As we have noted, the number of rounds (r) can vary from 0 to 255.

For simplicity, we shall assume that we are working on an input plain block with size 64 bits. The same principles operation will apply to other block sizes, in general.

In the first two steps of the one-time initial operation, the input plain text is divided into two 32-bit blocks A and B . The first two subkeys (we shall later see how they are generated) $S[0]$ and $S[1]$ are

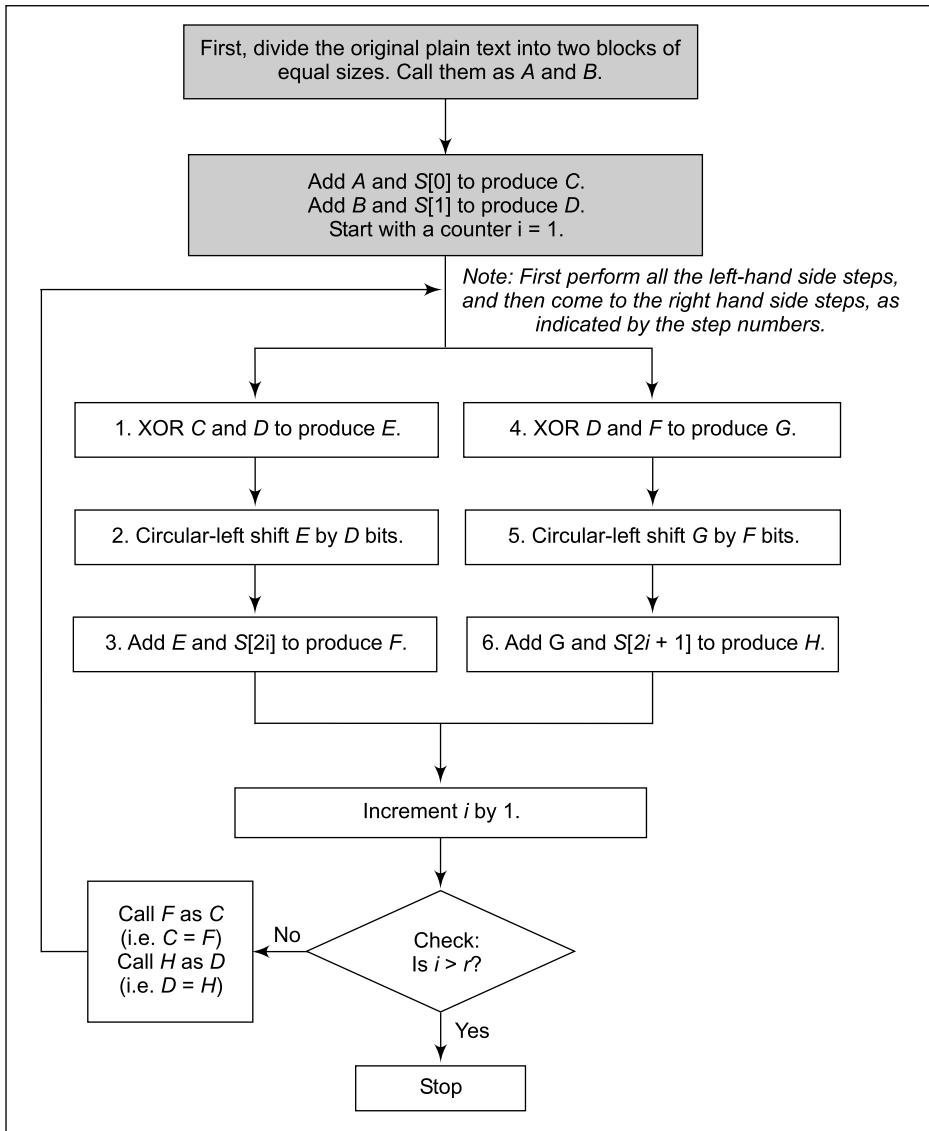


Fig. 3.55 Encryption using RC5

added to A and B , respectively. This produces C and D respectively, and marks the end of the one-time operation.

Then, the *rounds* begin. In each round, there are following operations:

- Bitwise XOR
- Left circular-shift
- Addition with the next subkey, for both C and D . This is the addition operation first, and then the result of the addition mod 2^w (since $w = 32$ here, we have 2^{32}) is performed.

If we observe the operations shown in the figure carefully, we will note that the output of one block is fed back as the input to another block, making the whole logic quite complicated to decipher.

Let us now look at the algorithm details in step-by-step fashion in the next few sections.

3. One-time Initial Operation

We will first take a look at the one-time initial operation, as shown in Fig. 3.56. This consists of two simple steps: first, the input plain text is divided into two equal-sized blocks, A and B . Then the first subkey, i.e. $S[0]$ is added to A , and the second sub-key, i.e. $S[1]$ is added to B . These operations are mod 2^{32} , as we have noted, and produce C and D , respectively.

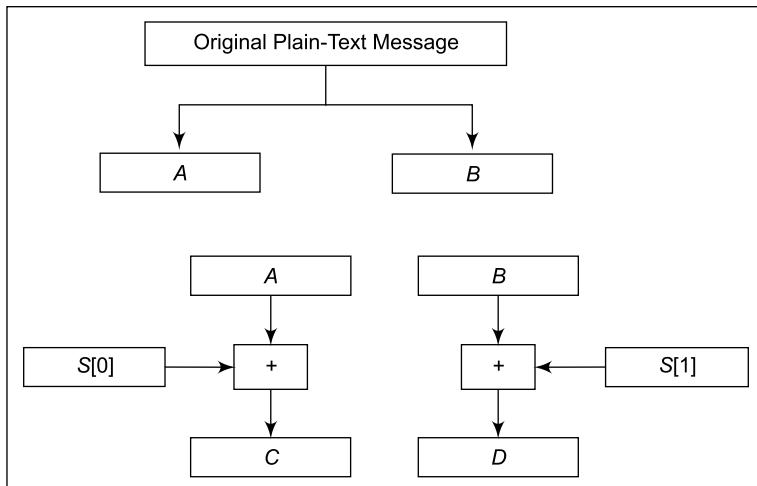


Fig. 3.56 One-time initial operation in RC5

4. Details of One Round

Now, we observe the results for the first *round*. The process for the first *round* will apply for further *rounds*. Therefore, we shall not discuss those *rounds* separately.

Step 1: XOR C and D In the first step of each *round*, C and D are XORed together to form E , as shown in Fig. 3.57.

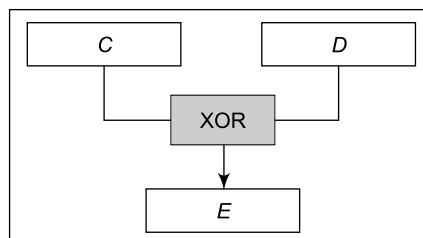


Fig. 3.57 Step 1 in a *round*

Step 2: Circular-left shift E Now, E is circular-left shifted by D positions, as shown in Fig. 3.58.

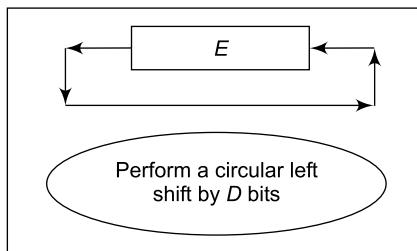


Fig. 3.58 Step 2 in a round

Step 3: Add E and Next Subkey In this step, E is added to the next subkey (which is $S[2]$ for the first round, and $S[2i]$ in general, for any round, where i starts with 1. The output of this process is F . This is shown in Fig. 3.59.

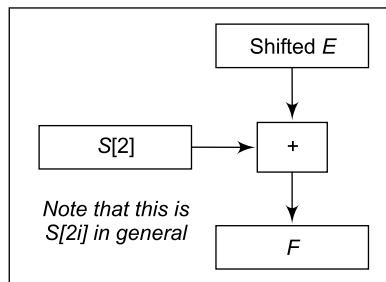


Fig. 3.59 Step 3 in a round

We would now note that the operations (XOR, circular-left shift and add) in steps 4 to 6 that follow are the same as the operations in steps 1 to 3. The only difference, of course, is that the inputs to the steps 4-6 are different from those to steps 1-3.

Step 4: XOR D and F This step is similar to step 1. Here, D and F are XORed to produce G . This is shown in Fig. 3.60.

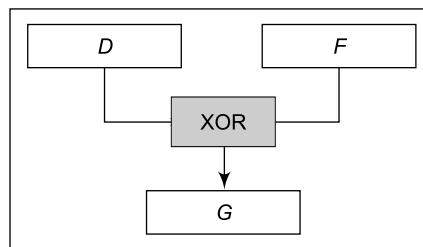


Fig. 3.60 Step 4 in a round

Step 5: Circular-left Shift G This step is similar to step 2. Here, G is circular-left shifted by F positions, as shown in Fig. 3.61.

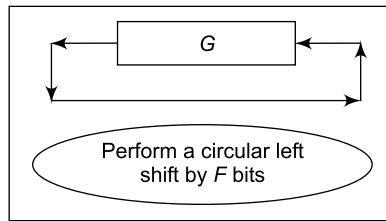


Fig. 3.61 Step 5 in a *round*

Step 6: Add G and Next Subkey In this step (which is identical to step 3), G is added to the next subkey (which is $S[3]$ for the first round, and $S[2i + 1]$ in general, for any round, where i starts with 1). The output of this process is H . This is shown in Fig. 3.62.

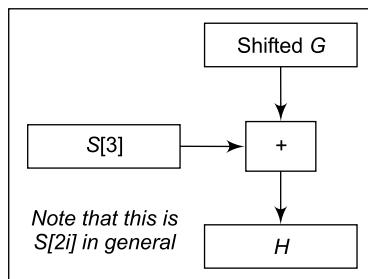


Fig. 3.62 Step 6 in a *round*

Step 7: Miscellaneous Tasks In this step, we check to see if all the rounds are over or not. For this, perform the following steps:

- Increment i by 1
- Check to see if $i < r$

Assuming that i is still less than r , we rename F as C and H as D , and return back to step 1. This is shown in Fig. 3.63.

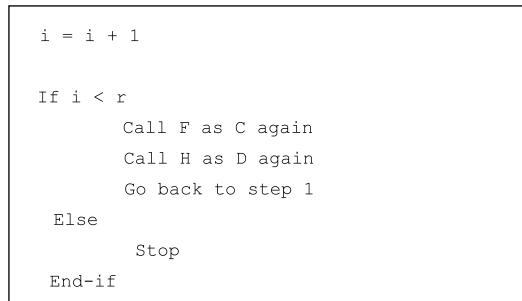


Fig. 3.63 Step 7 in a *round*

5. Mathematical Representation

Interestingly, all these operations (one-time initial operation and all the *rounds*) in RC5 can be signified mathematically in a very cryptic fashion, as shown in Fig. 3.64. Note that <<< means circular-left shift.

```

A = A + S[0]
B = B + S[1]

For i = 1 to r
    A = ((A XOR B) <<< B) + S[2i]
    B = ((B XOR A) <<< A) + S[2i + 1]
Next i

```

Fig. 3.64 Mathematical representation of RC5 encryption

Furthermore, we will also write the mathematical representation of RC5 decryption process as shown in Fig. 3.65, and leave its verification as an exercise. Note that >>> means circular right-shift.

```

For i = r to 1 step-1 (i.e. decrement i each time by 1)
    A = ((B - S[2i + 1]) >>> A) XOR A
    B = ((A - S[2i]) >>> B) XOR B
Next i
B = B - S[1]
A = A - S[0]

```

Fig. 3.65 Mathematical representation of RC5 decryption

6. Subkey Creation

Let us now examine the subkey creation process in RC5. This is a two-step process.

1. In the first step, the subkeys (denoted by $S[0], S[1], \dots$) are generated.
2. The original key is called L . In the second step, the subkeys ($S[0], S[1], \dots$) are mixed with the corresponding subportions of the original key (i.e. $L[0], L[1], \dots$).

This is shown in Fig. 3.66.

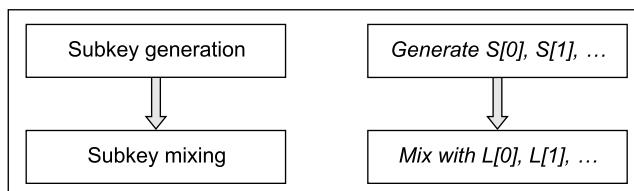


Fig. 3.66 Subkey generation process

Let us understand these two steps.

Step 1: Subkey Generation In this step, two constants P and Q are used. The array of subkeys to be generated is called as S . The first subkey $S[0]$ is initialized with the value of P .

Each next subkey (i.e. $S[1], S[2], \dots$) is calculated on the basis of the previous sub-key and the constant value Q , using the addition mod 2^{32} operations. The process is done $2(r + 1) - 1$ times, where r is the number of *rounds*, as before. Thus, if we have 12 rounds, this process will be done $2(12 + 1) - 1$ times, i.e. $2(13) - 1$ times, i.e. 25 times. Thus, we will generate subkeys $S[0], S[1], \dots, S[25]$.

This process is illustrated in Fig. 3.67.

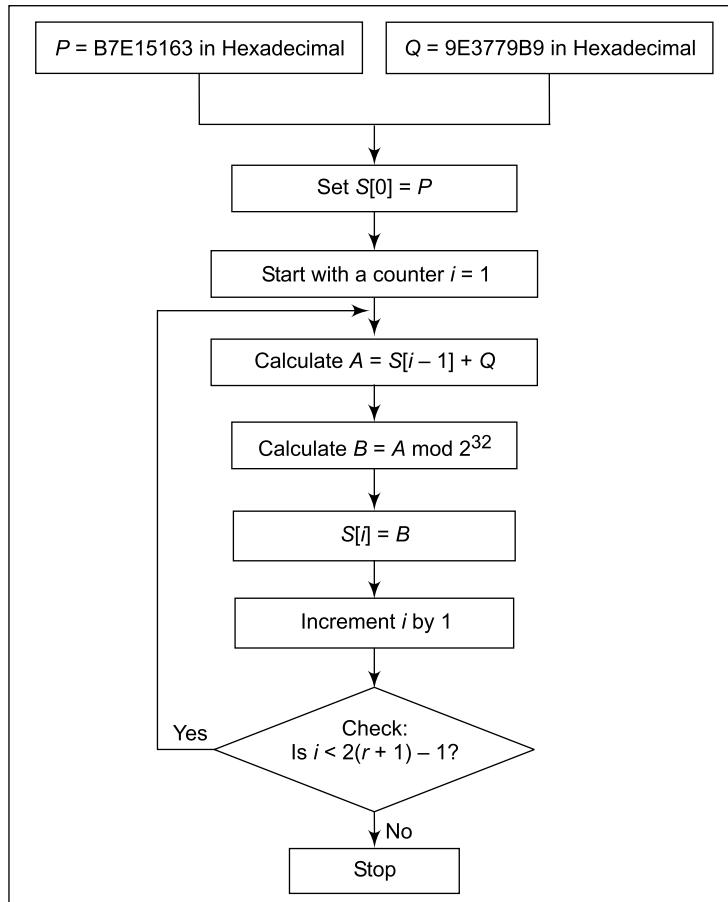


Fig. 3.67 Subkey generation

The mathematical form of this subkey generation is shown in Fig. 3.68.

```

S[0] = P

For i = 1 to 2 (r + 1) - 1
    S[i] = (S[i - 1] + Q) mod 232
Next i
  
```

Fig. 3.68 Mathematical representation of subkey generation

Step 2: Subkey Mixing In the subkey mixing stage, the subkeys $S[0], S[1], \dots$ are mixed with the subportions of the original key, i.e. $L[0], L[1], \dots L[c]$. Note that c is the last subkey position in the original key. This process is shown mathematically in Fig. 3.69.

```

i = j = 0
A = B = 0

Do 3n times (where n is the maximum of 2 (r + 1) and c)

    A = S[i] = (S[i] + A + B) <<< 3
    B = L[i] = (L[i] + A + B) <<< (A + B)

    i = (i + 1) mod 2 (r + 1)
    j = (j + 1) mod c

End-do

```

Fig. 3.69 Mathematical representation of subkey mixing

3.7.3 RC5 Modes

The overall performance of RC5 can be made better by implementing it in one of the following algorithm modes, as shown in Table 3.6. This is defined in RFC 2040.

Table 3.6 RC5 modes

Mode	Description
RC5 block cipher	This is also called Electronic Codebook (ECB) mode. It involves encryption of fixed-size input blocks consisting of $2w$ bits into cipher-text block of the same length.
RC5-CBC	This is the cipher chaining Block for RC5. In this, plain-text messages whose length is equal to a multiple of the RC5 block size (i.e. multiple of $2w$ bits) are encrypted. This mode offers better security, as the same plain-text blocks in the input produce different cipher-text blocks in the output.
RC5-CBC-Pad	This is a modified version of CBC. Here, the input can be of any length. The cipher text is longer than the plain text by at most the size of a single RC5 block. To handle length mismatches, padding is used. This makes the length of a message equal to a multiple of $2w$ bits. The length of the original message is considered to be a specific number of integer number of bytes. Padding of length 1 to bb bytes is added to the end of the message. How many such padding bytes are added is decided by a simple formula $bb = 2w / 8$. Thus, b equals the block size for RC5 in bytes. The value of all padding bytes is the same, and is set to a byte that represents the number of bytes used in padding. For instance, if there are 6 padding bytes, each padding byte contains a value of 6 in binary, i.e. 00000110.
RC5-CTS	This is called cipher-text stealing mode. This mode is similar to RC5-CBC-Pad. Here, the plain-text input can be of any length. The output cipher text is also of equal length.

■ 3.8 BLOWFISH ■

3.8.1 Introduction

Blowfish was developed by Bruce Schneier, and has the reputation of being a very strong symmetric-key cryptographic algorithm. According to Schneier, Blowfish was designed with the following objectives in mind.

- (a) Fast** Blowfish encryption rate on 32-bit microprocessors is 26 clock cycles per byte.
- (b) Compact** Blowfish can execute in less than 5 KB memory.
- (c) Simple** Blowfish uses only primitive operations, such as addition, XOR and table look-up, making its design and implementation simple.
- (d) Secure** Blowfish has a variable key length up to a maximum of 448 bits long, making it both flexible and secure.

Blowfish suits applications where the key remains constant for a long time (e.g. communications link encryption), but not where the key changes frequently (e.g. packet switching).

3.8.2 Operation

Blowfish encrypts 64-bit blocks with a variable-length key. It contains two parts, as follows.

- (a) Subkey Generation** This process converts the key up to 448 bits long to subkeys totaling 4168 bits.
- (b) Data Encryption** This process involves the iteration of a simple function 16 times. Each round contains a key-dependent permutation and key- and data-dependent substitution.

We will now discuss these two parts.

1. Subkey Generation

Let us understand the important aspects of the subkey generation process step by step.

- (a) Blowfish makes use of a very large number of subkeys. These keys have to be ready before encryption and decryption happen. The key size ranges from 32 bits to 448 bits. In other words, the key size ranges from 1 to 14 words, each comprising a word of 32 bits. These keys are stored in an array, as follows:

$$K_1, K_2, \dots, K_n \quad \text{where } 1 \leq n \leq 14$$

- (b) We then have the concept of a *P*-array, consisting of 18 32-bit sub-keys:

$$P_1, P_2, \dots, P_{18}$$

Creation of the *P*-array is described subsequently.

- (c) Four *S*-boxes, each containing 256 32-bit entries:

$$S_{1,0}, S_{1,1}, \dots, S_{1,255}$$

$$S_{2,0}, S_{2,1}, \dots, S_{2,255}$$

$S3,0, S3,1 \dots, S3,255$

$S4,0, S4,1 \dots, S4,255$

Creation of the P -array is described subsequently.

Now let us examine how all this information is used to generate subkeys.

- (a) Initialize the P -array first, followed by the four S -boxes, with a fixed string. Schneier recommends the usage of the bits of the fractional part (in hexadecimal form) of the constant pi (π) for this purpose. Thus, we will have:

$$P1 = 243F6A88$$

$$P2 = 85A308D3$$

....

$$S4,254 = 578FDDE3$$

$$S4,255 = 3AC372E6$$

- (b) Do a bitwise XOR of $P1$ with $K1$, $P2$ with $K2$, etc., until $P18$. This works fine till $P14$ and $K14$. At this stage, the key array (K) is exhausted. Hence, for $P15$ to $P18$, reuse $K1$ to $K4$. In other words, do the following:

$$P1 = P1 \text{ XOR } K1$$

$$P2 = P2 \text{ XOR } K2$$

....

$$P14 = P14 \text{ XOR } K14$$

$$P15 = P15 \text{ XOR } K1$$

$$P16 = P16 \text{ XOR } K2$$

$$P17 = P17 \text{ XOR } K3$$

$$P18 = P18 \text{ XOR } K4$$

- (c) Now take a 64-bit block, with all the 64 bits initialized to value of 0. Use the above P -arrays and S -boxes above (the P -arrays and S -boxes are called subkeys) to run the Blowfish encryption process (described in the next section) on the 64-bit all-zero block. In other words, to generate the subkeys themselves, the Blowfish algorithm is used. It is needless to say that once the final subkeys are ready, the Blowfish algorithm would be used to encrypt the actual plain text.

This step would produce a 64-bit cipher text. Divide this into two 32-bit blocks and replace the original values of $P1$ and $P2$ with these 32-bit block values, respectively.

- (d) Encrypt the output of step (c) above using Blowfish with the modified subkeys. The resulting output would again consist of 64 bits. As before, divide this into two blocks of 32 bits each. Now, replace $P3$ and $P4$ with the contents of these two ciphertext blocks.

- (e) In the same manner, replace all the remaining P -arrays (i.e. $P5$ through $P18$) and then all the elements of the four S -boxes, in order. In each step, the output of the previous step is fed to the Blowfish algorithm to generate the next two 32-bit blocks of the subkey (i.e. $P5$ and $P6$, followed by $P7$ and $P8$, etc.).

In all, 521 iterations of the Blowfish algorithm are required to generate all P -arrays and S -boxes.

2. Data Encryption and Decryption

The encryption of a 64-bit block plain-text input X is shown in an algorithmic fashion in Fig. 3.70. We use the P -arrays and S -boxes during the encryption and decryption processes.

1. Divide X into two blocks: XL and XR, of equal sizes. Thus, both XL and XR will consist of 32 bits each.
2. For $i = 1$ to 16
 - XL = XL XOR P_i
 - XR = F(XL) XOR XR
 - Swap XL, XR
3. Next i
3. Swap XL, XR (i.e. undo last swap).
4. $XL = XL \text{ XOR } P_{18}$.
5. Combine XL and XR back into X.

Fig. 3.70 Blowfish algorithm

This process is depicted in Fig. 3.71.

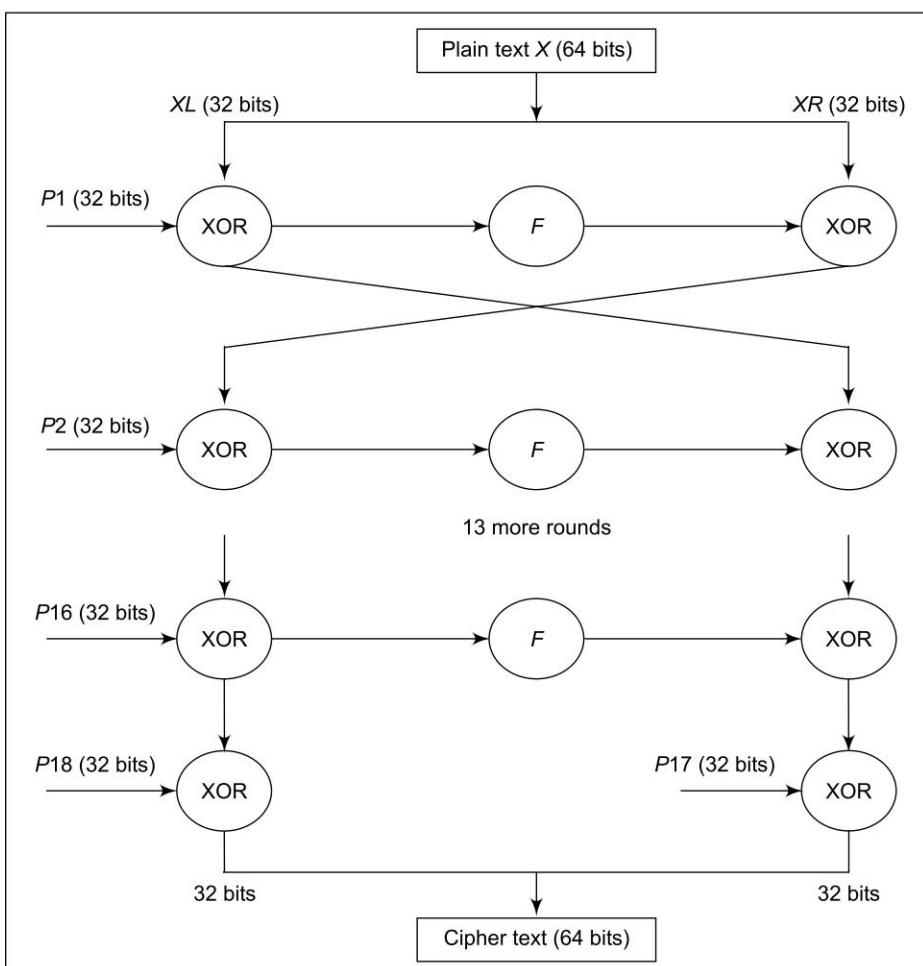


Fig. 3.71 Blowfish encryption

The function F is as follows:

- Divide the 32-bit XL block into four 8-bit sub-blocks, named a , b , c , and d .
- Compute $F[a, b, c, d] = ((S1,a + S2,b) \text{ XOR } S3,c) + S4,d$. For example, if $a = 10$, $b = 95$, $c = 37$, and $d = 191$, then the computation of F would be:

$$F[a, b, c, d] = ((S1,10 + S2,95) \text{ XOR } S3,37) + S4,191$$

The diagrammatic view of the function F is shown in Fig. 3.72.

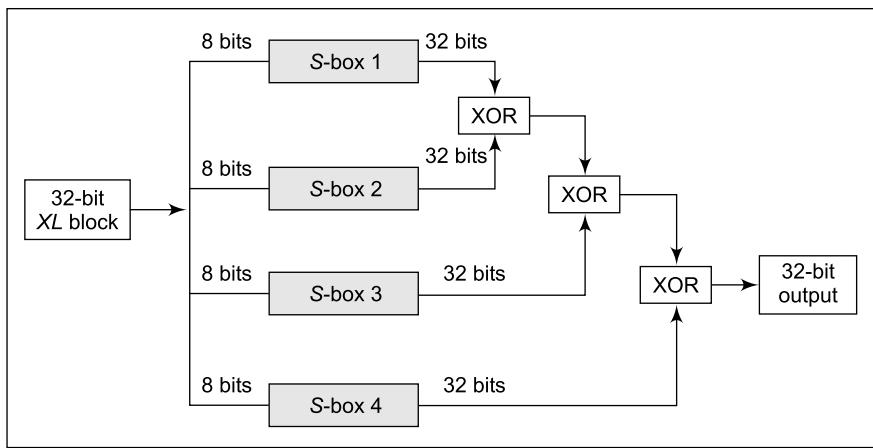


Fig. 3.72 Function F in Blowfish

The decryption process is shown in Fig. 3.73. As we can see, it is quite similar to the encryption process, with the reversal of P -array values.

■ 3.9 ADVANCED ENCRYPTION STANDARD (AES) ■

3.9.1 Introduction

In the 1990s, the US Government wanted to standardize a cryptographic algorithm, which was to be used universally by them. It was to be called the **Advanced Encryption Standard (AES)**. Many proposals were submitted, and after a lot of debate, an algorithm called **Rijndael** was accepted. Rijndael was developed by Joan Daemen and Vincent Rijmen (both from Belgium). The name Rijndael was also based on their surnames (Rijmen and Daemen).

The need for coming up with a new algorithm was actually because of the perceived weakness in DES. The 56-bit keys of DES were no longer considered safe against attacks based on exhaustive key searches, and the 64-bit blocks were also considered weak. AES was to be based on 128-bit blocks, with 128-bit keys.

In June 1998, the Rijndael proposal was submitted to NIST as one of the candidates for AES. Out of the initial 15 candidates, only 5 were shortlisted in August 1999, which were as follows:

1. Rijndael (From Joan Daemen and Vincent Rijmen; 86 votes)
2. Serpent (From Ross Anderson, Eli Biham, and Lars Knudsen; 59 votes)

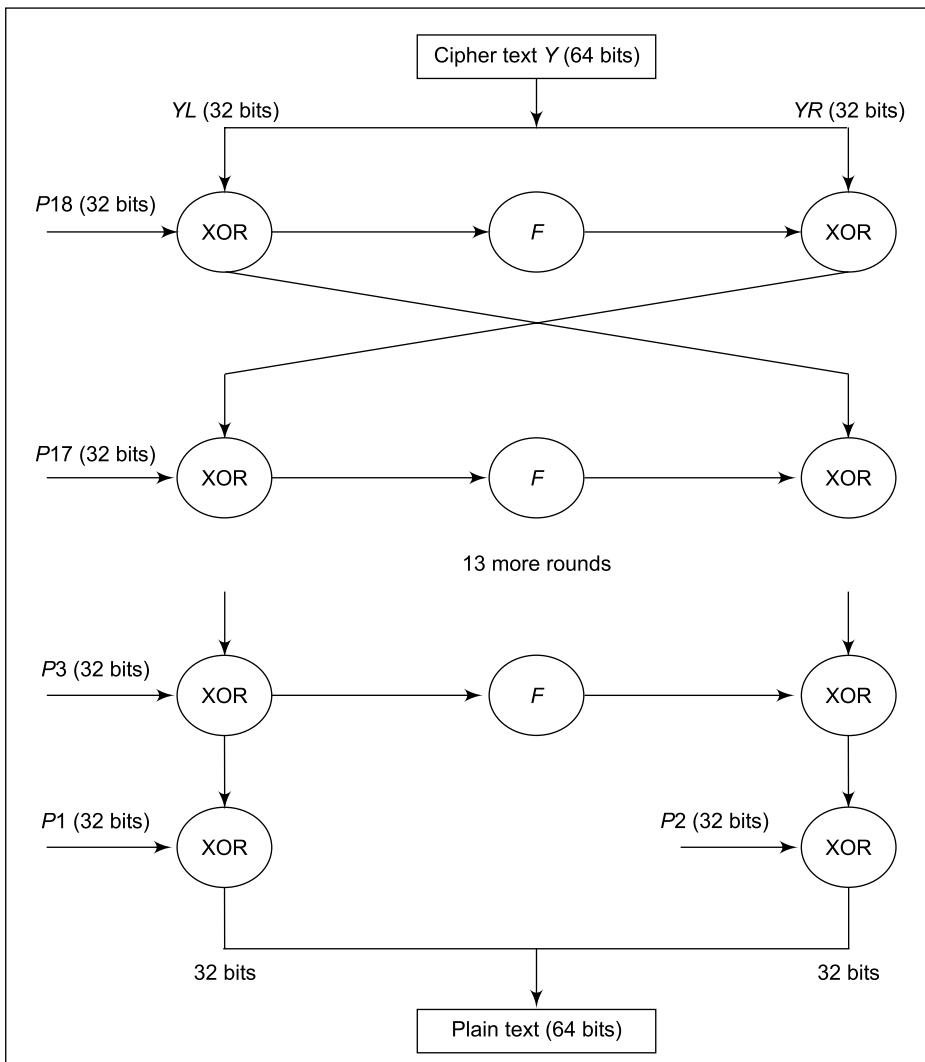


Fig. 3.73 Blowfish decryption

3. Twofish (From Bruce Schneier and others, 31 votes)
4. RC6 (From RSA Laboratories, 23 votes)
5. MARS (From IBM, 13 votes)

In October 2000, Rijndael was announced as the final selection for AES. In November 2001, Rijndael became a US Government standard published as Federal Information Processing Standard 197 (FIPS 197).

According to its designers, the main features of AES are as follows.

(a) Symmetric and Parallel Structure This gives the implementers of the algorithm a lot of flexibility. It also stands up well against cryptanalysis attacks.

(b) Adapted to Modern Processors The algorithm works well with modern processors (Pentium, RISC, parallel).

(c) Suited to Smart Cards The algorithm can work well with smart cards.

Rijndael supports key lengths and plain-text block sizes from 128 bits to 256 bits, in the steps of 32 bits. The key length and the length of the plain-text blocks need to be selected independently. AES mandates that the plain-text block size must be 128 bits, and key size should be 128, 192, or 256 bits. In general, two versions of AES are used: a 128-bit plain-text block combined with a 128-bit key block, and a 128-bit plain text block with a 256-bit key block.

Since the 128-bit plain text block and 128-bit key length are likely to pair as a commercial standard, we will examine that case only. Other principles remain the same. Since 128 bits give a possible key range of 2^{128} or 3×10^{38} keys, Andrew Tanenbaum outlines the strength of this key range in his unimitable style: *Even if NSA manages to build a machine with 1 billion parallel processors, each being able to evaluate one key per picosecond, it would take such a machine about 10^{10} years to search the key space. By then the sun would have burnt out, so the folks then present will have to read the results by candlelight.*

3.9.2 Operation

The basics of Rijndael are in a mathematical concept called **Galois field theory**. For the current discussion, we will keep those concepts out, and try and figure out how the algorithm itself works at a conceptual level.

Similar to the way DES functions, Rijndael also uses the basic techniques of substitution and transposition (i.e. permutation). The key size and the plain-text block size decide how many rounds need to be executed. The minimum number of rounds is 10 (when key size and the plain-text block size are each 128 bits) and the maximum number of rounds is 14. One key differentiator between DES and Rijndael is that all the Rijndael operations involve an entire byte, and not individual bits of a byte. This provides for more optimized hardware and software implementation of the algorithm.

Figure 3.74 describes the steps in Rijndael at a high level.

- (i) Do the following one-time initialization processes:
 - (a) Expand the 16-byte key to get the actual key *block* to be used.
 - (b) Do one time initialization of the 16-byte plain-text block (called *State*).
 - (c) XOR the *state* with the *key block*.
- (ii) For each round, do the following:
 - (a) Apply S-box to each of the plain-text bytes.
 - (b) Rotate row *k* of the plain-text block (i.e. *state*) by *k* bytes.
 - (c) Perform a *mix columns* operation.
 - (d) XOR the *state* with the *key block*

Fig. 3.74 The description of Rijndael

Let us now understand how this algorithm works, step by step.

3.9.3 One-time Initialization Process

We will describe each of the algorithms steps in detail now.

1. Expand the 16-byte key to get the actual key block to be used

The inputs to the algorithm are the key and the plain text, as usual. The key size is 16 bytes in this case. This step expands this 16-byte key into 11 arrays, and each array contains 4 rows and 4 columns. Conceptually, the key-expansion process can be depicted as shown in Fig. 3.75.

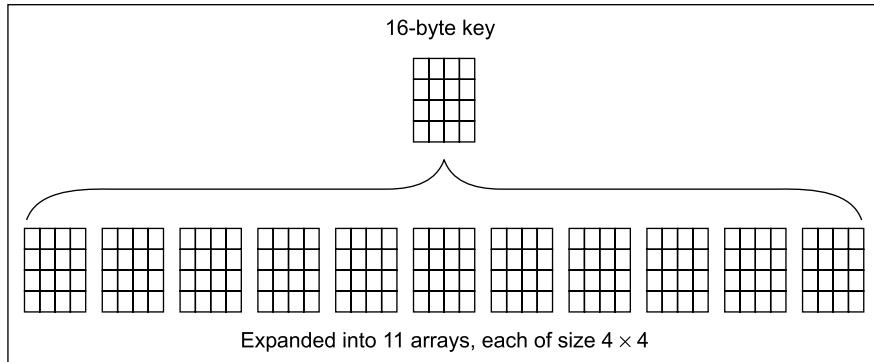


Fig. 3.75 Key expansion: conceptual view

In other words, the original 16-byte key array is expanded into a key containing $11 \times 4 \times 4 = 176$ bytes. One of these, 11 arrays are used in the initialization process and the other 10 arrays are used in the 10 rounds, one array per round.

The key-expansion process is quite complex, and can be safely ignored. We nevertheless describe it below for the sake of completeness. Now, let us start using the terminology of *word*, in the context of AES. A word means 4 bytes. Therefore, in the current context, our 16-byte initial key (i.e. $16/4 = 4$ -word key) will be expanded into 176-byte key (i.e. $176/4$ words, i.e. 44 words).

- (a) Firstly, the original 16-byte key is copied into the first 4 words of the expanded key (i.e. the first 4×4 array of our diagram), as is. This is shown in Fig. 3.76.

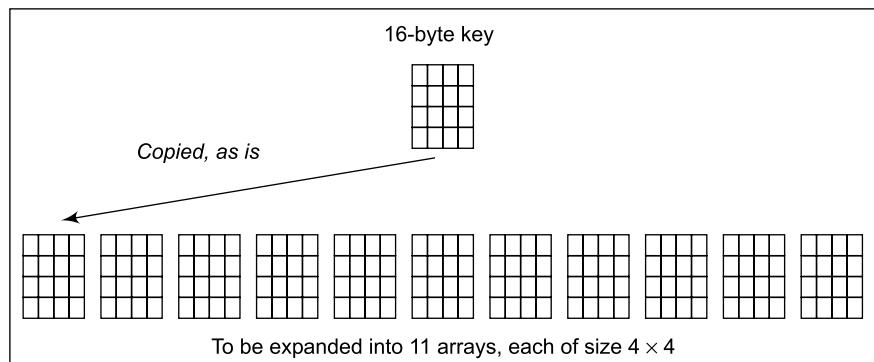


Fig. 3.76 Copying of original key into expanded key

Another representation of the same concept is shown in Fig. 3.77.

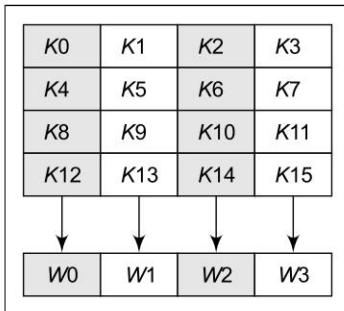


Fig. 3.77 Key expansion: first step

- (b) After filling the first array (for words numbered W_1 to W_3) of the expanded key block as explained above, the remaining 10 arrays (for words numbered W_4 to W_{43}) are filled one by one. Every time, one such 4×4 array (i.e. four words) gets filled. Every added key array block depends on the immediately preceding block and the block 4 positions earlier to it. That is, every added word $w[i]$ depends on $w[i-1]$ and $w[i-4]$. We have said that this fills four words at a time. For filling these four words at a time, the following logic is used:
 - (i) If the word in the w array is a multiple of four, some complex logic is used, explained later below. That is, for words $w[4], w[8], w[12] \dots w[40]$, this complex logic would come into picture.
 - (ii) For others, a simple XOR is used.

This logic is described in an algorithm shown in Fig. 3.78.

```

Expand Key (byte K [16], word W [44]) {
    word tmp;

    // First copy all the 16 input key blocks into first four words of output
    key
    for (i = 0; i < 4; i++) {
        W [i] = K [4*i], K [4*i + 1], K [4*i + 2], K [4*i + 3];
    }

    // Now populate the remaining output key words (i.e. W5 to W43)
    for (i = 4; i < 44; i++) {
        tmp = W [i-1];

        if (i mod 4 == 0)
            tmp = Substitute (Rotate (temp)) XOR Constant [i/4];

        w [i] = w [i-4] XOR tmp;
    }
}
  
```

Fig. 3.78 Key expansion described as an algorithm

We have already discussed the portion of copying all the 16 input key blocks (i.e. 16 bytes) into the first four words of the output key block. Therefore, we will not discuss this again. This is described by the first *for* loop.

In the second *for* loop, we check if the current word being populated in the output key block is a multiple of 4. If it is, we perform three functions, titled *substitute*, *rotate*, and *constant*. We will describe them shortly. However, if the current word in the output key block is not a multiple of four, we simply perform an XOR operation of the previous word and the word four places earlier, and store it as the output word. That is, for word $W[5]$, we would XOR $W[4]$ and $W[1]$ and store their output as $W[5]$. This should be clear from the algorithm. Note that a temporary variable called *tmp* is created, which stores $W[i - 1]$, which is then XORed with $W[i - 4]$.

Let us now understand the three functions, titled *Substitute*, *Rotate*, and *Constant*.

- (i) Function *Rotate* performs a circular left shift on the contents of the word by one byte. Thus, if an input word contains four bytes numbered $[B1, B2, B3, B4]$ then the output word would contain $[B2, B3, B4, B1]$.
- (ii) Function *Substitute* performs a byte substitution on each byte of the input word. For this purpose, it uses an *S-box*, shown in Fig. 3.79.

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

Fig. 3.79 AES S-box

- (iii) In the function *Constant*, the output of the above steps is XORed with a constant. This constant is a word, consisting of 4 bytes. The value of the constant depends on the round number. The last three bytes of a constant word always contain 0. Thus, XORing any input word with such a constant is as good as XORing only with the first byte of the input word. These constant values per round are listed in Fig. 3.80.

Round number	1	2	3	4	5	6	7	8	9	10
Value of constant to be used in Hex	01	02	04	08	10	20	40	80	1B	36

Fig. 3.80 Values of constants (per round) for usage in the *Constant* function

Let us understand how the whole thing works, with an example.

Suppose that our original unexpanded 4-word (i.e. 16-byte) key is as shown in Fig. 3.81.

Byte position (Decimal)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value (Hex)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F

Fig. 3.81 Key-expansion example—Step 1—Original 4-word key

- (i) In the first four rounds, the original 4-word input key would be copied into the first 4-word output key, as per our algorithm step as follows:

// First copy all the 16 input key blocks into first four words of output key

```
for (i = 0; i < 4; i++) {
```

```
    W[i] = K[4*i], K[4*i + 1], K[4*i + 2], K[4*i + 3];
```

```
}
```

Thus, the first 4 words of the output key (i.e. from $W[0]$ to $W[3]$) would contain values as shown in Fig. 3.82. This is constructed from the input 4-word (i.e. 16-byte) key as follows:

- Firstly, the first four bytes, i.e. one word, of the input key, namely 00 01 02 03 is copied into the first word of the output key $W[0]$.
- Now, the next four bytes of the input key, i.e. 04 05 06 07 would be copied into the second word of the output key, i.e. $W[1]$.

Needless to say, $W[2]$ and $W[3]$ would get populated with the remaining contents of the input key bytes, as shown in the figure.

$W[0]$	$W[1]$	$W[2]$	$W[3]$	$W[4]$	$W[5]$...	$W[44]$
00 01 02 03	04 05 06 07	08 09 0A 0B	0C 0D 0E 0F	?	?	?	?

Fig. 3.82 Key-expansion example—Step 2 (filling of first 4 output key words)

- (ii) Now let us understand how the next word of the output key block, i.e. $W[4]$ is populated. For this purpose, the following algorithm would be executed:

// Now populate the remaining output key words (i.e. W_5 to W_{43})

```
for (i = 4; i < 44; i++) {
```

```
    tmp =  $W[i - 1]$ ;
```

```
    if ( $i \bmod 4 == 0$ )
```

```
        tmp = Substitute (Rotate (temp)) XOR Constant [ $i/4$ ];
```

```
     $w[i] = w[i - 4]$  XOR tmp;
```

```
}
```

Based on this, we will have the following:

$$\text{tmp} = W[i - 1] = W[4 - 1] = W[3] = 0C\ 0D\ 0E\ 0F$$

Since $i = 4$, $i \bmod 4$ is 0. Therefore, we will now have the following step:

$$\text{tmp} = \text{Substitute}(\text{Rotate}(\text{tmp})) \text{ XOR Constant}[i/4];$$

- We know that *Rotate* (*temp*) will produce *Rotate* (*0C 0D 0E 0F*), which equals *0D 0E 0F 0C*.
- Now, we need to do *Substitute* (*Rotate* (*temp*)). For this purpose, we need to take one byte at a time, and look up in the S-box for substitution. For example, our first byte is *0D*. Looking it up in the S-box with $x = 0$ and $y = D$ produces *D7*. Similarly, *0E* produces *AB*, *0F* produces *76*, and *0C* produces *FE*. Thus, at the end of the *Substitute* (*Rotate* (*temp*)) step, our input of *0D 0E 0F 0C* is transformed into the output of *D7 AB 76 FE*.
- We now need to XOR this value with *Constant* [$i/4$]. Since $i = 4$, we need to obtain the value of *Constant* [$4/4$], i.e. *Constant* [1], which is 01, as per our earlier table of constants. As we know, we need to pad this with three more bytes, all set to 00. Therefore, our constant value is 01 00 00 00. Therefore, we have:

D7 AB 76 FE

XOR

10 00 00 00

= D6 AB 76 FE

Thus, our new *tmp* value is *D6 AB 76 FE*.

- Finally, we need to XOR this *tmp* value with *W* [$i - 4$], i.e. with *W* [$4 - 4$], i.e. with *W* [0]. Thus, we have:

D6 AB 76 FE

XOR

00 01 02 03

= D6 AA 74 FD

Thus, *W* [4] = *D6 AA 74 FD*.

We can use the same logic to derive the remaining expanded key blocks (*W* [5] to *W* [44]).

2. Do one-time initialization of the 16-byte plain-text block (called State)

This step is relatively simple. Here, the 16-byte plain-text block is copied into a two-dimensional 4×4 array called *state*. The order of copying is in the column order. That is, the first four bytes of the plain-text block get copied into the first column of the *state* array, the next four bytes of the plain-text block get copied into the second column of the *state* array, and so on. This is shown in Fig. 3.83 for every byte (numbered from *B1* to *B16*).

3. XOR the state with the key block

Now, the first 16 bytes (i.e. four words *W* [0], *W* [1], *W* [2], and *W* [3]) of the expanded key are XORed into the 16-byte *state* array (*B1* to *B16* shown above). Thus, every byte in the *state* array is replaced by the XOR of itself and the corresponding byte in the expanded key.

At this stage, the initialization is complete, and we are ready for rounds.

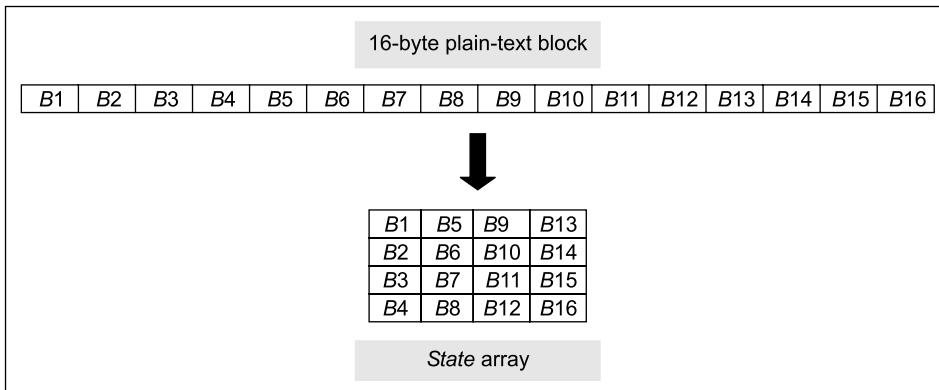


Fig. 3.83 Copying of the input text block into *State* array

3.9.4 Processes in Each Round

The following steps are executed 10 times, one per round.

1. Apply S-box to each of the plain-text bytes

This step is very straightforward. The contents of the *state* array are looked up into the *S*-box. Byte-by-byte substitution is done to replace the contents of the *state* array with the respective entries in the *S*-box. Note that only one *S*-box is used, unlike DES, which has multiple *S*-boxes.

2. Rotate row k of the plain-text block (i.e. state) by k bytes

Here, each of the four rows of the *state* array are rotated to the left. Row 0 is rotated 0 bytes (i.e. not rotated at all), row 1 is rotated by 1 byte, row 2 is rotated 2 bytes, and row 2 is rotated 3 bytes. This helps in diffusion of data. Thus, if the original 16 bytes of the *state* array contain values 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 then the rotate operation would change the values as shown below.

Original array	Modified array
1 5 9 13	1 5 9 13
2 6 10 14	6 10 14 2
3 7 11 15	11 15 3 7
4 8 12 16	16 4 8 12

3. Perform a mix-columns operation

Now, each column is mixed independent of the other. Matrix multiplication is used. The output of this step is the matrix multiplication of the old values and a constant matrix.

This is perhaps the most complex step to understand and also to explain. Nevertheless, we will make an attempt! This is based on an illuminating article by Adam Berent:

There are two aspects of this step. The first explains which parts of the state are multiplied against which parts of the matrix. The second explains how this multiplication is implemented over what's called a **Galois field**.

(a) Matrix Multiplication We know that the state is arranged into a 4×4 matrix.

The multiplication is performed one column at a time (i.e. on 4 bytes at a time). Each value in the column is eventually multiplied against every value of the matrix (i.e. 16 total multiplications are performed). The results of these multiplications are XORed together to produce only 4 resulting bytes for the next state. We together have 4 bytes of input, 16 multiplications, 12 XORs, and 4 bytes of output. The multiplication is performed one matrix row at a time against each value of a *state* column.

For example, consider that our matrix is as shown in Fig. 3.84.

Next, let us imagine that the 16-byte state array is as shown in Fig. 3.85.

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

Fig. 3.84 Multiplication matrix

b1	b5	b9	b13
b2	b6	b10	b14
b3	b7	b11	b15
b4	b8	b12	b16

Fig. 3.85 16-byte state array

The first result byte is calculated by multiplying four values of the *state* column with the four values of the first row of the matrix. The result of each multiplication is then XORed to produce one byte. For this, the following calculation is used:

$$b1 = (b1 * 2) \text{ XOR } (b2 * 3) \text{ XOR } (b3 * 1) \text{ XOR } (b4 * 1)$$

Next, the second result byte is calculated by multiplying the same four values of the *state* column against four values of the second row of the matrix. The result of each multiplication is then XORed to produce one byte.

$$b2 = (b1 * 1) \text{ XOR } (b2 * 2) \text{ XOR } (b3 * 3) \text{ XOR } (b4 * 1)$$

The third result byte is calculated by multiplying the same four values of the *state* column against four values of the third row of the matrix. The result of each multiplication is then XORed to produce one byte.

$$b3 = (b1 * 1) \text{ XOR } (b2 * 1) \text{ XOR } (b3 * 2) \text{ XOR } (b4 * 3)$$

The fourth result byte is calculated by multiplying the same four values of the *state* column against four values of the fourth row of the matrix. The result of each multiplication is then XORed to produce one byte.

$$b4 = (b1 * 3) \text{ XOR } (b2 * 1) \text{ XOR } (b3 * 1) \text{ XOR } (b4 * 2)$$

This procedure is repeated again with the next column of the *state*, until there are no more state columns.

To summarize:

The first column will include *state* bytes 1-4 and will be multiplied against the matrix in the following manner:

$$b1 = (b1 * 2) \text{ XOR } (b2*3) \text{ XOR } (b3*1) \text{ XOR } (b4*1)$$

$$b2 = (b1 * 1) \text{ XOR } (b2*2) \text{ XOR } (b3*3) \text{ XOR } (b4*1)$$

$$b3 = (b1 * 1) \text{ XOR } (b2*1) \text{ XOR } (b3*2) \text{ XOR } (b4*3)$$

$$b4 = (b1 * 3) \text{ XOR } (b2*1) \text{ XOR } (b3*1) \text{ XOR } (b4*2)$$

($b1$ = specifies the first byte of the state)

The second column will be multiplied against the second row of the matrix in the following manner.

$$b5 = (b5 * 2) \text{ XOR } (b6*3) \text{ XOR } (b7*1) \text{ XOR } (b8*1)$$

$$b6 = (b5 * 1) \text{ XOR } (b6*2) \text{ XOR } (b7*3) \text{ XOR } (b8*1)$$

$$b7 = (b5 * 1) \text{ XOR } (b6*1) \text{ XOR } (b7*2) \text{ XOR } (b8*3)$$

$$b8 = (b5 * 3) \text{ XOR } (b6*1) \text{ XOR } (b7*1) \text{ XOR } (b8*2)$$

And so on until all columns of the *state* are exhausted.

(b) Galois Field Multiplication The multiplication mentioned above is performed over a Galois field. The mathematics behind this is quite complex and beyond the scope of the current text. However, we will discuss the implementation of the multiplication, which can be done quite easily with the use of the following two tables, shown in hexadecimal formats. Refer to Fig. 3.86 and Fig. 3.87.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	01	03	05	0F	11	33	55	FF	1A	2E	72	96	A1	F8	13	35
1	5F	E1	38	48	D8	73	95	A4	F7	02	06	0A	1E	22	66	AA
2	E5	34	5C	E4	37	59	EB	26	6A	BE	D9	70	90	AB	E6	31
3	53	F5	04	0C	14	3C	44	CC	4F	D1	68	B8	D3	6E	B2	CD
4	4C	D4	67	A9	E0	3B	4D	D7	62	A6	F1	08	18	28	78	88
5	83	9E	B9	D0	6B	BD	DC	7F	81	98	B3	CE	49	DB	76	9A
6	B5	C4	57	F9	10	30	50	F0	0B	1D	27	69	BB	D6	61	A3
7	FE	19	2B	7D	87	92	AD	EC	2F	71	93	AE	E9	20	60	A0
8	FB	16	3A	4E	D2	6D	B7	C2	5D	E7	32	56	FA	15	3F	41
9	C3	5E	E2	3D	47	C9	40	C0	5B	ED	2C	74	9C	BF	DA	75
A	9F	BA	D5	64	AC	EF	2A	7E	82	9D	BC	DF	7A	8E	89	80
B	9B	B6	C1	58	E8	23	65	AF	EA	25	6F	B1	C8	43	C5	54
C	FC	1F	21	63	A5	F4	07	09	1B	2D	77	99	B0	CB	46	CA
D	45	CF	4A	DE	79	8B	86	91	A8	E3	3E	42	C6	51	F3	0E
E	12	36	5A	EE	29	7B	8D	8C	8F	8A	85	94	A7	F2	0D	17
F	39	4B	DD	7C	84	97	A2	FD	1C	24	6C	B4	C7	52	F6	01

Fig. 3.86 E-table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	19	01	32	02	1A	C6	4B	C7	1B	68	33	EE	DF	03	
1	64	04	E0	0E	34	8D	81	EF	4C	71	08	C8	F8	69	1C	C1
2	7D	C2	1D	B5	F9	B9	27	6A	4D	E4	A6	72	9A	C9	09	78
3	65	2F	8A	05	21	0F	E1	24	12	F0	82	45	35	93	DA	8E
4	96	8F	DB	BD	36	D0	CE	94	13	5C	D2	F1	40	46	83	38
5	66	DD	FD	30	BF	06	8B	62	B3	25	E2	98	22	88	91	10
6	7E	6E	48	C3	A3	B6	1E	42	3A	6B	28	54	FA	85	3D	BA
7	2B	79	0A	15	9B	9F	5E	CA	4E	D4	AC	E5	F3	73	A7	57
8	AF	58	A8	50	F4	EA	D6	74	4F	AE	E9	D5	E7	E6	AD	E8
9	2C	D7	75	7A	EB	16	0B	F5	59	CB	5F	B0	9C	A9	51	A0
A	7F	0C	F6	6F	17	C4	49	EC	D8	43	1F	2D	A4	76	7B	B7
B	CC	BB	3E	5A	FB	60	B1	86	3B	52	A1	6C	AA	55	29	9D
C	97	B2	87	90	61	BE	DC	FC	BC	95	CF	CD	37	3F	5B	D1
D	53	39	84	3C	41	A2	6D	47	14	2A	9E	5D	56	F2	D3	AB
E	44	11	92	D9	23	20	2E	89	B4	7C	B8	26	77	99	E3	A5
F	67	4A	ED	DE	C5	31	FE	18	0D	63	8C	80	C0	F7	70	07

Fig. 3.87 L-table

The result of the multiplication is actually the output of a look-up of the **L**-table, followed by the addition of the results, followed by a look-up of the **E**-table. Note that the term *addition* means a traditional mathematical addition, and not a bitwise AND operation.

All numbers being multiplied using the *Mix Column* function converted to HEX will form a maximum of a 2-digit hex number. We use the first digit in the number on the vertical index and the second number on the horizontal index. If the value being multiplied is composed of only one digit, we use 0 on the vertical index. For example if the two hex values being multiplied are *AF* * 8 we first look up **L** (*AF*) index which returns *B7* and then lookup **L** (08) which returns *4B*. Once the L-table look-up is complete, we can then simply add the numbers together. The only trick being that if the addition result is greater than *FF*, we subtract *FF* from the addition result. For example *B7* + *4B* = 102. Because 102 > *FF*, we perform: 102 – *FF* which gives us 03.

The last step is to look up the addition result on the E-table. Again, we take the first digit to look up the vertical index and the second digit to look up the horizontal index.

For example, *E*(03) = 0F. Therefore, the result of multiplying *AF* * 8 over a Galois field is 0F.

4. XOR the state with the key block

This step XORs the key for this round into the *state* array.

For decryption, the process can be executed in the reverse order. There is another option, though. The same encryption process, run with some different table values, can also perform decryption.

■ CASE STUDY: SECURE MULTIPARTY CALCULATION ■

Points for Classroom Discussions

1. Can you think of any practical situations where secure multiparty calculations would be required?

2. *Can symmetric key encryption alone suffice the needs of secure multiparty calculations? If yes, what are the possible issues/constraints?*
3. *Is an arbitrator mandatory in such a scheme?*

Suppose that we have the following problem:

Alice, Bob, Carol, and Dave are four people working in an organization. One fine day, they are interested in knowing their average salary. However, they want to (obviously) ensure that no one comes to know about the salary of anyone else. Unfortunately, there is no arbitrator, who can take this task upon itself.

How can this be achieved? An interesting protocol can be used to fulfill these requirements, as follows:

1. Alice generates a random number, adds that number to her salary, encrypts the resulting value with the public key of Bob and sends it to Bob.
2. Bob decrypts the information received from Alice with his private key. He adds his salary to the decrypted number (which is Alice's salary + random number). He then encrypts it with the public key of Carol and sends the result to Carol.
3. Carol decrypts the value received from Bob with her private key, adds her salary to it, encrypts the result with Dave's public key and sends the result to Dave.
4. Dave decrypts the value received from Carol with his private key, adds his salary to it, encrypts the result with Alice's public key and sends the result to Dave.
5. Alice decrypts the value received from Dave with her private key and subtracts the original random number from it. This gives her the total salary.
6. Alice divides the total salary by the number of people (4). This produces the value of the average salary, which she announces to Bob, Carol and Dave.



Summary

- In symmetric-key cryptography, the sender and the receiver share a single key, and the same key is used for both encryption and decryption.
- Some of the important symmetric-key cryptographic algorithms are DES (and its variations), IDEA, RC4, RC5 and Blowfish.
- An algorithm type defines what size of plain text should be encrypted in each step of the algorithm.
- There are two main types of algorithms: stream cipher and block cipher.
- In a stream cipher, each bit/byte of text is encrypted/decrypted individually.
- In a block cipher, a block of text is encrypted/decrypted at a time.
- An algorithm mode defines the details of the cryptographic algorithm, once the type is decided.
- Confusion is a technique of ensuring that a cipher text gives no clue about the original plain text.
- Diffusion increases the redundancy of the plain text by spreading it across rows and columns.
- There are four important algorithm modes: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB).

- There is a variation of the OFB mode, called Counter (CTR).
- Electronic Code Book (ECB) is the simplest mode of operation. Here, the incoming plain-text message is divided into blocks of 64 bits each. Each such block is then encrypted independently of the other blocks. For all blocks in a message, the same key is used for encryption.
- The Cipher Block Chaining (CBC) mode ensures that even if a block of plain text repeats in the input, these two (or more) identical plain-text blocks yield totally different cipher-text blocks in the output. For this, a feedback mechanism is used.
- The Cipher Feedback (CFB) mode encrypts data in units that are smaller (e.g. they could be of size 8 bits, i.e. the size of a character typed by an operator) than a defined block size (which is usually 64 bits).
- The Output Feedback (OFB) mode is extremely similar to the CFB. The only difference is that in the case of CFB, the cipher text is fed into the next stage of encryption process. But in the case of OFB, the output of the Initial Vector (IV) encryption process is fed into the next stage of encryption process.
- The Counter (CTR) mode is quite similar to the OFB mode, with one variation. It uses sequence numbers called counters as the inputs to the algorithm. After each block is encrypted, to fill the register, the next counter value is used.
- There are two problems in symmetric-key encryption: problem of key exchange, and one key is required per communication pair.
- The Data Encryption Standard (DES), also called the Data Encryption Algorithm (DEA) by ANSI and DEA-1 by ISO, has been a cryptographic algorithm used for over two decades. Of late, DES has been found vulnerable against very powerful attacks, and therefore, the popularity of DES has been slightly on the decline.
- DES is a block cipher. It encrypts data in blocks of size 64 bits each. That is, 64 bits of plain text goes as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.
- In 1990, Eli Biham and Adi Shamir introduced the concept of differential cryptanalysis. This method looks at pairs of cipher text whose plain texts have particular differences. The technique analyses the progress of these differences as the plain texts travel through the various rounds of DES.
- Invented by Mitsuru Matsui, the linear cryptanalysis attack is based on linear approximations. If we XOR some plain text bits together, XOR some cipher text bits together and then XOR the result, we will get a single bit, which is the XOR of some of the key bits.
- Timing attacks refer more to asymmetric-key cryptography. However, they can also apply to symmetric key cryptography. The idea is simple: observe how long it takes for the cryptographic algorithm to decrypt different blocks of cipher text. The idea is to try and obtain either the plain text or the key used for encryption by observing these timings.
- Two main variations of DES have emerged, which are double DES and triple DES.
- Double DES is quite simple to understand. Essentially, it does twice what DES normally does only once. Double DES uses two keys.
- Merkle and Hellman introduced the concept of the meet-in-the-middle attack. This attack involves encryption from one end, decryption from the other, and matching the results in the middle, hence the name meet-in-the-middle attack.

- Triple DES is DES three times. It comes in two kinds: one that uses three keys, and the other that uses two keys.
- The International Data Encryption Algorithm (IDEA) is perceived as one of the strongest cryptographic algorithms.
- IDEA is a block cipher. Like DES, it also works on 64-bit plain-text blocks. The key is longer, however, and consists of 128 bits. IDEA is reversible like DES, that is, the same algorithm is used for encryption and decryption. Also, IDEA uses both diffusion and confusion for encryption.
- RC4 was designed by Ron Rivest of RSA Security in 1987. The official name for this algorithm is “Rivest Cipher 4”. However, because of its ease of reference, the acronym RC4 has stuck.
- RC4 is a stream cipher. This means that the encryption happens byte by byte. However, this can be changed to bit-by-bit encryption (or to a size other than a byte/bit).
- RC5 is a symmetric-key block encryption algorithm developed by Ron Rivest. The main features of RC5 are that it is quite fast as it uses only the primitive computer operations (such as addition, XOR, shift, etc). It allows for a variable number of rounds, and a variable bit-size key to add to the flexibility.
- Blowfish was developed by Bruce Schneier, and has the reputation of being a very strong symmetric-key cryptographic algorithm.
- Blowfish encrypts 64-bit blocks with a variable-length key.
- In the 1990s, the US Government wanted to standardize a cryptographic algorithm, which was to be used universally by them. It was to be called the Advanced Encryption Standard (AES). Many proposals were submitted, and after a lot of debate, an algorithm called Rijndael was accepted.
- Rijndael supports key lengths and plain-text block sizes from 128 bits to 256 bits, in the steps of 32 bits. The key length and the length of the plain-text blocks need to be selected independently. AES mandates that the plain text block size must be 128 bits, and key size should be 128, 192, or 256 bits. In general, two versions of AES are used: 128-bit plain-text block combined with 128-bit key block, and 128-bit plain-text block with 256-bit key block.



Key Terms and Concepts

- Advanced Encryption Standard (AES)
- Algorithm type
- Chaining mode
- Cipher Feedback (CFB)
- Counter mode
- Differential cryptanalysis
- Double DES
- Galois field
- Initialization Vector (IV)
- Algorithm mode
- Block cipher
- Cipher Block Chaining (CBC)
- Confusion
- Data Encryption Standard (DES)
- Diffusion
- Electronic Code Book (ECB)
- Group
- International Data Encryption Algorithm (IDEA)

- Keystream
- Lucifer
- Output Feedback (OFB)
- RC5
- Stream cipher
- Triple DES
- Linear cryptanalysis
- Meet-in-the-middle attack
- RC4
- Rijndael
- Timing attacks



PRACTICE SET

■ Multiple-Choice Questions

1. In _____, one bit of plain text is encrypted at a time.
 - (a) stream cipher
 - (b) block cipher
 - (c) both stream and block cipher
 - (d) none of the above
2. In _____, one block of plain text is encrypted at a time.
 - (a) stream cipher
 - (b) block cipher
 - (c) both stream and block cipher
 - (d) none of the above
3. _____ increases the redundancy of plain text.
 - (a) confusion
 - (b) diffusion
 - (c) both confusion and diffusion
 - (d) neither confusion nor diffusion
4. _____ works on block mode.
 - (a) CFB
 - (b) OFB
 - (c) CCB
 - (d) CBC
5. DES encrypts blocks of _____ bits.
 - (a) 32
 - (b) 56
 - (c) 64
 - (d) 128
6. There are _____ rounds in DES.
 - (a) 8
 - (b) 10
 - (c) 14
 - (d) 16
7. _____ is based on the IDEA algorithm.
 - (a) S/MIME
 - (b) PGP
 - (c) SET
 - (d) SSL
8. The actual algorithm in the AES encryption scheme is _____.
 - (a) Blowfish
 - (b) IDEA
 - (c) Rijndael
 - (d) RC4
9. The Blowfish algorithm executes the _____ algorithm for subkey generation.
 - (a) Blowfish
 - (b) IDEA
 - (c) Rijndael
 - (d) RC4
10. In AES, the 16-byte key is expanded into _____.
 - (a) 200 bytes
 - (b) 78 bytes
 - (c) 176 bytes
 - (d) 184 bytes

11. In IDEA, the key size is _____.
(a) 128 bytes
(b) 128 bits
(c) 256 bytes
(d) 256 bits
12. In the _____ algorithm, once the initial key of 1-256 bytes is used to create a transformed key, the original key is discarded.
(a) Blowfish
(b) IDEA
(c) Rijndael
(d) RC4
13. The minimum number of rounds recommended in RC5 is _____.
(a) 8
14. The RC5 block cipher mode is also called _____.
(a) ECB
(b) RC5-CBC
(c) RC5-CBC-Pad
(d) RC5-CTS
15. The _____ step ensures that plain text is not vulnerable in block cipher mode.
(a) encryption
(b) round
(c) initial
(d) chaining

■ Exercises

1. Distinguish between stream and block ciphers.
2. Discuss the idea of algorithm modes with detailed explanation of at least two of them.
3. Write a note on the security and possible vulnerabilities of the various algorithm modes.
4. What is an Initialization Vector (IV)? What is its significance?
5. What are the problems with symmetric key encryption?
6. What is the idea behind meet-in-the-middle attack?
7. Explain the main concepts in DES.
8. How can the same key be reused in triple DES?
9. Explain the principles of the IDEA algorithm.
10. Distinguish between differential and linear cryptanalysis.
11. Explain the subkey generation in the Blowfish algorithm.
12. Explain the usage of the S array in the case of the RC4 algorithm.
13. Discuss how encryption happens in RC5.
14. How does the one-time initialization step work in AES?
15. Explain the steps in the various rounds of AES.

■ Design/Programming Exercises

1. Write a C program to implement the DES algorithm logic.
2. Write the same program as in step 1, in Java.
3. Write a Java program that contains functions, which accept a key and input text to be encrypted/decrypted. This program should use the key to encrypt/decrypt the input by using the triple DES algorithm. Make use of Java cryptography package.

4. Write a C program to implement the Blowfish algorithm.
5. Write the same program in Visual Basic.
6. Investigate Rijndael further and write a C program to implement the same.
7. Find out more about the vulnerabilities of DES from the Internet.
8. Write the RC4 logic in Java.
9. Try to implement the logic of the various algorithm modes in a programming language of your choice.
10. Create a visual tool that should take as input some plain text and a key, and then execute an algorithm of the user's choice (e.g. DES). It should provide a visual display of the output at each stage of the encryption process.
11. Using Java cryptography, encrypt the text "Hello world" using Blowfish. Create your own key using Java *keytool*.
12. Perform the same task by using the cryptography API in .NET.
13. Examine which software products in real life use which cryptographic algorithms. Try to find out the reasons behind the choice of these algorithms.
14. Implement DES-2 and DES-3 (with two keys) using Java cryptography and try to encrypt a large block of text. Find out if there is any significant performance difference.
15. Compare the results of the above with DES-3 (with three keys).



COMPUTER-BASED ASYMMETRIC-KEY CRYPTOGRAPHY ALGORITHMS

■ 4.1 INTRODUCTION ■

Symmetric-key cryptography is fast and efficient. However, it suffers from a big disadvantage of the problem of key exchange. The sender and the receiver of an encrypted message use the same key in symmetric-key cryptography, and it is very tough to agree upon a common key without letting anyone else know about it. Asymmetric-key cryptography solves this problem. Here, each communicating party uses two keys to form a key pair—one key (the private key) remains with the party, and the other key (the public key) is shared with everybody.

This chapter covers the asymmetric-key cryptography technology in detail. We examine the history behind this and also how using mechanisms of key wrapping and digital envelopes can optimize this. We then go on to study the concepts of message digest (hash). The various message-digest algorithms are explained in minute detail. We also study variations of this, called MAC.

Digital signatures have gained a lot of prominence in the last few years. They have assumed legal status in many countries, and others are following suit. Digital-signature technology is also covered with explanations of the appropriate algorithms.

This chapter also briefly introduces a few other asymmetric-key cryptographic algorithms.

■ 4.2 BRIEF HISTORY OF ASYMMETRIC-KEY CRYPTOGRAPHY ■

We have discussed the problem of key exchange, also called **key distribution** or **key agreement**, at great length. To revise it quickly, in any symmetric-key cryptographic scheme, the main issue is: How can the sender and the receiver of a message decide upon the key to be used for encryption and decryption? In computer-based cryptographic algorithms, this problem is even more serious, because the sender and the receiver may be in different countries. For example, suppose that the seller of some

goods has set up an online shopping Web site. A customer residing in a different country wants to place an order (in an encrypted form, so as to maintain confidentiality, for whatever reasons) over the Internet. How would the customer (i.e. the customer's computer: we shall use these terms interchangeably) encrypt the order details before sending them to the seller? Which key would he/she use? How would the customer then inform the seller about the key, so that the seller can decrypt the message? Remember that the encryption and decryption must be done using the same key. As we have seen, in symmetric-key cryptography, this problem of key exchange cannot be solved. Moreover, we need a unique key per communicating party. That is also quite cumbersome and undesired.

In the mid-1970s, Whitfield Diffie, a student at the Stanford University met with Martin Hellman, his professor, and the two began to think about the problem of key exchange. After some research and complicated mathematical analysis, they came up with the idea of asymmetric-key cryptography. Many experts believe that this development is the first—and perhaps the only—truly revolutionary concept in the history of cryptography. Diffie and Hellman can, therefore, be regarded as the fathers of asymmetric-key cryptography.

However, there is a lot of debate regarding who should get the credit for developing asymmetric-key cryptography. It is believed that James Ellis of the British *Communications Electronic Security Group (CSEG)* proposed the idea of asymmetric-key cryptography in the 1960s. His ideas were based on an anonymous paper written at the Bell Labs during the Second World War. However, Ellis could not devise a practical algorithm based on his ideas. He then met with Clifford Cocks, who joined the CSEG in 1973. After a short discussion between Ellis and Cocks, Cocks came up with a practical algorithm that could work! Next year, Malcolm Williamson, another employee at the CSEG, developed an asymmetric-key cryptographic algorithm. However, since the CSEG was a secret agency, these findings were never published, and therefore, it is believed that these people never got the credit that they deserved.

Simultaneously, the US *National Security Agency (NSA)* was also working on asymmetric-key cryptography. It is believed that the NSA system based on the asymmetric-key cryptography was operational in the mid-1970s.

Based on the theoretical framework of Diffie and Hellman, in 1977, Ron Rivest, Adi Shamir and Len Adleman at MIT developed the first major asymmetric-key cryptography system, and published their results in 1978. This method is called **RSA algorithm**. The name RSA comes from the first letters of the surnames of the three researchers. Actually, Rivest was working as a professor at the MIT. He had recruited Shamir and Adleman to work on the concept of asymmetric-key cryptography.

Even today, RSA is the most widely accepted public-key solution. It solves the problem of key agreements and distribution. As we mentioned, the approach used here is that each communicating party possesses a key pair, made up of one public key and one private key.

To communicate securely over any network, all one needs to do is to publish one's public key. All these public keys can then be stored in a database that anyone can consult. However, the private key only remains with the respective individuals.

■ 4.3 AN OVERVIEW OF ASYMMETRIC-KEY CRYPTOGRAPHY ■

In *asymmetric-key cryptography*, also called **public key cryptography**, two *different* keys (which form a *key pair*) are used. One key is used for encryption and only the other corresponding key must be used

for decryption. No other key can decrypt the message—not even the original (i.e. the first) key used for encryption! The beauty of this scheme is that every communicating party needs just a key pair for communicating with any number of other communicating parties. Once someone obtains a key-pair, he/she can communicate with anyone else.

There is a simple mathematical basis for this scheme. If you have an extremely large number that has only two factors, which are prime numbers, you can generate a pair of keys. For example, consider a number 10. The number 10 has only two factors, 5 and 2. If you apply 5 as an encryption factor, only 2 can be used as the decryption factor. Nothing else—not even 5 itself—can do the decryption. Of course, 10 is a very small number. Therefore, with minimal effort, this scheme can be broken into. However, if the number is very large, even years of computation cannot break the scheme.

One of the two keys is called the *public key* and the other is the *private key*. Let us assume that you want to communicate over a computer network such as the Internet in a secure manner. You would need to obtain a public key and a private key. We shall study later how these keys can be obtained. The private key remains with you as a secret. You must not disclose your private key to anybody. However, the public key is for the general public. It is disclosed to all parties that you want to communicate with. In this scheme, in fact, each party or node publishes its public key. Using this, a directory can be constructed where the various parties or nodes (i.e. their ids) and the corresponding public keys are maintained. One can consult this and get the public key for any party that one wishes to communicate with by a simple table search.

Suppose *A* wants to send a message to *B* without having to worry about its security. Then, *A* and *B* should each have a private key and a public key.

- *A* should keep her private key secret.
- *B* should keep her private key secret.
- *A* should inform *B* about her public key.
- *B* should inform *A* about her public key.

Thus, we have a matrix as shown in Fig. 4.1.

Key details	<i>A</i> should know	<i>B</i> should know
<i>A</i> 's private key	Yes	No
<i>A</i> 's public key	Yes	Yes
<i>B</i> 's private key	No	Yes
<i>B</i> 's public key	Yes	Yes

Fig. 4.1 Matrix of private and public Keys

Armed with this knowledge, asymmetric-key cryptography works as follows:

1. When *A* wants to send a message to *B*, *A* encrypts the message using *B*'s public key. This is possible because *A* knows *B*'s public key.
2. *A* sends this message (which was encrypted with *B*'s public key) to *B*.
3. *B* decrypts *A*'s message using *B*'s private key. Note that only *B* knows about her private key. Also note that the message can be decrypted only by *B*'s private key and nothing else! Thus, no one else can make any sense out of the message even if one can manage to intercept the message. This is

because the intruder (ideally) does not know about B 's private key. It is only B 's private key that can decrypt the message.

This is shown in Fig. 4.2.

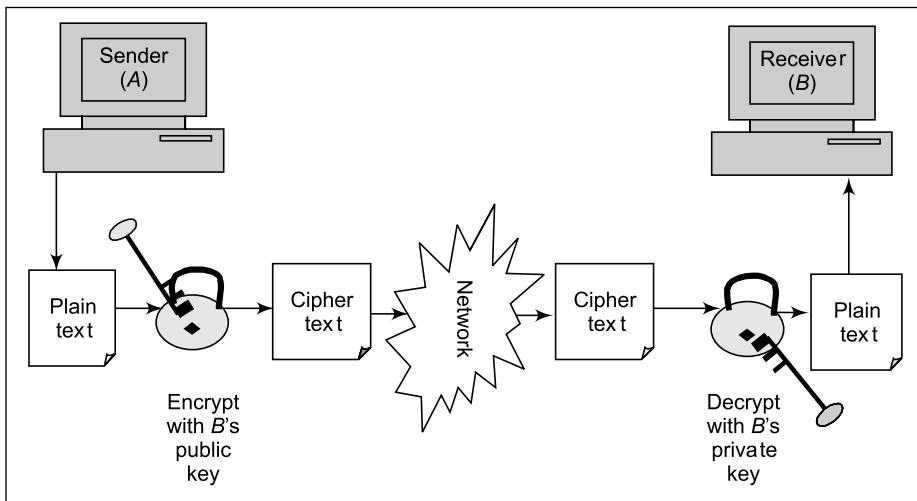


Fig. 4.2 Asymmetric-key cryptography

Similarly, when B wants to send a message to A , exactly reverse steps take place. B encrypts the message using A 's public key. Therefore, only A can decrypt the message back to its original form, using her private key.

We can consider a practical situation that describes asymmetric-key cryptography as used in real life. Suppose a bank accepts many requests for transactions from its customers over an insecure network. The bank can have a private-key–public key pair. The bank can publish its public key to all its customers. The customers can use this public key of the bank for encrypting messages before they send them to the bank. The bank can decrypt all these encrypted messages with its private key, which remains with itself. As we know, only the bank can perform the decryption, as it alone knows its private key. This concept is shown in Fig. 4.3. We do not show the details of the encryption and decryption processes explicitly, and assume their presence.

■ 4.4 THE RSA ALGORITHM ■

4.4.1 Introduction

The RSA algorithm is the most popular and proven asymmetric-key cryptographic algorithm. Before we discuss that, let us have a quick overview of prime numbers, as they form the basis of the RSA algorithm.

A prime number is the one that is divisible only by 1 and itself. For instance, 3 is a prime number, because it can be divided only by 1 or 3. However, 4 is not a prime number, because other than by 1 and 4, it can also be divided by 2. Similarly, 5, 7, 11, 13, 17, ... are prime numbers, whereas 6, 8, 9, 10, 12,

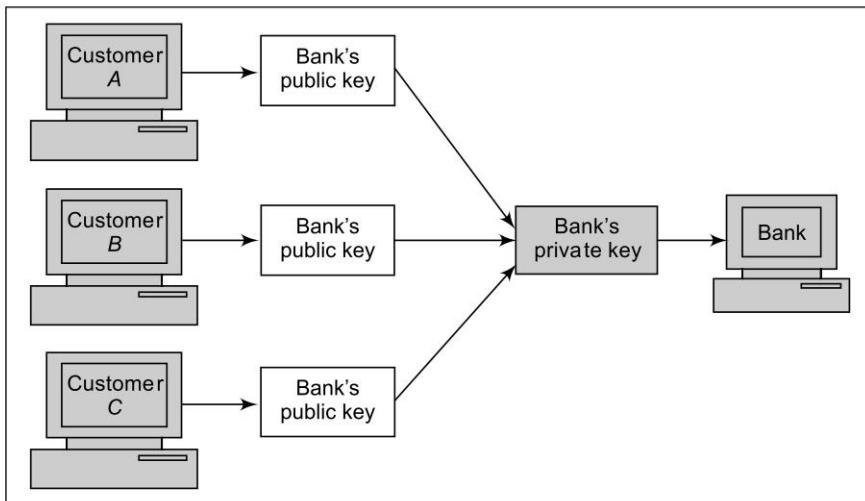


Fig. 4.3 Use of a public-key–private-key pair by a bank

... are non-prime numbers. A quick observation can also be made that a prime number above 2 must be an odd number (because all even numbers are divisible by 2, and therefore, all even numbers from 4 onwards are non-prime).

The RSA algorithm is based on the mathematical fact that it is easy to find and multiply large prime numbers together, but it is extremely difficult to factor their product. The private and public keys in RSA are based on very large (made up of 100 or more digits) prime numbers. The algorithm itself is quite simple (unlike the symmetric-key cryptographic algorithms). However, the real challenge in the case of RSA is the selection and generation of the public and private keys.

Let us now understand how the public and private keys are generated, and using them, how we can perform encryption and decryption in RSA. The whole process is shown in Fig. 4.4.

1. Choose two large prime numbers P and Q .
2. Calculate $N = P \times Q$.
3. Select the public key (i.e. the encryption key) E such that it is not a factor of $(P - 1)$ and $(Q - 1)$.
4. Select the private key (i.e. the decryption key) D such that the following equation is true:

$$(D \times E) \bmod (P - 1) \times (Q - 1) = 1$$
5. For encryption, calculate the cipher text CT from the plain text PT as follows:

$$CT = PT^E \bmod N$$
6. Send CT as the cipher text to the receiver.
7. For decryption, calculate the plain text PT from the cipher text CT as follows:

$$PT = CT^D \bmod N$$

Fig. 4.4 The RSA algorithm

4.4.2 Examples of RSA

Let us take an example of this process to understand the concepts. For ease of reading, we shall write the example values along with the algorithm steps. This is shown in Fig. 4.5.

1. Choose two large prime numbers P and Q.

Let $P = 47$, $Q = 17$.

2. Calculate $N = P \times Q$.

We have, $N = 7 \times 17 = 119$.

3. Select the public key (i.e. the encryption key) E such that it is not a factor of $(P - 1) \times (Q - 1)$.

- Let us find $(7 - 1) \times (17 - 1) = 6 \times 16 = 96$.
- The factors of 96 are 2, 2, 2, 2, 2, and 3 (because $96 = 2 \times 2 \times 2 \times 2 \times 2 \times 3$).
- Thus, we have to choose E such that none of the factors of E is 2 and 3. As a few examples, we cannot choose E as 4 (because it has 2 as a factor), 15 (because it has 3 as a factor), 6 (because it has 2 and 3 both as factors).
- Let us choose E as 5 (it could have been any other number that does not its factors as 2 and 3).

4. Select the private key (i.e. the decryption key) D such that the following equation is true:

$$(D \times E) \bmod (P - 1) \times (Q - 1) = 1$$

- Let us substitute the values of E, P and Q in the equation.
- We have: $(D \times 5) \bmod (7 - 1) \times (17 - 1) = 1$
- That is, $(D \times 5) \bmod (6) \times (16) = 1$
- That is, $(D \times 5) \bmod (96) = 1$
- After some calculations, let us take $D = 77$. Then the following is true: $(77 \times 5) \bmod (96) = 385 \bmod 96 = 1$, which is what we wanted.

5. For encryption, calculate the cipher text CT from the plain text PT as follows:

$$CT = PT^E \bmod N$$

Let us assume that we want to encrypt plain text 10. Then we have,

$$CT = 10^5 \bmod 119 = 100000 \bmod 119 = 40$$

6. Send CT as the cipher text to the receiver.

Send 40 as the cipher text to the receiver.

7. For decryption, calculate the plain text PT from the cipher text CT as follows:

$$PT = CT^D \bmod N$$

- We perform the following:
- $PT = CT^D \bmod N$
- That is, $PT = 40^{77} \bmod 119 = 10$, which was the original plain text of step 5.

Fig. 4.5 Example of RSA algorithm

Let us take the same example slightly differently.

1. Here, we shall take $P = 7$ and $Q = 17$.
2. Therefore, $N = P \times Q = 7 \times 17 = 119$.
3. As we can see, $(P - 1) \times (Q - 1) = 6 \times 16 = 96$. The factors of 96 are 2, 2, 2, 2, 2, and 3. Therefore, our public key E must not have a factor of 2 and 3. Let us choose the public key value of E as 5.
4. Select the private key D such that $(D \times E) \bmod (P - 1) \times (Q - 1) = 1$. Let us choose D as 77, because we can see that $(5 \times 77) \bmod 96 = 385 \bmod 96 = 1$, which satisfies our condition.

Now, based on these values, let us consider an encryption and decryption process as shown in Fig. 4.6. Here, A is the sender and B is the receiver. As we can see, here we use an encoding scheme of encoding alphabets as $A = 1$, $B = 2$, ..., $Z = 26$. Let us assume that we want to encrypt a single alphabet F using this scheme, and with B 's public key as 77 (known to A and B) and B 's private key (known only to B) as 5. The description follows the figure.

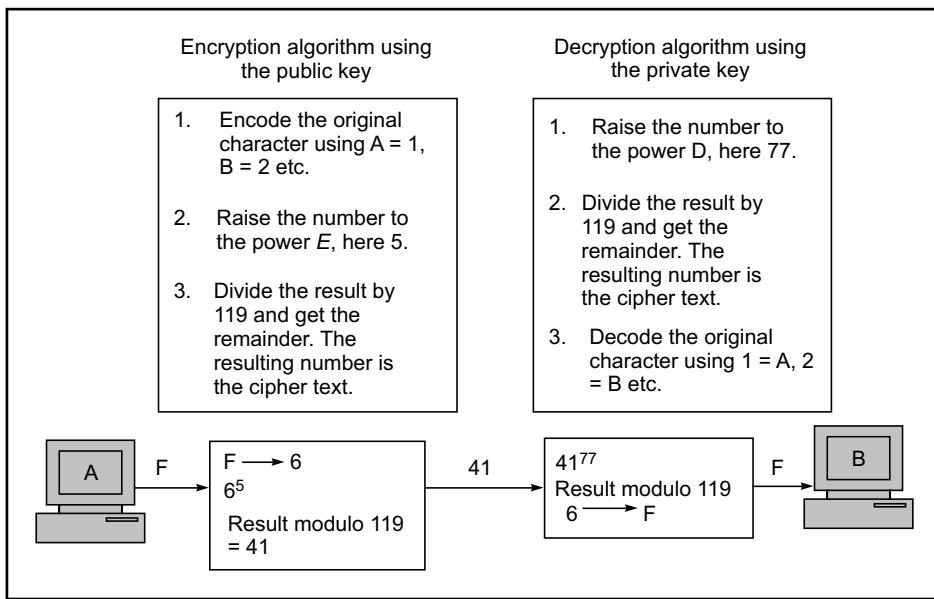


Fig. 4.6 Example of the RSA algorithm

This works as follows, assuming that the sender A wants to send a single character F to the receiver B . We have chosen such a simple case for the ease of understanding. Using the RSA algorithm, the character F would be encoded as follows:

1. Use the alphabet-numbering scheme (i.e. 1 for A , 2 for B , 3 for C , and so on). As per this rule, for F , we would have 6. Therefore, first, F would be encoded to 6.
2. Raise the number to the power of N , i.e. 6^5 .
3. Now calculate 6^5 modulo 119, which is 41. This is the encrypted information to be sent across the network.

At the receiver's end, the number 41 is decrypted to get back the original letter F as follows:

1. Raise the number to the power of N , i.e. 41^D , i.e. 41^{77} .

2. Now, calculate $41^{77} \bmod 119$. This gives 6.
3. Decode 6 back to F from our alphabet numbering scheme. This gives back the decrypted (original) plain text.

4.4.3 Understanding the Crux of RSA

Based on the calculations done in our examples, it should be clear that the RSA algorithm itself is quite simple: choosing the right keys is the real challenge. Suppose B wants to receive a confidential message from A , B must generate a private key (D), a public key (E) by using the mechanisms discussed earlier. B must then give the public key (E) and the number N to A . Using E and N , A encrypts the message and then sends the encrypted message to B . B uses her private key (D) to decrypt the message.

The question is: if B can calculate and generate D , anyone else should be able to do that as well! However, this is not straightforward, and herein lies the real crux of RSA.

It might appear that anyone (say an attacker) knowing about the public key E (5, in our second example) and the number N (119, in our second example) could find the private key D (77, in our second example) by trial and error. What does the attacker need to do? The attacker needs to first find out the values of P and Q using N (because we know that $N = P \times Q$). In our example, P and Q are quite small numbers (7 and 17, respectively). Therefore, it is quite easy to find out P and Q , given N . However, in the actual practice, P and Q are chosen as very large numbers. Therefore, factoring N to find P and Q is not at all easy. It is quite complex and time-consuming. Since the attacker cannot find out P and Q , he/she cannot proceed further to find out D , because D depends on P , Q and E . Consequently, even if the attacker knows N and E , she cannot find D , and therefore, cannot decrypt the cipher text.

Mathematical research suggests that it would take more than 70 years to find P and Q if N is a 100-digit number!

It is estimated that if we implement a symmetric algorithm such as DES and an asymmetric algorithm such as RSA in hardware, DES is faster by about 1000 times as compared to RSA. If we implement these algorithms in software, DES is faster by about 100 times.

4.4.4 Security of RSA

Although no successful attacks on RSA have been reported so far, chances of such attacks happening in the future have not been ruled out. We discuss some of the main possible attacks on RSA below.

1. Plain-text Attacks

The **plain-text attacks** are classified further into three sub-categories, as discussed next.

(a) Short-message Attack Here, the assumption is that the attacker knows some possible blocks of plain text. If this assumption is true, the attacker can try encrypting each plain-text block to see if it results into the known cipher text. To prevent this **short message attack**, it is recommended that we pad the plain text before encrypting it.

(b) Cycling Attack Here, the attacker presumes that the cipher text was obtained by doing permutation on the plain text in some manner. If this assumption is true then the attacker can do the reverse process, which is to continuously do permutations on the known cipher text to try and obtain the original

plain text. However, the trouble for the attacker could be that the attacker does not know what can be considered as the right plain text while using this method. Hence, the attacker can keep on doing permutation of the cipher text until he/she obtains the cipher text itself, in other words, completing one full cycle of permutations. If the attacker does obtain the original cipher text again by this method, the attacker knows that the text that was obtained in the step just prior to obtaining the original cipher text must be the original plain text. Hence, this attack is called **cycling attack**. However, this has not been found to be successful so far.

(c) Unconcealed Message Attack It is found in theory that in the case of some very rare plain-text messages, encryption gives cipher text which is the same as the original plain text! If this happens, the original plain-text message cannot be hidden. Hence, this attack is called **unconcealed message attack**. Hence, it may be prudent to ensure that after RSA encryption, the resulting cipher text is not the same as the original plain text before the cipher text is sent to the receiver, to prevent this attack.

2. Chosen-cipher text attack

In this complicated scheme called **chosen-cipher text attack**, the attacker is able to find out the plain text based on the original cipher text using what is called *extended Euclidean algorithm*.

3. Factorization Attack

The whole security of RSA is based on the assumption that it is infeasible for the attacker to factor the number N into its two factors P and Q . However, if the attacker is able to find out P or Q from the equation $N = P \times Q$ then the attacker can find out the private key, as we have discussed earlier. Assuming that N is at least 300 digits long in decimal terms, the attacker cannot find P and Q easily. Hence, the **factorization attack** fails.

4. Attacks on the Encryption Key

People well versed with the mathematics of RSA sometimes feel that it is quite slow because we use a large number for the public key or encryption key E . While this is true, it also makes RSA more secure. Hence, if we decide to try and make the working of RSA faster by using a small value for E , it can lead to potential attacks called **attacks on the encryption key** and hence it is recommended that we use E as $2^{16} + 1 = 65537$ or a value closer to this number.

5. Attacks on the Decryption Key

The **attacks on the decryption key** can be classified further into two categories, as discussed below.

(a) Revealed Decryption Exponent Attack If the attacker can somehow guess the decryption key D , not only are the cipher texts generated by encrypting the plain texts with the corresponding encryption key E are in danger; but even the future messages are also in danger. To prevent this **revealed decryption exponent attack**, it is recommended that the sender uses fresh values for P , Q , N , and E as well.

(b) Low Decryption Exponent Attack Similar to the case explained in the context of the encryption key, it is tempting to use a small value for decryption key D to make RSA work faster. This can help the attacker in guessing the decryption key D by launching the **low decryption exponent attack**.

■ 4.5 ElGamal CRYPTOGRAPHY ■

Taher ElGamal created ElGamal cryptography, more popularly known as **ElGamal cryptosystem**. We leave the complicated mathematics behind and explain the algorithm in a simpler form below. There are three aspects that need to be discussed: ElGamal key generation, ElGamal encryption, and ElGamal decryption.

4.5.1 ElGamal Key Generation

This involves the following steps:

1. Select a large prime number called P . This is the first part of the encryption key or public key.
2. Select the decryption key or private key D . There are some mathematical rules that need to be followed here, which we are omitting for keeping things simple.
3. Select the second part of the encryption key or public key $E1$.
4. The third part of the encryption key or public key $E2$ is computed as $E2 = E1^D \bmod P$.
5. The public key is $(E1, E2, P)$ and the private key is D .

For example, $P = 11$, $E1 = 2$, $D = 3$. Then $E2 = E1^D \bmod P = 2^3 \bmod 11 = 8$.

Hence, the public key is $(2, 8, 11)$ and the private key is 3 .

4.5.2 ElGamal Key Encryption

This involves the following steps:

1. Select a random integer R that fulfills some mathematical properties, which are ignored here.
2. Compute the first part of the cipher text $C1 = E1^R \bmod P$.
3. Compute the second part of the cipher text $C2 = (PT \times E2^R) \bmod P$, where PT is the plain text.
4. The final cipher text is $(C1, C2)$.

In our example, let $R = 4$ and plain text $PT = 7$. Then we have:

$$C1 = E1^R \bmod P = 2^4 \bmod 11 = 16 \bmod 11 = 5$$

$$C2 = (PT \times E2^R) \bmod P = (7 \times 2^8) \bmod 11 = (7 \times 4096) \bmod 11 = 6$$

Hence, our cipher text is $(5, 6)$.

4.5.3 ElGamal Key Decryption

This involves the following step:

Compute the plain text PT using the formula $PT = [C2 \times (C1^D)^{-1}] \bmod P$

In our example:

$$PT = [C2 \times (C1^D)^{-1}] \bmod P$$

$$PT = [6 \times (5^3)^{-1}] \bmod 11 = [6 \times 3] \bmod 11 = 7, \text{ which was our original plain text.}$$

■ 4.6 SYMMETRIC- AND ASYMMETRIC-KEY CRYPTOGRAPHY ■

4.6.1 Comparison Between Symmetric- and Asymmetric-Key Cryptography

Asymmetric-key cryptography (or the use of the receiver's public key for encryption) solves the problem of key agreement and key exchange, as we have seen. However, this does not solve all the problems in a practical security infrastructure. More specifically, symmetric-key cryptography and asymmetric-key cryptography differ in certain other respects, with both depicting certain advantages as compared to the other. Let us summarize them to appreciate how they are practically used in real life, as shown in Fig. 4.7. We have not discussed the last point (*usage*) in the table. However, it is mentioned for the sake of completeness. We shall visit those aspects shortly.

Characteristic	Symmetric-Key Cryptography	Asymmetric-Key Cryptography
Key used for encryption/decryption	Same key is used for encryption and decryption	One key used for encryption and another, different key is used for decryption
Speed of encryption/decryption	Very fast	Slower
Size of resulting encrypted text	Usually same as or less than the original clear text size	More than the original clear text size
Key agreement/exchange	A big problem	No problem at all
Number of keys required as compared to the number of participants in the message exchange	Equals about the square of the number of participants, so scalability is an issue	Same as the number of participants, so scales up quite well
Usage	Mainly for encryption and decryption (confidentiality), cannot be used for digital signatures (integrity and non-repudiation checks)	Can be used for encryption and decryption (confidentiality) as well as for digital signatures (integrity and non-repudiation checks)

Fig. 4.7 Symmetric versus asymmetric key cryptography

The above table shows that both symmetric-key cryptography and asymmetric-key cryptography have useful features. Also, both have some areas where better alternatives are generally desired. Asymmetric-key cryptography solves the major problem of key agreement/key exchange as well as scalability. However, it is far slower and produces huge chunks of cipher text as compared to symmetric-key cryptography (essentially because it uses large keys and complex algorithms as compared to symmetric-key cryptography).

4.6.2 The Best of Both Worlds

How economic it would be, if we can combine the two cryptography mechanisms, so as to achieve the better of the two, and yet do not compromise on any of the features! More specifically, we need to ensure that the following objectives are met:

1. The solution should be completely secure.
2. The encryption and decryption processes must not take a long time.

3. The generated cipher text should be compact in size.
4. The solution should scale to a large number of users easily, without introducing any additional complications.
5. The key-distribution problem must be solved by the solution.

Indeed, in practice, symmetric-key cryptography and asymmetric-key cryptography are combined to have a very efficient security solution. The way it works is as follows, assuming that A is the sender of a message and B is its receiver.

1. A 's computer encrypts the original plain-text message (PT) with the help of a standard symmetric-key cryptography algorithm, such as DES, IDEA or RC5, etc. This produces a cipher-text message (CT) as shown in Fig. 4.8. The key used in this operation ($K1$) is called one-time symmetric key, as it is used once and then discarded.

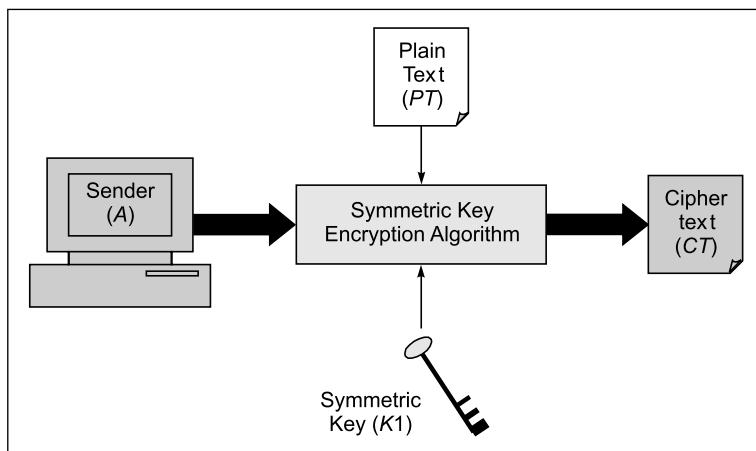


Fig. 4.8 Encrypting the plain text with a symmetric-key algorithm

2. We would now think, we are back to square one! We have encrypted the plain text (PT) with a symmetric-key operation. We must now transport this one-time symmetric key ($K1$) to the server, so that the server can decrypt the cipher text (CT) to get back the original plain-text message (PT). Does this not again lead us to the key-exchange problem? Well, a novel concept is used now. A now takes the one-time symmetric key of step 1 (i.e. $K1$), and encrypts $K1$ with B 's public key ($K2$). This process is called **key wrapping** of the symmetric key, and is shown in Fig. 4.9. We have shown that the symmetric key $K1$ goes inside a logical box, which is sealed by B 's public key (i.e. $K2$).
3. Now, A puts the cipher text $CT1$ and the encrypted symmetric key together inside a **digital envelope**. This is shown in Fig. 4.10.
4. The sender (A) now sends the digital envelope [which contains the cipher text (CT) and the one-time symmetric key ($K1$) encrypted with B 's public key, ($K2$)] to B using the underlying transport mechanism (network). This is shown in Fig. 4.11. We do not show the contents of the envelope, and assume that the envelope contains the two entities, as discussed.
5. B receives and opens the digital envelope. After B opens the envelope, it receives two things: cipher text (CT) and the one-time session key ($K1$) encrypted using B 's private key ($K2$). This is shown in Fig. 4.12.

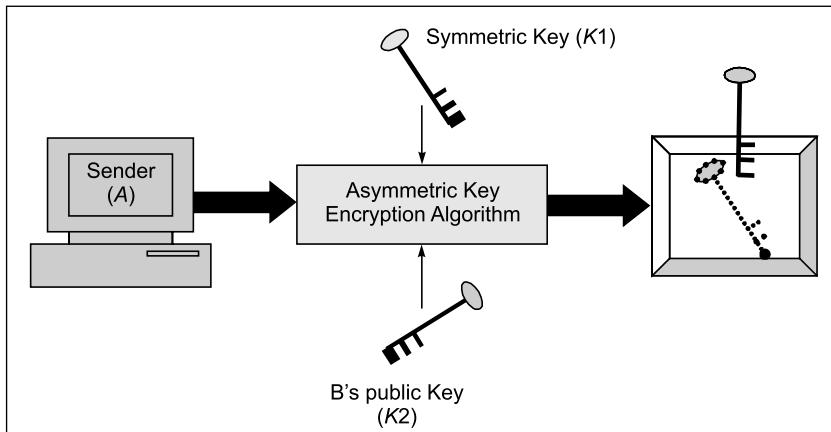


Fig. 4.9 Symmetric-key wrapping using the receiver's public key

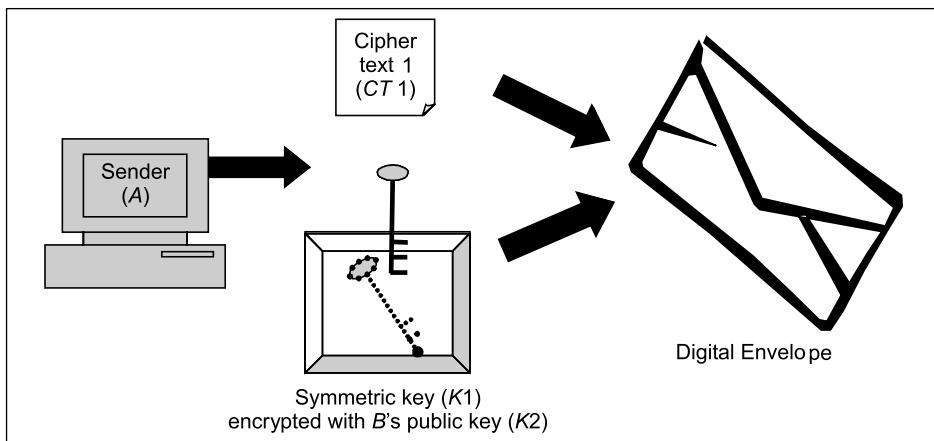


Fig. 4.10 Digital envelope

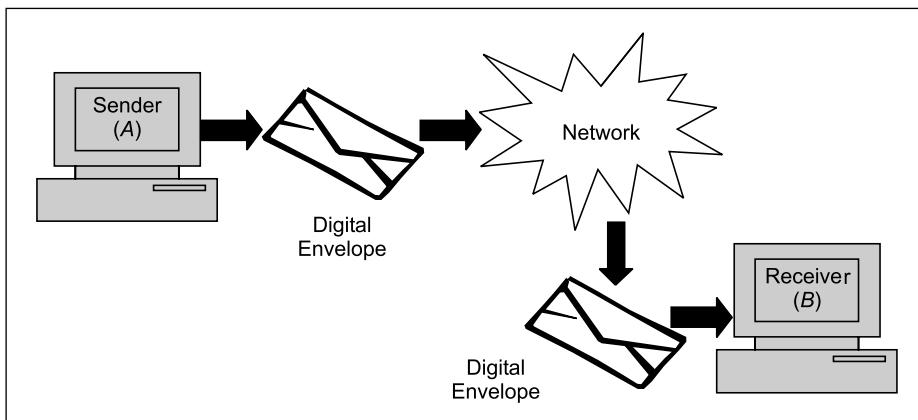


Fig. 4.11 Digital envelope reaches B over the network

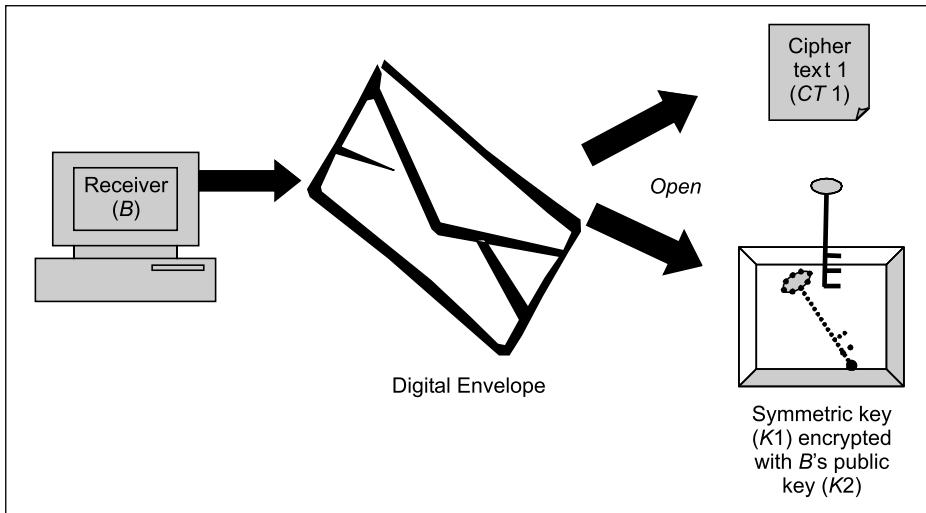


Fig. 4.12 *B* opens the digital envelope using her private key

6. *B* now uses the same asymmetric-key algorithm as was used by *A* and her private key (K_3) to decrypt (i.e. open up) the logical box that contains the symmetric key (K_1), which was encrypted with *B*'s public key (K_2). This is shown in Fig. 4.13. Thus, the output of the process is the one-time symmetric key K_1 .

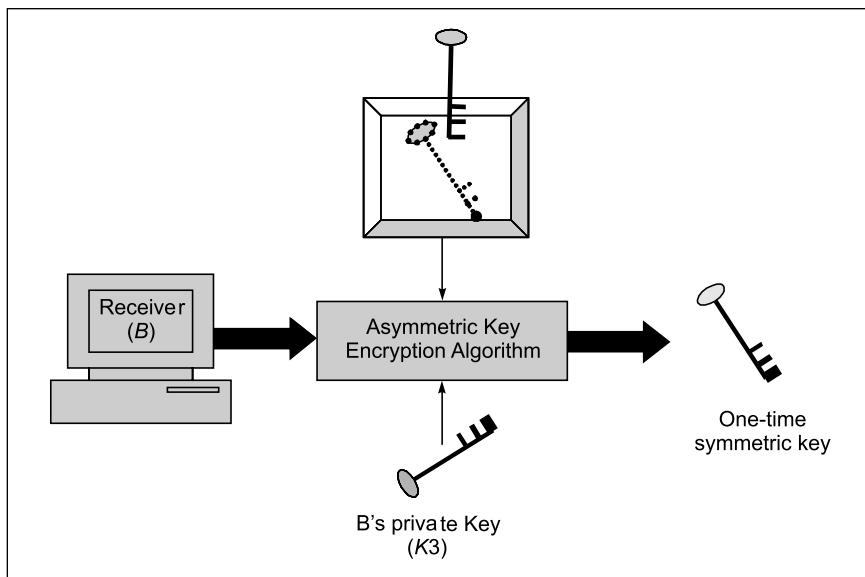


Fig. 4.13 Retrieval of one-time symmetric key

7. Finally, *B* applies the same symmetric-key algorithm as was used by *A*, and the symmetric key K_1 to decrypt the cipher text (CT). This process yields the original plain text (PT), as shown in Fig. 4.14.

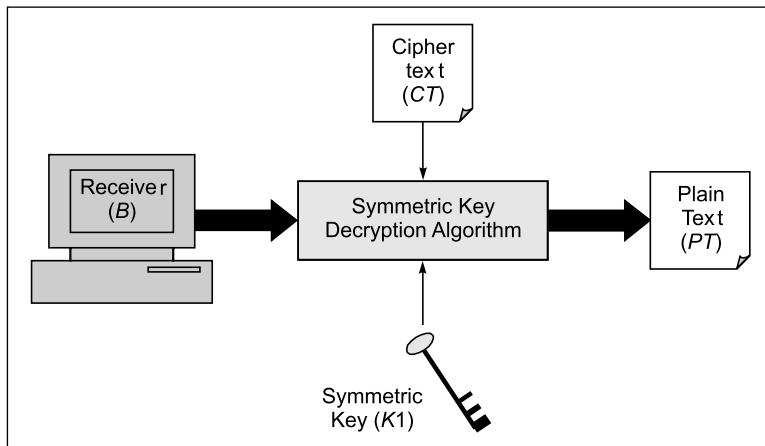


Fig. 4.14 Using symmetric key to retrieve the original plain text

One would think that this process looks complicated. How come this is actually efficient? The reason that this process based on digital envelopes is efficient is because of the following reasons:

- (a) Firstly, we encrypt the plain text (*PT*) with the one-time session key (*K1*) using a symmetric-key cryptographic algorithm. As we know, symmetric key encryption is fast, and the generated cipher text (*CT*) is of the same size of the original plain text (*PT*). Instead, if we had used an asymmetric-key encryption here, the operation would have been quite slow, especially if the plain text was of a large size, which it can be. Also, the output cipher text (*CT*) produced would have been of a size greater than the size of the original plain text (*PT*).
- (b) Secondly, we encrypted the one-time session key (*K1*) with *B*'s public key (*K2*). Since the size of *K1* is going to be small (usually 56 or 64 bits), this asymmetric-key encryption process would not take too long, and the resulting encrypted key would also not consume a lot of space.
- (c) Thirdly, we have been able to solve the problem of key exchange with this scheme, without losing the advantages of either symmetric-key cryptography or asymmetric-key cryptography!

A few questions are still unanswered. How does *B* know which symmetric- or asymmetric-key encryption algorithms *A* has used? *B* needs to know this because it must perform the decryption processes accordingly, using the same algorithms. Well, actually the digital envelope sent by *A* to *B* carries this sort of information as well. Therefore, *B* knows which algorithms to use to first decrypt the one-time session key (*K1*) using her private key (*K3*), and then which algorithm to use to decrypt the cipher text (*CT*) using the one-time session key (*K1*).

This is how, in the real world, symmetric- and asymmetric-key cryptographic techniques are used in combination. Digital envelopes have proven to be a very sound technology for transferring messages from the sender to the receiver, achieving confidentiality.

■ 4.7 DIGITAL SIGNATURES ■

4.7.1 Introduction

All along, we have been talking of the following general scheme in the context of asymmetric-key cryptography:

If A is the sender of a message and B is the receiver, A encrypts the message with B 's public key and sends the encrypted message to B .

We have deliberately hidden the internals of this scheme. As we know, actually this is based on digital envelopes as discussed earlier, wherein not the entire message but only the one-time session key used to encrypt the message is encrypted with the receiver's public key. But for simplicity, we shall ignore this technical detail, and instead, assume that the whole message is encrypted with the receiver's public key.

Let us now consider another scheme, as follows:

If A is the sender of a message and B is the receiver, A encrypts the message with A 's private key and sends the encrypted message to B .

This is shown in Fig. 4.15.

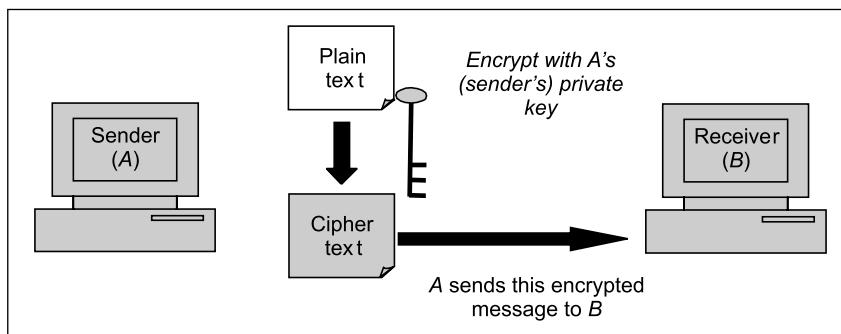


Fig. 4.15 Encrypting a message with the sender's private key

Our first reaction to this would be: what purpose would this serve? After all, A 's public key would be, well, *public*, i.e. accessible to anybody. This means that anybody who is interested in knowing the contents of the message sent by A to B can simply use A 's public key to decrypt the message, thus causing the failure of this encryption scheme!

Well, this is quite true. But here, when A encrypts the message with her private key, her intention is not to hide the contents of the message (i.e. not to achieve *confidentiality*), but it is something else. What can that intention be? If the receiver (B) receives such a message encrypted with A 's private key, B can use A 's public key to decrypt it, and therefore, access the plain text. Does this ring a bell? If the decryption is successful, it assures B that this message was indeed sent by A . This is because if B can decrypt a message with A 's public key, it means that the message must have been initially encrypted with A 's private key (remember that a message encrypted with a public key can be decrypted only with the corresponding private key, and vice versa). This is also because only A knows her private key. Therefore, someone posing as A (say C) could not have sent a message encrypted with A 's private key to B . A must have sent it. Therefore, although this scheme does not achieve confidentiality, it achieves *authentication* (identifying and proving A as the sender). Moreover, in the case of a dispute tomorrow, B can take the encrypted message, and decrypt it with A 's public key to prove that the message indeed came from A . This achieves the purpose of *non-repudiation* (i.e. A cannot refuse that she had sent this message, as the message was encrypted with her private key, which is supposed to be known only to her).

Even if someone (say C) manages to intercept and access the encrypted message while it is in transit, then uses A 's public key to decrypt the message, changes the message, that would still not achieve any

purpose. Because C does not have A 's private key, C cannot encrypt the changed message with A 's private key again. Therefore, even if C now forwards this changed message to B , B will not be fooled into believing that it came from A , as it was not encrypted with A 's private key.

Such a scheme, wherein the sender encrypts the message with his/her private key, forms the basis of digital signatures, as shown in Fig. 4.16.

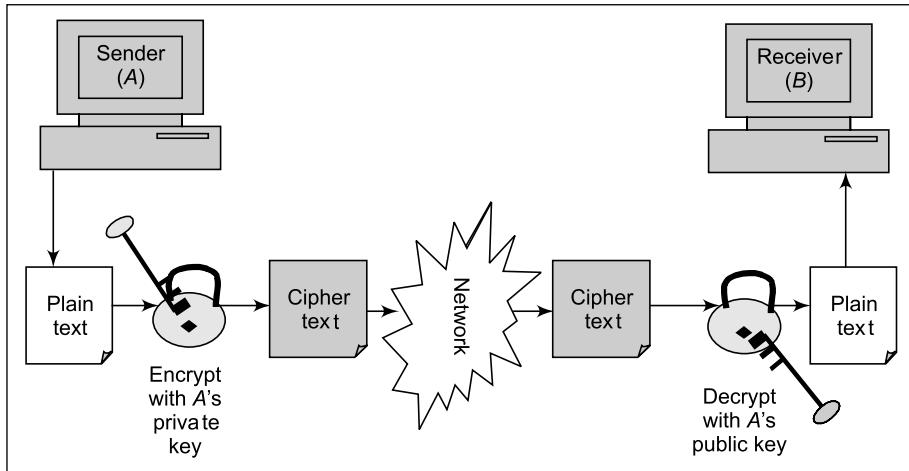


Fig. 4.16 Basis for digital signatures

Digital signatures have assumed great significance in the modern world of Web commerce. Most countries have already made provisions for recognizing a digital signature as a valid authorization mechanism, just like paper-based signatures. Digital signatures have legal status now. For example, suppose you send a message to your bank over the Internet, to transfer some amount from your account to your friend's account, and digitally sign the message, this transaction has the same status as the one wherein you fill in and sign the bank's paper-based money-transfer slip.

We have seen the theory behind digital signatures. However, there are some undesirable elements in this scheme, as we shall study next.

4.7.2 Message Digests

1. Introduction

If we examine the conceptual process of digital signatures, we will realize that it does not deal with the problems associated with asymmetric-key encryption, namely slow operation and large cipher-text size. This is because we are encrypting the whole of the original plain-text message with the sender's private key. As the size of the original plain text can be quite large, this encryption process can be really very slow.

We can tackle this problem using the digital envelope approach, as before. That is, A encrypts the original plain-text message (PT) with a one-time symmetric key ($K1$) to form the cipher text (CT). It then encrypts the one-time symmetric key ($K1$) with her private key ($K2$). She creates a digital envelope containing CT and $K1$ encrypted with $K2$, and sends the digital envelope to B . B opens the digital envelope, uses A 's public key ($K3$) to decrypt the encrypted one-time symmetric key, and

obtains the symmetric key $K1$. It then uses $K1$ to decrypt the cipher text (CT) and obtains the original plain text (PT). Since B uses A 's public key to decrypt the encrypted one-time symmetric key ($K1$), B can be assured that only A 's private key could have encrypted $K1$. Thus, B can be assured that the digital envelope came from A .

Such a scheme could work perfectly. However, in real practice, a more efficient scheme is used. It involves the usage of a **message digest** (also called **hash**).

A message digest is a *fingerprint* or the summary of a message. It is similar to the concepts of *Longitudinal Redundancy Check (LRC)* or *Cyclic Redundancy Check (CRC)*. That is, it is used to verify the *integrity* of the data (i.e. to ensure that a message has not been tampered with after it leaves the sender but before it reaches the receiver). Let us understand this with the help of an LRC example (CRC would work similarly, but will have a different mathematical base).

An example of LRC calculation at the sender's end is shown in Fig. 4.17. As shown, a block of bits is organized in the form of a list (as rows) in the *Longitudinal Redundancy Check (LRC)*. Here, for instance, if we want to send 32 bits, we arrange them into a list of four (horizontal) rows. Then we count how many 1 bits occur in each of the 8 (vertical) columns. [If the number of 1s in the column is odd then we say that the column has *odd parity* (indicated by a 1 bit in the shaded LRC row); otherwise if the number of 1s in the column is even, we call it *even parity* (indicated by a 0 bit in the shaded LRC row).] For instance, in the first column, we have two 1s, indicating an even parity, and therefore, we have a 0 in the shaded LRC row for the first column. Similarly, for the last column, we have three 1s, indicating an odd parity, and therefore, we have a 1 in the shaded LRC row for the last column. Thus, the parity bit for each column is calculated and a new row of eight parity bits is created. These become the parity bits for the whole block. Thus, the LRC is actually a *fingerprint* of the original message.

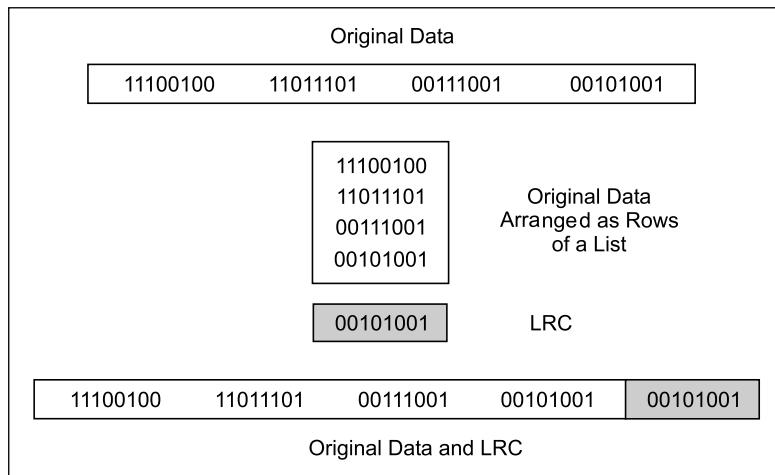


Fig. 4.17 Longitudinal Redundancy Check (LRC)

The data along with the LRC is then sent to the receiver. The receiver separates the data block from the LRC block (shown shaded). It performs its own LRC on the data block alone. It then compares its LRC values with the ones received from the sender. If the two LRC values match then the receiver has a reasonable confidence that the message sent by the sender has not been changed, while in transit.

2. Idea of a Message Digest

The concept of message digests is based on similar principles. However, it is slightly wider in scope. For instance, suppose that we have a number 4000 and we divide it by 4 to get 1000. Thus, 4 becomes a fingerprint of the number 4000. Dividing 4000 by 4 will always yield 1000. If we change either 4000 or 4, the result will not be 1000.

Another important point is, if we are simply given the number 4, but are not given any further information, we would not be able to trace back the equation $4 \times 1000 = 4000$. Thus, we have one more important concept here. The fingerprint of a message (in this case, the number 4) does not tell anything about the original message (in this case, the number 4000). This is because there are infinite other possible equations, which can produce the result 4.

Another simple example of a message digest is shown in Fig. 4.18. Let us assume that we want to calculate the message digest of a number 7391753. Then, we multiply each digit in the number with the next digit (excluding it if it is 0), and disregarding the first digit of the multiplication operation, if the result is a two-digit number.

Thus, we perform a hashing operation (or a message-digest algorithm) over a block of data to produce its hash or message digest, which is smaller in size than the original message. This concept is shown in Fig. 4.19.

So far, we are considering very simple cases of message digests. Actually, the message digests are not so small and straightforward to compute. Message digests usually consist of 128 or more bits. This means that the chance of any two message digests being the same is anything between 0 to at least 2^{128} . The message-digest length is chosen to be so long with a purpose. This ensures that the scope for two message digests is the same.

3. Requirements of a Message Digest

We can summarize the requirements of the message-digest concept, as follows:

- Given a message, it should be very easy to find its corresponding message digest. This is shown in Fig. 4.20. Also, for a given message, the message digest must always be the same.
- Given a message digest, it should be very difficult to find the original message for which the digest was created. This is shown in Fig. 4.21.

• Original number is 7391743	
Operation	Result
Multiply 7 by 3	21
Discard first digit	1
Multiply 1 by 9	9
Multiply 9 by 1	9
Multiply 9 by 7	63
Discard first digit	3
Multiply 3 by 4	12
Discard first digit	2
Multiply 2 by 3	6
• Message digest is 6	

Fig. 4.18 Simplistic example of a message digest

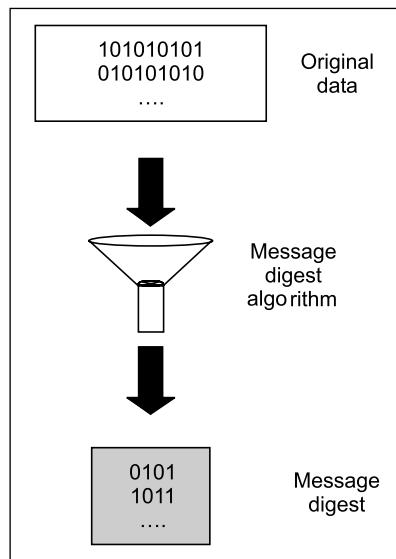


Fig. 4.19 Message-digest concept

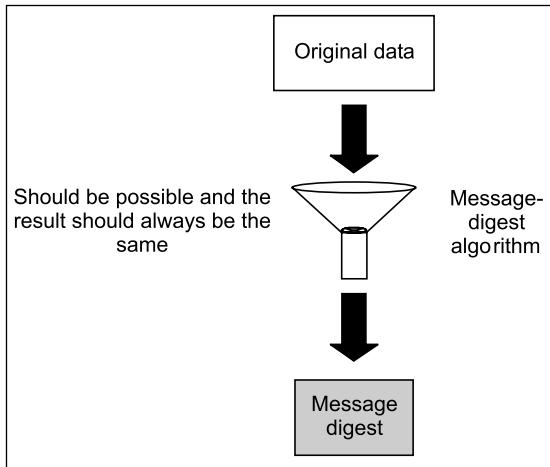


Fig. 4.20 Message digest for the original data should always be the same

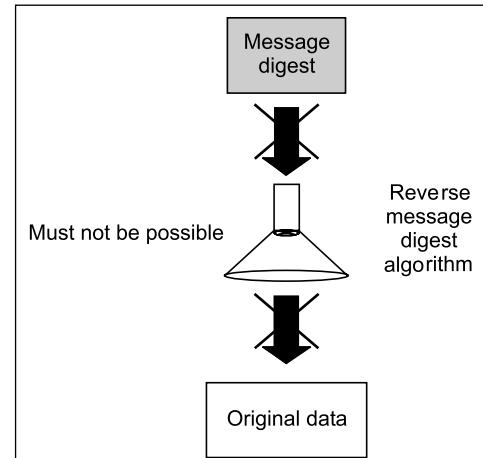


Fig. 4.21 Message digest should not work in the opposite direction

- (c) Given any two messages, if we calculate their message digests, the two message digests must be different. This is shown in Fig. 4.22.

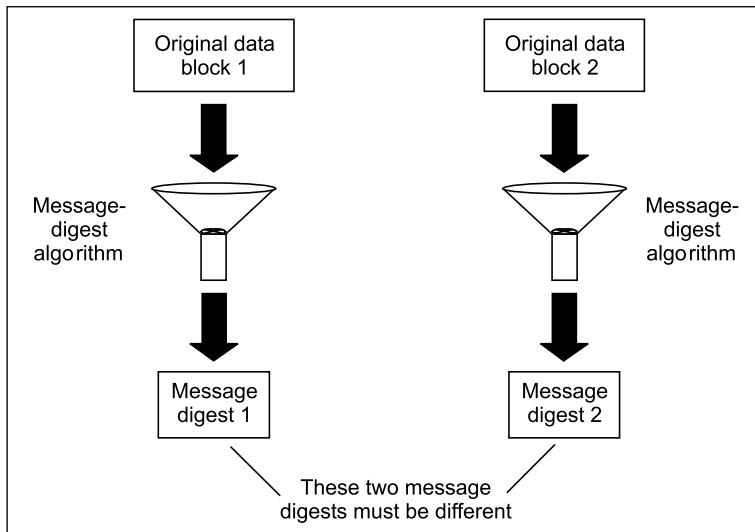


Fig. 4.22 Message digests of two different messages must be different

If any two messages produce the same message digest, thus violating our principle, it is called a **collision**. That is, if two message digests *collide*, they meet at the digest! As we shall study soon, the message-digest algorithms usually produce a message digest having a length of 128 bits or 160 bits. This means that the chances of any two message digests being the same are one in 2^{128} or 2^{160} , respectively. Clearly, this seems possible only in theory, but extremely rare in practice.

A specific type of security attack called **birthday attack** is used to detect collisions in message-digest algorithms. It is based on the principle of the *Birthday Paradox*, which states that if there are 23 people

in a room, chances are that more than 50% of the people will share the same birthday. At first, this may seem to be illogical. However, we can understand this in another manner. We need to keep in mind we are just talking about *any two* people (out of the 23) sharing the same birthday. Moreover, we are not talking about this sharing with a specific person. For instance, suppose that we have Alice, Bob, and Carol as three of the 23 people in the room. Therefore, Alice has 22 possibilities to share a birthday with anyone else (since there are 22 pairs of people). If there is no matching birthday for Alice, she leaves. Bob now has 21 chances to share a birthday with anyone else in the room. If he fails to have a match too, the next person is Carol. She has 20 chances, and so on. 22 pairs + 21 pairs + 20 pairs ... + 1 pair means that there is a total of 253 pairs. Every pair has a 1/365th chance of finding a matching birthday. Clearly, the chances of a match cross 50% at 253 pairs.

The birthday attack is most often used to attempt to discover collisions in hash functions, such as MD5 or SHA1.

This can be explained as follows:

If a message digest uses 64-bit keys then after trying 2^{32} transactions, an attacker can expect that for two different messages, he/she may get the same message digests. In general, for a given message, if we can compute up to N different message digests then we can expect the first collision after the number of message digests computed exceeds the square root of N . In other words, a collision is expected when the probability of collision exceeds 50%. This can lead to birthday attacks.

It might surprise you to know that even a small difference between two original messages can cause the message digests to differ vastly. The message digests of two extremely similar messages are so different that they provide no clue at all that the original messages were very similar to each other. This is shown in Fig. 4.23. Here, we have two messages (*Please pay the newspaper bill today* and *Please pay the newspaper bill tomorrow*), and their corresponding message digests. Note how similar the messages are, and yet how different their message digests are.

Message	Please pay the newspaper bill today
Message digest	306706092A864886F70D010705A05A3058020100300906052B0E03 021A0500303206092A864886F70D010701A0250423506C65617365 2070617920746865206E65777370617065722062696C6C20746F646
Message	Please pay the newspaper r bill tomorrow
Message digest	306A06092A864886F70D010705A05D305B020100300906052B0E 03021A0500303506092A864886F70D010701A0280426506C65617 3652070617920746865206E65777370617065722062696C6C20746

Fig. 4.23 Message-digest example

Looked another way, we are saying that given one message (M_1) and its message digest (MD), it is simply not feasible to find another message (M_2), which will also produce MD exactly the same, bit by bit. The message-digest scheme should try and prevent this to the maximum extent possible. This is shown in Fig. 4.24.

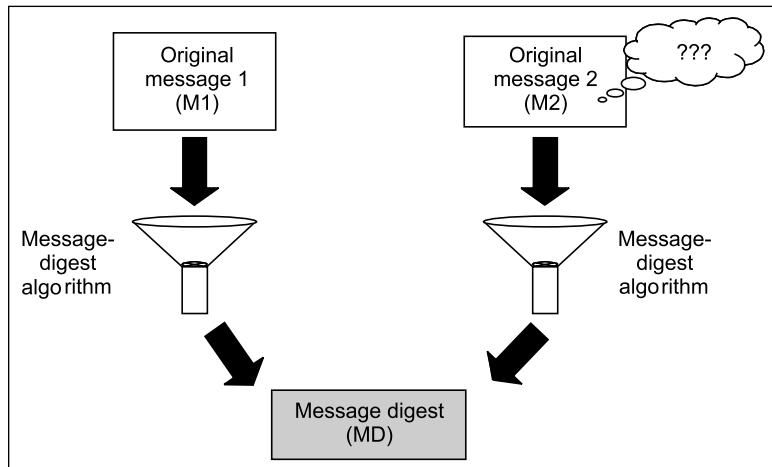


Fig. 4.24 Message digests should not reveal anything about the original message

4.7.3 MD5

1. Introduction

MD5 is a message-digest algorithm developed by Ron Rivest. MD5 actually has its roots in a series of message-digest algorithms, which were the predecessors of MD5, all developed by Rivest. The original message-digest algorithm was called **MD**. Rivest soon came up with its next version, **MD2**. He first developed it, but it was found to be quite weak. Therefore, Rivest began working on **MD3**, which was a failure (and therefore, was never released). Then, Rivest developed **MD4**. However, soon, MD4 was also found to be wanting. Consequently, Rivest released MD5.

MD5 is quite fast, and produces 128-bit message digests. Over the years, researchers have developed potential weaknesses in MD5. However, so far, MD5 has been able to successfully defend itself against collisions. This may not be guaranteed for too long, though.

After some initial processing, the input text is processed in 512-bit blocks (which are further divided into 16 32-bit sub-blocks). The output of the algorithm is a set of four 32-bit blocks, which make up the 128-bit message digest.

2. The Working of MD5

Step 1: Padding The first step in MD5 is to add padding bits to the original message. The aim of this step is to make the length of the original message equal to a value, which is 64 bits less than an exact multiple of 512. For example, if the length of the original message is 1000 bits, we add a padding of 472 bits to make the length of the message 1472 bits. This is because, if we add 64 to 1472, we get 1536, which is a multiple of 512 (because $1536 = 512 \times 3$).

Thus, after padding, the original message will have a length of 448 bits (64 bits less than 512), 960 bits (64 bits less than 1024), 1472 bits (64 bits less than 1536), etc.

The padding consists of a single 1 bit, followed by as many 0 bits, as required. Note that padding is always added, even if the message length is already 64 bits less than a multiple of 512. Thus, if the

message were already of a length of say 448 bits, we will add a padding of 512 bits to make its length 960 bits. Thus, the padding length is any value between 1 and 512.

The padding process is shown in Fig. 4.25.

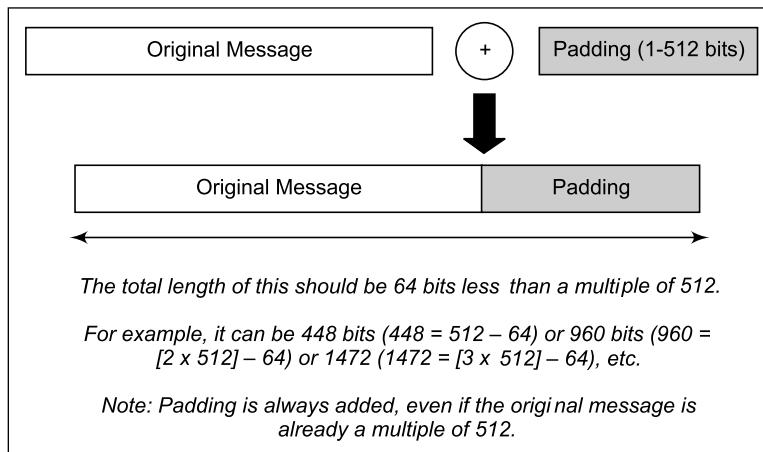


Fig. 4.25 Padding process

Step 2: Append Length After padding bits are added, the next step is to calculate the original length of the message, and add it to the end of the message, after padding. How is this done?

The length of the message is calculated, excluding the padding bits (i.e. it is the length before the padding bits were added). For instance, if the original message consisted of 1000 bits, and we added a padding of 472 bits to make the length of the message 64 bits less than 1536 (a multiple of 512), the length is considered 1000, and not 1472, for the purpose of this step.

This length of the original message is now expressed as a 64-bit value, and these 64 bits are appended to the end of the original message + padding. This is shown in Fig. 4.26. Note that if the length of the message exceeds 2^{64} bits (i.e. 64 bits are not enough to represent the length, which is possible in the case of a really long message), we use only the low-order 64 bits of the length. That is, in effect, we calculate the length mod 2^{64} in that case.

We will realize that the length of the message is now an exact multiple of 512. This now becomes the message whose digest will be calculated.

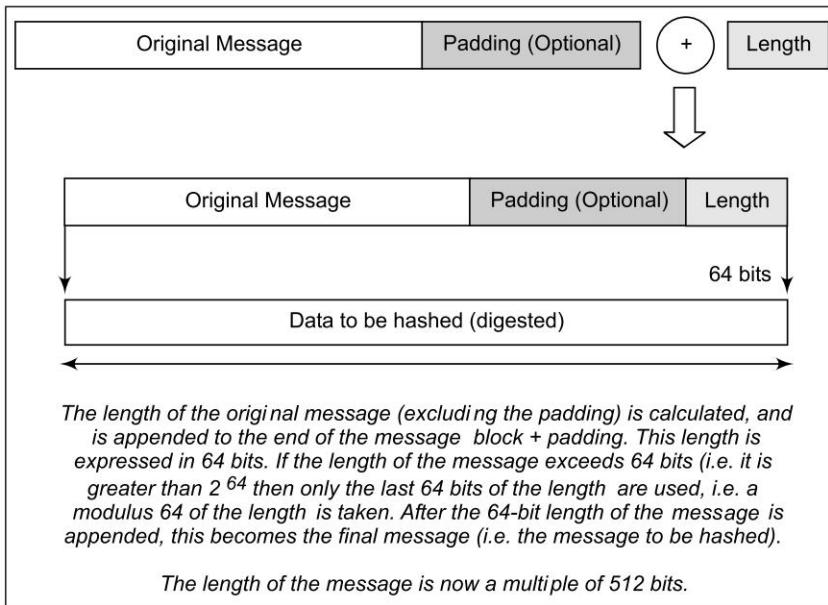
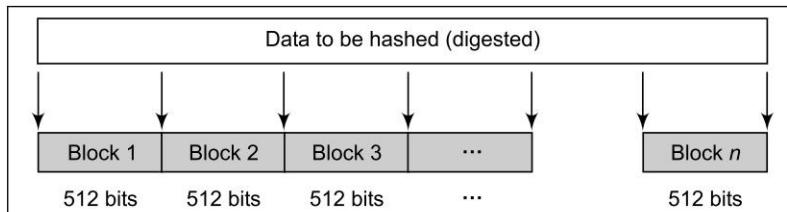
Step 3: Divide the Input into 512-bit Blocks Now, we divide the input message into blocks, each of length 512 bits. This is shown in Fig. 4.27.

Step 4: Initialize Chaining Variables In this step, four variables (called **chaining variables**) are initialized. They are called A , B , C and D . Each of these is a 32-bit number. The initial hexadecimal values of these chaining variables are shown in Fig. 4.28.

Step 5: Process Blocks After all the initializations, the real algorithm begins. It is quite complicated, and we shall discuss it step by step to simplify it to the maximum extent possible.

There is a loop that runs for as many 512-bit blocks as are in the message.

Step 5.1 Copy the four chaining variables into four corresponding variables, a , b , c and d (note the smaller case). Thus, we now have $a = A$, $b = B$, $c = C$ and $d = D$. This is shown in Fig. 4.29.

**Fig. 4.26** Append length**Fig. 4.27** Data is divided into 512-bit blocks

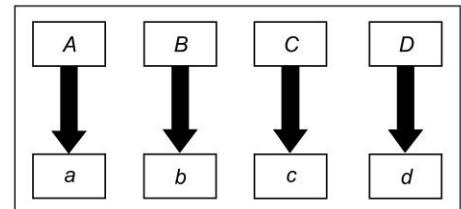
A	Hex	01	23	45	67
B	Hex	89	AB	CD	EF
C	Hex	FE	DC	BA	98
D	Hex	76	54	32	10

Fig. 4.28 Chaining variables

Actually, the algorithm considers the combination of a , b , c and d as a 128-bit single register (which we shall call $abcd$). This register ($abcd$) is useful in the actual algorithm operation for holding intermediate as well as final results. This is shown in Fig. 4.30.

Step 5.2 Divide the current 512-bit block into 16 sub-blocks. Thus, each sub-block contains 32 bits, as shown in Fig. 4.31.

Step 5.3 Now, we have four *rounds*. In each round, we process all the 16 sub-blocks belonging to a block. The inputs to each round are (a) all the 16 sub-blocks, (b) the variables a , b , c , d , and (c) some constants, designated as t . This is shown in Fig. 4.32.

**Fig. 4.29** Copying chaining variables into temporary variables

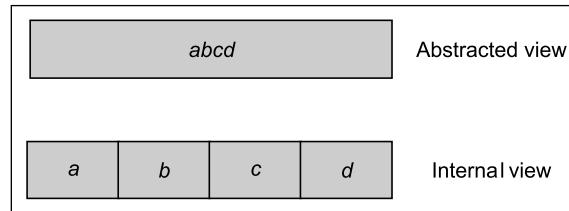


Fig. 4.30 Abstracted view of the chaining variables

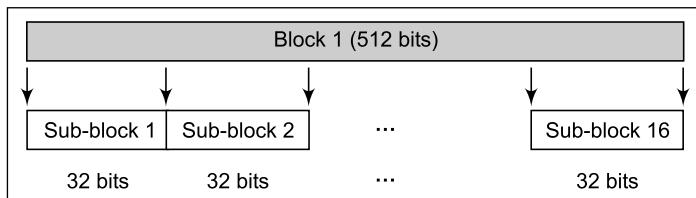


Fig. 4.31 Sub-blocks within a block

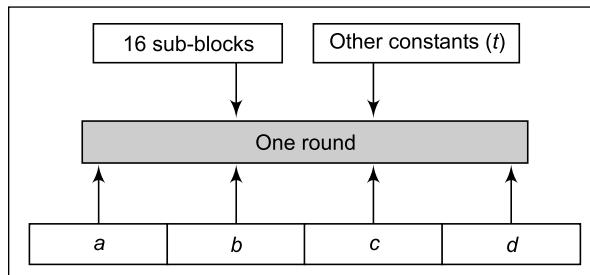


Fig. 4.32 Conceptual process within a *round*

What is done in these four rounds? All the four rounds vary in one major way: step 1 of the four rounds has different processing. The other steps in all the four rounds are the same.

- In each round, we have 16 input sub-blocks, named $M[0]$, $M[1]$, ..., $M[15]$, or in general, $M[i]$, where i varies from 1 to 15. As we know, each sub-block consists of 32 bits.
- Also, t is an array of constants. It contains 64 elements, with each element consisting of 32 bits. We denote the elements of this array t as $t[1]$, $t[2]$, ..., $t[64]$, or in general as $t[k]$, where k varies from 1 to 64. Since there are four rounds, we use 16 out of the 64 values of t in each round.

Let us summarize these iterations of all the four rounds. In each case, the output of the intermediate as well as the final iteration is copied into the register *abcd*. Note that we have 16 such iterations in each round.

1. A process P is first performed on *b*, *c* and *d*. This process P is different in all the four rounds.
2. The variable *a* is added to the output of the process P (i.e. to the register *abcd*).
3. The message sub-block $M[i]$ is added to the output of step 2 (i.e. to the register *abcd*).
4. The constant $t[k]$ is added to the output of step 3 (i.e. to the register *abcd*).
5. The output of step 4 (i.e. the contents of register *abcd*) is circular-left shifted by s bits. (The value of s keeps changing, as we shall study).

6. The variable b is added to the output of step 5 (i.e. to the register $abcd$).
7. The output of step 6 becomes the new $abcd$ for the next step.

This is shown in Fig. 4.33.

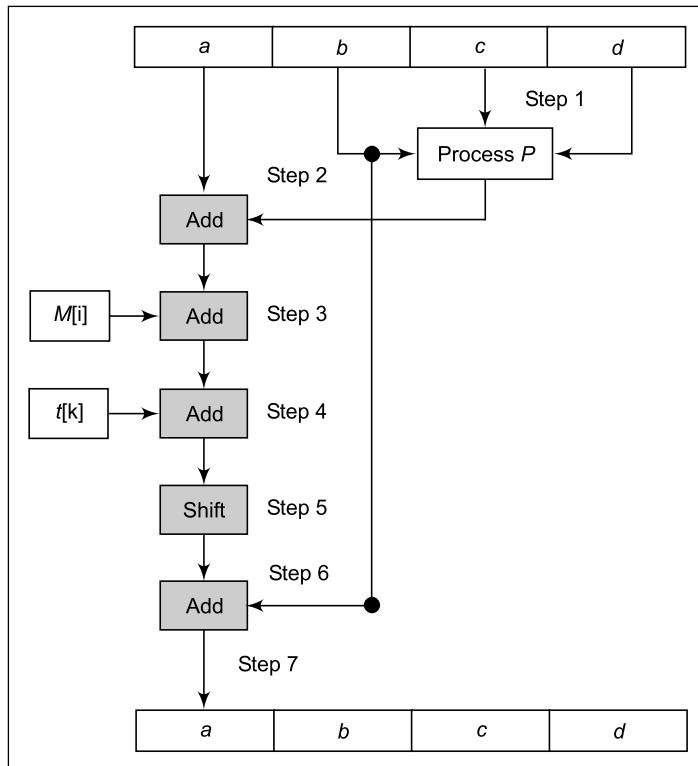


Fig. 4.33 One MD5 operation

We can mathematically express a single MD5 operation as follows:

$$a = b + ((a + \text{Process } P(b, c, d) + M[i] + t[k]) \ll\ll s)$$

where,

a, b, c, d = Chaining variables, as described earlier

$\text{Process } P$ = A non-linear operation, as described subsequently

$M[i] = M[q \times 16 + i]$, which is the i th 32-bit word in the q th 512-bit block of the message

$t[k]$ = A constant, as discussed subsequently

$\ll\ll_s$ = Circular-left shift by s bits

Understanding the Process P As we can see, the most crucial aspect here is to understand the process P , as it is different in the four rounds. In simple terms, the process P is nothing but some basic Boolean operations on b, c and d . This is shown in Fig. 4.34.

Note that in the four rounds, only the process P differs. All the other steps remain the same. Thus, we can substitute the actual details of process P in each of the round, and keep everything else constant.

Round	Process P
1	$(b \text{ AND } c) \text{ OR } ((\text{NOT } b) \text{ AND } (d))$
2	$(b \text{ AND } d) \text{ OR } (c \text{ AND } (\text{NOT } d))$
3	$B \text{ XOR } c \text{ XOR } d$
4	$C \text{ XOR } (b \text{ OR } (\text{NOT } d))$

Fig. 4.34 Process P in each round

3. Sample Execution of MD5

Having discussed the MD5 algorithm in detail, now it is time to take a look at the actual values as they appear in a round. As we have seen, the inputs to any round are as follows:

- a
- b
- c
- d
- $M[i = 0 \text{ to } 15]$
- s
- $t[k = 1 \text{ to } 16]$

Once we know these values, we know what to do with them! For the first iteration of the first round, we have the values as follows:

$a = \text{Hex}$	01	23	45	67
$b = \text{Hex}$	89	AB	CD	EF
$c = \text{Hex}$	FE	DC	BA	98
$d = \text{Hex}$	76	54	32	10

$M[0]$ = Whatever is the value of the first 32-bit sub-block

$s = 7$

$t[1] = \text{Hex} \quad D7 \quad 6A \quad A4 \quad 78$

For the second iteration, we move the positions a , b , c and d one position right. So, for the second iteration, we now have:

$a = \text{Output value of } d \text{ of iteration 1}$
$b = \text{Output value of } a \text{ of iteration 1}$
$c = \text{Output value of } b \text{ of iteration 1}$
$d = \text{Output value of } c \text{ of iteration 1}$

$M[1]$ = Whatever is the value of the second 32-bit sub-block

$s = 12$

$t[2] = \text{Hex} \quad E8 \quad C7 \quad B7 \quad 56$

This process will continue for the remaining 14 iterations of round 1, as well as for all the 16 iterations of rounds 2, 3 and 4. In each case, before the iteration begins, we move a , b , c and d to one position right, and use a different s and $t[i]$ defined by MD5 for that step/iteration combination. The actual four rounds are tabulated in Fig. 4.35. The 64 possible values of t are shown after that.

Iteration	a	b	c	d	M	s	t
1	a	b	c	d	$M[0]$	7	$t[1]$
2	d	a	b	c	$M[1]$	12	$t[2]$
3	c	d	a	b	$M[2]$	17	$t[3]$
4	b	c	d	a	$M[3]$	22	$t[4]$
5	a	b	c	d	$M[4]$	7	$t[5]$
6	d	a	b	c	$M[5]$	12	$t[6]$
7	c	d	a	b	$M[6]$	17	$t[7]$
8	b	c	d	a	$M[7]$	22	$t[8]$
9	a	b	c	d	$M[8]$	7	$t[9]$
10	d	a	b	c	$M[9]$	12	$t[10]$
11	c	d	a	b	$M[10]$	17	$t[11]$
12	b	c	d	a	$M[11]$	22	$t[12]$
13	a	b	c	d	$M[12]$	7	$t[13]$
14	d	a	b	c	$M[13]$	12	$t[14]$
15	c	d	a	b	$M[14]$	17	$t[15]$
16	b	c	d	a	$M[15]$	22	$t[16]$

Fig. 4.35 (a) Round 1

Iteration	a	B	c	d	M	s	t
1	a	b	c	d	$M[1]$	5	$t[17]$
2	d	a	b	c	$M[6]$	9	$t[18]$
3	c	d	a	b	$M[11]$	14	$t[19]$
4	b	c	d	a	$M[0]$	20	$t[20]$
5	a	b	c	d	$M[5]$	5	$t[21]$
6	d	a	b	c	$M[10]$	9	$t[22]$
7	c	d	a	b	$M[15]$	14	$t[23]$
8	b	c	d	a	$M[4]$	20	$t[24]$
9	a	b	c	d	$M[9]$	5	$t[25]$
10	d	a	b	c	$M[14]$	9	$t[26]$
11	c	d	a	b	$M[3]$	14	$t[27]$
12	b	c	d	a	$M[8]$	20	$t[28]$
13	a	b	c	d	$M[13]$	5	$t[29]$
14	d	a	b	c	$M[2]$	9	$t[30]$
15	c	d	a	b	$M[7]$	14	$t[31]$
16	b	c	d	a	$M[12]$	20	$t[32]$

Fig. 4.35 (b) Round 2

Iteration	a	b	c	d	M	s	t
1	a	b	c	d	$M[5]$	4	$t[33]$
2	d	a	b	c	$M[8]$	11	$t[34]$
3	c	d	a	b	$M[11]$	16	$t[35]$
4	b	c	d	a	$M[14]$	23	$t[36]$
5	a	b	c	d	$M[1]$	4	$t[37]$
6	d	a	b	c	$M[4]$	11	$t[38]$
7	c	d	a	b	$M[7]$	16	$t[39]$
8	b	c	d	a	$M[10]$	23	$t[40]$
9	a	b	c	d	$M[13]$	4	$t[41]$
10	d	a	b	c	$M[0]$	11	$t[42]$
11	c	d	a	b	$M[3]$	16	$t[43]$
12	b	c	d	a	$M[6]$	23	$t[44]$
13	a	b	c	d	$M[9]$	4	$t[45]$
14	d	a	b	c	$M[12]$	11	$t[46]$
15	c	d	a	b	$M[15]$	16	$t[47]$
16	b	c	d	a	$M[2]$	23	$t[48]$

Fig. 4.35 (c) Round 3

Iteration	a	b	c	d	M	s	t
1	a	b	c	d	$M[0]$	6	$t[49]$
2	d	a	b	c	$M[7]$	10	$t[50]$
3	c	d	a	b	$M[14]$	15	$t[51]$
4	b	c	d	a	$M[5]$	21	$t[52]$
5	a	b	c	d	$M[12]$	6	$t[53]$
6	d	a	b	c	$M[3]$	10	$t[54]$
7	c	d	a	b	$M[10]$	15	$t[55]$
8	b	c	d	a	$M[1]$	21	$t[56]$
9	a	b	c	d	$M[8]$	6	$t[57]$
10	d	a	b	c	$M[15]$	10	$t[58]$
11	c	d	a	b	$M[6]$	15	$t[59]$
12	b	c	d	a	$M[13]$	21	$t[60]$
13	a	b	c	d	$M[4]$	6	$t[61]$
14	d	a	b	c	$M[11]$	10	$t[62]$
15	c	d	a	b	$M[2]$	15	$t[63]$
16	b	c	d	a	$M[9]$	21	$t[64]$

Fig. 4.35 (d) Round 4

The table t contains values (in hex) as shown in Fig. 4.36.

$t[i]$	Value	$t[i]$	Value	$t[i]$	Value	$t[i]$	Value
$t[1]$	D76AA478	$t[17]$	F61E2562	$t[33]$	FFFA3942	$t[49]$	F4292244
$t[2]$	E8C7B756	$t[18]$	C040B340	$t[34]$	8771F681	$t[50]$	432AFF97
$t[3]$	242070DB	$t[19]$	265E5A51	$t[35]$	699D6122	$t[51]$	AB9423A7
$t[4]$	C1BDCEEE	$t[20]$	E9B6C7AA	$t[36]$	FDE5380C	$t[52]$	FC93A039
$t[5]$	F57C0FAF	$t[21]$	D62F105D	$t[37]$	A4BEEA44	$t[53]$	655B59C3
$t[6]$	4787C62A	$t[22]$	02441453	$t[38]$	4BDECFA9	$t[54]$	8F0CCC92
$t[7]$	A8304613	$t[23]$	D8A1E681	$t[39]$	F6BB4B60	$t[55]$	FFEFFF47D
$t[8]$	FD469501	$t[24]$	E7D3FBC8	$t[40]$	BEBFBC70	$t[56]$	85845DD1
$t[9]$	698098D8	$t[25]$	21E1CDE6	$t[41]$	289B7EC6	$t[57]$	6FA87E4F
$t[10]$	8B44F7AF	$t[26]$	C33707D6	$t[42]$	EAA127FA	$t[58]$	FE2CE6E0
$t[11]$	FFFF5BB1	$t[27]$	F4D50D87	$t[43]$	D4EF3085	$t[59]$	A3014314
$t[12]$	895CD7BE	$t[28]$	455A14ED	$t[44]$	04881D05	$t[60]$	4E0811A1
$t[13]$	6B901122	$t[29]$	A9E3E905	$t[45]$	D9D4D039	$t[61]$	F7537E82
$t[14]$	FD987193	$t[30]$	FCEFA3F8	$t[46]$	E6DB99E5	$t[62]$	BD3AF235
$t[15]$	A679438E	$t[31]$	676F02D9	$t[47]$	1FA27CF8	$t[63]$	2AD7D2BB
$t[16]$	49B40821	$t[32]$	8D2A4C8A	$t[48]$	C4AC5665	$t[64]$	EB86D391

Fig. 4.36 Values of the table t

4. MD5 versus MD4

Let us list down the key differences between MD5 and its predecessor, MD4, as shown in Fig. 4.37.

Point of discussion	MD4	MD5
Number of rounds	3	4
Use of additive constant t	Not different in all the iterations	Different in all the iterations
Process P in round 2	((b AND c) OR (b AND d) OR (c AND d))	(b AND d) OR (c AND (NOT d))— <i>This is more random</i>

Fig. 4.37 Differences between MD5 and MD4

Additionally, the order of accessing the sub-blocks in rounds 2 and 3 is changed to introduce more randomness.

5. The Strength of MD5

We can see how complex MD5 can get! The attempt of Rivest was to add as much complexity and randomness as possible to the MD5 algorithm, so that no two message digests produced by MD5 on any two different messages are equal. MD5 has a property that every bit of the message digest is some function of every bit in the input. The possibility that two messages produce the same message digest using MD5 is in the order of 2^{64} operations. Given a message digest, working backwards to find the original message can lead up to 2^{128} operations.

The following attacks have been launched against MD5.

- (a) Tom Berson could find two messages that produce the same message digest for each of the four individual rounds. However, he could not come up with two messages that produce the same message digest for all the four rounds taken together.

- (b) den Boer and Bosselaers showed that the execution of MD5 on a single block of 512 bits will produce the same output for two different values in the chaining variable register *abcd*. This is called **pseudocollision**. However, they could not extend this to a full MD5 consisting of four rounds, each containing 16 steps.
- (c) Dobbertin provided the most serious attack on MD5. Using his attack, the operation of MD5 on two different 512-bit blocks produces the same 128-bit output. However, this has not been generalized to a full message block.

The general recommendation now is not to trust MD5 (although it is not practically broken into, as yet). Consequently, the quest for better message algorithms has led to the possible alternative, called **SHA-1**. We will study it now.

4.7.4 Secure Hash Algorithm (SHA)

1. Introduction

The National Institute of Standards and Technology (NIST) along with NSA developed the **Secure Hash Algorithm (SHA)**. In 1993, SHA was published as a Federal Information Processing Standard (FIPS PUB 180). It was revised to FIPS PUB 180-1 in 1995, and the name was changed to SHA-1. As we shall study, SHA is a modified version of MD4, and its design closely resembles MD4.

SHA works with any input message that is less than 2^{64} bits in length. The output of SHA is a message digest, which is 160 bits in length (32 bits more than the message digest produced by MD5). The word *Secure* in SHA was decided based on two features. SHA is designed to be computationally infeasible to

- (a) obtain the original message, given its message digest, and
- (b) find two messages producing the same message digest.

2. The Working of SHA

As we have mentioned before, SHA is closely modeled after MD5. Therefore, we shall not discuss in detail those features of SHA, which are similar to MD5. Instead, we shall simply mention them and point out the differences. The reader can go back to the appropriate descriptions of MD5 to find out more details.

Step 1: Padding Like MD5, the first step in SHA is to add padding to the end of the original message in such a way that the length of the message is 64 bits short of a multiple of 512. Like MD5, the padding is always added, even if the message is already 64 bits short of a multiple of 512.

Step 2: Append Length The length of the message excluding the length of the padding is now calculated and appended to the end of the padding as a 64-bit block.

Step 3: Divide the Input into 512-bit Blocks The input message is now divided into blocks, each of length 512 bits. These blocks become the input to the message-digest processing logic.

Step 4: Initialize Chaining Variables Now, five *chaining variables* *A* through *E* are initialized. Remember that we had four chaining variables, each of 32 bits in MD5 (which made the total length of the variables $4 \times 32 = 128$ bits). Recall that we stored the intermediate as well as the final results into the combined register made up of these four chaining variables, i.e. *abcd*. Since in the case of SHA, we want to produce a message digest of length 160 bits, we need to have five chaining variables here ($5 \times 32 = 160$ bits). In SHA, the variables *A* through *D* have the same values as they had in MD5. Additionally, *E* is initialized to Hex *C3 D2 E1 F0*.

Step 5: Process Blocks Now the actual algorithm begins. Here also, the steps are quite similar to those in MD5.

Step 5.1 Copy the chaining variables $A-E$ into variables $a-e$. The combination of $a-e$, called $abcde$, will be considered as a single register for storing the temporary intermediate as well as the final results.

Step 5.2 Now, divide the current 512-bit block into 16 sub-blocks, each consisting of 32 bits.

Step 5.3 SHA has four rounds, each round consisting of 20 steps. Each round takes the current 512-bit block, the register $abcde$ and a constant $K[t]$ (where $t = 0$ to 79) as the three inputs. It then updates the contents of the register $abcde$ using the SHA algorithm steps. Also notable is the fact that we had 64 constants defined as t in MD5. Here, we have only four constants defined for $K[t]$, one used in each of the four rounds. The values of $K[t]$ are as shown in Fig. 4.38.

Round	Value of t between	$K[t]$ in hexadecimal	$K[t]$ in decimal (Only integer portion of the value shown)
1	1 and 19	5A 92 79 99	$2^{30} \times \sqrt{2}$
2	20 and 39	6E D9 EB A1	$2^{30} \times \sqrt{3}$
3	40 and 59	9F 1B BC DC	$2^{30} \times \sqrt{5}$
4	60 and 79	CA 62 C1 D6	$2^{30} \times \sqrt{10}$

Fig. 4.38 Values of $K[t]$

Step 5.4 SHA consists of four rounds, each round containing 20 iterations. This makes it a total of 80 iterations. The logical operation of a single SHA iteration looks as shown in Fig. 4.39.

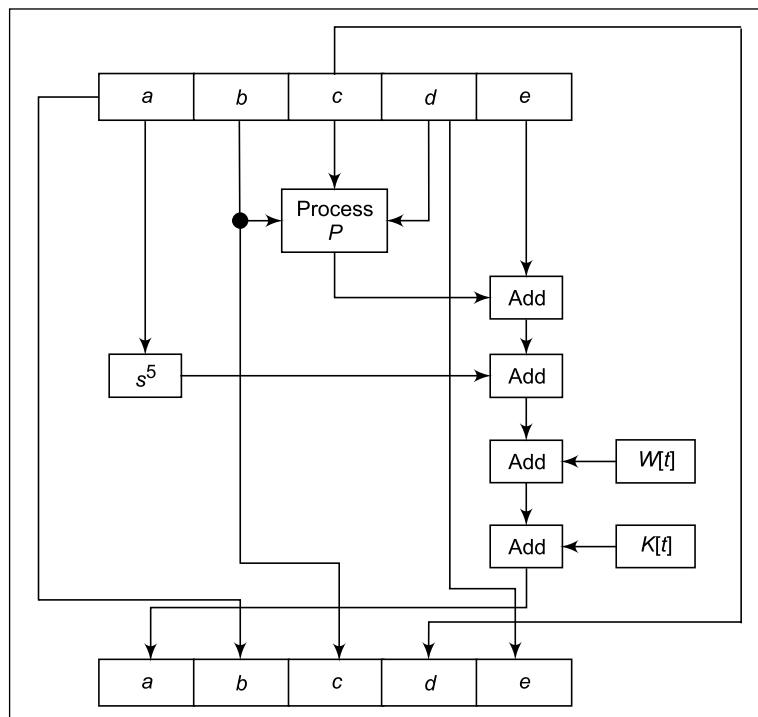


Fig. 4.39 Single SHA-1 iteration

Mathematically, an iteration consists of the following operations:

$$abcde = (e + \text{Process } P + s^5(a) + W[t] + K[t]), a, s^{30}(b), c, d$$

where,

$abcde$ = The register made up of the five variables a, b, c, d and e

$\text{Process } P$ = The logical operation, which we shall study later

s^t = Circular-left shift of the 32-bit sub-block by t bits

$W[t]$ = A 32-bit derived from the current 32-bit sub-block, as we shall study later

$K[t]$ = One of the five additive constants, as defined earlier

We will notice that this is very similar to MD5, with a few variations (which are induced to try and make SHA more complicated in comparison with MD5). We have to now see what are the meanings of the process P and $W[t]$ in the above equation.

Figure 4.40 illustrates the process P .

Round	Process P
1	(b AND c) OR ((NOT b) AND (d))
2	B XOR c XOR d
3	(b AND c) OR (b and D) OR (c AND d)
4	B XOR c XOR d

Fig. 4.40 Process P in each SHA-1 round

The values of $W[t]$ are calculated as follows:

For the first 16 words of W (i.e. $t = 0$ to 15), the contents of the input message sub-block $M[t]$ become the contents of $W[t]$ straightaway. That is, the first 16 blocks of the input message M are copied to W .

The remaining 64 values of W are derived using the equation:

$$W[t] = s^1(W[t - 16] \text{ XOR } W[t - 14] \text{ XOR } W[t - 8] \text{ XOR } W[t - 3])$$

As before, s^1 indicates a circular-left shift (i.e. rotation) by 1 bit position.

Thus, we can summarize the values of W as shown in Fig. 4.41.

For $t = 0$ to 15	Value of $W[t]$
$W[t] =$	Same as $M[t]$
$W[t] =$	$s^1(W[t - 16] \text{ XOR } W[t - 14] \text{ XOR } W[t - 8] \text{ XOR } W[t - 3])$

Fig. 4.41 Values of W in SHA-1

3. Comparison of MD5 and SHA-1

As we know, the basis for both MD5 and SHA is the MD4 algorithm. Therefore, it is quite reasonable to compare MD5 and SHA, and see what the difference between the two is, as depicted in Fig. 4.42.

4. Security of SHA-1

In 2005, a possible vulnerability was discovered in SHA-1. Just before that, NIST had announced its intentions to seek more secure versions of SHA by 2010. Hence, various options are being discussed

Point of discussion	MD5	SHA
Message-digest length in bits	128	160
Attack to try and find the original message given a message digest	Requires 2^{128} operations to break in	Requires 2^{160} operations to break in, therefore more secure
Attack to try and find two messages producing the same message digest	Requires 2^{64} operations to break in	Requires 2^{80} operations to break in
Successful attacks so far	There have been reported attempts to some extent (as we discussed earlier)	No such claims so far
Speed	Faster (64 iterations, and 128-bit buffer)	Slower (80 iterations, and 160-bit buffer)
Software implementation	Simple, does not need any large programs or complex tables	Simple, does not need any large programs or complex tables

Fig. 4.42 Comparison of MD5 and SHA-1

now. In 2002, NIST had come up with a new version of SHA in standard document FIPS 1802, called SHA-256, SHA-284, and SHA-512; with the number after the word SHA indicating the length of the message digest in bits. Figure 4.43 summarizes the various SHA versions.

Parameter	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size (in bits)	160	256	384	512
Message size (in bits)	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size (in bits)	512	512	1024	1024
Word size (in bits)	32	32	64	64
Steps in algorithm	80	64	80	80

Fig. 4.43 Parameters for the Versions of SHA

As a consequence, we need to understand how a *more secure SHA* algorithm works. Therefore, we now discuss SHA-512. Naturally, SHA-512 works similar to SHA-1 with some additions/changes.

4.7.5 SHA-512

The SHA-512 algorithm takes a message of length 2^{128} bits, and produces a message digest of size 512 bits. The input is divided into blocks of size 1024 bits each.

SHA-512 is closely modeled after SHA-1, which itself is modeled on MD5. Therefore, we shall not discuss in detail those features of SHA-512, which are similar to these two algorithms. Instead, we shall simply mention them and point out the differences. The reader can go back to the appropriate descriptions of MD5 to find out more details.

Step 1: Padding

Like MD5 and SHA-1, the first step in SHA is to add padding to the end of the original message in such a way that the length of the message is 128 bits short of a multiple of 1024. Like MD5 and SHA-1, the padding is always added, even if the message is already 128 bits short of a multiple of 1024.

Step 2: Append Length

The length of the message excluding the length of the padding is now calculated and appended to the end of the padding as a 128-bit block. Hence, the length of the message is exactly a multiple of 1024 bits.

Step 3: Divide the Input into 1024-bit Blocks

The input message is now divided into blocks, each of length 1024 bits. These blocks become the input to the message-digest processing logic.

Step 4: Initialize Chaining Variables

Now, eight *chaining variables*, a through h , are initialized. Remember that we had (a) four chaining variables, each of 32 bits in MD5 (which made the total length of the variables $4 \times 32 = 128$ bits), and (b) five chaining variables each of 32 bits (which made the total length of the variables $5 \times 32 = 160$ bits) in SHA-1. Recall that we stored the intermediate as well as the final results into the combined register made up of these chaining variables, i.e. $abcd$ in MD5 and $abcde$ in SHA-1. Since in the case of SHA-256, as we want to produce a message digest of length 512 bits, we need to have eight chaining variables, each containing 64 bits here ($8 \times 64 = 512$ bits). In SHA-512, these eight variables have values as shown in Fig. 4.44.

$A = 6A09E667F3BCC908$	$B = BB67AE8584CAA73B$
$C = 3C6EF372FE94F82B$	$D = A54FF53A5F1D36F1$
$E = 510E527FADE682D1$	$F = 9B05688C2B3E6C1F$
$G = 1F83D9ABFB41BD6B$	$H = 5BE0CD19137E2179$

Fig. 4.44 SHA-512 chaining variables

Step 5: Process Blocks

Now the actual algorithm begins. Here also, the steps are quite similar to those in MD5 and SHA-1.

Step 5.1 Copy the chaining variables $A-H$ into variables $a-h$. The combination of $a-h$, called $abcdefg$, will be considered as a single register for storing the temporary intermediate as well as the final results.

Step 5.2 Now, divide the current 1024-bit block into 16 sub-blocks, each consisting of 64 bits.

Step 5.3 SHA-512 has 80 rounds. Each round takes the current 1024-bit block, the register $abcdefg$ and a constant $K[t]$ (where $t = 0$ to 79) as the three inputs. It then updates the contents of the register $abcdefg$ using the SHA-512 algorithm steps. The operation of a single round is shown in Fig. 4.45.

Each round consists of the following operations:

$$\text{Temp1} = h + \text{Ch}(e, f, g) + \text{Sum}(e_i \text{ for } i = 1 \text{ to } 512) + W_t + K_t$$

$$\text{Temp2} = \text{Sum}(a_i \text{ for } i = 0 \text{ to } 512) + \text{Maj}(a, b, c)$$

$$a = \text{Temp1} + \text{Temp2}$$

$$b = a$$

$$c = b$$

$$\begin{aligned}
 d &= c \\
 e &= d + \text{Temp1} \\
 f &= e \\
 g &= f \\
 h &= g
 \end{aligned}$$

where:

t = Round number

$\text{Ch}(e, f, g) = (e \text{ AND } f) \text{ XOR } (\text{NOT } e \text{ AND } g)$

$\text{Maj}(a, b, c) = (a \text{ AND } b) \text{ XOR } (a \text{ AND } c) \text{ XOR } (b \text{ AND } c)$

$\text{Sum}(a) = \text{ROTR}(a_i \text{ by 28 bits}) \text{ XOR } \text{ROTR}(a_i \text{ by 34 bits}) \text{ XOR } \text{ROTR}(a_i \text{ by 39 bits})$

$\text{Sum}(e_i) = \text{ROTR}(e_i \text{ by 14 bits}) \text{ XOR } \text{ROTR}(e_i \text{ by 18 bits}) \text{ XOR } \text{ROTR}(e_i \text{ by 41 bits})$

$\text{ROTR}(x)$ = Circular right shift, i.e. rotation, of the 64-bit array x the specified number of bits

W_t = 64-bit word derived from the current 512-bit input block

K_t = 64-bit additive constant

+ (or Add) = Addition mod 2^{64}

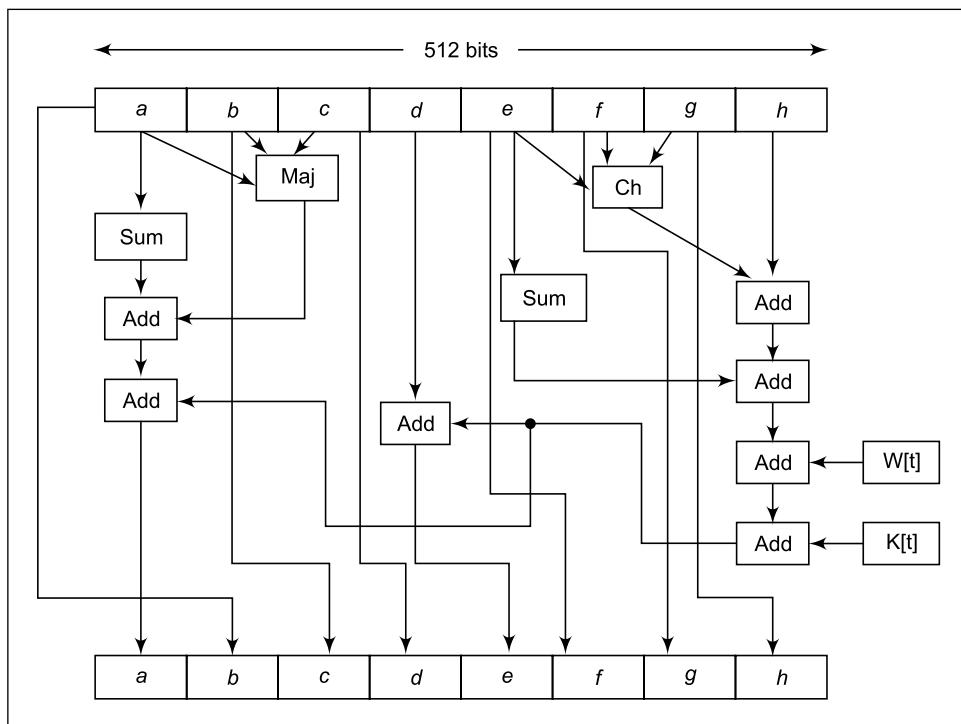


Fig. 4.45 Single SHA-512 iteration

The 64-bit word values for W_i are derived from the 1024-bit message using certain mappings, which we shall not describe here. Instead, we will simply point out this:

- For the first 16 rounds (0 to 15), the value of W_i is equal to the corresponding word in the message block.
- For the remaining 64 steps, the value of W_i is equal to the circular left shift by one bit of the XOR of the four preceding values of W_i , with two of them subjected to shift and rotate operations.

This makes the message digest more complex and difficult to break.

4.7.6 SHA-3

The MD5 algorithm is broken, but nobody has been able to break SHA-1 so far. However, because SHA-1 is quite similar in nature to MD5, it is feared that in the future, it can be broken as well. Hence, more and more people have started using the next version of the SHA family of algorithms, collectively known as **SHA-2** (i.e. SHA-256, SHA-384, and SHA-512). SHA-512 is considered to be the toughest of them to break. However, considering the fact that at some point of time in the future, SHA-2 would also get broken, a search for the next leap in message-digest algorithms is being taken by searching for **SHA-3**.

A competition was announced in 2007 for coming up with SHA-3 by NIST. Certain pre-requisites need to be fulfilled by SHA-3:

1. The application need not have to make extensive changes to replace SHA-2 with SHA-3. In other words, we should be able to simply replace the earlier algorithms with SHA-3. This automatically means that SHA-3 must support message digests of lengths 224, 256, 384, and 512 bits.
2. The basic characteristic of being able to process smaller blocks of original text to generate the message digest in SHA-2 needs to be carried forward in SHA-3.

Other expectations from SHA-3 are that it should be as secure as possible while defeating attacks that are likely to succeed on SHA-2. In other words, SHA-3 should be different in its operation from MD5 and SHA-1/SHA-2. It is also expected to work quite fast while consuming minimum resources. Also, it should be simple and it should be possible to add parameters to it to make it more flexible.

4.7.7 Message Authentication Code (MAC)

The concept of **Message Authentication Code (MAC)** is quite similar to that of a message digest. However, there is one difference. As we have seen, a message digest is simply a fingerprint of a message. There is no cryptographic process involved in the case of message digests. In contrast, a MAC requires that the sender and the receiver should know a shared symmetric (secret) key, which is used in the preparation of the MAC. Thus, MAC involves cryptographic processing. Let us see how this works.

Let us assume that the sender A wants to send a message M to a receiver B . How the MAC processing works is shown in Fig. 4.46.

1. A and B share a symmetric (secret) key K , which is not known to anyone else. A calculates the MAC by applying key K to the message M .
2. A then sends the original message M and the MAC $H1$ to B .

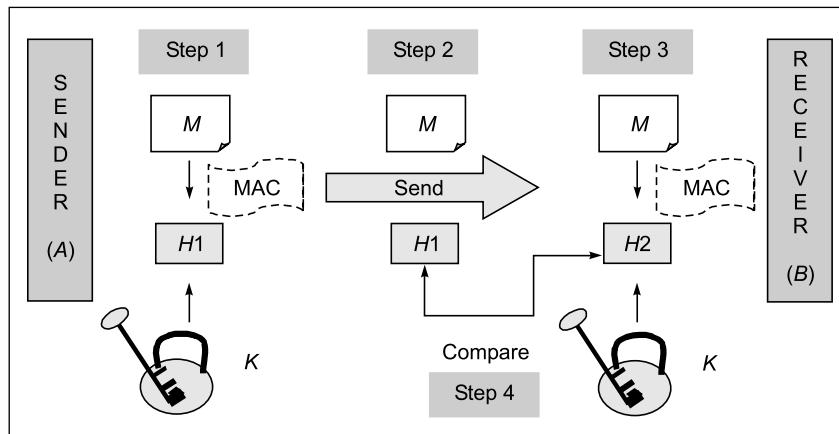


Fig. 4.46 Message Authentication Code (MAC)

3. When B receives the message, B also uses K to calculate its own MAC $H2$ over M .
4. B now compares $H1$ with $H2$. If the two match, B concludes that the message M has not been changed during transit. However, if $H1 \neq H2$, B rejects the message, realizing that the message was changed during transit.

The significances of a MAC are as follows:

1. The MAC assures the receiver (in this case, B) that the message is not altered. This is because if an attacker alters the message but does not alter the MAC (in this case, $H1$) then the receiver's calculation of the MAC (in this case, $H2$) will differ from it. Why does the attacker then not also alter the MAC? Well, as we know, the key used in the calculation of the MAC (in this case, K) is assumed to be known only to the sender and the receiver (in this case, A and B). Therefore, the attacker does not know the key, K , and therefore, she cannot alter the MAC.
2. The receiver (in this case, B) is assured that the message indeed came from the correct sender (in this case, A). Since only the sender and the receiver (A and B , respectively, in this case) know the secret key (in this case, K), no one else could have calculated the MAC (in this case, $H1$) sent by the sender (in this case, A).

Interestingly, although the calculation of the MAC seems to be quite similar to an encryption process, it is actually different in one important respect. As we know, in symmetric-key cryptography, the cryptographic process must be reversible. That is, the encryption and decryption are the mirror images of each other. However, note that in the case of MAC, both the sender and the receiver are performing encryption process only. Thus, a MAC algorithm need not be reversible—it is sufficient to be a one-way function (encryption) only.

We have already discussed two main message-digest algorithms, namely MD5 and SHA-1. Can we reuse these algorithms for calculating a MAC, in their original form? Unfortunately, we cannot reuse them, because they do not involve the usage of a secret key, which is the basis of MAC. Consequently, we must have a separate practical algorithm implementation for MAC. The solution is **HMAC**, a practical algorithm to implement MAC.

4.7.8 HMAC

1. Introduction

HMAC stands for **Hash-based Message Authentication Code**. HMAC has been chosen as a mandatory security implementation for the Internet Protocol (IP) security, and is also used in the Secure Socket Layer (SSL) protocol, widely used on the Internet.

The fundamental idea behind HMAC is to reuse the existing message-digest algorithms, such as MD5 or SHA-1. Obviously, there is no point in reinventing the wheel. Therefore, what HMAC does is to work with any message-digest algorithm. That is, it treats the message digest as a black box. Additionally, it uses the shared symmetric key to encrypt the message digest, which produces the output MAC. This is shown in Fig. 4.47.

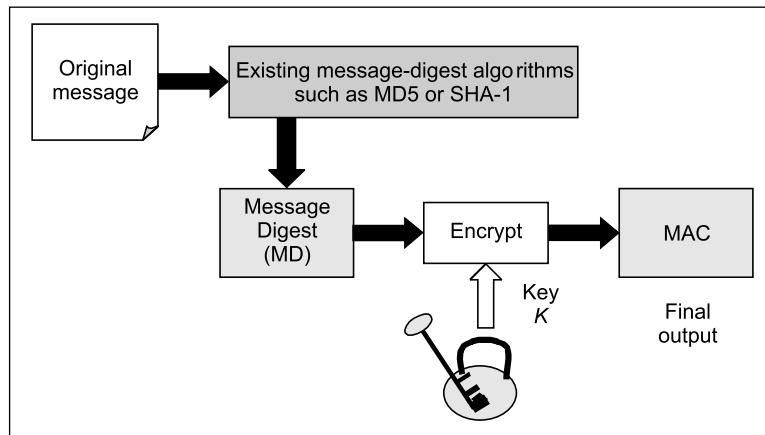


Fig. 4.47 HMAC concept

2. The Working of HMAC

Let us now take a look at the internal working of HMAC. For this, let us start with the various variables that will be used in our HMAC discussion.

MD = The message digest/hash function used (e.g. MD5, SHA-1, etc.)

M = The input message whose MAC is to be calculated

L = The number of blocks in the message M

b = The number of bits in each block

K = The shared symmetric key to be used in HMAC

$ipad$ = A string 00110110 repeated $b/8$ times

$opad$ = A string 01011010 repeated $b/8$ times

Armed with this input, we shall use a step-by-step approach to understand the HMAC operation.

Step 1: Make the length of K equal to b The algorithm starts with three possibilities, depending on the length of the key K :

- **Length of $K < b$** In this case, we need to expand the key (K) to make the length of K equal to the number of bits in the original message block (i.e. b). For this, we add as many 0 bits as required to

the left of K . For example, if the initial length of $K = 170$ bits, and $b = 512$ then we add 342 bits, all with a value 0, to the left of K . We shall continue to call this modified key as K .

- **Length of $K = b$** In this case, we do not take any action, and proceed to step 2.
- **Length of $K > b$** In this case, we need to trim K to make the length of K equal to the number of bits in the original message block (i.e. b). For this, we pass K through the message-digest algorithm (H) selected for this particular instance of HMAC, which will give us a key K , trimmed so that its length is equal to b .

This is shown in Fig. 4.48.

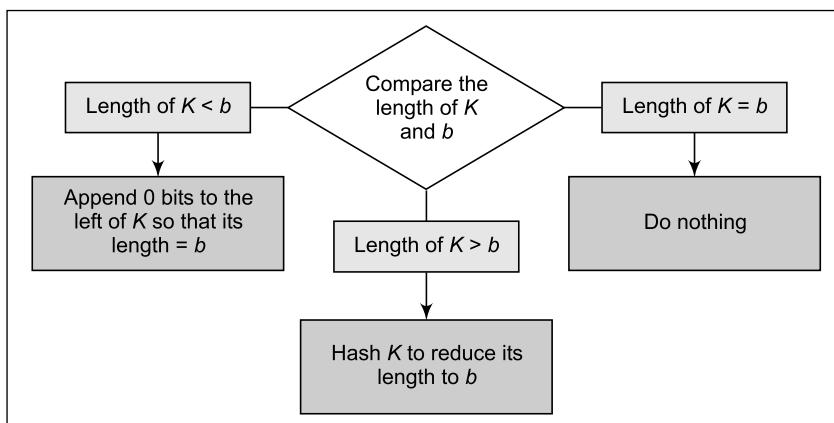


Fig. 4.48 Step 1 of HMAC

Step 2: XOR K with ipad to produce $S1$ We XOR K (the output of step 1) and $ipad$ to produce a variable called $S1$. This is shown in Fig. 4.49.

Step 3: Append M to $S1$ We now take the original message (M) and simply append it to the end of $S1$ (which was calculated in step 2). This is shown in Fig. 4.50.

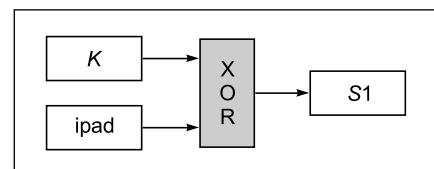


Fig. 4.49 Step 2 of HMAC

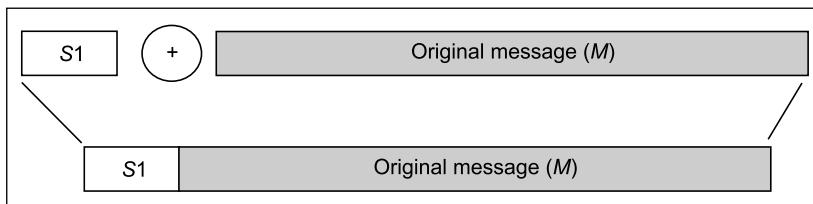


Fig. 4.50 Step 3 of HMAC

Step 4: Message-digest algorithm Now, the selected message-digest algorithm (e.g. MD5, SHA-1, etc.) is applied to the output of step 3 (i.e. to the combination of $S1$ and M). Let us call the output of this operation as H . This is shown in Fig. 4.51.

Step 5: XOR K with opad to produce $S2$ Now, we XOR K (the output of step 1) with $opad$ to produce a variable called as $S2$. This is shown in Fig. 4.52.

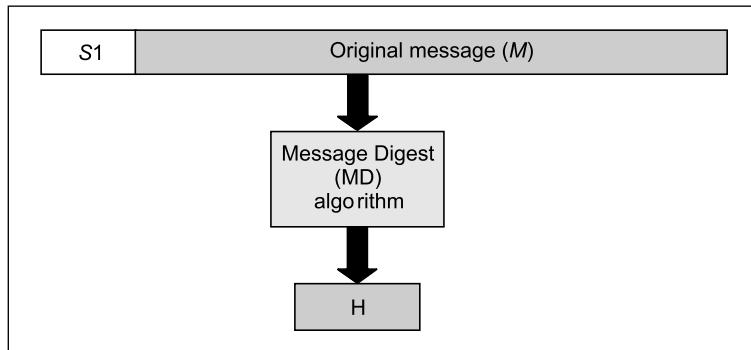


Fig. 4.51 Step 4 of HMAC

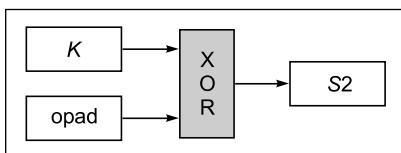


Fig. 4.52 Step 5 of HMAC

Step 6: Append H to $S2$ In this step, we take the message digest calculated in step 4 (i.e. H) and simply append it to the end of $S2$ (which was calculated in step 5). This is shown in Fig. 4.53.

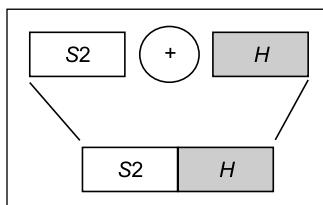


Fig. 4.53 Step 6 of HMAC

Step 7: Message-digest algorithm Now, the selected message-digest algorithm (e.g. MD5, SHA-1, etc.) is applied to the output of step 6 (i.e. to the concatenation of $S2$ and H). This is the final MAC that we want. This is shown in Fig. 4.54.

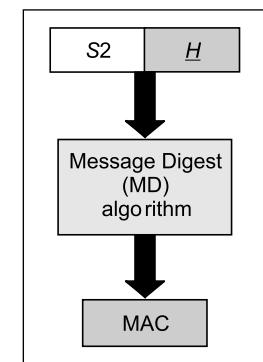


Fig. 4.54 Step 7 of HMAC

Let us summarize the seven steps of HMAC, as shown in Fig. 4.55.

3. Disadvantages of HMAC

It appears that the MAC produced by the HMAC algorithm fulfills our requirements of a digital signature. From a logical perspective, firstly we calculate a fingerprint (message digest) of the original message, and then encrypt it with a symmetric key, which is known only to the sender and the receiver. This gives sufficient confidence to the receiver that the message came from the correct sender, and also that it was not altered during the transit. However, if we observe the HMAC scheme carefully, we will realize that it does not solve all our problems. What are these problems?

1. We assume in HMAC that only the sender and the receiver know about the symmetric key. However, we have studied in great detail that the problem of symmetric-key exchange is quite serious, and cannot be solved easily. The same problem of key exchange is present in the case of HMAC.

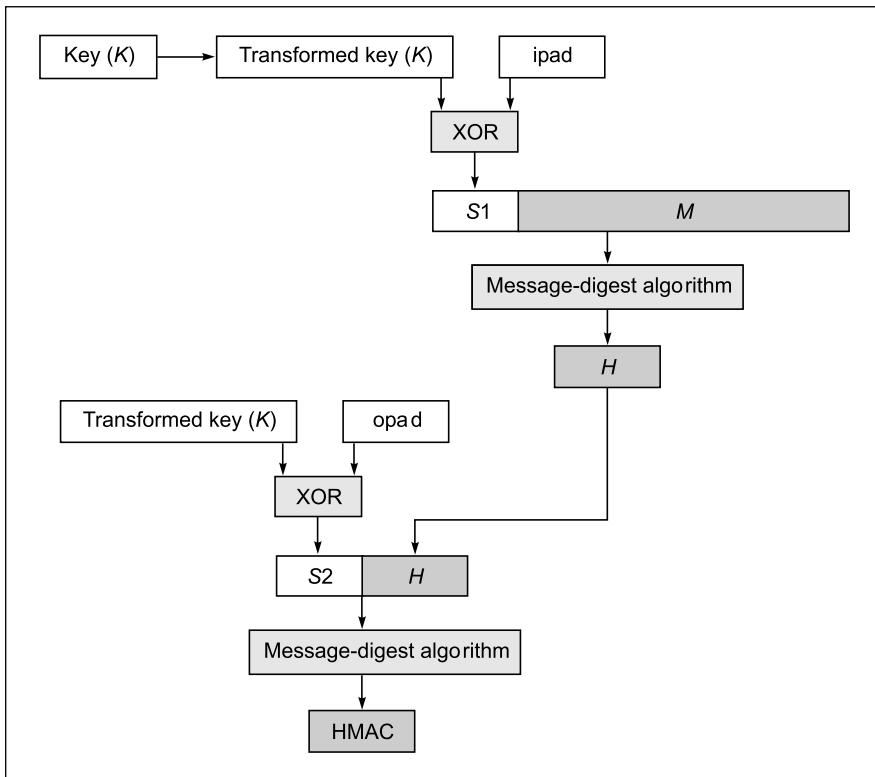


Fig. 4.55 Complete HMAC operation

2. Even if we assume that somehow the key-exchange problem is resolved, HMAC cannot be used if the number of receivers is greater than one. This is because, to produce a MAC by using HMAC, we need to make use of a symmetric key. The symmetric key is supposed to be shared only by two parties: one sender and one receiver. Of course, this problem can be solved if multiple parties (one sender and all the receivers) share the same symmetric key. However, this resolution leads to a third problem.
3. The new problem is that how does a receiver know that the message was prepared and sent by the sender, and not by one of the *other* receivers? After all, all the co-receivers also know the symmetric key. Therefore, it is quite possible that one of the co-receivers might have created a false message on behalf of the alleged sender, and using HMAC, the co-receiver might have prepared a MAC for the message, and sent the message and the MAC as if it originated at the alleged sender! There is no way to prevent or detect this.
4. Even if we somehow solve the above problem, one major concern remains. Let us go back to our simple case of one sender and one receiver. Now, only two parties—the sender (say *A*, a bank customer) and the receiver (say *B*, a bank) share the symmetric-key secret. Suppose that one fine day, *B* transfers all the balance standing in the account of *A* to a third person's account, and closes *A*'s bank account. *A* is shocked, and files a suit against *B*. In the court of law, *B* argues that *A* had sent an electronic message in order to perform this transaction, and produces that message as evidence. *A* claims that she never sent that message, and that it is a forged message. Fortunately,

the message produced by B as evidence also contained a MAC, which was produced on the original message. As we know, only encrypting the message digest of the original message with the symmetric key shared by A and B could have produced it—and here is where the trouble is! Even though we have a MAC, how in the world are we now going to prove that the MAC was produced by A or by B ? After all, both know about the shared secret key! It is equally possible for either of them to have created the original message and the MAC!

As it turns out, even if we are able to somehow resolve the first three problems, we have no solution for the fourth problem. Therefore, we cannot trust HMAC to be used in digital signatures. Better schemes are required for increased security.

4.7.9 Digital Signature Techniques

1. Introduction

Due to the problems associated with MAC as mentioned earlier, the **Digital Signature Standard (DSS)** was developed for performing digital signatures. The National Institute of Standards and Technology (NIST) published the DSS standard as the Federal Information Processing Standard (FIPS) PUB 186 in 1991, which was revised in 1993 and 1996. DSS makes use of the SHA-1 algorithm for calculating the message digest over an original message, and uses the message digest to perform the digital signature, as we shall study. For this, DSS makes use of an algorithm, called **Digital Signature Algorithm (DSA)**. Note that DSS is the standard, and DSA is the actual algorithm.

One important point needs to be stated here. Like RSA, DSA is also based on asymmetric-key cryptography. However, their objectives are totally different. As we know, RSA is primarily used for encrypting a message. As we shall see, additionally, RSA can also be used for performing a digital signature over a message. In contrast, DSA is used only for performing digital signature over a message. It cannot be used for encryption.

2. The Politics of Digital Signature Algorithms

The acceptance of DSA was not straightforward. As we know, the RSA algorithm can also be used for performing digital signatures. One of the aims of NIST—the developers of DSA—was to make DSA a free piece of digital-signature software. However, RSA Data Security Inc. (RSADSI), which controls the licensing of all RSA products (which are not free) had invested a great amount of money and efforts in the RSA algorithm. Therefore, they were quite keen to promote RSA, and not DSA, as the algorithm of choice for digital signatures. Moreover, big companies such as IBM, Novell, Lotus, Apple, Microsoft, DEC, Sun, Northern Telecom, etc., had made large investments in implementing the RSA algorithm. Therefore, they were also against the use of DSA. There were a lot of allegations and speculations regarding the strength of DSA. All of them were addressed, making DSA a reliable algorithm. However, all problems are not resolved yet!

Although NIST holds the patent on DSA, at least three other parties also claim that DSA infringes on their patents. This issue is still not completely sorted out.

Before we study how DSA works, let us understand how RSA can be used for performing digital signatures.

3. RSA and Digital Signatures

We have mentioned that RSA can be used for performing digital signatures. Let us understand how this works in a step-by-step fashion. For this, let us assume that the sender (*A*) wants to send a message *M* to the receiver (*B*) along with the digital signature (*S*) calculated over the message (*M*).

Step 1 The sender (*A*) uses the SHA-1 message-digest algorithm to calculate the message digest (*MD1*) over the original message (*M*). This is shown in Fig. 4.56.

Step 2 The sender (*A*) now encrypts the message digest with her private key. The output of this process is called the digital signature (*DS*) of *A*. This is shown in Fig. 4.57.

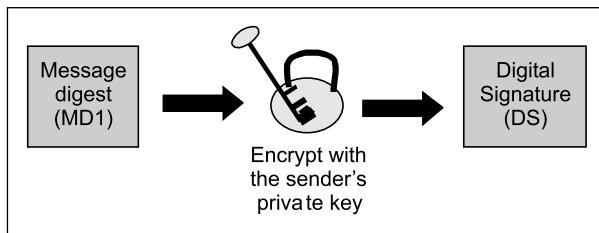


Fig. 4.57 Digital-signature creation

Step 3 Now the sender (*A*) sends the original message (*M*) along with the digital signature (*DS*) to the receiver (*B*). This is shown in Fig. 4.58.

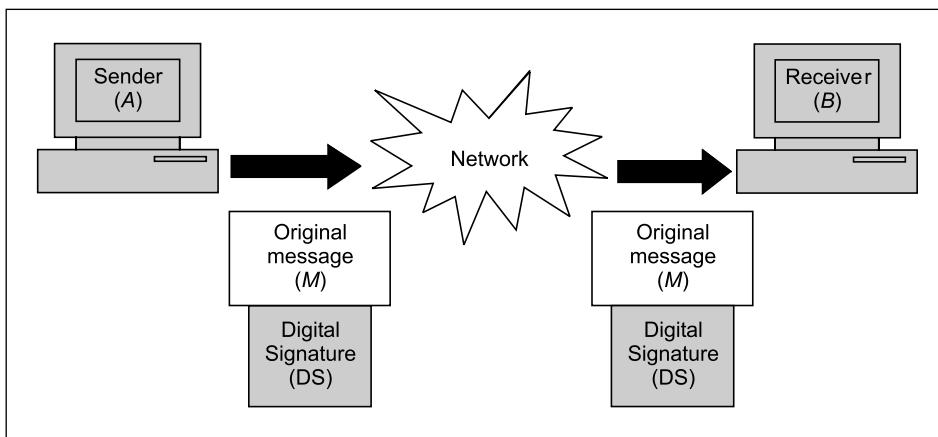


Fig. 4.58 Transmission of original message and digital signature simultaneously

Step 4 After the receiver (*B*) receives the original message (*M*) and the sender's (*A*'s) digital signature, *B* uses the same message-digest algorithm as was used by *A*, and calculates its own message digest (*MD2*) as shown in Fig. 4.59.

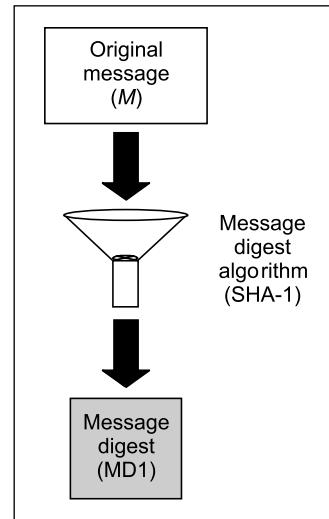


Fig. 4.56 Message-digest calculation

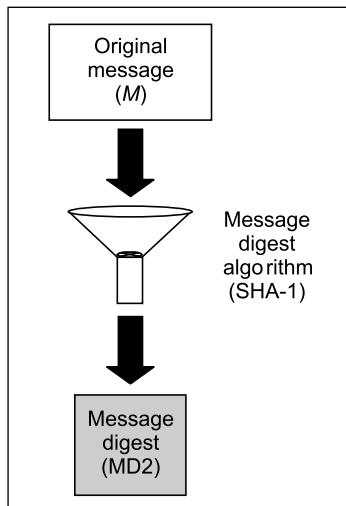


Fig. 4.59 Receiver calculates its own message digest

Step 5 The receiver (*B*) now uses the sender's (*A*'s) public key to decrypt (sometimes also called **de-sign**) the digital signature. Note that *A* had used her private key to encrypt her message digest (MD1) to form the digital signature. Therefore, only *A*'s public key can be used to decrypt it. The output of this process is the original message digest as was calculated by *A* (MD1) in step 1. This is shown in Fig. 4.60.

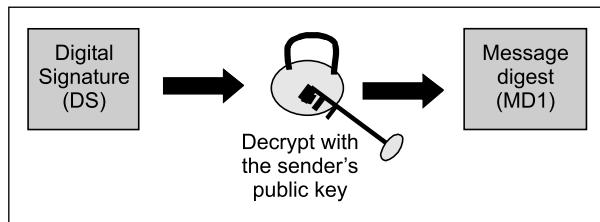


Fig. 4.60 Receiver retrieves sender's message digest

Step 6 *B* now compares the following two message digests:

- MD2, which it had calculated in step 4
- MD1, which it retrieved from *A*'s digital signature in step 5

If MD1 = MD2, the following facts are established:

- *B* accepts the original message (*M*) as the correct, unaltered, message from *A*.
- *B* is also assured that the message came from *A*, and not from someone posing as *A*.

This is shown in Fig. 4.61.

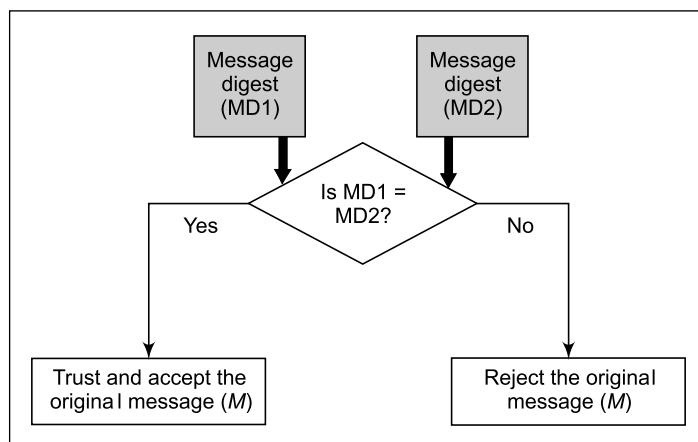


Fig. 4.61 Digital-signature verification

The basis for the acceptance or the rejection of the original message on the basis of the outcome of the message-digest comparison (i.e. step 6) is simple. We know that the sender (*A*) had used her private

key to encrypt the message digest to produce the digital signature. If decrypting the digital signature produces the correct message digest, the receiver (B) can be quite sure that the original message and the digital signature came indeed from the sender (A). This also proves that the message was not altered by an attacker while in transit. Because, if the message was altered while in transit, the message digest calculated by B in step 4 (i.e. MD2) over the received message would differ from the one sent (of course, in encrypted form) by A (i.e. MD1). Why can the attacker not alter the message, recalculate the message digest, and sign it again? Well, as we know, the attacker can very well perform the first two steps (i.e. alter the message, and recalculate the message digest over the altered message); but it cannot sign it again, because for that to be possible, the attacker needs A 's private key. Since only A knows about A 's private key, the attacker cannot use A 's private key to encrypt the message digest (i.e. sign the message) again.

Thus, the principle of digital signatures is quite strong, secure and reliable.

4. Attacks on RSA Signature

Some attacks are attempted by attackers on RSA digital signatures. We briefly outline the important ones below.

(a) Chosen-message Attack In the **chosen-message attack**, the attacker creates two different messages, M_1 and M_2 . They need not have close resemblance. The attacker somehow manages to persuade the genuine user to sign these two messages M_1 and M_2 using the RSA digital-signature scheme. After these attempts are successful, the attacker computes a new message $M = M_1 \times M_2$ and then claims that the genuine user has signed this message M .

(b) Key-only Attack In the **key-only attack**, the assumption is that the attacker only has access to the genuine user's public key. The attacker somehow then obtains a genuine message M and its signature S . The attacker then tries to create another message MM such that the same signature S looks to be valid on MM . However, it is not an easy attack to launch since the mathematical complexity beyond this is quite high.

(c) Known-message Attack In the **known-message attack**, the attacker tries to use a feature of RSA whereby two different messages having two different signatures can be so combined so that their signatures also combine. To take an example, let us say that we have two different messages M_1 and M_2 with respective digital signatures as S_1 and S_2 . Then if $M = (M_1 \times M_2) \text{ mod } n$, mathematically $S = (S_1 \times S_2) \text{ mod } n$. Hence, the attacker can compute $M = (M_1 \times M_2) \text{ mod } n$ and then $S = (S_1 \times S_2) \text{ mod } n$ to forge a signature.

5. DSA and Digital Signatures

The description of DSA is quite complicated and mathematical in nature. The reader can safely skip it without any loss of continuity.

The DSA algorithm makes use of the following variables:

p = A prime number of length L bits. L = A multiple of 64 between 512 and 1024 (i.e. $L = 512$ or 576 or 640 or ... 1024). In the original standard, p was always 512 bits long. This led to a lot of technical criticism, and was changed by NIST.

$q = A$ 160-bit prime factor of $(p - 1)$.

$g = h^{(p-1)/q} \text{ mod } p$, where h is a number less than $(p - 1)$ such that $h^{(p-1)/q} \text{ mod } p$ is greater than 1.

x = A number less than q .

$y = g^x \bmod p$.

H = Message-digest algorithm (usually SHA-1).

The first three variables, p , q and g are public in nature, and can be sent across an insecure network freely. The private key is x , whereas the corresponding public key is y .

Let us assume that the sender wants to sign a message m and send the signed message to the receiver. Then, the following steps take place.

(a) The sender generates a random number k , which is less than q .

(b) The sender now calculates:

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

- $s = (k^{-1} (H(m) + xr)) \bmod q$

The values r and s are the signatures of the sender. The sender sends these values to the receiver.

To verify the signature, the receiver calculates:

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

$$u1 = (H(m) * w) \bmod q$$

$$u2 = (rw) \bmod q$$

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

If $v = r$, the signature is said to be verified. Otherwise, it is rejected.

■ 4.8 KNAPSACK ALGORITHM ■

Actually, Ralph Merkle and Martin Hellman developed the first algorithm for public-key encryption, called the **Knapsack algorithm**. It is based on the **Knapsack problem**. This is actually a simple problem. Given a pile of items, each with different weights, is it possible to put some of them in a bag (i.e. knapsack) in such a way that the knapsack has a certain weight?

That is, if $M1, M2, \dots, Mn$ are the given values and S is the sum, find out bi so that:

$$S = b1M1 + b2M2 + \dots + bnMn$$

Each bi can be 0 or 1. A 1 indicates that the item is in the knapsack, and a 0 indicates that it is not.

A block of plain text equal in length to the number of items in the pile would select the items in the knapsack. The cipher text is the resulting sum. For example, if the knapsack is 1, 7, 8, 12, 14, 20 then the plain text and the resulting cipher text is as shown in Fig. 4.62.

Plain text	0 1 1 0 1 1	1 1 1 0 0 0	0 1 0 1 1 0
Knapsack	1 7 8 12 14 20	1 7 8 12 14 20	1 7 8 12 14 20
Cipher text	$7 + 8 + 14 + 20 = 49$	$1 + 7 + 8 = 16$	$7 + 12 + 14 = 33$

Fig. 4.62 Knapsack example

■ 4.9 ELGAMAL DIGITAL SIGNATURE ■

We have discussed the ElGamal cryptosystem earlier. The **ElGamal digital-signature** scheme uses the same keys, but a different algorithm. The algorithm creates two digital signatures. In the verification step, these two signatures are tallied. The key-generation process here is the same as what we had discussed earlier and hence we would not repeat the discussion. The public key remains $(E1, E2, P)$ and the private key continues to be D .

4.9.1 Signature

The signature process works as follows:

1. The sender selects a random number R .
2. The sender computes the first signature $S1$ using the equation $S1 = E1^R \bmod P$.
3. The sender computes the second signature $S2$ using the equation $S2 = (M - D \times S1) \times R^{-1} \bmod (P - 1)$, where M is the original message that needs to be signed.
4. The sender sends $M, S1$, and $S2$ to the receiver.

For example, let $E1 = 10, E2 = 4, P = 19, M = 14, D = 16$, and $R = 5$.

Then we have:

$$S1 = E1^R \bmod P = 10^5 \bmod 19 = 3$$

$$S2 = (M - D \times S1) \times R^{-1} \bmod (P - 1) = (14 - 16 \times 3) \times 5^{-1} \bmod 18 = 4$$

Hence, the signature is $(S1, S2)$ i.e. $(3, 4)$. This is sent to the receiver.

4.9.2 Verification

The verification process works as follows:

1. The receiver performs the first part of verification called $V1$ using the equation $V1 = E1^M \bmod P$.
2. The receiver performs the second part of verification called as $V2$ using the equation $V2 = E2^{S1} \times S1^{S2} \bmod P$.

In our example:

$$V1 = E1^M \bmod P = 10^{14} \bmod 19 = 16$$

$$V2 = E2^{S1} \times S1^{S2} \bmod P = 4^3 \times 3^4 \bmod 19 = 5184 \bmod 19 = 16$$

Since $V1 = V2$, the signature is considered valid.

■ 4.10 ATTACKS ON DIGITAL SIGNATURES ■

In general, three types of attacks are attempted against digital signatures, as outlined below:

1. Chosen-message Attack

In the **chosen-message attack**, the attacker tricks a genuine user into digitally signing messages that the user does not normally intend to sign. As a result, the attacker obtains a pair of the original message

that was signed and the signature. Using this, the attacker tries to create a new message that she wants the genuine user to sign and uses the previous signature.

2. Known-message Attack

In the **known-message attack**, the attacker obtains the previous few messages and the corresponding digital signatures from a genuine user. Like the known-plain text attack in the case of encryption, the attacker now tries to create a new message and forge the digital signature of the genuine user onto it.

3. Key-only Attack

In the **key-only attack**, the assumption is that some information was made public by a genuine user. This attacker now tries to misuse this public information. This is similar to the cipher text-only attack in encryption. Here, the attacker tries to create the signature of the genuine user.

■ 4.11 PROBLEMS WITH THE PUBLIC-KEY EXCHANGE ■

If we have read all the discussion so far very carefully, we will observe that throughout the discussion, we assume that the sender (say Alice) knows the value of the public key of the receiver (say Bob), and the receiver (Bob) knows the value of the public key of the sender (Alice). How can this be achieved?

Well, Alice can simply send her public key to Bob, and request for Bob's public key, in turn. What is the problem? The *man-in-the-middle attack* (discussed earlier with reference to Diffie–Hellman key—exchange algorithm) can be launched by Tom—the attacker, here, as shown in Fig. 4.63.

As the figure shows, the public-key values for the sender (Alice), attacker (Tom) and receiver (Bob) are respectively 20, 17 and 13.

1. When Alice wants to send a message securely to Bob, she sends Bob her public key (20) and asks Bob for Bob's public key.
2. Tom—the attacker—intercepts Alice's message. He changes the public-key value in Alice's original message from 20 to his own (17) and forwards this message to Bob.
3. Bob sends back his public key (13) in response to Alice's message.
4. Tom intercepts Bob's message, changes the public key value to 17, and forwards it to Alice.
5. Alice thinks that Bob's public key is 17. Therefore, she encrypts the confidential message to be sent to Bob with 17 and sends to Bob.
6. Tom intercepts this message, uses his private key to decrypt the message, processes it (that is does whatever he wants to do with the message—it could be mere reading of the message or alterations), re-encrypts the message with Bob's public key (13) and forwards it to Bob.
7. Bob decrypts the message coming from Tom with his private key, and depending on the message, forms a reply. He encrypts the reply with what he believes to be Alice's public key (17). He then sends the reply back to Alice.
8. Tom intercepts Bob's reply, uses his private key to decrypt the message, performs whatever actions he wants to on the message, re-encrypts it with the real public key of Alice (20), and sends it to Alice. Alice can decrypt it with her private key.

This process can go on and on, without either Alice or Bob realizing that an attacker is playing havoc! This is especially truer if Tom can perform the encryption and decryption operations really fast!

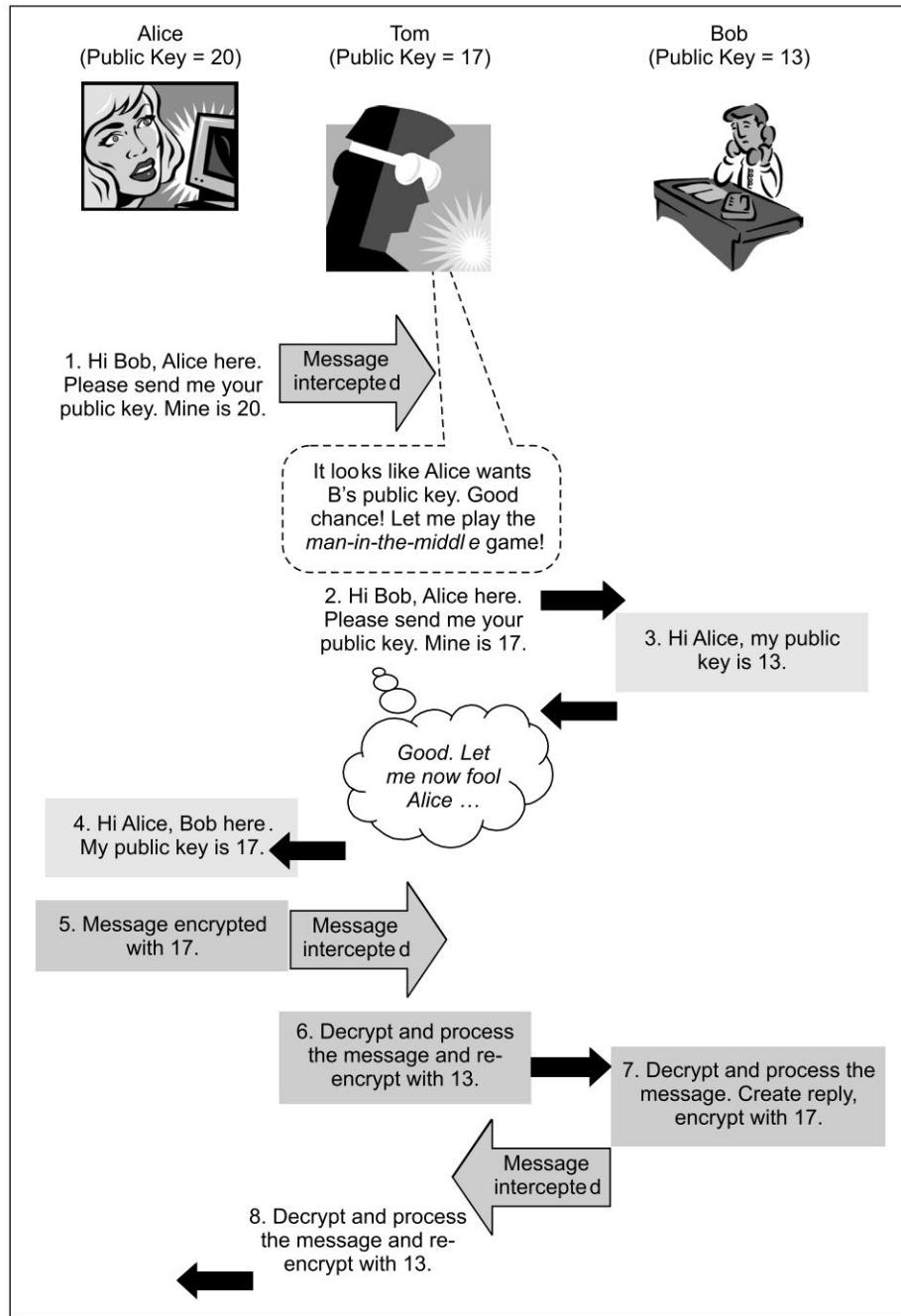


Fig. 4.63 Man-in-the-middle attack

This also means that although the asymmetric-key cryptography technique resolves the problem of key exchange, and therefore, the security issues in all the communication thereafter, it does not resolve the

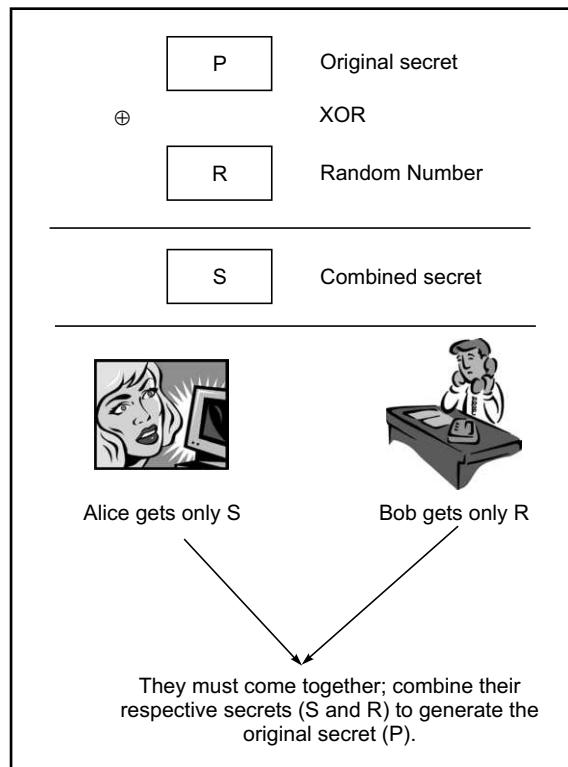
basic problem of making available the public keys of the correspondents to each other. How can the sender and the receiver make their public keys available to each other, then? How can they prevent the *man-in-the-middle attack*?

We shall try and discover the answer to this question next.

■ CASE STUDY 1: VIRTUAL ELECTIONS ■

Points for Classroom Discussions

1. Is it technically possible to have elections on the Internet? How? What sort of infrastructure would be needed for this?
2. What would be the main concerns in such a virtual election?
3. What would be the use of digital signatures and encryption in virtual elections?



Another situation where cryptography is useful is virtual elections. Computerized voting would become quite common in the next few decades. As such, it is important that the protocol for virtual elections should protect individual privacy and should also disallow cheating. Consider the following protocol in order that voters can send their votes electronically to the Election Authority (EA).

1. Each voter casts the vote and encrypts it with the public key of the EA.

2. Each voter sends the encrypted vote to the EA.
3. The EA decrypts all the votes to retrieve the original vote, tabulates all the votes and announces the result of the election.

Is this protocol secure and does it provide comfort both to the voters as well as to the EA? Not at all! There are following problems in this scheme:

1. The EA does not know whether the authorized voters have voted or it has received fake (bogus) votes.
2. Secondly, there is no mechanism to prevent duplicate voting.

What is the advantage of this protocol? Clearly, no one would be able to change another voter's vote, because it is first encrypted with the EA's public key and is then sent to the EA. However, if we observe this scheme carefully, an attacker need not change someone's vote at all. The attacker can simply send duplicate votes!

How can we improve upon this protocol to make it more robust? Let us rewrite it, as follows:

1. Each voter casts the vote and signs it with her private key.
2. Each voter then encrypts the signed vote with the public key of the EA.
3. Each voter sends the vote to the EA.
4. The EA decrypts the voter with its private key and verifies the signature of the voter with the help of the voter's public key.
5. The EA then tabulates all the votes and announces the result of the election.

This protocol would now ensure that duplicate voting is disallowed. Because the voter has signed the vote (with her private key) in Step 1, this can be checked. Similarly, no one can change another voter's vote. This is because a vote is digitally signed and any changes to it will be detected and exposed in the signature verification process.

Although this protocol is a lot better, the trouble with this scheme is that the EA would come to know who voted for whom, leading to privacy concerns. We shall leave it to the reader to figure out how this problem can be solved.

■ CASE STUDY 2: CONTRACT SIGNING ■

Points for Classroom Discussions

1. *When can a contract in real-life considered to be complete?*
2. *Is it sufficient if only one of the parties signs a contract?*
3. *What mechanism can be used in cryptography to sign electronic contracts?*
4. *How can disputes be settled if a party later refuses having signed a contract?*

There is a builder, Bob, who has constructed a building for residential purposes. This building contains 20 flats (apartments). Bob wants to sell all of them to individual customers. Alice is one such customer, who is interested in buying a flat. She approaches Bob over telephone and expresses her desire to buy the flat. After a couple of such discussions, Alice decides to go ahead with her purchase. However, Bob

has only one requirement. This requirement deals with the way the contract or the agreement between Bob and Alice is signed. Since Bob is Internet-savvy, he wants that the contract between him and Alice be digitally signed over the Internet and that Alice deposit the money into his account by using Internet banking.

In this discussion, we shall restrict our scope to only the first aspect, i.e. digitally signing contracts. How would Bob ensure that the contract signing process is complete in all respects? How would Alice be sure that she is not being cheated? How would a dispute be settled, if it arises?

In order to ensure that the contract signing happens smoothly, Bob contacts a respected third party, Trent. Alice also speaks with Trent over phone and is assured that Trent is a responsible third party, in which she could trust. With these ideas in place, let us write down the steps that would ensure that the digital contract signing is complete and comprehensive in all respects.

1. Alice digitally signs a copy of the contract and sends it to Trent over the Internet.
2. Bob digitally signs a copy of the contract and sends it to Trent over the Internet.
3. Trent sends a message to both Alice and Bob, informing them that he has received the digitally signed contracts from both.
4. At this stage, Alice digitally signs two more copies of the contract and sends both of them to Bob.
5. Bob now digitally signs both the contract copies received from Alice. He keeps one of the copies for his records and sends the other one to Alice.
6. Alice and Bob both inform Trent that they have a copy of the contract, which is digitally signed by both of them.
7. Trent now destroys the original contract copies received from Alice and Bob in Steps 1 and 2.

Is this solution foolproof? Let us examine it.

Can Alice deny at a later date that she never signed the contract? She cannot, because Bob has a copy of the contract, which was signed by him as well as by Alice. It is also the case the other way round. Bob cannot refute having signed the contract, because Alice has a copy of the contract signed by both.

Thus, the solution is indeed comprehensive. This solution also involves the use of a third party (also called as an *arbitrator*), Trent.



Summary

- Asymmetric-key cryptography aims at resolving the key-exchange problem of symmetric-key cryptography.
- RSA is a very popular asymmetric-key cryptography.
- Each communicating party needs a key pair in asymmetric-key cryptography.
- Public key is shared with everybody, private key must be kept secret by the individual.
- Prime numbers are very important in asymmetric-key cryptography.
- Digital envelopes combine the best features of symmetric-and asymmetric-key cryptography.
- A message digest (also called hash) identifies a message uniquely.

- Message digests have three important properties: (1) Two different messages must have different message digests. (b) Given a message digest, we must always get the same message digest if the algorithm is not changed. (c) Given a message digest, we must not be able to obtain the original message.
- MD5 and SHA-1 are message-digest algorithms.
- MD5 is considered vulnerable to attacks now.
- SHA-512 is the latest algorithm in the SHA family.
- HMAC is a message-digest algorithm that involves encryption.
- HMAC suffers from practical issues.
- DSA and RSA algorithms can be used for digital signatures.
- RSA is more popular than DSA.
- RSA can be used both for encryption and digital signatures.



Key Terms and Concepts

- | | |
|---|--|
| <ul style="list-style-type: none"> ● Collision ● Digital Signature Algorithm (DSA) ● Hash ● Key wrapping ● Message Authentication Code (MAC) ● Pseudocollision ● SHA | <ul style="list-style-type: none"> ● Digital envelope ● Digital Signature Standard (DSS) ● HMAC ● MD5 ● Message digest ● RSA algorithm |
|---|--|



PRACTICE SET

■ Multiple-Choice Questions

1. In asymmetric key cryptography, _____ keys are required per communicating party.
 - (a) 2
 - (b) 3
 - (c) 4
 - (d) 5
 2. The private key _____.
 - (a) must be distributed
 - (b) must be shared with everyone
- (c) must remain secret with an individual
 - (d) none of the above
 3. If A and B want to communicate securely with each other, B must not know _____.
 - (a) A 's private key
 - (b) A 's public key
 - (c) B 's private key
 - (d) B 's private key
 4. _____ are very crucial for the success of asymmetric-key cryptography.

- (a) Integers
 (b) Prime numbers
 (c) Negative numbers
 (d) Fractions
5. Symmetric-key cryptography is _____ than asymmetric key cryptography.
 (a) always slower
 (b) of the same speed
 (c) faster
 (d) usually slower
6. While creating a digital envelope, we encrypt the _____ with the _____.
 (a) sender's private key, one-time session key
 (b) receiver's public key, one-time session key
 (c) one-time session key, sender's public key
 (d) one-time session key, receiver's public key
7. If the sender encrypts the message with his/her private key, it achieves the purpose of _____.
 (a) confidentiality
 (b) confidentiality and authentication
 (c) confidentiality but not authentication
 (d) authentication
8. A _____ is used to verify the integrity of a message.
 (a) message digest
 (b) decryption algorithm
 (c) digital envelope
 (d) none of the above
9. When two different message digests have the same value, it is called _____.
 (a) attack
- (b) collision
 (c) hash
 (d) none of the above
10. _____ is a message-digest algorithm.
 (a) DES
 (b) IDEA
 (c) MD5
 (d) RSA
11. MAC is _____ a message digest.
 (a) same as
 (b) different from
 (c) subset of
 (d) none of the above
12. RSA _____ be used for digital signatures.
 (a) must not
 (b) cannot
 (c) can
 (d) should not
13. The strongest message-digest algorithm is considered as _____.
 (a) SHA-1
 (b) SHA-256
 (c) SHA-128
 (d) SHA-512
14. To verify a digital signature, we need the _____.
 (a) sender's private key
 (b) sender's public key
 (c) receiver's private key
 (d) receiver's public key
15. To decrypt a message encrypted using RSA, we need the
 (a) sender's private key
 (b) sender's public key
 (c) receiver's private key
 (d) receiver's public key

Exercises

1. Discuss the history of asymmetric-key cryptography in brief.
2. If A wants to send a message securely to B , what would be the typical steps involved?
3. What is the real crux of RSA?
4. Describe the advantages and disadvantages of symmetric and asymmetric-key cryptography.
5. What is key wrapping? How is it useful?
6. What are the key requirements of message digests?
7. Why is SHA more secure than MD5?

8. What is the difference between MAC and message digest?
9. What is the important aspect that establishes trust in digital signatures?
10. What is the problem with exchanging of public keys?
11. Write a short note on ElGamal digital signatures.
12. Discuss the possible attacks on RSA digital signatures.
13. Discuss the security of RSA.
14. Explain El-Gamal cryptography.
15. What is SHA-3?

■ Design/Programming Exercises

1. Write a Java program that generates the message digest (not in the true technical sense) of a number, using the algorithm shown in Fig. 4.18.
2. Write a C program that calculates the message digest of a text using the MD5 algorithm.
3. Write a Visual Basic program to calculate the 32-bit CRC of a string.
4. Write a Visual Basic program to calculate the 16-bit CRC of a string.
5. Write a Java program to implement the RSA algorithm.
6. As we know, cryptographic operations can be very slow, especially for large numbers. One of the operations we need to perform is to first raise a number to a certain exponent, and then find the modulus of the result. This can be very expensive, and in fact, impossible, for very large numbers. One solution to this problem is to use the result of the previous step in a given step, and find out the final answer. Although not the most optimized solution, this is one way to solve the problem.

The algorithm for this is:

To find $a^b \bmod n$, do the following:

Start

C = 1

For i = 1 to b

Calculate C = (C × a) mod n

Next I

End

For instance, consider the following example:

To find $7^5 \bmod 119$, we can have:

$$(1 \times 7) \bmod 119 = 7$$

$$(7 \times 7) \bmod 119 = 49$$

$$(49 \times 7) \bmod 119 = 105$$

$$(105 \times 7) \bmod 119 = 21$$

$$(21 \times 7) \bmod 119 = 28$$

As we can see, $7^5 \bmod 119 = 28$.

Using this technique, find $8^9 \bmod 117$.

7. Write a C program to implement the above logic.
8. Write a Java program to implement the same logic.
9. Consider a plain-text alphabet G . Using the RSA algorithm and the values as $E = 3$, $D = 11$ and $N = 15$, find out what this plain-text alphabet encrypts to, and verify that upon decryption, it transforms back to G .

10. Given two prime numbers $P = 17$ and $Q = 29$, find out N, E and D in an RSA encryption process.
11. In RSA, given $N = 187$ and the encryption key (E) as 17, find out the corresponding private key (D).
12. Can we use a conventional lossless compression mechanism as message digest? Why?
13. Can we use checksum as a message-digest mechanism? Why?
14. We have a message consisting of 20,000 characters. After computing its message digest using SHA-1, we decide to change the last 19 characters in the original message. How many bits in the digest will change, and why?
15. Should we have variable-length message digests, instead of fixed-length ones? Why?



PUBLIC KEY INFRASTRUCTURE (PKI)

■ 5.1 INTRODUCTION ■

Public Key Infrastructure (PKI) technology has attracted significant attention, and has become the central focus of modern security mechanisms on the Internet. PKI is not just a buzzword, or a trivial idea. Substantial investments, commitments and efforts are needed to establish, maintain and grow a PKI solution. PKI is the road ahead for almost all cryptographic systems.

This chapter discusses PKI in great detail. PKI is closely related to the ideas of asymmetric-key cryptography, mainly including message digests, digital signatures and encryption services. We have already discussed them. The chief requirement to enable all these services is the technology of digital certificates. Digital certificates are termed *passports on the Web*. We examine them in substantial detail.

Digital certificates bring with them a plethora of issues, concerns and points of debate. This chapter takes the reader through all of them, one by one. We examine the role of Certification Authorities (CA), Registration Authorities (RA), how one CA is related with another, and other concepts such as root CA, self-signed certificates and cross-certification.

Validating digital certificates is both crucial and complex. Special protocols such as CRL, OCSP and SCVP are designed to handle this task. We examine all of them and compare them.

Some other expectations from a CA are maintaining and archiving user keys, and making them available when needed. Also, ideas such as roaming certificates are also gaining momentum. The chapter deals with all these issues.

PKIX and PKCS are the two popular standards for digital certificates and PKI. The chapter discusses them, with detailed explanations of the key areas. XML security has gained prominence these days, because of its features, as well as because XML is a worldwide standard for information exchange now. The chapter discusses the aspects of XML security as well.

■ 5.2 DIGITAL CERTIFICATES ■

5.2.1 Introduction

We have discussed the problem of *key agreement* or *key exchange* in great detail. We have also seen how even the algorithm such as *Diffie–Hellman Key Exchange* designed specifically to tackle this problem also has its own pitfalls. The asymmetric-key cryptography can be a very good solution. But it also has one unresolved issue, which is *how* the parties/respondents (i.e. the sender and the receiver of a message) exchange their public keys with each other. Obviously, they cannot exchange them openly—this can very easily lead to a *man-in-the-middle attack* on the public key itself!

This problem of key exchange or key agreement is, therefore, quite severe, and in fact, is one of the most difficult challenges to tackle in designing any computer-based cryptographic solution. After a lot of thought, this problem was resolved with a revolutionary idea of using **digital certificates**. We shall study this in great detail.

Conceptually, we can compare digital certificates to the documents such as our passports or driving licenses. A passport or a driving license helps in establishing our identity. For instance, my passport proves beyond doubt a variety of aspects, the most important ones being

- My full name
- My nationality
- My date and place of birth
- My photograph and signature

Likewise, my digital certificate would also prove something very critical, as we shall study.

5.2.2 The Concept of Digital Certificates

A digital certificate is simply a small computer file. For example, my digital certificate would actually be a computer file with a file name such as `atul.cer` (where `.cer` signifies the first three characters of the word *certificate*). Of course, this is just an example: in actual practice, the file extensions can be different.) Just as my passport signifies the association between me and my other characteristics such as my full name, nationality, date and place of birth, photograph and signature, my digital certificate simply signifies the association between my public key and me. This concept of digital certificates is shown in Fig. 5.1. Note that this is merely a conceptual view, and does not depict the actual contents of a digital certificate.

We have not specified who is officially approving the association between a user and the user's digital certificate. Obviously, it has to be some authority in which all the concerned parties have a great amount of trust and belief. Imagine a situation where our passports are not issued by a government office, but by an ordinary shopkeeper. Would we trust the passports? Similarly, digital certificates must be issued by some trusted entity. Otherwise we will not trust anybody's digital certificate.

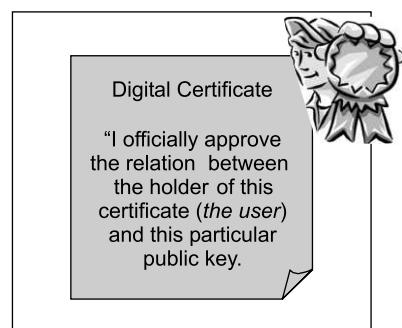


Fig. 5.1 Conceptual view of a digital certificate

As we have noted, a digital certificate establishes the relation between a user and his/her public key. Therefore, a digital certificate must contain the user name and the user's public key. This will prove that a particular public key belongs to a particular user. Apart from this, what does a digital certificate contain? A simplified view of a sample digital certificate is shown in Fig. 5.2.

We will notice a few interesting things here. First of all, my name is shown as **subject name**. In fact, any user's name in a digital certificate is always referred to as *subject name* (this is because a digital certificate can be issued to an individual, a group or an organization). Also, there is another interesting piece of information called **serial number**. We shall see what it means in due course of time. The certificate also contains other pieces of information, such as the validity date range for the certificate, and who issued it (**issuer name**). Let us try to understand the meanings of these pieces of information by comparing them with the corresponding entries in my passport. This is shown in Fig. 5.3.

As the figure shows, the digital certificate is actually quite similar to a passport. Just as every passport has a unique passport number, every digital certificate has a unique serial number. As we know, no two passports issued by the same issuer (i.e. government) can have the same passport number. Similarly, no two digital certificates issued by the same issuer can have the same serial number. Who can issue these digital certificates? We shall soon answer this question.

5.2.3 Certification Authority (CA)

A **Certification Authority (CA)** is a trusted agency that can issue digital certificates. Who can be a CA? The authority of acting as a CA has to be with someone who everybody trusts. Consequently, the governments in various countries decide who can and who cannot be a CA. (It is another matter that not everybody trusts the government in the first place!) Usually, a CA is a reputed organization, such as a post office, financial institution, software company, etc. Two of the world's most famous CAs are VeriSign and Entrust. Safescrypt Limited, a subsidiary of Satyam Infoway Limited, became the first Indian CA in February 2002.

Thus, a CA has the authority to issue digital certificates to individuals and organizations, who want to use those certificates in asymmetric-key cryptographic applications.

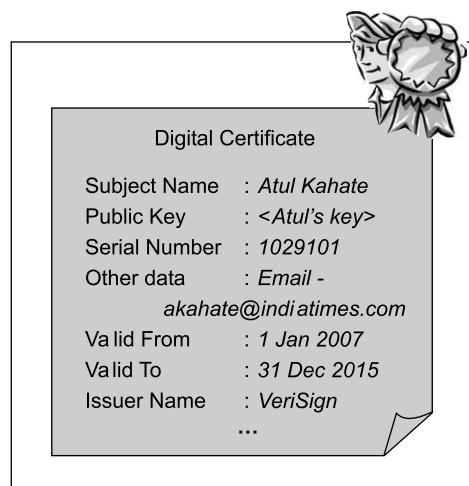


Fig. 5.2 Example of a digital certificate

Passport entry	Corresponding digital-certificate entry
Full name	Subject name
Passport number	Serial number
Valid from	Same
Valid to	Same
Issued by	Issuer name
Photograph and signature	Public key

Fig. 5.3 Similarities between a passport and a digital certificate

5.2.4 Technical Details of a Digital Certificate

Let us now take a technical look at the contents of a digital certificate. We have already examined this from a conceptual point of view. Now, we shall take a look at a digital certificate from a technical perspective. Readers who are not very keen to understand this can skip this portion without any loss of continuity.

A standard called **X.509** defines the structure of a digital certificate. The International Telecommunication Union (ITU) came up with this standard in 1988. At that time, it was a part of another standard called **X.500**. Since then, X.509 was revised twice (in 1993 and again in 1995). The current version of the standard is Version 3, called **X.509V3**. The Internet Engineering Task Force (IETF) published the RFC2459 for the X.509 standard in 1999. Figure 5.4 shows the structure of a X.509V3 digital certificate.

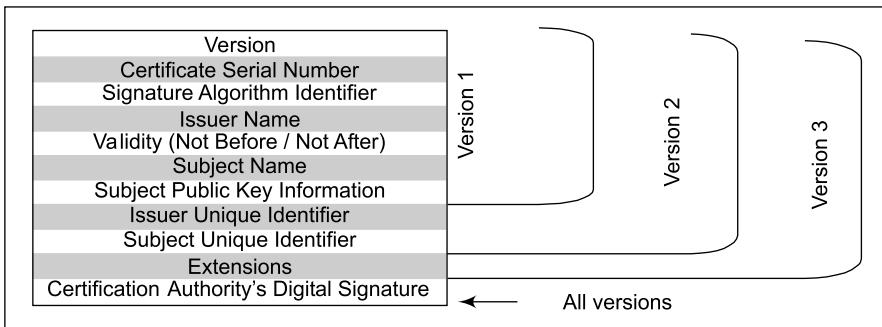


Fig. 5.4 Contents of a digital certificate

The figure shows the various fields of a digital certificate according to the X.509 standard. Moreover, it also specifies which version of the standard contains which fields. As we can see, version 1 of the X.509 standard contained seven basic fields, version 2 added two more fields, and version 3 added one more field. These additional fields are called **extensions** or **extended attributes** of versions 2 and 3, respectively. Of course, we have one additional common field in the end for all the versions. Let us now examine these fields, as shown in figures 5.5 (a), 5.5 (b), and 5.5 (c).

Version 2 introduced two new fields to deal with the possibility that the *Issuer Name* (i.e. the CA's name) and the *Subject Name* (i.e. the certificate holder's name) might be unintentionally duplicated over time. However, the digital certificate standard (RFC2459) specifies that the same *Issuer Name* or the same *Subject Name* should never be used more than once in the first place. Therefore, although these fields are added by version 2, their usage is discouraged and both these fields are made optional. However, if used, these fields help distinguish between two issuers or subjects, if they are duplicated for some reason.

Version 3 of the X.509 standard has added many extensions to the structure of a digital certificate. These extensions are listed in Fig. 5.5 (c).

Field	Description
Version	Identifies a particular version of the X.509 protocol, which is used for this digital certificate. Currently, this field can contain 1, 2 or 3.
Certificate Serial Number	Contains a unique integer number, which is generated by the CA.
Signature Algorithm Identifier	Identifies the algorithm used by the CA to sign this certificate. (We shall examine this later).
Issuer Name	Identifies the Distinguished Name (DN) of the CA that created and signed this certificate.
Validity (Not Before/Not After)	Contains two date-time values (<i>Not Before</i> and <i>Not After</i>), which specify the time frame within which the certificate should be considered valid. These values generally specify the date and time up to seconds or milliseconds.
Subject Name	Identifies the <i>Distinguished Name (DN)</i> of the end entity (i.e. the user or the organization) to whom this certificate refers. This field must contain an entry unless an alternative name is defined in Version 3 extensions.
Subject Public Key Information	Contains the subject's public key and algorithms related to that key. This field can never be blank.

Fig. 5.5 (a) Description of the various fields in a X.509 digital certificate—*Version 1*

Field	Description
Issuer Unique Identifier	Helps identify a CA uniquely if two or more CAs have used the same <i>IssuerName</i> over time.
Subject Unique Identifier	Helps identify a subject uniquely if two or more subjects have used the same <i>Subject Name</i> over time.

Fig. 5.5 (b) Description of the various fields in a X.509 digital certificate—*Version 2*

5.2.5 Digital-Certificate Creation

1. Parties Involved

Having understood the conceptual and technical aspects of a digital certificate, let us now think the typical process that requires to be carried out in order to create a digital certificate. Who are the parties involved, and what do they need to do? We already know two of the three parties involved in this process, namely the subject (end user) and the issuer (CA). A third party is also (optionally) involved in the certificate creation and management.

Since a CA can be overloaded with a variety of tasks such as issuing new certificates, maintaining the old ones, revoking the ones that have become invalid for whatever reason, etc., the CA can delegate some of its tasks to this third party, called a **Registration Authority (RA)**. From an end user's perspective, there is little difference between the CA and the RA. Technically, the RA is an intermediate entity between the end users and the CA, which assists the CA in its day-to-day activities. This is shown in Fig. 5.6.

Field	Description
Authority Key Identifier	A CA may have multiple private-public key pairs. This field defines which of these key pairs is used to sign (and therefore, which corresponding key should be used to verify) this certificate.
Subject Key Identifier	A subject may have multiple private-public key pairs. The reason for this is explained later. This field defines which of those key pairs is used to sign (and therefore, which corresponding key should be used to verify).
Key Usage	Defines the scope of operations of the public key of this particular certificate. For example, we can specify whether this particular public key can be used for all cryptographic operations, or only for encryption, or only for Diffie-Hellman key exchange, or only for performing digital signatures, or some combinations of these, etc.
Extended Key Usage	Can be used in addition to, or in the place of the <i>Key Usage</i> field. Specifies which protocols this certificate can interoperate with (we shall study these protocols later). Examples of these protocols are Transport Layer Security (TLS), client authentication, server authentication, time stamping, etc.
Private Key Usage Period	Allows defining different usage period limits for the private and the public keys corresponding to this certificate. If this field is empty, the validity of the private key corresponding to this certificate is the same as that of the public key.
Certificate Policies	Defines the policies and optional qualifier information that the CA associates with a given certificate. We need not discuss this.
Policy Mappings	Used only when the subject of a given certificate is also a CA. That is, when a CA issues a certificate to another CA, the certifier CA specifies which of its policies the CA being certified must follow.
Subject Alternative Name	Optionally defines one or more alternative names for the subject of a given certificate. However, if the <i>Subject Name</i> field in the main certificate format is empty, this field must contain some value.
Issuer Alternative Name	Optionally defines one or more alternative names for the issuer of a given certificate.
Subject Directory Attributes	Can be used to provide additional information about the subject, such as the subject's phone/fax numbers, email ids, etc.
Basic Constraints	Indicates whether the subject in this certificate may act as a CA. This field also specifies if the subject can, in turn, allow other subjects to become CAs. For instance, if CA X is issuing this particular certificate to another CA Y, X can not only specify if Y can act as CA and issue certificates to other subjects, but also whether Y can grant the authority of becoming CA to other subjects, in turn.
Name Constraints	Specifies the name space. We need not discuss this.
Policy Constraints	Used only for CA certificates. We need not discuss this.

Fig. 5.5 (c) Description of the various fields in a X.509 digital certificate—Version 3

The RA commonly provides the following services:

- Accepting and verifying registration information about new users
- Generating keys on behalf of the end users

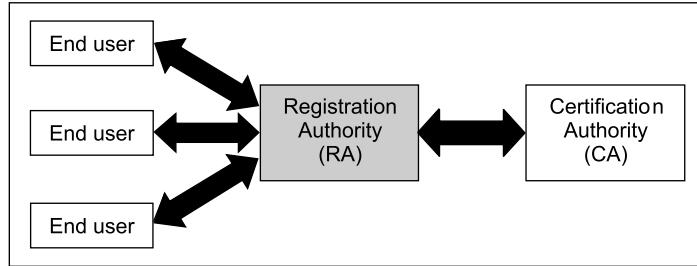


Fig. 5.6 Registration Authority (RA)

- Accepting and authorizing requests for key backups and recovery
- Accepting and authorizing the requests for certificate revocation

Another very critical impact of having an RA in between the end user and the CA is that the CA then becomes an isolated entity, which makes it less susceptible to security attacks. Since the end users communicate only with the RA, the communication between the RA and the CA can be highly protected, making it difficult to break into this portion of the connection.

However, it must be noted that the RA is mainly set up for facilitating the interaction between the end users and the CA. The RA cannot issue digital certificates. The CA must handle this. Additionally, after a certificate is issued, the CA is responsible for all the certificate management aspects, such as tracking its status, issuing revocation notices if the certificate needs to be invalidated for some reason, etc.

2. Certificate Creation Steps

The creation of a digital certificate consists of several steps. These steps are outlined in Fig. 5.7.

Let us now examine these steps, required for the creation of a digital certificate.

Step 1: Key Generation The action begins with the subject (i.e. the user/organization) who wants to obtain a certificate. There are two different approaches for this purpose:

- (a) The subject can create a private key and public key pair using some software. This software is usually a part of the Web browser or Web server. Alternatively, special software programs can be used for this. The subject must keep the private key thus generated, secret. The subject then sends the public key along with other information and evidences about herself to the RA. This is shown in Fig. 5.8.
- (b) Alternatively, the RA can generate a key pair on the subject's (user's) behalf. This can happen in cases where either the user is not aware of the technicalities involved in the generation of a key pair, or if a particular requirement demands that all the keys must be centrally generated and distributed by the RA for the ease of enforcing security policies and key management. Of course, the major disadvantages of this approach are the possibility of the RA knowing the private key of

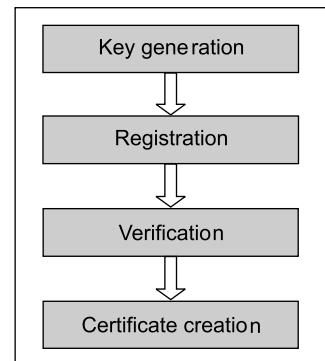


Fig. 5.7 Digital-certificate creation steps

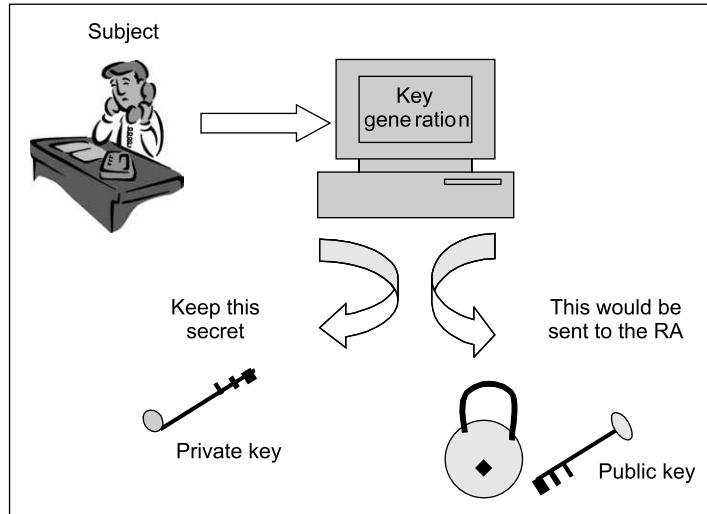


Fig. 5.8 Subject generating its own key pair

the user, as well as the scope for this key to be exposed to others while in transit after it is generated and sent to the appropriate user. This is shown in Fig. 5.9.

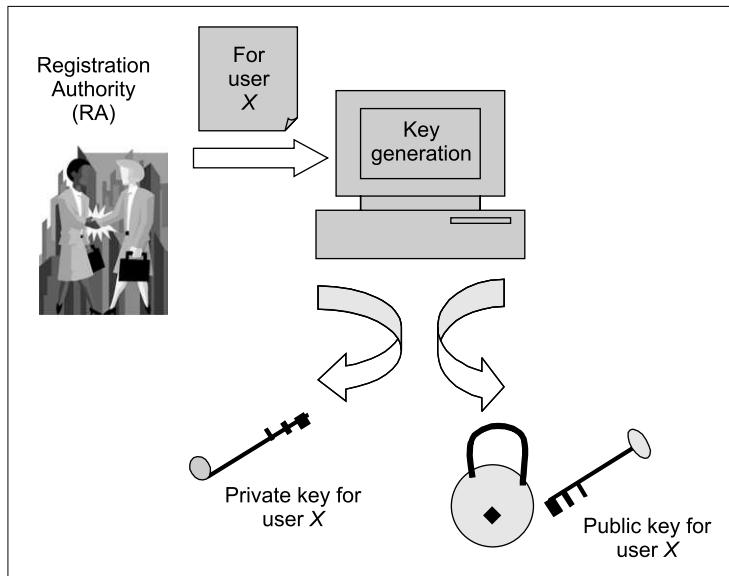


Fig. 5.9 RA generating a key pair on behalf of the subject

Step 2: Registration This step is required only if the user generates the key pair in the first step. If the RA generates the key pair on the user's behalf, this step will also be a part of the first step itself.

Assuming that the user has generated the key pair, the user now sends the public key and the associated registration information (e.g. subject name, as it is desired to appear in the digital certificate) and all the

evidence about herself to the RA. For this, the software provides a wizard in which the user enters data and when all data is correct, submits it. This data then travels over the network/Internet to the RA. The format for the certificate requests has been standardized, and is called **Certificate Signing Request (CSR)**. This is one of the **Public Key Cryptography Standards (PKCS)**, as we shall study later. CSR is also called **PKCS#10**.

The evidence, however, may not be in the form of computer data, and usually consists of paper-based documents (such as a copy of the passport or business documents or income/tax statements, etc). This is shown in Fig. 5.10.

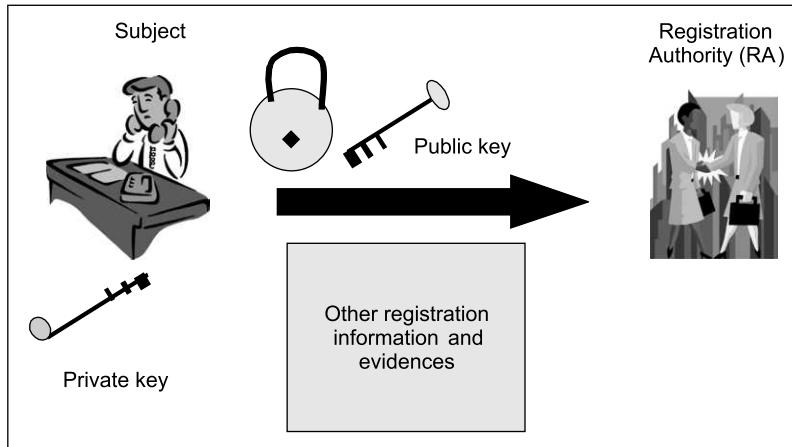


Fig. 5.10 Subject sends public key and evidences to the RA

Note that the user must not send the private key to the RA—the user must retain it securely. In fact, as far as possible, the private key must not leave the user’s computer at all.

The actual portion of a sample certificate request page in real life is shown in Fig. 5.11.

After this, the user usually gets a request identifier for tracking the progress of the certificate request, as shown in Fig. 5.12.

Step 3: Verification After the registration process is complete, the RA has to verify the user’s credentials. This verification is in two respects, as follows.

- Firstly, the RA needs to verify the user’s credentials such as the evidences provided are correct, and that they are acceptable. If the user were actually an organization then the RA would perhaps like to check the business records, historical documents and credibility proofs. If it is an individual user then simpler checks are in call, such as verifying the postal address, email id, phone number, passport or driving-license details can be sufficient.
- The second check is to ensure that the user who is requesting for the certificate does indeed possess the private key corresponding to the public key that is sent as a part of the certificate request to the RA. This is very important, because there must be a record that the user possesses the private key corresponding to the given public key. Otherwise, this can create legal problems (what if a user claims that he/she never possessed the private key, when a document signed with him/her private key causes some legal problems?) This check is called checking the **Proof Of Possession**

The screenshot shows a web browser window titled "Netscape® Certificate Management System". The main content area is titled "Certificate Manager". On the left, a sidebar lists various management options: Browser, Manual, Server, SSL Server, Registration Manager, Certificate Manager, OCSP Responder, Other, Object Signing (Browser), and Object Signing (PKCS10). The "Certificate Manager" option is selected.

Manual User Enrollment
Use this form to submit a request for a personal certificate. After you click the Submit button, your request will be submitted to an issuing agent for approval. When an issuing agent has approved your request you will receive the certificate in email, along with instructions for installing it.

Important: Be sure to request your certificate on the same computer on which you plan to use the certificate.

User's Identity
Enter values for the fields you want to have in your certificate. Your site may require you to fill in certain fields.
(* = required field)

* Full name:	Atul Kahate
Login name:	ekahate
Email address:	ekahate@indiatimes.com
Organization unit:	Personal
Organization:	Personal
Country:	IN

Challenge Phrase Password (optional)
Enter a challenge phrase password which can be used for certificate revocation.

Fig. 5.11 Digital-certificate request

The screenshot shows a web browser window titled "Netscape® Certificate Management System". The main content area is titled "Certificate Manager". The sidebar on the left shows the "Request Successfully Submitted" section, indicating the success of the previous submission.

Request Successfully Submitted
Congratulations, your request has been successfully submitted to the Certificate Manager. Your request will be processed when an authorized agent verifies and validates the information in your request.

Your request ID is **20**.

Your can check on the status of your request with an authorized agent or local administrator by referring to this request ID.

Fig. 5.12 Acceptance receipt of a certificate request by the CA

(POP) of the private key. How can the RA perform this check? There are many approaches to this, the chief ones being as follows.

- (i) The RA can demand that the user must digitally sign his/her Certificate Signing Request (CSR) using his/her private key. If the RA can verify the signature (i.e. de-sign the CSR) correctly using the public key of the user, the RA can believe that the user indeed possesses the private key.
- (ii) Alternatively, at this stage, the RA can create a random number challenge, encrypt it with the user's public key and send the encrypted challenge to the user. If the user can successfully decrypt the challenge using his/her private key, the RA can assume that the user possesses the right private key.
- (iii) Thirdly, the RA can actually generate a dummy certificate for the user, encrypt it using the user's public key and send it to the user. The user can decrypt it only if he/she can decrypt the encrypted certificate, and obtain the plain-text certificate.

Step 4: Certificate Creation Assuming that all the steps so far have been successful, the RA passes on all the details of the user to the CA. The CA does its own verification (if required) and creates a digital certificate for the user. There are programs for creating certificates in the X.509 standard format. The CA sends the certificate to the user, and also retains a copy of the certificate for its own record. The CA's copy of the certificate is maintained in a **certificate directory**. This is a central storage location maintained by the CA. The contents of the certificate directory are similar to that of a telephone directory. This facilitates for a single-point access for certificate management and distribution.

There is no single standard that describes the structure of a certificate directory. However, the X.500 standard is emerging as a popular choice. It allows the storage of not only the digital certificates, but also information about servers, printers, network resources, as well as the user's personal information, such as telephone numbers/extensions, email ids, etc., at a central place in a controlled manner. The directory clients can request for and access information from this central repository using a directory access protocol, such as the **Lightweight Directory Access Protocol (LDAP)**. LDAP allows users and applications to access X.500 directories, depending on their privileges.

The CA then sends the certificate to the user. This can be attached to an email, or the CA can send an email to the user, informing that the certificate is ready, and that the user should download it from the CA's site. The latter possibility is depicted in Fig. 5.13. As shown, the user gets a screen, which informs the user that his/her digital certificate is ready, and that he/she should download it from the CA's site.

After the user opts for downloading this certificate, he/she gets to see the following screen, as shown in Fig. 5.14. Note that this screen informs the user about the version, serial number, algorithms used for calculating the message digest and for signing the certificate, issuer, validity, subject details, etc.

The digital certificate is actually in an unreadable format to human eyes. This is shown in Fig. 5.15. As depicted, we can make little sense out of this certificate. However, an application program actually *parses* or *interprets* the certificate, and shows us its details in a human-readable format.

When we invoke the Internet Explorer browser (an application program) to view the certificate, we can see its details in a human-readable format, as shown in Fig. 5.16.

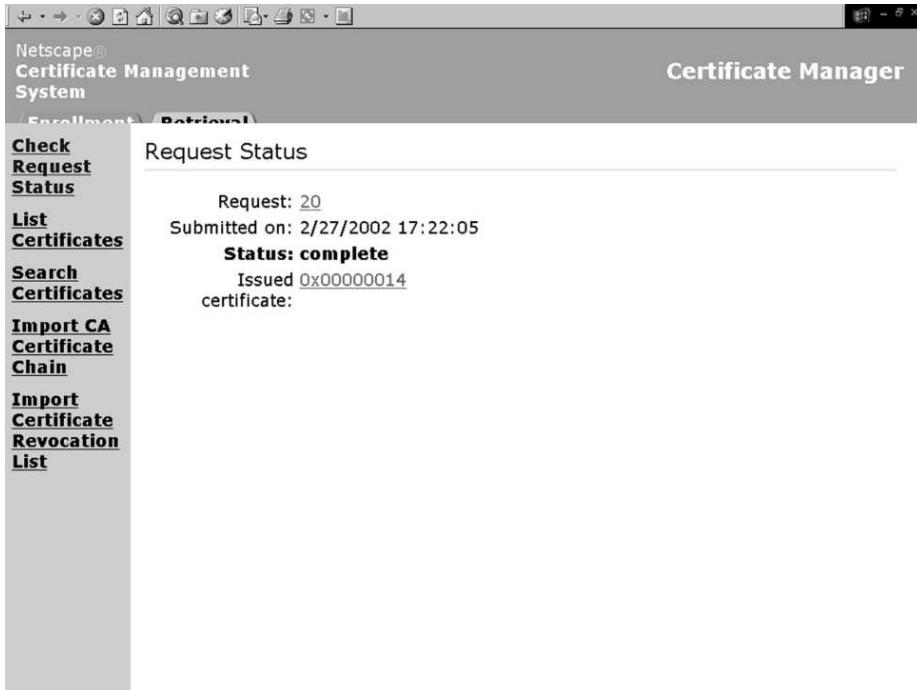


Fig. 5.13 CA informs the user that the certificate is ready and can be downloaded

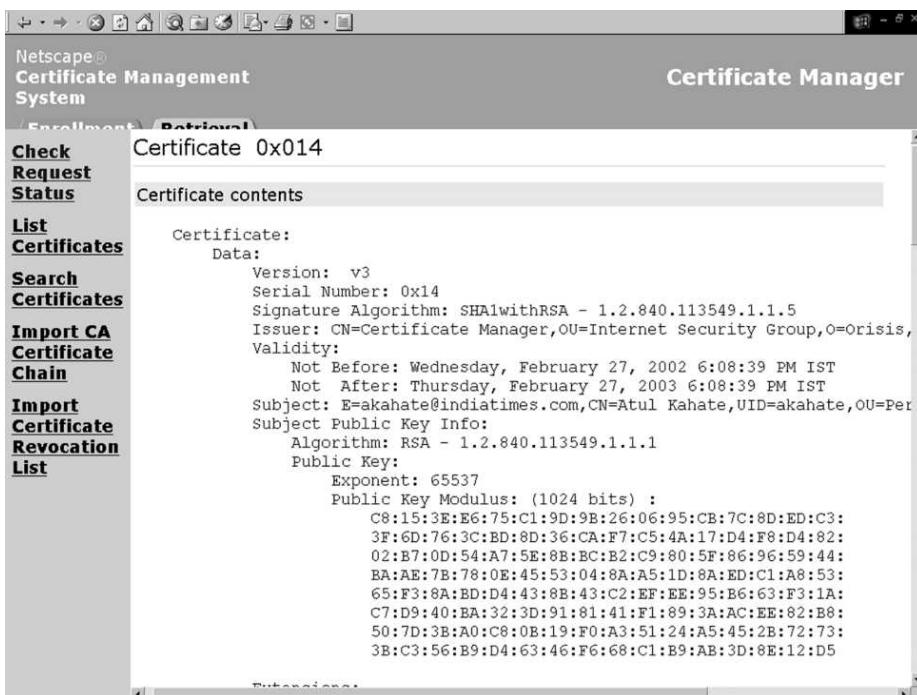


Fig. 5.14 Contents of a certificate

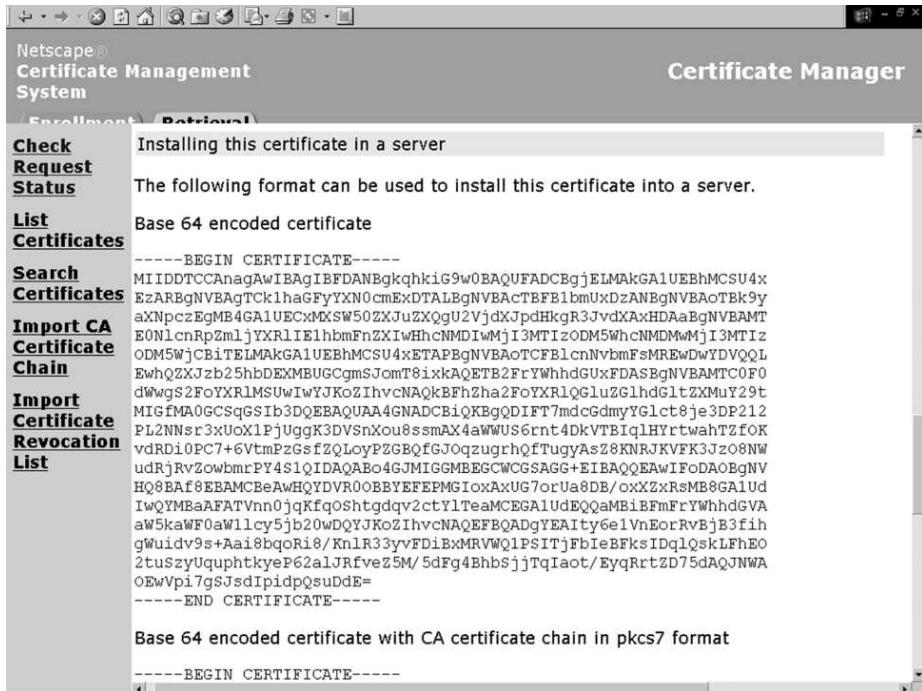


Fig. 5.15 Unreadable format of a certificate

5.2.6 Why Should We Trust Digital Certificates?

1. Introduction

We have deliberately not answered one extremely important question. Why should one trust a digital certificate? To answer this question, let us think about why we trust a passport. We do not trust a passport because it contains some information about the passport holder, but because it is in a pre-determined format, and is stamped and signed by an authority. So, would we trust a digital certificate just because it contains some information (most importantly, the public key) about a user? Obviously, we would not trust a digital certificate on this basis alone. After all, a digital certificate is simply a computer file. Therefore, I can as well create a digital certificate file with whatever public key I want to use, and use the certificate in business transactions! Even if this may not necessarily lead to serious business losses, it is clearly a lot of nuisance!

Consequently, a CA always signs a digital certificate with its private key. In effect, the CA says:

I have signed this certificate to guarantee that this user possesses the specified public key. Trust me!

2. How a CA Signs a Certificate

Now suppose that we have a digital certificate, which we want to verify. What should we do? Since the CA has signed the certificate with its private key, we would need to first verify the CA's signature. For this, we will use the CA's public key and check if it can de-sign the certificate correctly (we shall

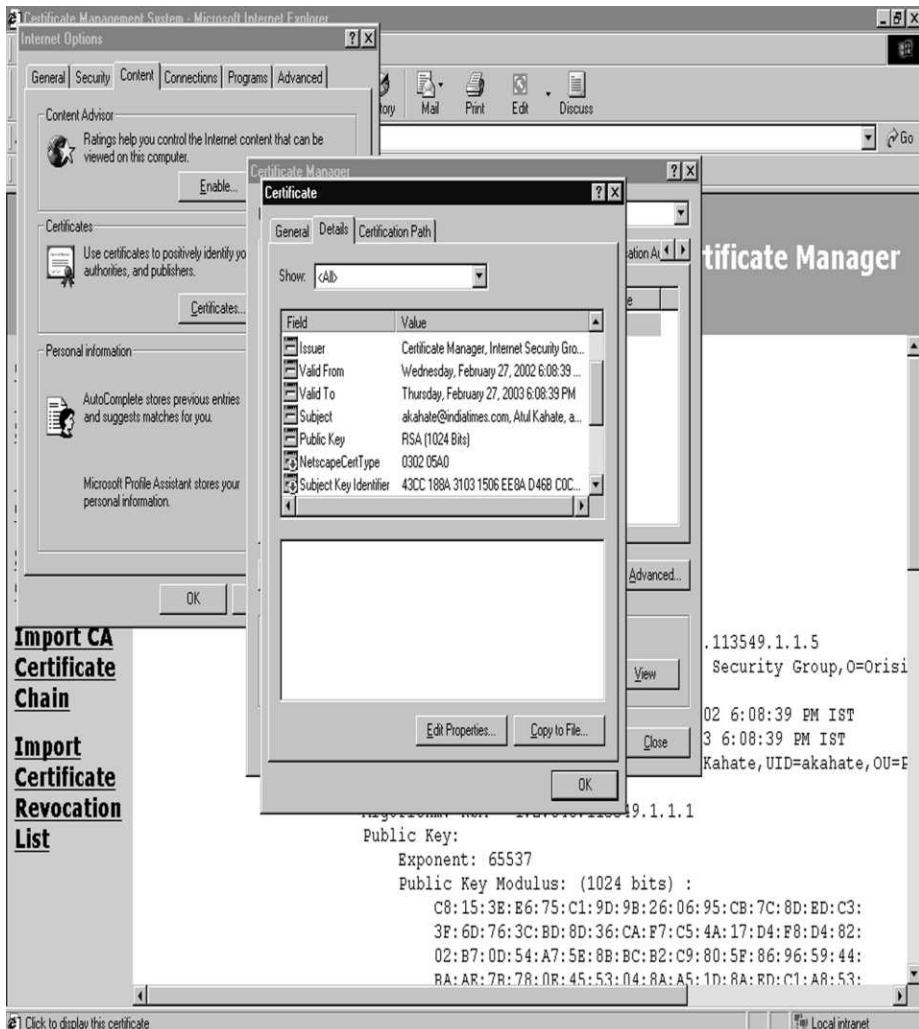
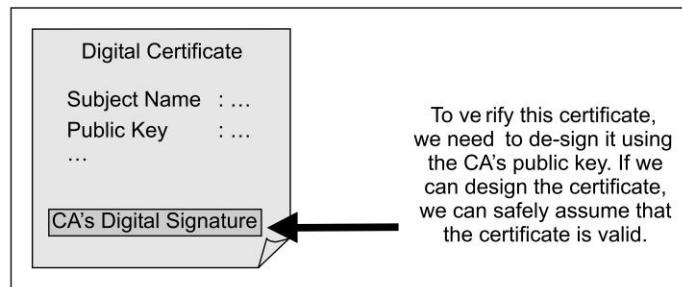
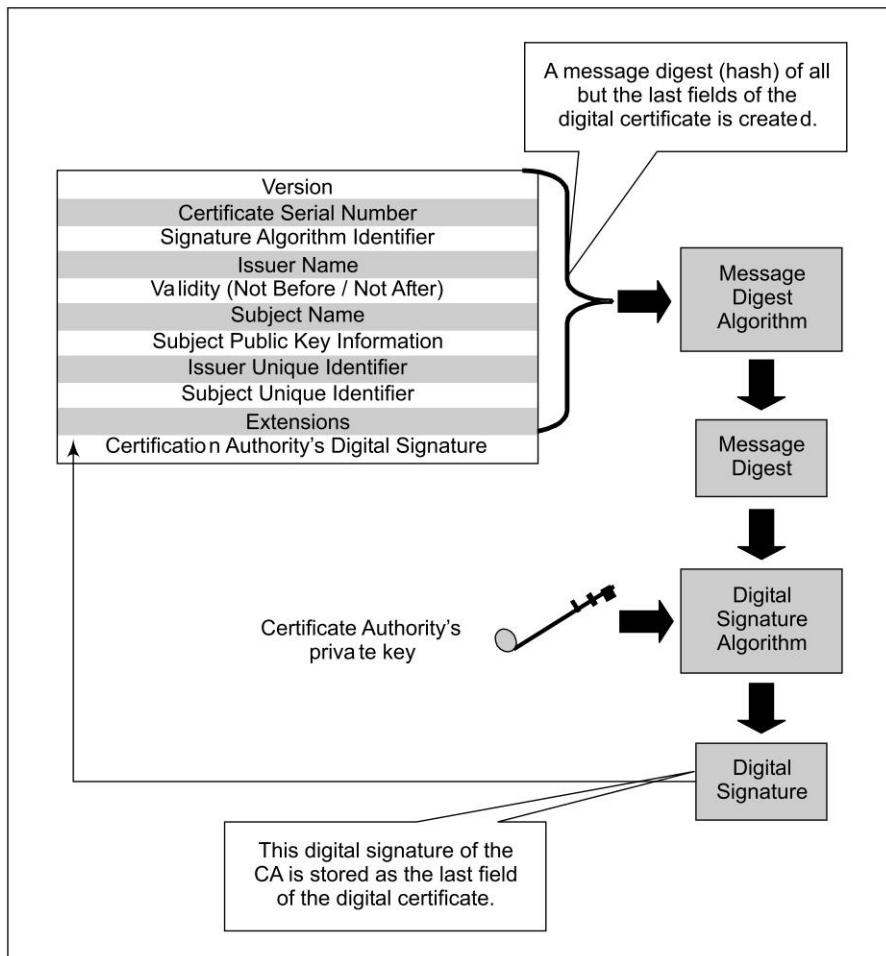


Fig. 5.16 Certificate in a human-readable format

shortly see what this means). If the de-signing works successfully, we can consider the certificate to be a valid one. This is the single and yet most important reason why we can safely trust digital certificates. This process is shown in Fig. 5.17.

Let us now understand how we can de-sign a certificate and check its trustworthiness. However, before that, we need to understand how the CA signs a certificate in the first place. If we recall our earlier discussion about the structure of an X.509 certificate, we will note that the last field in a digital certificate is always the digital signature of the CA. That is, every digital certificate not only contains the user's information (such as subject name, public key, etc.) but also the CA's digital signature. Like a passport, therefore, a digital certificate is always signed or certified. The way CA signs a certificate is shown in Fig. 5.18.

**Fig. 5.17** CA signs a certificate**Fig. 5.18** Creation of the CA signature on a certificate

As shown in the figure, before issuing a digital certificate to a user, the CA first calculates a message digest over all the fields of the certificate (using a standard message-digest algorithm such as MD5 or

SHA-1) and then encrypts the message digest with its private key (using an algorithm such as RSA) to form the CA's digital signature. The CA then inserts its digital signature thus calculated, as the last field in the digital certificate of the user. This is very similar to how an authority embosses, stamps and signs a passport after it is ready.

Of course, all of this process happens automatically, using computer-based cryptography programs.

3. How a Digital Certificate can be Verified

Having understood how the CA signs a digital certificate, let us now think how the verification of a certificate takes place. Suppose we receive a digital certificate of a user, which we want to verify. What should we do for this? Clearly, we need to verify the digital signature of the CA. Let us understand what steps are involved in this process, as shown in Fig. 5.19.

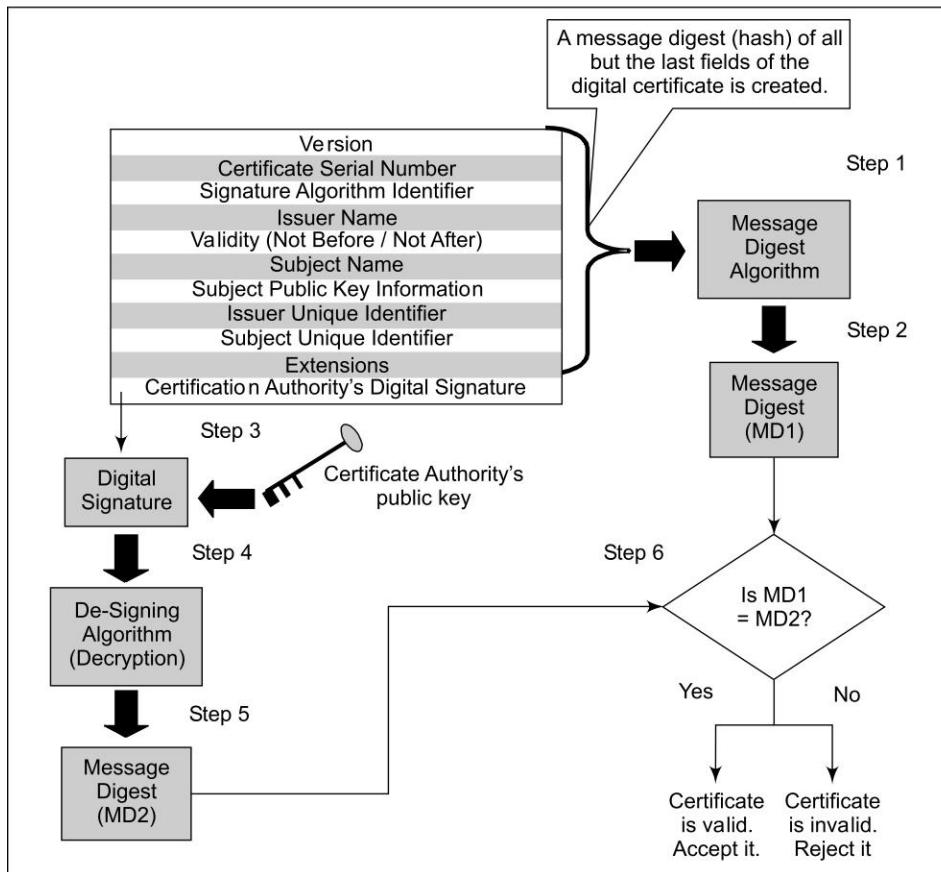


Fig. 5.19 Verification of the CA signature on a certificate

The verification of a digital certificate consists of the following steps.

- The user passes all fields except the last one of the received digital certificate to a message-digest algorithm. This algorithm should be the same as the one used by the CA while signing the certifi-

cate. The CA mentions the algorithm used for signing along with the signature in the certificate, so the user here knows which algorithm is to be used.

- (b) The message-digest algorithm calculates a message digest (hash) of all fields of the certificate, except for the last one. Let us call this message digest as MD1.
- (c) The user now extracts the digital signature of the CA from the certificate (remember, it is the last field in a certificate).
- (d) The user de-signs the CA's signature (i.e. the user decrypts the signature with the CA's public key).
- (e) This produces another message digest, which we shall call MD2. Note that MD2 is the same message digest as would have been calculated by the CA during the signing of the certificate (i.e. before it encrypted the message digest with its private key to create its digital signature over the certificate).
- (f) Now, the user compares the message digest it calculated (MD1) with the one, which is the result of de-signing the CA's signature (MD2). If the two match, i.e. if $MD1 = MD2$, the user is convinced that the digital certificate was indeed signed by the CA with its private key. If this comparison fails, the user will not trust the certificate, and reject it.

5.2.7 Certificate Hierarchies and Self-signed Digital Certificates

Another question that we might have at this stage is the **verification** of a digital certificate. In general, it looks quite satisfactory, except for one potential threat. Suppose that Alice has received Bob's digital certificate, and she wants to verify it. As we know, this means that Alice needs to de-sign the certificate using the CA's public key. Now, how does Alice know what is the CA's public key?

One possibility is that the CA of Alice and Bob is the same. In such a case, there is no problem, as Alice would already know the public key of the CA. However, this cannot always be guaranteed. For instance, Alice and Bob may not have obtained their certificates from the same CA. In such a case, how can Alice obtain the public key of the CA?

To resolve such problems, a **Certification Authority hierarchy** is created. This is also called the **chain of trust**. In simple terms, all the CAs are grouped into multiple levels of a CA hierarchy. This is shown in Fig. 5.20.

As the figure shows, the *Certification Authority (CA) hierarchy* begins with the **root CA**. The root CA has one or more second-level CAs below. Each of these second-level CAs can have one or more third-level CAs, which in turn can have lower level CAs, and so on. This is like a reporting hierarchy in an organization, where the CEO or the Managing Director is the highest authority. Many senior managers report to the CEO or the Managing Director. Many managers report to one senior manager. Many people would report to each manager, and so on.

What is the purpose of creating this hierarchy? Just as this sort of structure relieves the CEO or the Managing Director from performing all the possible tasks in all the departments, this sort of hierarchy relieves the root CA from having to manage all the possible digital certificates. Instead, the root CA can delegate this job to the second-level CAs. This delegation can happen region-wise (e.g. one second-level CA could be responsible for the Western region, another for the Eastern region, a third one for the Northern region, and a fourth one for the Southern region, etc.). Each of these second-level CAs could appoint third-level CAs state-wise within that region. Each third-level CA could delegate its responsibilities to a fourth-level CA city-wise, and so on.

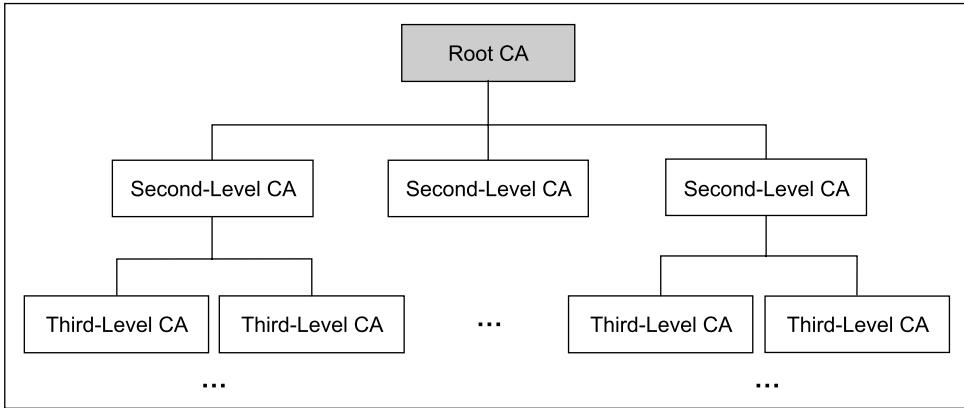


Fig. 5.20 CA hierarchy

Thus, in our example, if Alice has obtained her certificate from a third-level CA and Bob has obtained his certificate from a *different* third-level CA, how can Alice verify Bob's certificate? Let us understand this by naming the CAs for simplification, as shown in Fig. 5.21.

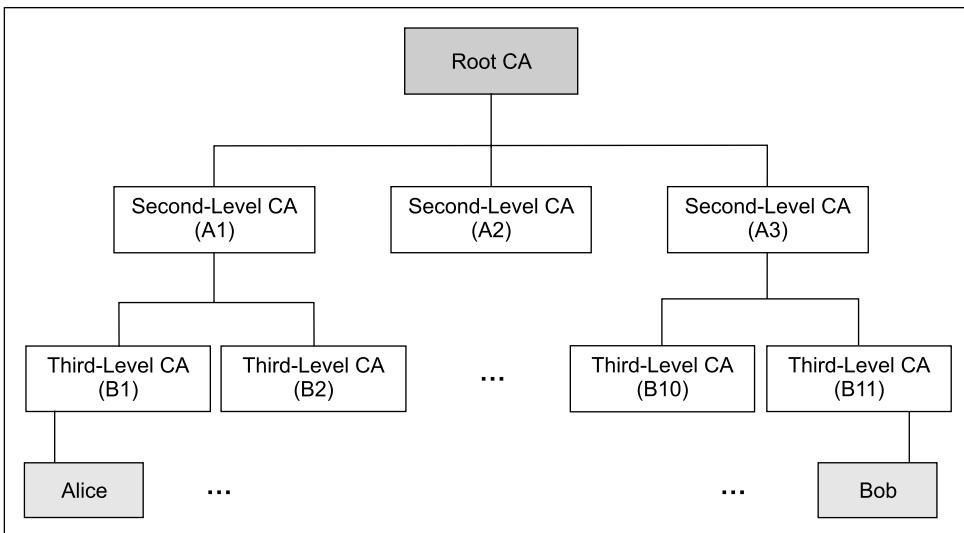


Fig. 5.21 Users belonging to different CAs under the same root CA

Now, suppose that Alice's CA is B_1 , whereas Bob's CA is B_{11} . Clearly, Alice cannot straightaway know the public key of B_{11} . Therefore, in addition to his own certificate, Bob must send the certificate of his CA (i.e. B_{11}) to Alice. This would tell Alice the public key of B_{11} . Now, using the public key of B_{11} , Alice can de-sign and verify Bob's certificate.

However, this raises the next question. How can Alice believe that B_{11} 's certificate can be trusted? What if it is a forged certificate, and does not actually belong to B_{11} ? This implies that Alice must also be able to verify the certificate of B_{11} . What would be required for this purpose? Obviously, for this, Alice needs to verify the signature on B_{11} 's certificate. From our diagram, we can see that B_{11} 's

certificate would be issued and signed by $A3$. Thus, Alice needs the certificate of $A3$ as well. $A3$ would have signed $B11$'s certificate. Therefore, Alice has to de-sign $B11$'s certificate using $A3$'s public key, and verify $B11$'s certificate.

You would have guessed the next problem by now. How can Alice come to know of $A3$'s public key? May be, she could ask Bob about it. But that leads to the same question: how can Alice be sure that the public key belongs to $A3$, and is not a forged one? Consequently, from our previous discussion, it should be straightforward to realize that Alice needs $A3$'s certificate as well. Using $A3$'s public key certificate, Alice can verify $B11$'s certificate.

Going one step further, Alice now needs to verify $A3$'s certificate. For this, on the basis of the diagram as well as our discussion so far, it should be quite clear that Alice needs the root CA's certificate as well. Assuming that she gets it, we have been able to successfully verify $A3$'s certificate.

Unfortunately, we have still not come out of the problem loop! The question is, how can we verify the root CA (i.e. the root CA's public key)? Where are we heading? Is this going to go on and on? Also, since the root CA is the last in the validation chain, there seems to be no way to validate its certificate. Who would issue a certificate to the root CA? This problem is shown in Fig. 5.22.

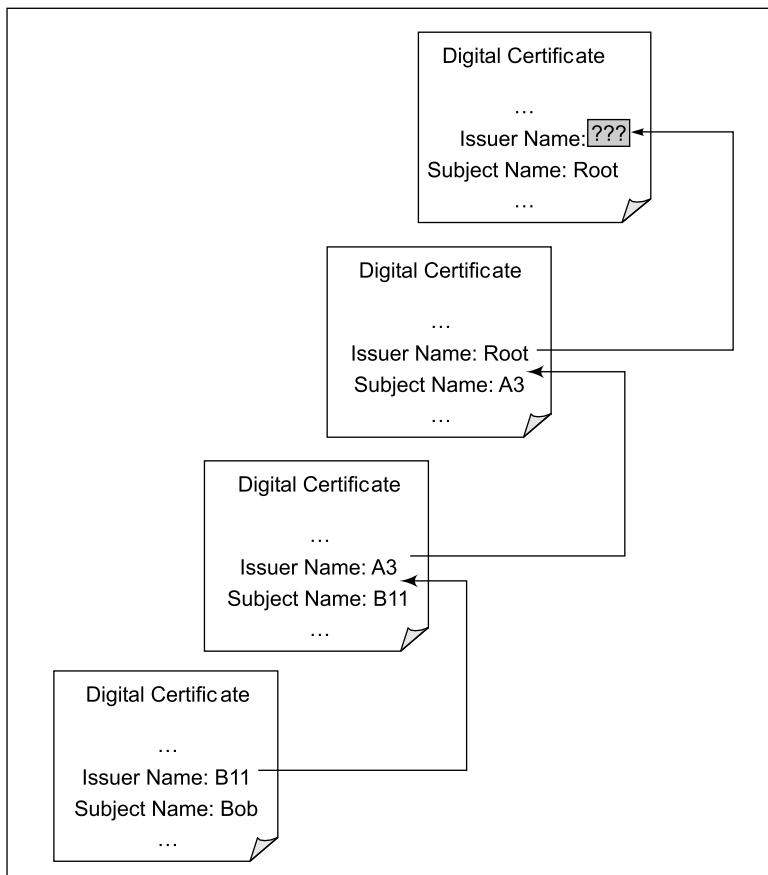


Fig. 5.22 Certificate hierarchy and the problem of the verification of a root CA

Fortunately, there is a way out. The root CA (and many times, even the second or the third-level CAs) are automatically considered trusted CAs. How is this achieved? For this, Alice's software (usually the Web browser, but this can be any other piece of software that is able to store and verify certificates) contains a pre-programmed, hard-coded certificate of the root CA. Also, this certificate of the root CA is a **self-signed certificate**, i.e. the root CA signs its own certificate. Technically, what does this mean? It is quite simple to understand, as shown in Fig. 5.23. Simply put, the issuer name and the subject name both point to the root CA in this certificate.

Since this certificate comes as a part of the basic software such as a Web browser or a Web server, Alice need not worry about the authenticity of the root certificate, unless the basic software that she is using itself comes from an untrusted site. As long as Alice restricts herself to industry-standard, well-accepted software applications (i.e. usually Web browsers and Web servers), she can be confident about the validity of the root CA's certificate.

This process of verifying the chain of certificates is shown in Fig. 5.24.

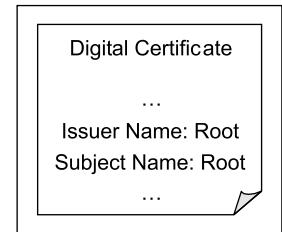


Fig. 5.23 Self-signed certificate

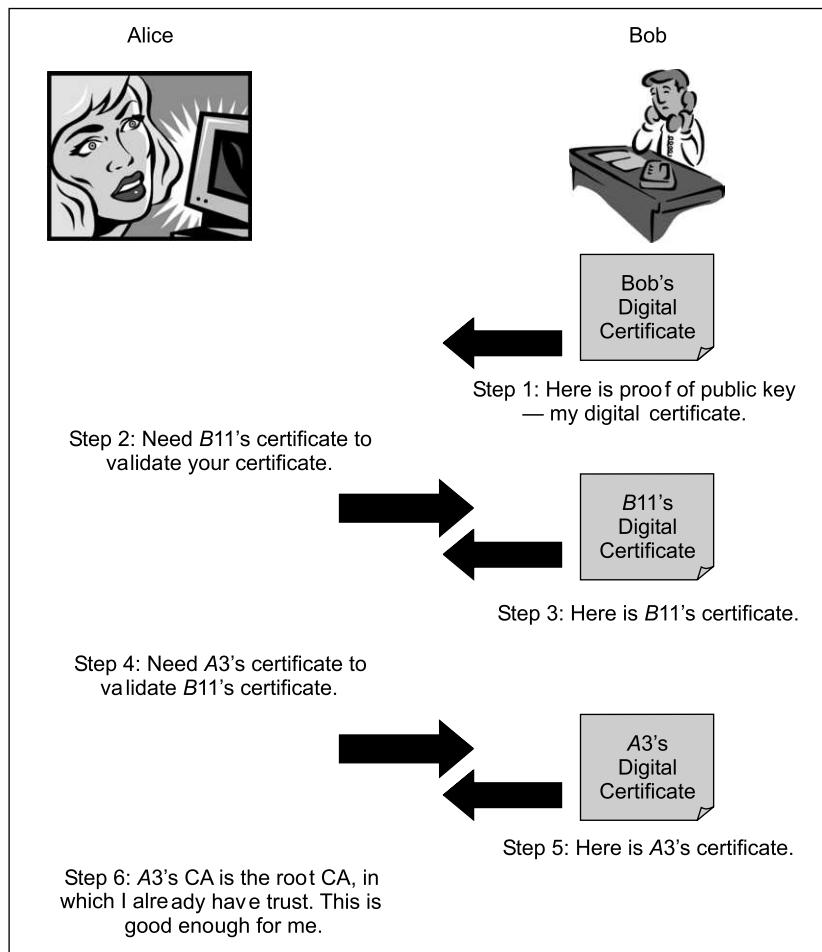


Fig. 5.24 Verification of a root CA

Of course, the actual sequence of operations need not be so elaborate in real life. Perhaps, Bob would send all the certificates up to the root CA (i.e. his own, and those of $B11$ and $A3$) in the first message itself to Alice. This is called the **push model**. However, we have shown these steps in great detail, so as to understand the process in minute detail, which is also allowed in practice, and is called the **pull model**.

5.2.8 Cross-Certification

We have not resolved all the problems yet. It is quite possible that Alice and Bob live in different countries. This would mean that their root CAs themselves could be different. This is because, generally each country appoints its own root CA. In fact, one country can have multiple root CAs as well. For instance, the root CAs in the US are VeriSign, Thawte, and the US Postal Service. In such cases, there is no *single* root CA, which can be trusted by all the concerned parties. In our example, why should Alice—a Japanese national, trust Bob’s root CA—a US-based organization?

We can imagine that this can lead us to the same old story of a never-ending chain of certification authority hierarchy and their validations. Thankfully, there is an alternative, called **cross-certification**. This came about because the concept of a single monolithic CA certifying every possible user in the world is quite unlikely. Instead, the concept of decentralized CAs for different countries, political organizations and businesses is far better. This allows the CAs not only to worry about a smaller population of users, but also work independently. Moreover, *cross-certification* allows CAs and end users from different PKI domains to interact.

More specifically, cross-certification certificates are issued by the CAs to create a non-hierarchical trust path. Let us understand this with an example as shown in Fig. 5.25.

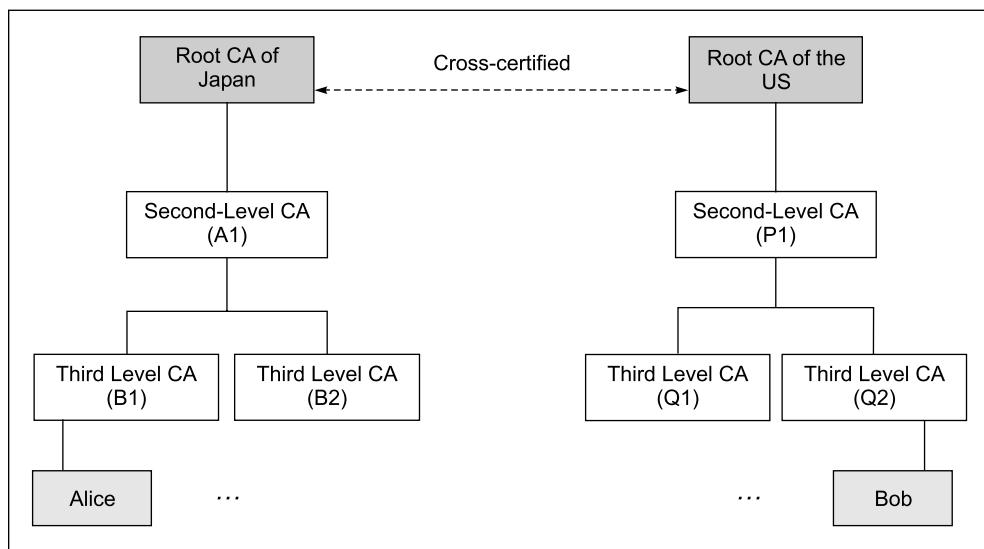


Fig. 5.25 Cross-certification of CAs

As the figure shows, the root CAs of Alice and Bob are different—but they have cross-certified each other. Technically, this means that Alice’s root CA has obtained a certificate for itself from Bob’s root

CA. Similarly, Bob's root CA has obtained a certificate for itself from Alice's root CA. Now, even if Alice's software trusts only her own root CA, it is not a problem. Since Bob's root CA is certified by Alice's root CA, Alice can and does trust the root CA of Bob. This also means that Alice can verify Bob's certificate using the path Bob—Q2—P1—Bob's root CA – Alice's root CA. We shall not describe how this happens, as we have already explained this process in great detail earlier.

The morale of the story is that with the technologies of certificate hierarchies, self-signed certificates and cross-certification, virtually any user can verify any other user's digital certificate and based on that, decide to either trust it or reject it.

5.2.9 Certificate Revocation

1. Introduction

If your credit card is lost or if it gets stolen, you would normally immediately report the loss to the concerned bank. The bank would cancel your credit card. Similarly, a digital certificate can also be revoked. What can be the reasons for the revocation of a digital certificate? Some of the most common ones are as follows:

- The holder of the digital certificate reports that the private key corresponding to the public key specified in the digital certificate is compromised (i.e. someone has stolen it).
- The CA realizes that it had made some mistake while issuing a certificate.
- The certificate holder leaves a job, and the certificate was issued specifically while the person was employed in that job.

In all such cases, the digital certificate of the user gets the same status as that of a stolen credit card. Therefore, the certificate must be treated as invalid. For this purpose, the certificate must be revoked. What is the process for this? Normally, just as a credit-card user reports a credit-card loss or theft, a certificate holder should report that a certificate should be revoked. Of course, if the user leaves an organization or indulges in an illegal act because of which the certificate needs to be revoked, the organization should initiate the process. Finally, if the CA realizes its own mistake in providing a wrong certificate, the CA initiates the certification revocation process.

In any case, the CA must come to know about this certificate revocation request. Also, the CA must authenticate the certificate revocation requester before accepting this revocation request. Otherwise, someone can misuse the certificate revocation process to potentially request for the revocation of a certificate that belongs to another user!

Let us assume that Alice wants to use Bob's certificate for securely communicating with him. However, before she uses Bob's certificate, Alice wants answers to the following two questions:

- Does this certificate really belong to Bob?
- Is this certificate valid, or is it revoked?

We know that Alice can use the certificate chain to establish the first fact. Let us assume that Alice is satisfied that the certificate indeed belongs to Bob. Now, she has to check the second aspect, which is, whether the certificate is still valid or not. How can Alice do this check? For this, the CA provides the facilities as shown in Fig. 5.26.

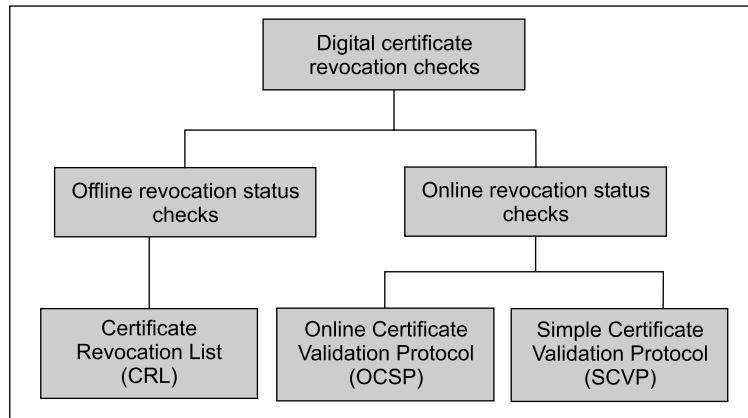


Fig. 5.26 Certificate revocation status mechanisms

Let us discuss these revocation status check schemes now.

2. Offline Certificate Revocation Status Checks

The **Certificate Revocation List (CRL)** is the primary means of checking the status of a digital certificate offline. In its simplest form, a CRL is a list of certificates published regularly by each CA, identifying all the certificates that have been revoked through the life of the CA. However, this list does not include certificates whose validity period is over. A CRL lists only those certificates whose validity period is still within the acceptable range, but are revoked for some other reason (such as the ones listed earlier).

Each CA issues its own CRL. The respective CA signs each CRL. Therefore, a CRL can be easily verified. A CRL is simply a sequential file that grows over time to include all the certificates that have not expired, but have been revoked. Thus, it is a superset of all the previous CRLs issued by that CA. Each CRL entry lists the certificate serial number, the date and time on which the certificate was revoked, and the reason behind the revocation. At the top level, the CRL also includes the information such as the date and the time this CRL was published and when the next CRL will be published. This logical view of a CRL file is shown in Fig. 5.27.

CA: XYZ Certificate Revocation List (CRL) This CRL: 1 Jan 2007, 10:00 am Next CRL: 12 Jan 2007, 10:00 am		
Serial Number	Date	Reason
1234567	30-Dec-01	Private key compromised
2819281	30-Dec-01	Changed job
...

Fig. 5.27 Logical view of a CRL

Thus, when Alice receives Bob's certificate, and wants to see if she should trust it, she should do the following, in the given sequence:

Certificate Expiry Check Compare the current date with the validity period of the certificate to ensure that the certificate has not expired

Signature Check Check that Bob's certificate can be verified in terms of the signature by his CA

Certificate Revocation Check Consult the latest CRL issued by Bob's CA to ensure that Bob's certificate is not listed there as a revoked certificate

Only after Alice is assured of all these three aspects, she can trust Bob's certificate. This is shown in Fig. 5.28.

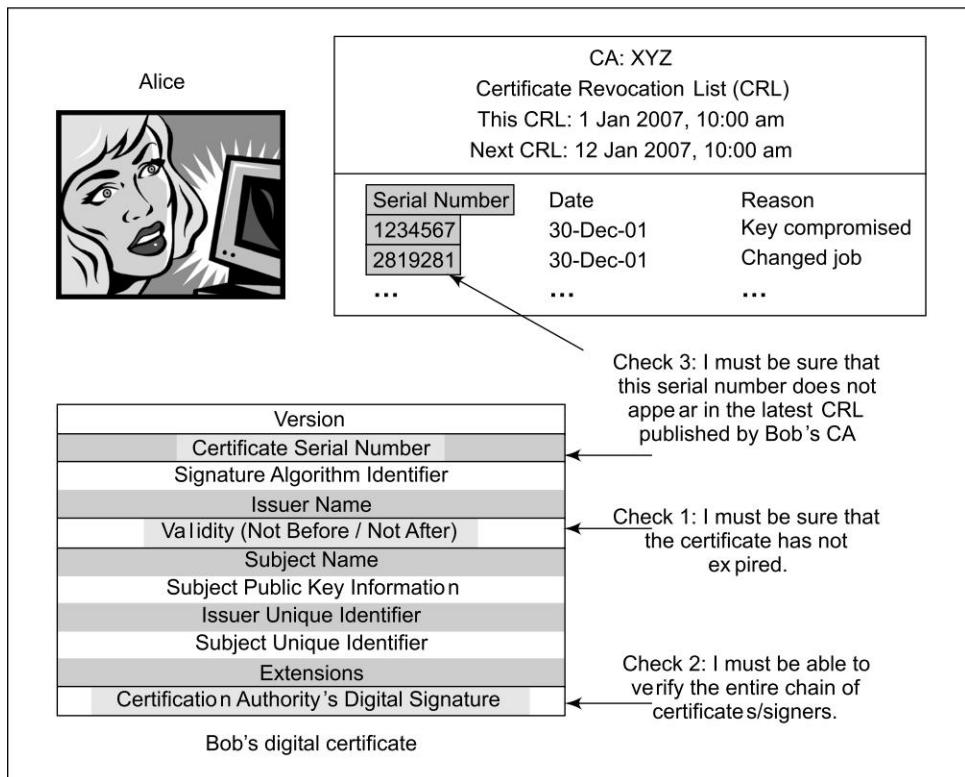


Fig. 5.28 Validating a certificate and CRL's role in the validation process

A CRL can become really quite big over time. The general assumption is that about 10% of unexpired certificates will be revoked every year. Thus, if a CA has about 100,000 users, in two years time, its CRL could contain 20,000 entries! This is quite large. Moreover, remember that CRL checks need to be performed even by users who want to use the certificates of other users in applications that execute on handheld devices! If a mobile phone has to store and check a CRL of size 20,000 users, it has to receive the CRL file first over the network, which is a great bottleneck. This problem led to the concept of **delta CRL**.

Initially, a CA can send a one-time full up-to-date CRL to the users who want to use the CRL services. This is called the **base CRL**. However, at the time of the next update, the CA need not send the entire CRL file once again. Instead, the CA can simply issue the changes (called a *delta*) to the CRL since the last update. This mechanism makes the CRL file size small, and therefore, its transmission easier. The changes to the base CRL are called **delta CRL**. The delta CRL is also a file, like CRL, which is also signed by the CA. The difference between issuing a complete CRL every time versus only a delta CRL is shown in Fig. 5.29.

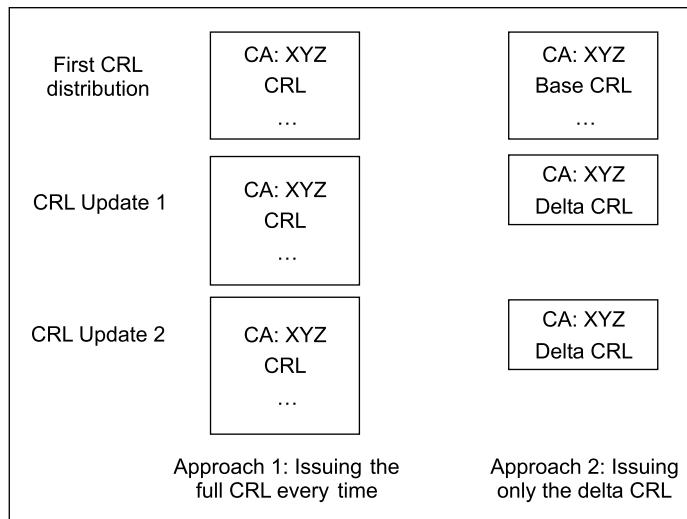


Fig. 5.29 Delta CRL

Several points need to be kept in mind when working with delta CRLs. Firstly, a delta CRL file contains an indicator called the **delta CRL indicator**. This indicator informs the user that this CRL file is not a complete, comprehensive CRL file, but instead, it is only a delta CRL. Therefore, the user knows that she has to refer to the base CRL along with this CRL file in order to obtain the complete CRL. Secondly, each CRL also contains a *sequence number*, which allows the user to check if all the delta CRLs are available with her, or if any intermediate delta CRL file is missing. Thirdly, a base CRL may also contain an indicator called **delta information**, which informs a user that delta CRLs are also available corresponding to this base CRL. The location of the delta CRLs and optionally the time when the next delta CRL will be available are also provided.

One point must be kept in mind about delta CRLs. They merely help reduce the network transmission overheads, but they do not reduce the certificate revocation check processing time or efforts in any manner. This is because the user has to anyway check the base CRL and all the delta CRLs to be satisfied that a particular certificate has not been revoked.

All the while, we have not studied why CRL is called an *offline* certificate revocation status check. The reason why CRL is offline is that the CRL is issued periodically by the CA. This period can vary from a few hours to a few weeks. Thus, Alice can have a CRL from Bob's CA, issued on 1 January 2002, and the current date could actually be 10 January 2002. This means that theoretically there is a chance that in-between 1st and 10th January, Bob's certificate could have been revoked. However, Alice has

no chance to know this—she cannot check Bob’s certificate status online—hence the name *offline* for CRLs. This latency is a major drawback of the CRL approach.

Another important point about CRLs is that the CA always issues new CRLs periodically even if there are no changes to the previous CRL (i.e. even if there is no new case of certificate revocation). This helps the CRL users to know that they are always using the latest CRL.

Let us now take a look at the standard format of a CRL. The various fields that constitute a CRL are shown in Fig. 5.30.

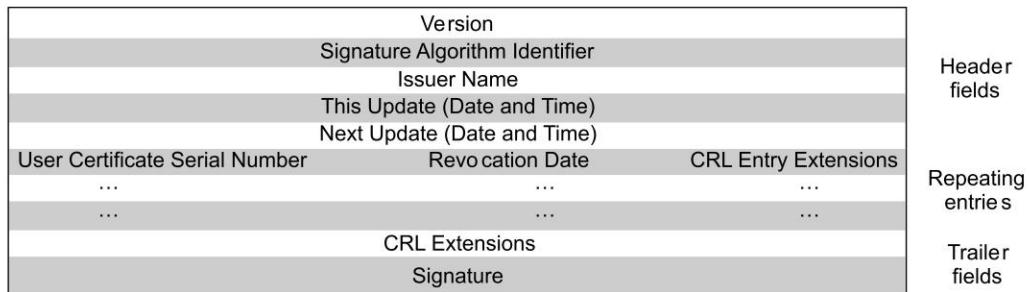


Fig. 5.30 Format of a CRL

As shown in the figure, there are some header fields, some repeating fields, and then some trailer fields. Obviously, fields such as the serial number, the revocation date and CRL Entry Extensions (discussed subsequently) repeat for every revoked certificate entry in the CRL. Other fields form the header and the trailer portions. Let us now describe these fields, as shown in Fig. 5.31. We have already discussed some of these in brief.

Field	Description
Version	Indicates the version of the CRL.
Signature Algorithm Identifier	Identifies the algorithm used by the CA to sign the CRL (e.g. it could be SHA-1 with RSA, which indicates that the CA first calculated the message digest of the CRL using the SHA-1 algorithm, and then signed it (i.e. encrypted the message digest with its private key) using the RSA algorithm.
Issuer Name	Identifies the Distinguished Name (DN) of the CA.
This Update (Date and Time)	Contains the date and time value when this CRL was issued.
Next Update (Date and Time)	Contains the date and time value when the next CRL will be issued.
User Certificate Serial Number	Contains the certificate number of the revoked certificate. This field repeats for every revoked certificate.
Revocation Date	Contains the revocation date and time of the revoked certificate. This field repeats for every revoked certificate.
CRL Entry Extensions	Discussed subsequently, these extensions are one per revoked certificate.
CRL Extensions	Discussed subsequently, these extensions are one per entire CRL.
Signature	Contains the CA signature.

Fig. 5.31 Description of the fields of a CRL

We should clearly distinguish between *CRL Entry Extensions* and *CRL Extensions*. Whereas the *CRL Entry Extensions* repeat for every revoked certificate, the *CRL Extensions* are for the entire CRL as a whole.

Let us now discuss the *CRL Entry Extensions* first. The CRL format currently comes in two versions: 1 and 2. Just as a X.509 digital certificate can be enhanced by using extensions in versions 2 and 3, the version 2 of CRLs have the provision for extensions that allow the CA to convey additional information about each individual certificate revocation. There are four possible *CRL Entry Extensions* in version 2 of the CRL format, as shown in Fig. 5.32.

Field	Description
Reason Code	Specifies the reason for certificate revocation. This reason can be Unspecified, Key compromise, CA Compromise, Superseded, Certificate hold.
Hold Instruction Code	Interestingly, a certificate can be put on hold or suspended temporarily. This means that the certificate should be considered as invalid for a specified time period (may be because the user is on a long holiday, and wants to ensure that during his/her absence, nobody can misuse the certificate). In such cases, this field is used to specify why the certificate is on hold.
Certificate Issuers	Identifies the name of the certificate issuer along with an indirect CRL. An indirect CRL is provided by a third party, and not by the CA who initially issued the certificates. Thus, the third party can consolidate the CRLs of multiple CAs, and issue a single, combined, indirect CRL, making life easier for the requester of the CRL information.
Invalidity Date	Contains the date and time value when the suspected or known compromise of the private key occurred.

Fig. 5.32 CRL Entry Extensions

For a given CRL, there can be many CRL extensions, as shown in Fig. 5.33.

Field	Description
Authority Key Identifier	Can be used to distinguish between the multiple CRL signing keys used by a CA.
Issuer Alternative Name	Associates one or more alternative names with an issuer.
CRL Number	Contains a sequence number (which increases with every CRL release) that helps a user know if the user has seen all the CRLs prior to this one.
Delta CRL Indicator	If used, flags the CRL as a delta CRL.
Issuing Distribution Point	Identifies the CRL distribution point (also called CRL partition) for a given CRL. A CRL distribution point can be used when a CRL becomes too large. Instead of a single, large CRL, several smaller CRLs are created for distribution. The requestors of the CRL request for and process these smaller CRLs. The Issuing Distribution Point supplies a pointer to the location of the smaller CRLs (i.e. the DNS name, IP address or file names of the smaller CRLs).

Fig. 5.33 CRL entry extensions

Incidentally, as we know, like end users, CAs themselves are identified by certificates. Just as end-user certificates need revocation, so can CA certificates. An **Authority Revocation List (ARL)** provides a list of the revocation information for CA certificates, just as a CRL provides the revocation information for end user certificates.

3. Online Certificate Revocation Status Checks

Realizing that CRL may not always be the best way to check the revocation of certificates (because of its size as well as the likelihood of it being stale), two new protocols were developed for checking the status of a certificate online, namely the **Online Certificate Status Protocol (OCSP)** and **Simple Certificate Validation protocol (SCVP)**.

(a) Online Certificate Status Protocol (OCSP) The **Online Certificate Status Protocol (OCSP)** can be used to check if a given digital certificate is valid at a particular moment. Thus, it is an *online check*. OCSP allows the certificate validators to check for the status of certificates in real time, thus providing for a quicker, simpler and more efficient mechanism for digital certificate validations. Unlike CRL, there is no downloading required here. Let us understand how OCSP works, step-by-step.

- (i) The CA provides a server, called as an **OCSP responder**. This server contains the latest certificate revocation information. The requestor (client) has to send a query (called **OCSP request**) about a particular certificate to check if it is revoked or not. The underlying protocol for OCSP is most commonly HTTP, although other application-layer protocols (such as SMTP) can be used. This is shown in Fig. 5.34. Actually, this is technically not completely right. In practice, the OCSP request contains the OCSP protocol version, the service requested and one or more certificate identifiers (which, in turn, consist of a message digest of the issuer's name, a message digest of the issuer's public key and the certificate serial number). However, we shall ignore this for the sake of simplicity.

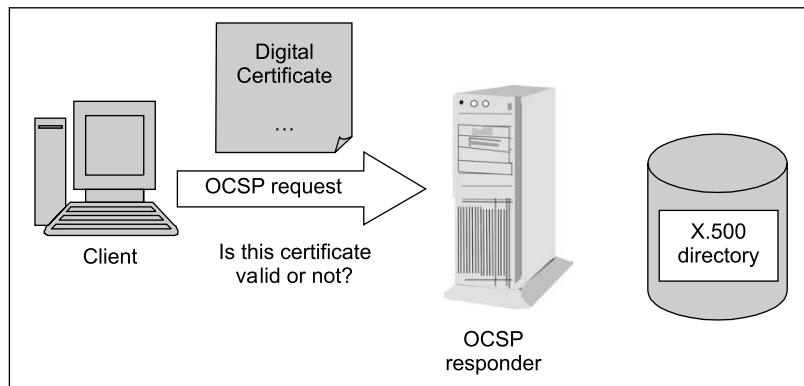


Fig. 5.34 OCSP request

- (ii) The OCSP responder consults the server's X.500 directory (in which the CA continuously feeds the certificate revocation information) to see if the particular certificate is valid or not. This is shown in Fig. 5.35.

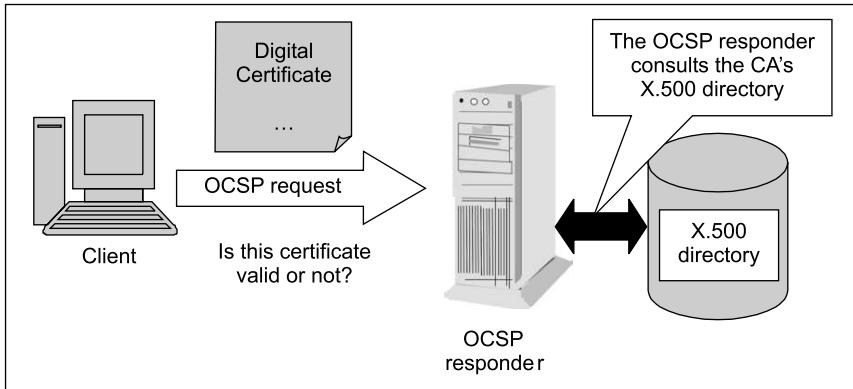


Fig. 5.35 OCSP certificate revocation status check

- (iii) Based on the result of the status check from the X.500 directory lookup, the OCSP responder sends back a digitally signed **OCSP response** for each of the certificates in the original request to the client. This response can take one of the three forms, namely, *Good*, *Revoked*, or *Unknown*. The *OCSP response* may also include the date, time and reason for revocation, if the certificate is revoked. The client has to determine what action to take accordingly. Generally, the recommendation is to consider the certificate as valid only if the *OCSP response* is '*good*'. This is shown in Fig. 5.36.

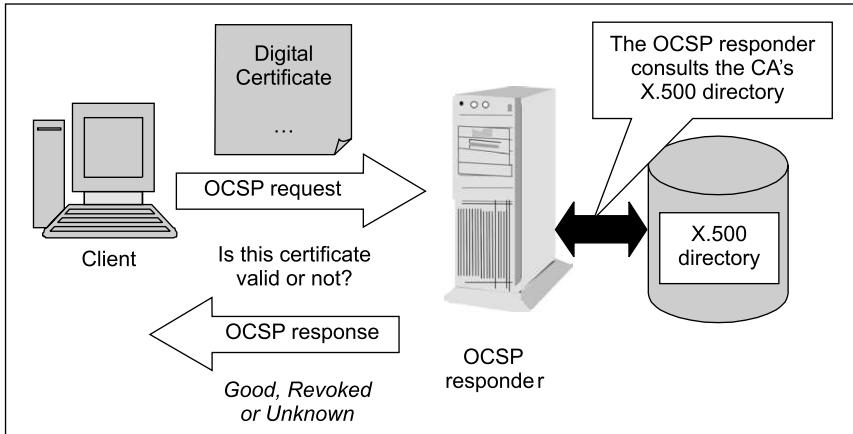


Fig. 5.36 OCSP response

However, OCSP itself has some problems. OCSP does not check the validity of the chain of certificates associated with the current certificate. For instance, suppose Alice wants to verify Bob's certificate using OCSP. Then OCSP will simply inform Alice if Bob's certificate is valid or not. OCSP will not validate the certificate of that CA who had issued the certificate to Bob, or any of the higher level CAs in the chain. All this logic (of verifying the chain of trust higher up) must be contained in the client application that uses OCSP. Also, the certificate validity period, key usage compliance and other constraints must be checked by the client application.

Interestingly, an OCSP responder may be programmed to interact with a CRL, instead of a X.500 directory certificate store. This means that even the use of OCSP in this case would lead to stale information!

(b) Simple Certificate Validation Protocol (SCVP) The **Simple Certificate Validation Protocol (SCVP)** is in the draft stage as of the current writing. SCVP is an online certificate status reporting protocol, designed to deal with the drawbacks of OCSP. Since SCVP is conceptually similar to OSCP, we shall simply point out the differences between the two, as shown in Fig. 5.37.

Point	OCSP	SCVP
Client request	The client sends just the certificate serial number to the server.	The client sends the entire certificate to the server. Consequently, the server can perform many more checks.
Chain of trust	Only the given certificate is checked.	The client can provide a collection of the intermediate certificates, which the server can check.
Checks	The only check is whether the certificate is revoked or not.	The client can request for additional checks (e.g. please check the full chain of trust), type of revocation information to be considered (i.e. whether the server should, in turn, use CRL or OCSP for revocation checks), etc.
Returned information	Only the status of the certificate is returned by the server.	The client can specify what additional information it is interested in (e.g. proof of the revocation status should be returned by the server, or the chain of certificates used for chain of trust validation should be returned by the server, etc.).
Additional features	None	The client can request for a certificate to be checked for a backdated event. For instance, suppose Bob has sent Alice a signed document along with his certificate. Then, Alice can use SCVP to check if Bob's certificate was valid at the time of signing (and not at the time of verification of the signature).

Fig. 5.37 Differences between OCSP and SCVP

Interestingly, the OCSP protocol itself is being enhanced, and its enhanced version, **OCSP Extensions** (or **OCSP-X**) is currently in the proposal stage. The goals of OCSP-X are similar to those of SCVP.

5.2.10 Certificate Types

Not all digital certificates have the same status and cost. Depending on the requirement, these differ. For instance, a digital certificate can be used by a user only for encrypting messages, but not for digitally signing any messages. In contrast, a merchant setting up its online shopping site may use a high-cost digital certificate, which covers many areas.

Generally, the certificate types can be classified as follows:

(a) Email Certificates Email certificates include the user's email id. This is used to verify that the signer of an email message has an email id that is the same as it appears in that user's certificate.

(b) Server-side SSL Certificates These certificates are useful for merchants who want to allow buyers to purchase goods or services from their online Web site. We shall discuss this in detail later. Since a misuse of this certificate can cause serious damages, such certificates are issued only after a careful scrutiny of the merchant's credentials.

(c) Client-side SSL Certificates These certificates allow a merchant (or any other server-side entity) to verify a client (browser-side entity). We shall discuss these certificates in detail later.

(d) Code-signing Certificates Many people do not like to download client-side code such as Java applets or ActiveX controls, because of the inherent risks associated with them. In order to alleviate these concerns, the code (i.e. the Java applets or ActiveX controls) can be signed by the signer. When a user hits a Web page that contains such code, the browser displays a warning message, indicating that the page contains such pieces of code, signed by the appropriate developer/organization, and whether the user would like to trust that developer/organization. If the user responds affirmatively, the Java applets or ActiveX controls are downloaded and get executed on the browser. However, if the user rejects the offer, the processing ends there. It must be noted that mere signing of code does not make it safe—the code could cause havoc. It simply specifies where the code originates.

■ 5.3 PRIVATE-KEY MANAGEMENT ■

5.3.1 Protecting Private Keys

All this while, we have been concentrating on the digital certificates, and therefore, on the public key of a user. However, we have not thought about the private key at all. As we know, a user must hold the private key secretly. It must not be possible for another user to access someone's private key. How can a private key be protected? There are several mechanisms, as shown in Fig. 5.38.

In many situations, the private key of the user might be required to be transported from one location to another. For instance, suppose the user wants to change his/her PC. To handle these situations, there is a cryptographic standard by the name PKCS#12. This allows a user to export his/her digital certificate and private key in the form of a computer file. Obviously, the certificate and the private key must be protected as they are moved to another location. For this, the PKCS#12 standard ensures that they are encrypted using a symmetric key, which is derived from the user's private-key protection password.

5.3.2 Multiple Key Pairs

The PKI approach also recommends that in serious business applications, users should possess multiple digital certificates, which also means multiple key pairs. The need for this is that one certificate could be strictly used for signing, and another for encryption. This ensures that the loss of one of the private keys does not affect the complete operations of the user. The following guidelines are generally helpful:

- The private key that is used for digital signing (non-repudiation) must not be backed up or archived after it expires. It must be destroyed. This ensures that it is not used by someone else for signing on

Mechanism	Description
Password protection	This is the simplest and most common mechanism to protect a private key. The private key is stored on the hard disk of the user's computer as a disk file. This file can be accessed only with the help of a password or a <i>Personal Identification Number (PIN)</i> . Since anyone who can guess the password correctly can access the private key, this is considered the least secure method of protecting a private key.
PCMCIA cards	The Personal Computer Memory Card International Association (PCMCIA) cards are actually chip cards. The private key is stored on such a card, which means that it need not be on the user's hard disk. This reduces the chances of it being stolen. However, for a cryptographic application such as signing or encryption, the key must travel from the PCMCIA card to the memory of the user's computer. Therefore, there is still scope for it being captured from there by an attacker.
Tokens	A token stores the private key in an encrypted format. To decrypt and access it, the user must provide a one-time password (which means that the password is valid only for that particular access; next time, this password becomes invalid, and another must be used). We shall later discuss how this works. This is a more secure method.
Biometrics	The private key is associated with a unique characteristic of an individual (such as fingerprint, retina scan or voice comparison). This is similar in concept to the tokens, but here the user need not carry anything with him/her, unlike the token.
Smart cards	In a smart card, the private key of the user is stored in a tamperproof card. This card also contains a computer chip, which can perform cryptographic functions such as signing and encryption. The biggest benefit of this scheme is that the private key never leaves the smart card. Thus, the scope for its compromise is tremendously reduced. The disadvantage of this scheme is that the user needs to carry the smart card with him/her, and compatible smart-card readers must be available to access it.

Fig. 5.38 Mechanisms for protecting private keys

behalf of the person at a future date (although chances are that this will be detected by CRL/OCSP checks or certificate expiry date checks, you cannot say this with a 100% guarantee).

- In contrast, the private key used for encryption/decryption must be backed up after its expiry, so that encrypted information can be recovered even at a later date.

5.3.3 Key Update

Good security practices demand that the key pairs should be updated periodically. This is because over time, keys become susceptible to cryptanalysis attacks. Causing a digital certificate to expire after a certain date ensures this. This requires an update to the key pair. The expiry of a certificate can be dealt with in one of the following two ways:

- The CA reissues a new certificate based on the original key pair (of course, this is not recommended unless there is an all-around confidence in the strength of the original key pair).
- A fresh key pair is generated, and the CA issues a new certificate based on that new key pair.

The key update process itself can be handled in two ways, as follows:

- In the first approach, the end user has to detect that the certificate is about to expire, and request the CA to issue a new one.
- In the other approach, the expiry date of the certificate is automatically checked every time it is used, and as soon as it is about to expire, its renewal request is sent to the CA. For this, special systems need to be in place.

5.3.4 Key Archival

The CA must plan for and maintain the history of the certificates and the keys of its users. For instance, suppose that someone approaches the CA of Alice, requesting the CA to make Alice's digital certificate available, as was used three years back to sign a legal document for verification purposes. If the CA has not archived the certificates, how can the CA provide this information? This can cause serious legal problems. Therefore, key archival is a very significant aspect of any PKI solution.

■ 5.4 THE PKIX MODEL ■

As we know, the X.509 standard defines the digital-certificate structure, format and fields. It also specifies the procedure for distributing the public keys. In order to extend such standards and make them universal, the Internet Engineering Task Force (IETF) formed the **Public Key Infrastructure X.509 (PKIX)** working group. This extends the basic philosophy of the X.509 standard, and specifies how the digital certificates can be deployed in the world of the Internet. Additionally, other PKI models have been defined for use by applications in various domains. For example, the ANSI ASC X9F standards are used by financial institutions. We will restrict our discussion to an overview of the PKIX model.

5.4.1 PKIX Services

PKIX identifies the primary goals of a PKI infrastructure in the form of the following broad level services:

- (a) Registration** It is the process where an end-entity (subject) makes itself known to a CA. Usually, this is via an RA.
- (b) Initialization** This deals with the basic problems, such as the methodology of verifying that the end-entity is talking to the right CA. We have seen how this can be tackled.
- (c) Certification** In this step, the CA creates a digital certificate for the end-entity and returns it to the end-entity, maintains a copy for its own records, and also copies it in public directories, if required.
- (d) Key-Pair Recovery** Keys used for encryption may be required to be recovered at a later date for decrypting some old documents. Key archival and recovery services can be provided by a CA or by an independent key-recovery system.
- (e) Key Generation** PKIX specifies that the end-entity should be able to generate private-and public-key pairs, or the CA/RA should be able to do this for the end-entity (and then distribute these keys securely to the end-entity).

(f) Key Update This allows a smooth transition from one expiring key pair to a fresh one, by the automatic renewal of digital certificates. However, there is a provision for manual digital certificate renewal request and response.

(g) Cross-certification Helps in establishing trust models, so that end-entities that are certified by different CAs can cross-verify each other.

(h) Revocation PKIX provides support for the checking of the certificate status in two modes: online (using OCSP) or offline (using CRL).

We have discussed these services earlier.

5.4.2 PKIX Architectural Model

PKIX has developed comprehensive documents that describe five areas of its architectural model. These classifications allow refinement of the basic X.509 standard description. These areas are the following:

(a) X.509 V3 Certificate and V2 Certificate Revocation List Profiles As we know, the X.509 standard allows the use of various options while describing the extensions of a digital certificate. PKIX has grouped all the options that are deemed fit for Internet users. It calls this group of options as the profile of the Internet users. This profile is described in RFC2459, and specifies which attributes *must/may/may not be* supported. Appropriate value ranges for the values used in each extension category are also provided. For instance, the basic X.509 standard does not specify the instruction codes when a certificate is suspended (put on hold)—PKIX defines them.

(b) Operational Protocols These define the underlying protocols that provide the transport mechanism for delivering certificates, CRLs and other management and status information to a PKI user. Since each of these requirements demands a different way of service, how to use HTTP, LDAP, FTP, X.500, etc., are defined for this purpose.

(c) Management Protocols These protocols enable exchange of information between the various PKI entities (e.g. how to carry registration requests, revocation status or cross-certification requests and responses). The management protocols specify the structure of the messages that float between the entities, and they also specify what details are required to process these messages. Examples of management protocols include the **Certificate Management Protocol (CMP)** for requesting a certificate.

(d) Policy Outlines PKIX defines the outlines for **Certificate Policies (CP)** and **Certificate Practice Statements (CPS)** in RFC2527. These define the policies for the creation of a document such as a CP, which determines what considerations are important when choosing a type of certificate for a particular application domain.

(e) Timestamp and Data Certification Services **Timestamping service** is provided by a trusted third party called **Time Stamp Authority**. The purpose of this service is to sign a message to guarantee that it existed prior to a specific date and time. This is helpful in dealing with non-repudiation claims. The **Data Certification Service (DCS)** is a trusted third-party service, which verifies the correctness of the data that it receives. This is similar to the notary service in real life, where, for instance, one can use it for getting one's property certified.

■ 5.5 PUBLIC KEY CRYPTOGRAPHY STANDARDS (PKCS) ■

5.5.1 Introduction

We have mentioned the **Public Key Cryptography Standard (PKCS)** without really explaining them. Let us now quickly review what they mean and what they stand for.

The PKCS model was initially developed by RSA Laboratories with help from representatives of the government, industry, and academia. The main purpose of PKCS is to standardize Public Key Infrastructure (PKI). The standardization is in many respects, such as formatting, algorithms and APIs. This would help organizations develop and implement inter-operable PKI solutions, rather than everyone choosing their own *standard*.

Figure 5.39 summarizes the PKCS standards. We have already discussed some of these standards, and we shall discuss a few more, later.

Let us discuss some of the important PKCS standards now. We have already discussed many ideas in PKCS standards earlier, and therefore, would not repeat them here. Instead, we shall simply cover the areas that we have not touched so far.

5.5.2 PKCS#5—Password Based Encryption (PBE) Standard

Password-Based Encryption (PBE) is a solution for keeping the symmetric session keys safe. This technique ensures that the symmetric keys are protected from an unauthorized access. The PBE method uses a password-based technique for encrypting a session key. This works as shown in Fig. 5.40.

As the figure shows, we first encrypt the plain-text message with the symmetric key, and then encrypt the symmetric key with a **Key Encryption Key (KEK)**. This protects the symmetric key from an unauthorized access. This is conceptually very similar to the concept of digital envelopes. Anyone wishing to access the symmetric key must have an access to the KEK. Obviously, the next question then is, where do we store the KEK, and how do we protect it?

To protect KEK, the best way is to never store it anywhere! This will ensure that no one will have an access to it. However, this also means that we cannot use KEK to decrypt the symmetric key as and when required! Consequently, the approach used in PBE is to *generate* it on demand, *use* it for encrypting/decrypting the symmetric key, and then *discard* it immediately. This means that we must have the capability of generating the KEK as and when required. For this very purpose, a password is used. The password is the input to a key-generation process (usually a message digest algorithm), the output of which is the KEK. This is shown in Fig. 5.41.

The drawback of this approach is that an attacker can launch a **dictionary attack** against this scheme. This means that the attacker can simply pre-compute all the possible English words and their permutations-combinations, store them in a file, and try each word from that file as a password. Since many times, passwords are simple English words, this attack may succeed, and the attacker could have access to the KEK. To prevent such an attack, apart from the password, two additional pieces of information are used in the key-generation mechanism. They are: **salt** and **iteration count**.

Salt is simply a bit string, which is combined with the password to produce the KEK. *Iteration count* specifies the number of operations that must be performed on the combination of the password and the salt to generate the KEK. This is shown in Fig. 5.42.

Standard	Purpose	Details
PKCS#1	RSA Encryption Standard	Defines the basic formatting rules for RSA public key functions, more specifically the digital signature. It defines how digital signatures should be calculated, including the structure of the data to be signed as well as the format of the signature. The standard also defines the syntax for RSA private and public keys.
PKCS#2	RSA Encryption Standard for Message Digests	This standard outlined the message-digest calculation. However, this is now merged with PKCS#1, and does not have an independent existence.
PKCS#3	Diffie–Hellman Key Agreement Standard	Defines a mechanism to implement Diffie–Hellman Key Agreement protocol.
PKCS#4	NA	Merged with PKCS#1.
PKCS#5	Password Based Encryption (PBE)	Describes a method for encrypting an octet string with a symmetric key. The symmetric key is derived from a password.
PKCS#6	Extended Certificate Syntax Standard	Defines syntax for extending the basic attributes of an X.509 digital certificate.
PKCS#7	Cryptographic Message Syntax Standard	Specifies a format/syntax for data that is the result of a cryptographic operation. Examples of this are digital signatures and digital envelopes. This standard provides many formatting options, such as messages that are only signed, only enveloped, signed and enveloped, etc.
PKCS#8	Private Key Information Standard	Describes the syntax for private-key information (i.e. the algorithm and attributes used to generate the private key).
PKCS#9	Selected Attribute Types	Defines selected attribute types for use in PKCS#6 extended certificates (e.g. email address, unstructured name and address).
PKCS#10	Certificate Request Syntax Standard	Defines syntax for requesting for digital certificates. A certificate request contains a Distinguished Name (DN) and public key.
PKCS#11	Cryptographic Token Interface Standard	This standard, also called Cryptoki , specifies an API for the single-user devices that contain cryptographic information, such as private keys and digital certificates. These devices are also capable of performing cryptographic functions. Smart cards are examples of such devices.
PKCS#12	Personal Information Exchange Syntax Standard	Defines syntax for personal identity information, such as private keys, digital certificates, etc. This allows the users to transfer their certificates and other personal identity information from one device to another, using a standard mechanism.
PKCS#13	Elliptic Curve Cryptography Standard	Currently under development, this standard deals with a new cryptographic mechanism called <i>Elliptic Curve Cryptography</i> .
PKCS#14	Pseudo-Random Number Generation Standard	Currently under development, this standard will specify the requirements and process of random number generation. Since random numbers are extensively used in cryptography, standardizing their generation is important.
PKCS#15	Cryptographic Token Information syntax standard	Defines a standard for cryptographic tokens, so that they can interoperate.

Fig. 5.39 PKCS standards

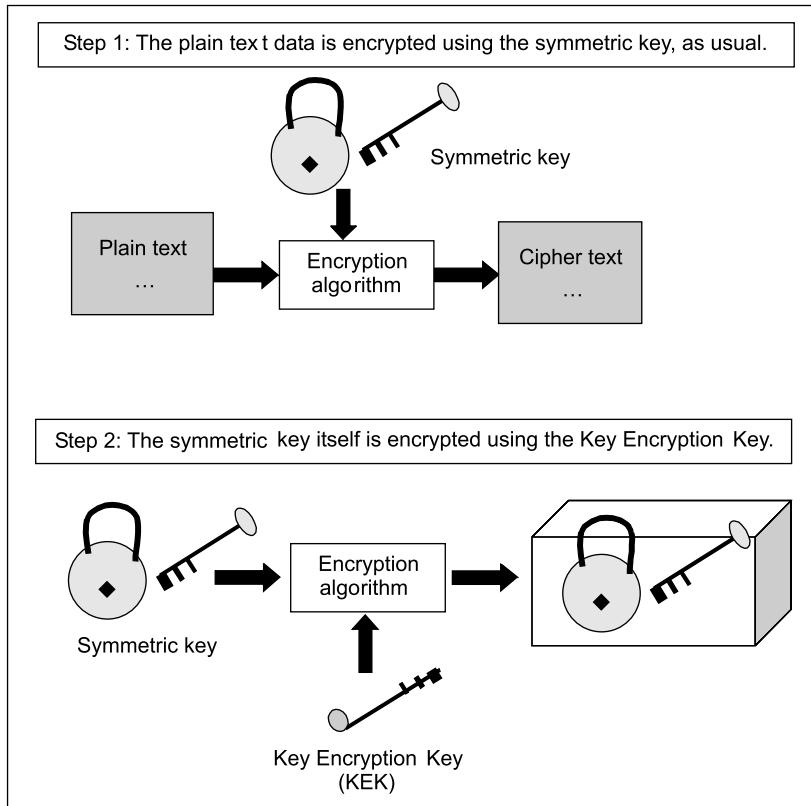


Fig. 5.40 Password Based Encryption (PBE) concept

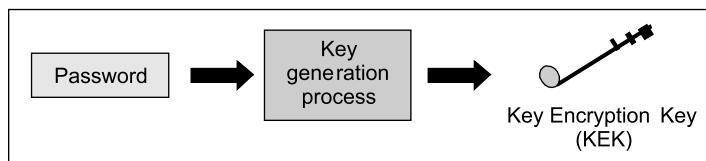


Fig. 5.41 KEK generation using a password

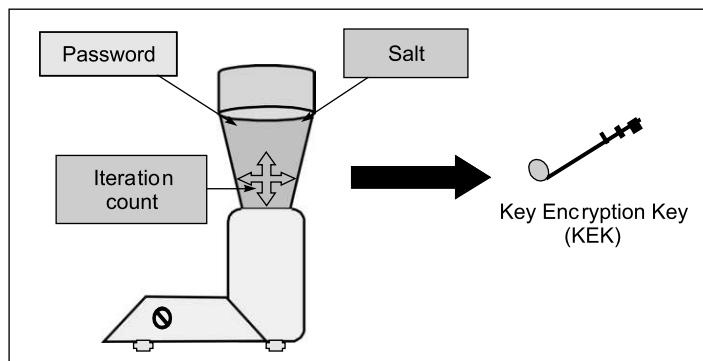


Fig. 5.42 KEK generation using a password, salt and iteration count

Interestingly, the salt and iteration count need not be secret—actually, they *must not* be secret (i.e. must not be stored in an encrypted format). If they are secret, how can one access them? But if they are not secret, an attacker can also access them! An attacker can use the well-known salt and the iteration count along with a dictionary attack on the password to try and generate the KEK. This is highly possible. However, the biggest difference between this attack and the previous attack will be that the attacker would not be able to launch a dictionary attack based on pre-computed values alone, as was possible previously. Now, the attacker has to first combine each word from the dictionary with the salt, and perform the key-generation process for *iteration count* number of times. This makes the task of the attacker quite difficult.

For example, suppose one of the possible passwords in the dictionary of the attacker is *year*. In the absence of salt, the attacker had to try out only the word *year* as the password. But now, assuming a 3-digit salt, the attacker has to try out all the passwords in the possible range, i.e. *year000*, *year001*, *year002*, ..., *year999*, etc.

In PBE, the KEK is used for encrypting the symmetric key. The KEK is never stored anywhere. It is generated, used, and discarded immediately. Three inputs (password, salt and iteration count) are used to generate the KEK. The password must be kept secret, whereas the salt and the iteration count need not be secret.

5.5.3 PKCS#8—Private Key Information Syntax Standard

This standard describes the syntax for storing the private key of a user securely. This standard also describes how to store a few other attributes along with the private key, which we shall ignore here. The standard also describes the syntax for encrypting private keys, so that they cannot be attacked. A Password Based Encryption algorithm (using PKCS #5) could be used to encrypt the private-key information.

PKCS#8 can be considered the predecessor to PKCS#12 (discussed later).

5.5.4 PKCS#10—Certificate Request Syntax Standard

We have already seen how a user can create and send a certificate request. PKCS#10 describes the syntax for certification requests. A certification request consists of a distinguished name, a public key, and optionally a set of attributes, signed together by the entity requesting certification. Certification requests are sent to a certification authority, which transforms the request to an X.509 public-key certificate, or a PKCS#6 extended certificate.

A certification request consists of three aspects: certification-request information, a signature-algorithm identifier, and a digital signature on the certification-request information. The certification-request information consists of the entity's distinguished name, the entity's public key, and a set of attributes providing other information about the entity. The entity requesting for the certificate then signs these values together with his/her private key and sends the certificate-request information, the signed request and the signature algorithm used to the CA. The CA verifies the signature and other aspects regarding the entity, and if they are found alright, issues a certificate.

5.5.5 PKCS#11—Cryptographic Token Interface Standard

This standard specifies the operations performed using a hardware token, such as a smart card. A smart card is similar to a credit card or an ATM card in terms of its look and feel. However, a smart card is *smart* in the sense that it has its own cryptographic processor and memory on the card itself. In simple terms, a smart card is a small microprocessor with its own memory, inside a plastic cover. The ISO7816 standard specifies the shape, thickness, contact positions, electrical signals, and protocols of smart cards.

Cryptographic operations such as key generation, encryption or digital signatures are performed directly inside the card itself. The user's digital certificate and private key are also stored inside the card, and the private key is never exposed to an outside application. It cannot also be copied from the smart card to another location. The small size of the card makes it transportable. This means that users can carry their digital certificates and private keys along with them, without requiring a floppy disk.

Just like ATM cards need ATM machines, smart cards need smart-card readers. A smart-card reader is a small device, which gives power to a smart card, and enables the communication between a smart card and the outside world of applications built on top of it. The reader provides the electrical signals to set up, power on and work with a smart card. These days, desktop computers and laptops come with built-in *smart-card readers*, so that external *smart-card readers* are not required. Many mobile phones also have this facility now.

5.5.6 PKCS#12—Personal Information Exchange Syntax

The PKCS#12 standard was developed to solve the problem of certificate and private-key storage and transfer. More specifically, how does one store/transfer one's certificate and private key securely, without worrying about their tampering? This is especially true in the case of Web-browser users. PKCS#12 can be considered as an increment to PKCS#8.

Prior to the emergence of PKCS#12, Microsoft had developed a format called PFX (personal file exchange), which they did not implement initially, but Netscape did. The PFX format was considered the mechanism for personal information (such as private key, certificate, etc.) storage and exchange. After the emergence of PKCS#12, both Microsoft Internet Explorer and Netscape Navigator browser allow only the import of PFX files (for compatibility with older files), but not their export. However, both browsers allow the import and export of PKCS#12 files (which have the extension of .P12). To add to the confusion, Internet Explorer files are internally PKCS#12, and yet they have an extension of PFX!

5.5.7 PKCS#14—Psuedo-Random Number Generation Standard

As we know, random numbers are extremely crucial in cryptography. Consequently, this PKCS standard defines the requirements for generating random numbers. For this, we must first understand what is random. A series of numbers is random if given a number n in that series of random numbers; we cannot predict what the $n+1^{\text{th}}$ number is, in that series.

A **Random Number Generator** (often abbreviated as **RNG**) is a device that is very specifically designed to generate a series of numbers or symbols that do not exhibit any specific pattern. In other words, they appear to be quite random. As we shall discuss below, computers can also attempt to generate random numbers. However, it is seen that they are often not good enough at that. Other than com-

puting, methods for generating random values have been around since quite ancient times. Examples include dice, coin flipping, the shuffling of playing cards, etc.

There are two main techniques used for generating random numbers.

- The first technique measures some physical property that is expected to be random. The method then compensates for possible biases that exist in the measurement process.
- The second technique uses computational power. Here, we produce long sequences of random results, which turn out to be not so random, as explained in the following sections.

It is said that a random number generator based purely on deterministic computational technique cannot really be considered as a perfect random number generator. This is because its output is predictable. Distinguishing between true and seemingly true random numbers is not easy.

Most computer programming languages provide support for random number generators in the form of library functions. They are usually so designed that they can provide a random byte, or a floating-point number uniformly distributed between the range of 0 and 1. These library functions are often found to have poor statistical properties and some will repeat patterns after a few cycles. They are usually initialized using a computer's clock as the *seed*. These functions may provide enough randomness for certain simple tasks (e.g. computer-based games), but they are not recommended in situations that demand high-quality randomness. Examples of these situations are cryptographic applications, statistical applications or numerical applications. Hence, specialized random-number generators are also available on a majority of operating systems.

We might feel that computers can generate random numbers. In fact, many programming languages provide facilities to generate random numbers. However, this is not quite correct. Random numbers generated by computers are not truly random—over a period of time, we can predict them. This is simply because computers are rule-based machines, which have a finite range for generating (*the so called*) random numbers. Therefore, we must make computers generate random numbers by using some external means. This process is called **psuedo-random number** generation.

Actually, there are three ways to generate psuedo-random numbers using computers, as follows:

(a) Monitor Hardware that Generates Random Data This is the best but costliest approach of generating random numbers using computers. The generator is usually an electronic circuit, which is sensitive to some random physical event, such as diode noise or atmospheric changes. This unpredictable sequence of events is transformed into a random number.

(b) Collect Random Data from User Interaction In this approach, user interaction such as keyboard key presses or mouse movements are used as random inputs to the generator.

(c) Collect Data from Inside the Computer This approach involves the collection of data from inside the computer, which is hard to predict. This data can be the system clock, the number of files on the disk, the number of disk blocks, the amount of free and unused memory, etc.

Note that choosing an appropriate mechanism is very important. Netscape was using the system clock and some other attributes to generate random numbers, which form the basis of the SSL protocol. In 1995, some graduate students made an attempt to reverse-engineer the SSL algorithm, and in fact, succeeded in breaking into it. This was because of the predictability of generating the random numbers used in SSL. Consequently, the SSL protocol algorithm was enhanced to incorporate more random and unpredictable inputs to the random number generation process.

5.5.8 PKCS#15 Cryptographic Token Information Syntax Standard

As we shall discuss later, smart cards can be used to securely store personal information about users, such as their certificates and private keys. This prevents attacks on the private keys, since they are protected by hardware as well as software. However, the biggest problem with smart cards today is the lack of interoperability. Smart-card vendors provide their own interface (API), which is not interoperable with the interface of other vendors. Therefore, one cannot buy a smart card from *X* and use the software from *Y*. This is a big problem. As we have mentioned, PKCS#11 attempts to resolve this problem with the help of a uniform interface to which all smart-card vendors are expected to comply, in the coming years.

The other problem with smart cards is in the area of information representation. More specifically, information pieces such as user certificates, private keys, etc., are stored on smart cards differently for different vendors. These differences are in the form of data structures, file organizations, directory hierarchies, etc. PKCS#15 specifies a uniform (standardized) token format to resolve these incompatibilities. If and when the smart-card and other hardware token vendors comply with this, smart-card applications should become interoperable even in terms of data access.

■ 5.6 XML, PKI AND SECURITY ■

Although the technology of PKI is quite promising and exciting, there are a few hurdles in implementing it. The chief of these hurdles is the lack of operability among vendor solutions. For instance, it is not very easy to integrate the PKI offering of Vendor *X* with that of Vendor *Y* easily.

The EXtensible Markup Language (XML) is at the centerstage of the modern world of technology. XML forms the backbone of the upcoming technologies, such as Web services. Almost every aspect of Internet programming is concerned with XML. We request the reader to study XML through another resource, as it is not the purpose of the current text. However, we shall discuss the key elements of XML security and its relation to PKI. The overall technology related to XML and security can be summarized as shown in Fig. 5.43.

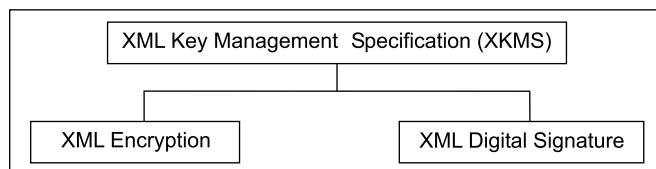


Fig. 5.43 XML and security

We shall examine these aspects of XML security now.

5.6.1 XML Encryption

The most interesting part about **XML encryption** is that we can encrypt an entire document, or its selected portions. This is very difficult to achieve in the non-XML world. We can encrypt one or all of the following portions of an XML document:

- The entire XML document
- An element and all its sub-elements
- The content portion of an XML document
- A reference to a resource outside of an XML document

The steps involved in XML encryption are quite simple, and are as follows:

1. Select the XML to be encrypted (one of the items listed earlier, i.e. all or part of an XML document).
2. Convert the data to be encrypted in a canonical form (optional).
3. Encrypt the result using public key encryption.
4. Send the encrypted XML document to the intended recipient.

Figure 5.44 shows a sample XML document, containing the details of a credit card user.

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://mybank.org'>
    <Name> John Smith </Name>
    <CreditCard Limit='10000' Currency='USD'>
        <Number> 1617 1718 0181 9910 </Number>
        <Issuer> Master </Issuer>
        <Expires> 05/05 </Expires>
    </CreditCard>
</PaymentInfo>
```

Fig. 5.44 Sample XML document showing a user's credit-card details

We shall not describe the various details of this XML document, and would simply remark that it contains the credit-card details, such as the user's name, credit limit, currency, card number, issuer name, and expiry details. Let us assume that we want to encrypt this. When we perform an XML encryption, a standard tag called *EncryptedData* comes into picture. As we have mentioned before, we can choose to encrypt selected portions of the XML document, or we can encrypt it as a whole. For illustration purposes, we shall see what happens when we encrypt only the actual credit card-details (such as its number, issuer, and expiry details). The result is shown in Fig. 5.45. We can see that the encrypted text is embedded inside the tag *CipherData*. This is another standard tag in XML encryption.

As we can see, the credit-card details are now encrypted, and therefore, cannot be read/changed. The fact that we have encrypted the contents of the XML document is signified by using the *xmlenc#Content* value. If we had encrypted the full *CreditCard* element, this would have been changed to *xmlenc#Element*.

5.6.2 XML Digital Signature

As we can see, a digital signature is calculated over the complete message. It cannot be calculated only for specific portions of a message. The simple reason for this is that the first step in a digital signature creation is the calculation of the message digest over the whole message. Many practical situations demand that users be able to sign only specific portions of a message. For instance, in a purchase request, the purchase manager may want to authorize only the quantity portion, whereas the accounting manag-

```

<?xml version='1.0'?>
<PaymentInfo xmlns='http://mybank.org'>
    <Name> John Smith </Name>
    <CreditCard Limit='10000' Currency='USD'>
        <EncryptedData Type =
            http://www.w3.org/2001/04/xmlenc#Content'
            xmlns='http://www.w3.org/2001/04/xmlenc#'>
            <CipherData>
                <CipherValue> D7T60UB67 </CipherValue>
            </CipherData>
        </EncryptedData >
        </CreditCard>
    </PaymentInfo>

```

Fig. 5.45 Encrypted credit card details

er may want to sign only the rate portion. In such cases, **XML digital signatures** can be used. They are also useful from the perspective of new technologies such as XML digital signature. This technology treats a message or a document as consisting of many elements, and provides for the signing of one or more such elements. This makes the signature process flexible and more practical in nature.

The XML digital-signature specification defines a number of XML elements, which describe the characteristics of an XML signature. These are tabulated in Fig. 5.46.

Element	Description
SignedInfo	Contains the signature itself (i.e. the output of the signing process).
Canonicalization Method	Specifies the algorithm used to canonicalize the SignedInfo element, before it is digested as a part of the signature creation.
Signature Method	Specifies the algorithm used to transform the canonicalized SignedInfo element into the SignatureValue element. This is a combination of a message-digest algorithm and key-dependent algorithm.
Reference	Includes the mechanism used for calculating the message digest and the resulting digest value over the original data.
KeyInfo	Indicates a key that can be used to validate the digital signature. This can consist of digital certificates, key names, key agreement algorithms used, etc.
Transforms	Specifies the operations performed before calculating the digest, such as compression, encoding, etc.
Digest Method	Specifies the algorithm used to calculate the message digest.
Digest Value	Contains the message digest of the original message.

Fig. 5.46 Elements in XML digital signature process

The steps in performing XML digital signatures are as follows.

1. Create a SignedInfo element with SignatureMethod, CanonicalizationMethod and References.
2. Canonicalize the XML document.
3. Calculate the SignatureValue, depending on the algorithms specified in the SignedInfo element.

- Create the digital signature (i.e. Signature element), which also includes the SignedInfo, Key-Info, and SignatureValue elements.

A simplistic example of an XML digital signature is shown in Fig. 5.47. We shall also explain the important aspects of the signature.

```
<Signature>
  <SignedInfo>
    <SignatureMethod Algorithm="xmlsig#rsa-sha1"/>
  </SignedInfo>
  <SignatureValue>
    0WjB5MQswCQYDVQQGEwJJTjEOMAwGA1UEChMFaWZsZXgxDDAKBgNV
    WMBQGCgmsJomT8ixkAQETBnNlcnZlcjETMBEGA1UEAxMKZ2lyaNlcnZlcjEf
    GCSqGSIB3DQEJARYQc2VydmyQGlmbGV4LmNvbTCBnzANBqkqhkiG9w0B
    BjQAwgYkCgYEArislLRowIrIVxu/Mie8q0rUCQ5GtqMBWeJtuJM0vn2Qk5XaWc
    ylnJ/zc90v7qsx33X/sw5aRjh1ApOvParQhK9PAyPhCcCIUEovUYnxFmu8YE9U
  </SignatureValue>
</Signature>
```

Fig. 5.47 XML Digital signature example (simplified)

Let us discuss the contents of the digital signature in brief.

- **<Signature> ... </Signature>** - This block identifies the start and end of the XML digital signature.
- **<SignedInfo> ... </SignedInfo>** - This block specifies the algorithm used: firstly for calculating the message digest (which is SHA-1, in this case) and then for preparing the XML digital signature. (which is RSA, in this case).
- **<SignatureValue> ... </SignatureValue>** - This block contains the actual XML digital signature.

XML digital signatures can be classified into three types: **enveloped**, **enveloping**, and **detached**, as shown in Fig. 5.48.

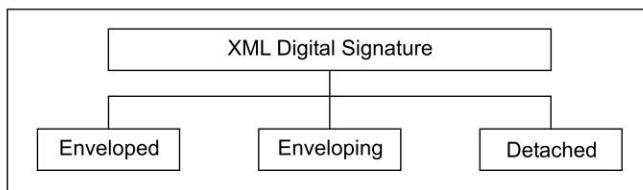


Fig. 5.48 Classification of digital signatures

The difference between these types of XML digital signatures is easy to understand.

- In the *enveloped* XML digital signatures, the signature is inside the original document (which is being digitally signed).
- In the *enveloping* XML digital signatures, the original document is inside the signature.
- A *detached* digital signature has no enveloping concept at all, it is separate from the original document.

This idea is shown in Fig. 5.49.

```

Enveloped Signature

<Original_document>
    <Signature> ... </Signature>
</Original_document>

Enveloping Signature

<Signature>
    <Original_document>
        </Original_document>
    </Signature>

Detached Signature

<Original_document>
</Original_document>
<Signature>
</Signature>

```

Fig. 5.49 XML digital signature types illustrated

5.6.3 XML Key Management Specification (XKMS)

The **XML Key Management Specification (XKMS)** is an initiative of the World Wide Web Consortium (W3C), which aims to delegate the trust-related decisions in an XML encryption/signature process to one or more specified trust processors. This allows businesses to manage XML encryption and digital signature to be managed quite easily. This also resolves the issue of differences between different PKI vendor implementations. XKMS was jointly proposed by Microsoft, VeriSign and WebMethods. It is backed by many other strong parties, including Baltimore, Entrust, HP, IBM, Iona, RSA, etc.

XKMS specifies protocols for distributing and registering public keys, and works very well with XML encryption and XML signatures. XKMS consists of two parts, as shown in Fig. 5.50.

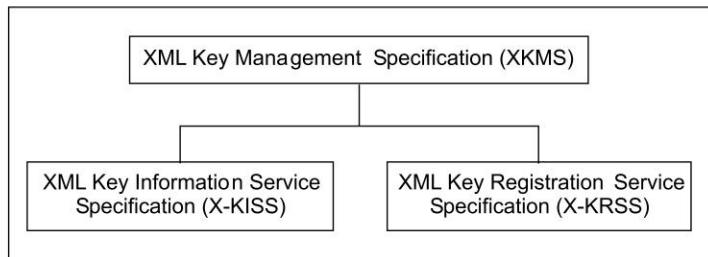


Fig. 5.50 Classification of XKMS

- The **XML Key Information Service Specification (X-KISS)** specifies a protocol for a trust service, which resolves the public-key information contained in documents that conform to the XML signature standard. This protocol allows a client of such a service to delegate some or all of the tasks needed to process an XML signature element. The underlying PKI can be based on different specifications, such as X.509 or Pretty Good Privacy (PGP), and yet X-KISS shields the application from these differences.

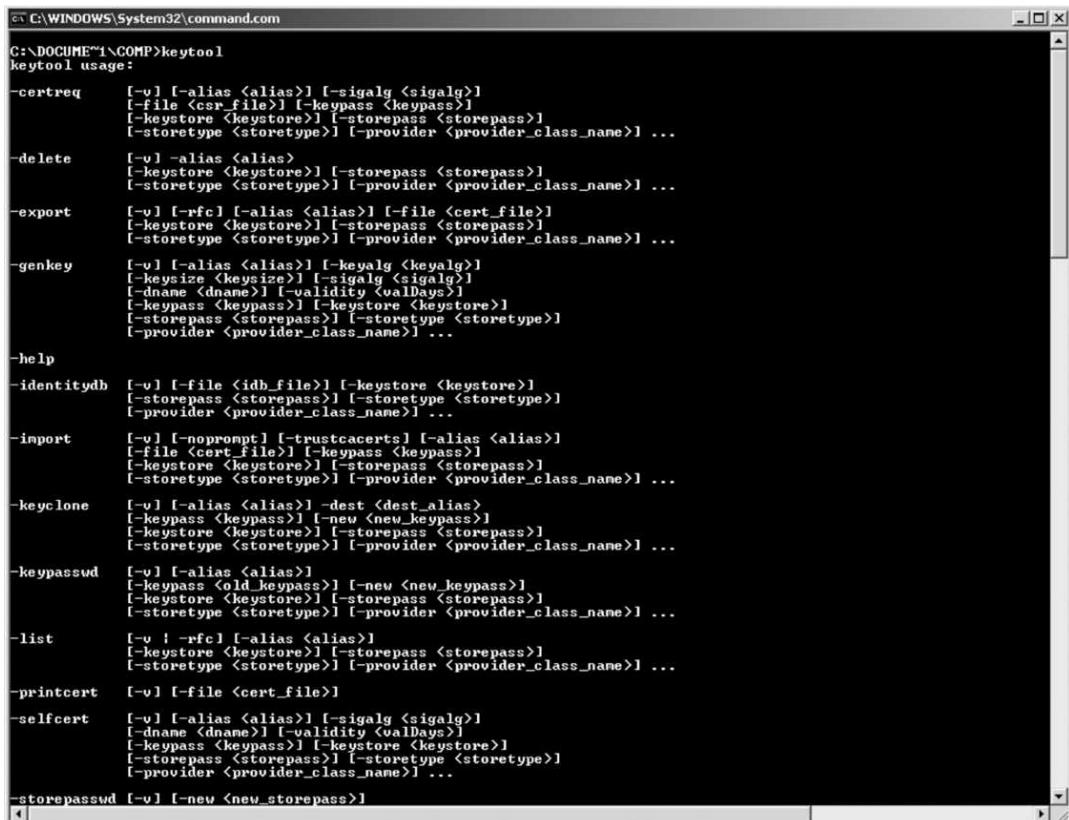
- The **XML Key Registration Service Specification (X-KRSS)** defines a protocol for a Web service, that accepts the registration of public-key information. Once registered, the public key can be used in relation with other Web services, including X-KISS. This protocol can also be used to later retrieve the private key. The protocol has provisions for authentication of the applicant and proof of possession of the private key.

Creating Digital Certificates using Java

The Java programming environment provides two very useful utilities called **keystore** and **keytool**.

- Keytool* is a command-line utility, which allows us to create keys, certificates, and exporting/importing them the way we want. This section describes how to create and view these certificates. These certificates can also be used in any application programs that we write to perform cryptographic operations, such as encryption, message digests, authentication, and digital signatures.
- Keystore* is a collection of the keys and certificates that we create using *keytool*. It can hold trusted certificates and keys.

The simplest way to use these utilities is to go to the command prompt, and type *keytool*. We get to see all the options available with *keytool*, as shown in Fig. 5.51.



```
C:\WINDOWS\System32\command.com
C:\DOCUMENTS\COMP>keytool
keytool usage:

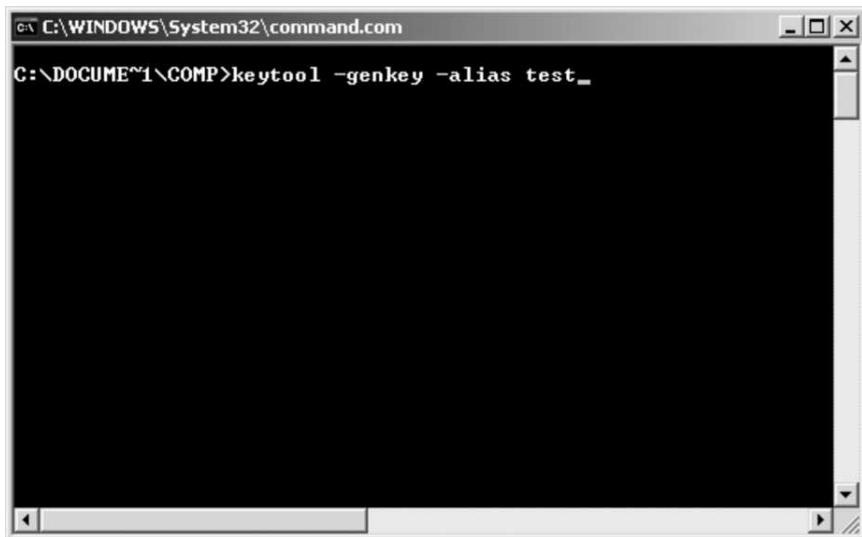
-crtreq      [-v] [-alias <alias>] [-sigalg <sigalg>]
              [-file <csr_file>] [-keypass <keypass>]
              [-keystore <keystore>] [-storepass <storepass>]
              [-storetype <storetype>] [-provider <provider_class_name>] ...
-delete     [-v] -alias <alias>
              [-keystore <keystore>] [-storepass <storepass>]
              [-storetype <storetype>] [-provider <provider_class_name>] ...
-export      [-v] [-rfc] [-alias <alias>] [-file <cert_file>]
              [-keystore <keystore>] [-storepass <storepass>]
              [-storetype <storetype>] [-provider <provider_class_name>] ...
-genkey     [-v] [-alias <alias>] [-keyalg <keyalg>]
              [-keysize <keysize>] [-sigalg <sigalg>]
              [-dname <dname>] [-validity <valDays>]
              [-keypass <keypass>] [-keystore <keystore>]
              [-storepass <storepass>] [-storetype <storetype>]
              [-provider <provider_class_name>] ...
-help       ...
-identitydb [-v] [-file <idb_file>] [-keystore <keystore>]
              [-storepass <storepass>] [-storetype <storetype>]
              [-provider <provider_class_name>] ...
-import     [-v] [-noprompt] [-trustcacerts] [-alias <alias>]
              [-file <cert_file>] [-keypass <keypass>]
              [-keystore <keystore>] [-storepass <storepass>]
              [-storetype <storetype>] [-provider <provider_class_name>] ...
-keyclone   [-v] [-alias <alias>] -dest <dest_alias>
              [-keypass <keypass>] [-new <new_keypass>]
              [-keystore <keystore>] [-storepass <storepass>]
              [-storetype <storetype>] [-provider <provider_class_name>] ...
-keypasswd  [-v] [-alias <alias>]
              [-keypass <old_keypass>] [-new <new_keypass>]
              [-keystore <keystore>] [-storepass <storepass>]
              [-storetype <storetype>] [-provider <provider_class_name>] ...
-list       [-v] [-rfc] [-alias <alias>]
              [-keystore <keystore>] [-storepass <storepass>]
              [-storetype <storetype>] [-provider <provider_class_name>] ...
-printcert [-v] [-file <cert_file>]
-selfcert   [-v] [-alias <alias>] [-sigalg <sigalg>]
              [-dname <dname>] [-validity <valDays>]
              [-keypass <keypass>] [-keystore <keystore>]
              [-storepass <storepass>] [-storetype <storetype>]
              [-provider <provider_class_name>] ...
-storepasswd [-v] [-new <new_storepass>]
```

Fig. 5.51 Keytool options

We can provide specific options to accomplish the intended tasks. For example, we can supply the following command:

```
keytool -genkey -alias test
```

This command would create a *keystore* which we will refer to with a name (as an alias) *test*. This command is entered on the command prompt, as shown in Fig. 5.52.



```
C:\WINDOWS\System32\command.com
C:\DOCUME~1\COMP>keytool -genkey -alias test
```

Fig. 5.52 Creating a keystore—Step 1

Keytool asks us for a password, as shown in Fig. 5.53.



```
C:\WINDOWS\System32\command.com
C:\DOCUME~1\COMP>keytool -genkey -alias test
Enter keystore password: password
```

Fig. 5.53 Creating a keystore—Step 2

When we enter a password (we have chosen the word *password* as the password here, to keep things simple), *keytool* asks us a number of questions, for which we need to provide answers as shown in Fig. 5.54. At the end, it creates our keystore.

```
C:\>DOCUME~1\COMP>keytool -genkey -alias test
Enter keystore password: password
What is your first and last name?
[Unknown]: Atul Kahate
What is the name of your organizational unit?
[Unknown]: PrimeSourcing
What is the name of your organization?
[Unknown]: i-flex solutions limited
What is the name of your City or Locality?
[Unknown]: Pune
What is the name of your State or Province?
[Unknown]: Maharashtra
What is the two-letter country code for this unit?
[Unknown]: IN
Is CN=Atul Kahate, OU=PrimeSourcing, O=i-flex solutions limited,
[no]: yes

Enter key password for <test>
      (RETURN if same as keystore password):
```

C:\>DOCUME~1\COMP>

Fig. 5.54 Creating a keystore—Step 3

We can list the contents of the keystore as shown in Fig. 5.55.

```
C:\>DOCUME~1\COMP>keytool -v -list
Enter keystore password: password

Keystore type: jks
Keystore provider: SUN

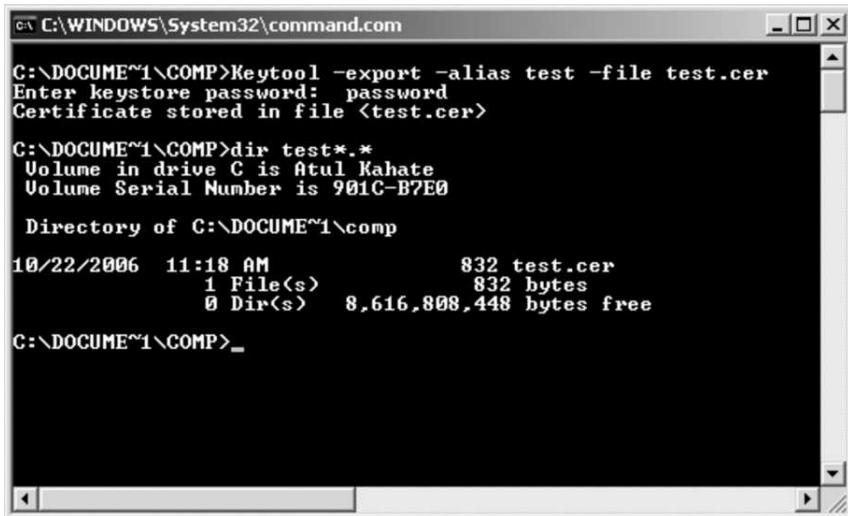
Your keystore contains 1 entry

Alias name: test
Creation date: Oct 22, 2006
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Atul Kahate, OU=PrimeSourcing, O=i-flex solutions limited, L=Pune, ST=Maharashtra, C=IN
Issuer: CN=Atul Kahate, OU=PrimeSourcing, O=i-flex solutions limited, L=Pune, ST=Maharashtra, C=IN
Serial number: 453b05cf
Valid from: Sun Oct 22 11:16:55 IST 2006 until: Sat Jan 20 11:16:55 IST 2007
Certificate fingerprints:
      MD5: 4D:42:42:58:AB:FC:1C:Ea:C7:06:E1:37:F1:13:AB:AD
      SHA1: 19:7F:3E:14:EB:76:84:6F:C5:37:7B:F8:74:C8:28:83:3E:4C:03:50

*****
```

Fig. 5.55 Creating a keystore—Step 4

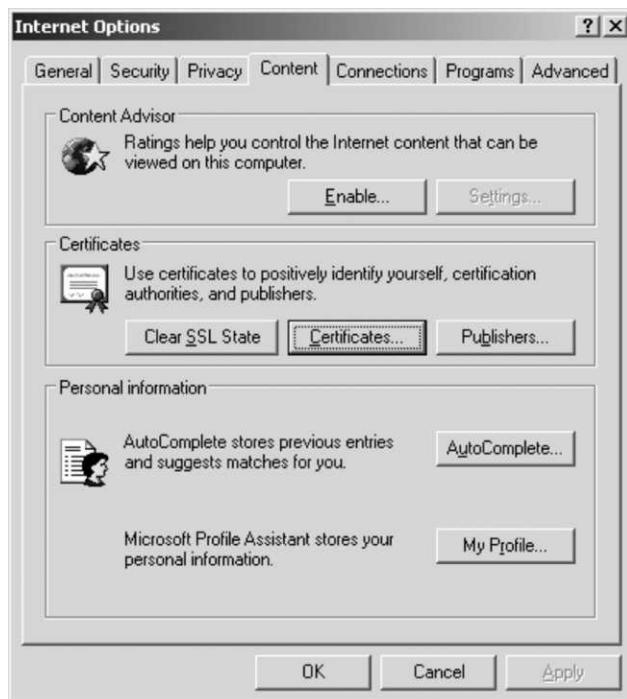
We can now export the contents of the *keystore* into a certificate file (which has an extension of *.cer*). This is shown in Fig. 5.56. Thus, we would have a file named *test.cer* on our disk, containing our digital certificate in X.509 format.



```
C:\WINDOWS\System32\command.com
C:\DOCUME~1\COMP>Keytool -export -alias test -file test.cer
Enter keystore password: password
Certificate stored in file <test.cer>
C:\DOCUME~1\COMP>dir test.*.*
Volume in drive C is Atul Kahate
Volume Serial Number is 901C-B7E0
Directory of C:\DOCUME~1\comp
10/22/2006 11:18 AM           832 test.cer
   1 File(s)      832 bytes
   0 Dir(s)  8,616,808,448 bytes free
C:\DOCUME~1\COMP>_
```

Fig. 5.56 Creating a keystore – Step 5

Now, we can import this certificate file into our browser's list of certificates, so that we can use the certificate in our applications. This is shown step by step in the following diagrams. For this purpose, we need to go to the *Internet options* tab of the Internet Explorer Web browser. For other browsers, we need to access the appropriate option.

**Fig. 5.57** Importing certificate – Step 1

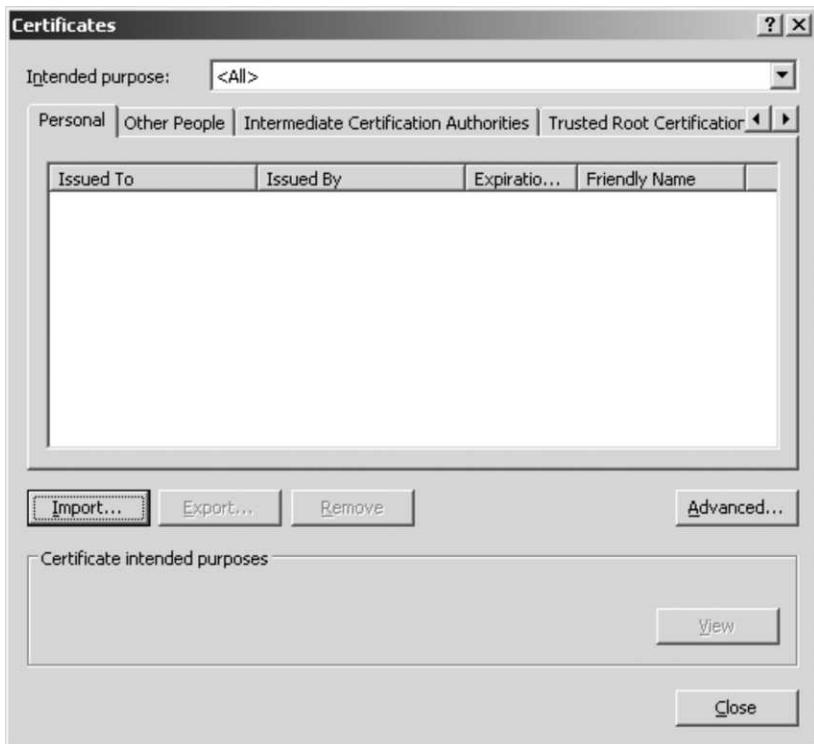


Fig. 5.58 Importing certificate—Step 2



Fig. 5.59 Importing certificate—Step 3

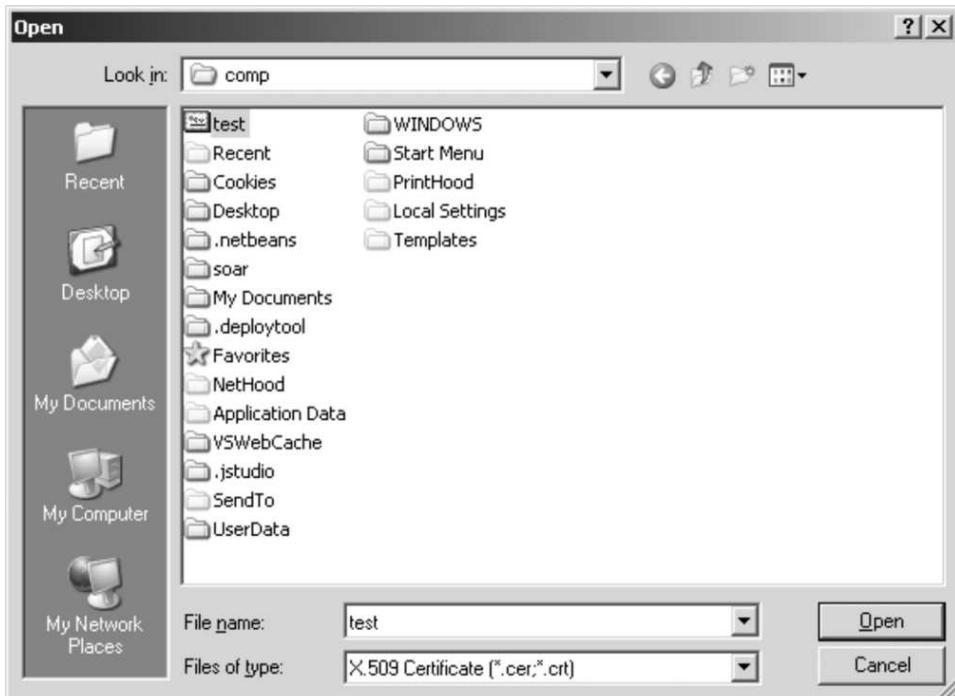


Fig. 5.60 Importing certificate—Step 4



Fig. 5.61 Importing certificate—Step 5



Fig. 5.62 Importing certificate—Step 6



Fig. 5.63 Importing certificate—Step 7



Fig. 5.64 Importing certificate—Step 8

■ CASE STUDY: CROSS SITE SCRIPTING VULNERABILITY (CSSV) ■

Points for Classroom Discussions

1. *What is the purpose of scripting technologies on the Internet?*
2. *What can prevent CSSV attacks?*
3. *What sort of testing can the creators of a Web site perform in order to guard against possible CSSV attacks?*

Cross Site Scripting Vulnerability (CSSV) is a relatively new form of attacks that exploits inadequate validations on the server-side. The term *Cross Server Scripting Vulnerability (CSSV)* is actually not completely correct. However, this term was coined when the problem was not completely understood and has stuck ever since. Cross-site scripting happens when malicious tags and/or scripts attack a Web browser via another site's dynamically generated Web pages. The attacker's target is not a Website, but rather its users (i.e. clients or browsers).

The idea of CSSV is quite simple to understand and is based on exploiting the scripting technologies, such as JavaScript, VBScript or JScript. Let us understand how this works. Consider the following Web page containing a form as shown in the figure in which the user is expected to enter her postal address. Suppose that the URL of the site sending this page is www.test.com and when the user submits this form, it would be processed by a server-side program called `address.asp`.

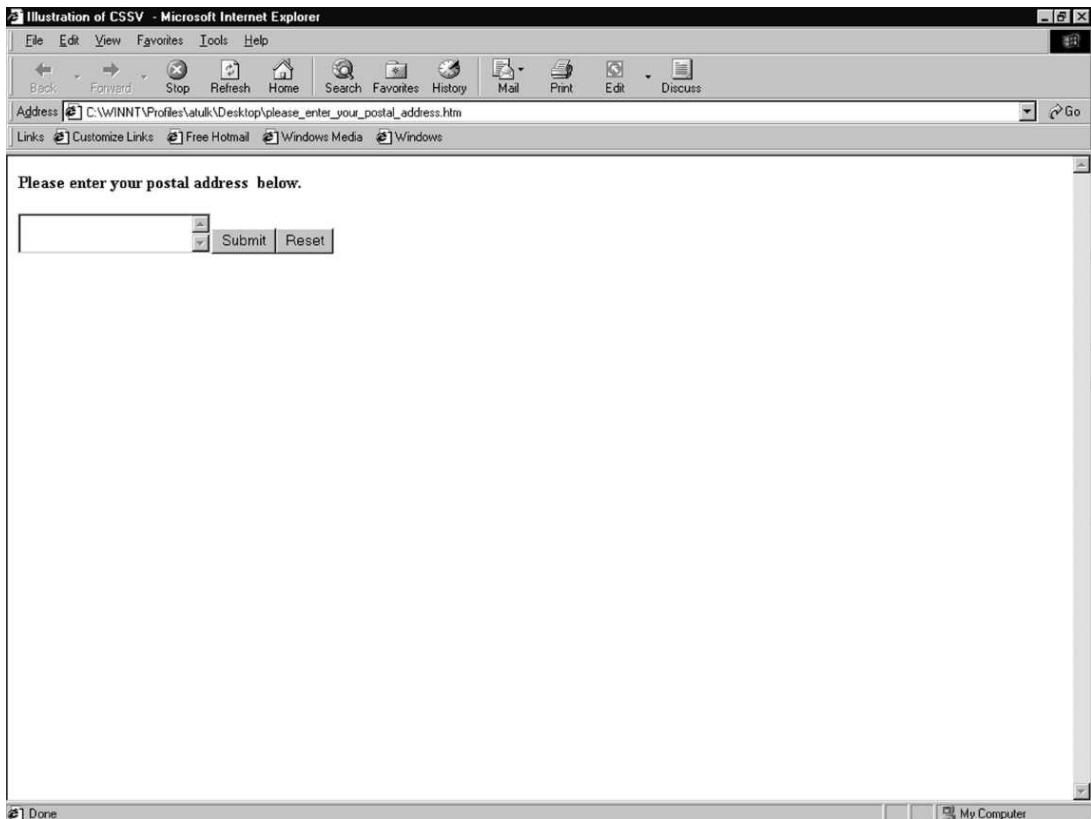
We would typically expect the user to enter the house number, street name, city, postal code and country, etc. However, imagine that the user enters the following weird string, instead:

```
<SCRIPT>Hello World</SCRIPT>
```

As a result, the URL submitted would be something like www.test.com/address.asp?address=<SCRIPT>Hello World </SCRIPT>.

Now suppose that the server-side program `address.asp` does not validate the input sent by the user and simply sends the value of the field `address` to the next Web page. What would this translate to? It would mean that the next Web page would receive the value of `address` as `<SCRIPT>Hello World</SCRIPT>`.

As we know, this would most likely treat the value of the `address` field as a script, which would be executed as if it is written in a scripting language, such as JavaScript etc on the Web browser. Therefore, the user would get to see *Hello World*.



Sample HTML form

Obviously, no serious damage is done. However, extrapolate this possibility to other situations where a user can actually send damaging scripts to the server. This can cause the same or another client to receive a Web page whose contents/look-and-feel are changed. In a more damaging case, the confidential information entered by a user could also be captured and sent to another user and so on. How can this be done?

When a JavaScript program gets downloaded on a browser through a CSSV attack, the JavaScript, in turn, can call up the services of an ActiveX control. An ActiveX control is a small program that gets downloaded from the server to the client and executes on the client. The ActiveX control can write to the disk or read from it and perform many such tasks. Once downloaded to the client via the malicious JavaScript call, the ActiveX control, therefore, can do real damage in this case.

The chief measure to protect against these attacks is to validate all the input fields for tags that look suspicious (e.g., <, >, SCRIPT, APPLET, OBJECT, etc.). The server-side program should not trust a browser-based user to enter data with only good intention. It can very well be malicious, causing damage to other clients.

One can test for CSSV on any Web site, by simply trying to enter scripts or script-like tags in the input areas, such as text boxes.



Summary

- Digital certificates solve the problem of key exchange.
- Digital certificates are required to conduct commercial transactions over the Internet.
- Digital-certificate technology makes it possible to verify the identity of the user or system at the other end, and then it can also be used for cryptographic operations.
- Digital certificates can be compared to a person's driving license or passport.
- A digital certificate is a disk file.
- A digital certificate binds a user with the user's public key.
- The private key of the user is held by the user, and should never leave the user.
- A Certification Authority (CA) can issue digital certificates.
- The top-level CA is called root CA.
- Doing CA's job is quite a responsible and complex task.
- X.509 protocol is used to specify the structure of digital certificates.
- The latest version of X.509 standard is version 3.
- Since a CA can have great load, it can offload some of its tasks to a Registration Authority (RA).
- The CA and the RA can be the same organization, or they can be different.
- Root CA uses self-signed certificates.
- CAs operate in a level of hierarchy.
- CA hierarchy helps reduce the burden on a single CA.
- Cross-certification is needed for different CAs to interoperate with each other.
- The status of a certificate can be validated using protocols such as CRL, OCSP, and SCVP.
- Certificate Revocations List (CRL) is offline check.
- CRL is a file that is updated at pre-defined frequencies.
- To avoid downloading the entire CRL file, delta CRL files can also be used.
- CRL may or may not provide real-time status of certificates. Information in a CRL file can be slightly obsolete.
- OCSP and SCVP are online checks.
- Online Certificate Status Protocol (OCSP) works on a request-response mechanism.
- The party interested in knowing the status of a certificate needs to send a request to an OCSP server, called OCSP responder. The OCSP responder comes back with an answer regarding the status of the certificate.
- Simple Certificate Validation Protocol (SCVP) provides not only the status of the certificate, but many more details.
- There is no guarantee that OCSP or SCVP in turn are implemented as real-time certificate validation protocols, or using CRLs internally.
- Digital certificates can be obtained for general/special purposes.

- A CA has to provide for key management, archival, storage, and retrieval.
- Private key management is important.
- Losing private keys can be very risky.
- PKIX model deals with the issues related to PKI.
- PKCS standards touch upon the various aspects of the PKI technology.
- PKCS standards were initially developed by RSA.
- As an example of a standard, PKCS tells us how to represent an encrypted message, so that it is standardized across all usages.
- XML security is now becoming an important concept.
- Java provides the features of *keystore* and *keytool* to create and work with digital certificates.
- Using the *keytool*, we can create, export, view certificates.



Key Terms and Concepts

- Attribute certificate
- Base CRL
- Certificate Management Protocol (CMP)
- Certificate Signing Request (CSR)
- Certification Authority hierarchy
- Cross-certification
- Dictionary attack
- Keystore (Java)
- Lightweight Directory Access Protocol (LDAP)
- Proof Of Possession (POP)
- Public Key Cryptography Standards (PKCS)
- Registration Authority (RA)
- Self-signed certificate
- X.500
- XML digital signature
- XML Key Information Service Specification (X-KISS)
- XML Key Management Specification (XKMS)
- Authority Revocation List (ARL)
- Certificate directory
- Certificate Revocation List (CRL)
- Certification Authority (CA)
- Chain of trust
- Delta CRL
- Digital certificate
- Keytool (Java)
- Online Certificate Status Protocol (OCSP)
- Psuedo-random number
- Public Key Infrastructure X.509 (PKIX)
- Roaming certificate
- Simple Certificate Validation protocol (SCVP)
- X.509
- XML encryption
- XML Key Registration Service Specification (X-KRSS)



PRACTICE SET

■ Multiple-Choice Questions

1. The final solution to the problem of key exchange is the use of
 - (a) passport
 - (b) digital envelope
 - (c) digital certificate
 - (d) message digest
2. A digital certificate binds a user with
 - (a) the user's private key
 - (b) the user's public key
 - (c) the user's passport
 - (d) the user's driving license
3. The _____ of the user should never appear in a certificate.
 - (a) public key
 - (b) private key
 - (c) organization name
 - (d) name
4. A _____ can issue digital certificates.
 - (a) CA
 - (b) government
 - (c) shopkeeper
 - (d) bank
5. The _____ standard defines the structure of a digital certificate.
 - (a) X.500
 - (b) TCP/IP
 - (c) ASN.1
 - (d) X.509
6. A Registration Authority (RA) _____ issue digital certificates.
 - (a) can
 - (b) may or may not
7. The CA signs a digital certificate with _____.
 - (a) the user's public key
 - (b) the user's private key
 - (c) its own private key
 - (d) its own public key
8. The CA with the highest authority is called _____ CA.
 - (a) root
 - (b) head
 - (c) main
 - (d) chief
9. To solve the problem of trust, the _____ is used.
 - (a) public key
 - (b) self-signed certificate
 - (c) private key
 - (d) digital signature
10. CRL is _____.
 - (a) online
 - (b) online and offline
 - (c) offline
 - (d) not defined
11. OCSP is _____.
 - (a) online
 - (b) online and offline
 - (c) offline
 - (d) not defined

12. A digital certificate.
 - (a) may or may not expire
 - (b) must have an expiry date
 - (c) must not have an expiry date
 - (d) must expire within a year
13. The two additional parameters to Password Based Encryption, other than the password, are _____ and _____.
 - (a) private key, public key
 - (b) private key, salt
 - (c) public key, salt
 - (d) salt, iteration count
14. We trust a digital certificate because it contains _____.
 - (a) owner's public key
 - (b) CA's public key
 - (c) CA's signature
 - (d) owner's signature
15. Requesting for a certificate results into the creation of a _____ file.
 - (a) PKCS#7
 - (b) PKCS#9
 - (c) PKCS#10
 - (d) PKCS#12

■ Exercises

1. What are the typical contents of a digital certificate?
2. What is the role of a CA and a RA?
3. Name the four key steps in the creation of a digital certificate.
4. Discuss any one mechanism used by a RA for checking the user's proof of possession of the private key.
5. What is the idea behind Certification Authority hierarchy?
6. Why is a self-signed certificate needed?
7. Describe how cross-certification is useful.
8. What are the common causes for revoking a digital certificate?
9. What are the broad-level differences between CRL, OCSP and SCVP?
10. Describe the mechanisms of protecting the private key of a user.
11. Discuss Password Based Encryption.
12. Discuss XML security concepts in brief.
13. Why do we trust a digital certificate?
14. Consider a situation: an attacker (*A*) creates a certificate, puts a genuine organization's name (say bank *B*), and the puts the attacker's own public key. You get this certificate from the attacker, without knowing that the attacker is sending it. You think it is from the bank (*B*). How can this be prevented or resolved?
15. In the other situation, the attacker (*A*) changes the bank's genuine certificate (*B*) by replacing the bank's public key in the certificate with his own. How can this be prevented or resolved?

■ Design/Programming Exercises

1. Many programming languages allow the generation of random numbers. However, these numbers are not really random—in fact, they are predictable. Write a C program that generates a series of 10 random numbers. Repeat the same program execution many times and see how the random numbers are repeated (i.e. they are not random).

2. Write a Java program to generate and display a random number between 1 and 6.
3. Write a Java program, which accepts the details of a base-64 encoded digital certificate (i.e. cut-and-paste), parses it and displays its main contents such as issuer name, serial number, subject name, valid from and valid to.
4. Explore the Java classes related to digital certificates. (Hint: Refer to the JCE package).
5. Try to create a digital certificate request (CSR) using the software provided by a Web server.
6. Create a certificate of your own by using the Java keytool.
7. Export the certificate in a Web browser.
8. Use the certificate in a Java application for encryption.
9. Try replacing the public-key portion of a certificate with your own public key. Would that work?
10. Can we tamper with the digital signature made by the CA in a certificate? Why?
11. Can you become an intermediate CA even for experimentation purposes? Why?
12. Suppose that a bank wants to communicate with all its customers using digital certificates. What infrastructure would the bank need, and the customers need? Draw a diagram showing the flow of events.
13. Can we create a certificate programmatically in Java or .NET? Try doing it.
14. What are the important features provided by .NET for certificate-related areas?
15. If we can create a certificate of our own, so can the attacker. Where is the security then?

6



INTERN-SECURITY PROTOCOLS

■ 6.1 INTRODUCTION ■

Internet security has assumed great importance, as it is the world's prime computer network (or network of networks). This chapter examines the various security protocols associated with the Internet. Since Internet technology is vast and encompasses many areas, there are various aspects associated with Internet security. Various security mechanisms exist for specialized Internet services, such as email, electronic commerce and payments, wireless Internet, etc.

The chapter briefly introduces the way the Internet works. This section is for readers who are not familiar with the Internet technology and architecture. The chapter then discusses the world's most famous security protocol, the Secure Socket Layer (SSL), now more widely known in its modified form, the Transport Layer Security (TLS). Another protocol had competed with SSL in the past, but has not succeeded that much. We take a quick look at it, namely SHTTP. We also study a recent protocol, called Time Stamping Protocol (TSP).

Electronic commerce involves payments on the Internet. Security of these payment transactions is extremely crucial for the success of electronic commerce. The chapter discusses the Secure Electronic Transaction (SET) protocol, which is designed for this purpose. 3-D Secure is another recent extension of SET, which is explained next. The chapter then introduces the concept of electronic money, and also examines its types, model and the security issues/mechanisms therein.

Email is perhaps the most popular Internet application used worldwide. Three protocols for email security are quite popular: Privacy Enhanced Mail (PEM), Pretty Good Privacy (PGP), and Secure MIME (S/MIME). We study all of these. The chapter concludes with an overview of wireless Internet security.

■ 6.2 BASIC CONCEPTS ■

6.2.1 Static Web Pages

We will not go into the technicalities of how the Internet works. That would require a very detailed discussion, which is not in the scope of the current text. However, in order to understand the

Internet-security protocols, we need to quickly review the basic concepts behind the working of the Internet. Those familiar with the basic concepts of how the Internet works can safely skip this section.

The main players in any Internet-based communication or transaction are the Web browser (client) and the Web server (server). The Hyper Text Transfer Protocol (HTTP) is used for communication between the browser and the server. This is of a request-response form. That is, the browser sends an HTTP request, the server sends an HTTP response, and the communication between them ends. These types of Web pages are called **static Web pages**. The reason for this is simple. A Web page (written in a language called Hyper Text Mark Up Language, or HTML) is created by an application developer/designer and is stored on a Web server. Whenever any user requests for that page, the Web server sends back the page without performing any additional processing. All it does is to locate that page on its hard disk, add HTTP headers, and send back an HTTP response. Thus, the contents of the Web page do not change depending on the request—they are always the same (unless, of course, they are physically changed on the server's hard disk). Hence, the name *static* has been assigned. Static Web pages follow a simple HTTP request-response flow as shown in Fig. 6.1.

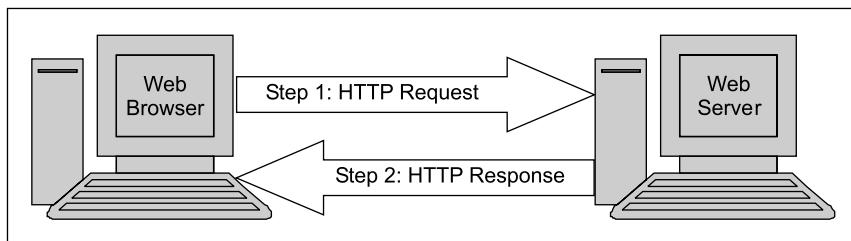


Fig. 6.1 Static Web page

Let us study an example of this HTTP request and response model. In this example, the browser (i.e. the client) retrieves an HTML document from the Web server.

As shown in Fig. 6.2, the client sends a GET command to retrieve an image with the path /files/new/image1. That is, the name of the file is image1, and it is stored in the files/new directory of the Web server. Instead, the Web browser could have, of course, requested for an HTML page (i.e. a file with an *html* extension). In response, the Web server sends an appropriate return code of 200, which means that the request was successfully processed, and also the image data, as requested.

The browser sends a request with the GET command, as discussed. It also sends two more parameters by using two *Accept* commands. These parameters specify that the browser is capable of handling images in the GIF and JPEG formats. Therefore, the server should send the image file only if it is in one of these formats.

In response, the server sends a return code of 200 (OK). It also sends the information about the date and time when this response was sent back to the browser. The server's name is the same as the domain name. Finally, the server indicates that it is sending 3010 bytes of data (i.e. the image file is made up of bits equivalent to 3010 bytes). This is followed by the actual data of the image file (not shown in the figure).

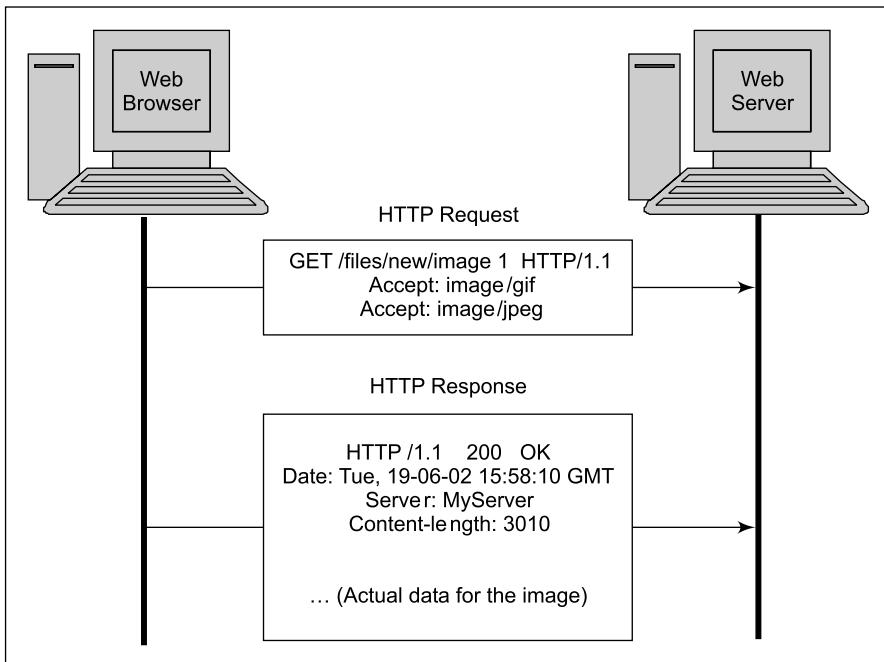


Fig. 6.2 Sample HTTP request and response interaction between a Web browser and a Web server

6.2.2 Dynamic Web Pages

Static Web pages are not always useful. They are suitable for contents that do not change often. For instance, a static Web page would be a good candidate for showing the corporate information of an organization on the organization's home page, or the history of a country on the country's home page. However, for information that changes quite often, for example, stock prices, weather information, news and sports updates, static Web pages would not serve the purpose. Imagine a situation wherein somebody (usually the person who maintains a Web site) has to physically change the Web page every 10 seconds to show the latest update of the stock prices! Clearly, it is simply infeasible to physically alter the HTML page so frequently.

Dynamic Web pages offer a solution to such problems. A dynamic Web page is, as dynamic as its name suggests! The contents of a dynamic Web page can vary all day depending on a number of parameters. For instance, they could change to reflect the latest stock prices, or depending on whether user *A* or *B* has asked for certain information, the contents could be filtered out and only those allowed for *A* or *B* could be shown. It is obvious that dynamic Web pages, therefore, are more complex than static Web pages. In fact, dynamic Web pages involve much more than only HTML. Creating dynamic Web pages involves server-side programming.

Conceptually, when a user requests for a dynamic Web page, the Web server cannot simply send back an HTML page unlike what happens in the case of a static Web page. Here, the Web server actually invokes a program that resides on its hard disk. The program might in turn access databases, perform transaction processing, etc. However, in any case, the program outputs HTML, which is used to

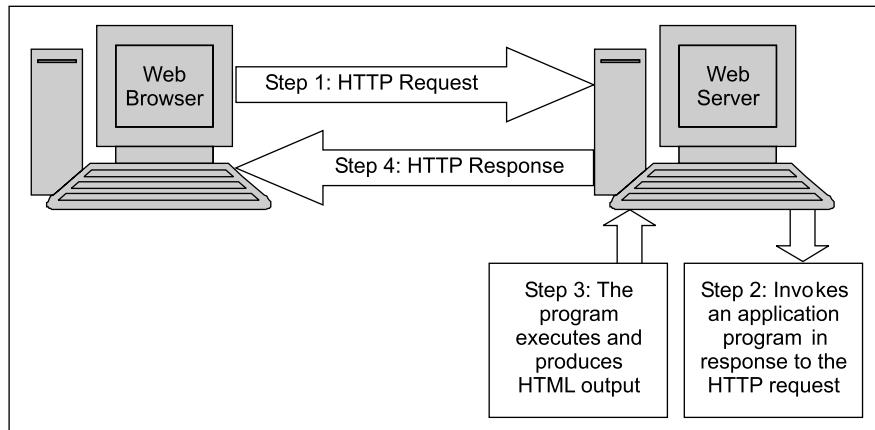


Fig. 6.3 Dynamic Web page

construct an HTTP response by the Web server. The Web server sends the HTTP response thus formed, back to the Web browser. This is shown in Fig. 6.3.

For instance, a dynamic Web page could be written for reading the latest stock prices from a database and using them to respond to HTTP requests for that page. The actual process of updating the database with the latest stock prices need not anyway depend on the dynamic Web page. That could be handled totally separately.

As we can see, the major difference between a static Web page and a dynamic Web page is the involvement of an application program on the server-side in the latter case. However, to reiterate, both static and dynamic Web pages have to return back HTML contents to the Web browser using the HTTP protocol, so that the browser can interpret them and display them.

Many tools allow the creation of dynamic Web pages. Common Gateway Interface (CGI) was quite a popular dynamic-Web-page-creation technology for many years. This was followed by Microsoft's Active Server Pages (ASP). These days, the more popular dynamic Web-page technologies are Sun Microsystems's Java Servlets and Java Server Pages (JSP), Microsoft's ASP.NET, and the PHP technology. We also need to mention that some recent technologies such as AJAX (Asynchronous JavaScript and XML) are also making waves in this space.

6.2.3 Active Web Pages

With the arrival of the popular programming language Java, **active Web pages** became quite popular. The idea behind active Web pages is actually quite simple. When a client sends an HTTP request for an active Web page, the Web server sends back an HTTP response that contains an HTML page as usual. In addition, the HTML page also contains a small program that executes on the client computer inside the Web browser. This is shown in Fig. 6.4.

Usually, the small program sent to the browser along with the HTML page is called a Java applet. An applet is a client-side program written in the Java programming language that can be executed by a Web browser. Thus, a Web browser needs to have a Java interpreter to interpret the code of an applet and execute it on the client side.

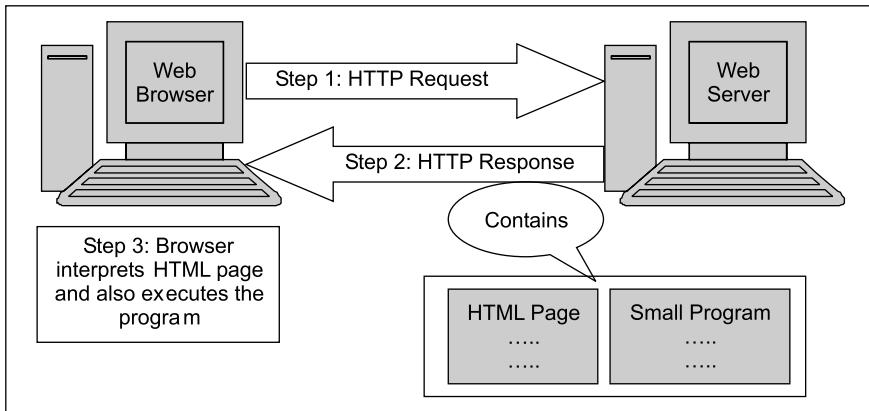


Fig. 6.4 Active Web page

Applets can be used to perform a variety of tasks such as painting images, graphs, charts and other drawing objects on the client browser screen. In addition, it can be used for other purposes such as requesting the Web page to automatically refresh after a fixed interval (say every 30 seconds). Thus, we can construct an active Web page containing stock prices in the HTML format and an applet that refreshes the HTML contents after every 30 seconds. Note that in order to do this, the applet would need to open a connection with the server every 30 seconds. Since the client keeps requesting information automatically from the server (i.e. the client *pulls* information) after a specified interval, this technology is called **client pull**.

Microsoft had also implemented active Web pages technology with its ActiveX controls. ActiveX controls are conceptually quite similar to Java applets. However, the main difference between applets and ActiveX controls is that whereas applets have a lot of restrictions on them, ActiveX controls are reasonably free to do what they want. For instance, an applet cannot write to the client's hard disk. On the other hand, an ActiveX control has no such restrictions. Therefore, the general tendency is to trust applets more than ActiveX controls. However, these days, signed applets are allowed far more access to the client computer, blurring the differences between applets and ActiveX controls.

Another significant difference between applets and ActiveX controls is that an applet is downloaded with an active Web page, executed inside the browser, and destroyed when the user exits that Web page. On the other hand, once downloaded, an ActiveX control remains on the client computer until it is explicitly deleted. This every-time-downloading makes applets quite slow as compared to ActiveX controls.

This should give us a reasonable idea about the typical communication between a Web browser and a Web server using the Internet.

6.2.4 Protocols and TCP/IP

Having discussed the basic means of communication between the browser and the Web server, let us now think how the browser and the server actually know how to communicate with each other. This is similar to thinking about any conversation, such as a telephone conversation. When we speak with someone over the phone, we follow certain conventions. For instance, when the person called picks up the phone, she usually greets the caller with a *Hello*. The caller then identifies himself (*e.g. Hi, this is Atul*).

The caller then describes the purpose of calling (e.g. *I have called up to inquire if we could have dinner tonight*). The conversation then continues, and ends with both sides deciding to hang up (*Bye*).

Other conversations can be more difficult. For instance, suppose a person knowing only Hindi wants to communicate with someone who knows only English. In such a case, the two persons cannot communicate with each other directly. They require the help of a translator, who knows both Hindi and English, and therefore, can facilitate the conversation between the two persons.

Also, in any conversation, one of the persons may not hear the other person clearly. In such a case, that person may request the other person to repeat what he/she said. Alternatively, one of the persons could be speaking too quickly, in which case, the other person may request him/her to speak a bit slowly.

Similar situations occur in computer-to-computer communications over the Internet. Since the Internet is made up of computers and networks that have different hardware as well as software characteristics, there must be some universal *translator*, which can facilitate the communication between all these computers. This is precisely what the protocol software does in any data communications, including Internet communications. Protocol software defines an abstract model of communication hierarchy, which is independent of all physical characteristics of the computers and computer networks. As long as all participating computers and computer networks adhere to the standards specified by the protocol software, they can communicate with each other without worrying about their inherent differences.

The **Transmission Control Protocol/Internet Protocol (TCP/IP)** software is the *translator* that makes this magic work on the Internet. TCP/IP is a combination of many protocols that facilitate communication between computers over the Internet. It specifies, for example, how a browser should identify a server, how it should send an HTTP request to the server, how should a server respond, what to do in the case of an error, and so on. Of course, apart from HTTP, TCP/IP also supports other applications, such as email, file transfer, etc. However, we need not discuss that here.

From our current scope and perspective, what is important to know is how the TCP/IP protocol suite looks like. This is because only then can we appreciate what is required to make the communication using TCP/IP secure. The original TCP/IP protocol suite consisted of four layers. However, for the sake of understanding, we will slightly modify this specification and consider that the TCP/IP protocol suite consists of five layers, as shown in Fig. 6.5.

Note that the classical TCP/IP protocol suite combines the bottom-most two layers (data link and physical), and hence talks about only four layers. However, we are showing them separately for the purpose of understanding. Sometimes, our five-layer TCP/IP layered organization is shown in a different form, as illustrated in Fig. 6.6.

Layer Number	Layer Name
5 (Highest)	Application
4	Transport
3	Internet
2	Data link
1 (Lowest)	Physical

Fig. 6.5 TCP/IP layers

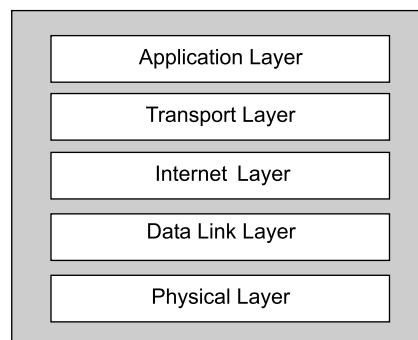


Fig. 6.6 TCP/IP layers

Each of these layers performs a specific pre-defined task. For instance, all the application programs, such as HTTP, email, etc., are a part of the application layer. Thus, when a Web browser communicates with a Web server using the HTTP protocol, the application layer comes into action. The application layer on the client computer interacts with the transport layer of the same computer, which in turn, interacts with the Internet layer on the same computer, which again in turn, interacts with the data link layer of the same computer, which finally interacts with the physical layer of the same computer. At this stage, the bits are sent as voltage or current pulses via the physical transmission medium.

On the server side, after the bits are received by the physical layer in the form of voltage or current pulses, the direction of communication is the reverse (physical layer to application layer). This concept is shown in Fig. 6.7. Here, we assume that X is the browser and Y is the Web server.

Note that the intermediate nodes (i.e. the computers between the browser and the server) do not have any interaction at the application and transport layers, because they are merely involved in the forwarding of information (actually, packets) from the source (X) to the destination (Y).

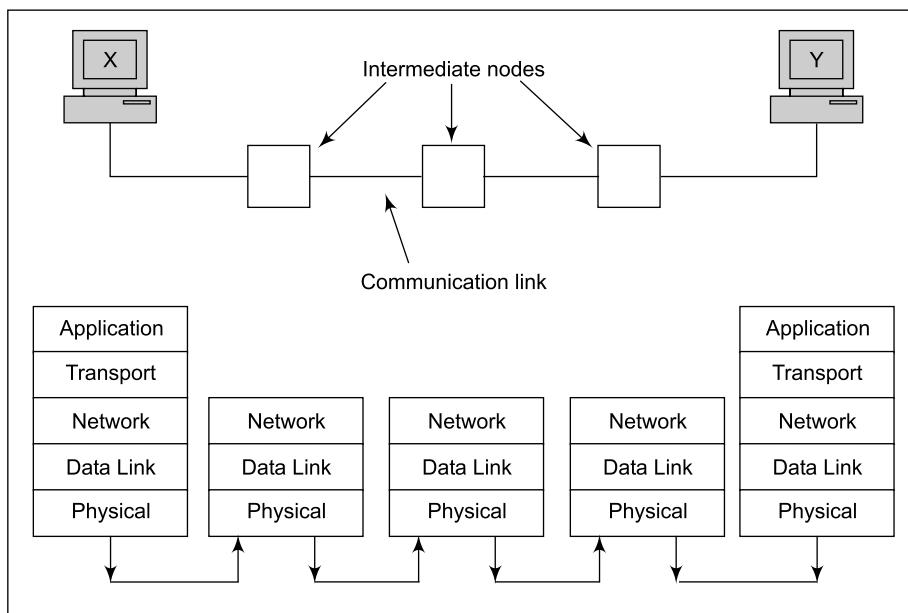


Fig. 6.7 Communication using TCP/IP

Note that within a host (either X or Y in this example), each layer calls upon the services of its lower layer. For instance, layer 5 uses the services provided by layer 4. Layer 4 in turn, uses the services of layer 3, and so on. Between X and Y , the communication appears to be taking place between the layers at the same level. This is called as *virtual communication* or *virtual path* between X and Y . For instance, layer 5 on host X *thinks* that it is communicating directly with layer 5 on host Y . Similarly, layer 4 on host X and layer 4 on host Y have a *virtual communication* connection between them.

6.2.5 Layered Organization

The application-layer software running at the source node creates the data (in the form of data units called *frames* or *packets*) to be transmitted to the application-layer software running at a destination node (remember *virtual path*?). It hands it over to the transport layer at the source node. Each of the remaining TCP/IP layers from this point onwards adds its own header to the frame as it moves from this layer (transport layer) to the bottom-most layer (the physical layer) at the source node. At the lowest physical layer, the data is transmitted as voltage pulses across the communication medium such as coaxial cable.

That means that, the application layer (layer 5) hands over the entire data to the transport layer. Let us call this *L5 data*. After the transport layer receives and processes this data, it adds its own header to the original data and sends it to the next layer in the hierarchy (i.e. the Internet layer). Therefore, from the fourth (transport) layer to the third (Internet) layer, the data is sent as *L5 data + H4*, where *H4* is the header added by the fourth (transport) layer.

Now, for the third (Internet) layer, *L5 data + H4* is the input data. Let us call this together *L4 data*. When the third (Internet) layer sends this data to the next, i.e. the second (data link) layer, it sends the original data (which is *L4 data*) plus its own header *H3* together, i.e. *L4 data + H3*, and so on. In the end, the original data (*L5*) and all the headers are sent across the physical medium.

Figure 6.8 illustrates this process.

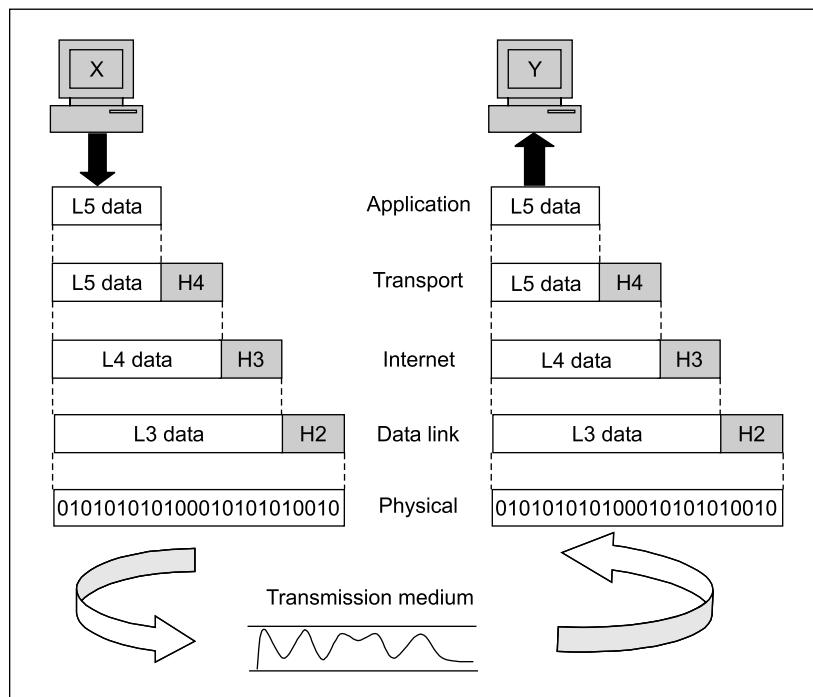


Fig. 6.8 Data exchange using TCP/IP layers

This concludes our discussion about the basics of the Internet and TCP/IP communication concepts.

■ 6.3 SECURE SOCKET LAYER (SSL) ■

6.3.1 Introduction

The **Secure Socket Layer (SSL)** protocol is an Internet protocol for the secure exchange of information between a Web browser and a Web server. It provides two basic security services: authentication and confidentiality. Logically, it provides a secure *pipe* between the Web browser and the Web server. Netscape Corporation developed SSL in 1994. Since then, SSL has become the world's most popular Web-security mechanism. All the major Web browsers support SSL. Currently, SSL comes in three versions: 2, 3 and 3.1. The most popular of them is Version 3, which was released in 1995.

6.3.2 The Position of SSL in TCP/IP Protocol Suite

SSL can be conceptually considered as an additional layer in the TCP/IP protocol suite. The SSL layer is located between the application layer and the transport layer, as shown in Fig. 6.9.

Assuch, the communication between the various TCP/IP protocol layers is now as shown in Fig. 6.10.

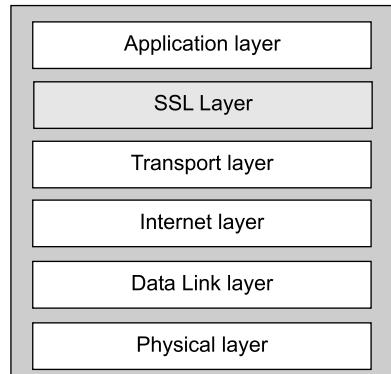


Fig. 6.9 Position of SSL in TCP/IP

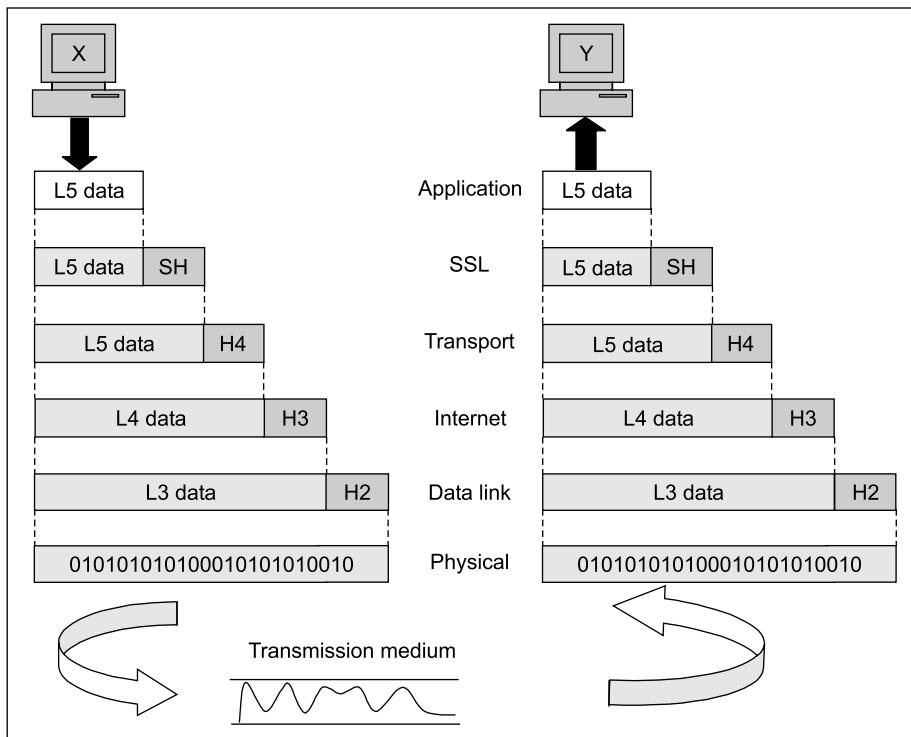


Fig. 6.10 SSL is located between application and transport layers

As we can see, the application layer of the sending computer (X) prepares the data to be sent to the receiving computer (Y), as usual. However, unlike what happens in the normal case, the application-layer data is not passed directly to the transport layer now. Instead, the application-layer data is passed to the SSL layer. Here, the SSL layer performs encryption on the data received from the application layer (which is indicated by a different color), and also adds its own encryption information header, called SSL Header (SH) to the encrypted data. We shall later study what exactly happens in this process.

After this, the SSL layer data (L5) becomes the input for the transport layer. It adds its own header (H4), and passes it on to the Internet layer, and so on. This process happens exactly the way it happens in the case of a normal TCP/IP data transfer. Finally, when the data reaches the physical layer, it is sent in the form of voltage pulses across the transmission medium.

At the receiver's end, the process happens pretty similar to how it happens in the case of a normal TCP/IP connection, until it reaches the new SSL layer. The SSL layer at the receiver's end removes the SSL Header (SH), decrypts the encrypted data, and gives the plain-text data back to the application layer of the receiving computer.

Thus, only the application layer data is encrypted by SSL. The lower-layer headers are not encrypted. This is quite obvious: if SSL has to encrypt all the headers, it must be positioned below the data-link layer. That would serve no purpose at all. In fact, it would lead to problems. If SSL encrypted all the lower-layer headers, even the IP and physical addresses of the computers (sender, receiver, and intermediate nodes) would be encrypted, and become unreadable. Thus, where to deliver the packets would be a big question. To understand the problem, imagine what would happen if we put the address of the sender and the receiver of a letter inside the envelope! Clearly, the postal service would not know where to send the letter! This is also why there is no point in encrypting the lower-layer headers. Therefore, SSL is required between the application and the transport layers.

6.3.3 The Working of SSL

SSL has three sub-protocols, namely the **Handshake Protocol**, the **Record Protocol**, and the **Alert Protocol**. These three sub-protocols constitute the overall working of SSL.

We shall take a look at all the three protocols now.

1. The Handshake Protocol

The *handshake protocol* of SSL is the first sub-protocol used by the client and the server to communicate using an SSL-enabled connection. This is similar to how Alice and Bob would first shake hands with each other accompanied with a *hello* before they start conversing.

As the figure shows, the handshake protocol consists of a series of messages between the client and the server. Each of these messages has the format shown in Fig. 6.11.

Type	Length	Content
1 byte	3 bytes	1 or more bytes

Fig. 6.11 Format of the *handshake protocol* messages

As shown in the figure, each handshake message has three fields, as follows:

(a) Type (1 byte) This field indicates one of the ten possible message types. These ten message types are listed in Fig. 6.12.

(b) Length (3 bytes) This field indicates the length of the message in bytes.

(c) Content (1 or more bytes) This field contains the parameters associated with this message, depending on the message type, as listed in Fig. 6.12.

Let us now take a look at the possible messages exchanged by the client and the server in the handshake protocol, along with their corresponding parameters, as shown in Fig. 6.12.

Message Type	Parameters
Hello request	None
Client hello	Version, Random number, Session id, Cipher suite, Compression method
Server hello	Version, Random number, Session id, Cipher suite, Compression method
Certificate	Chain of X.509V3 certificates
Server-key exchange	Parameters, signature
Certificate request	Type, authorities
Server hello done	None
Certificate verify	Signature
Client-key exchange	Parameters, signature
Finished	Hash value

Fig. 6.12 SSL handshake-protocol message types

The handshake protocol is actually made up of four phases, as shown in Fig. 6.13. These phases are

- Establish security capabilities
- Server authentication and key exchange
- Client authentication and key exchange
- Finish

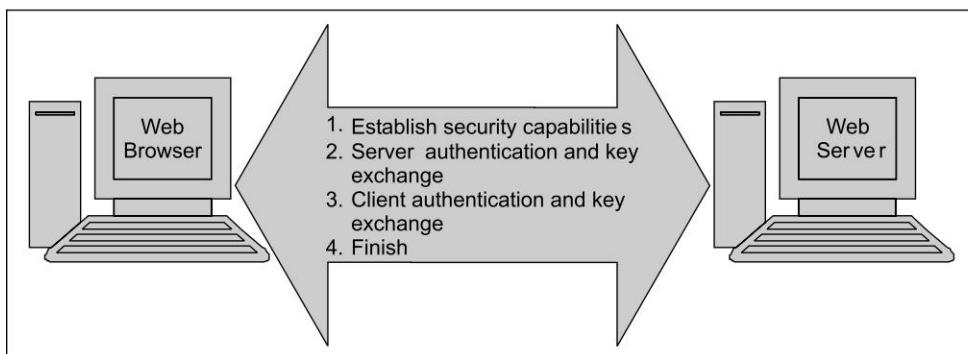


Fig. 6.13 SSL handshake phases

Let us now study these four phases one by one.

Phase 1. Establish Security Capabilities This first phase of the SSL handshake is used to initiate a logical connection and establish the security capabilities associated with that connection. This consists of two messages, the **client hello** and the **server hello**, as shown in Fig. 6.14.

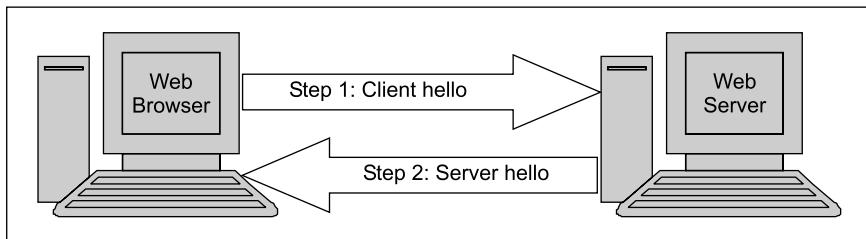


Fig. 6.14 SSL Handshake protocol *Phase 1: Establish security capabilities*

As shown in the figure, the process starts with a *client hello* message from the client to the server. It consists of the following parameters:

■ **Version** This field identifies the highest version of SSL that the client can support. As we have seen, at the time of this writing, this can be 2, 3 or 3.1.

■ **Random** This field is useful for the later, actual communication between the client and the server. It contains two sub-fields:

- A 32-bit date-time field that identifies the current system date and time on the client computer.
- A 28-byte random number generated by the random-number generator software built inside the client computer.

■ **Session id** This is a variable-length session identifier. If this field contains a non-zero value, it means that there is already a connection between the client and the server, and the client wishes to update the parameters of that connection. A zero value in this field indicates that the client wants to create a new connection with the server.

■ **Cipher suite** This list contains a list of the cryptographic algorithms supported by the client (e.g. RSA, Diffie-Hellman, etc.), in the decreasing order of preference.

■ **Compression method** This field contains a list of the compression algorithms supported by the client.

The client sends the *client hello* message to the server and waits for the server's response. Accordingly, the server sends back a *server hello* message to the client. This message also contains the same fields as in the *client hello* message. However, their purpose is now different. The *server hello* message consists of the following fields:

■ **Version** This field identifies the lower of the versions suggested by the client and the highest supported by the server. For instance, if the client had suggested version 3, but the server also supports version 3.1, the server will select 3.

■ **Random** This field has the same structure as the *Random* field of the client. However, the *Random* value generated by the server is completely independent of the client's *Random* value.

■ Session id If the session id value sent by the client was non-zero, the server uses the same value. Otherwise, the server creates a new session id and puts it in this field.

■ Cipher suite Contains a single cipher suite, which the server selects from the list sent earlier by the client.

■ Compression method Contains a compression algorithm, which the server selects from the list sent earlier by the client.

Phase 2. Server Authentication and Key Exchange The server initiates this second phase of the SSL handshake, and is the sole sender of all the messages in this phase. The client is the sole recipient of all these messages. This phase contains four steps, as shown in Fig. 6.15. These steps are **Certificate**, **Server key exchange**, **Certificate request**, and **Server hello done**.

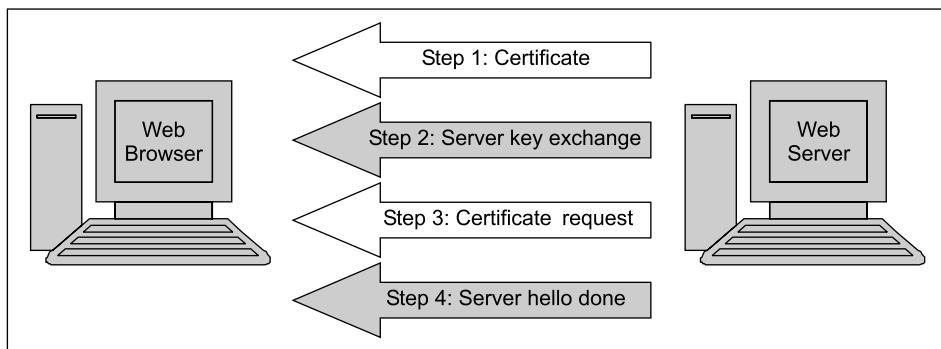


Fig. 6.15 SSL Handshake protocol Phase 2: Server authentication and key exchange

Let us discuss the four steps of this phase.

In the first step (*certificate*), the server sends its digital certificate and the entire chain leading up to root CA to the client. This will help the client to authenticate the server using the server's public key from the server's certificate. The server's certificate is mandatory in all situations, except if the key is being agreed upon by using Diffie-Hellman.

The second step (*Server key exchange*) is optional. It is used only if the server does not send its digital certificate to the client in step 1 above. In this step, the server sends its public key to the client (as the certificate is not available).

The third step (*certificate request*), the server can request for the client's digital certificate. The client authentication in SSL is optional, and the server may not always expect the client to be authenticated. Therefore, this step is optional.

The last step (*server hello done*) message indicates to the client that its portion of the *hello* message (i.e. the *server hello* message) is complete. This indicates to the client that the client can now (optionally) verify the certificates sent by the server, and ensure that all the parameters sent by the server are acceptable. This message does not have any parameters. After sending this message, the server waits for the client's response.

Phase 3. Client Authentication and Key Exchange The client initiates this third phase of the SSL handshake, and is the sole sender of all the messages in this phase. The server is the sole re-

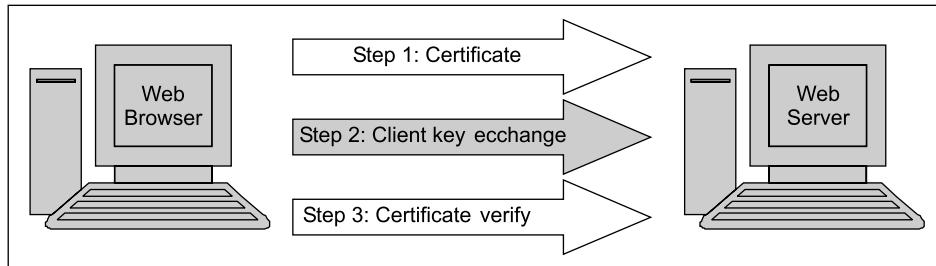


Fig. 6.16 SSL Handshake protocol *Phase 3: Client authentication and key exchange*

cipient of all these messages. This phase contains three steps, as shown in Fig. 6.16. These steps are **Certificate**, **Client key exchange**, and **Certificate verify**.

The first step (*certificate*) is optional. This step is performed only if the server had requested for the client's digital certificate. If the server has requested for the client's certificate, and if the client does not have one, the client sends a *No certificate* message, instead of a *Certificate* message. It then is up to the server to decide if it wants to still continue or not.

Like the *server key exchange* message, this second step (*client key exchange*) allows the client to send information to the server, but in the opposite direction. This information is related to the symmetric key that both the parties will use in this session. Here, the client creates a 48-byte *pre-master secret*, and encrypts it with the server's public key and sends this encrypted *pre-master secret* to the server.

The third step (*Certificate verify*) is necessary only if the server had demanded client authentication. As we know, if this is the case, the client has already sent its certificate to the server. However, additionally, the client also needs to prove to the server that it is the correct and authorized holder of the private key corresponding to the certificate. For this purpose, in this optional step, the client combines the *pre-master secret* with the random numbers exchanged by the client and the server earlier (in *Phase 1: Establish security capabilities*) after hashing them together using MD5 and SHA-1, and signs the result with its private key.

Phase 4. Finish The client initiates this fourth phase of the SSL handshake, which the server ends. This phase contains four steps, as shown in Fig. 6.17. The first two messages are from the client: **Change cipher specs**, **Finished**. The server responds back with two identical messages: **Change cipher specs**, **Finished**.

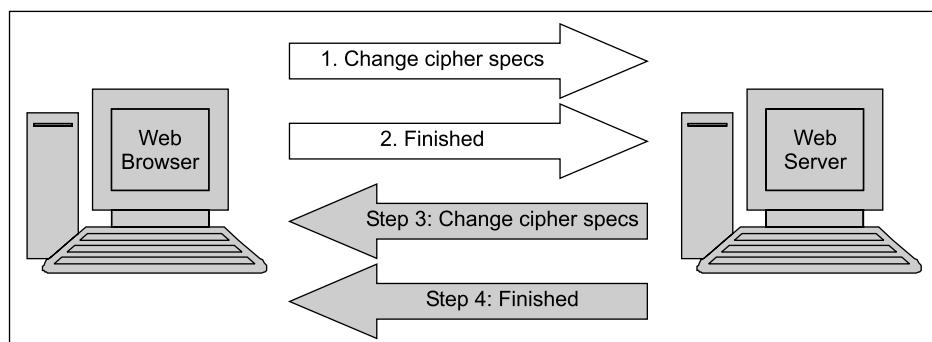


Fig. 6.17 SSL Handshake protocol *Phase 4: Finished*

Based on the *pre-master secret* that was created and sent by the client in the *Client key exchange* message, both the client and the server create a *master secret*. Before secure encryption or integrity verification can be performed on records, the client and server need to generate shared secret information known only to them. This value is a 48-byte quantity called the *master secret*. The *master secret* is used to generate keys and secrets for encryption and MAC computations. The *master secret* is calculated after computing message digests of the *pre-master secret*, client random and server random, as shown in Fig. 6.18.

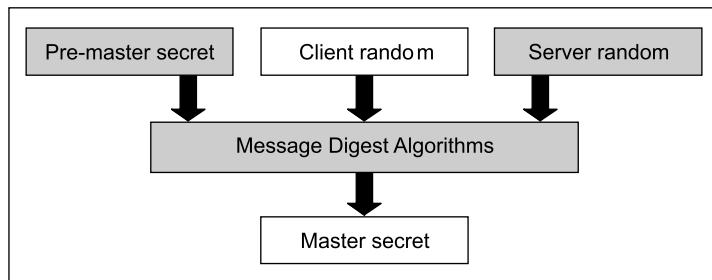


Fig. 6.18 Master secret generation concept

The technical specification for calculating the *master secret* is as follows:

Master_secret =

$$\begin{aligned}
 & \text{MD5}(\text{pre_master_secret} + \text{SHA}('A' + \text{pre_master_secret} + \\
 & \quad \text{ClientHello.random} + \text{ServerHello.random})) + \\
 & \text{MD5}(\text{pre_master_secret} + \text{SHA}('BB' + \text{pre_master_secret} + \\
 & \quad \text{ClientHello.random} + \text{ServerHello.random})) + \\
 & \text{MD5}(\text{pre_master_secret} + \text{SHA}('CCC' + \text{pre_master_secret} + \\
 & \quad \text{ClientHello.random} + \text{ServerHello.random}))
 \end{aligned}$$

Finally, the symmetric keys to be used by the client and the server are generated. For this, the conceptual process is used as shown in Fig. 6.19.

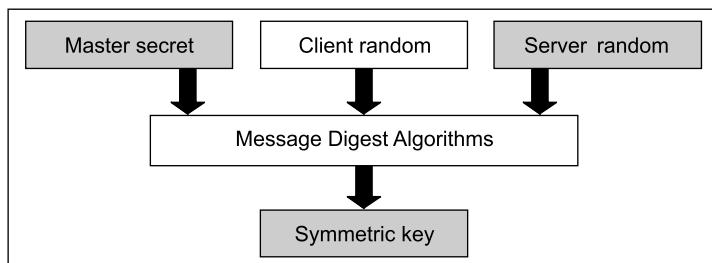


Fig. 6.19 Symmetric-key generation concept

The actual key-generation formula is as follows:

```
key_block =
    MD5(master_secret + SHA(`A' + master_secret +
        ServerHello.random +
        ClientHello.random)) +
    MD5(master_secret + SHA(`BB' + master_secret +
        ServerHello.random +
        ClientHello.random)) +
    MD5(master_secret + SHA(`CCC' + master_secret +
        ServerHello.random +
        ClientHello.random))
```

After this, the first step (*Change cipher specs*) is a confirmation from the client that all is well from its end, which it strengthens with the *Finished* message. The server sends identical messages to the client.

2. The Record Protocol

The *Record Protocol* in SSL comes into picture after a successful handshake is completed between the client and the server. That is, after the client and the server have optionally authenticated each other and have decided what algorithms to use for secure information exchange, we enter into the SSL *record protocol*. This protocol provides two services to an SSL connection, as follows:

(a) Confidentiality This is achieved by using the secret key that is defined by the *handshake protocol*.

(b) Integrity The *handshake protocol* also defines a shared secret key (MAC) that is used for assuring the message integrity.

The operation of the *record protocol* is shown in Fig. 6.20.

As the figure shows, the SSL record protocol takes an application message as input. First, it fragments it into smaller blocks, optionally compresses each block, adds MAC, encrypts it, adds a header and gives it to the transport layer, where the TCP protocol processes it like any other TCP block. At the receiver's end, the header of each block is removed; the block is then decrypted, verified, decompressed, and reassembled into application messages. Let us discuss these steps in more detail.

(c) Fragmentation The original application message is broken into blocks, so that the size of each block is less than or equal to 2^{14} bytes (16,384 bytes).

(d) Compression The fragmented blocks are optionally compressed. The compression process must not result into the loss of the original data, which means that this must be a lossless compression mechanism.

(e) Addition of MAC Using the shared secret key established previously in the *handshake protocol*, the Message Authentication Code (MAC) for each block is calculated. This operation is similar to the HMAC algorithm.

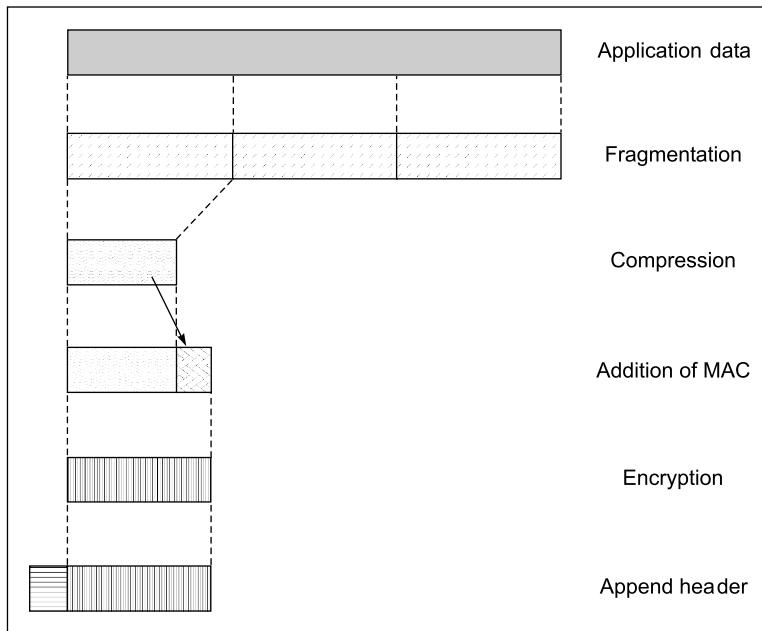


Fig. 6.20 SSL record protocol

(f) Encryption Using the symmetric key established previously in the *handshake protocol*, the output of the previous step is now encrypted. This encryption may not increase the overall size of the block by more than 1024 bytes. Figure 6.21 lists the permitted encryption algorithms.

Stream cipher		Block cipher	
Algorithm	Key size	Algorithm	Key size
RC4	40	AES	128, 256
RC4	128	IDEA	128
		RC2	40
		DES	40
		DES	56
		DES-3	168
		Fortezza	80

Fig. 6.21 Permitted SSL encryption algorithms

(g) Append Header Finally, a header is added to the encrypted block. The header contains the following fields:

Content type (8 bits) Specifies the protocol used for processing the record in the next higher level (e.g. handshake, alert, change cipher).

Major version (8 bits) Specifies the major version of the SSL protocol in use. For instance, if SSL version 3.1 is in use, this field contains 3.

Minor version (8 bits) Specifies the minor version of the SSL protocol in use. For instance, if SSL version 3.0 is in use, this field contains 0.

Compressed length (16 bits) Specifies the length in bytes of the original plain-text block (or the compressed block, if compression is used).

The final SSL message now looks as shown in Fig. 6.22.

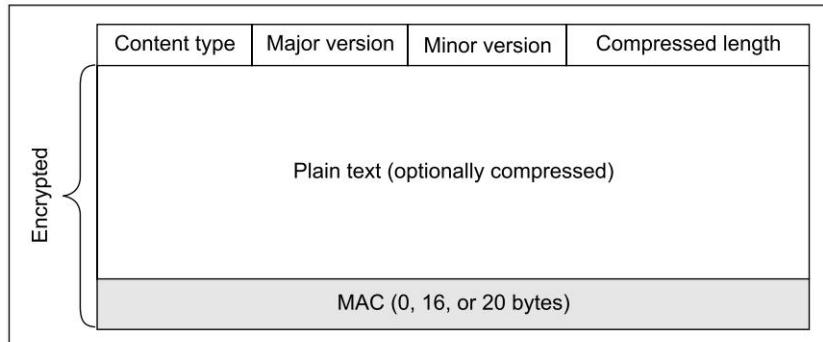


Fig. 6.22 Final output after SSL record protocol operations

3. The Alert Protocol

When either the client or the server detects an error, the detecting party sends an *alert message* to the other party. If the error is fatal, both the parties immediately close the SSL connection (which means that the transmission from both the ends is terminated immediately). Both the parties also destroy the session identifiers, secrets and keys associated with this connection before it is terminated. Other errors, which are not so severe, do not result in the termination of the connection. Instead, the parties handle the error and continue.

Each *alert message* consists of two bytes. The first byte signifies the type of error. If it is a warning, this byte contains 1. If the error is fatal, this byte contains 2. The second byte specifies the actual error. This is shown in Fig. 6.23.

We list the fatal alerts (errors) in Fig. 6.24.

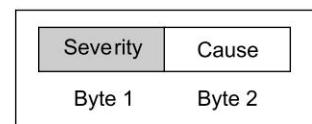


Fig. 6.23 Alert protocol message format

Alert	Description
Unexpected message	An inappropriate message was received.
Bad record MAC	A message is received without a correct MAC.
Decompression failure	The decompression function received an improper input.
Handshake failure	Sender was unable to negotiate an acceptable set of security parameters from the available options.
Illegal parameters	A field in the handshake message was out of range or was inconsistent with the other fields.

Fig. 6.24 Fatal alerts

The remaining (non-fatal) alerts are shown in Fig. 6.25.

Alert	Description
No certificate	Sent in response to certificate request if an appropriate certificate is not available.
Bad certificate	A certificate was corrupt (its digital signature verification failed).
Unsupported certificate	The type of the received certificate is not supported.
Certificate revoked	The signer of a certificate has revoked it.
Certificate expired	A received certificate has expired.
Certificate unknown	An unspecified error occurred while processing the certificate.
Close notify	Notifies that the sender will not send any more messages in this connection. Each party must send this message before closing its side of the connection.

Fig. 6.25 Non-fatal alerts

6.3.4 Closing and Resuming SSL Connections

Before ending their communication, the client and the server must inform each other that their side of the connection is ending. As we have noted, each party sends a *Close notify* alert to the other party. This ensures a graceful closure of the connection. When a party receives this alert, it must immediately stop whatever it is doing, send its own *Close notify* alert and end the connection from its side as well. If an SSL connection ends without a *Close notify* from either party, it cannot be resumed.

The *handshake protocol* in SSL is quite complex and time consuming, as it uses asymmetric-key cryptography. Therefore, if desired, a client and a server can decide to reuse or resume an earlier SSL connection, rather than creating a fresh one with a new handshake. However, for this to be possible, both the parties must agree on the reuse. If either party feels that it is dangerous to reuse the earlier connection, or if the other party's certificate has expired since the last connection, it can force the other party to perform a fresh handshake. As per the SSL specifications, any SSL connection should not be reused after 24 hours in any case.

6.3.5 Buffer Overflow Attacks on SSL

A **buffer overflow** occurs when a program or process tries to store more data in a buffer (a temporary data storage area) than what it was designed to hold. Because buffers are created to contain a fixed amount of data, the extra information—which has to go somewhere—can overflow into adjacent buffers, corrupting or overwriting the valid data held in them. Although this may happen accidentally through programming error, buffer overflow is an increasingly common type of security attack on data integrity. In buffer overflow attacks, the extra data may contain codes designed to cause specific actions, thus sending new instructions to the attacked computer. This could damage the user's files, change data, or compromise confidential information.

OpenSSL is an open-source implementation of the Secure Sockets Layer (SSL) protocol. OpenSSL is subject to four remotely exploitable buffer overflows. The buffer overflow vulnerabilities can allow an attacker to execute arbitrary code on the target (victim) computer with the privilege level of the

OpenSSL process, as well as providing opportunities for launching a denial-of-service attack. These have been more in theory than in practice.

1. Three of the four buffer-overflow vulnerabilities occur in the SSL handshakes. The last one deals with 64-bit operating systems.
2. The first vulnerability is found in the key exchange implemented in SSL Version 2. A client can be used to send an oversized master key to an SSL Version 2 server, enabling denial of service or malicious code execution on the server.
3. The second buffer overflow is contained in the SSL Version 3 handshake. A malicious server can execute code on an OpenSSL client by sending a malformed session ID during the first phase of the handshake.
4. The third buffer overflow exists in OpenSSL servers running SSL Version 3 with Kerberos authentication enabled. A malicious client can send an oversized master key to a Kerberos-enabled SSL server.
5. The fourth set of overflows exists only on 64-bit operating systems. Several buffers used to store ASCII representations of integers are smaller than required.

■ 6.4 TRANSPORT LAYER SECURITY (TLS) ■

Transport Layer Security (TLS) is an IETF standardization initiative, whose goal is to come out with an Internet standard version of SSL. Netscape wanted to standardize SSL, and hence handed the protocol over to IETF. There are subtle differences between SSL and TLS. However, the core idea and implementation are quite similar. TLS is defined in RFC 2246.

Figure 6.26 summarizes the differences between SSL and TLS.

Property	SSL	TLS
Version	3.0	1.0
Cipher suite	Supports an algorithm called Fortezza	Does not support Fortezza
Cryptography secret	Computed as explained earlier in the chapter	Uses a pseudorandom function to create master secret
Alert protocol	As explained earlier in the chapter	The <i>No certificate</i> alert message is deleted. The following are newly added: <i>Decryption failed</i> , <i>Record overflow</i> , <i>Unknown CA</i> , <i>Access denied</i> , <i>Decode error</i> , <i>Export restriction</i> , <i>Protocol version</i> , <i>Insufficient security</i> , <i>Internal error</i> .
Handshake protocol	As explained earlier in the chapter	Some details are changed
Record protocol	Uses MAC	Uses HMAC

Fig. 6.26 Differences between SSL and TLS

■ 6.5 SECURE HYPER TEXT TRANSFER PROTOCOL (SHTTP) ■

The **Secure Hyper Text Transfer Protocol (SHTTP)** is a set of security mechanisms defined for protecting the Internet traffic. This includes the data-entry forms and Internet-based transactions.

Note that an HTTP request sent by using SSL is identified as HTTPS (e.g. [HTTPS://www.yahoo.com](https://www.yahoo.com)), whereas this is SHTTP (e.g. [sHTTP://www.yahoo.com](shttp://www.yahoo.com)). The services offered by SHTTP are quite similar to those of SSL. However, SSL has become highly successful—SHTTP has not. SHTTP works at the application layer, and is therefore, tightly coupled with HTTP, unlike SSL (which sits between the application and the transport layers). Figure 6.27 illustrates the difference.

SHTTP supports both authentication and encryption of HTTP traffic between the client and the server. The encryption and digital signature formats used in SHTTP have origins in the PEM protocol, which we shall study later.

The key difference between SSL and SHTTP is that SHTTP works at the level of individual messages. It can encrypt and sign individual messages. On the other hand, SSL does not differentiate between different messages. Instead, it aims at making the connection between a client and the server, regardless of the messages that they are exchanging. Also, SSL cannot perform digital signatures.

SHHTTP is very rarely used, and therefore, we shall not discuss it any further.

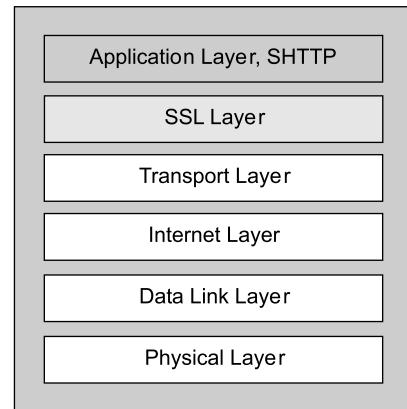


Fig. 6.27 Positions of SHTTP and SSL in TCP/IP protocol suite

■ 6.6 SECURE ELECTRONIC TRANSACTION (SET) ■

6.6.1 Introduction

The **Secure Electronic Transaction (SET)** is an open encryption and security specification that is designed for protecting credit-card transactions on the Internet. The pioneering work in this area was done in 1996 by MasterCard and Visa jointly. They were joined by IBM, Microsoft, Netscape, RSA, Terisa, and VeriSign. Starting from that time, there have been many tests of the concept, and by 1998 the first generation of SET-compliant products appeared in the market.

The need for SET came from the fact that MasterCard and Visa realized that for e-commerce payment processing, software vendors were coming up with new and conflicting standards. Microsoft mainly drove these on one hand, and IBM on the other. To avoid all sorts of future incompatibilities, MasterCard and Visa decided to come up with a standard, ignoring all their competition issues, and in the process, involving all the major software manufacturers.

SET is not a payment system. Instead, it is a set of security protocols and formats that enable the users to employ the existing credit-card payment infrastructure on the Internet in a secure manner. SET services can be summarized as follows:

1. It provides a secure communication channel among all the parties involved in an e-commerce transaction.
2. It provides authentication by the use of digital certificates.
3. It ensures confidentiality, because the information is only available to the parties involved in a transaction, and that too only when and where necessary.

SET is a very complex specification. In fact, when released, it took 971 pages to describe SET across three books! (Just for the record, SSL Version 3 needs 63 pages to describe). Thus, it is not possible to discuss it in great detail. However, we shall summarize the key points.

6.6.2 SET Participants

Before we discuss SET, let us summarize the participants in the SET system.

(a) Cardholder Using the Internet, consumers and corporate purchasers interact with merchants for buying goods and services. A cardholder is an authorized holder of a payment card such as MasterCard or Visa that has been issued by an **issuer** (discussed subsequently).

(b) Merchant A merchant is a person or an organization that wants to sell goods or services to cardholders. A merchant must have a relationship with an **acquirer** (discussed subsequently) for accepting payments on the Internet.

(c) Issuer The **issuer** is a financial institution (such as a bank) that provides a payment card to a cardholder. The most critical point is that the issuer is ultimately responsible for the payment of the cardholder's debt.

(d) Acquirer This is a financial institution that has a relationship with merchants for processing payment-card authorizations and payments. The reason for having acquirers is that merchants accept credit cards of more than one brand, but are not interested in dealing with so many bankcard organizations or issuers. Instead, an acquirer provides the merchant an assurance (with the help of the issuer) that a particular cardholder account is active and that the purchase amount does not exceed the credit limits, etc. The acquirer also provides electronic funds transfer to the merchant account. Later, the issuer reimburses the acquirer using some payment network.

(e) Payment Gateway This is a task that can be taken up by the acquirer or it can be taken up by an organization as a dedicated function. The payment gateway processes the payment messages on behalf of the merchant. Specifically in SET, the payment gateway acts as an interface between SET and the existing card-payment networks for payment authorizations. The merchant exchanges SET messages with the payment gateway over the Internet. The payment gateway, in turn, connects to the acquirer's systems using a dedicated network line in most cases.

(f) Certification Authority (CA) As we know, this is an authority that is trusted to provide public key certificates to cardholders, merchants and payment gateways. In fact, CAs are very crucial to the success of SET.

6.6.3 The SET Process

Let us now take a simplistic look at the SET process before we describe the technical details of the SET process.

1. The Customer Opens an Account

The customer opens a credit-card account (such as MasterCard or Visa) with a bank (issuer) that supports electronic payment mechanisms and the SET protocol.

2. The Customer Receives a Certificate

After the customer's identity is verified (with the help of details such as passport, business documents, etc.), the customer receives a digital certificate from a CA. The certificate also contains details such as the customer's public key and its expiration date.

3. The Merchant Receives a Certificate

A merchant that wants to accept a certain brand of credit cards must possess a digital certificate.

4. The Customer Places an Order

This is a typical *shopping-cart* process wherein the customer browses the list of items available, searches for specific items, selects one or more of them, and places the order. The merchant, in turn, sends back details such as the list of items selected, their quantities, prices, total bill, etc., back to the customer for his record, with the help of an order form.

5. The Merchant is Verified

The merchant also sends its digital certificate to the customer. This assures the customer that he/she is dealing with a valid merchant.

6. The Order and Payment Details are Sent

The customer sends both the order and payment details to the merchant along with the customer's digital certificate. The order confirms the purchase transaction with reference to the items mentioned in the order form. The payment contains credit-card details. However, the payment information is so encrypted that the merchant cannot read it. The customer's certificate assures the merchant of the customer's identity.

7. The Merchant Requests Payment Authorization

The merchant forwards the payment details sent by the customer to the payment gateway via the acquirer (or to the acquirer if the acquirer also acts as the payment gateway) and requests the payment gateway to authorize the payment (i.e. ensure that the credit card is valid and that the credit limits are not breached).

8. The Payment Gateway Authorizes the Payment

Using the credit-card information received from the merchant, the payment gateway verifies the details of the customer's credit card with the help of the issuer, and either authorizes or rejects the payment.

9. The Merchant Confirms the Order

Assuming that the payment gateway authorizes the payment, the merchant sends a confirmation of the order to the customer.

10. The Merchant Provides Goods or Services

The merchant now ships the goods or provides the services as per the customer's order.

11. The Merchant Requests Payment

The payment gateway receives a request from the merchant for making the payment. The payment gateway interacts with the various financial institutions such as the issuer, acquirer, and the clearing house to effect the payment from the customer's account to the merchant's account.

6.6.4 How SET Achieves its Objectives

The main concern with online-payment mechanisms is that they demand that the customer send his/her credit-card details to the merchant. There are two aspects of this. One is that the credit-card number travels in clear text, which provides an intruder with an opportunity to know that number and use it with malicious intentions (for instance, to make his/her own payments using that credit-card number). The second issue is that the credit-card number is available to the merchant, who can misuse it.

The first concern is generally dealt with by SSL. Since all information exchange in SSL happens in an encrypted form, such that an intruder cannot make any sense out of it. Therefore, even if an intruder is able to listen to an active conversation between a client and a server over the Internet, as long as the session is SSL-enabled, the intruder's intentions will be defeated. However, SSL does not achieve the second objective, which is of protecting the credit-card number from the merchant. In this context, SET is very important, as it hides the credit-card details from the merchant.

The way SET hides the cardholder's credit-card details from the merchant is pretty interesting. For this, SET relies on the concept of a digital envelope. The following steps illustrate the idea.

1. The SET software prepares the Payment Information (PI) on the cardholder's computer (which primarily contains the cardholder's credit-card details) exactly the same way as it happens in any Web-based payment system.
2. However, what is specific to SET is that the cardholder's computer now creates a one-time session key.
3. Using this one-time session key, the cardholder's computer now encrypts the payment information.
4. The cardholder's computer now wraps the one-time session key with the public key of the payment gateway to form a digital envelope.
5. It then sends the encrypted payment information (step 3) and the digital envelope (step 5) together to the merchant (who has to pass it on to the payment gateway).

Now, the following points are important. The merchant has access only to the encrypted payment information, so it cannot read it. If it were to read it, it would need to know the one-time session key that was used to encrypt the payment information. However, the one-time session key itself is further encrypted by the payment gateway's public key (to form a digital envelope).

The only way to open the digital envelope, that is to obtain the original one-time session key, is to use the payment gateway's private key. And as we know very well, the whole idea behind a private key is that it must be kept private. So, it is expected that only the payment gateway knows its private key—the merchant does not know it. Therefore, it cannot open the envelope and know the one-time session key, and thus it cannot also decrypt the original payment information.

Thus, SET achieves its objective of hiding the payment details from the merchant using the concept of digital envelope.

6.6.5 SET Internals

Let us discuss the major transactions supported by SET. They are **Purchase Request**, **Payment Authorization**, and **Payment Capture**.

1. Purchase Request

Before the **Purchase Request** transaction begins, the cardholder is assumed to have completed browsing, selecting, and ordering of items. This preliminary phase ends when the merchant sends a completed order form to the customer over the Web. SET is not used in any of these steps. SET begins when the Purchase Request starts. The Purchase Request exchange is made up of four messages: **Initiate Request**, **Initiate Response**, **Purchase Request**, and **Purchase Response**.

Step 1: Initiate Request In order to send SET messages to the merchant, the cardholder must have the digital certificates of the merchant as well as that of the payment gateway. There are three agencies involved: (a) The agency that issues credit cards (the issuer, which is a Financial Institution or FI), (b) Certification Authority (CA), and (c) Payment Gateway (PG), which can be the same as the acquirer. There can be only one, two or three organizations carrying out these functions, as one organization can perform more than one function. However, for the sake of clarity, we shall assume that there are three separate entities, which are described in brief as follows:

- (a) A **Financial Institution (FI)** such as MasterCard or Visa issues credit cards for people to make purchases without making cash payments.
- (b) We have discussed **Certification Authorities (CA)** earlier in detail. They authenticate individuals/organizations and issue digital certificates to them for conducting electronic-commerce transactions. CAs help in ensuring non-fraudulent transactions on the Web.
- (c) **Payment gateways** are third-party payment processors, who process payments on behalf of merchants by tying up with FIs and banks. We have discussed them earlier.

In some cases, the financial institutions outsource the functions of the payment gateway to third parties. Thus, there can be various models for this. The cardholder requests the merchant's certificates in the *Initiate Request* message. The cardholder also sends the name of its credit-card company and an id created by the cardholder for this interaction process, to the merchant in this message as shown in Fig. 6.28.

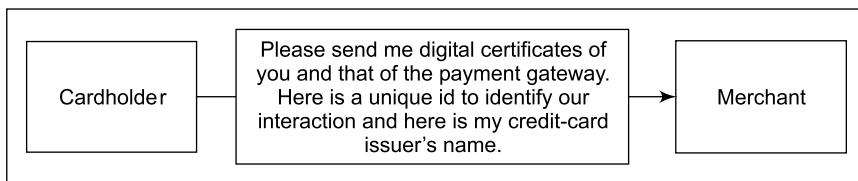


Fig. 6.28 Initiate request

Step 2: Initiate Response The merchant generates a response and signs it with its private key. The response includes a transaction id for this transaction (created by the merchant), the merchant's digital certificate and the payment gateway's digital certificate. This message is called *Initiate Response*, as shown in Fig. 6.29.

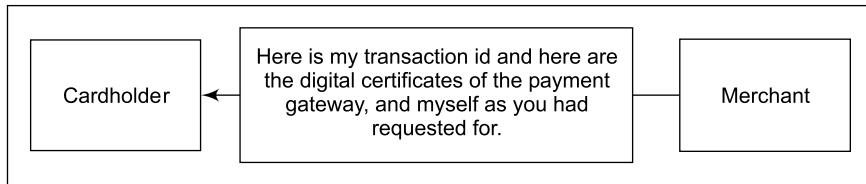


Fig. 6.29 Initiate response

Step 3: Purchase Request The cardholder verifies the digital certificates of the merchant and that of the payment gateway by means of their respective CA signatures and then creates an **Order Information (OI)** and **Payment Information (PI)**. The transaction id created by the merchant is added to both OI and PI. The OI does not contain explicit order details such as item numbers and prices. Instead, it has references to the shopping phase between the customer and the selected merchant (such as order number, transaction date, and card type) that precedes the Purchase Request phase (i.e. using the shopping cart saved in the merchant's database). PI contains details such as credit-card information, purchase amount and order description. The cardholder now prepares the *Purchase Request*. For this, he/she generates a one-time symmetric key (say K). The *Purchase Request* message contains the following:

- (i) *Purchase-related Information* This information is mainly for the payment gateway.
 - (a) It contains (a) PI, (b) digital signature calculated over PI and OI, and (c) OI Message Digest (OIMD), which is the message digest calculated over OI by signing it with the cardholder's private key.
 - (b) All these are encrypted with K.
 - (c) Finally, the digital envelope is created by encrypting K with the payment gateway's public key. The name *envelope* signifies that it must be decrypted first before any other PI can be accessed. The value of K is not made available to the merchant, and therefore, it cannot read any of the payment-related information. Instead, it forwards this to the payment gateway.

(ii) *Order-related Information* The merchant needs this information. It consists of the OI, the signature calculated over PI and OI and the PI Message Digest (PIMD) (which is calculated by encrypting a small portion of the PI with the cardholder's private key). The PIMD is needed by the merchant in order to verify the signature calculated over PI and OI.

(iii) *Cardholder Certificate* This contains the cardholder's public key, required by the merchant as well as by the payment gateway.

This is shown in Fig. 6.30.

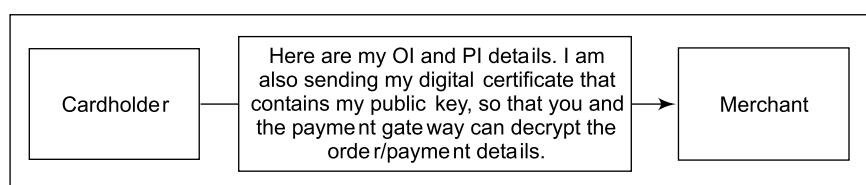


Fig. 6.30 Purchase request

An interesting aspect of this process is the **dual signature**. This ensures that the merchant and the payment gateway receive the information that they require, and yet the cardholder protects the credit-card details from the merchant. The concept is shown in Fig. 6.31.

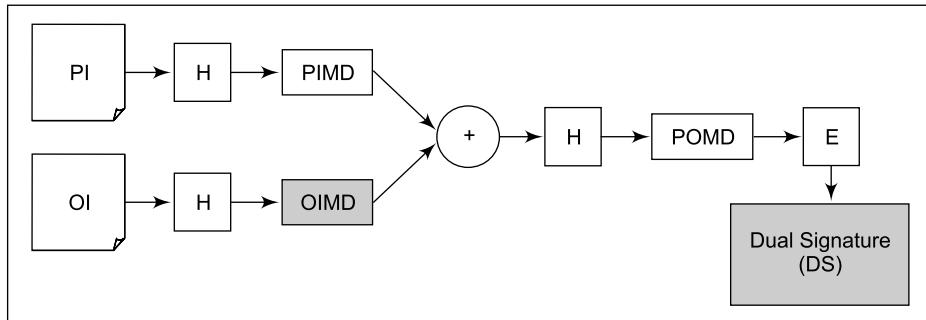


Fig. 6.31 Dual signature

Let us describe this process in brief, at the cost of some repetition.

- The cardholder performs a message digest or hash (H) on the PI to generate PIMD. The cardholder also hashes OI to generate OIMD. The cardholder then combines PIMD and OIMD, and hashes them together to form POMD. It then encrypts the POMD with its own private key to generate the *Dual Signature (DS)*. The POMD is available to both the merchant and the payment gateway.
- The cardholder sends the merchant the OI, DS and PIMD. Note that the merchant must not get PI (we shall see how the cardholder achieves it, soon). Using these pieces of information, the merchant verifies that the order came from the cardholder, and not from someone posing as the cardholder. For this, the merchant performs the actions as shown in Fig. 6.32.
- The payment gateway gets PI, DS and OIMD. Note that the payment gateway need not get OI. Using these, the payment gateway can verify POMD. This verification satisfies the payment gateway that the payment information came from the cardholder, and not from someone posing as the cardholder. For this purpose, the payment gateway performs the actions as shown in Fig. 6.33.

An important question now is: How does the cardholder protect the payment information from the merchant? For this, the cardholder performs the following process:

- Cardholder creates PI, DS and OIMD and encrypts the whole thing with a one-time session key K .
- Cardholder then encrypts session key K with the payment gateway's public key.
- These two together form a digital envelope.
- The cardholder sends the digital envelope to the merchant, instructing it to forward it to the payment gateway. Since the merchant does not have the private key of the payment gateway, it cannot decrypt the envelope and obtain the payment details.

Step 4: Purchase Response When the merchant receives the Purchase Request, it does the following:

- (a) Verifies the cardholder's certificates by means of its CA signatures.
- (b) Verifies the signature created over PI and OI using the cardholder's public key (which is a part of the cardholder's digital certificate). This ensures that the order has not been tampered with while in transit, and that it was signed using the cardholder's private key.

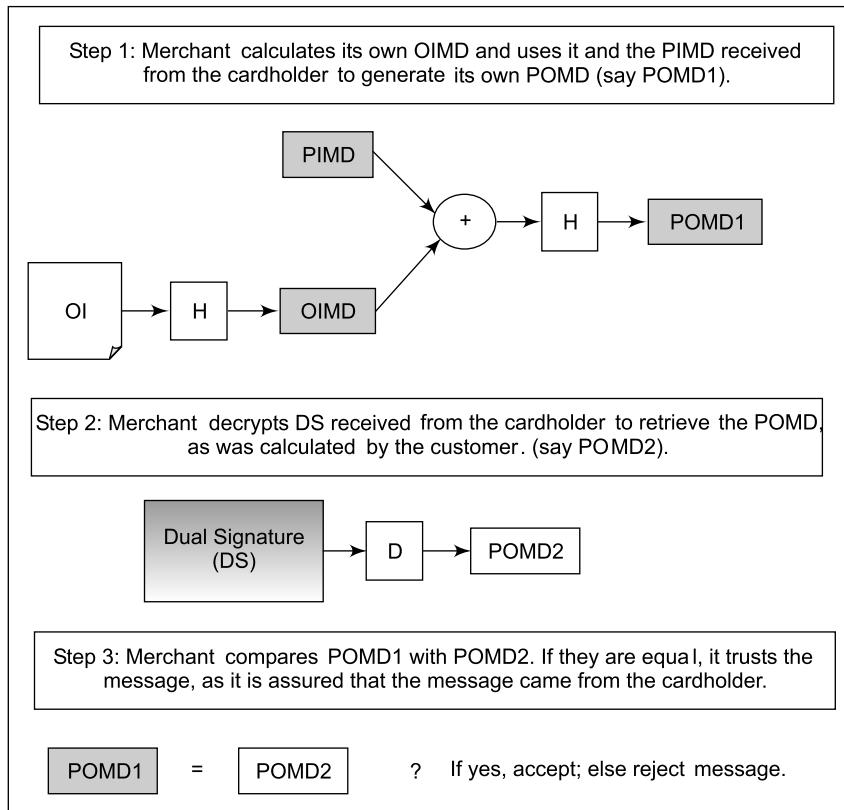


Fig. 6.32 Verification of cardholder's authenticity by the merchant

- (c) Processes the order and forwards the Payment Information (PI) to the payment gateway for authorization (discussed later).
- (d) Sends a *Purchase Response* back to the cardholder, as shown in Fig. 6.34.

The *Purchase Response* message includes a message acknowledging the order and references of the corresponding transaction number. The merchant signs the message using its private key. The message and its signature are sent along with the merchant's digital certificate to the cardholder. When the cardholder software receives the *Purchase Response* message, it verifies the merchant's certificate and then takes some action, such as displaying a message to the user.

2. Payment Authorization

This process ensures that the issuer of the credit card approved the transaction. The **Payment Authorization** step happens when the merchant sends the payment details to the payment gateway. The payment gateway verifies these details and authorizes the payment, which ensures that the merchant will receive payment. Therefore, the merchant can provide the services or goods to the cardholder, as ordered. The *Payment Authorization* exchange consists of two messages: **Authorization Request** and **Authorization Response**.

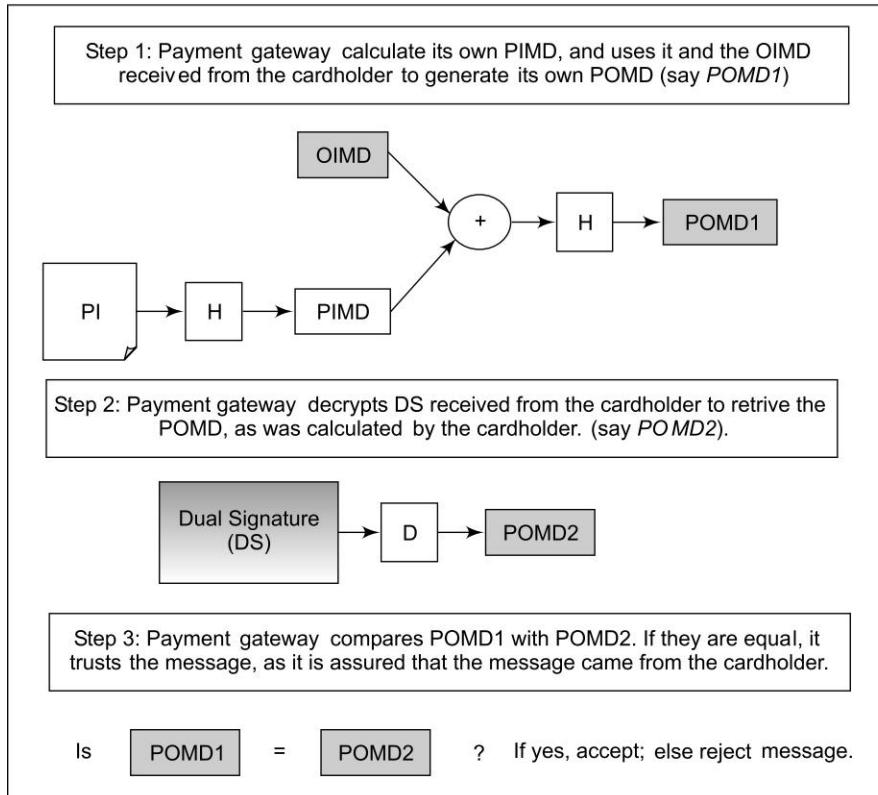


Fig. 6.33 Verification of cardholder's authenticity by the payment gateway

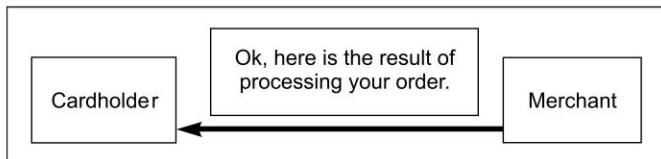


Fig. 6.34 Purchase Response

Step 1: Authorization Request The merchant sends an *Authorization Request* to the payment gateway, which consists of the following steps:

(a) *Purchase-related Information* This information is obtained by the merchant from the cardholder and includes the PI, the signature calculated over PI and OI (signed with the cardholder's private key), the OI Message Digest (OIMD) and the digital envelope, as discussed earlier.

(b) *Authorization-related Information* This information is generated by the merchant and consists of the transaction id signed with the merchant's private key and encrypted with a one-time symmetric key generated by the merchant and the digital envelope.

(e) Certificates The merchant also sends the cardholder's digital certificate needed for verifying the cardholder's digital signature and the merchant's digital certificate needed for verifying the merchant's digital signature.

This is shown in Fig. 6.35.

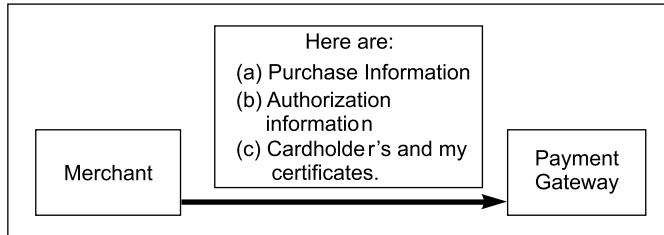


Fig. 6.35 Authorization request

As a result, the payment gateway performs the following tasks:

- It verifies all certificates.
- It decrypts the digital envelope to obtain the symmetric key and then decrypts the authorization block.
- It verifies the merchant's signature on the authorization information received from the merchant.
- Performs steps 2 and 3 for the payment information received from the cardholder (PI).
- Matches the transaction id received from the merchant with the transaction id received from the PI (indirectly) from the cardholder.
- Requests and receives an authorization from the credit-card issuer (i.e. the cardholder's bank) for the payment from the cardholder to the merchant.

Step 2: Authorization Response Having obtained authorization from the issuer, the payment gateway returns an *Authorization Response* message to the merchant. This message contains the following:

(a) *Authorization-related Information* This includes an authorization block that is signed with the gateway's private key and encrypted with a one-time symmetric key generated by the gateway. It also includes a digital envelope that contains the one-time key encrypted with the merchant's public key.

(b) *Capture Token Information* This information would be used for effecting the payment transaction later. The basic structure of this piece of information is the same as the authorization-related information. This token is not processed by the merchant, and is instead, passed back to the customer as it is.

(c) *Certificate* The gateway's digital certificate is also included in the message.

This is shown in Fig. 6.36.

With this authorization from the payment gateway, the merchant can provide the goods or services to the cardholder.

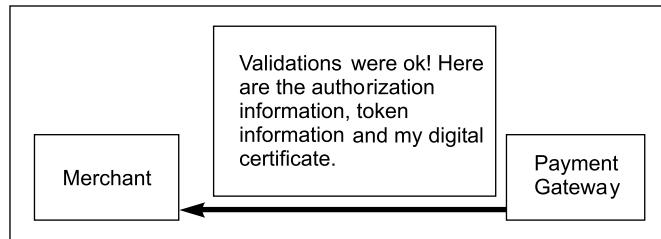


Fig. 6.36 Authorization response

3. Payment Capture

For obtaining payment, the merchant engages the payment gateway in a **Payment Capture** transaction. It also contains two messages: **Capture Request** and **Capture Response**.

Step 1: Capture Request Here, the merchant generates, signs and encrypts a *Capture Request* block that includes the payment amount and the transaction id. This message also includes the encrypted capture token received earlier (in the *Authorization Response* transaction), the merchant's digital signature, and digital certificate.

When the payment gateway receives the *Capture Request* message, it decrypts and verifies the *Capture Request* block, and decrypts and verifies the capture token as well. It then checks for consistency between the *Capture Request* and the capture token. It then creates a clearing request that is sent to the issuer over the private payment network. This request results into a funds transfer to the merchant's account.

This is shown in Fig. 6.37.

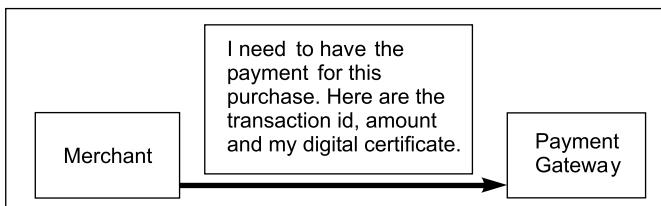


Fig. 6.37 Capture request

Step 2: Capture Response In this message, the payment gateway notifies the merchant of the payment. The message includes a *Capture Response* block, which is signed and encrypted by the payment gateway. The message also includes the payment gateway's digital certificate. The merchant software processes this message and stores the information therein for reconciliation with the payment received from the bank. This is shown in Fig. 6.38.

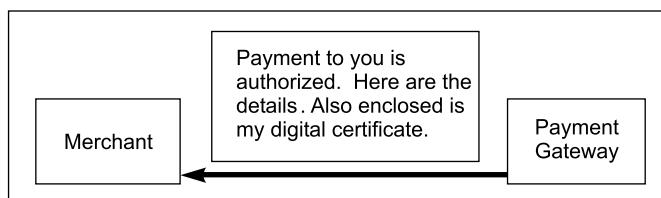


Fig. 6.38 Capture response

6.6.6 SET Conclusions

From the discussion, it should become clear that although SSL and SET are both used for facilitating secure exchange of information, their purposes are quite different. Whereas SSL is primarily used for secure exchange of information of any kind between only two parties (a client and a server), SET is specifically designed for conducting e-commerce transactions. SET involves a third party called a payment gateway, which is responsible for issues such as credit-card authorization, payment to the merchant, etc. This is not the case with SSL. SSL primarily deals with encryption and decryption of information between two parties. It does not specify how payment would be made. The architecture of SET ensures this as well.

6.6.7 SET Model

Having looked at the detailed processing involved in SET, let us summarize the concepts learnt by studying the overall processing model of SET. As we have studied, the authentication provided by SET is pretty strong. In order that the identification and verification of the customer (cardholder), merchant and the payment gateway are ensured, the SET protocol requires that all the parties involved in this should have a valid digital certificate and that they use digital signatures. This means that all the three concerned parties must have a valid digital certificate from an approved certification authority.

Let us discuss a simple model for implementing SET. Note that this can be with some other approaches as well. However, here we are interested only in trying to understand how a typical set-up for SET might look like. First, let us take a look at Fig. 6.39.

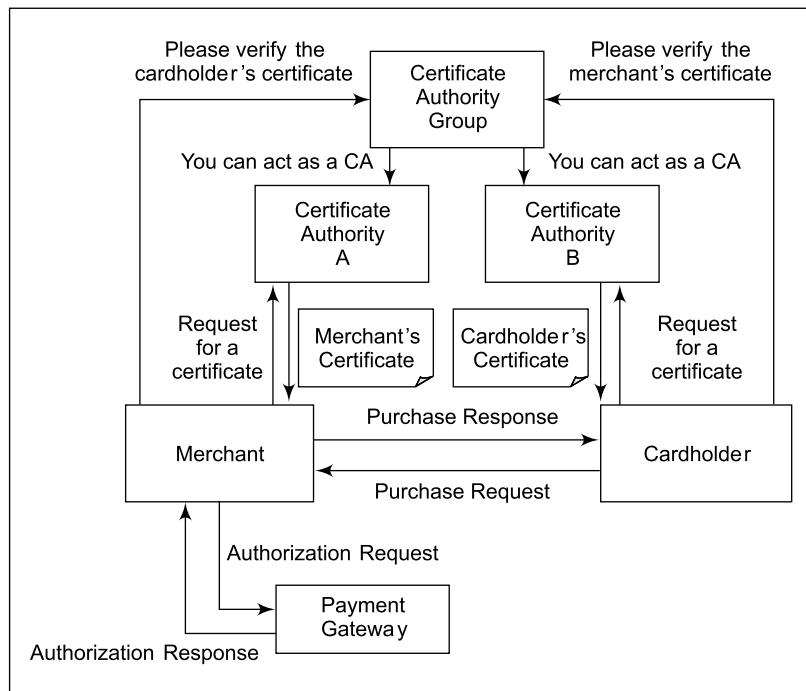


Fig. 6.39 The SET model

The figure shows the (simplified) SET model for a typical purchase transaction. The three main parties involved in the actual transaction are, of course, the customer, the merchant and the payment gateway. The merchant and the customer make requests for their respective certificates. Interestingly, we have shown two different certification authorities. Of course, it is very much possible that both the merchant and the customer receive certificates from the same certification authority.

In general, the certificate to a customer is issued by the bank or the credit card company who has issued the card to the customer, or sometimes also by a third-party agency representing the credit-card company.

On the other hand, a financial institution, also called an *acquirer*, issues a merchant's certificate. An acquirer is usually a financial institution such as MasterCard or Visa (or their appointed agencies), who can authorize payments made by their brands of credit cards. Therefore, a merchant needs to have as many certificates as the number of different brands of credit cards that it accepts (e.g. one for MasterCard, one for Visa, one for Amex, and so on). Thus, when a customer receives a merchant certificate, it is also assured that the merchant is authorized to accept payments for that brand of credit card. This is similar to the boards displayed by real-life stores and restaurants that they accept certain credit cards.

As discussed, the transactions between a customer and merchant are for purchases, and those between the merchant and the payment authority are for authorization of payment. This is described in detail earlier.

■ 6.7 SSL VERSUS SET ■

Having discussed SSL and SET in detail, let us take a quick look at the differences between them, as shown in Fig. 6.40.

Issue	SSL	SET
Main aim	Exchange of data in an encrypted form	E-commerce related payment mechanism
Certification	Two parties exchange certificates	All the involved parties must be certified by a trusted third party
Authentication	Mechanisms in place, but not very strong	Strong mechanisms for authenticating all the parties involved
Risk of merchant fraud	Possible, since customer gives financial data to merchant	Unlikely, since customer gives financial data to payment gateway
Risk of customer fraud	Possible, no mechanisms exist if a customer refuses to pay later	Customer has to digitally sign payment instructions
Action in case of customer fraud	Merchant is liable	Payment gateway is liable
Practical usage	High	Low at the moment, expected to grow

Fig. 6.40 SSL versus SET

This table should give us an idea that SET is a standard that describes a very complex authentication mechanism that makes it almost impossible for either party to commit any sort of fraud. However, there

is no such mechanism in SSL. In SSL, data is exchanged securely. However, the customer provides critical data such as credit-card details to a merchant, and hopes that the customer does not misuse them. This is not possible in SET. Also, in the case of SSL, a merchant believes that the credit card really belongs to the customer, and that he/she is not using a stolen card. In the case of SET, this is very unlikely, and even if it happens, the merchant is safe, since the payment gateway has to ensure that the customer is not committing fraud. The whole point is, whereas SSL was created for exchanging secure messages over the Internet, SET was specifically designed for secure e-commerce transactions involving online payment. So, these differences should not surprise anybody.

■ 6.8 3-D SECURE PROTOCOL ■

In spite of its advantages, SET has one limitation: it does not prevent a user from providing someone else's credit-card number. The credit-card number is protected from the merchant. However, how can one prevent a customer from using another person's credit-card number? That is not achieved in SET. Consequently, a new protocol developed by Visa has emerged, called **3-D Secure**.

The main difference between SET and 3-D Secure is that any cardholder who wishes to participate in a payment transaction involving the usage of the **3-D Secure** protocol has to enroll on the issuer bank's *Enrollment Server*. That is, before a cardholder makes a card payment, he/she must enroll with the issuer bank's enrollment server. This process is shown in Fig. 6.41.

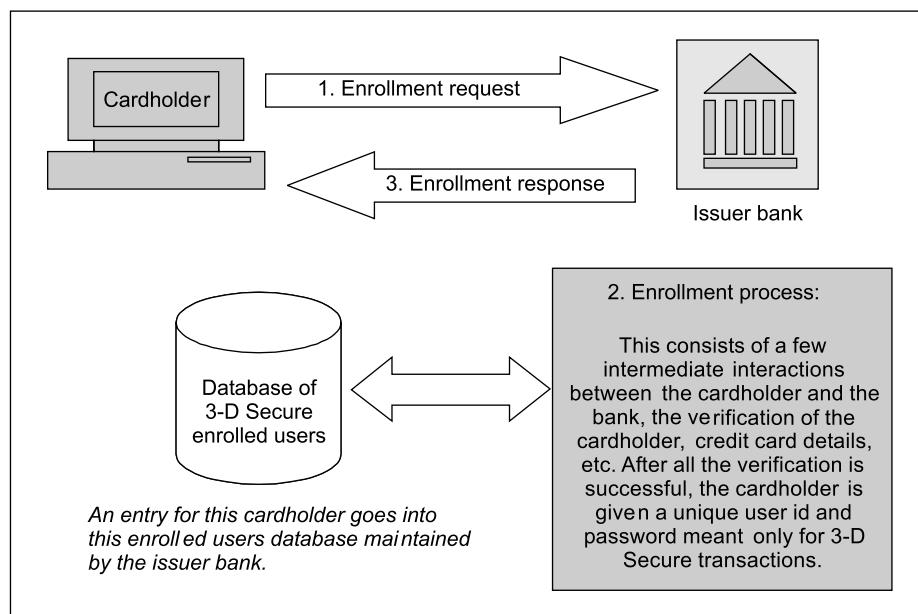


Fig. 6.41 User enrollment

At the time of an actual 3-D Secure transaction, when the merchant receives a payment instruction from the cardholder, the merchant forwards this request to the issuer bank through the Visa network. The issuer bank requires the cardholder to provide the user id and password that were created at the time of user-enrollment process. The cardholder provides these details, which the issuer bank verifies

against its 3-D Secure enrolled users database (against the stored card number). Only after the user is authenticated successfully, does the issuer bank inform the merchant that it can accept the card-payment instruction.

6.8.1 Protocol Overview

Let us understand how the 3-D Secure protocol works, step-by-step.

Step 1

The user shops using the shopping cart on the merchant site, and decides to pay the amount. The user enters the credit-card details for this purpose, and clicks on the OK button, as shown in Fig. 6.42.

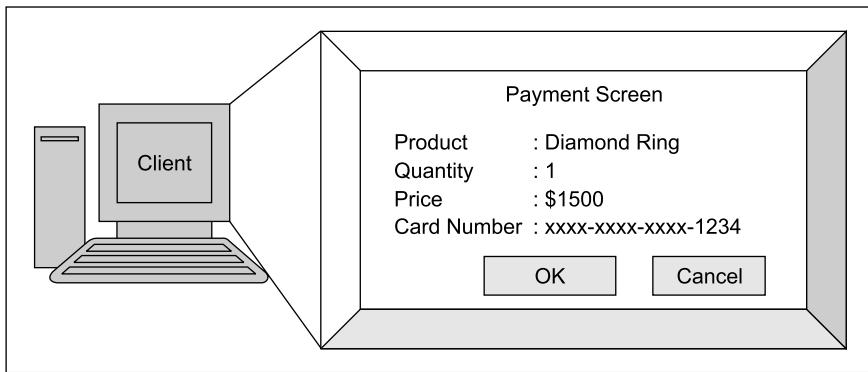


Fig. 6.42 Step 1 in 3-D Secure

Step 2

When the user clicks on the *OK* button, the user will be redirected to the issuer bank's site. The bank site will produce a pop-up screen, prompting the user to enter the password provided by the issuer bank. This is shown in Fig. 6.43. The bank (issuer) authenticates the user by the mechanism selected by the user earlier. In this case, we consider a simple static id and password-based mechanism. Newer trends involve sending a number to the user's mobile phone and asking the user to enter that number on the screen. However, that falls outside of the purview of the 3-D Secure protocol.

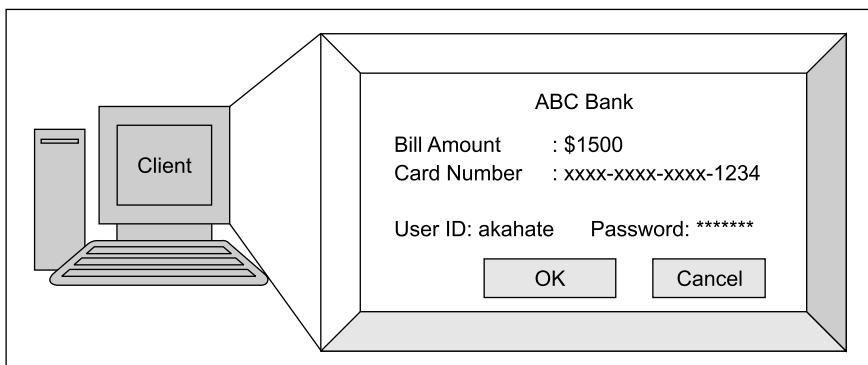


Fig. 6.43 Step 2 in 3-D Secure

At this stage, the bank verifies the user's password by comparing it with its database entry. The bank sends an appropriate success/failure message to the merchant, based on which the merchant takes an appropriate decision, and shows the corresponding screen to the user.

6.8.2 Happenings Behind the Scene

Figure 6.44 depicts the internal operations of 3-D Secure. The process uses SSL for confidentiality and server authentication.

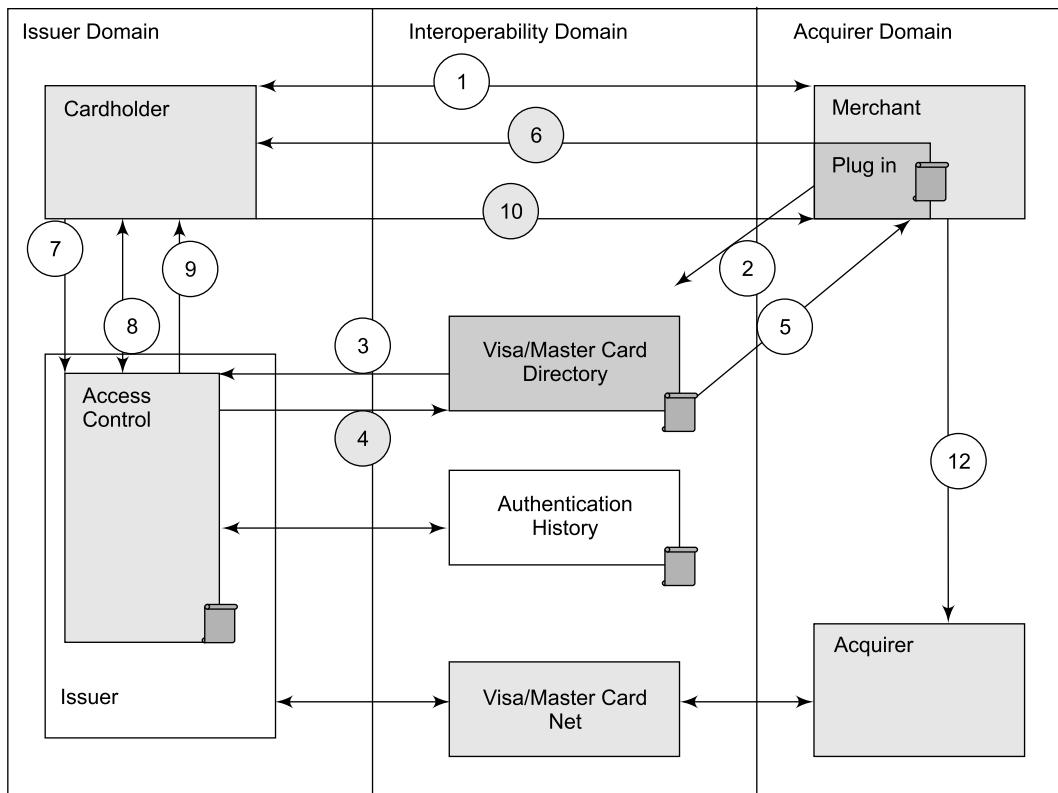


Fig. 6.44 3-D Secure internal flow

The flow can be described as follows.

1. The customer finalizes on the payment on the merchant site (the merchant has all the data of this customer).
2. A program called *merchant plug-in*, which resides at the merchant Web server, sends the user information to the Visa/MasterCard directory (which is LDAP-based).
3. The Visa/MasterCard directory queries the access control server running at the issuer bank (i.e. the customer's bank), to check the authentication status of the customer.
4. The access-control server forms the response for the Visa directory and sends it back to the Visa/MasterCard directory.

5. The Visa/MasterCard directory sends the payer's authentication status to the merchant plug-in.
6. After getting the response, if the user is currently not authenticated, the plug-in redirects the user to the bank site, requesting the bank or the issuer site to perform the authentication process.
7. The access-control server (running on the bank's site) receives the request for authentication of the user.
8. The authentication server performs authentication of the user based on the mechanism of authentication chosen by the user (e.g. password, dynamic password, mobile etc.)
The access control server sends the information to the repository where the history of the user authentication is kept for legal purpose.
9. The plug-in receives the response of the access-control server through the user's browser. This contains the digital signature of the access-control server.
10. The plug-in validates the digital signature of the response and the response from the access control server.
11. If the authentication was successful and the digital signature of the access-control server is validated, the merchant sends the authorization information to its bank (i.e. the acquire bank).

■ 6.9 EMAIL SECURITY ■

6.9.1 Introduction

Electronic mail (email) is perhaps the most widely used application on the Internet. Using email, an Internet user can send a message (and these days, pictures, video, sound, etc.) to other Internet user(s). Consequently, the security of email messages has become an extremely important issue. Before we study email security, let us quickly review the email technology in brief.

RFC 822 defines a format for text email messages. An email message is considered to be made up of two portions: its *contents* (or *body*) and *headers*. This is very similar to the way our manual postal system works. There also, we have letters (similar to email contents) and envelopes (similar to email headers). Therefore, an email message consists of a number of header lines followed by the actual message contents. A header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments. Examples of header keywords are *From*, *To*, *Subject* and *Date*.

Figure 6.45 shows an email message, and distinguishes between its headers and contents.

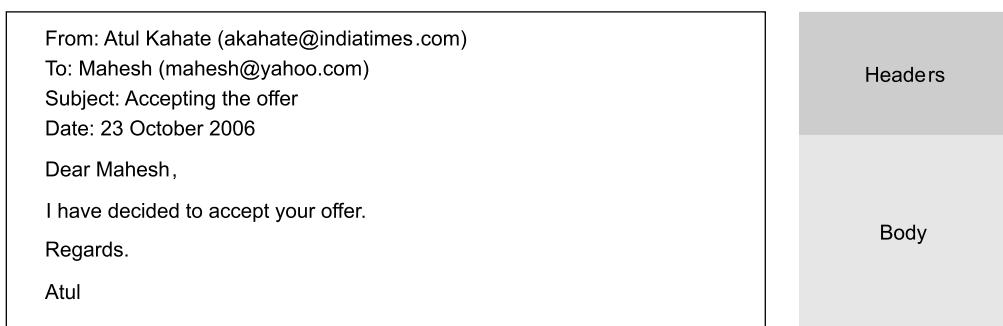


Fig. 6.45 Email headers and body

The **Simple Mail Transfer Protocol (SMTP)** is used for email communications. The email software at the sender's end gives the email message to the local SMTP server. This SMTP server actually transfers the email message from the SMTP server of the receiver. Its main job is to carry the email message between the sender and the receiver. This is shown in Fig. 6.46. Of course, it uses the TCP/IP protocol underneath. That is, SMTP runs on top of TCP/IP (in the application layer).

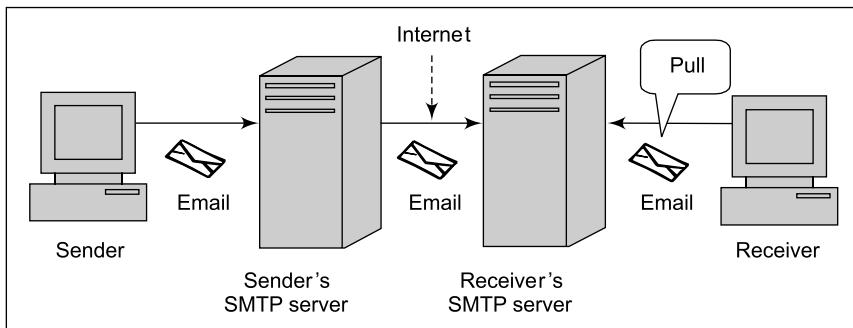


Fig. 6.46 Email using the SMTP protocol

The basic phases of an email communication consists of the following steps:

1. At the sender's end, an SMTP server takes the message sent by a user's computer.
2. The SMTP server at the sender's end then transfers the message to the SMTP server of the receiver.
3. The receiver's computer then pulls the email message from the SMTP server at the receiver's end, using other email protocols such as *Post Office Protocol (POP)* or *Internet Mail Access Protocol (IMAP)*, which we need not discuss here.

SMTP is actually quite simple. The communication between a client and a server using SMTP consists of human-understandable ASCII text. We shall first describe the steps and then list the actual interaction steps. Note that although we describe the communication between the two SMTP servers, the sender's SMTP server assumes the role of a client, whereas the receiver's SMTP server assumes the role of the server.

1. Based on the client's request for an email message transfer, the server sends back a *READY FOR MAIL* reply, indicating that it can accept an email message from the client.
2. The client then sends a *HELO* (abbreviation of *HELLO*) command to the server, and identifies itself.
3. The server then sends back an acknowledgement in the form of its own DNS name.
4. The client can now send one or more email messages to the server. The email transfer begins with a *MAIL* command that identifies the sender.
5. The recipient allocates buffers to store the incoming email message, and sends back an *OK* response to the client. The server also sends back a return code of 250, which essentially means *OK*. The reason both *OK* and a return code of 250 are sent back is to help both humans and application programs understand the server's intentions (humans prefer *OK*, application programs prefer a return code such as 250).

6. The client now sends the list of the intended recipients of the email message by one or more RCPT commands (one per recipient). The server must send back a 250 OK or 550: *No such user here* reply back to the client for each recipient.
7. After all RCPT commands, the client sends a *DATA* command, informing the server that the client is ready to start transmission of the email message.
8. The server responds back with a 354 *Start mail input* message, indicating that it is ready to accept the email message. It also tells the client what identifiers it should send to signify that the message is over.
9. The client sends the email message and when it is over, sends the identifier provided by the server to indicate that its transmission is over.
10. The server sends back a 250 OK response.
11. The client sends a *QUIT* command to the server.
12. The server sends back a 221 *Service closing transmission channel* message, indicating that it is also closing its portion of the connection.

The actual interaction between the client and the server as described above, is shown in Fig. 6.47. Here, a user, Atul at host yahoo.com, sends an email to users Ana and Juhi at host hotmail.com. The SMTP client software on the host yahoo.com establishes a TCP connection with the SMTP server software on the host hotmail.com (not shown in the figure). Thereafter, the exchange of messages happens as shown below (S indicates messages sent by the server to the client, and C indicates messages from the client to the server). Also note that the server has informed the client that it would like to see a <CR><LF><LF> identifier when the client's transmission is over. Accordingly, the client sends this when its transmission is over.

```
S: 220 hotmail.com Simple Mail Transfer Service Ready
C: HELO yahoo.com
S: 250 hotmail.com

C: MAIL FROM: <Atul@yahoo.com>
S: 250 OK

C: RCPT TO: <Ana@hotmail.com>
S: 250 OK

C: RCPT TO: <Juhi@hotmail.com>
S: 250 OK

C: DATA
S: 354 Start mail input; end with <CR><LF><LF>
C: ... actual contents of the message ...
C: ....
C: ....
C: <CR><LF><LF>
S: 250 OK

C: QUIT
S: 221 hotmail.com Service closing transmission channel
```

Fig. 6.47 Example of an email message using SMTP protocol

Having discussed the basic concepts of email communication, let us now study the three main email security protocols: **Privacy Enhanced Mail (PEM)**, **Pretty Good Privacy (PGP)** and **Secure MIME (S/MIME)**.

6.9.2 Privacy Enhanced Mail (PEM)

1. Introduction

The **Privacy Enhanced Mail (PEM)** is an email security standard adopted by the Internet Architecture Board (IAB) to provide secure electronic mail communication over the Internet. PEM was initially developed by the Internet Research Task Force (IRTF) and Privacy Security Research Group (PSRG). They then handed over the PEM standard to the Internet Engineering Task Force (IETF) PEM Working Group. PEM is described in four specification documents, which are RFC numbers 1421 to 1424. PEM supports the three main cryptographic functions of encryption, non-repudiation, and message integrity, as shown in Fig. 6.48.

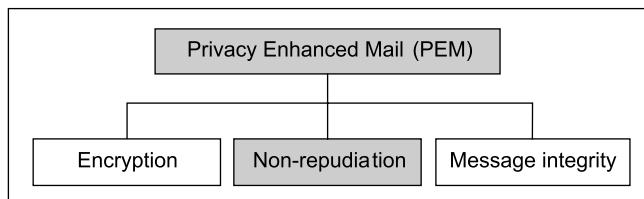


Fig. 6.48 Security features offered by PEM

2. The Working of PEM

The broad-level steps in PEM are illustrated in Fig. 6.49. As shown, PEM starts with a **canonical conversion**, which is followed by digital signature, then by encryption and finally, **Base-64 encoding**.

PEM allows for three security options when sending an email message. These options are

- Signature only (Steps 1 and 2)
- Signature and Base-64 encoding (Steps 1, 2 and 4)
- Signature, Encryption and Base-64 encoding (Steps 1 to 4)

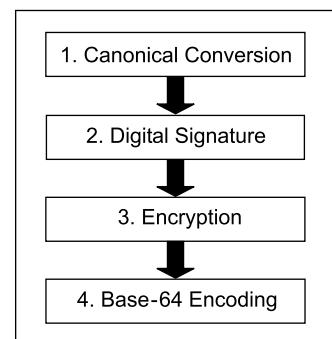


Fig. 6.49 PEM operations

Let us now discuss the four steps shown earlier in the figure. Note that the receiver has to perform these four steps in the reverse direction to retrieve the original plain-text email message.

Step 1: Canonical Conversion There is a distinct possibility that the sender and the receiver of an email message use computers that have different architectures and operating systems. This is because the Internet works on any computer that has a TCP/IP stack, regardless of its architecture or operating system. Therefore, it is quite possible that the same thing is represented differently in these different computers. For example, in the MS-DOS operating system, a *new line* (i.e. the result of the keyboard's *Enter* key) is represented by two characters, which is not the case in an operating system such as Unix, wherein it is a single character. This can create problems when creating message digests, and therefore, digital signatures. For instance, the message digest for an email message composed on

a computer that runs MS-DOS can differ from that on a computer that runs Unix, as the input for the message-digest creation is not the same in the two cases.

Consequently, PEM transforms each email message into an abstract, canonical representation. This means that regardless of the architecture and the operating system of the sending and the receiving computers, the email message always travels in a uniform, independent format.

Step 2: Digital Signature This is a typical process of digital signature, which we have studied many times before. It starts by creating a message digest of the email message using an algorithm such as MD2 or MD5, as shown in Fig. 6.50.

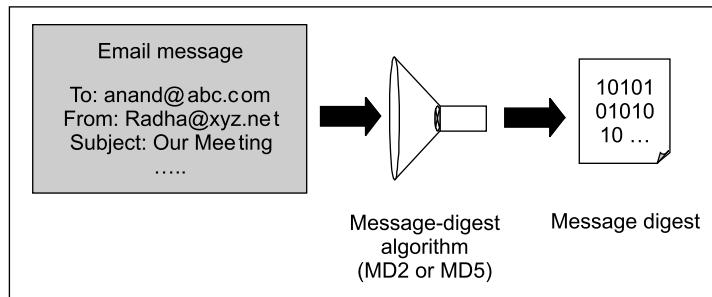


Fig. 6.50 Message-digest creation of the original email message

The message digest thus created is then encrypted with the sender's private key to form the sender's digital signature. This process is shown in Fig. 6.51.

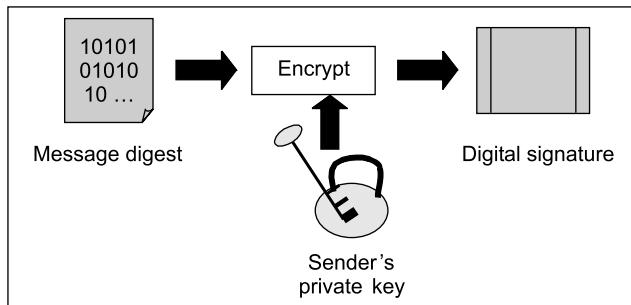
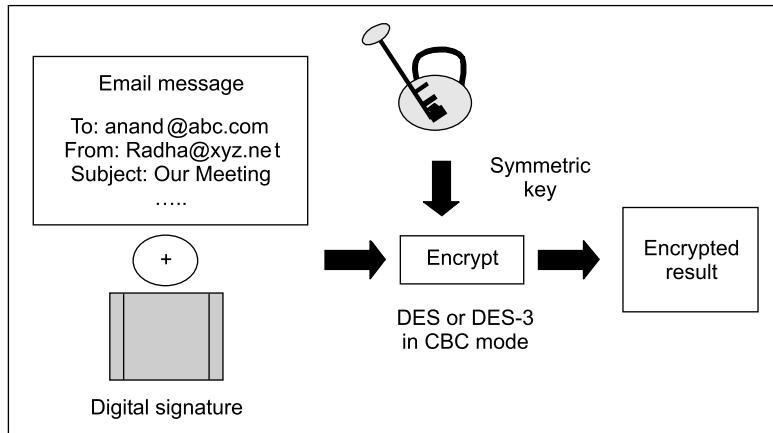
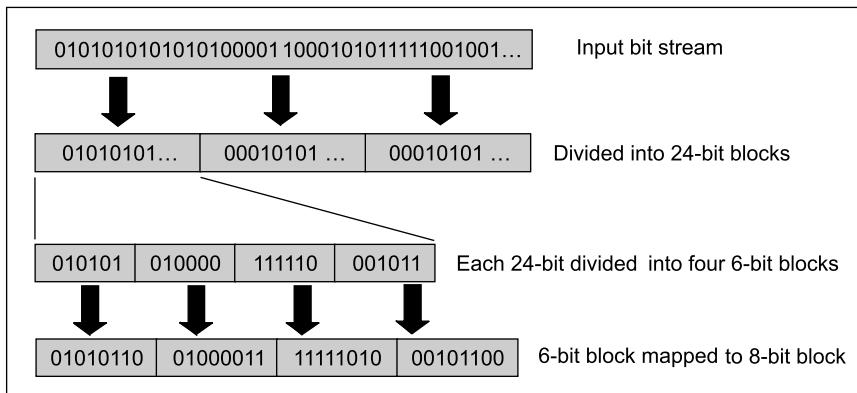


Fig. 6.51 Creation of the sender's digital signature over the email message

Step 3: Encryption In this step, the original email and the digital signature are encrypted together with a symmetric key. For this, the DES or DES-3 algorithm in CBC mode is used. This is shown in Fig. 6.52.

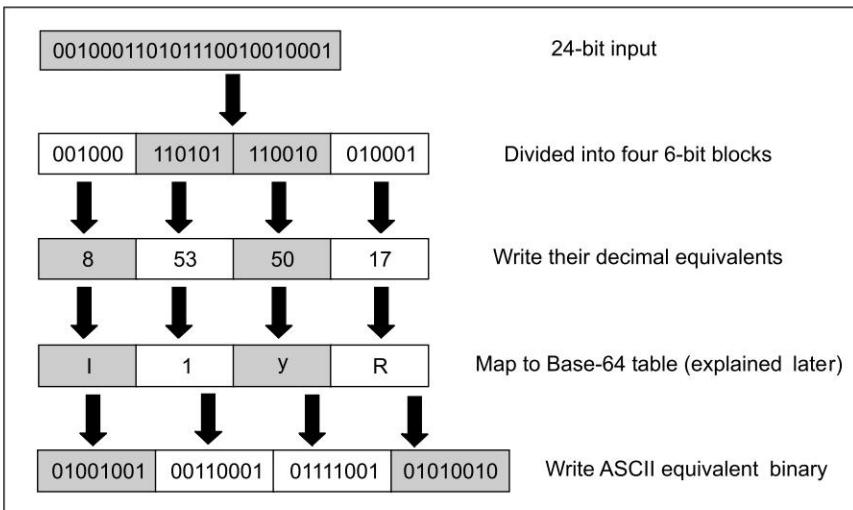
Step 4: Base-64 Encoding This is the last step in PEM. The Base-64 encoding (also called **Radix-64 encoding** or **ASCII armour**) process transforms arbitrary binary input into printable character output. In this technique, the binary input is processed in blocks of 3 octets, or 24 bits. These 24 bits are considered to be made up of 4 sets, each of 6 bits. Each such set of 6 bits is mapped into an 8-bit output character in this process. This concept is shown in Fig. 6.53. Note that the values in this figure are not correctly used—they are shown merely for introducing the concepts.

**Fig. 6.52** Encryption in PEM**Fig. 6.53** Base-64 encoding concept

This seems to be a fairly straightforward process. However, one key question is, what is the logic used for mapping a 6-bit input block into an output 8-bit block? For this, a mapping table is used, as explained in the example below.

In our example of Base-64 encoding, let us consider a 24-bit raw stream as 001000110101110010010001. The Base-64 encoding process for this stream is illustrated in Fig. 6.54, and explained after the figure.

The process shown in the figure for is quite straightforward. Therefore, we need not describe it. The only aspect that we shall touch upon is mapping to Base-64 table. What happens here is, a standard pre-defined table is used, as shown in Fig. 6.55. The decimal number generated is looked up into this table. The character found at the position specified by the decimal number in this table is mentioned in the output. For example, in our example, the first decimal number is 8, and the 8th position in our mapping table indicates a character I. Similarly, the second position specifies the number 53. In our mapping table, we see that the character 1 is found in the 53rd position, and so on. Finally, the binary equivalent corresponding to the 8-bit ASCII of this character is written.

**Fig. 6.54** Base-64 encoding example

6-bit value	Character						
0	A	16	Q	32	G	48	w
1	B	17	R	33	H	49	x
2	C	18	S	34	I	50	y
3	D	19	T	35	J	51	z
4	E	20	U	36	K	52	0
5	F	21	V	37	L	53	1
6	G	22	W	38	M	54	2
7	H	23	X	39	N	55	3
8	I	24	Y	40	O	56	4
9	J	25	Z	41	P	57	5
10	K	26	a	42	Q	58	6
11	L	27	B	43	R	59	7
12	M	28	C	44	S	60	8
13	N	29	D	45	T	61	9
14	O	30	E	46	u	62	+
15	P	31	F	47	V	63	/
						(Padding)	=

Fig. 6.55 Base-64 encoding mapping table

6.9.3 Pretty Good Privacy (PGP)

Phil Zimmerman is the father of the **Pretty Good Privacy (PGP)** protocol. He is credited with the creation of PGP. The most significant aspects of PGP are that it supports the basic requirements of cryptography, is quite simple to use, and is completely free, including its source code and documentation.

Moreover, for those organizations that require support, a low-cost commercial version of PGP is available from an organization called Viacrypt (now Network Associates). PGP has become extremely popular and is far more widely used, as compared to PEM. The email cryptographic support offered by PGP is shown in Fig. 6.56.

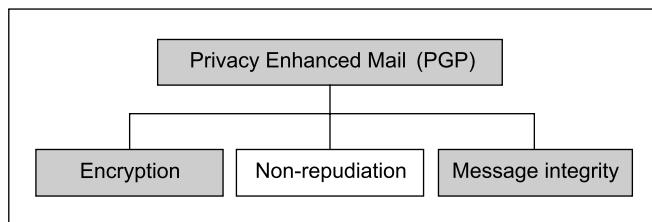


Fig. 6.56 Security features offered by PGP

1. The Working of PGP

In PGP, the sender of the message needs to include the identifiers of the algorithm used in the message, along with the value of the keys. The broad-level steps in PEM are illustrated in Fig. 6.57. As shown, PGP starts with a digital signature, which is followed by compression, then by encryption, then by digital enveloping and finally, by Base-64 encoding.

PGP allows for four security options when sending an email message. These options are

- Signature only (Steps 1 and 2)
- Signature and Base-64 encoding (Steps 1, 2 and 5)
- Signature, Encryption, Enveloping, and Base-64 encoding (Steps 1 to 5)

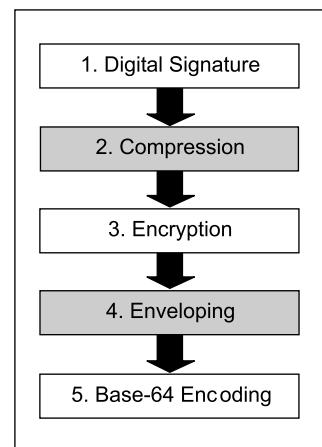


Fig. 6.57 PGP operations

Let us discuss the five steps in PGP now. Note that the receiver has to perform these four steps in the reverse direction to retrieve the original plain text email message.

Step 1: Digital Signature This is a typical process of digital signature, which we have studied many times before. In PGP, it consists of the creation of a message digest of the email message using the SHA-1 algorithm. The resulting message digest is then encrypted with the sender's private key. The result is the sender's digital signature. We shall not repeat that discussion here.

Step 2: Compression This is an additional step in PGP. Here, the input message as well as the digital signature are compressed together to reduce the size of the final message that will be transmitted. For this, the famous ZIP program is used. ZIP is based on the **Lempel–Ziv algorithm**.

The *Lempel–Ziv algorithm* looks for repeated strings or words, and stores them in variables. It then replaces the actual occurrence of the repeated word or string with a pointer to the corresponding variable. Since a pointer requires only a few bits of memory as compared to the original string, this method results in the data being compressed.

For instance, consider the following string:

What is your name? My name is Atul.

Using the Lempel–Ziv algorithm, we would create two variables, say *A* and *B* and replace the words *is* and *name* by pointers to *A* and *B*, respectively. This is shown in Fig. 6.58.

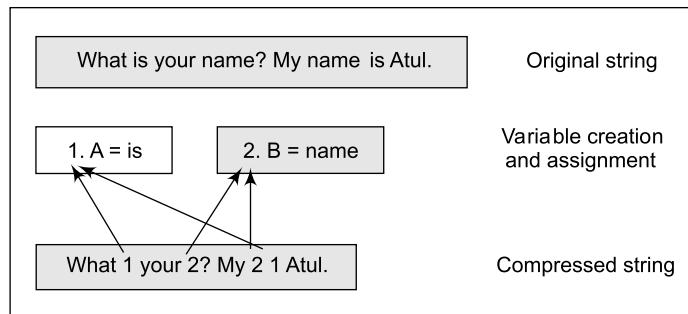


Fig. 6.58 Lempel-Ziv algorithm, as used by the ZIP program

As we can see, the resulting string *What 1 your 2? My 2 1 Atul.* Is smaller compared to the original string *What is your name? My name is Atul.* Of course, the bigger the original string, the better the compression gets. The same process works for PGP.

Step 3: Encryption In this step, the compressed output of step 2 (i.e. the compressed form of the original email and the digital signature together) are encrypted with a symmetric key. For this, generally the IDEA algorithm in CFB mode is used. We shall not describe this process, as we have already discussed it in great detail for PEM.

Step 4: Digital Enveloping In this case, the symmetric key used for encryption in step 3 is now encrypted with the receiver's public key. The output of step 3 and step 4 together form a digital envelope, as we had discussed earlier. This is shown in Fig. 6.59.

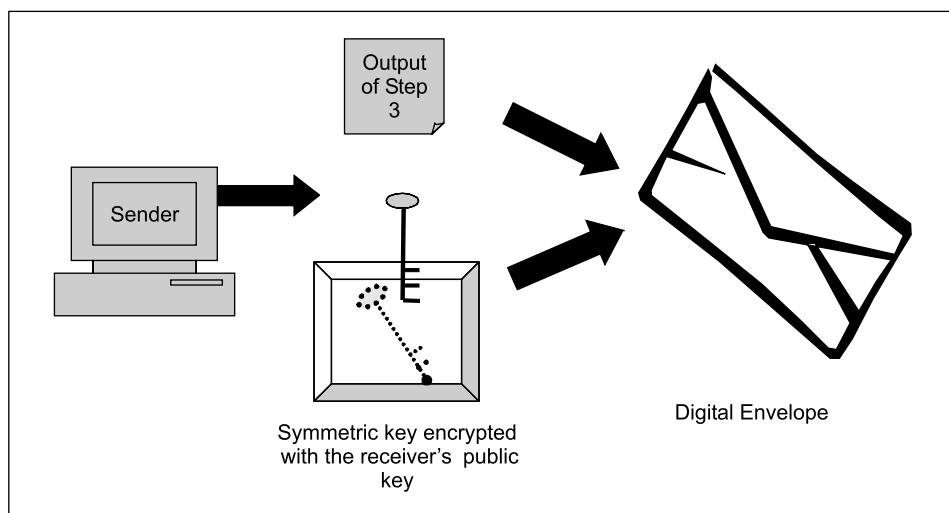


Fig. 6.59 Formation of digital envelope

Step 5: Base-64 encoding The output of step 4 is Base-64 encoded now, as described earlier. We will not repeat that description here.

2. PGP Algorithms

PGP supports a number of algorithms. The most common of them are listed in Fig. 6.60.

Algorithm type	Description
Asymmetric key	RSA (Encryption and signing, Encryption only, Signing only)
	DSS (Signing only)
Message digest	MD5, SHA-1, RIPE-MD
Encryption	IDEA, DES-3, AES

Fig. 6.60 PGP algorithms

How these algorithms are used is explained below.

3. Key Rings

When a sender wants to send an email message to a single recipient, there is not too much of a problem. Complexities are introduced when a message has to be sent to multiple recipients. If Alice needs to correspond with 10 people, Alice needs the public keys of all these 10 people. Hence, Alice is said to need a **key ring** of 10 public keys. Additionally, PGP specifies a ring of public-private keys. This is because Alice may want to change her public-private key pair, or may want to use a different key pair for different groups of users (e.g. one key pair when corresponding with someone in her family, another when corresponding with friends, a third in business correspondence, etc.). In other words, every PGP user needs to have two sets of key rings: (a) A ring of her own public-private key pairs, and (b) A ring of the public keys of other users.

The concept of key rings is shown in Fig. 6.61. Note that in one of the key rings, Alice maintains a set of key pairs; while in the other, she just maintains the public keys (and not key pairs) of other users. Obviously, she cannot have the private keys of the other users. Similarly, other users in a PGP system will have their own two key rings.

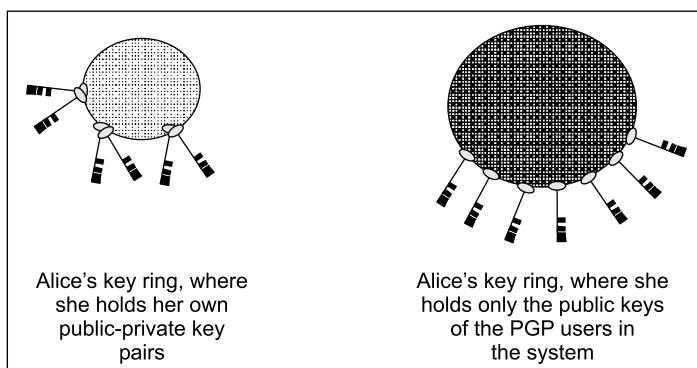


Fig. 6.61 Key rings maintained by a user in PGP

The usage of these key rings should be fairly easy to understand. Nevertheless, we provide a brief explanation below.

There would be two possible situations:

- I. Alice needs to send a message to another user in the system.
 - (a) Alice creates a message digest of the original message (using SHA-1), and encrypts it using her own private key (via the RSA or DSS algorithm) from one of the key pairs shown in the left side of the diagram. This produces a digital signature.
 - (b) Alice creates a one-time symmetric key.
 - (c) Alice uses the public key of the intended recipient (by looking up the key ring shown on the right side for the appropriate recipient) to encrypt the one-time symmetric key created above. RSA algorithm is used for this.
 - (d) Alice encrypts the original message with the one-time symmetric key (using IDEA or DES-3 algorithm).
 - (e) Alice encrypts the digital signature with the one-time symmetric key (using IDEA or DES-3 algorithm).
 - (f) Alice sends the output of steps (d) and (e) above to the receiver. What would the receiver need to do? This is explained below.
- II. Now suppose that Alice has received a message from one of the other users in the system.
 - (a) Alice uses her private key to obtain the one-time symmetric key created by the sender. Refer to steps (b) and (c) in the earlier explanation if you do not understand this.
 - (b) Alice uses the one-time symmetric key to decrypt the message. Refer to steps (b) and (d) in the earlier explanation if you do not understand this.
 - (c) Alice computes a message digest of the original message (say *MD1*).
 - (d) Alice now uses this one-time symmetric key to obtain the original digital signature. Refer to steps (b) and (e) in the earlier explanation if you do not understand this.
 - (e) Alice uses the sender's public key from the key ring shown in the right side of the diagram to decrypt the digital signature and gets back the original message digest (say *MD2*).
 - (f) Alice compares message digests *MD1* and *MD2*. If they match, Alice is sure about the message integrity and authentication of the message sender.

4. PGP Certificates

In order to trust the public key of a user, we need to have that user's digital certificate. PGP can use certificates issued by a CA, or can use its own certificate system.

As explained in our discussion of digital certificates, in X.509, there is a root CA, who issues certificates to the second-level CAs. The second level CAs can issue certificates to the third level CAs, and so on. This can continue up to the required number of levels. At the lowest level, the last CA issues certificates to the end users.

In PGP, things work differently. There is no CA. Anyone can sign a certificate belonging to anyone else in the ring. Atul can sign the certificate for Ana, Jui, Harsh, and so on. There is no hierarchy of trust, or

a treelike structure. This creates a situation where a user can have certificates issued by different users. For example, Jui may have a certificate signed by Atul, and another one by Anita. This is shown in Fig. 6.62. Hence, if Harsh wants to verify Jui's certificate, he has two paths: Jui <174> Atul, and Jui <174> Anita. Harsh may fully trust Atul, but not Anita! Hence, there can be multiple paths in the line of trust from a fully or partially trusted authority to a certificate.

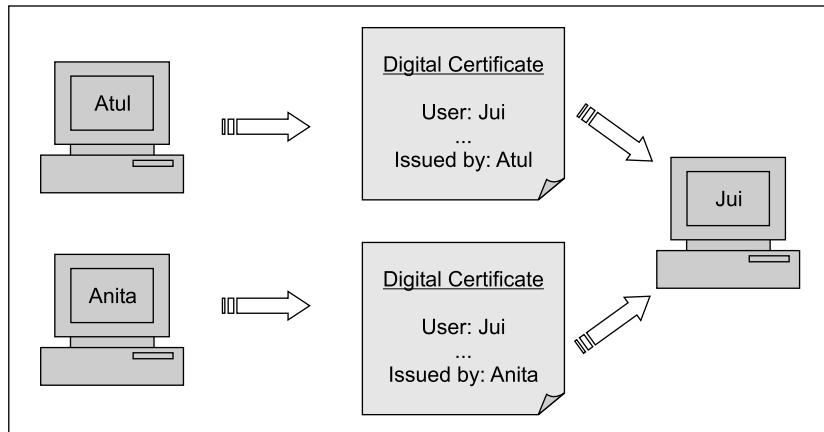


Fig. 6.62 Anyone can issue certificates to anyone else in PGP

The equivalent of CA (i.e. a user who issues certificates) in PGP is called an **introducer**.

The whole concept can be understood better with the help of three ideas:

- Introducer trust
- Certificate trust
- Key legitimacy

We discuss these three concepts now.

(a) Introducer Trust We have mentioned that there is no concept of a hierarchical CA structure in PGP. Hence, it is natural that the ring of trust in PGP cannot be very large, if every user has to trust every other user in the system. Think about this. In real life, we do not fully trust everyone we know. Do we?

To resolve this issue, PGP provides for multiple levels of trust. The number of levels depends on the decision of implementing PGP in a certain way. However, for simplicity, let us say that we have decided to implement three levels of trust to an *introducer*. Let us call these levels as *none*, *partial*, and *complete*. The **introducer trust** then specifies what level of trust the introducer wants to allocate to other users in the system. For example, Atul may now say that he fully trusts Jui, whereas Anita says she only partially trusts Jui. Jui, in turn, says that she does not trust Harsh. Harsh suggests that he partially trusts Anita in turn, and so on. This is shown in Fig. 6.63.

(b) Certificate Trust When a user *A* receives a certificate of another user *B* issued by a third user *C*, depending on the level of trust that *A* has in *C*, *A* assigns a **certificate trust** level to that certificate while storing it. It is normally the same as the *introducer trust* level that issued the certificate. This is shown in Fig. 6.64.

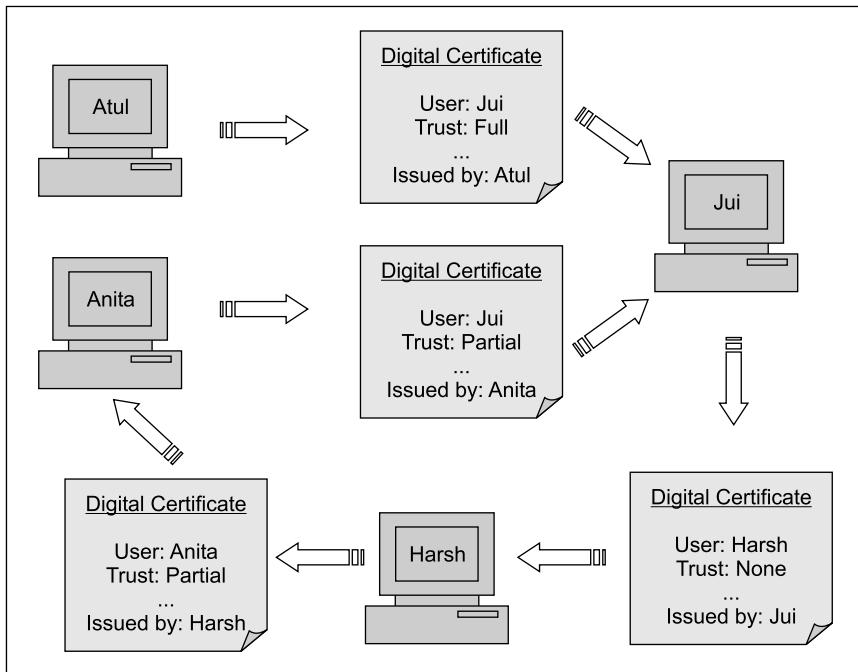


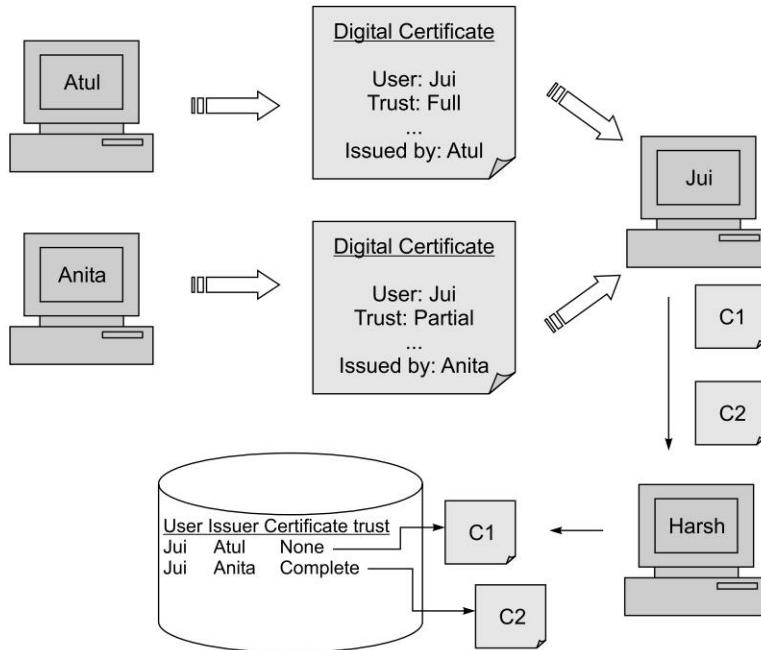
Fig. 6.63 Introducer trust

This concept is explained in the diagram. Let us take another example to ensure that there is no confusion. Imagine that there is a set of users in the system. Assume that Mahesh fully trusts Naren, partially trusts Ravi and Amol, and has no trust in Amit.

- (i) Naren issues two certificates: one to Amrita (with public key $K1$) and another to Pallavi (with public key $K2$). Mahesh stores the public keys and certificates of Amrita and Pallavi in his ring of public keys with *certificate trust* level equal to *full*.
- (ii) Ravi issues a certificate to Uday (with public key $K3$). Mahesh stores the public key and certificate of Uday in his ring of public keys with *certificate trust* level equal to *partial*.
- (iii) Amol issues two certificates: one to Uday (with public key $K3$), and another to Parag (with public key $K4$). Mahesh stores the public keys and certificates of Uday and Parag in his ring of public keys with *certificate trust* level equal to *partial*. Note that Mahesh now has two certificates for Uday, one issued by Ravi, and the other issued by Amol, both with *partial* level of *certificate trust*.
- (iv) Amit issues a certificate to Pramod (with public key $K4$). Mahesh stores the public key and certificate of Pramod in his ring of public keys with *certificate trust* level equal to *none*. Mahesh can also discard this certificate.

(c) Key Legitimacy The objectives behind introducer trust and certificate trust is to decide whether to trust the public key of a user. In PGP terms, this is called **key legitimacy**. Mahesh needs to know how legitimate are the public keys of Amrita, Pallavi, Uday, Parag, Pramod, and so on.

Background information: Atul and Anita have issued certificates to Jui. Jui sends these certificates to Harsh, so that Harsh can extract Jui's public key out of any of those certificates and use it in communication with Jui. However, Harsh does not trust Atul at all, but trusts Anita fully.



Result: When Jui sends the two certificates (issued by Atul and Anita to her) to Harsh, Harsh adds them to his database of certificates. It is actually the ring of public keys of other users, as discussed earlier. Apart from adding them there, Harsh records the fact that it does not want to trust Jui's certificate issued by Atul (since Harsh does not trust Atul), but wants to trust Jui's certificate issued by Anita (since Harsh trusts Anita).

Fig. 6.64 Certificate trust

PGP defines the following simple rule to decide the key legitimacy: *The level of key legitimacy for a user is the weighted trust level for that user.* For instance, suppose we have assigned certain weights to certificate trust levels, as shown in Fig. 6.65.

In this situation, in order to trust a public key (i.e. certificate) of any other user, Mahesh needs one fully trusted certificate or two partially trusted certificates. Thus, Mahesh can fully trust Amrita and Pallavi based on the certificates they had received from Naren. Mahesh can also trust Uday, based on the two partially trusted certificates that Uday had received from Ravi and Amol.

Weight	Meaning
0	No trust
$\frac{1}{2}$	Partial trust
1	Complete or full trust

Fig. 6.65 Assigning weights to certificate trust levels

Interestingly, the legitimacy of a public key belonging to an entity has nothing to do with the trust level of that person. For instance, Naren may be trusting Amit. Hence, Naren can encrypt a message with the public key derived from Amit's certificate, and can send the encrypted message to Amit. However, Mahesh will continue to reject certificates issued by Amit, since he does not trust Amit.

5. Web of Trust

The earlier discussion leads to a potential problem. What happens if nobody creates a certificate for a fully or partially trusted entity? In our example, on what basis would we trust Naren's public key if no one has created a certificate for Naren? To resolve this problem, several schemes are possible in PGP, as outlined below.

- (a) Mahesh can physically obtain the public key of Naren by meeting in person and getting the key on a piece of paper or as a disk file.
- (b) This can be done telephonically as well.
- (c) Naren can email his public key to Mahesh. Both Naren and Mahesh compute a message digest of this key. If MD5 is used, the result is a 16-byte digest. If SHA-1 is used, the result is a 20-byte digest. In hexadecimal, the digest becomes a 32-digit value in MD5, and a 40-digit value in SHA-1. This is displayed as 8 groups of 4-digit values in MD5, or 10 groups of 4-digit values in SHA-1, and is called **fingerprint**. Before Mahesh adds the public key of Naren to his ring, he can call up Naren to tell him what fingerprint value he has obtained to cross-check with the fingerprint value that is separately obtained by Naren. This ensures that the public key value is not changed in the email transit. To make matters better, PGP assigns a unique English word to a 4-digit hexadecimal number group, so that instead of speaking out the hexadecimal string of numbers, users can speak out normal English words, as defined by PGP. For example, PGP may have assigned a word *India* to a hexadecimal pattern of *4A0B*, etc.
- (d) Mahesh can, of course, obtain Naren's public key from a CA.

Regardless of the mechanism, eventually this process of obtaining keys of other users and sending our own to others creates what is called a **web of trust** between groups of people. This keeps the public key ring getting bigger and bigger, and helps secure the email communication.

Whenever a user needs to revoke his/her public key (because of loss of private key, etc), he/she needs to send a *key revocation certificate* to the other users. This certificate is self-signed by the user with his/her private key.

6.9.4 Secure Multipurpose Internet Mail Extensions (S/MIME)

1. Introduction

The traditional email systems using the SMTP protocol (based on RFC 822) are text-based, which means that a person can compose a text message using an editor and then send it over the Internet to another recipient. However, in the modern era, exchanging only text messages is not quite sufficient. People want to exchange multimedia files, documents in various arbitrary formats, etc. To cater to these needs, the **Multipurpose Internet Mail Extensions (MIME)** system extends the basic email system by permitting users to send binary files using the basic email system. MIME is defined in RFCs 2045 to 2049.

Knowing why SMTP cannot work with non-text data is important. SMTP uses the 7-bit ASCII representation for characters of an email message. 7-bit ASCII cannot represent special characters above the ASCII value of 127. SMTP cannot also send binary data.

A MIME email message contains a normal Internet text message along with some special headers and formatted sections of text. Each such section can hold an ASCII-encoded portion of data. Each section starts with an explanation as to how the data that follows should be interpreted/decoded at the recipient's end. The recipient's email system uses this explanation to decode the data.

Let us consider a simple example containing MIME header. Figure 6.66 shows an email message (after the sender has composed the email message and has attached a graphics file having .GIF extension). The figure shows the email message as it actually travels to the recipient. The Content-Type MIME header shows the fact that the sender has attached a .GIF file to this message. The actual image would be sent as a part of the message after this, and would appear gibberish when viewed in the text form (because it would be the binary representation of the image). However, the recipient's email system shall recognize that this is a .GIF file, and as such, it should invoke an appropriate program that can read, interpret, and display contents of a .GIF file.

```
From: Atul Kahate <akahate@indiatimes.com>
To: Anita Kahate <akahate@yahoo.com>
Subject: Cover image for the book
MIME-Version: 1.0
Content-Type: image/gif

<Actual image data in the binary form such as R019a0asdjas0 ...>
```

Fig. 6.66 MIME extensions to an email message

When we enhance the basic MIME system to provide for security features, it is called **Secure Multipurpose Internet Mail Extensions (S/MIME)**.

2. MIME Overview

We have seen that the email system provides for email headers such as *From*, *To*, *Date*, *Subject*, etc. The MIME specification adds five new headers to the email system, which describe information about the body of the message (we have already seen two MIME headers in our example just now). Thus, when MIME is in use, the email message looks as shown in Fig. 6.67.

These headers are described below.

(a) MIME-Version This contains the MIME version number. Currently, it has a value of 1.1. This field is reserved for future use, when newer versions of MIME are expected to emerge. This field indicates that the message conforms to RFCs 2045 and 2046.

(b) Content-Type This describes the data contained in the body of the message. The details provided are sufficient so that the receiver email system can deal with the received email message in an appropriate manner. The contents are specified as

Type/Sub-type

MIME specifies 7 content types, and 15 content sub-types. Figure 6.68 lists these types and sub-types.

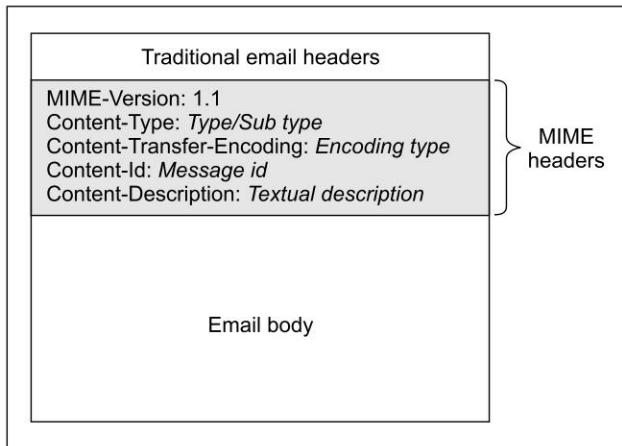


Fig. 6.67 MIME headers in an email message

Type	Sub-type	Description
Text	Plain	Free form text.
	Enriched	Text with formatting details.
Multipart	Mixed	Email contains multiple parts. All parts must be delivered together, and in sequence.
	Parallel	Email contains multiple parts. All parts must be delivered differently, in different sequence.
	Alternative	Email contains multiple parts. These parts represent the alternative versions of the same information. They are sent so that the receiver's email system can select the best fit from them.
	Digest	Similar to Mixed. Detailed discussion is out of scope of the current text.
Message	RFC822	The body itself is an encapsulated message that conforms to RFC 822.
	Partial	Used in fragmentation of larger email messages.
	External-body	Contains a pointer to an object that exists somewhere else.
Image	Jpeg	An image in JPEG format.
	Gif	An image in GIF format.
Video	Mpeg	A video in MPEG format.
Audio	Basic	Sound format.
Application	PostScript	Adobe PostScript.
	octet-stream	General binary data.

Fig. 6.68 MIME content types

(c) Content-Transfer-Encoding Specifies the type of transformation that has been used to represent the body of the message. In other words, the method used to encode the messages into zeroes and ones is defined here. There are five content-encoding methods, as shown in Fig. 6.69.

Type	Description
7-bit	NVT ASCII characters and short lines
8-bit	Non-ASCII characters and short lines
Binary	Non-ASCII characters with unlimited-length lines
Base-64	6-bit blocks of data encoded into 8-bit ASCII characters
Quoted-Printable	Non-ASCII characters encoded as an equal to sign, followed by an ASCII code

Fig. 6.69 Content-transfer-encoding values

(d) Content-ID Identifies the MIME entities uniquely with reference to multiple contexts.

(e) Content-Description Used when the body is not readable (e.g. video).

3. S/MIME Functionality

In terms of the general functionality, S/MIME is quite similar to PGP. Like PGP, S/MIME provides for digital signatures and encryption of email messages. More specifically, S/MIME offers the functionalities as depicted in Fig. 6.70.

Functionality	Description
Enveloped data	Consists of encrypted content of any type, and the encryption key encrypted with the receiver's public key.
Signed data	Consists of a message digest encrypted with the sender's private key. The content and the digital signature are both Base-64 encoded.
Clear-signed data	Similar to Signed data. However, only the digital signature is Base-64 encoded.
Signed and Enveloped data	Signed-only and Enveloped-only entities can be combined, so that the Enveloped data can be signed, or the Signed/Clear-signed data can be enveloped.

Fig. 6.70 S/MIME functionalities

4. Cryptographic Algorithms used in S/MIME

In respect of the cryptographic algorithms, S/MIME prefers the usage of the following cryptographic algorithms:

- Digital Signature Standard (DSS) for digital signatures
- Diffie–Hellman for encrypting the symmetric session keys
- RSA for either digital signatures or for encrypting the symmetric session keys
- DES-3 for symmetric key encryption

Interestingly, S/MIME defines two terms: *must* and *should* for describing the usage of the cryptographic algorithms. What is the meaning of these? Let us understand.

(a) Must This word specifies that these cryptographic algorithms are an absolute requirement. The user systems of S/MIME have to necessarily support these algorithms.

(b) Should There could be reasons because of which algorithms in this category cannot be supported. However, as far as possible, these algorithms should be supported.

Based on these terminologies, S/MIME supports the various cryptographic algorithms as shown in Fig. 6.71.

Functionality	Algorithm support recommended by S/MIME
Message Digest	<i>Must</i> support MD5 and SHA-1. <i>Should</i> use SHA-1.
Digital Signature	Sender and receiver both <i>must</i> support DSS. Sender and receiver <i>should</i> support RSA.
Enveloping	Sender and receiver <i>must</i> support Diffie–Hellman. Sender and receiver <i>should</i> support RSA.
Symmetric key encryption	Sender <i>should</i> support DES-3 and RC4. Receiver <i>must</i> support DES-3 and <i>should</i> support RC2.

Fig. 6.71 Guidelines provided by S/MIME for cryptographic algorithms

5. S/MIME Messages

Let us now discuss the general procedures for preparing an S/MIME message.

S/MIME secures a MIME entity with a signature, encryption, or both. When we talk of *MIME entity*, it can mean an entire message, or a sub-part of the whole message. The MIME entity is prepared as per the usual MIME rules. This is processed by S/MIME, along with security-related data, such as identifiers of algorithms and digital certificates. The output of this process is called a **Public Key Cryptography Standard (PKCS)** object. This PKCS object itself is now considered as a message content and is wrapped inside MIME, with the addition of appropriate MIME headers.

As we have mentioned, for an email message, S/MIME supports digital signature, encryption or both. S/MIME processes the email messages along with the other security-related data, such as the algorithms used and the digital certificates to produce a PKCS object. As mentioned earlier, it is then treated like a message content. This means that appropriate MIME headers are added to it. For this purpose, S/MIME has two new content types and six new sub-types, as shown in Fig. 6.72.

Type	Sub-type	Description
Multipart	Signed	A clear signed message consisting of the message and the digital signature.
Application	PKCS#7 MIME Signed Data	A signed MIME entity.
	PKCS#7 MIME Enveloped Data	An enveloped MIME entity.
	PKCS#7 MIME Degenerate Signed Data	An entity that contains only digital certificates.
	PKCS#7 Signature	The content type of the signature subpart of a multipart/signed message.
	PKCS#10 MIME	A certificate registration request.

Fig. 6.72 S/MIME content types

6. S/MIME Certificate Processing

S/MIME uses X.509V3 certificates. The key-management scheme used by S/MIME is a bit of a mixture of the X.509 certificate hierarchy and the web of trust, as specified in PGP. Like PGP, S/MIME needs a configuration of the list of trusted keys and CRLs. Certificates are signed by CAs, as usual.

An S/MIME user performs three key-management functions, as listed in Fig. 6.73.

Function	Description
Key generation	The user with some administrative capabilities must be able to create Diffie–Hellman and DSS key pairs and should be able to create RSA key pairs.
Registration	A user's public key must be registered with a CA to receive an X.509 digital certificate.
Certificate storage and retrieval	A user needs digital certificates of other users to decrypt incoming messages and validate signatures of incoming messages. These must be maintained by a local administrative entity.

Fig. 6.73 Key-management functions

7. S/MIME Additional Security Features

Three additional features are proposed in the S/MIME protocol, as a draft. They are summarized below.

(a) Signed Receipts This message can be used as acknowledgment of an original message. This provides proof of delivery of a message to the original sender. The recipient signs the entire message (including the original message sent by the sender, the signature of the sender, and the acknowledgment) and creates an S/MIME message type out of it.

(b) Security Labels A security label may be added to a message to identify its sensitivity (how confidential it is), access control (who can access it), and priority (secret, confidential, restricted, etc.).

(c) Secure Mailing Lists An S/MIME Mailing List Agent (MLA) can be created to take over the processing that is required per recipient whenever a sender sends a message to multiple users. For example, if a message is being sent to 10 recipients, it may have to be encrypted with the 10 respective public keys of the recipients. An MLA can take a single incoming message, perform the recipient-specific encryption, and forward the message. This means that the original sender needs to only encrypt the message once (with the public key of the MLA) and only send it once (to the MLA). The MLA then does the remaining work.

6.9.5 DomainKeys Identified Mail (DKIM)

DomainKeys Identified Mail (DKIM) is a proposed Internet standard. Already adopted by a number of email providers such as GMail, Yahoo! and many other companies and Internet Service Providers (ISP), DKIM can be useful in establishing identity related to emails. In simple terms, the user's email message is digitally signed by a private key belonging to the administrative domain from where the email is sent (e.g. that of Gmail or Yahoo!). The digital signature is done on the entire email contents plus a few headers. The receiver's email system verifies the digital signature on the message by using the public key of the sender's administrative domain. This assures the receiver of the email message that the email indeed originated from the claimed sender's domain and that its integrity was preserved during transit.

We need to distinguish between traditional email security protocols such as PGP, S/MIME, and DKIM. In the traditional email security protocols, the sender herself signs the message whereas in DKIM, the sender's domain signs the message. In other words, if my email account is akahate@gmail.com then by using PGP or S/MIME, I myself would sign the email message; but by using DKIM, GMail would sign the email message on my behalf.

There are several reasons why DKIM was proposed in addition to the existing email security protocols. Unlike S/MIME, no special processing needs to be done in DKIM both by the sender and the receiver beyond signing and verification of messages. Also, DKIM signs message contents and headers, unlike S/MIME. Because DKIM is implemented at the domain level, individual users need not worry about it. They are mostly not even aware of its existence and usage.

Like the traditional email security protocols, we can use message-digest algorithms such as SHA-256 to compute a message digest of the original email message plus headers and then apply the RSA digital signature algorithm to compute a digital signature of this email message.

■ 6.10 WIRELESS APPLICATION PROTOCOL (WAP) SECURITY ■

6.10.1 Introduction

In the late 1990's, wireless computing stormed the world of the Internet. Until that time, the Internet was accessible only if you had a PC. However, that changed in 1997 with the arrival of a new set of standards for wireless Internet access through wireless handheld devices and Personal Digital Assistants (PDAs). The **Wireless Application Protocol (WAP)** had arrived. In simple terms, WAP is a communication protocol that enables wireless mobile devices to have an access to the Internet.

The WAP architecture was developed keeping in mind the basic Internet architecture, and yet realizing that WAP had to deal with the limitations of the mobile devices. Thus, the idea was to model WAP on the Internet architecture, and at the same time, make sure that the differences between the wired and the wireless world would also be taken care of. Thus, the basic principle was to give the end user a look and feel that is similar to what the Internet gives, and also keep the communication between the content providers and the end users to a basic minimum. This also explains why the Internet architecture of TCP/IP and HTTP/FTP, etc., could not be directly used in the case of mobile devices. These Internet protocols are too complex for the small and less powerful mobile devices and the mobile channels of communication. For instance, establishing and closing a TCP connection between a client and a server takes a lot of time and bandwidth, which is fine for the wired world. However, for the mobile world, where the processing power and bandwidth are already areas of concern, these additional requirements of connection, establishment and closing can make it very slow. The mobile devices cannot be expected to perform high amount of information processing, or also carry high amount of data even for establishing connections (the way TCP does, for example). In short, TCP/IP or HTTP is too heavy for them.

In case of the WAP architecture, we have an additional level between the client and the server: the **WAP gateway**. Simplistically, the job of the WAP gateway is to translate client requests to the server from WAP to HTTP, and on the way back from the server to the client, from HTTP to WAP as shown in Fig. 6.74. The WAP requests first originate from the mobile device (usually a mobile phone), which travel to the network carrier's base station (shown as a tower), and from there, they are relayed on to the WAP gateway where the conversion from WAP to HTTP takes place. The WAP gateway then interacts with

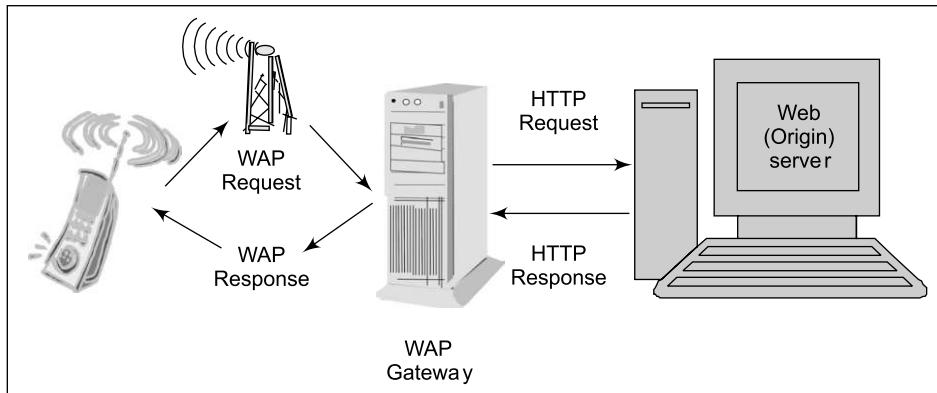


Fig. 6.74 Interaction of a mobile phone with the Internet

the Web server (also called **origin server**) as if it is a Web browser, i.e. it uses HTTP protocol for interacting with the Web server. On return, the Web server sends an HTTP response to the WAP gateway, where it is converted into a WAP response, which first goes to the base station, and from there on, to the mobile device.

6.10.2 The WAP stack

It is now time to examine the WAP stack, in the context of WAP security. More specifically, we shall attempt to map the WAP stack on to the TCP/IP stack of the Internet, so as to get a feel of the similarities and differences between them. However, we must note that the WAP stack is based more on the OSI model, rather than the TCP/IP model. Figure 6.75 shows the WAP stack.

As the figure shows, the WAP stack consists of five protocol layers, of which we are interested in only one:

Security Layer

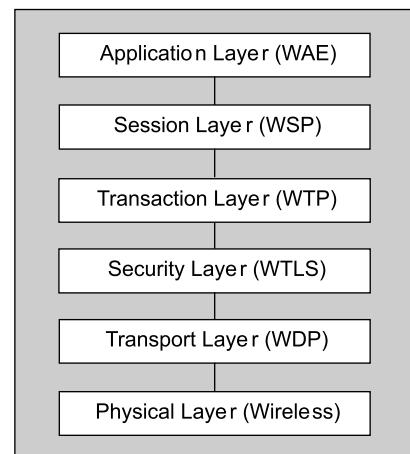


Fig. 6.75 WAP stack

The **security layer** in the WAP stack is also called **Wireless Transport Layer Security (WTLS)** protocol. It is an optional layer, that when present, provides features such as authentication, privacy and secure connections—as required by many modern e-commerce and m-commerce applications.

6.10.3 The Security Layer—Wireless Transport Layer Security (WTLS)

The wireless world is more vulnerable to security issues as compared to the wired world, as the number of parties involved is more, and the chances of people not taking proper security measures when on the move are significantly higher. As a result, the WAP protocol stack includes the Wireless Transport Layer Security (WTLS) as an additional layer, which is not found in other similar protocol stacks. WTLS is optional. It is based on the *Transport Layer Security (TLS)* protocol, which, in turn, is based on the *Secure Socket Layer (SSL)* protocol. When present, WTLS runs on top of the transport layer of WAP (WDP).

As we know, SSL has made tremendous impact on the way e-commerce transactions can be conducted in the traditional Internet world. SSL allows two parties involved in a transaction to make it totally secure and reliable. WLTS makes similar attempts in the wireless world. WTLS ensures four things: *privacy, server authentication, client authentication, and data integrity*.

- **Privacy** ensures that the messages passing between the client and the server are not accessible to anybody else. Encrypting the messages, as discussed earlier, does this.
- **Server authentication** gives the client a confidence that the server is indeed what it is depicting as, and not someone who is posing as the server, with or without malicious intentions.
- **Client authentication**, on similar lines, gives the server a confidence that the client is indeed what it is depicting as, and not someone who is posing as the client, with or without malicious intentions.
- **Data integrity** ensures that no one can tamper with the messages going between the client and the server, by modifying their contents in any manner.

Figure 6.76 shows how the communication between a WAP client and the origin server can be made secure. Between the WAP client and the WAP gateway, we have WTLS to ensure a secure-mode transaction. Between the WAP gateway and the origin server, SSL takes care of security, as usual. Thus, the WAP gateway performs the translations between WTLS and SSL in both directions.

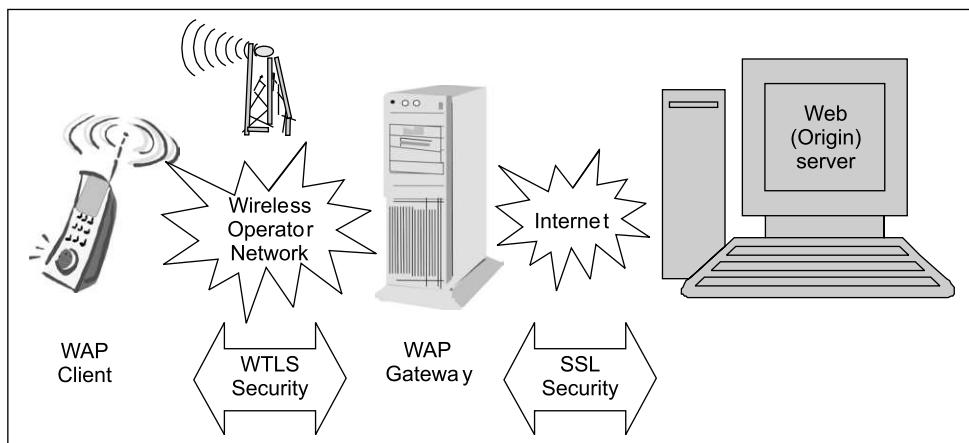


Fig. 6.76 WTLS and SSL security

The conversion between WTLS and SSL is a major point for debate. This is because; the WAP gateway first converts WTLS text into plain text and then applies SSL (or vice versa). Therefore, it has access to the non-encrypted message in its original form! The WAP gateway performs this conversion in its memory and never stores any portions of it on its disk. Clearly, if it stores it on its disk, it can be a major cause for worry. However, even the fact that it performs this conversion in its memory has made many people quite unhappy about the amount of security thus provided. They feel that even a momentary lapse here could cause havoc. As a consequence, many banks, merchants and financial institutions supporting WAP transactions prefer to have their own WAP gateways to make sure that the WTLS-to-SSL and SSL-to-WTLS conversion is under their control.

The most important difference between SSL and WTLS is that SSL needs a reliable transport layer, i.e. TCP underneath for it to guarantee a secure mode of transaction between the client and the server.

In contrast, in case of WAP, the reliable/unreliable mode of transactions is decided by protocols *above* WTLS (namely, by WTP and WSP). Therefore, WTLS does not require a reliable transmission mode. In other words, it can work as well with unreliable mode of transport, which is not possible with SSL. To achieve this, WTLS defines a sequence number field in its frame, which is not done in case of SSL. Instead, SSL relies on TCP to perform sequencing and error checking.

■ 6.11 SECURITY IN GSM ■

In the earlier days of mobile telephony, analog technologies such as **Advanced Mobile Phone System (AMPS)** were used. There was little or no security in such technologies. Each mobile phone in such a system has a 32-bit serial number and a 10-digit telephone number in its PROM. The telephone number is made up of a 3-digit area code, represented by 10 bits and a 7-digit subscriber number in 24 bits. When a mobile telephone is switched on, it sends out its 32-bit serial number and a 34-bit number. All this information was sent out in clear text. So, anyone trying to listen to the passing wireless communication could do so, and also access the serial number and the telephone number to his/her advantage.

An improvement over AMPS was to make it digital. This was in the form of a technology called **Digital AMPS (D-AMPS)**. D-AMPS is used extensively in the US and Japan (with certain modifications). Another similar voice technology called **Global System for Mobile Communications (GSM)** is used widely in Europe, and it has now spread its wings even in the US. In the last few years, alternatives to the WAP standard have emerged, and have actually become quite popular. **General Packet Radio Service (GPRS)** is an emerging wireless data service that offers a mobile data experience similar to current analog modems without wires and with access wherever GSM wireless service is available. Just to reiterate, GSM is for voice, whereas GPRS is for data. Both these technologies are together called 2.5th Generation wireless technologies (2.5G).

We shall briefly discuss the security features in GSM to get a feel of where wireless security is moving at the lower layers, although this is not directly related to Internet security.

There are three key aspects to GSM security:

- Subscriber identity authentication
- Signaling data confidentiality
- User data confidentiality

Each subscriber is identified with a unique *International Mobile Subscriber Identity (IMSI)*. Each subscriber also has a unique *subscriber authentication key (Ki)*. GSM authentication and encryption work in such a way that this sensitive information is never transmitted across the mobile network. Instead, a challenge-response mechanism is used to perform authentication. The actual transmissions are encrypted with the help of a temporary, randomly generated ciphering key (*Kc*).

The security is distributed in three different elements of the GSM infrastructure: the *Subscriber Identity Module (SIM)*, which is a plastic card inside a mobile phone, the GSM handset and the GSM network

- The SIM contains the IMSI, *Ki*, the ciphering key generation algorithm (*A8*), the authentication algorithm (*A3*) as well as a Personal Identification Number (PIN).
- The GSM handset contains the ciphering algorithm (*A5*).

- The Authentication Center (AUC), which is a part of the GSM network, contains the encryption algorithms ($A3$, $A5$ and $A8$) as well as a database of identification and authentication information about the subscribers.

Armed with this information, let us now think how security is achieved in GSM.

(a) Authentication The GSM network authenticates a subscriber as shown in Fig. 6.77, which we shall discuss now.

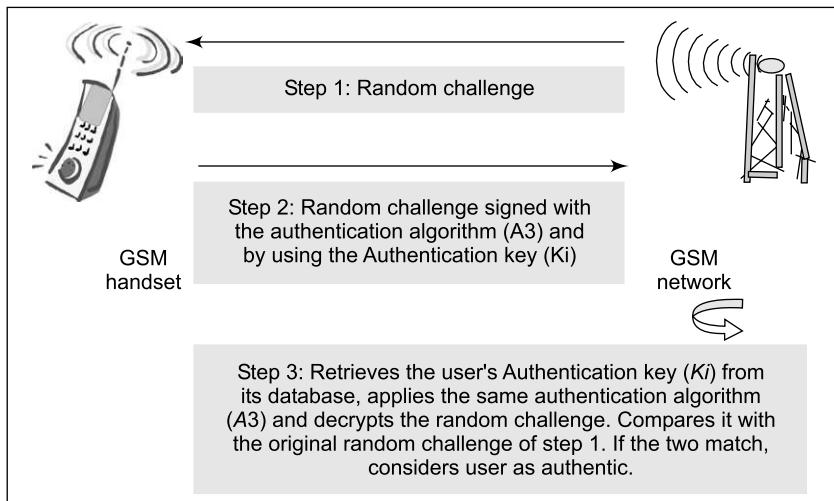


Fig. 6.77 GSM authentication

The process begins with a challenge-response mechanism. The network sends a 128-bit random number to the subscriber when authentication begins. After this, 32-bit signed response using the authentication algorithm ($A3$) and the subscriber authentication key (Ki) is prepared by the handset, and sent back to the network. The network retrieves its value of Ki from its database, performs the same operation using the $A3$ algorithm on the original 128-bit random number, and compares this result with the one received from the handset. If the two match, the user is considered as successfully authenticated. Since the calculation of the signed response takes place inside the SIM, the IMSI or Ki never have to leave the SIM. That makes authentication secure.

(b) Signaling and Data Confidentiality As we have mentioned earlier, the SIM contains the ciphering key generation algorithm ($A8$). This is used to produce the 64-bit ciphering key (Kc). The value of Kc is obtained by applying the same random number as used in authentication to the $A8$ algorithm with the individual subscriber authentication key (Ki). This key (Kc) is later used for secure communications between the subscriber and the mobile telephony base station. This process is shown in Fig. 6.78.

(c) Voice and Data Security The $A5$ algorithm is used to encrypt the voice and data traffic between the user's handset and the GSM network. For this, the subscriber's handset sends a *ciphering mode request* to the GSM network. The network, in response, starts encryption and decryption of the traffic using the ciphering algorithm ($A5$) and the ciphering key (Kc).

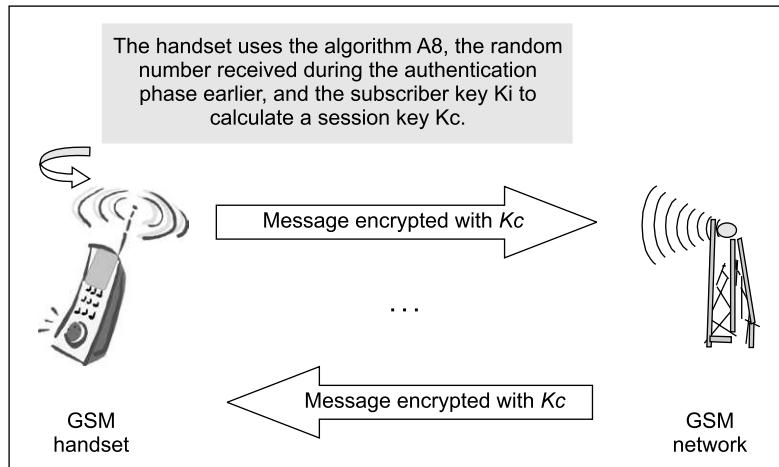


Fig. 6.78 GSM encryption

The algorithms ($A3$, $A5$, $A8$) are kept secret, and are not available to the general public. However, they have been discovered, published on the Internet, and their implementations in C and other programming languages are available in many books/resources.

■ 6.12 SECURITY IN 3G ■

GPRS has naturally evolved into **Universal Mobile Telephone System (UMTS)**. UMTS is an extension of the basic premises on which GPRS is based. As we have noted, GPRS is called a technology that is somewhat in-between the second and third generation of wireless technologies (2.5), whereas UMTS is called the **third generation of wireless/mobile technology (3G)**. UMTS extends the wireless system performance of GPRS networks 2.5 by offering expanded data services and enhanced data speeds.

UMTS can be used to deliver high-tech applications such as video on demand, video/audio streaming, high-speed multimedia, videoconferencing, multi-player gaming and improved mobile Internet access. The main benefit will be high-end service capabilities, which include substantially enhanced capacity, quality and data rates that are currently available. UMTS also allows the concurrent usage of multiple services.

We shall now discuss the authentication process of UMTS.

The UMTS authentication process involves three parties: User's mobile handset, Home location and the Current location (note that the user of a mobile phone can move around, and therefore the current location of the user may be different from his/her home location). The user authentication process consists of four steps, as follows.

1. The user's mobile handset sends its International Mobile Subscriber Id (IMSI) to the Home Location. The IMSI is unique for a handset, and is optionally encrypted before it is sent to the Home Location.

2. The Home Location performs the following steps now:
 - (a) It generates a random number (RAND).
 - (b) Next, it retrieves the secret key it shares with this user's handset from its database.
 - (c) It uses the random number and the key to generate the following items:
 - (i) Response (RES)
 - (ii) Confidentiality Key (CK)
 - (iii) Integrity Key (IK)
 - (iv) Authentication Key (AK)
 - (d) The Home Location and the user's handset share a secret, called Sequence Number (SEQ). The Home Location calculates a MAC on the combination of the random number (RAND) and the sequence number (SEQ), i.e. it does MAC (RAND, SEQ).
 - (e) It now does an XOR operation on the sequence number (SEQ) and Authentication Key (AK), i.e. it does (SEQ XOR AK).
 - (f) Finally, it sends the following items to the Current Location of the user:

RAND, RES, CK, IK, MAC, (SEQ XOR AK)
3. The Current Location receives these values from the Home Location, and sends the following to the user's handset:
 - (a) Random number (RAND)
 - (b) XOR of sequence number (SEQ) and Authentication Key (AK), i.e. (SEQ XOR AK)
 - (c) MAC
4. The user's handset receives these values, and performs the following tasks:
 - (a) It calculates the response (RES), using the random number (RAND) received from the Current Location, and the secret key shared with its Home Location.
 - (b) It also generates the various keys such as CK, IK and AK in the same fashion, as was done by the Home Location in step 1.
 - (c) It now performs an XOR operation using AK over the value (SEQ XOR AK). That is, it performs (SEQ XOR AK) XOR AK. Clearly, this would yield back the sequence number (SEQ).
 - (d) Using the random number (RAND) and the sequence number (SEQ), it performs a MAC operation, as MAC (RAND, SEQ).
 - (e) It compares the MAC calculated in step 4 with the MAC received from the Current Location (see step 3).
 - (f) If all these substeps happen successfully, the user's handset sends the response (RES) to the Current Location. If this response matches with the response (RES) possessed by the Current Location (recall that it had received this from the Home Location in step 1), the Current Location considers this user as authentic, and makes the appropriate entries in its database.

The authentication process ends here. Figure 6.79 shows the conceptual view of the authentication process.

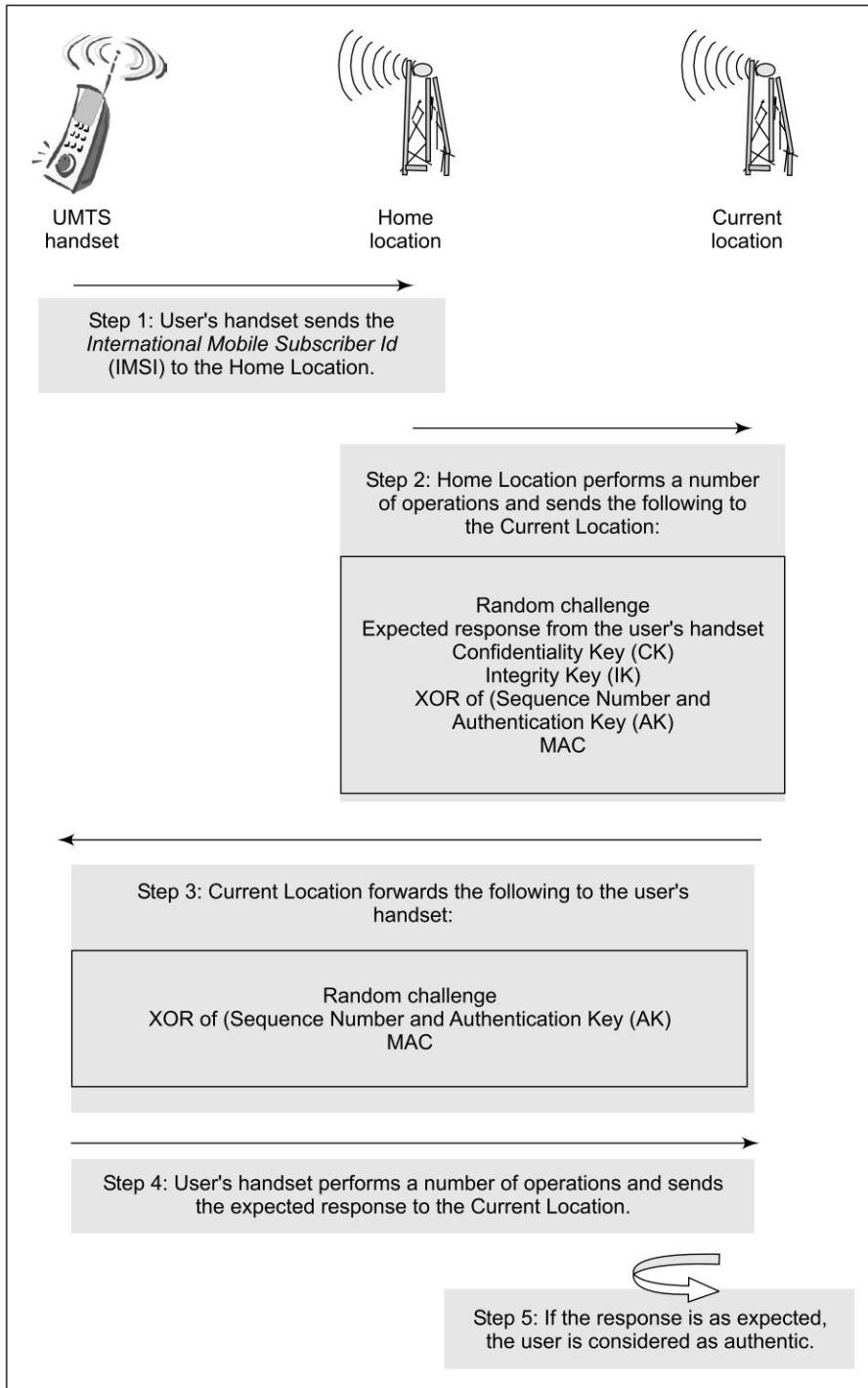


Fig. 6.79 UMTS authentication

It should be obvious by now that the keys generated during authentication, such as the Confidentiality Key (CK) and the Integrity Key (IK) can be used for future cryptographic/security operations, such as message confidentiality and message integrity.

■ 6.13 IEEE 802.11 SECURITY ■

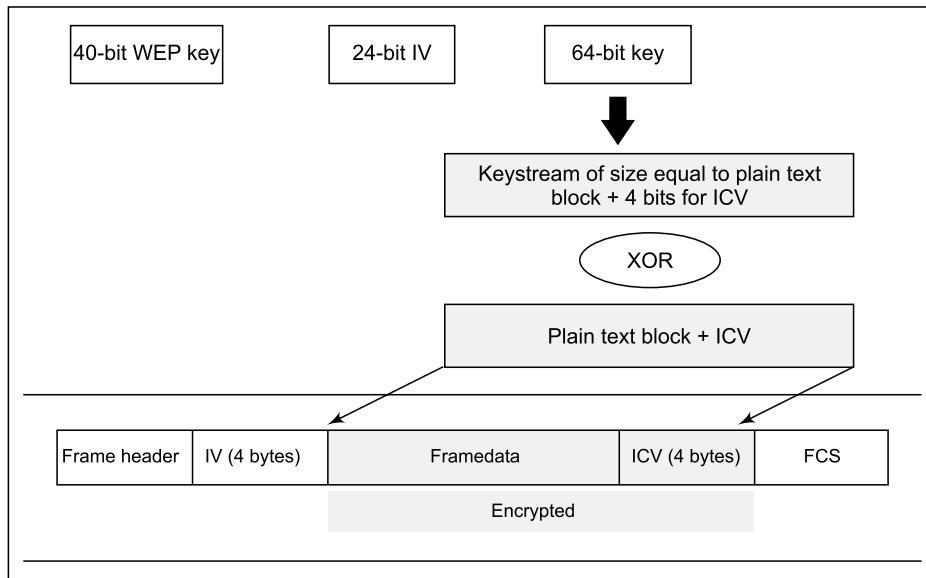
6.13.1 Wired Equivalent Privacy (WEP)

As the name suggests, the aim of the **Wired Equivalent Privacy (WEP)** is to provide security in a wireless network that is similar to that in a wired network. Andrew Tanenbaum summarizes the state of affairs beautifully: *"The goal of WEP is to make the security of a wireless LAN as good as that of wired LAN. Since the default for a wired LAN is no security at all, this goal is easy to achieve, and WEP achieves it!"*

In effect, WEP is very weak, has been exploited, and is no longer trusted. Nevertheless, for the sake of completeness, we must describe it. The way it works is as follows:

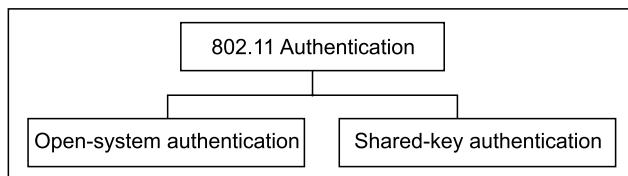
1. Every host in a wireless network is supposed to have a shared key with the AP. How these keys should be distributed or agreed upon is not stated in the 802.11 specifications. Hence, it is left to the person who implements WEP in an 802.11 wireless network.
2. The actual encryption is based on the RC4 algorithm using stream cipher (i.e. individual bytes are encrypted). The algorithm uses a 40-bit key, which is hardly acceptable by today's standards. The RC4 algorithm was developed by Ron Rivest (one of the creators of the famous RSA algorithm), and was kept secret (something that should never be done)! But in September 1994, it was hacked and its description was posted on the Internet in a mailing list. Worse yet, the way it was implemented in WEP made it quite vulnerable.
3. WEP works as follows and is shown in Fig. 6.80.
 - (a) We have mentioned that the RC4 protocol is based on a 40-bit symmetric key. A 24-bit random value called as Initial Vector (IV) is added to this 40-bit key to produce a 64-bit key.
 - (b) From this 64-bit key, a *keystream* is generated, with the size equal to the size of the plain-text block plus the size of the Integrity Check Value (ICV). The size of ICV is 4 bits.
 - (c) The *keystream* is XORed with the plain-text block plus the ICV. This is the resulting cipher text.
 - (d) The final cipher text frame structure contains the following:
 - (i) Frame header and IV in plain text
 - (ii) Cipher text and a 4-bit ICV in encrypted format
 - (iii) Frame Check Sequence (FCS) in plain text as an equivalent of checksum

The trouble with the whole thing is that in many implementations of WEP, a key that is supposed to be a secret key between a host and an AP is actually shared across several hosts! In another case, even if let us say that the keys are not shared and hence they are different, their lifetime is so long that they can be guessed by way of repeated attempts of attacks. To counter this, the 802.11 standard recommends that at least the IV used every time should be different. But the implementers of wireless network *outsmart* (!) them by setting IV to 0! In the very worst case, even if non-null IV is used, it is just 24 bits long.

**Fig. 6.80** WEP

6.13.2 IEEE 802.11 Authentication

The 802.11 standard supports two major authentication mechanisms, as shown in Fig. 6.81.

**Fig. 6.81** 802.11 authentication mechanisms

1. Open-System Authentication

Open-system authentication is also called **null authentication algorithm**. This is because no real authentication takes place here! The mechanism is based on a two-message sequence. In the first message, the party (host) that wants to authenticate itself with the other party (AP) sends a message containing the authentication information. The AP would validate this and return a response in the form of a message such as either *success* or *failure* or *not supported*. This concept is shown in Fig. 6.82.

2. Shared-key Authentication

The **shared-key authentication** mechanism is a four-step process.

- The host that is trying to authenticate itself sends a request message to the AP, indicating that it wants to start the authentication process.
- The AP sends back a response that contains an authentication challenge (a string of randomly generated bytes using the **WEP** protocol, which is discussed separately).

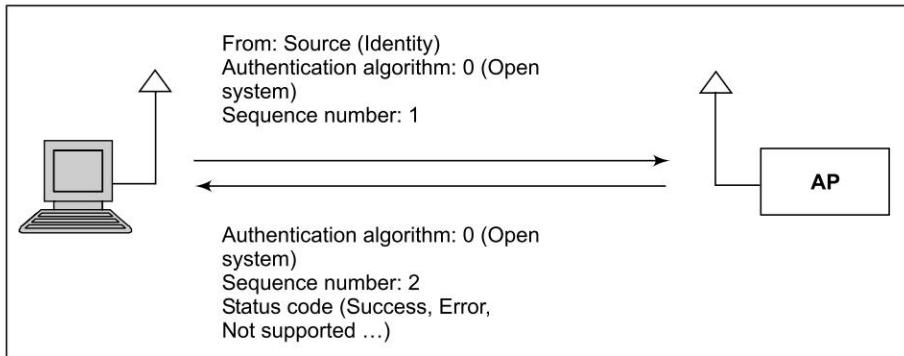


Fig. 6.82 Open-system authentication

- (c) The host encrypts the authentication challenge that was sent by the AP and sends it back to the AP.
- (d) The AP decrypts the encrypted authentication challenge received from the host in step 3. It matches it with the original random challenge that it had originally created in step 2. If the two match, the AP is convinced that the host is genuine (because the host could encrypt the random challenge successfully in step 3 with the same key with which it was decrypted back now in step 4). It sends a success message back to the host. Otherwise, the AP would send back a failure message to the host.

This process is illustrated in Fig. 6.83.

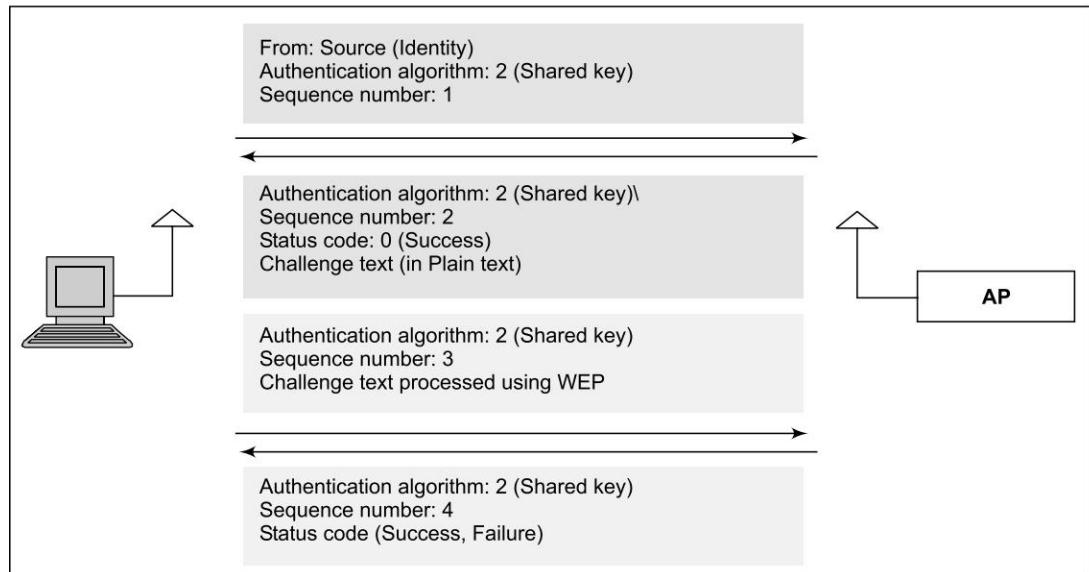


Fig. 6.83 Shared-key authentication

To overcome the drawbacks of WEP, in October 2002, a new standard was implemented. We shall discuss it now.

6.13.3 Wi-Fi Protected Access (WPA)

The **Wi-Fi Protected Access (WPA)** overcomes the drawbacks of WEP. It provides the following services:

(a) Authentication For this, WPA makes use of a separate dedicated Authentication Server (AS). It performs mutual authentication and handles key management. It generates temporary keys to be used between the host and the AP.

(b) Encryption It provides stronger encryption by using the AES protocol.

(c) Message Integrity The protocol also takes care of message-integrity checks.

Let us first take a look at authentication in WPA. Figure 6.84 shows the diagrammatic view.

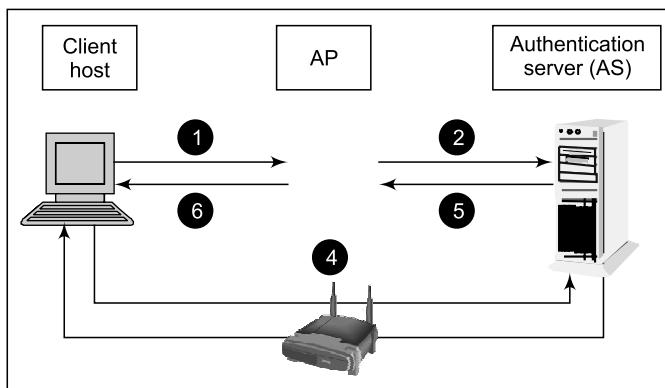


Fig. 6.84 Authentication in WPA

Let us understand this process step by step.

1. The client host (i.e. end user's computer) contacts the AP, with a request to get itself authenticated. For this purpose, it uses a protocol by the name EAP, which we shall discuss separately.
2. The AP passes the request on to the Authentication Server (AS). The AS is a **Remote Authentication Dial In User Service (RADIUS)** server. RADIUS is a networking protocol that uses access servers to provide centralized management of access to large networks. RADIUS is commonly used by ISPs and corporations managing access to the Internet or to any internal networks.
3. The AS sends a random challenge to the host computer.
4. When the user enters his/her password in the host computer, the received random challenge is encrypted with the help of that password. This encrypted random challenge is sent back to the AS.
5. The AS decrypts the random challenge sent by the user's host computer with the help of the user's password (which is known to AS). The AS compares the decrypted random challenge with the one it had created in step 3 before sending it to the client host. If the two match, it considers the user to be successfully authenticated and sends an appropriate message to the AP.

The AP opens a port for the user on the client host to be able to access the AP now.

■ 6.14 LINK SECURITY VERSUS NETWORK SECURITY ■

Many times, there is a confusion between *link security* and *network security*. Hence, a brief clarification is necessary.

In **link security**, the attempt is to cover a particular area of interest and make it secure. An example of this is the SSL/TLS protocol. As we have discussed in detail earlier, in this protocol, the aim is to secure the *link* between a Web browser (client) and a Web server (server). In other words, regardless of the underlying network mechanisms and hardware/software differences, here every single message exchanged by the client and the server is expected to be secured. Of course, SSL goes beyond just confidentiality and also provides authentication and message-integrity services.

Network security, on the other hand, attempts to cover the scope of an entire network and possibly beyond. An example of this is the use of firewalls to prevent unauthorized access and to scan every single packet. Therefore, the idea in network security is not necessarily to secure communication between two end points. Instead, the aim is to secure an entire network, which optionally may also include a link.

■ CASE STUDY 1: SECURE INTER-BRANCH PAYMENT TRANSACTIONS ■

Points for Classroom Discussions

1. What is the technology to achieve non-repudiation? How is this guaranteed?
2. How is the problem of key distribution resolved in PKI?
3. Why are cryptographic toolkits required?
4. How can smart cards be used in cryptography?

General Bank Of India (GBI) has implemented an Electronic Payment System called *EPS* in about 1200 branches across the country. This system transfers payment instructions between two computerized branches of GBI. A central server is maintained at the EPS office located in Mumbai. The branch offices connect to the Local VSAT of a private network by using dial-up connection. The local VSAT has a connectivity established with the EPS office. GBI utilizes its proprietary messaging service called as *GBI-Transfer* to exchange payment instructions.

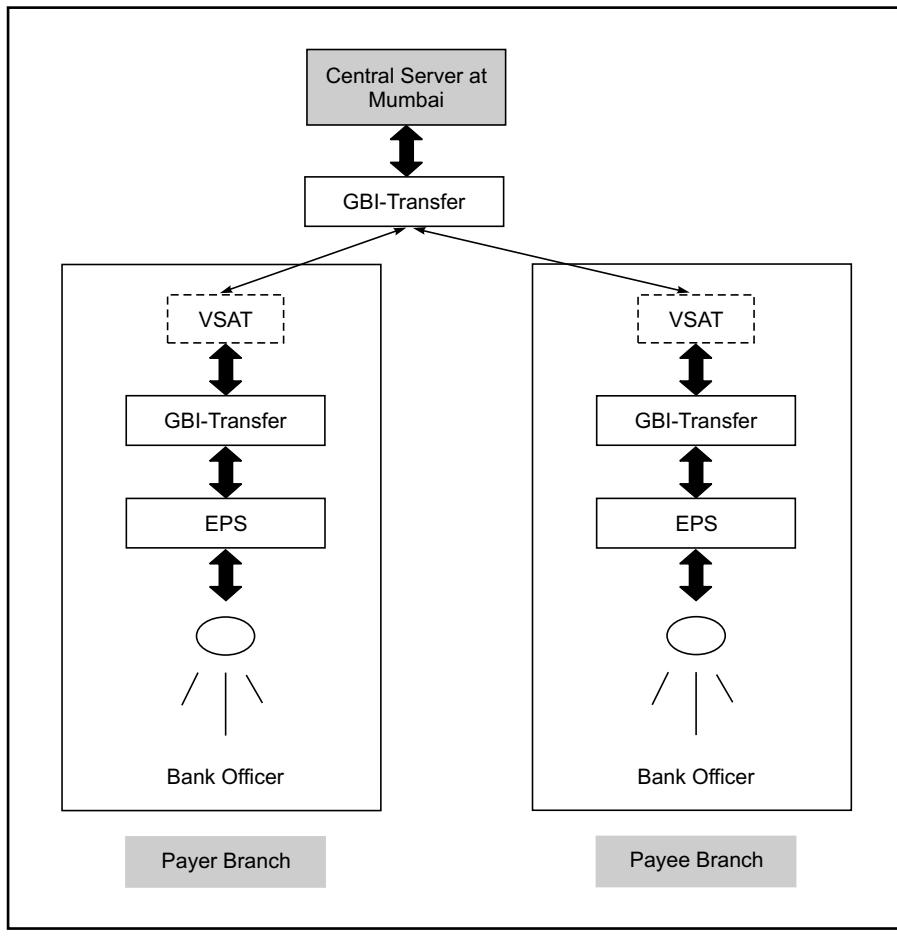
Currently, EPS has minimal data security. As the system operates in a closed network, the current security infrastructure may suffice the need. The data moving across the network is in encrypted format.

Current EPS Architecture

EPS is used to transmit payment details from the payer branch to the payee branch via the central server in Mumbai. The figure depicts the flow, which is also described step by step.

A typical payment transfer takes the following steps:

1. A data-entry person in the *Payer Branch* enters transaction details through the EPS interface.
2. A Bank Officer checks the validity of the transaction through the EPS interface.



EPS transaction flow

3. After validating the transaction, the Bank Officer authorizes the transaction. Authorized transaction is stored in a local Payment Master (PM) database.
4. Once the transaction is stored in PM, a copy of the same is encrypted and stored in a file. This transaction file is stored in OUT directory.
5. The *GBI-Transfer* application looks for any pending transactions (i.e. for the presence of any files in the OUT directory) by a polling mechanism and if it finds such transactions, it sends all these files one-by-one to the EPS central office located in Mumbai by dialing the local VSAT.
6. The local VSAT gets connectivity to the EPS central office and the transaction is transferred and stored in the IN directory at the EPS central office.
7. The interface program at the EPS central office collects the file pending in the IN directory and sends it to the PM application at that office.
8. In order to send the Credit Request to PM, the transaction headers are changed. The transaction with changed headers in encrypted format is then placed in OUT directory of the EPS central office.

9. The GBI-Transfer application at the EPS central office collects the transactions pending in the OUT directory and sends them to the *Payee Bank* through the VSAT.
10. The transaction is transferred and stored in the IN directory of the *Payee Branch*.
11. The interface program at the *Payee Branch* collects the transaction and posts it in PM.
12. PM marks the credit entry and returns back an acknowledgement of the same. The acknowledgement is placed in OUT directory of the *Payee Branch*.
13. The acknowledgement is picked by GBI-Transfer at the *Payee Branch* and sent to the EPS central office through the VSAT.
14. The EPS central office receives the credit acknowledgement and forwards it to *Payer Branch*.
15. The *Payer Branch* receives the credit acknowledgement receipt. This completes the transaction.

Requirements to Enhance EPS

As GBI is in the process of complete automation and setting up connectivity over the Internet or a private network, they need to ensure stringent security measures, which demand the usage of a Public Key Infrastructure (PKI) framework.

As a part of implementing security, GBI wants the following aspects to be ensured:

- Non-repudiation (Digital Signatures)
- Encryption – 128-bit (Upgrade to the current 56-bit encryption)
- Smart card support for storing sensitive data & on-card digital signing
- Closed loop Public Key Infrastructure

Proposed Solution

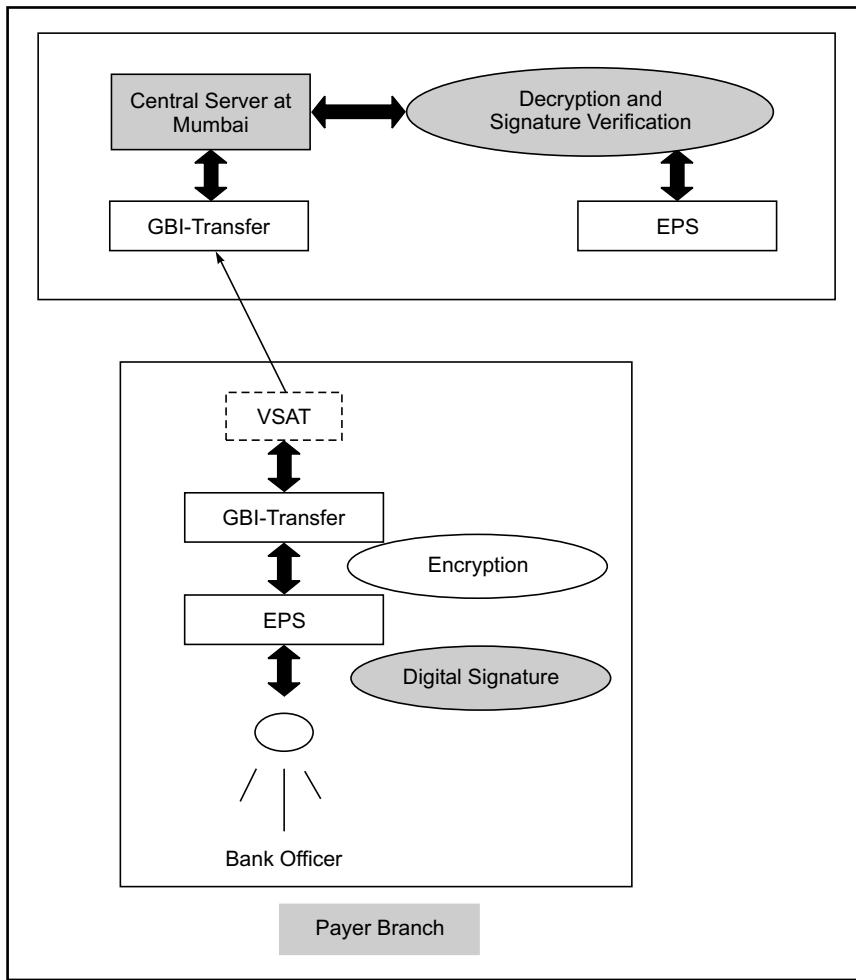
Since providing cryptographic functionalities require the usage of a cryptographic toolkit, it is assumed that GBI will implement an appropriate Certification Authority (CA) infrastructure and a PKI infrastructure offering.

The transaction will be digitally signed and encrypted/decrypted at the Payer and Payee branches, as well as at the EPS central office. The signing operation can be performed on the system or on external hardware like a smart card. On the server side, a provision of automated signing without any manual intervention will be provided.

The transaction flow described earlier would now be split into two legs:

- The Payer Leg (*Payer Branch* to the EPS central office)
- The Payee Leg (EPS central office to the *Payee Branch*)

The architecture for the Payer Leg is shown in the figure. As shown, after verifying the transaction, the EPS Officer authorizes the transaction at the *Payer Branch*. Internally, the application digitally signs the transaction. This signature, along with the transaction data is stored in the local PM Database and then encrypted and placed in the IN directory. For signature and encryption, a cryptographic toolkit is required at the *Payer Branch*. The signed-and-encrypted transaction is sent to the EPS central office in the same way as before.

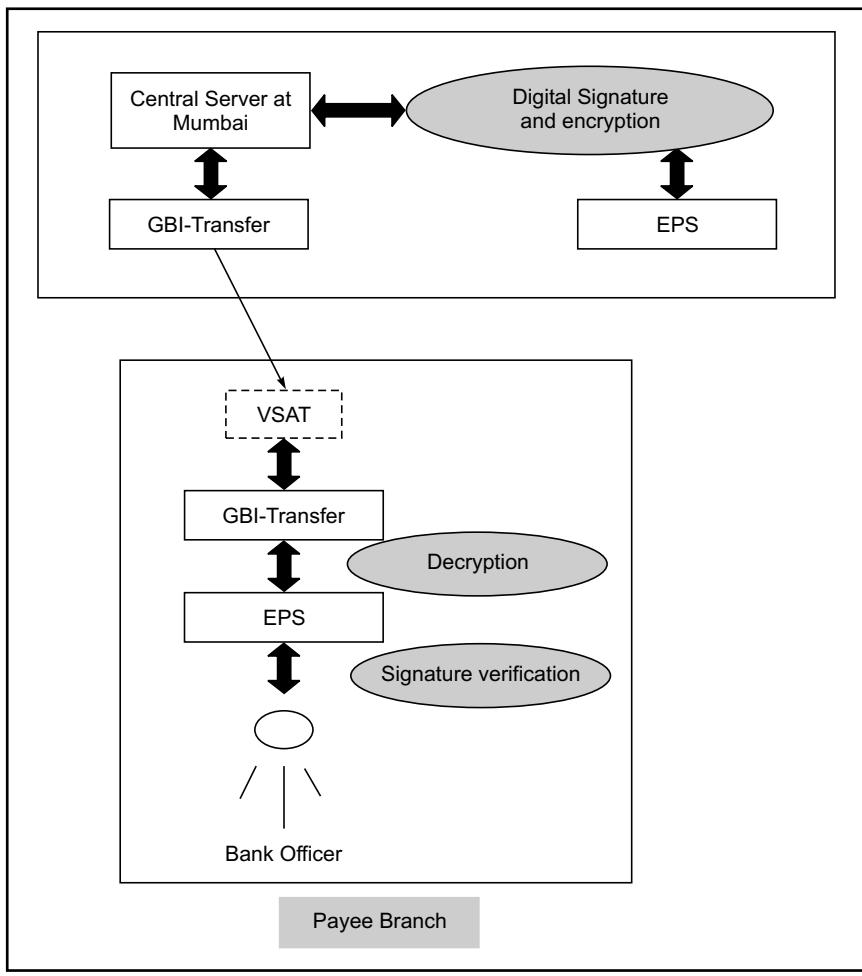


New EPS transaction flow at the Payer Branch

The encrypted file is decrypted at EPS central office. Before storing the transaction in the database, the digital signature is verified using an appropriate cryptographic toolkit. The verification process may also check the status of the user's digital certificate by either CRL or OCSP check. If the status of the certificate is invalid, the transaction will be rejected, otherwise it will be stored in the local PM database.

On the Payee Leg, the EPS central office will create a Credit Request as before, sign and encrypt it with the bank officer's digital certificate. This signed-and-encrypted request will be forwarded to the *Payee Branch*. The flow is shown in the figure.

In the Payee Leg, the PM software at the EPS central office will generate a Credit Request for the *Payee Bank*. This request will be digitally signed. The signature along with the Credit Request will be encrypted and sent to the *Payee Branch*.



New EPS transaction flow at the payee branch

The *Payee Branch* will decrypt the Credit Request and verify the digital signature. If the signature is verified successfully, the transaction is entered into database. Otherwise, it gets rejected and the status of the same is sent to EPS central Office. The Credit Response to the EPS central office can also be digitally signed and encrypted in a similar fashion.

■ CASE STUDY 2: COOKIES AND PRIVACY ■

Points for Classroom Discussions

1. Discuss the concept of a cookie.
2. What are the practical usages of cookies?
3. Which technologies can create and read cookies?
4. How can cookies damage privacy?

Cookies are simple text files stored on client computers. Whenever a user browses the Internet, the server-side code can create a small text file, called as cookie, on the user's computer. This is because the next time the user visits the same site again (during the same or a different session), the server can identify the client, based on the cookie. This overcomes the drawback of the HTTP protocol, which is stateless. This means that every request from a client, even to the same server in the same session, is treated as a completely new request.

Cookies can be transient (live for the lifetime of a session) or persistent (remain on the user's computer beyond the lifetime of a session). Cookies themselves cannot cause any damage to the user's computer, since they cannot contain executable code. However, cookies can be exploited to send targeted advertisements to the users, without their knowledge. This works as follows:

1. An advertising agency (say *XYZ Advertisements Limited*) contacts major Web sites and places banner ads for its corporate clients' products on their pages. It pays a fees to the site owners for this.
2. Instead of providing an image (GIF/JPEG), it provides a URL to add to each page.
3. Each URL contains a unique number in the *file* part, for example: <http://www.XYZAdverts.com/07041973.gif>
4. When a user visits a page P for the first time, the browser fetches the advertisement image from *XYZ* along with the main HTML page for the site it is visiting.
5. *XYZ* sends a cookie to the browser containing a unique user ID and records the relationship between this user ID and the file name.
6. Later, when the same user visits another page, the browser sees another reference to *XYZ*.
7. The browser sends the previous cookie to *Sneaky* and also fetches the current page from *XYZ*, as before.
8. *XYZ* knows that the same user has visited another Web page now.
9. It adds this reference to its database.
10. Over time, *XYZ* has a lot of information about the Web pages the user visits, the actions it performs, etc.
11. The *advertisement* from *XYZ* can be a single pixel in the same background color, making it even more difficult for the user to know that advertisements are appearing!



Summary

- The Internet uses the HTTP protocol for request-response, and TCP/IP for actual communication.
- Secure Socket Layer (SSL) is the world's most widely used protocol for securing communications on the Internet.
- SSL encrypts the connection between client and server.
- SSL provides encryption and message integrity services.
- SSL does not take care of digital signatures.
- SSL works between the application layer and the transport layer.
- SSL starts with a handshake between the client and the server.

- The SSL handshake establishes the necessary trust between the client and the server.
- The record protocol follows the handshake protocol in SSL.
- Alert protocol is used in SSL if one of the party senses an error.
- TLS is similar to SSL with a few differences.
- SHTTP encrypts individual messages, and works at the application layer.
- Time Stamping Protocol (TSP) is used to prove that a document existed at a particular point in time.
- Secure Electronic Transaction (SET) is a protocol devised by MasterCard and Visa jointly for secure credit-card payments on the Internet.
- SET involves many parties, such as cardholder, merchant, issuer, acquirer, payment gateway, and certification authority.
- In SET, the merchant does not know the customer's credit-card number.
- SET uses a mechanism called *dual signature* to achieve its objectives.
- SET is quite complex to implement.
- SET has not become popular, but it is good to study its specifications to understand how to design payment protocols on the Internet.
- 3-D Secure is an enhancement to SET.
- 3-D Secure demands user authentication while doing any transaction. This resolves the problem of someone using another person's credit-card details, to a certain extent.
- Electronic money is the computer representation of money.
- Electronic money can be online or offline, identified or anonymous.
- In online electronic money, the bank is actively involved when a transaction happens.
- In offline electronic money, the bank is not actively involved when a transaction happens.
- In identified electronic money, the money can be traced back to the person who had originally obtained it from the bank.
- In anonymous electronic money, the money cannot be traced back to the person who had originally obtained it from the bank.
- Anonymous offline money can lead to the double-spending problem.
- Email security can be achieved by the PEM, PGP, and S/MIME protocols.
- Privacy Enhanced Mail (PEM) is no longer a widely used protocol. However, it is good to study, since other protocols are variations of the basic PEM scheme.
- PEM provides services such as encryption, message digest, and digital signatures.
- S/MIME adds security to the email protocol called Multipurpose Internet Mail Extension (MIME).
- MIME allows non-text data to be sent via email.
- S/MIME secures MIME contents in the form of encryption, message digests, and digital signatures.
- The output of S/MIME is a PKCS object.
- S/MIME is quite useful these days, since it is quite common to send multimedia and binary data using email.

- Pretty Good Privacy (PGP) is perhaps the world's most popular email security protocol.
- The services offered by PGP are similar to those by PEM and S/MIME.
- PGP is quite simple to understand and implement.
- PGP supports either digital certificates or key rings to establish trust between users.
- PGP has interesting mechanisms to create trust relationships, namely introducer trust, certificate trust, and key legitimacy.
- Wireless Transport Layer Security (WTLS) provides security in WAP.
- WTLS is similar in concept to SSL, but is optimized for mobile devices.
- GSM security is located in the lower layers.
- GSM security can be in addition to the WTLS security.
- 3G mobile systems also have their own mechanisms for security now.



Key Terms and Concepts

- | | |
|--|--|
| <ul style="list-style-type: none"> ● 3-D Secure ● Acquirer ● Advanced Mobile Phone System (AMPS) ● Anonymous electronic money ● Blinded number ● Cardholder ● D-AMPS ● Double spending problem ● Dynamic Web page ● Electronic money ● General Packet Radio Service (GPRS) ● Handshake protocol in SSL ● Introducer ● Issuer ● Key ring ● Master secret in SSL ● Multipurpose Internet Mail Extension (MIME) ● Online electronic money ● Pre-master secret in SSL ● Privacy Enhanced Mail (PEM) ● Secure Electronic Transaction (SET) | <ul style="list-style-type: none"> ● 3G (Third Generation of mobile networks) ● Active Web page ● Alert protocol in SSL ● Base-64 encoding ● Blinding factor ● Certificate trust ● Digital cash ● Dual signature ● Electronic cash ● Fingerprint ● Global System for Mobile Communications (GSM) ● Identified electronic money ● Introducer trust ● Key legitimacy ● Lempel-Ziv algorithm ● Merchant ● Offline electronic money ● Origin server ● Pretty Good Privacy (PGP) ● Record protocol in SSL ● Secure MIME (S/MIME) |
|--|--|

- Secure Socket Layer (SSL)
- Time Stamping Authority (TSA)
- Transport Layer Security (TLS)
- Universal Mobile Telephone System (UMTS)
- Web of trust
- Wireless Transport Layer Security (WTLS)
- Static Web page
- Time Stamping Protocol (TSP)
- Transmission Control Protocol/Internet Protocol (TCP/IP)
- WAP gateway
- Wireless Access Protocol (WAP)



PRACTICE SET

■ Multiple-Choice Questions

1. SSL works between _____ and _____.
 - (a) Web browser, Web server
 - (b) Web browser, application server
 - (c) Web server, application server
 - (d) application server, database server
2. SSL layer is located between _____ and _____.
 - (a) transport layer, network layer
 - (b) application layer, transport layer
 - (c) data-link layer, physical layer
 - (d) network layer, data-link layer
3. _____ in SSL is optional.
 - (a) Server authentication
 - (b) Database authentication
 - (c) Application authentication
 - (d) Client authentication
4. The record protocol is the _____ message in SSL.
 - (a) first
 - (b) second
 - (c) last
 - (d) none of the above
5. The _____ protocol is similar to SSL.
 - (a) HTTP
 - (b) HTTPS
 - (c) TLS
 - (d) SHTTP
6. The main purpose of SET is related to _____.
 - (a) secure communication between browser and server
 - (b) digital signatures
 - (c) message digests
 - (d) secure credit card payments on the Internet
7. SET uses the concept of _____.
 - (a) double signature
 - (b) dual signature
 - (c) multiple signature
 - (d) single signature
8. Credit-card details are not available to the _____ in SSL.
 - (a) merchant
 - (b) customer
 - (c) payment gateway
 - (d) issuer
9. Many standard Web-based email services implement.
 - (a) DKIM
 - (b) PGP
 - (c) PEM
 - (d) SET

10. The _____ protocol overcomes the drawbacks of WEP.
 - (a) WiFi
 - (b) WiMax
 - (c) WPA
 - (d) 802.11
11. PEM allows for _____ security options.
 - (a) 2
 - (b) 3
 - (c) 4
 - (d) 5
12. The _____ protocol needs to identify the content type before email can be transmitted.
 - (a) PEM
 - (b) PGP
 - (c) SMTP
 - (d) MIME
13. In _____, we have the concept of key rings.
 - (a) PEM
 - (b) PGP
 - (c) SMTP
 - (d) MIME
14. In _____, the user needs to authenticate before using a credit card in an electronic transaction.
 - (a) SET
 - (b) SSL
 - (c) 3-D secure
 - (d) WTLS
15. The security layer in WAP is between the _____ layer and the _____ layer.
 - (a) transaction, transport
 - (b) application, transport
 - (c) transport, physical
 - (d) session, transport

■ Exercises

1. Why is the SSL layer positioned between the application layer and the transport layer?
2. What is the purpose of the SSL alert protocol?
3. Explain the SSL handshake protocol.
4. How is SHTTP different from SSL?
5. What is the significance of the time-stamping protocol?
6. Which are the key participants in SET?
7. How does SET protect payment information from the merchant?
8. Outline the broad-level steps in SET.
9. How is 3-D Secure different from SET?
10. Explain WEP and WPA.
11. What is DKIM? What purpose does it serve?
12. Mention the broad-level steps in PEM and PGP.
13. Explain the concept of key rings in PGP.
14. What is the security concern in WAP?
15. How does GSM security work?

■ Design/Programming Exercises

1. Many real-life algorithms, including RSA and SSL; use the principle of GCD. Find out why the GCD of any two consecutive integers, n and $n + 1$ is always 1. (*Hint:* Refer to Appendix A).
2. Find out the GCD of two numbers: 42120 and 46510.

3. Write a Java implementation of the GCD method provided in Appendix A (which provides a similar implementation in C).
4. Write a C program to test whether an inputted number is prime or not.
5. Achieve the same objective as above, using Java.
6. Assume a 24-bit input as 101010111100000101100110, and transform it into its Base-64 equivalent, using the algorithm shown in this chapter.
7. Write a C program to perform Base-64 encoding on a 24-bit input.
8. Think about what will happen if we do not have Base-64 encoding mechanism.
9. Can we implement the concept of key rings on the Internet (e.g. when we purchase something over the Internet)? Why?
10. Consider the following text:
Welcome to the world of security. The world of security is full of interesting problems and solutions.
How would the Lempel-Ziv algorithm compress this text? Consider a conceptual view, where we want to compress (replace) the words *to*, *the*, *of* and *security*.
11. Welcome to the world of security. The world of security is full of interesting problems and solutions.
12. How would the Lempel-Ziv algorithm compress this text? Consider a conceptual view, where we want to compress (replace) the words *to*, *the*, *of* and *security*.
13. Investigate more about the differences between GSM and 3G technologies from a security stand point.
14. If we implement both WTLS and GSM/3G security, would it offer any extra security? Would one of these be redundant? Why?
15. SSL talks about security at the transport layer. What if we want to enforce security at the lower layers?
16. Investigate how to use SSL in Java and .NET. Write an SSL client and server in these technologies.

Apache has implemented an open-source library of SSL components. Investigate more and implement SSL using it.



USER-AUTHENTICATION MECHANISMS

■ 7.1 INTRODUCTION ■

One of the key aspects of cryptography and network/Internet security is authentication. Authentication helps establish trust by identifying the particular user/system. Authentication ensures that the claimant is really who he/she claims to be. This chapter discusses the various aspects of authentication mechanisms.

There are many ways to authenticate a user. Traditionally, user ids and passwords have been used. But there are many security concerns in this mechanism. Passwords can travel in clear text or can be stored in clear text on the server, both of which are dangerous propositions. Modern password-based authentication techniques use alternatives as encrypting passwords, or using something derived from the passwords in order to protect them.

Authentication tokens add randomness to the password-based mechanism, and make it far more secure. This mechanism requires the user to possess the tokens. Authentication tokens are quite popular in applications that demand high security.

Certificate-based authentication has emerged as a modern authentication mechanism, thanks to the emergence of the PKI technology. This is also quite strong, if implanted correctly. Smart cards can also be used in conjunction with this technology. Smart cards facilitate cryptographic operations inside the card, making the whole process a lot more secure and reliable.

Biometrics is also getting a lot of attention these days, and is based on human biological characteristics. However, it has still not matured completely.

This chapter examines all these mechanisms of authentication in great detail. It discusses the advantages and drawbacks of each one of them. The chapter then concludes with the coverage of Kerberos, a single sign on mechanism implemented in many real-life systems.

■ 7.2 AUTHENTICATION BASICS ■

Who are you? This is a question that we ask everyday and get asked. It has a lot of importance and significance in the world of cryptography. As we studied earlier, the whole concept of authentication

is based on determining who an individual user is, before allowing the user to go ahead and perform actual business transactions using the system.

Authentication can be defined as determining an identity to the required level of assurance. Authentication is the first step in any cryptographic solution. We say this because unless we know who is communicating, there is no point in encrypting what is being communicated. As we know, the whole purpose of encryption is to secure communication between two or more parties. Unless we are absolutely sure that the parties really are what they claim to be, there is no point in encrypting the information flowing between them. Otherwise, there is a chance that an unauthorized user can access the information. In cryptographic terms, we can put this in other words: there is no use of encryption without authentication.

We see authentication checks many times every day. We are required to wear and produce our identity cards at work, whenever demanded. To use our ATM card, we must make use of the card as well as the PIN. Many such examples can be given.

The whole idea of authentication is based on secrets. Most likely, the entity being authenticated and the authenticator both share the same secret (e.g. the PIN in the ATM example). Another variation of this technique is the case where the entity being authenticated knows a secret, and the authenticator knows a value that is derived from the secret. We shall study this during the course of this chapter.

■ 7.3 PASSWORDS ■

7.3.1 Introduction

Passwords are the most common form of authentication. A password is a string of alphabets, numbers and special characters, which is supposed to be known only to the entity (usually a person) that is being authenticated. There are great myths about passwords. People believe that the use of passwords is the simplest and the least expensive authentication mechanism, because it does not require any special hardware or software support. However, as we shall see, this is quite wrong a perception!

7.3.2 Clear-Text Password

1. Its Working

This is the simplest password-based authentication mechanism. Usually, every user in the system is assigned a user id and an initial password. The user changes the password periodically for security reasons. The password is stored in clear text in the user database against the user id on the server. The authentication mechanism works as described below.

Step 1: Prompt for User Id and Password During authentication, the application sends a screen to the user, prompting for the user id and password. This is shown in Fig. 7.1.

Step 2: User Enters User Id and Password The user enters his/her id and password, and presses the *OK* (or an equivalent) button. This causes the user id and password to travel in clear text to the server. This is shown in Fig. 7.2.

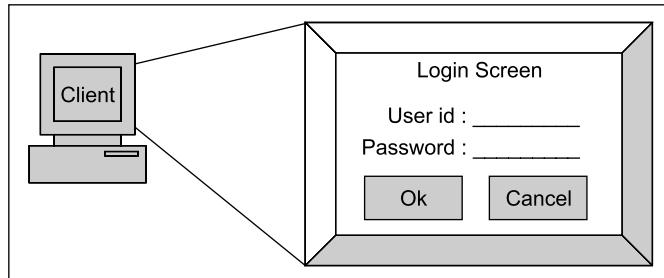


Fig. 7.1 Prompt for user id and password

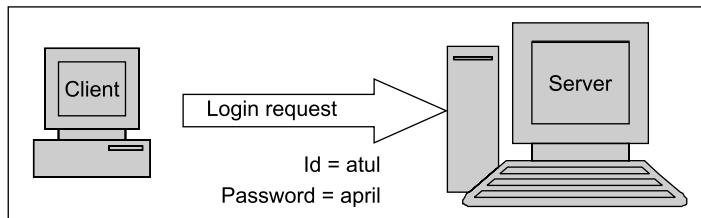


Fig. 7.2 User id and password travel in clear text to the server

Step 3: User Id and Password Validation The server consults the user database to see if this particular user id and password combination exists there. Usually, this is the job of a **user-authenticator** program, as shown in Fig. 7.3. This is a program that takes a user id and password, checks it against the user database, and returns the result of the authentication (success or failure). Of course, there are many ways to do this.

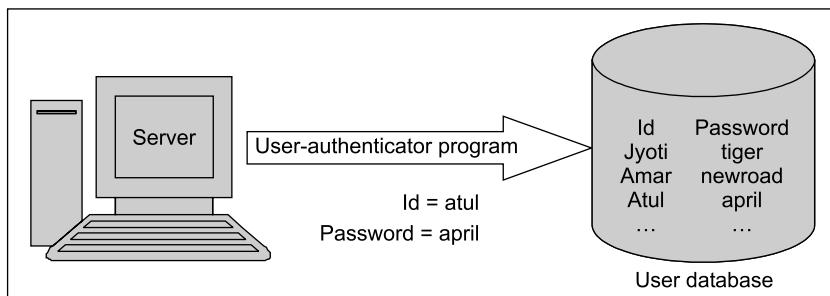


Fig. 7.3 User authenticator checks the user id and password against the user database

Step 4: Authentication Result Depending on the success or failure of the validation of the user id and the password, the user-authenticator program returns an appropriate result back to the server. This is shown in Fig. 7.4. Here, we assume that the user was authenticated successfully.

Step 5: Inform User Accordingly Depending on the outcome (success/failure), the server sends back an appropriate screen to the user. If the user authentication was successful, the server typically sends a menu of options for the user, which lists the actions the user is allowed to perform. If the result of the user authentication was a failure, the server sends an error screen to the user. This is shown in Fig. 7.5. Here, we assume that the user was authenticated successfully.

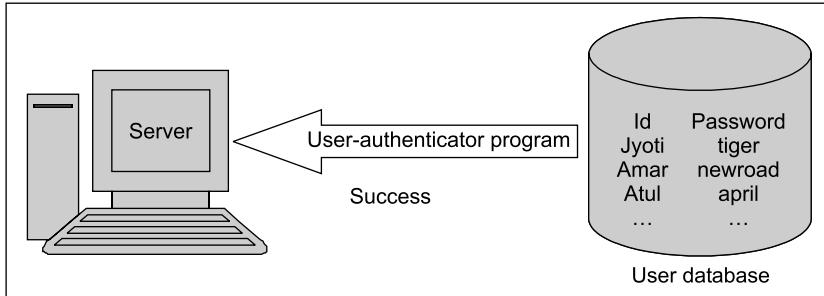


Fig. 7.4 User-authenticator program returns a success or failure message to the server

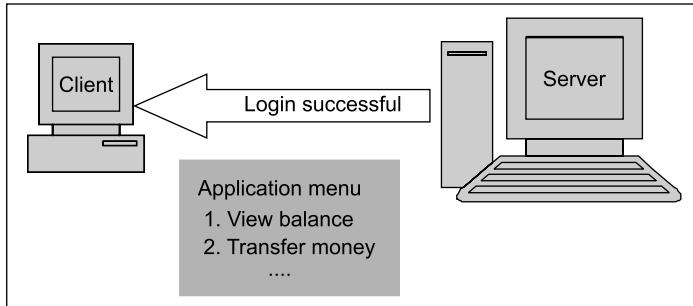


Fig. 7.5 Server returns a success or failure result back to the user

2. Problems with the Scheme

As we can see, this approach is not secure at all. There are two major problems in this approach:

(a) Problem 1—Database Contains Passwords in Clear Text Firstly, the user database contains user ids and passwords in clear text. Therefore, if an attacker succeeds in obtaining an access to the database, the whole list of user ids and passwords is available to the attacker. Consequently, it is advised that the passwords should not be stored in clear text in the database. Instead, they should first be encrypted and then stored in the database. Whenever a user attempts to log on, on the server side, the user's password should first be encrypted, compared with the encrypted password in the database, and depending on whether they match or not, a decision should be taken. This is shown in Fig. 7.6.

(b) Problem 2—Password Travels in Clear Text from the User's Computer to the Server Even if we store encrypted passwords in the database, the password would travel in clear text from the user to the server. Therefore, if an attacker breaks into the communication link between the user's computer and the server, the attacker can easily obtain the clear-text password. We shall study how this problem can be tackled in the next section.

7.3.3 Something Derived from Passwords

1. Introduction

The variation from the basic password-based authentication is not to use the password itself, but to use something that is *derived from* the password. That is, instead of storing the password as it is, or in an

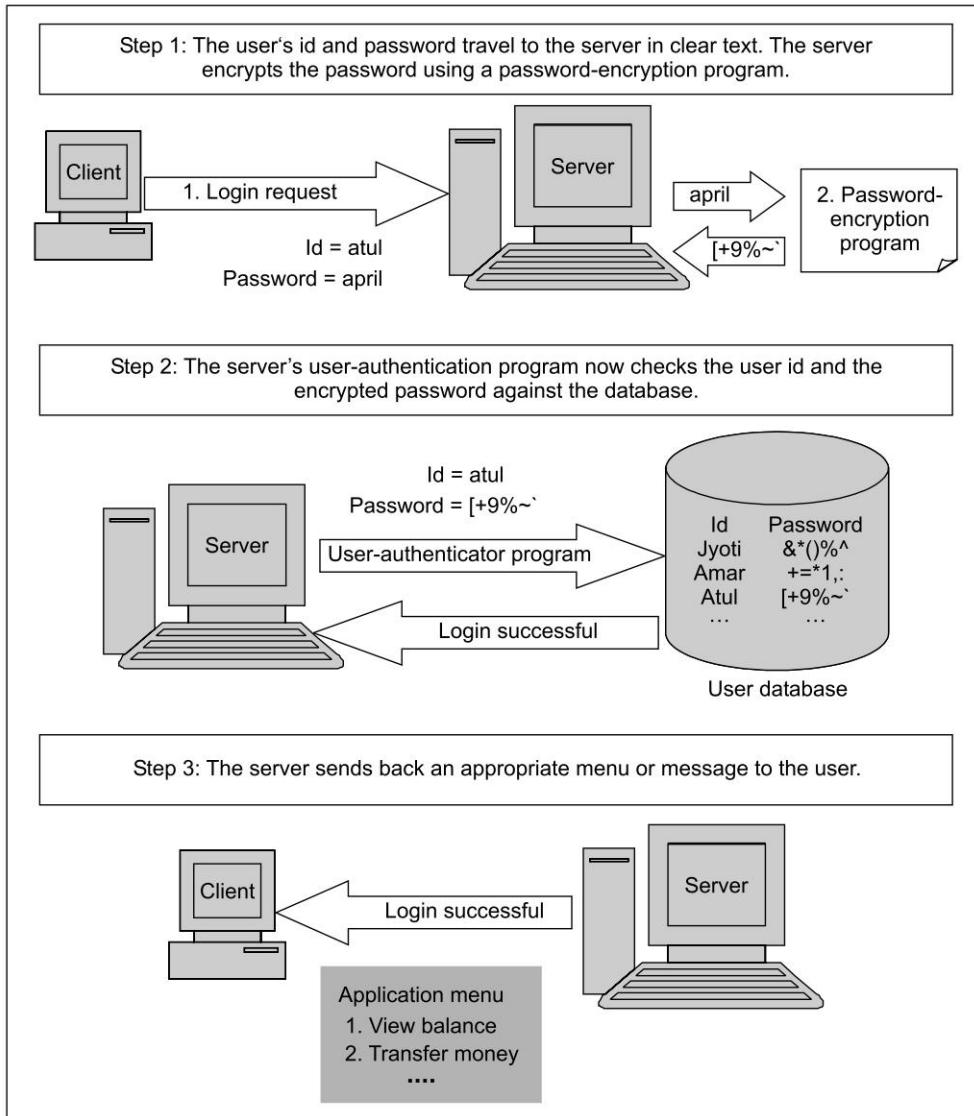


Fig. 7.6 Encrypting passwords before they are stored and verified

encrypted format, we can run some algorithm on the password and store the output of this algorithm as the (derived) password in the database. When the user wants to be authenticated, the user enters the password, the user's computer performs the same algorithm locally, and sends the derived password to the server, where it is verified.

Several requirements need to be met to ensure that this scheme works correctly:

- Each time the algorithm is executed for the same password, it must produce the same output.
- The output of the algorithm (i.e. something derived from the password) must not provide any clues regarding the original password.

- It should be infeasible for an attacker to provide an incorrect password, and yet obtain the correct derived password.

As we can see, these requirements closely match those of a message digest, so an algorithm such as MD5 or SHA-1 can be a suitable fit. Therefore, using message digests of passwords is one good option. We shall study this and the other options now.

2. Message Digests of Passwords

A simple technology to avoid the storage and transmission of clear-text passwords is the use of message digests. Let us understand how this works.

Step 1: Storing Message Digests as Derived Passwords in the User Database Rather than storing passwords, we can store the message digests of the passwords in the database, as shown in Fig. 7.7.

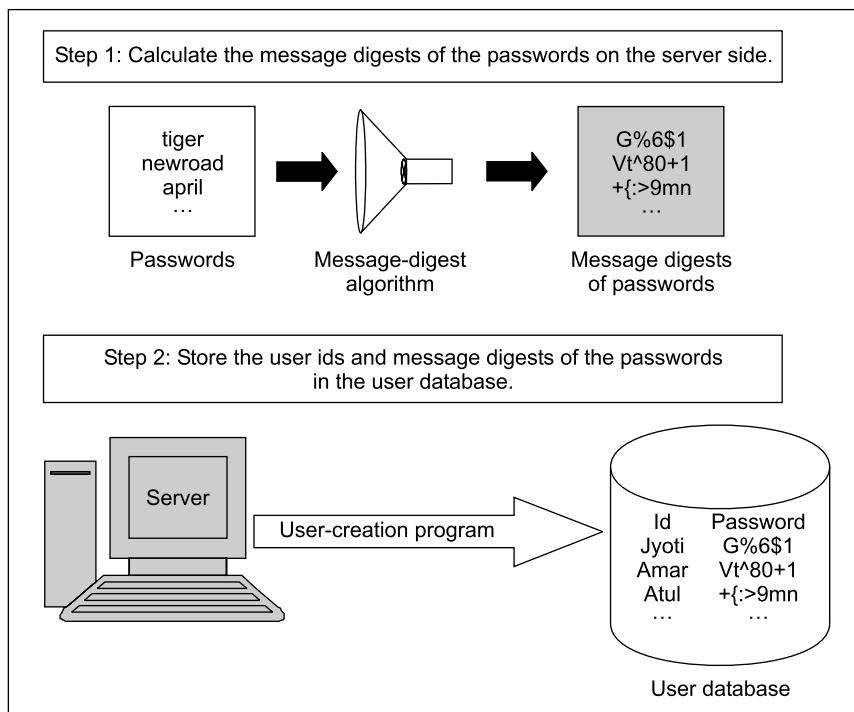


Fig. 7.7 Storing message digests of the passwords in the user database

Step 2: User Authentication When a user needs to be authenticated, the user enters the id and password, as usual. Now, the user's computer computes the message digest of the password, and sends the user id and the message digest of the password to the server for authentication. This is shown in Fig. 7.8.

Step 3: Server-side Validation The user id and the message digest of the password travel to the server over the communication link. The server passes these values to the user-authentication program,

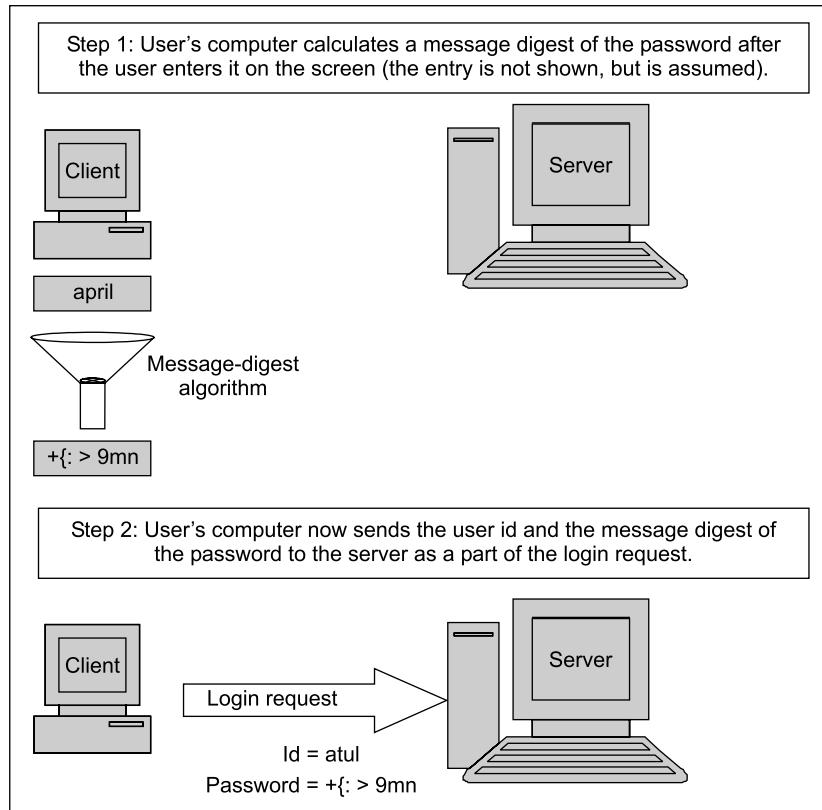


Fig. 7.8 User authentication involving message digest of the password

which validates the user id and the message digest of the password against the database, and returns an appropriate response back to the server. The server uses the result of this operation to return an appropriate message back to the user. This is shown in Fig. 7.9.

Is this approach of using the message digests of passwords completely secure then? Let us review our original requirements:

- Each time the algorithm is executed for the same password, it must produce the same output.
- The output of the algorithm (i.e. something derived from the password) must not provide any clues regarding the original password.
- It should be infeasible for an attacker to provide an incorrect password, and yet obtain the correct derived password.

The use of message digests will guarantee that all of the above conditions are satisfied. Therefore, can we safely say that this is a safe scheme? Well, not quite! The attacker may not be able to use the message digest to work backwards and retrieve the original password. But the fact is, he/she need not even attempt that! The attacker can simply *listen* to the communication involving a login request-response pair between a user's computer and the server. As we know, this would involve the transmission of the user id and the message digest of the password from the user's computer to the server. The attacker can simply copy the user id and the message digest of the password, and submit them after some time to the

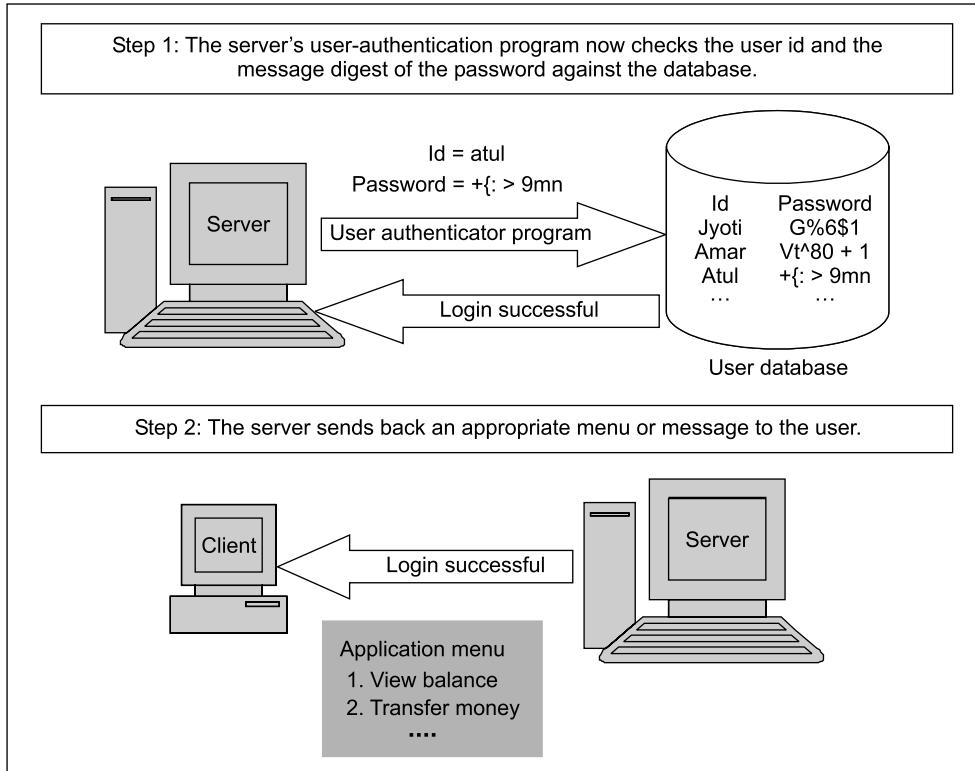


Fig. 7.9 User-authenticator program validates the user id and the message digest of the password

same server as part of a new login request. The server has no way of knowing that this login attempt is not from a legitimate user, but that it is actually from an attacker. Therefore, the server would authenticate the attacker successfully! This is called a **replay attack**, because the attacker simply *replays* the sequence of the actions of a normal user.

Therefore, we need better schemes.

3. Adding Randomness

To improve the security of the solution, we need to add a bit of unpredictability or randomness to the earlier scheme. This is to ensure that although the message digest of the password is always the same, the exchange of information between the user's computer and the server is never the same. This will ensure that a *replay attack* is foiled. This can be achieved by using a simple technique, as discussed below.

Step 1: Storing Message Digests as Derived Passwords in the User Database This step is exactly the same as in the previous case, and therefore, we shall not discuss it here again. We simply store the message digests of the user passwords, and not the passwords themselves, in our user database.

Step 2: User Sends a Login Request This is an intermediate step, unlike our previous login processes. Here, the user sends the login request only with his/her user id (and neither the password, nor the message digest of the password). This is shown in Fig. 7.10. We shall use this concept on many

occasions. As we progress further, we will notice that this results into two different login requests from the user to the server, the first one containing only the user id, and the second one containing some additional information.

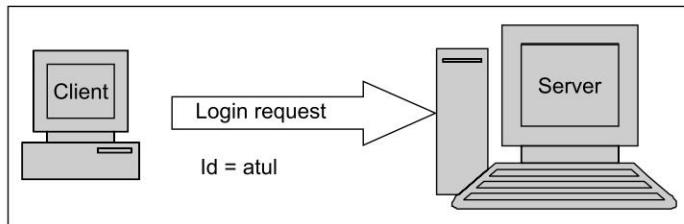


Fig. 7.10 Login request now contains only the user id

Step 3: Server Creates a Random Challenge When the server receives the user's login request containing the user id alone, it first checks to see if the user id is a valid one (note that only the user id is checked). If it is not, it sends an appropriate error message back to the user. If the user id is valid, the server now creates a **random challenge** (a random number, generated using a pseudo-random number generation technique), and sends it back to the user. The random challenge can travel as plain text from the server to the user's computer, as shown in Fig. 7.11.

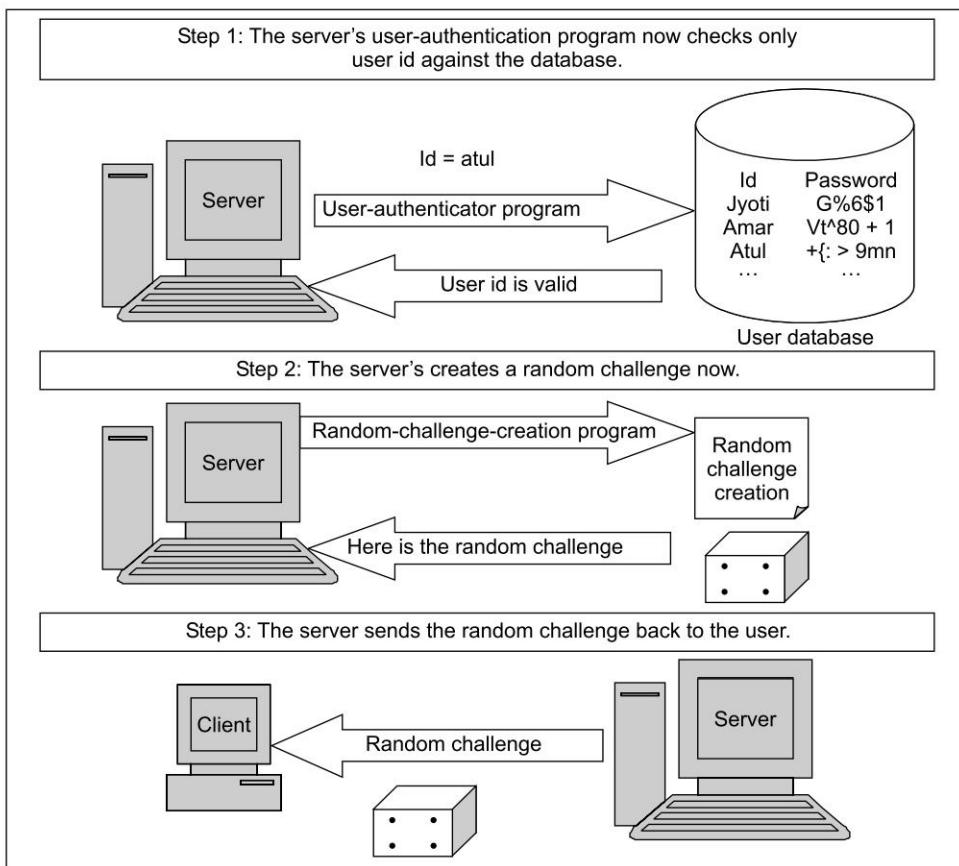


Fig. 7.11 Random-challenge creation and transmission to the user

Step 4: User Signs the Random Challenge with the Message Digest of the Password At this stage, the application displays the password-entry screen to the user. In response, the user enters the password on the screen. The application executes the appropriate message-digest algorithm on the user's computer to create a message digest of the password entered by the user. This message digest of the password is now used to encrypt the random challenge received from the server. This encryption is, of course, of a symmetric-key encryption form. This is shown in Fig. 7.12.

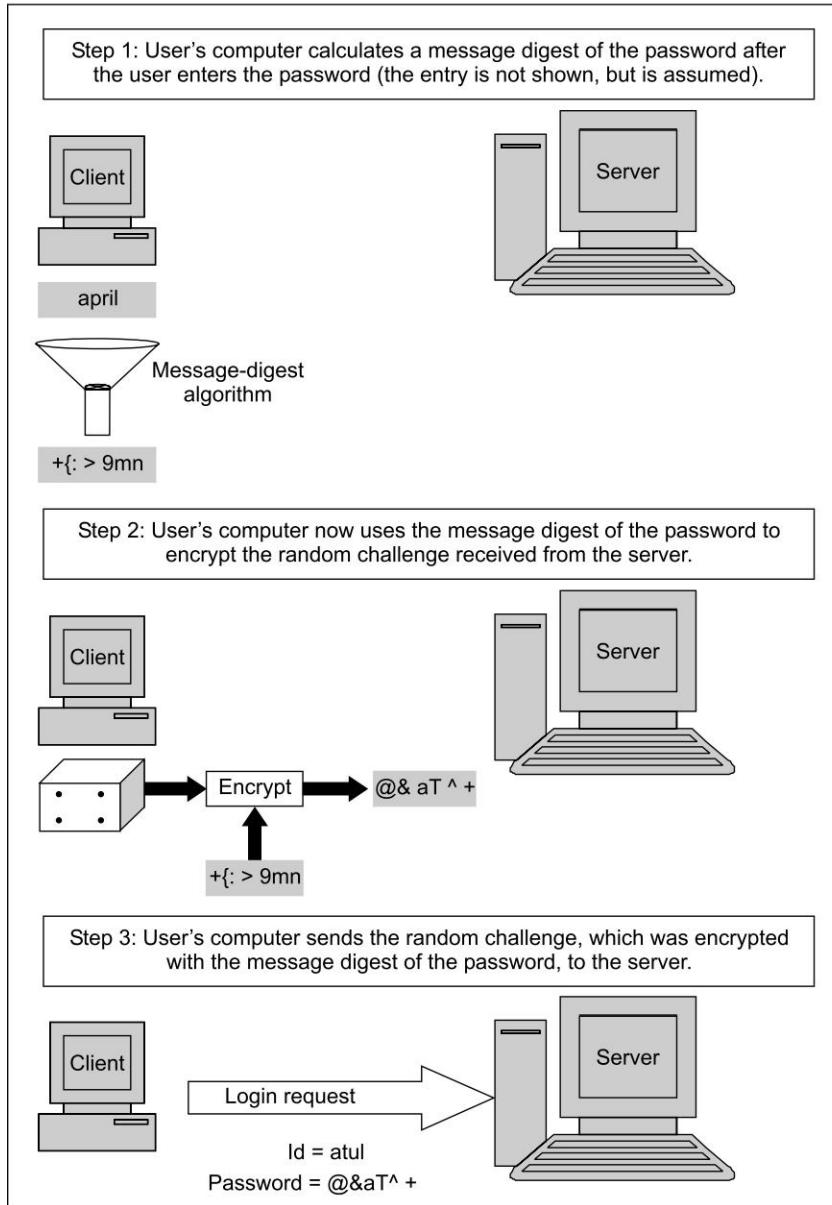


Fig. 7.12 User sends the user id and encrypted random challenge to the server

Step 5: Server Verifies the Encrypted Random Challenge Received from the User The server receives the random challenge, which was encrypted by the password of the user's message digest. In order to verify that the random challenge was indeed encrypted by the message digest of the user's password, the server must perform an identical operation. This can be done in two ways:

- The server can decrypt the encrypted random challenge received from the user with the message digest of the user's password. As we know, the message digest of the user's password is available to the server via the user database. If this decryption matches the original random challenge available on the server, the server can be assured that the random challenge was indeed encrypted by the message digest of the user's password.
- Alternatively, the server can simply encrypt its own version of the random challenge (i.e. the one which was sent earlier to the user) with the message digest of the user's password. If this encryption produces an encrypted random challenge, which matches with the encrypted random challenge received from the user, the server can be assured that the random challenge was indeed encrypted by the message digest of the user's password.

Regardless of the mode of operation, the server can thus determine if the user is really what he/she claims to be. We illustrate the second approach in Fig. 7.13. We could have as well used the first approach, instead.

Step 6: Server Returns an Appropriate Message back to the User Finally, the server sends an appropriate message back to the user, depending on whether the previous operations yielded success or failure. This is shown in Fig. 7.14.

Note that the random challenge is different every time. Therefore, the random challenge encrypted with the message digest of the password would also be different every time. Therefore, an attacker attempting a *replay attack* is quite unlikely to succeed now. This is the basis for many real-life authentication mechanisms, including Microsoft Windows NT 4.0. Windows NT 4.0 uses the MD4 message-digest algorithm to produce the message digests of the passwords, and uses 16-bit random numbers as the random challenges.

One variation of this scheme is to use the password (and not the message digest of the password) to encrypt the random challenge, both at the user's computer, as well as on the server. However, this approach has the drawback of storing the user's password in the user database. Therefore, it is not recommended.

4. Password Encryption

To resolve our earlier problem of the transmission of clear-text passwords, we can first encrypt the password on the user's computer, and then send it to the server for authentication. This means that we must provide for some sort of cryptographic functionalities on the user's computer (i.e. the client side). In fact, this functionality is needed even in the approach of encrypting the random number with the message digest of the user's password. This is not a problem in the case of client-server applications, where the client is capable of performing computations anyway. However, in the case of Internet applications, where the client is a Web browser, which does not have any special programming capabilities, this can be a problem. Consequently, we must resort to technologies such as Secure Socket Layer (SSL). That is, a secure SSL connection must be established between the client and the server, which would involve the step wherein the client verifies the server's authenticity based on the server's digital

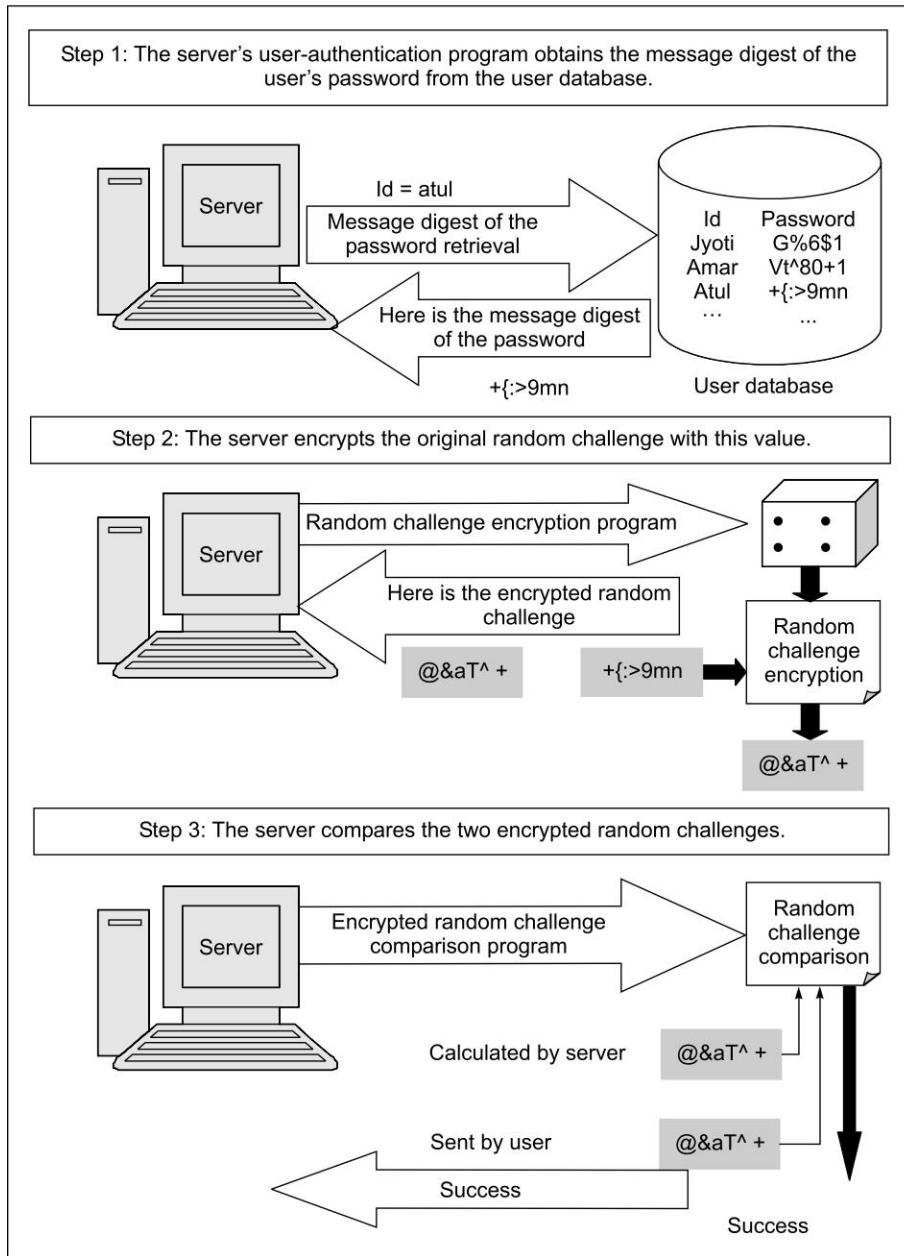


Fig. 7.13 Server compares the two encrypted random challenges

certificate. After this, all the communication between the client and the server will be encrypted using SSL, so the password need not have any application-level protection mechanisms now. SSL would perform the required encryption operations. This is shown in Fig. 7.15.

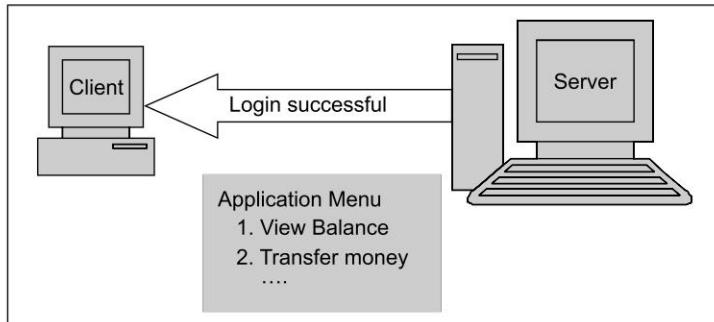


Fig. 7.14 Server sends an appropriate message back to the user

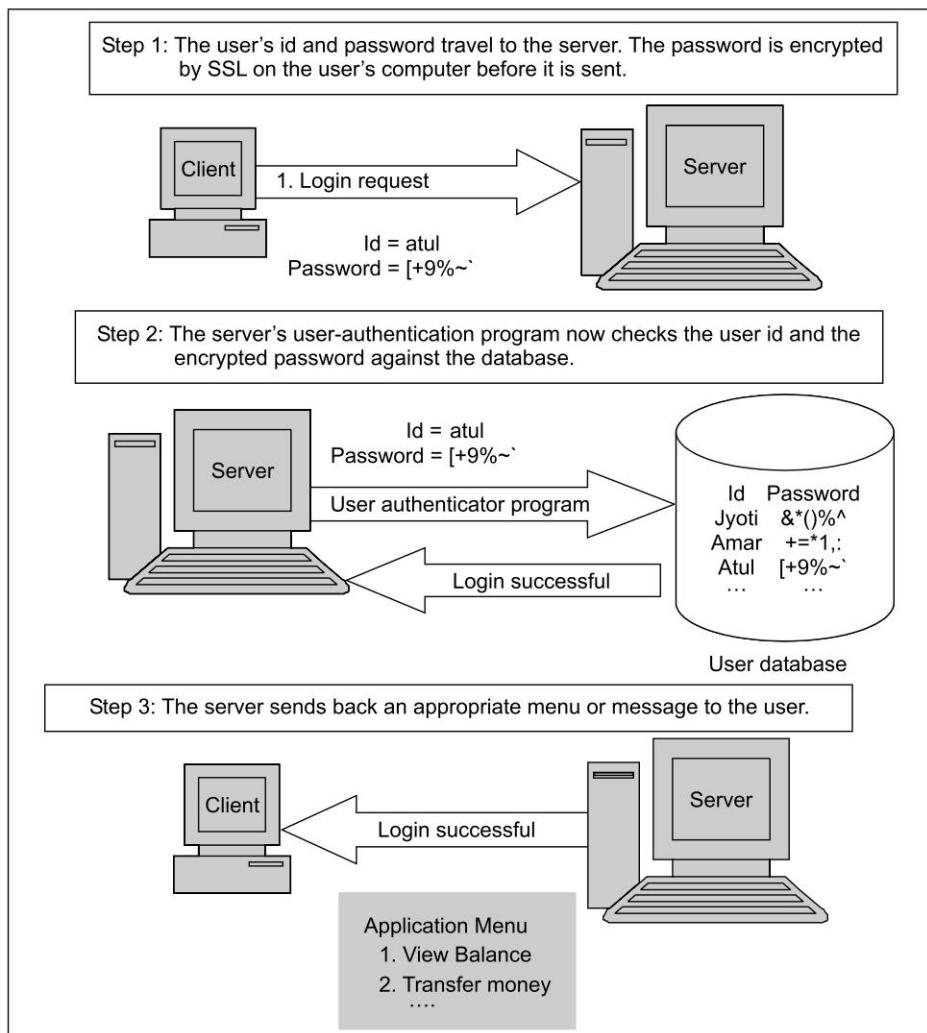


Fig. 7.15 Encrypting password on the client's computer and also in the database

Of course, the figure is not hundred per cent correct. We have two encryption processes:

- The first encryption happens before a password is stored in the user database.
- The other encryption is performed on the user's computer to encrypt the password before it is transmitted to the server.

These two encryption operations are in no way directly related to each other. They may even be using totally different approaches to encryption (for example, the user's computer would use the symmetric key shared between the user and the server first for encryption and then the SSL session key for secure transmission, whereas the server could only use the shared symmetric key, as it does not have to perform any transmission). Therefore, the encrypted password in the database would not actually be the same as the encrypted password coming from the user's computer. However, the main idea here is that both are encrypted passwords—neither of them is in clear text. That is the purpose of this illustration. The fact that the encrypted versions of these two passwords may not be the same, and that the server-side application logic would perform the necessary conversions between the two for verification is a minor technical variation, which we shall ignore.

7.3.4 The Problems with Passwords

As we mentioned, a major misconception is that passwords are the simplest and the cheapest form of authentication mechanisms. An end user might actually be right in having this perspective. However, from an application or system administration point of view, this is quite incorrect.

Typically, an organization has a number of applications, networks, shared resources, and intranets. Worse yet, these applications have varying needs of security measures, and they grow over a period of time. Therefore, each such resource demands its own user id and password. This means that an end user has to remember and correctly use many user ids and passwords. To overcome the troubles associated with this, most users either keep the same password for all the resources, or record their passwords at some place. These places can be really weird. For instance, many users keep the passwords in their electronic diaries, or paper diaries, or cupboards, below the keyboard, or sometimes even stick to their monitor! Either way, this creates great concerns for the security of the passwords and, therefore, the access of the resources.

Password maintenance is a very big concern for system administrators. A study shows that system administrators spend about 40% of their time creating, resetting or changing user passwords! This can truly be a nightmare for them.

Organizations specify **password policies**, which mandate the structure of passwords. For instance, an organization policy could have some of the following policies governing the passwords of its users:

- The password length must be at least 8 characters.
- It must not contain any blanks.
- There must be at least one lower-case alphabet, one upper-case alphabet, one digit and one special character in the password.
- The password must begin with an alphabet.

As we can see, this (like a salt in PBE, as discussed earlier) can be a significant deterrent to dictionary attacks, whereby attackers simply take normal words (from a dictionary) and try them as passwords. However, this creates a problem of remembering cryptic passwords for the end users. Therefore, end

users resort to writing their passwords somewhere, which can defeat the whole purpose of a password policy!

In a nutshell, there are no easy solutions here!

■ 7.4 AUTHENTICATION TOKENS ■

7.4.1 Introduction

An **authentication token** is an extremely useful alternative to a password. An authentication token is a small device that generates a new random value every time it is used. This random value becomes the basis for authentication. The small devices are typically of the size of small key chains, calculators or credit cards. Usually, an authentication token has the following features:

- Processor
- Liquid Crystal Display (LCD) for displaying outputs
- Battery
- (Optionally) a small keypad for entering information
- (Optionally) a real-time clock

Each authentication token (i.e. each device) is pre-programmed with a unique number, called a **random seed**, or just **seed**. The seed forms the basis for ensuring the uniqueness of the output produced by the token. How does this work? Let us understand this step by step.

Step 1: Creation of a Token

Whenever an authentication token is created, the corresponding random seed is generated for the token by the **authentication server** (a special server that is configured to work with authentication tokens). This seed is stored or pre-programmed inside the token, as well as its entry is made against that user's

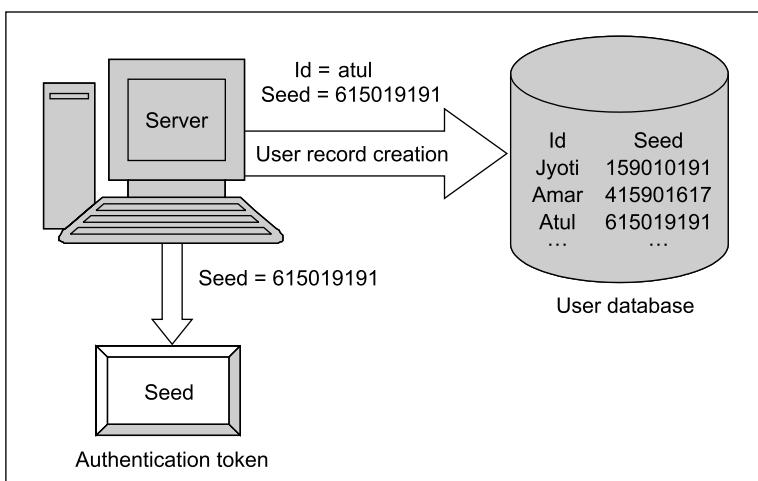


Fig. 7.16 Random seed storage in the database and the authentication token

record in the user database. Conceptually, think about this seed as the user's password (although this is technically completely different from a password). Also, the user does not know about the value of the seed, unlike a password. This is because the seed is used automatically by the authentication token, as we shall discuss soon. This is shown in Fig. 7.16.

Step 2: Use of Token

An authentication token automatically generates pseudorandom numbers, called **one-time passwords** or **one-time passcodes**. One-time passwords are generated randomly by an authentication token, based on the seed value that they are pre-programmed with. They are *one-time* because they are generated, used once, and discarded for ever. When a user wants to be authenticated, the user will get a screen to enter the user id and the latest one-time password. For this, the user will enter the user id, and the one-time password obtained from the authentication token. The user id and password travel to the server as a part of the login request. The server obtains the seed corresponding to the user id from the user database, using a *seed-retrieval program*. It then calls another program called *password-validation program*, to which the server gives the seed and the one-time password. This program knows how to establish the relationship between the seed and the one-time password. How this is done is beyond the scope of the current text, but to explain it in simple terms, the program uses synchronization techniques, to generate the same one-time password as was done by the authentication token. However, the main point to be noted here is that the authentication server can use this program to determine if a particular seed value relates to a particular one-time password or not. This is shown in Fig. 7.17.

A question at this stage could be: What would happen if a user loses an authentication token? Can another user simply grab it and use it? To deal with such situations, the authentication token is generally protected by a password or a 4-digit pin. Only when this PIN is entered can the one-time password be generated. This is also the basis for what is called **multi-factor authentication**. What are these factors? There are three most common factors:

- Something you know, e.g. a password or PIN
- Something you have, e.g. a credit card or an identity card
- Something you are, e.g. your voice or fingerprint

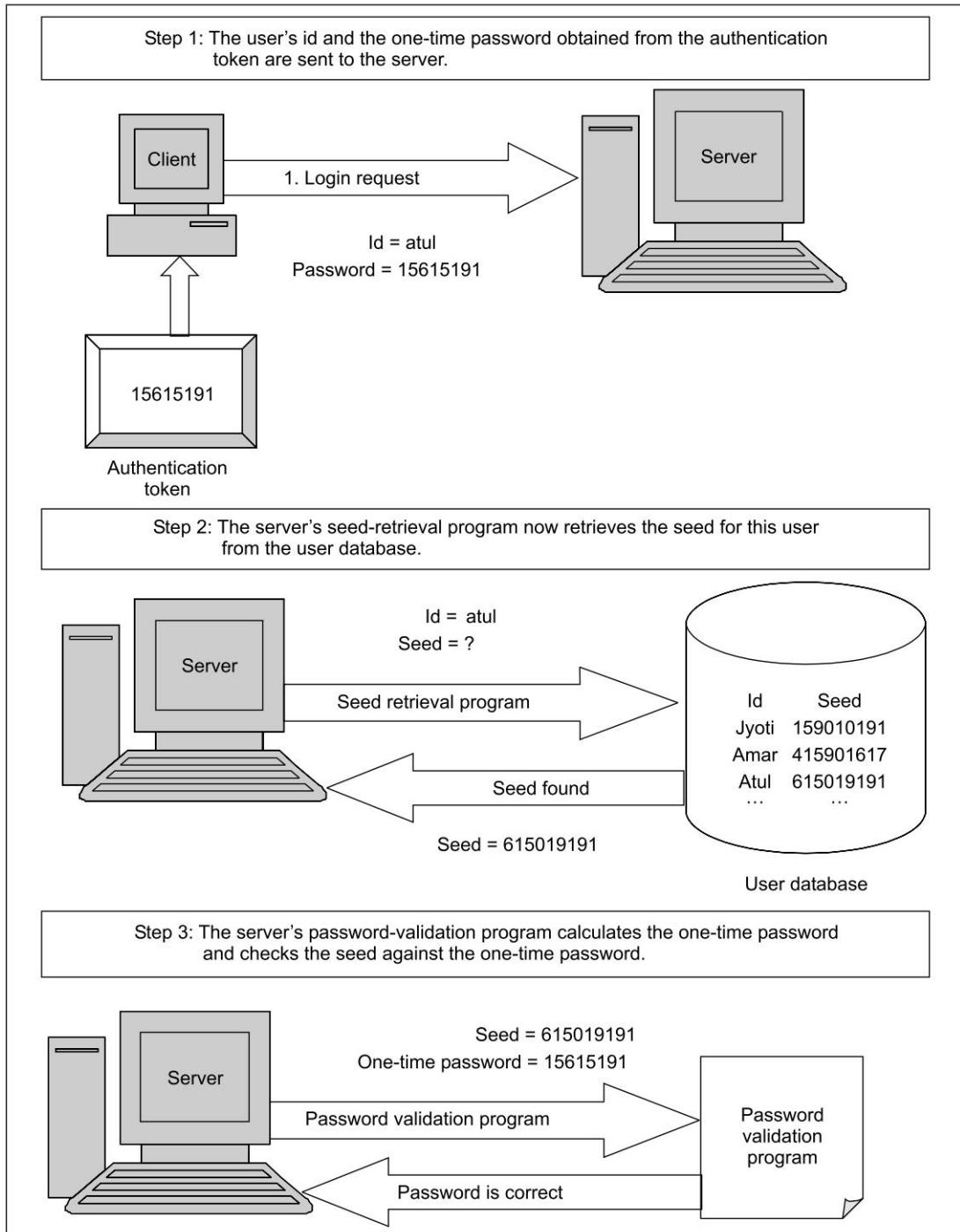
Based on these principles, we can see that a password is a **1-factor authentication**, because it is only something that *you know*. In contrast, authentication tokens are examples of **2-factor authentication**, because here you must *have something* (the authentication token itself) and you must also *know something* (the PIN used to protect it). Someone *only knowing* the PIN or *only having* the token cannot use it—both the factors are required for the authentication token to be used. We shall discuss the third type (*something you are*) later.

Step 3: Server Returns an Appropriate Message back to the User

Finally, the server sends an appropriate message back to the user, depending on whether the previous operations yielded success or failure. This is shown in Fig. 7.18.

7.4.2 Authentication Token Types

There are two main types of authentication tokens, as shown in Fig. 7.19.

**Fig. 7.17** Server validates the one-time password

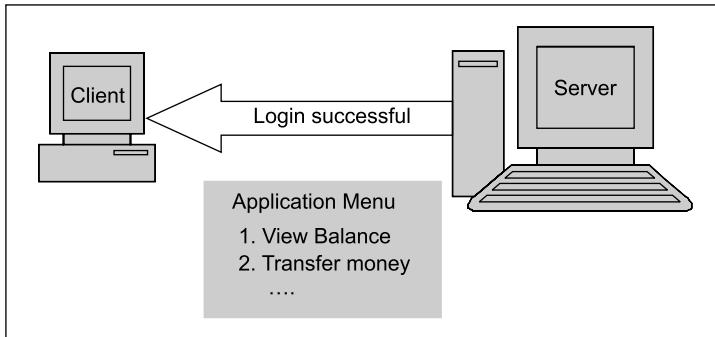


Fig. 7.18 Server sends an appropriate message back to the user

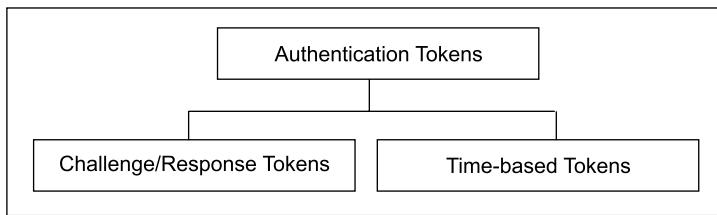


Fig. 7.19 Types of authentication tokens

Let us discuss these two types of authentication tokens now.

1. Challenge/Response Tokens

The idea used in the **challenge/response tokens** is actually a combination of the techniques we have discussed so far. We know that the seed pre-programmed inside an authentication token is secret, and unique. This fact is the basis for the challenge/response tokens. In fact, as we shall see, the seed becomes the encryption key in this technique.

Step 1: User Sends a Login Request In this technique, the user sends the login request only with his/her user id (and not the one-time password). This is shown in Fig. 7.20.

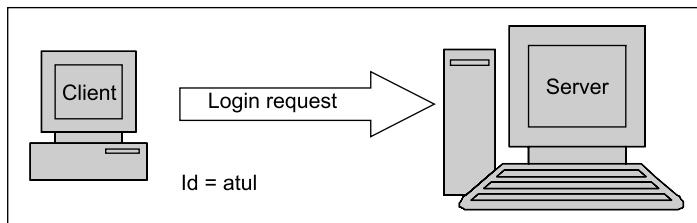


Fig. 7.20 Login request now contains only the user id

Step 2: Server Creates a Random Challenge Now, the server employs the technique that we have explained earlier. However, since there are subtle differences in the way this is implemented here, we shall discuss it in detail. When the server receives the user's login request containing the user id alone, it first checks to see if the user id is a valid one (note that only the user id is checked). If it is not, it sends an appropriate error message back to the user. If the user id is valid, the server now creates a

random challenge (a random number, generated using a pseudo-random number generation technique), and sends it back to the user. The random challenge can travel as plain text from the server to the user's computer, as shown in Fig. 7.21.

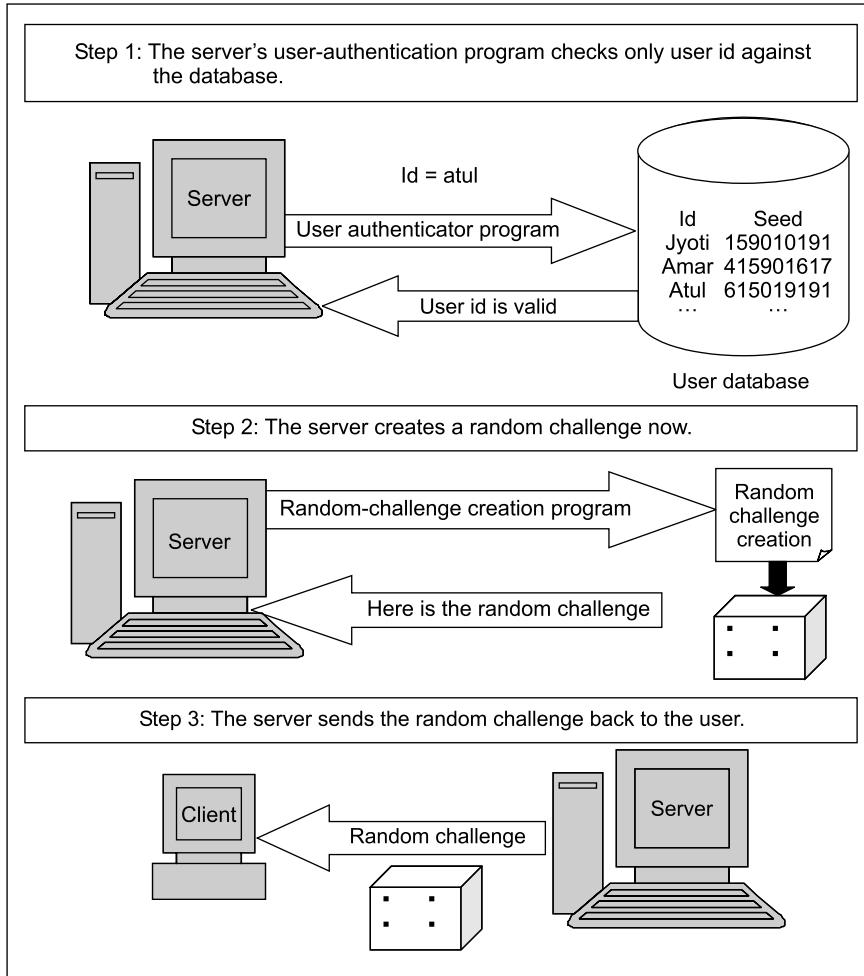


Fig. 7.21 Random-challenge creation and transmission to the user

Step 3: User Signs the Random Challenge with the Message Digest of the Password The user gets a screen, which displays the user id, the random challenge received from the server, and a data entry field, with the label *Password*. Let us assume that the random challenge sent by the user was 8102811291012. This is shown in Fig. 7.22.

At this stage, the user reads the random challenge displayed on the screen. It first opens the token using her PIN and then keys in the random challenge received from the server inside the token. In order to do this, the token has a small keypad. The token accepts the random challenge, and encrypts it with the seed value, which is known only to itself. The result is the random challenge encrypted with the seed. This result is displayed on the display (LCD) of the token. The user reads this value and types it in the

Login Screen	
User Id	Atul
Random Challenge	8102811291012
Password	<input type="password"/>

Fig. 7.22 Challenge/Response Tokens login screen

Password field on the screen. This request is then sent to the server as the *login request*. The whole process is shown in Fig. 7.23.

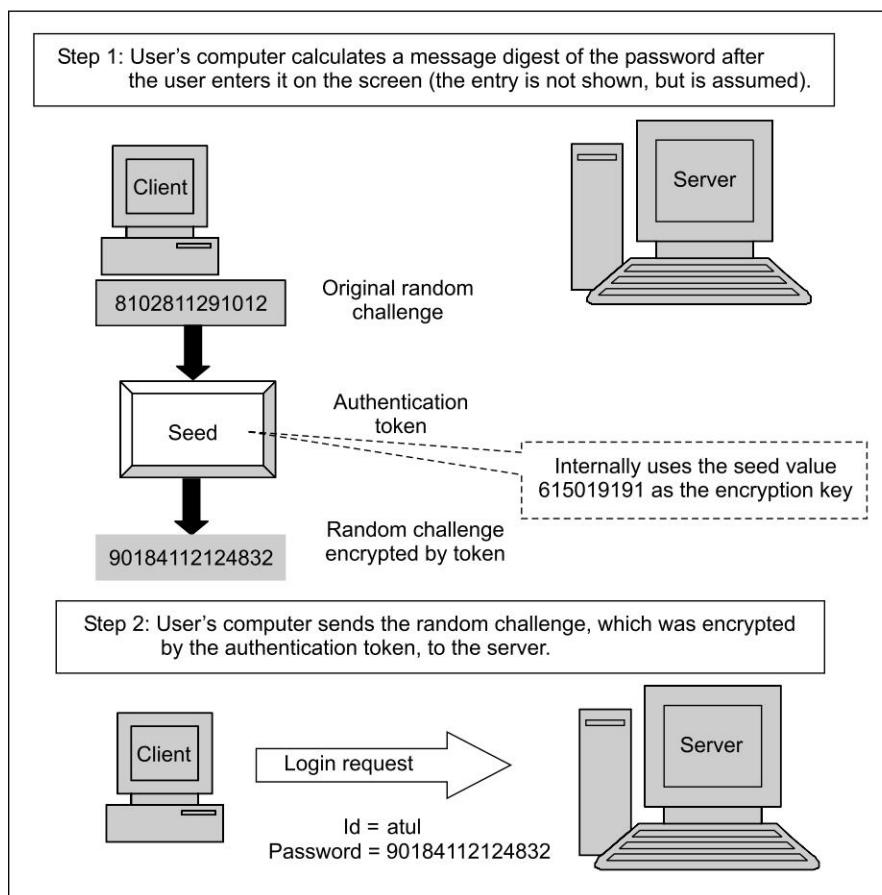


Fig. 7.23 User sends the user id and encrypted random challenge to the server

Step 4: Server Verifies the Encrypted Random Challenge Received from the User The server receives the random challenge, which was encrypted with the seed by the user's authentication token. In order to verify that the random challenge was indeed encrypted by the correct seed, the server must perform an identical operation. This can be done in two ways:

- The server can decrypt the encrypted random challenge received from the user with the seed value for the user. As we know, the seed for the user is available to the server via the user database. If this decryption matches the original random challenge available on the server, the server can be assured that the random challenge was indeed encrypted by the correct seed of the user's authentication token.
- Alternatively, the server can simply encrypt its own version of the random challenge (i.e. the one which was sent earlier to the user) with the seed for the user. If this encryption produces an encrypted random challenge, which matches with the encrypted random challenge received from the user, the server can be assured that the random challenge was indeed signed by the correct seed.

Regardless of the mode of operation, the server can thus determine if the user is really what he/she claims to be. As before, we illustrate the second approach in Fig. 7.24. We could have as well used the first approach, instead.

Step 5: Server Returns an Appropriate Message Back to the User Finally, the server sends an appropriate message back to the user, depending on whether the previous operations yielded success or failure. This is shown in Fig. 7.25.

The only problem with this scheme is that it can result into the generation of long strings. For instance, if we use 128-bit seeds and 128-bit keys, the encrypted seed will also consist of 128 bits (i.e. 16 characters). That is, the user will have to read 16 characters from the LCD of the authentication token and enter that on the screen as password. This can be quite cumbersome to most users. Therefore, an alternative approach is to use the message-digest technique. Here, the authentication token combines the seed with the random challenge, produces a message digest, truncates it to a pre-determined number of bits, transforms it into user-readable format and displays it on the LCD. The user reads this text of a smaller size, and enters it as the password. The server also performs similar processing.

2. Time-based Tokens

In spite of the improvement to the Challenge/Response mechanism discussed earlier (using message digest instead of encryption), there is still a practical problem. Note that the user has to make three entries: firstly, the user has to enter the PIN to access the token; secondly, the user has to read the random challenge from the screen and key in the random number challenge into the token, and thirdly, the user has to read the encrypted random challenge from the LCD of the token and enter it into the *Password* field. Users generally make quite a few mistakes in all these processes, resulting into a lot of flow of wasteful information between the user's computer, the server and the authentication token.

In **time-based tokens**, these disadvantages are addressed. Here, the server need not send any random challenge to the user. The token need not have a keypad for entry. The theory behind this is the usage of time as the variable input to the authentication process, in place of the random challenge. The process works as follows.

Step 1: Password Generation and Login Request The token is pre-programmed with a seed, as usual. The copy of the seed is also available to the authentication server. As we know, a challenge/

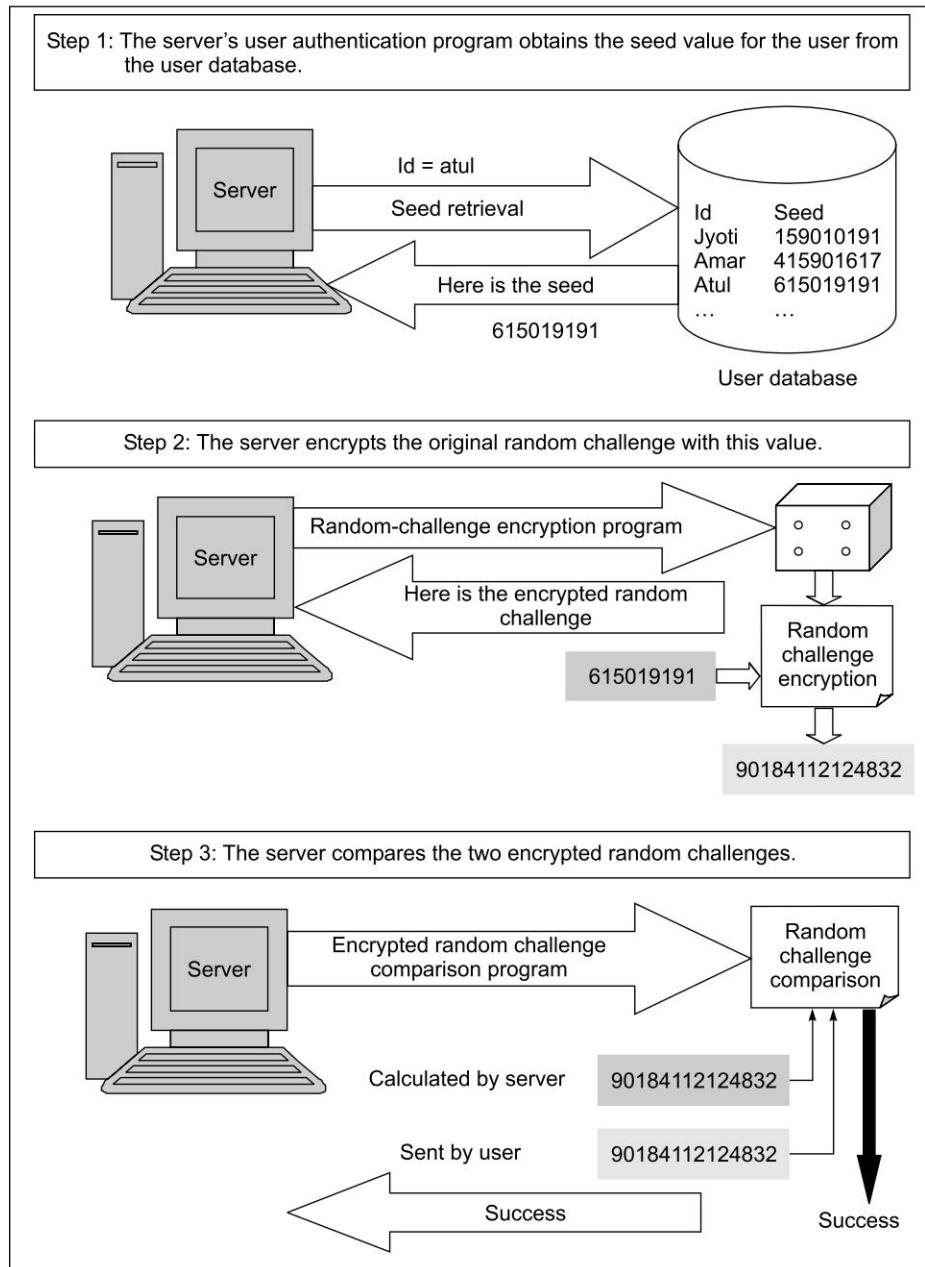


Fig. 7.24 Server compares the two encrypted random challenges

response token performs an operation such as encryption or message-digest creation only based on the user's inputs. However, in the case of time-based tokens, this is handled differently. These tokens do not require any user inputs. Instead, these tokens automatically generate a password every 60 seconds, and display the latest password on the LCD output for the user to read and use it.

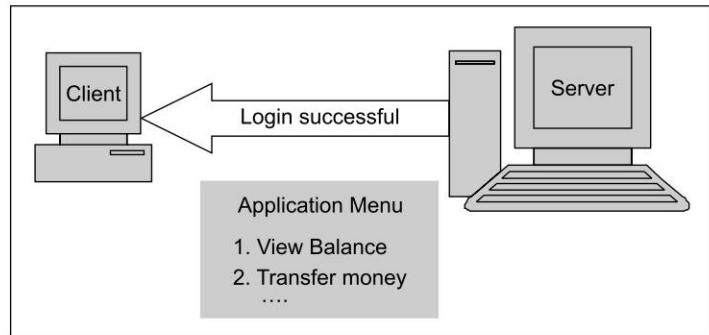


Fig. 7.25 Server sends an appropriate message back to the user

For generating the password, the time-based tokens use two parameters: the seed and the current system time. It performs some cryptographic function on these two input parameters to produce the password automatically. The token then displays it onto the LCD. Whenever a user wishes to log on, he/she takes a look at the LCD display, reads the password from there, and uses his/her id and this password for login. This is shown in Fig. 7.26.

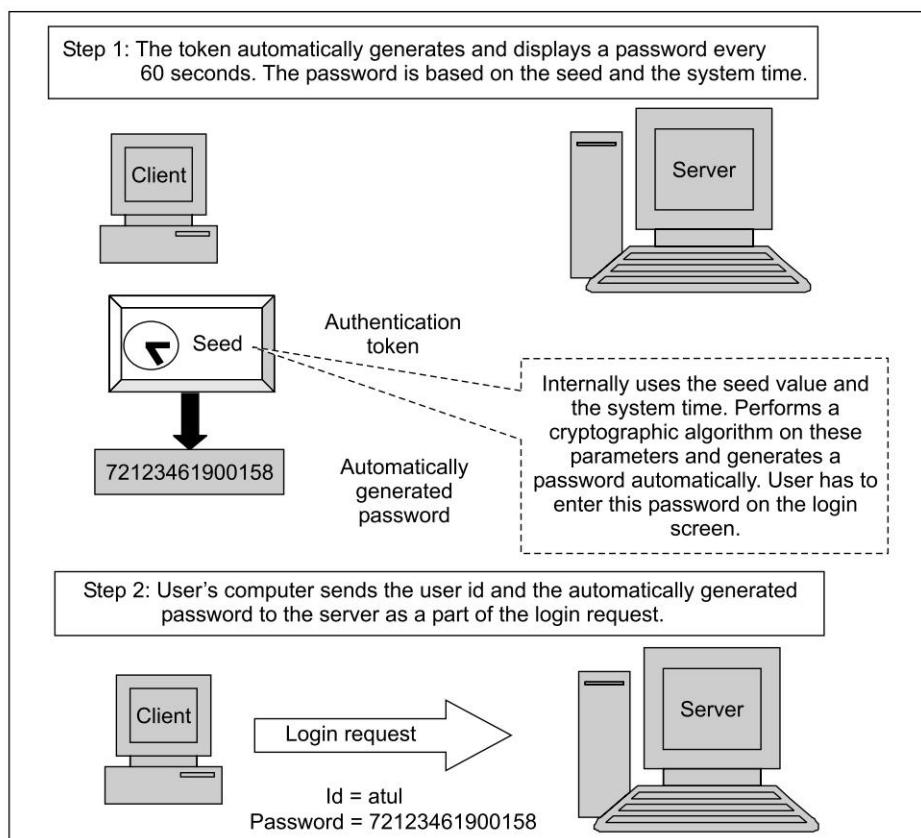


Fig. 7.26 User sends login request to the server

Step 2: Server-side Verification The server receives the password. It also performs an independent cryptographic function on the user's seed value and the current system time to generate its version of the password. If the two values match, it considers the user as a valid one. This is shown in Fig. 7.27.

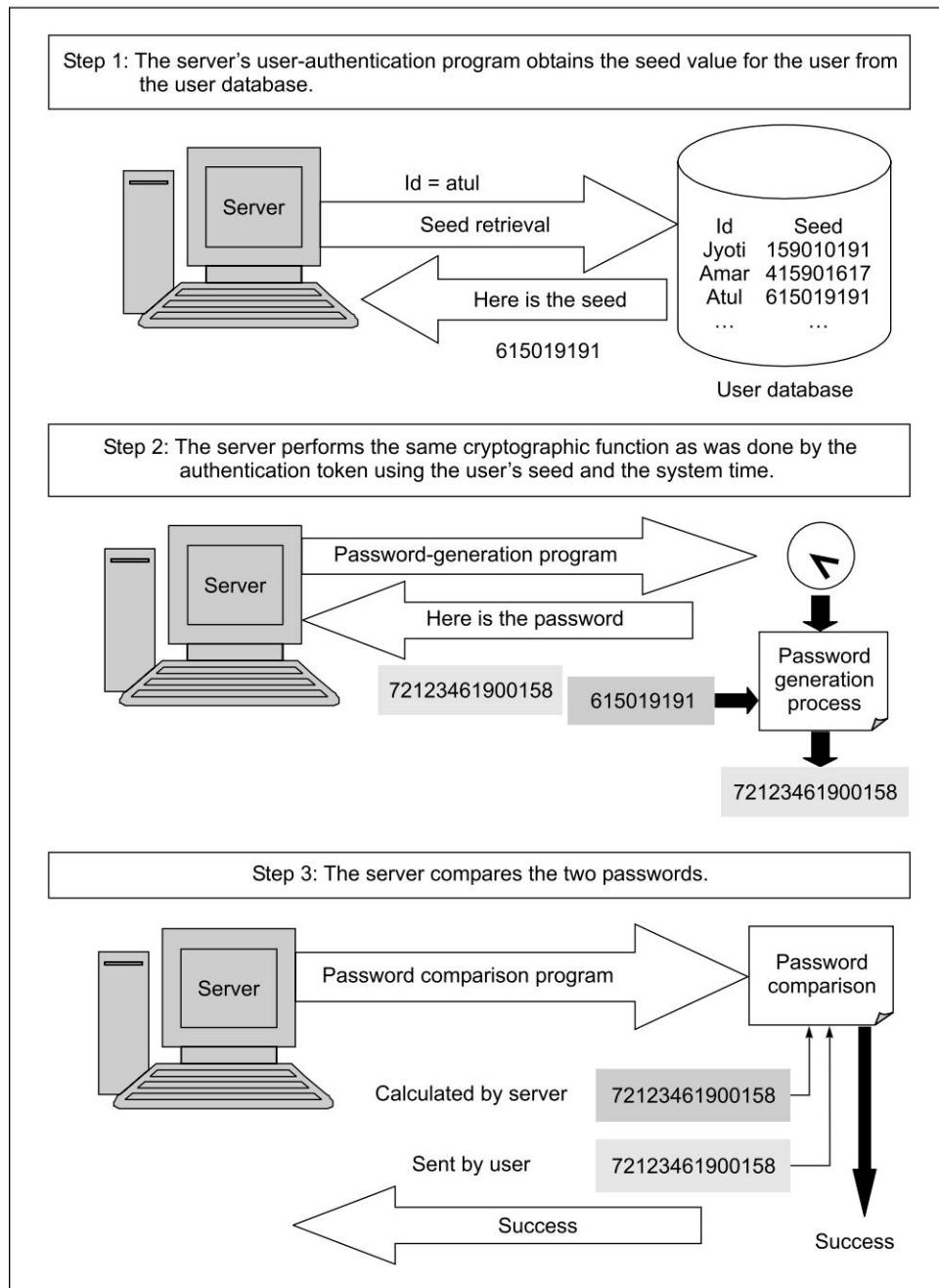


Fig. 7.27 Server verifies the password sent by the user

Step 3: Server Returns an Appropriate Message Back to the User Finally, the server sends an appropriate message back to the user, depending on whether the previous operations yielded success or failure. This is shown in Fig. 7.28.

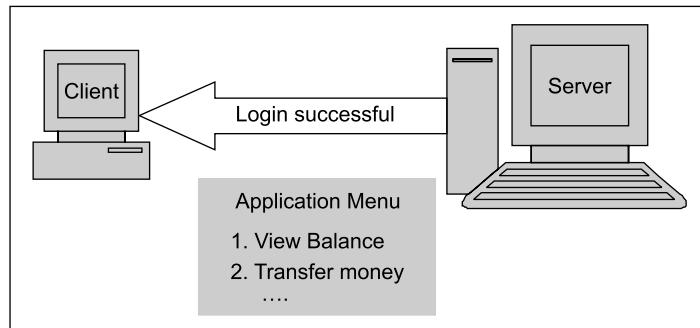


Fig. 7.28 Server sends an appropriate message back to the user

Due to their automated nature (as compared to challenge/response tokens), time-based tokens are used more often in real life. However, if we think about their mechanism minutely, we will realize that there is a concern here as well. What happens if the time *window* of 60 seconds is crossed from the time the user's login request reaches the server but before the authentication is over? That is, simplistically thinking, suppose that the time at the user's end is 17:47:57 hours at the time of sending the login request. When the request reaches the server and the authentication begins, the server's time is suppose 17:48:01 hours. Now, the server would consider the user as invalid, because its 60-second window does not match with that of the user. To resolve such problems, the approach of retrials is used. When a time window expires, the user's computer sends a new login request by advancing its time by 1 minute. If this also fails, the user's computer sends another login request with time advanced by 2 minutes, and so on.

Another question might be, if time-based tokens do not have a keypad, how does a user enter her PIN? To deal with this problem, the user actually enters the PIN in the login screen itself. The software is intelligent enough to use it for accessing the token. Moreover, these days, for critical applications, time-based tokens with keypads are also emerging.

■ 7.5 CERTIFICATE-BASED AUTHENTICATION ■

7.5.1 Introduction

Another emerging authentication mechanism is that of **certificate-based authentication**. This is based on the digital certificate of a user. FIPS-196 is a standard that specifies the operation of this mechanism. As we know, in PKI, the server and (optionally) the client are required to possess digital certificate in order to perform digital transactions. The digital certificates can then be reused for user authentication as well. In fact, if we use SSL, the server *must* have a digital certificate, whereas the clients (users) *may* have digital certificates. This is because the client authentication is optional in SSL, but not the server authentication.

Certificate-based authentication is a stronger authentication mechanism as compared to a password-based authentication mechanism, because here the user is expected to *have* something (certificate) and not *know* something (password). At the time of login, the user is requested to send his/her certificate to the server, over the network as a part of the login request. A copy of the certificate exists on the server, which can be used to verify that the certificate is indeed a valid one. However, this is not all that simple. How do we deal with the following situations?

- Suppose user *A* has gone for a cup of tea. User *B* uses this opportunity to login from *A*'s computer, using *A*'s certificate and performs some high-value transaction, posing as *A*.
- Without *A*'s knowledge, *B* copies *A*'s certificate (which is nothing but a computer file on the disk) on a floppy disk, copies it back on to his/her own computer, and starts logging in as *A*, as and when he/she likes.

As we can see, the main concern here is the misuse of someone else's certificate. How do we prevent that? Well, to tackle such issues, in practice, certificate-based authentication is also made a 2-factor process (*have something* and *know something*), as we shall see.

7.5.2 The Working of Certificate-based Authentication

Step 1: Creation, Storage and Distribution of Digital Certificates

The first step in certificate-based authentication is actually a pre-requisite. Here, the digital certificates are created by the CA for each user and the certificates are sent to the respective users. Moreover, a copy of the certificate is stored by the server in its database (usually in a binary format), in order to verify the certificate during the user's certificate-based authentication. This process is shown in Fig. 7.29.

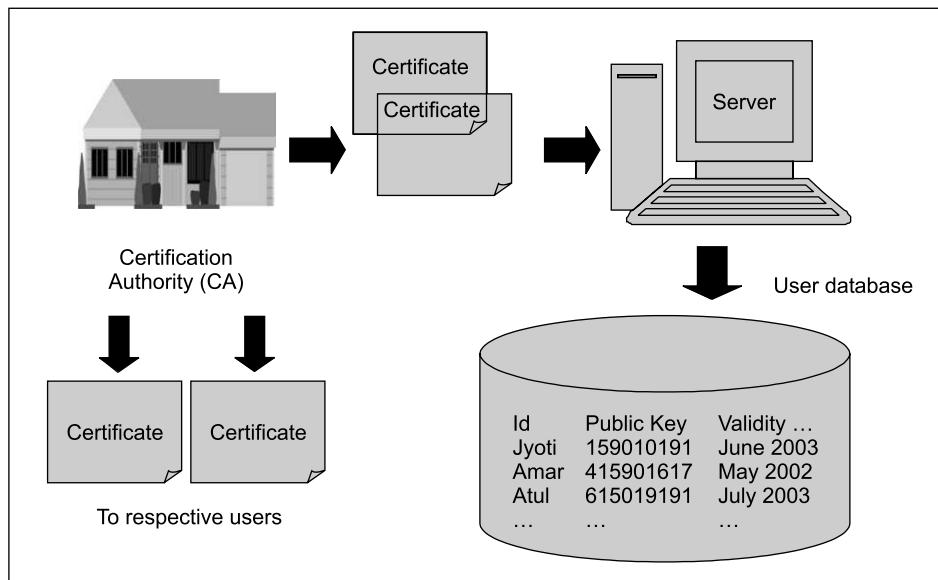


Fig. 7.29 Digital certificate creation, distribution and storage

Step 2: Login Request

During a login request, the user sends only his/her user id to the server, as shown in Fig. 7.30.

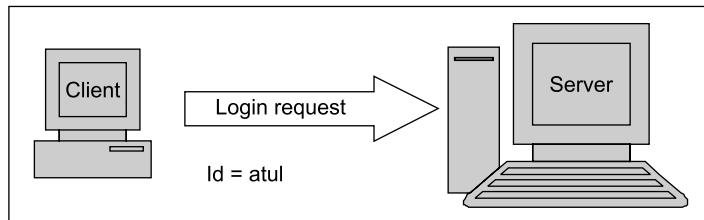


Fig. 7.30 Login request now contains only the user id

Step 3: Server Creates a Random Challenge

Now, the server employs the technique that we have explained earlier. When the server receives the user's login request containing the user id alone, it first checks to see if the user id is a valid one (note that only the user id is checked). If it is not, it sends an appropriate error message back to the user. If the user id is valid, the server now creates a random challenge (a random number, generated using a pseudo-random number generation technique), and sends it back to the user. The random challenge can travel as plain text from the server to the user's computer, as shown in Fig. 7.31. We have deliberately not shown the user database in its earlier form, to keep it simple; we now show only the user id and the public key.

Step 4: User Signs the Random Challenge

The user has to now sign the random challenge with her private key. As we know, this private key corresponds to the user's public key, with the latter being mentioned in the user's certificate. For this purpose, the user needs to access his/her private key, which is stored as a disk file on her computer. However, the private keys are not available directly to anybody. In order to protect them, passwords are used. Only a correct password can open a private-key file. Therefore, the user must enter the secret password for opening up the private key file, as shown in Fig. 7.32.

After the user enters the correct password, the user's private-key file is opened by the application. It retrieves the private key from that file, and uses it to encrypt the random challenge received from the server to create the user's digital signature. Technically, this can be a two-step process, wherein first a message digest of the random challenge is created, and the message digest is then encrypted with the user's private key. However, we shall consider it as a single, logical operation for simplicity. This process is shown in Fig. 7.33.

The server now needs to verify the user's signature. For this purpose, the server consults the user database to obtain the user's public key (since that can alone verify the user's signature). It then uses this public key to decrypt (also called de-sign) the signed random challenge received from the user. After this, it compares this decrypted random challenge with its original random challenge. This is shown in Fig. 7.34.

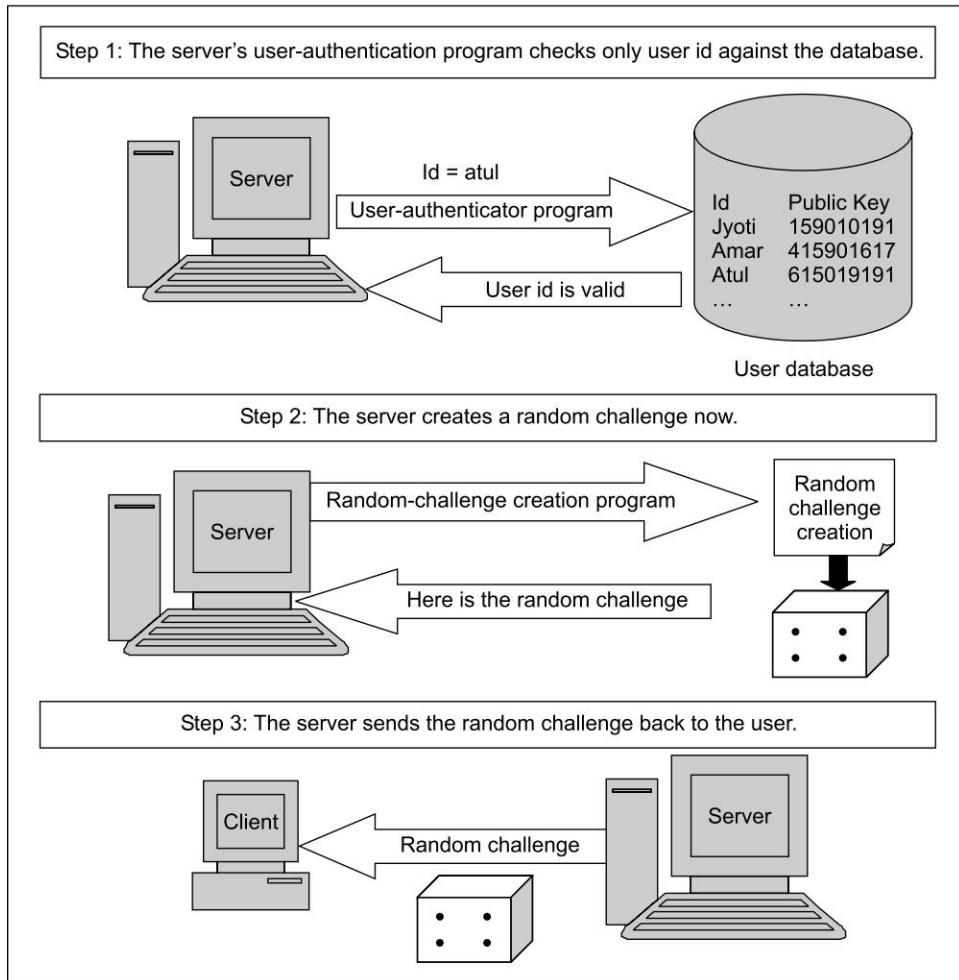


Fig. 7.31 Random-challenge creation and transmission to the user

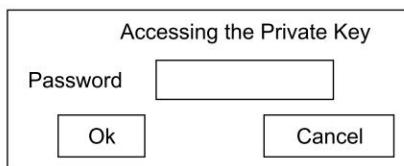


Fig. 7.32 Prompt for password to open private-key file

Step 5: Server Returns an Appropriate Message Back to the User

Finally, the server sends an appropriate message back to the user, depending on whether the previous operations yielded success or failure. This is shown in Fig. 7.35.

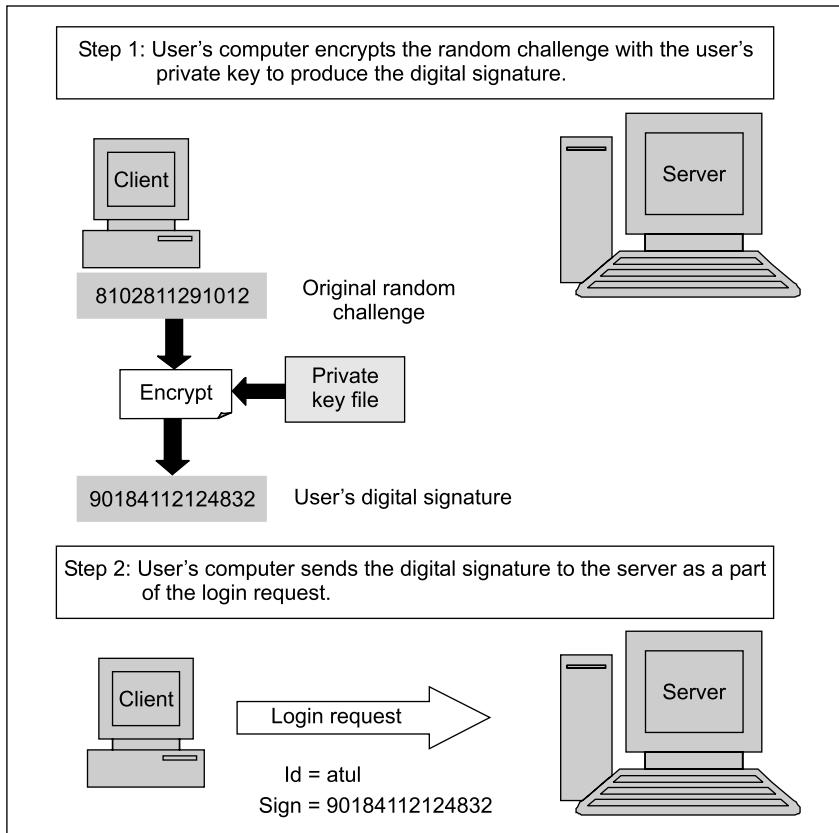


Fig. 7.33 User's computer signs the random challenge and sends it back to the server

7.5.3 Use of Smart Cards

The use of smart cards can actually be related to certificate-based authentication. This is because smart cards allow the generation of private-public key pairs *within* the card. They also support the storage of digital certificates within the card. The private key always remains inside the card in a secure, tamper-free fashion. The public key and the certificate can be exported outside. Also, smart cards are capable of performing cryptographic functions such as encryption, decryption, message digest creation and signing within the card.

Thus, during certificate-based authentication, the signing of random challenge sent by the server can be performed within the card. That is, the random challenge can be fed as input to the smart-card, which can accept it, encrypt it with the smart card holder's private key, thus producing digital signature within the card, which it outputs back to the application.

However, one thing must be noted about the usage of smart cards. They must be used judiciously to perform selective cryptographic operations. For instance, if we wish to sign a 1 MB document using a smart card, it would be quite cumbersome if we expect the smart card to first produce a message digest of the document, and then sign it. This is because just moving the 1 MB data to the smart card through the half-duplex 9,600 bits per second (bps) smart-card interface would take about 15 minutes! Clearly,

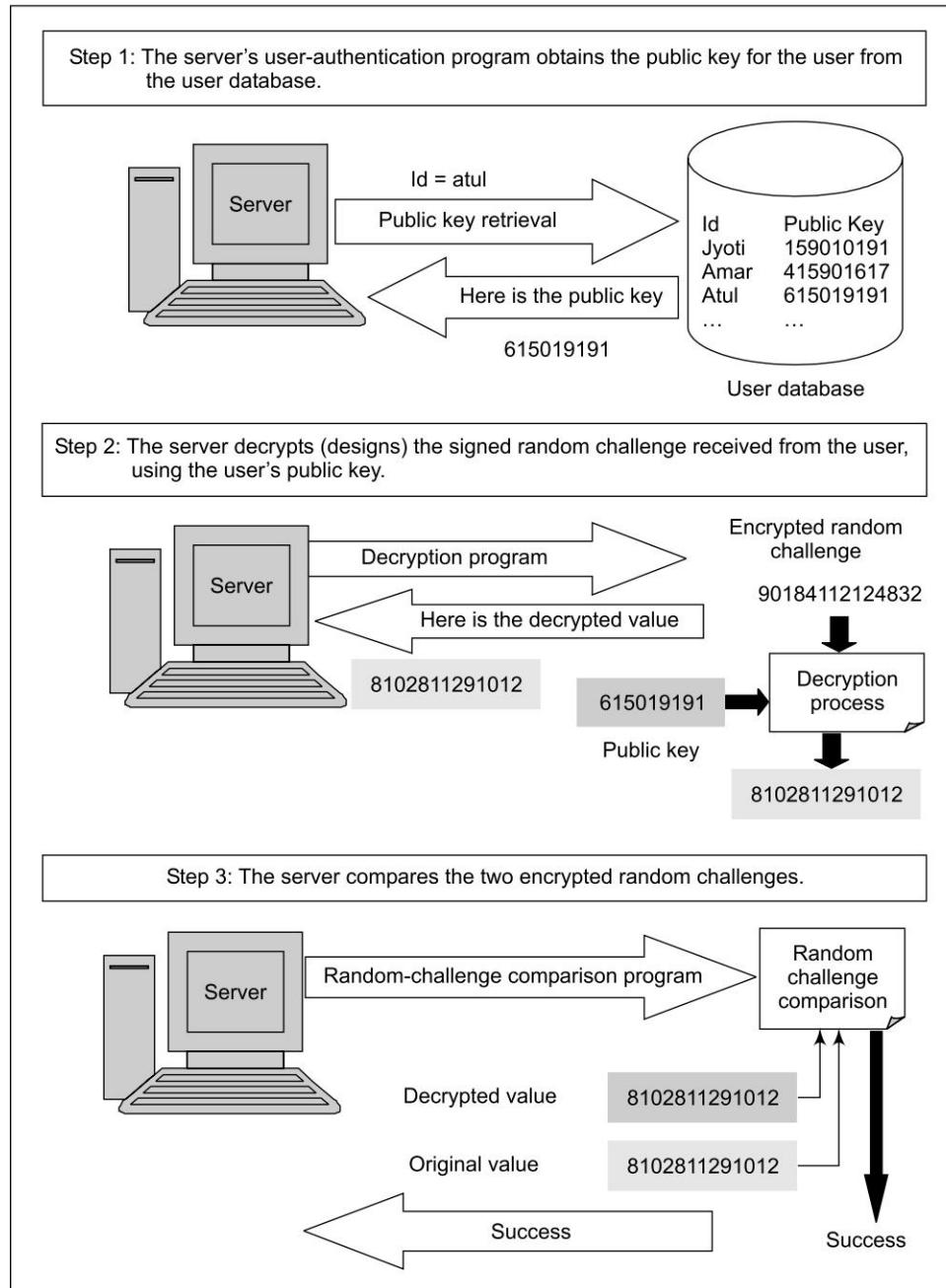


Fig. 7.34 Server compares the two random challenges

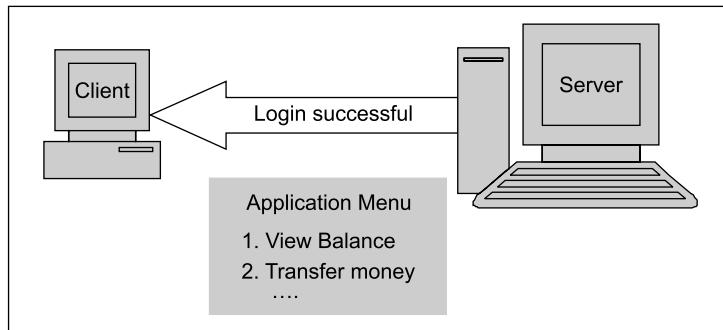


Fig. 7.35 Server sends an appropriate message back to the user

the approach should be to first generate a message digest outside the smart card (i.e. inside the computer), and feed it to the smart card just for encrypting it to produce the digital signature.

Since smart cards are portable, one can virtually walk around with one's private key and digital certificate. Traditionally, smart cards have problems associated with them. These issues and their emerging solutions are listed in Fig. 7.36.

Problem/Issue	Emerging solution
Smart-card readers are not yet a part of a desktop computer, unlike a hard-disk drive or a floppy-disk drive	The new versions of computers and mobile devices are expected to come with smart-card readers <i>out of the box</i> .
Non-availability of smart-card reader driver software	Microsoft has made the PC/SC smart card framework an integral part of the Windows 2000 operating system. Most smart-card reader manufacturers ship the PC/SC compliant reader drivers, making the process of adding a reader hardware to the computer a plug-and-play operation.
Non-availability of smart card aware cryptographic services software	Smart-card aware software such as Microsoft Crypto API (MS-CAPI) comes free with Internet Explorer.
Cost of smart cards and card readers is high	This is reducing now. Smart cards are available for about \$5, and the card readers for about \$20.

Fig. 7.36 Problems and their solutions related to smart-card technology

There is still a lack of standardization and inter-operability between smart-card vendors. This should change, as the industry matures. This would mean that the security of PKI solutions can really become very strong.

■ 7.6 BIOMETRIC AUTHENTICATION ■

7.6.1 Introduction

Biometric-authentication mechanisms are receiving a lot of public attention. A biometric device is perhaps the ultimate attempt in trying to prove who you are. A biometric device works on the basis of

some human characteristics, such as fingerprint, voice, or pattern of lines in the iris of your eye. The user database contains a sample of the user's biometric characteristics. During authentication, the user is required to provide another sample of the user's biometric characteristics. This is matched with the one in the database, and if the two samples are the same then the user is considered to be a valid one.

The important idea in biometrics is that the sample produced during every authentication process can vary slightly. This is because the physical characteristics of the user may change for a number of reasons. For instance, suppose the fingerprint of the user is captured and used for authentication every time. The sample taken for every authentication may not be the same, because the finger can be dirty, can have cuts, other marks or the finger's position on the reader can be different, and so on. Therefore, an exact match of the sample need not be required. An approximate match can be acceptable.

This is also the reason why, during the user registration process, multiple samples of the user biometric data are created. They are combined and their average stored in the user database, so that the different possibilities of the user's samples during the actual authentication can roughly map to this average sample. Using this basic philosophy, any biometric authentication system defines two configurable parameters: the **False Accept Ratio (FAR)** and the **False Reject Ratio (FRR)**. The FAR is a measurement of the chance that a user who should be rejected is actually accepted by a system as *good enough*. FRR is a measurement of the chance that a user who should be accepted as valid is actually rejected by a system as *not good enough*. Thus, FAR and FRR are exactly opposite of each other.

Perhaps the best security solution is to combine the password/PIN, a smart card and biometrics. It covers all the three key aspects related to authentication: who you are, what you have, and what you know. However, this can turn out to be an extremely complex system to build and/or use.

7.6.2 The Working of Biometrics

A typical authentication process involving biometrics firstly involves the creation of the user's sample and its storage in the user database. During the actual authentication, the user is required to provide a sample of the same nature (e.g. a retina scan or a fingerprint). This is usually sent across an encrypted session (e.g. by using SSL) to the server. On the server, the user's current sample is decrypted, and compared with the one stored in the database. If the two samples match to the expected degree on the basis of the particular values of FAR or FRR, the user is considered as authenticated successfully. Otherwise, the user is considered as invalid.

7.6.3 Biometric Techniques

Biometric techniques are generally classified into two sub-categories, namely **physiological** and **behavioral**. Let us discuss these in brief.

1. Physiological Techniques

As the name suggests, these techniques rely on the physical characteristics of human beings. Since the aim is to identify humans uniquely, these characteristics must be very prominent and distinguishable from one person to another. Several such techniques are used, as mentioned below.

(a) Face In this technique, the idea is to check and measure the distance between the various facial features such as eyes, nose, and mouth. This distance measurement is done using geometrical techniques.

(b) Voice Human voice can be uniquely identified based on the characteristics of the sound waves of a voice. Some of these characteristics are the pitch and tone.

(c) Fingerprint Medical science tells us that every human being has a unique fingerprint, at least after a specific age. The fingerprint-based authentication uses two approaches: minutiae-based and image-based. In the minutiae-based technique, a graph of the individual ridge positions is drawn. In the image-based technique, an image of the fingerprints is taken and stored in the database for subsequent comparisons. While fingerprints can change due to ageing or diseases, they have been used extensively for authentication.

(d) Iris Amazingly, each person has some unique pattern inside the iris. This technique is based on identifying a person uniquely based on this pattern. This mechanism is considered quite sound and reliable. For checking the iris pattern, usually laser beams are employed.

(e) Retina Retina scanning is not very common. The main reason behind this is its high cost. In this mechanism, the vessels carrying blood supply at the back of a human eye are examined. They provide a unique pattern, which is used to authenticate an individual.

2. Behavioral Techniques

The idea in behavioral techniques is to observe a person to ensure that he/she is not trying to claim to be someone else. In other words, here the emphasis is on checking that a person's behavior is not unusual or abnormal. Two main techniques are used here, as discussed below.

(a) Keystroke Several characteristics such as the speed of typing, strength of keystrokes, time between two keystrokes, error percentage and frequency, etc., can be measured for identifying users. However, it is not as reliable as many other authentication mechanisms.

(b) Signature This is an old technique. Cheques and many other documents are expected to be physically signed by the authorizer. This is now extended by keeping a scanned copy of a person's signature and comparing this computer-based scanned signature with the paper signature as and when the need arises.

■ 7.7 KERBEROS ■

7.7.1 Introduction

Many real-life systems use an authentication protocol called **Kerberos**. The basis for Kerberos is another protocol, called **Needham-Shroeder**. Designed at MIT to let workstations allow network resources in a secure manner, the name Kerberos signifies a multi-headed dog in Greek mythology (apparently used to keep outsiders away). Version 4 of Kerberos is found in most practical implementations. However, Version 5 is also in use now.

7.7.2 The Working of Kerberos

There are four parties involved in the Kerberos protocol:

Alice The client workstation.

Authentication Server (AS) Verifies (authenticates) the user during login.

Ticket Granting Server (TGS) Issues **tickets** to certify *proof of identity*.

Bob The server offering services such as network printing, file sharing or an application program.

The job of AS is to authenticate every user at the login time. AS shares a unique secret password with every user. The job of TGS is to certify to the servers in the network that a user is really who he/she claims to be. For proving this, the mechanism of tickets (which allow entry into a server, just as a ticket allows parking a car or entering a music concert) is used.

There are three primary steps in the Kerberos protocol. We shall study them one by one now.

Step 1: Login

To start with, Alice, the user, sits down at an arbitrary public workstation and enters her name. The workstation sends her name in plain text to the AS, as shown in Fig. 7.37.

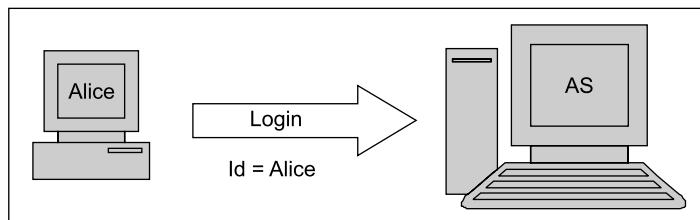


Fig. 7.37 Alice sends a login request to AS

In response, the AS performs several actions. It first creates a package of the user name (Alice) and a randomly generated session key (KS). It encrypts this package with the symmetric key that the AS shares with the Ticket Granting Server (TGS). The output of this step is called the **Ticket Granting Ticket (TGT)**. Note that the TGT can be opened only by the TGS, since only it possesses the corresponding symmetric key for decryption. The AS then combines the TGT with the session key (KS), and encrypts the two together using a symmetric key derived from the password of Alice (KA). Note that the final output can, therefore, be opened only by Alice.

After this message is received, Alice's workstation asks her for the password. When Alice enters it, the workstation generates the symmetric key (KA) derived from the password (in the same manner as AS would have done earlier) and uses that key to extract the session key (KS) and the Ticket Granting Ticket (TGT). The workstation destroys the password of Alice from its memory immediately to prevent an attacker from stealing it. Note that Alice cannot open the TGT, as it is encrypted with the key of the TGS.

Step 2: Obtaining a Service Granting Ticket (SGT)

Now, let us assume that after a successful login, Alice wants to make use of Bob—the email server—for some email communication. For this, Alice would inform her workstation that she needs to contact Bob. Therefore, Alice needs a ticket to communicate with Bob. At this juncture, Alice's workstation creates a message intended for the Ticket Granting Server (TGS), which contains the following items:

- The TGT as in step 1

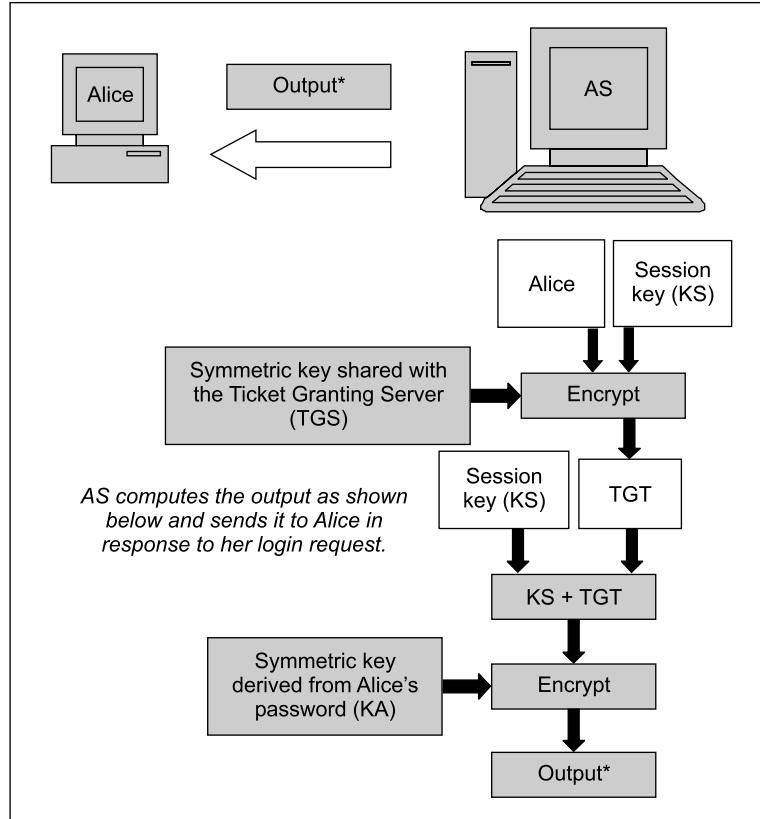


Fig. 7.38 AS sends back encrypted session key and TGT to Alice

- The id of the server (Bob) whose services Alice is interested in
- The current time stamp, encrypted with the same session key (KS)

This is shown in Fig. 7.39.

As we know, the TGT is encrypted with the secret key of the Ticket Granting Server (TGS). Therefore, only the TGS can open it. This also serves as a proof to the TGS that the message indeed came from Alice. Why? This is because, if you remember, the TGT was created by the AS (remember that only the AS and the TGS know the secret key of TGS). Furthermore, the TGT and the *KS* were encrypted together by the AS with the secret key derived from the password of Alice. Therefore, only Alice could have opened that package, and retrieved the TGT.

Once the TGS is satisfied of the credentials of Alice, the TGS creates a session key KAB, for Alice to have secure communication with Bob. TGS sends it twice to Alice: once combined with Bob's id (Bob) and encrypted with the session key (KS), and a second time, combined with Alice's id (Alice) and encrypted with Bob's secret key (KB). This is shown in Fig. 7.40.

Note that an attacker, Tom, can try and obtain the first message in this step sent by Alice, and attempt a replay attack. However, this would fail as the message from Alice contains the encrypted time stamp. Tom cannot replace the time stamp, because he does not have the session key (KS). Even if Tom at-

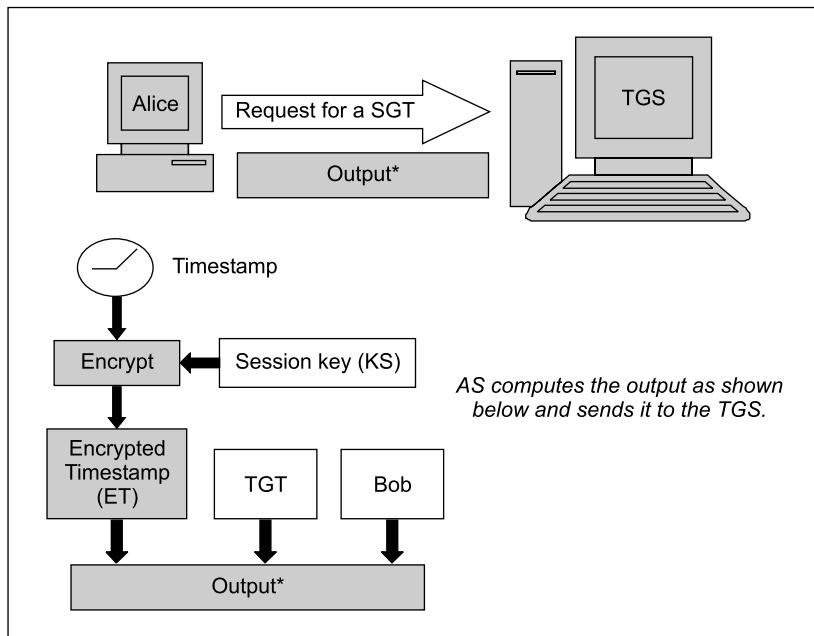


Fig. 7.39 Alice sends a request for an SGT to the TGS

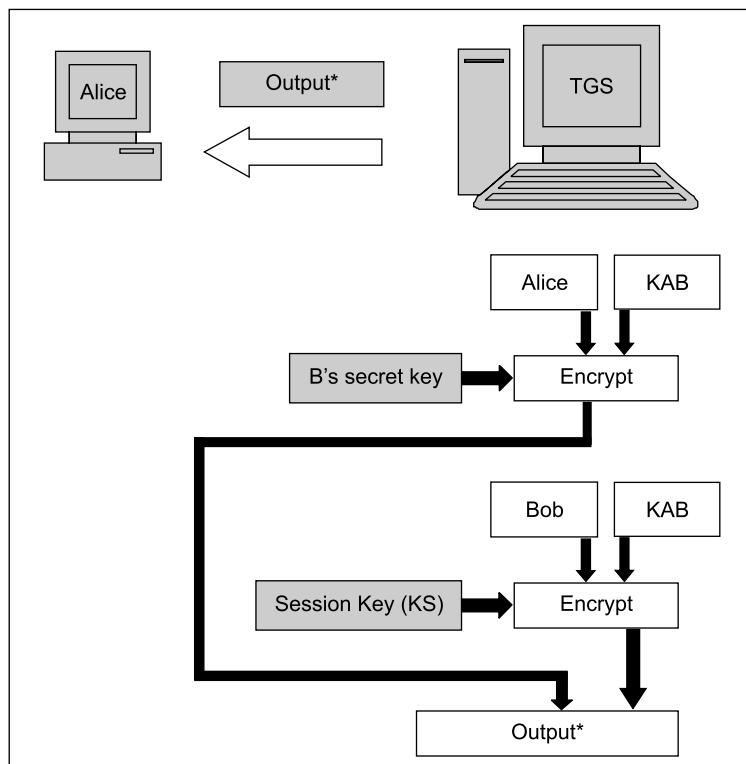


Fig. 7.40 TGS sends response back to Alice

tempts a reply attack really quickly, all that he will get back is the above message from TGS, which Tom cannot open, as he does not have access to either Bob's secret key or the session key (K_S).

Step 3: User Contacts Bob for Accessing the Server

Alice can now send KAB to Bob in order to enter into a session with him. Since this exchange is also desired to be secure, Alice can simply forward KAB encrypted with Bob's secret key (which she had received from the TGS in the previous step) to Bob. This will ensure that only Bob can access KAB . Furthermore, to guard against replay attacks, Alice also sends the time stamp, encrypted with KAB to Bob. This is shown in Fig. 7.41.

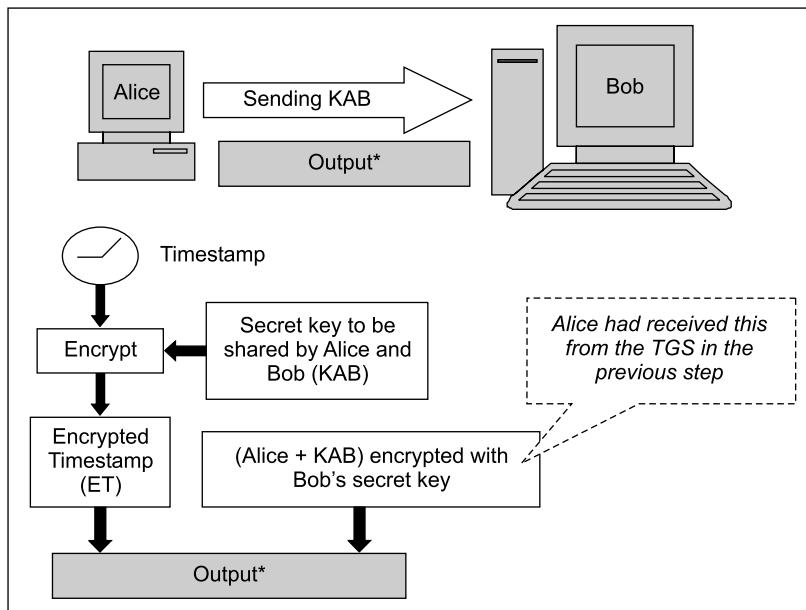


Fig. 7.41 Alice sends KAB securely to Bob

Since only Bob has his secret key, he uses it to first obtain the information $(Alice + KAB)$. From this, it gets the key KAB , which he uses to decrypt the encrypted time stamp value.

Now how would Alice know if Bob received KAB correctly or not? In order to satisfy this query, Bob now adds 1 to the time stamp sent by Alice, encrypts the result with KAB and sends it back to Alice. This is shown in Fig. 7.42. Since only Alice and Bob know KAB , Alice can open this packet, and verify that the timestamp incremented by Bob was indeed the one sent by her to Bob in the first place.

Now, Alice and Bob can communicate securely with each other. They would use the shared secret key KAB to encrypt messages before sending, and also to decrypt the encrypted messages received from each other.

An interesting point here is that if Alice now wants to communicate with another server, say Carol, she simply needs to obtain another shared key from the TGS, only now specifying Carol instead of Bob, in her message. The TGS will do the needful, as explained earlier. The outcome is that Alice can now access all the resources of the network in a similar manner, each time obtaining a unique ticket (secret

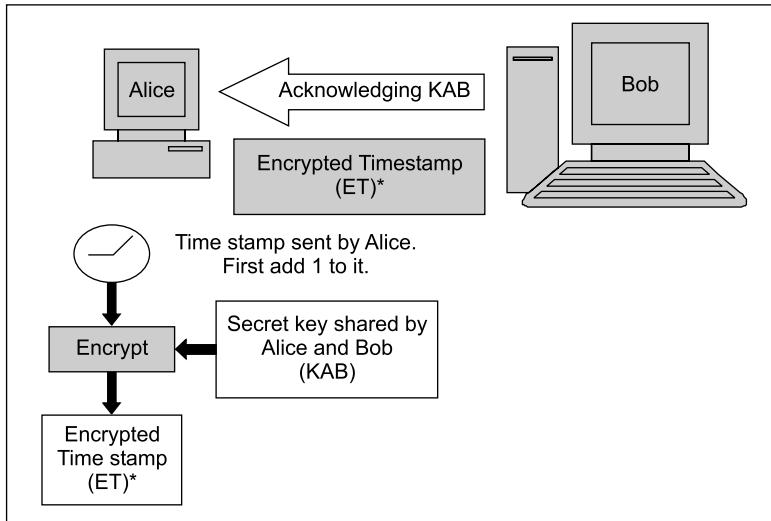


Fig. 7.42 Bob acknowledges the receipt of KAB

key) from the TGS to communicate with a different resource. Of course, if Alice wants to continue communicating with Bob alone, she need not obtain a new ticket every time. Only for the first time that she wants to communicate with a server is when she needs to contact TGS and obtain a ticket. Also, Alice's password never leaves her workstation, adding to the security.

Since Alice needs to authenticate or sign on only once, this mechanism is called **Single Sign On (SSO)**. Alice need not prove her identity to every resource in the network individually. She needs to authenticate herself only to the central AS only once. That is good enough for all the other servers/network resources to be convinced of Alice's identity. SSO is a very important concept for corporate networks, because they grow over a period of time, with multiple authentication mechanisms and diverse implementations. These can be segregated into a single, uniform authentication mechanism using SSO. In fact, Microsoft's **passport** technology on the Internet is also based on this philosophy. Microsoft Windows NT also uses the Kerberos mechanism heavily. This is also why once you log on to a Windows NT workstation, you can access your emails and other secret resources without requiring explicit log-ons, as long as the correct mappings are done by the system administrator.

Clearly, not every server in the world would trust a single AS and TGS. Therefore, the designers of Kerberos provide a support for multiple **realms**, each having its own AS and TGS.

7.7.3 Keberos Version 5

Version 5 of Kerberos overcomes some of the shortcomings of Version 4. Version 4 demands the use of DES. Version 5 allows flexibility in terms of allowing the choice of other algorithms. Version 4 depends on IP addresses as identifiers. However, Version 5 allows the use of other types as well (for this, it tags network addresses with type and length). Following are the key differences between Kerberos Versions 4 and 5.

- The key salt algorithm has been changed to use the entire principal name.
 - This means that the same password will not result in the same encryption key in different realms or with two different principals in the same realm.
- The network protocol has been completely redone and now uses ASN.1 encoding everywhere.
- There is now support for **forwardable**, **renewable**, and **postdatable** tickets.

Forwardable The user can use this ticket to request a new ticket, but with a different IP address. Thus, a user can use his/her current credentials to get credentials valid on another machine.

Renewable A renewable ticket can be renewed by asking the KDC for a new ticket with an extended lifetime. However, the ticket itself has to be valid (in other words, we cannot renew a ticket that has expired; we have to renew it before it expires). A renewable ticket can be renewed up until the maximum renewable ticket lifetime.

Postdatable These are tickets which are initially invalid, and have a starting time some time in the future. To use a postdatable ticket, the user must send it back to the KDC to have it validated during the ticket's valid lifetime.

- Kerberos tickets can now contain multiple IP addresses and addresses for different types of networking protocols.
- A generic crypto interface module is now used, so other encryption algorithms beside DES can be used.
- There is now support for replay caches, so authenticators are not vulnerable to replay.
- There is support for transitive cross-realm authentication.

■ 7.8 KEY DISTRIBUTION CENTER (KDC) ■

Key Distribution Center (KDC) is a central *authority* dealing with keys for individual computers (nodes) in a computer network. It is similar to the concept of the Authentication Server (AS) and Ticket Granting Server (TGS) in Kerberos. The basic idea is that every node shares a unique secret key with the *KDC*. Whenever user *A* wants to communicate securely with user *B*, the following happens:

1. The background is that *A* has a shared secret key *KA* with KDC. Similarly, *B* is assumed to share a secret key *KB* with the KDC.
2. *A* sends a request to KDC encrypted with *KA*, which includes
 - (a) Identities of *A* and *B*
 - (b) A random number *R*, called a **nonce**
3. KDC responds with a message encrypted with *KA*, containing
 - (a) One-time symmetric key *KS*
 - (b) Original request that was sent by *A*, for verification
 - (c) Plus, *KS* encrypted with *KB* and ID of *A* encrypted with *KB*.
4. *A* and *B* can now communicate by using *KS* for encryption.

This is depicted in Fig. 7.43.

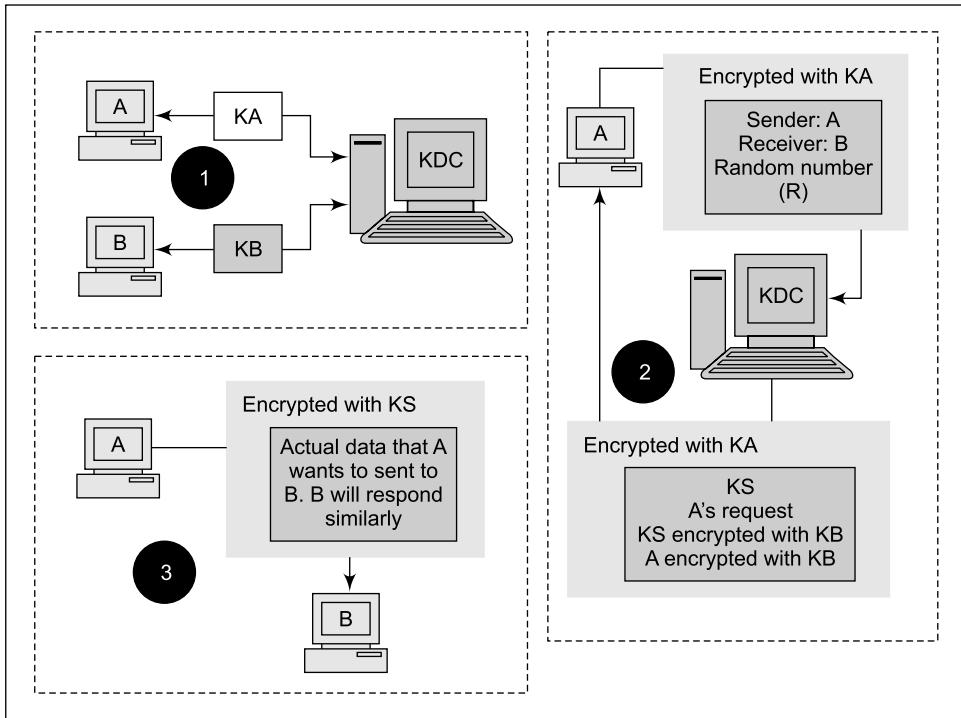


Fig. 7.43 Key Distribution Center (KDC) concept

■ 7.9 SECURITY HANDSHAKE PITFALLS ■

Having discussed several mechanisms for user authentication, let us now think about the possible **security handshake pitfalls** in the various approaches. These occur at the stage of the **handshake**, i.e. when the communicating parties are trying to authenticate each other.

There are two broad-level schemes for carrying out the handshake, as outlined in Fig. 7.44, namely **one-way authentication** and **mutual authentication**.

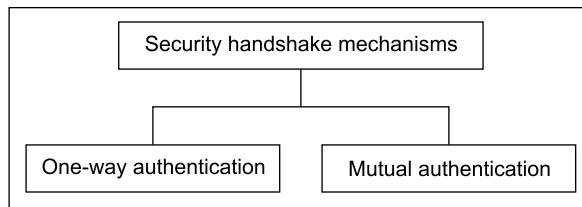


Fig. 7.44 Security handshake mechanisms

We will discuss these approaches now.

7.9.1 One-way Authentication

The idea behind one-way authentication is simple. If there are two users *A* and *B*, *B* authenticates *A*, but *A* does not authenticate *B*. Hence, we call it *one-way* authentication. There are various ways in

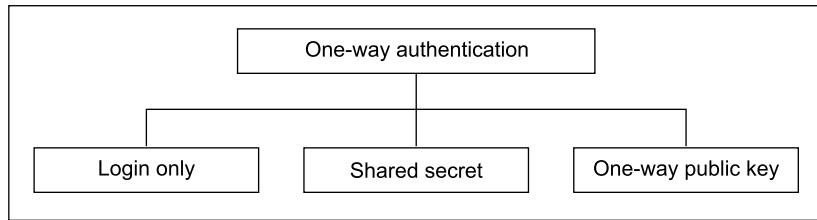


Fig. 7.45 One-way authentication approaches

which this type of authentication scheme can be implemented. Some of the notable ones are **login only**, **shared secret**, and **one-way public key**. This is shown in Fig. 7.45.

We discuss these approaches now.

1. Login Only

In this very simple scheme:

- (a) User *A* sends her user name and password in the plain-text form to the other user, *B*.
- (b) *B* verifies the user name and password. If the user name and password are correct, communication starts happening between *A* and *B*. No further encryption or integrity checks are performed.

This is shown in Fig. 7.46.

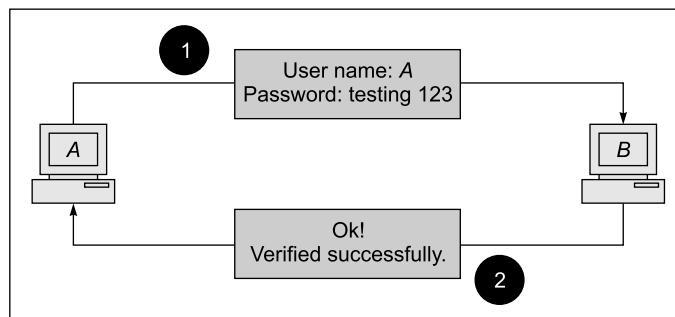


Fig. 7.46 Login-only authentication

Clearly, this approach is quite simple to implement and understand. However, it is quite ineffective. We need better mechanisms. We have seen some examples where the message digest of a password is sent by *A* to *B*, instead of sending the original password. Password encryption is another approach.

2. Shared Secret

Here, there is an assumption that *A* and *B* have agreed on a shared symmetric key *KAB*, before the actual communication begins. Hence, we have the name *shared secret* for this approach. The protocol then works as follows:

1. *A* sends her user name and password to *B*.
2. *B* creates a random challenge *R*, and sends it to *A*.

3. A encrypts the random challenge (R) with the shared symmetric key between A and B (KAB) and sends the encrypted R to B. B also encrypts the original random challenge (R) with the same shared symmetric key (KAB). If this encrypted challenge matches with the one sent by A, B considers A to be authentic.

This is shown in Fig. 7.47.

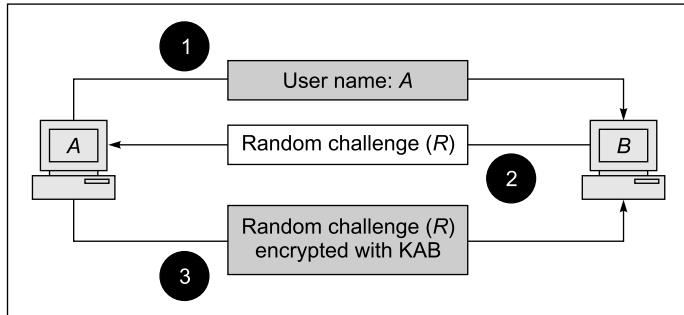


Fig. 7.47 Shared secret

This is a better scheme, because the random challenge (R) is different every time. Therefore, an attacker cannot use a previous R subsequently. However, there is a problem as well. This is one-way authentication. B authenticates A, but A does not authenticate B. Hence, an attacker C can send an old random challenge (R) to A. After A encrypts R with KAB and sends it back to C, C simply ignores it. Now A incorrectly thinks that C is B. That is, C can impersonate B and communicate with A as if it is A.

A variation of this protocol is illustrated in Fig. 7.48. Here, instead of sending the random challenge (R) as is to A, B first encrypts it with the shared symmetric key (KAB) and sends the encrypted random challenge to A. A has to decrypt it successfully, thus obtaining the original unencrypted random challenge (R), and send it back to B to authenticate herself.

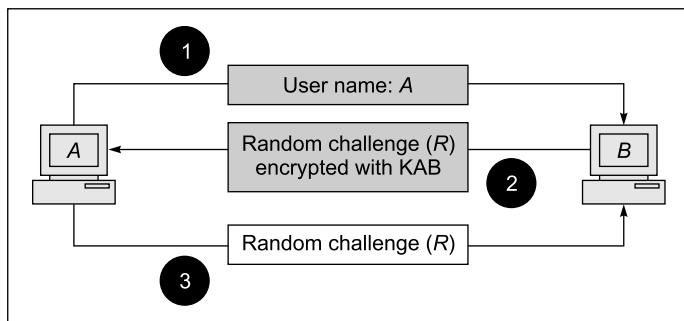


Fig. 7.48 Shared secret—Modified approach

There is yet another variation of the basic shared-secret scheme, which requires just one message from A to B. Here, there is no need for a random challenge. Instead, A simply encrypts the current time stamp with the shared symmetric key (KAB) and sends this encrypted timestamp to B. B decrypts it and if the time stamp is as expected, B thinks A is genuine. For this to happen, A and B need to synchronize their time stamps beforehand. This approach is shown in Fig. 7.49.

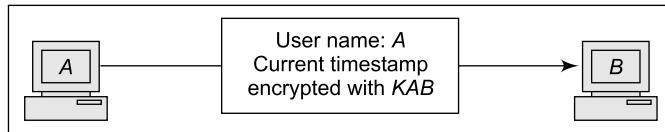


Fig. 7.49 Encrypting current time stamp with the shared symmetric key

There are certain advantages of this protocol. This protocol is quite easy to implement as a replacement for sending plain-text password. Instead of sending the user id and the plain-text password, *A* now sends the user name and the encrypted current time stamp to *B*. The protocol also does all its work in just one message, thus saving the costs of two additional messages.

However, there are drawbacks too. If the attacker is very quick, he/she can try to send a message to *B*, similar to what *A* has sent. *B* may think that this is a second, separate message from *A*. Also, if *A* uses the same authentication mechanism for multiple servers (and not only for *B*), the attacker has a good chance. When *A* sends the original message to *B*, the attacker *C* simply copies it and then uses it to try and impersonate as Alice to another server (not *B*). To prevent this, Alice should add the name of the server (i.e. *B*) to the current timestamp in her message, before encrypting it with the shared symmetric key KAB.

3. One-way Public Key

The earlier protocols are based on some shared secrets (the shared symmetric key *KAB*). If the attacker can read *B*'s database, he/she will have access to this key (*KAB*). She can then very easily act as *A* to *B*. That is, the attacker impersonates *A*. This can be avoided if the shared secret is replaced by the public-key mechanism.

Here, the idea is simple.

1. *A* sends her user name to *B*.
2. *B* sends the random challenge (*R*) to *A*.
3. *A* encrypts the random challenge (*R*) with her private key and sends it to *B*. *B* uses the public key of *A* to decrypt the encrypted random challenge, and matches it with the original random challenge (*R*).

This is shown in Fig. 7.50.

As before, this can be slightly modified to have the following flow of messages:

1. *A* sends her user name to *B*.
2. *B* creates the random challenge (*R*) and encrypts it with the public key of *A*. *B* sends this encrypted random challenge to *A*.
3. *A* decrypts the encrypted random challenge (*R*) with her private key and sends it to *B*. *B* matches it with the original random challenge (*R*).

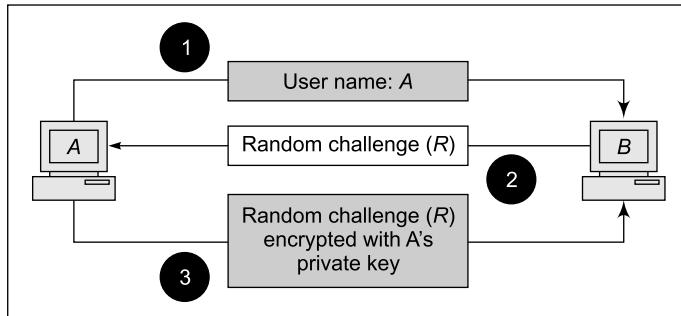


Fig. 7.50 One-way public key—Approach 1

This is shown in Fig. 7.51.

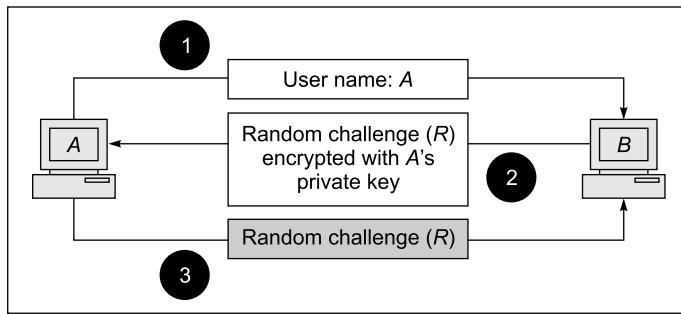


Fig. 7.51 One-way public key—Approach 2

Although these protocols are better than the earlier shared secret mechanisms, they suffer from very serious problems.

In the first approach, an attacker wants A to sign an electronic cheque for a million dollars. The attacker poses as B , and sends this electronic cheque as a random challenge to A in step 2. In step 3, A happily signs this cheque and sends it to the attacker! Thus, this approach can be used to trick someone into signing something without their knowledge!

In the second approach, the attacker's aim is different. The attacker has a message from the past, which was meant for A . Hence, the message was encrypted with A 's public key (so that only A could use her private key to decrypt it). Now, the attacker takes this encrypted message and sends as the random challenge in step 2. Poor A thinks that she is signing some random challenge with her private key in step 3. Instead, she is decrypting a message that was encrypted with her public key! She, therefore, decrypts the message meant only for her, and sends it in plain text to the attacker in step 3.

To prevent this, it needs to be mandated that the signing key should be different from the encryption key. That is, every user should have two pairs of public-private keys. One pair should be used for signing and verifying purposes; while the second should exclusively be used for encryption and decryption purposes.

7.9.2 Mutual Authentication

In mutual authentication, A and B both authenticate each other. Hence, we have the term *mutual authentication* here. This approach can also be implemented in various ways, namely **shared secret**, **public keys**, and **time stamp-based**. This is depicted in Fig. 7.52.

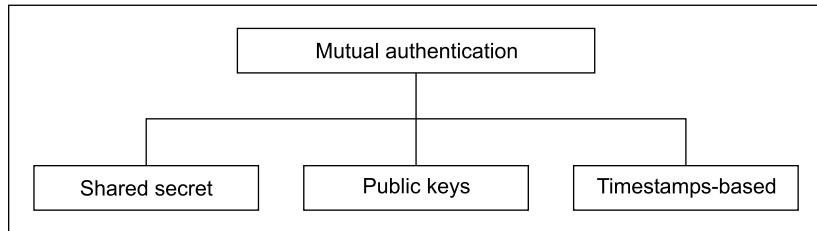


Fig. 7.52 Mutual authentication approaches

Let us discuss these approaches now.

1. Shared Secret

This protocol assumes that A and B have a shared symmetric key KAB . The protocol works as follows.

- A sends her user name to B .
- B sends a random challenge $R1$ to A .
- A encrypts $R1$ with KAB and sends it to B .
- A sends a different random challenge $R2$ to B .
- B encrypts $R2$ with KAB and sends it to A .

Now, B authenticates A as before (in steps 2 and 3). However, what is new is that A also authenticates B (in steps 4 and 5). Hence, it is *mutual authentication*. This is depicted in Fig. 7.53.

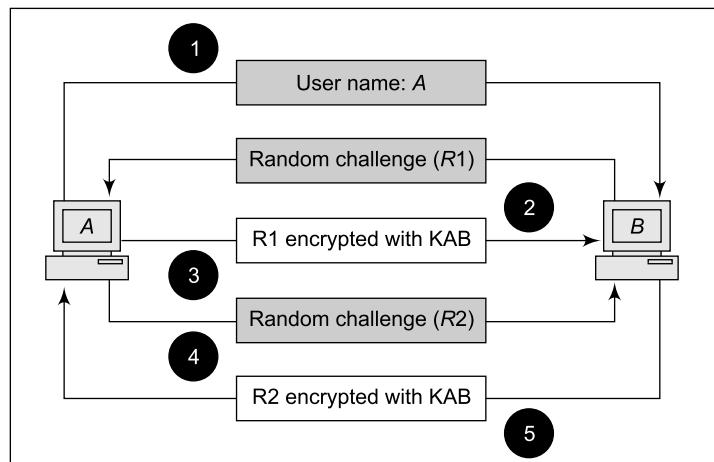


Fig. 7.53 Mutual authentication based on a shared secret

We can see that too many messages are exchanged, making this protocol inefficient. We can reduce this to just three messages, by putting more information in those three messages. This modified approach is described below.

- A sends the user name and a random challenge ($R2$) to B .
- B encrypts $R2$ with the shared symmetric key KAB , generates a new random challenge ($R1$); and sends these two to A .
- A verifies $R2$, encrypts $R1$ with the shared symmetric key KAB ; and sends it to B . B verifies $R1$.

This process is depicted in Fig. 7.54.

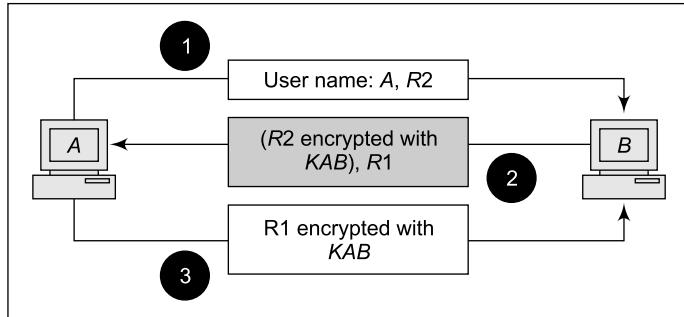


Fig. 7.54 Optimized mutual authentication based on a shared secret

This version of the protocol reduces the number of messages to three. However, it suffers from a problem called **reflection attack**. Suppose that attacker C wants to pose as A to B . First, the attacker C starts the protocol as follows:

- C sends a message to B containing user id of A and random challenge $R2$.
- B encrypts $R2$ with the shared symmetric key KAB , generates a new random challenge ($R1$); and sends these two to C . B thinks he is sending these to A .

This is shown in Fig. 7.55.

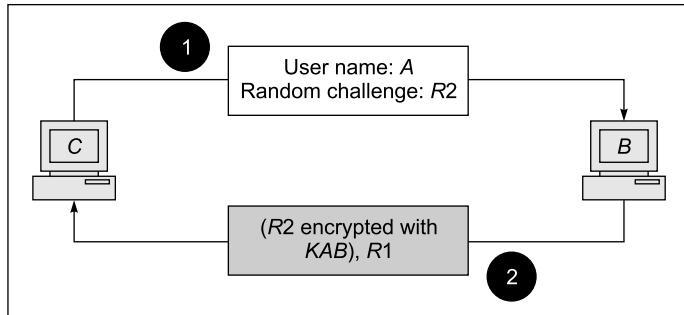


Fig. 7.55 Reflection attack—Part 1

The attacker C cannot encrypt $R1$ with KAB . However, she has managed to have B encrypt $R2$.

Now, the attacker C opens a second session with B , distinct from the first session, which is still active. Now, the following happens.

- C sends a message to B containing user id of A and random challenge $R1$.
- B encrypts $R1$ with the shared symmetric key KAB , generates a new random challenge ($R3$); and sends these two to C . B thinks he is sending these to A .

This is shown in Fig. 7.56.

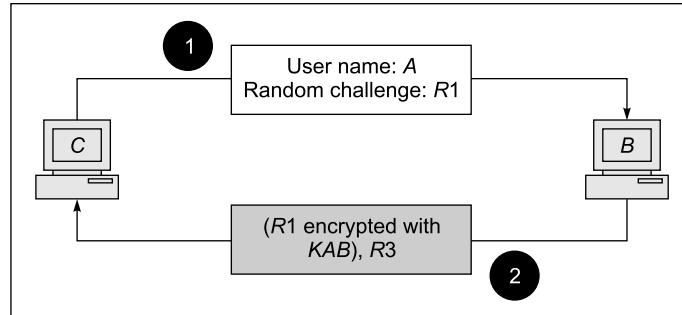


Fig. 7.56 Reflection attack—Part 2

The attacker C cannot proceed with this second session, since she cannot encrypt the new random challenge $R3$. However, she need not anyway proceed with this session. Instead, she can go back to her first session opened with B earlier. Remember she could not encrypt $R1$ with KAB in that session, and was hence waiting? Now C has $R1$ encrypted with KAB , thanks to this second session. She sends it to B and completes authentication! This is shown in Fig. 7.57.

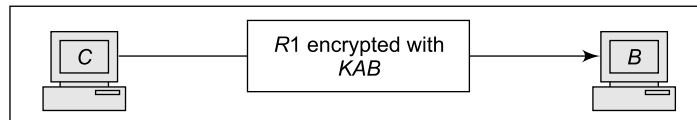


Fig. 7.57 Reflection attack—Part 3

Thus, C is able to convince B that she is A !

How can one resolve this *reflection attack*? One idea is to use different keys (say KAB and KBA). KAB should be used when A wants to encrypt something and send it to B . KBA should be used in the other direction. Therefore, B will not be able to encrypt $R1$ using KAB . This means that C cannot misuse it later, as it happens in the case of the reflection attack.

2. Public Keys

Mutual authentication can also be accomplished by using the public-key technology. If A and B know each other's public key, three messages are required to complete the mutual-authentication process, as follows:

- A sends her user name and a random challenge ($R2$) encrypted with B 's public key.

- (b) B decrypts the random challenge ($R2$) with his private key. B creates a new random challenge ($R1$) and encrypts it with A 's public key. B sends these two things (decrypted $R2$ and encrypted $R1$) to A .
- (c) A decrypts the random challenge ($R1$) with her private key and sends it to B . B verifies $R1$.

This process is shown in Fig. 7.58.

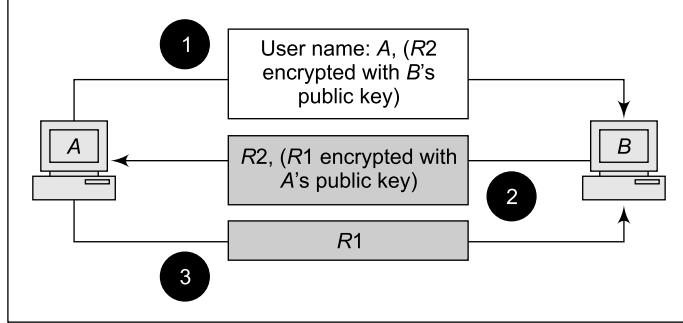


Fig. 7.58 Mutual authentication using public keys

As usual, we can have a variation of this scheme, where:

- (a) A sends her user name and $R2$ to B .
- (b) B encrypts $R2$ with his private key and sends it and $R1$ to A .
- (c) A signs $R1$ and returns it back to A .

This is shown in Fig. 7.59.

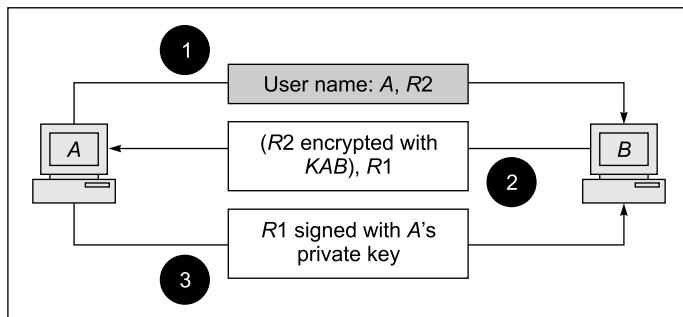


Fig. 7.59 Mutual authentication using public keys

3. Time Stamps

We can reduce the mutual-authentication process to just two steps by using time stamps, instead of random numbers as challenges. This would work as follows:

1. A sends her user name and the current time stamp encrypted with a shared symmetric key (KAB) to B .

2. B retrieves the time stamp by decrypting the above block using KAB and adds one to the timestamp. B encrypts the result with KBA (not $KAB!$) and sends it to A , along with his user name.

This approach is depicted in Fig. 7.60.

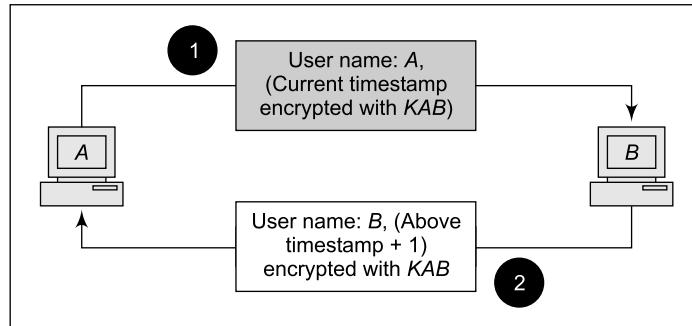


Fig. 7.60 Mutual authentication using time stamps

■ 7.10 SINGLE SIGN ON (SSO) APPROACHES ■

As we discussed, Kerberos helps in achieving SSO. There are two broad-level approaches to achieving SSO, as shown in Fig. 7.61.

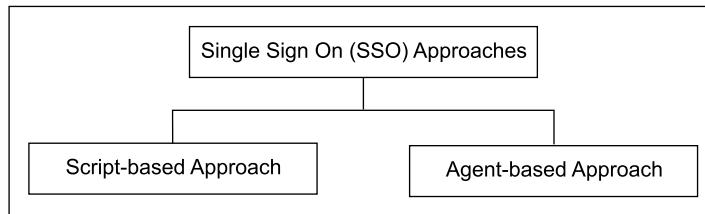


Fig. 7.61 Approaches to achieving SSO

Let us discuss these approaches now.

7.10.1 Scripting

In the **scripting** technique, the SSO software mimics user actions. It does this by interpreting a program (the script), which simulates the user-depressing keyboard keys, and reacting to individual end-system sign-on prompts. The SSO product itself holds and manages the different sets of authentication information required by the end-systems. It then extracts this information from its database and inserts it in the data stream simulated to be from the user, at the appropriate points. If needed, the script can be programmed to prompt the user to enter information at the appropriate places in the script.

In this approach, batch files and scripts containing authentication information (usually user ids and passwords, along with the login commands, if any) for each application/platform are created. When a user requests an access, a script runs in the background, and performs the same commands/tasks that the user would have to perform. The scripts can contain macros to replay user keystrokes/commands

within a shell. This is easy for users, but quite tough for system administrators, as they have to first play a role in the creation of scripts, have to maintain these scripts securely (as they contain user ids and passwords, etc.) and have to also ensure change coordination when users want to change their passwords.

7.10.2 Agents

In the **agent-based** approach, every Web server running an application must have a piece of software, called as an agent. Additionally, there is a single SSO server, which interacts with the user database to validate user credentials. Agents interact with the SSO server to achieve single sign on.

Whenever a user wants to access an application/a site participating in SSO, the agent sitting on the particular Web server intercepts the user's HTTP request, and checks for the presence of a cookie. There are two possibilities now.

- (a) If the cookie is not present, the agent sends the login page to the user, where the user must enter the SSO user id and password. This login request goes to the SSO server, which validates the user credentials, and if this process is successful, it creates a cookie for the user.
- (b) If the cookie is present, the agent opens it, validates its contents, and if they are found OK, allows further processing of the user's request.

■ 7.11 ATTACKS ON AUTHENTICATION SCHEMES ■

The most common attacks on authentication schemes are around the areas of **session hijacking** or **person-in-the-middle attack**. The idea here is that the attacker somehow manages to impersonate a legitimate user. For this to be possible, the attacker taps into electronic, i.e. computer-based conversation, between two genuine users, servers, or their combinations. Then the attacker poses as one of the users and tries to fool the other side. These attacks can be either passive or active. Passive attacks are launched to observe the conversation between two legitimate users without causing any active damage. Active attacks are launched not only to observe the traffic but also to perform an action such as stealing resources, modifying content, running an illegal transaction, utilizing network resources, etc.

A very common form of authentication attacks is the **replay attack**. As the name suggests, the attacker captures information passing from a user to a server or to another user. The attacker then tries to replay the same packets. In other words, the attacker tries to perform the same actions that the genuine user had performed earlier. Interestingly, even if the original information was encrypted, the attacker can try replaying it. For example, suppose a legitimate user entered his/her password in response to an authentication-demand request. Due to security concerns, the password was encrypted on the user's computer and then it was sent to the server. An attacker captures this encrypted password during transmission. Because the password is in an encrypted form, the attacker cannot make any sense of it. However, it does not matter! The attacker can simply replay the packet containing the encrypted password after some time when the legitimate user logs off, in response to an authentication demand request. In other words, the attacker can now try to log into the server as if he/she is the legitimate user by replaying the encrypted password of the legitimate user. Therefore, without needing to know the original password, the attacker can compromise the authentication system.

To prevent such attacks, time stamps or increasing sequence numbers are usually added to messages. This ensures that older messages that are being replayed are immediately recognized as duplicates and are discarded. This prevents the replay attacks from being successful.

■ CASE STUDY: SINGLE SIGN ON (SSO) ■

Points for Classroom Discussions

1. *What is Single Sign On (SSO)?*
2. *Why is SSO required?*
3. *What are the main ways of achieving SSO?*
4. *Discuss the working of Kerberos as an SSO protocol.*

Functional and Technical Requirements

The National Bank of India (NBI) is a very successful bank in India for many years. To keep itself tuned to the modern world, the bank had started its computerization many years ago. Now, the bank has moved into the arena of Internet banking. The bank was into retail, corporate, and investment banking. All these services were moved to the Internet. Therefore, the bank's customers could access all the necessary banking services via the Internet. Within each category of services, the bank offered many individual applications. For instance, within the retail-banking segment, the bank offered solutions in the areas of Internet account access, electronic bill payment, direct debits, etc.

This case study deals with the retail-banking segment only. Each of the applications within this segment for the bank works fine. Customers are very happy with the Internet account access facilities and with the provisions of online bill payment and direct debits. They use these services quite frequently and with ease. This has caused more customers to opt for Internet banking services of the bank. However, a major concern has surfaced of late, which is described as follows.

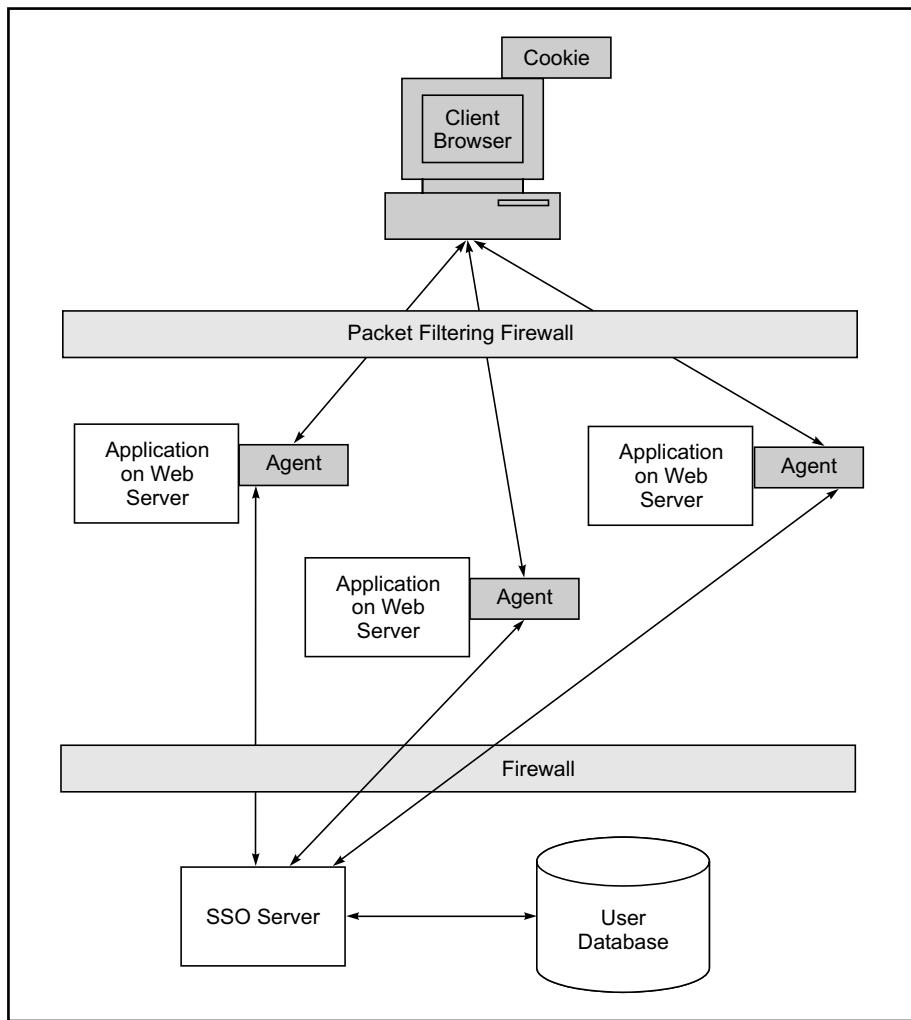
The applications within each segment work very fine. However, since the applications were developed with an isolated design in mind, each application has its own user authentication model. That is, Internet account access, electronic bill payment and direct debit, all maintain their own user databases and the user has to log on to the particular application as and when she wants to access it. For instance, suppose that the user logs on to the electronic bill payment application and pays her electricity bill. To check the effect of this payment on her bank account, the user is required to now log on to the Internet account access module separately! This is quite annoying for the users, since the bank hosts all the applications and the end users feel that they should not have to worry about the internal design issues of the applications hosted by the bank. They should get a single authentication module. In other words, once they log on to any one application (say Internet account access) using the id and password, they should automatically be logged on to the other applications (i.e. electronic bill payment and direct debit). It is quite tiring for them to have to remember three separate user ids and passwords and use them during the application logons.

Thus, the requirement is to group all the user logins into a single login and offer a single user id and password for the bank users. A user should be able to log on to the bank's site using this id and password and once she is logged on, she must not have to log on separately to access each of the applications. The applications should automatically detect that the user has already authenticated her to one of the applications and simply reuse the credentials of that authentication.

Clearly, this requirement calls for the solution of Single Sign On (SSO). SSO provides a single authentication interface to end-users. Once a user logs on to one of the applications within a group of applications successfully, she does not have to log on to other applications separately. The authentication credentials of the user are simply picked up from the first log on and are reused by the other applications.

Proposed Solution

SSO solutions are based on one of the two broad level approaches: the script approach and the agent approach. We can choose either one. However, since the agent approach is considered more suitable for Web-based applications, we shall use it here. As we know, an agent is a small program that runs on each of the Web servers that host an application within the application framework. This agent helps coordinate the SSO workflow in terms of user authentication and session handling.



SSO Architecture

The bank's application runs on Intel-based servers, on Windows NT 4.0 operating system. These applications are developed on the Microsoft technology, using ASP 2.0 and SQL Server 6.0. The Web server is Microsoft's Internet Information Server (IIS) 4.0. There is an involvement of Microsoft Transaction Server (MTS) for transaction handling. However, the SSO requirement need not be concerned with it.

In order to develop the agents, no special hardware/software requirements are visualized. The agents are simple programs sitting on the IIS Web server and they can be written in the form of ISAPI applications (i.e. the filters on the IIS Web server).

The broad level solution architecture is depicted in the figure.

As we can see, the SSO architecture contains two main pieces: the agents sitting on the Web server and a dedicated SSO server. The purpose of these two pieces is as follows:

Agents An agent would intercept every HTTP request arriving at the Web server. There is one agent per Web server, which hosts an application. It interacts with the client browser on the user side and with the SSO server on the application side.

SSO Server The SSO server uses transient cookies to provide session management functionalities. A cookie contains information such as the user id, session id, session creation time, session expiration time, etc.

Application Flow

The application flow would be as follows:

1. For every HTTP request that is intercepted, the agent will look for the existence of a valid cookie. There are two possibilities:
 - (b) If the cookie is not found, it will initiate a challenge screen to allow the user to enter her credentials. The credentials may be a simple user id/password or user id and digital certificate, depending on the mechanism chosen for user authentication. The agent would receive these details entered by the user and forward them to the SSO server, which would validate them against the user database.

If the user is authenticated successfully, the SSO server will respond back with a credential token. The agent may forward part of the token to the client browser as a cookie. The cookie may contain basic information like session identifier, session expiry time, etc.
 - (c) If the agent finds an existing cookie along with an intercepted HTTP request, it will request the SSO server to decrypt the same and determine whether:
 - The user is already authenticated
 - The authentication is still valid
 - The user can access the application associated with this agent

If the authentication has expired, it will ask the user to provide authentication details once again.

2. The SSO server will receive authentication requests from the agents. It will then initiate a call to an authentication ASP. This ASP will authenticate the user against the user database and return success or failure.

On successful authentication, the SSO server will build a credential token with some information and return the whole or part of this token to the agent.

If the user is already authenticated and the agent requests for verification, the SSO server will determine whether the user is allowed an access to the system. Accordingly, it will initiate the authentication process or will inform the agent to allow user to access the application, if the session is still valid.



Summary

- Authentication is concerned with establishing the identity of a user/ system.
- Clear-text passwords are the most common mechanism of authentication.
- Clear-text passwords have security issues associated with them.
- Message digests of passwords are also used to avoid transmission of passwords over the network.
- Encryption of passwords is a better scheme.
- Something derived from passwords is also used for authentication.
- Random challenge adds to the security of password mechanisms.
- Password policies can help make password-based authentication mechanisms secure.
- Authentication tokens are more secure.
- Every login request of an authentication token generates a new password.
- Authentication token is a two-factor authentication mechanism.
- Authentication tokens can be of challenge/response or time-based types.
- Time-based tokens are more popular, as they are more automated.
- Smart cards are highly secure devices, as they perform cryptographic functionalities inside the card.
- Smart cards currently have many incompatibility issues.
- Certificate-based authentication is an effective authentication mechanism.
- In certificate-based authentication, the user's private key is used to sign a random challenge.
- For certificate-based authentication to work, the user's certificate details should be available with the server.
- Biometric devices are based on human characteristics.
- Kerberos is a widely used authentication protocol.
- Kerberos allocates the job of authenticating users to a central server, and the job of allowing users access to various systems/servers to a different server.
- Kerberos uses the concept of *tickets*.
- There are certain variations between Kerberos version 4 and 5.
- Security Handshake Pitfalls is an interesting problem to solve.
- Authentication can be classified in another manner: one-way and mutual.
- In one-way authentication, one party authenticates the other, but the other party does not authenticate the first one.
- One-way authentication leads to various kinds of problems.

- At a broad level, one-way authentication can be classified into login only, shared secret, and one-way public key.
- Mutual authentication involves authentication of both parties.
- Mutual authentication is more reliable.
- Mutual authentication can be classified into shared secret, public keys, and time-stamp-based.
- Reflection attack involves an attacker opening two sessions to impersonate a user.
- Single Sign On (SSO) allows users to have a single user id and password for multiple services/applications.
- SSO can be achieved using scripts or agents.



Key Terms and Concepts

- | | |
|---|--|
| <ul style="list-style-type: none">● 1-factor authentication● Agent● Biometric authentication● Challenge/response token● Handshake● Key distribution Center (KDC)● Multi-factor authentication●Nonce● One-way authentication● Password● Postdatable tickets● Reflection attack● Replay attack● Shared secret● Scripting● Time-stamp-based | <ul style="list-style-type: none">● 2-factor authentication● Authentication token● Certificate-based authentication● Forwardable tickets● Kerberos● Login only● Mutual authentication● One-time password● One-way public key● Password policy● Random challenge● Renewable tickets● Security handshake pitfalls● Single Sign On (SSO)● Time-based token● User authenticator |
|---|--|



PRACTICE SET

■ Multiple-Choice Questions

1. Determining the identity of a user is called _____.
(a) authentication
(b) authorization
(c) confidentiality
(d) access control
2. _____ is the most common authentication mechanism.
(a) Smart card
(b) PIN
(c) Biometrics
(d) Password
3. Many organizations specify a _____ for setting up rules regarding passwords.
(a) authentication law
(b) password law
(c) password policy
(d) user id rule
4. _____ forms the basis for the randomness of an authentication token.
(a) Password
(b) Seed
(c) User id
(d) Message digest
5. Password-based authentication is an example of _____ authentication.
(a) 1-factor
(b) 2-factor
(c) 3-factor
(d) 4-factor
6. In time-based tokens, the variable factor is _____.
(a) seed
(b) random challenge
(c) time
(d) password
7. In certificate-based authentication, the user needs to enter a password for accessing _____.
(a) public-key file
(b) private-key file
(c) seed
(d) random challenge
8. _____ are capable of cryptographic operations.
(a) Credit cards
(b) ATM cards
(c) Debit cards
(d) Smart cards
9. Biometric authentication works on the basis of _____.
(a) human characteristics
(b) passwords
(c) smart cards
(d) PINs
10. Kerberos provides for _____.
(a) encryption
(b) SSO
(c) remote login
(d) local login
11. To launch reflection attack, an attacker needs to open _____ sessions.
(a) 2
(b) 3
(c) 4
(d) 0 or 1
12. In _____ authentication mechanism, only one party authenticates the other.
(a) one-way
(b) mutual
(c) time-stamp-based
(d) mutual with public keys
13. In Kerberos, the server that allows users to access various applications/servers is called _____.
(a) AS
(b) TGT

- (c) TGS
 (d) file server
14. In one-way public-key based authentication mechanism, ideally a total of _____ messages get exchanged between the communicating parties.
 (a) 0
 (b) 2
 (c) 0-2
15. In Kerberos, _____ shares a unique password with every user in the system.
 (a) AS
 (b) TGT
 (c) TGS
 (d) file server

■ Exercises

1. What are the problems associated with clear-text passwords?
2. What is the improvement over clear-text passwords? What are its drawbacks?
3. How does ‘something derived from a password’ work? What is the main drawback here?
4. How can one add unpredictability to the mechanism of something derived from the password?
5. Can an unauthorized user use an authentication token?
6. What are the three aspects of a 3-factor authentication?
7. What is the difference between challenge/response tokens and time-based tokens?
8. How does one prevent the misuse of another user’s certificate in certificate-based authentication?
9. What is the problem with smart cards if large data needs to be processed?
10. Why do we need to take multiple samples during the user-registration process?
11. Explain the security handshake pitfalls.
12. What is reflection attack? How can it be prevented ?
13. Explain any one one-way authentication mechanism with its advantages and drawbacks.
14. Explain any one mutual authentication mechanism with its advantages and drawbacks.
15. What is SSO?

■ Design/Programming Exercises

1. The UNIX password is combined with a 2-byte *salt* to prevent dictionary attacks. Would the security of the password be increased substantially if the *salt* size is increased to four bits? Why?
2. In many in-house applications (typically banking), two or more persons must type their passwords to create a combined password, which allows access to the system resources. Why do you think this is necessary?
3. Can the above scheme also be used for storing the private keys of users in a PKI implementation? How?
4. How do sites such as Yahoo and Indiatimes normally accept your passwords? Are they in clear text? How do some of these sites provide security as an addition to this mechanism of transmitting passwords?
5. SSL can also involve client-side authentication (using the client’s digital certificate), which is one way of certificate-based authentication. However, this is not foolproof. Why is this the case?
6. In any SSO solution, one always questions the possibility of cross-domain SSO. What does this refer to? How can it be achieved?
7. Implement a certificate-based authentication mechanism in Java.

8. Investigate more about the various security tokens available in the market and analyze their features.
9. It is said that a 2-factor authentication is not necessarily better. Why?
10. Some sites (e.g. Yahoo) ask the user to view an image containing some text and enter the characters seen there on the screen as an added level of authentication. What is the reason for this and how does it work? What is it called?
11. What is phishing? How is it related to authentication? How would you prevent possible phising attacks?
12. Do you think a mobile phone can be made a part of user authentication? How?
13. Write a Java program to encrypt users' passwords before they are stored in a database table, and to retrieve them whenever they are to be brought back for verification.
14. Can the user's user ID be considered a session ID in a Web-based application? Why?
15. Is it safe to store the user's user ID in a cookie? Why?



PRACTICAL IMPLEMENTATIONS OF CRYPTOGRAPHY/ SECURITY

■ 8.1 INTRODUCTION ■

Real-life cryptography involves a lot of infrastructure issues. It is very tough for everyone to implement cryptographic algorithms themselves (e.g. implement RSA in Java). This would not only lead to a lot of incompatibilities, but also create a lot of security concerns, insufficient testing issues, etc. That is why it is preferable to use industry-standard ready-made cryptographic solutions.

Practical cryptographic implementations are of three types: (a) Java cryptography, (b) Microsoft .NET cryptography, and (c) Third-party solutions. Both Java and Microsoft cryptographic solutions are free, and do not require any licensing. However, third-party cryptographic toolkits can be expensive, but they also provide many more features.

The US government had earlier imposed restrictions on strong cryptography to defeat the malicious intentions of terrorist organizations. However, these restrictions have now been taken off.

This chapter discusses all the practical technologies, implementations, and issues of cryptography. It explains the way Java and Microsoft solutions work, and also introduces the third-party cryptographic solutions. The main stress is on cementing the basic concepts, rather than going into deep syntactical details.

The chapter then looks at the security of operating systems. It focuses on key security concepts in UNIX and Windows 2000 operating systems.

Finally, the chapter discusses issues in database security.

■ 8.2 CRYPTOGRAPHIC SOLUTIONS USING JAVA ■

8.2.1 Introduction

The Java programming language has become one of the major success stories of modern computing. Java is everywhere—it is on the Web browsers (in the form of applets), on the Web servers (in the form of servlets or Java Server Pages, i.e. JSP), on the application servers (in the form of Enterprise Java Beans, i.e. EJB) as well as for making all these technologies work together in the form of technologies such as Remote Method Invocation (RMI), Java Messaging Service (JMS), Java Database Connectivity (JDBC), and so on. This automatically means that Java should be a safe language to use (i.e. an applet must not misbehave on the Web browser client, for example), and should provide for cryptographic functionalities (i.e. facilities for encryption, message digests, digital signatures, etc.).

There are several mechanisms in place, which ensure that Java is a safe language to use. We shall not discuss those mechanisms, and their implications here. The focus of our discussion will be the cryptographic services provided by Java.

At a very broad level, we can consider the Java cryptographic framework as consisting of two main technologies, as shown in Fig. 8.1.

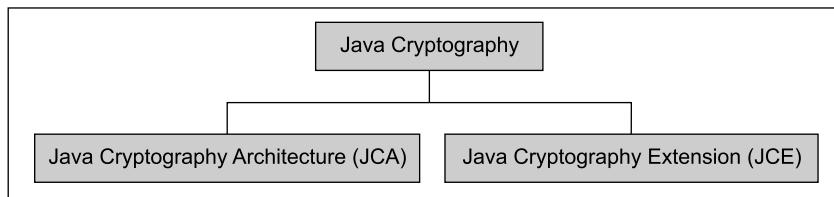


Fig. 8.1 Java cryptographic framework

Let us examine what this means now.

- **Java Cryptography Architecture (JCA)** is a set of classes that provide cryptographic capabilities to Java programs. Most significantly, JCA is a part of the default Java application development environment, i.e. Java Development Kit (JDK) itself. This means that when you have JDK, you automatically have JCA. JCA was introduced for the first time in JDK version 1.1, and was significantly enhanced in JDK version 1.2 (which was more commonly known as *Java 2*).
- **Java Cryptography Extension (JCE)**, on the other hand, is not a part of the core Java JDK *per se*. Instead, it is an additional piece of software that requires special licensing. The reason for separating JCE from JCA is the export restrictions imposed by the US government, as we shall see.

We shall now examine JCA and JCE from a conceptual point of view.

8.2.2 Java Cryptography Architecture (JCA)

1. Introduction

As we mentioned, JCA is a part of the core Java framework. It automatically comes with the JDK software, and does not need any special licensing. JCA provides the basic cryptographic functionalities

to a programmer using the Java language. The cryptographic functionalities (such as access control, permissions, key pairs, message digests, and digital signatures) are provided as a set of abstract classes in a Java package called *security*. Sun provides the actual implementation of these classes in the JDK. Additionally, we can provide our own implementations of these abstract classes. Let us understand this in more detail.

JCA is commonly known as *provider architecture*. The primary goal in the design of JCA is to separate the cryptographic concepts (i.e. the *interfaces*) from their actual algorithmic implementations (i.e. *implementations*). Let us examine this in more detail.

In order to achieve programming-language independence, the object-oriented principle of **interfaces** is used. An interface is simply a set of functions (methods) that signify *what* that interface can do (i.e. the behavior of the interface). It does not contain the **implementation** details (i.e. *how* it is done). Let us understand this with a simple example.

When we buy an audio system, we do not worry about the internal aspects like the electronic components inside, the voltages and currents required to work with them, etc. Instead, the manufacturer provides a set of *interfaces* to us. We can press a button to eject a CD, change the volume or skip a track. Internally, that translates to various operations at the electronic component level. This set of internal operations is called *implementation*. Thus, our life becomes easy since we do not have to worry about the internal details (implementation) of an operation. We need to only know how to use them (interface). This is shown in Fig. 8.2.

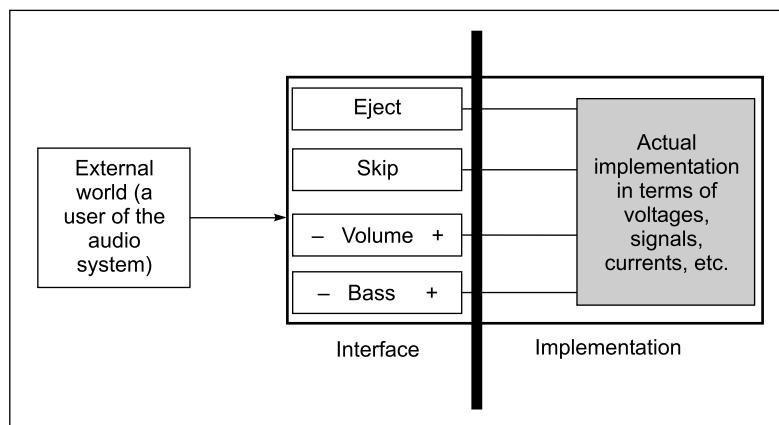


Fig. 8.2 Interface and implementation

The main purpose that this approach serves is the provision of a *plug-able* architecture. This allows the user to change the internal details (such as implementing the volume-control mechanism differently) without needing to change the outer interface (i.e. the volume-control buttons). This is precisely what a *provider architecture* such as JCA does. In JCA, we provide conceptual cryptographic functionalities, and allow them to be implemented in a variety of ways. This allows different vendors to provide their own implementations of the cryptographic tools. This makes the JCA architecture vendor-independent and expandable.

In order to achieve this, the JCA package consists of a number of classes, called **engine classes**. An engine class is a logical representation of a cryptographic functionality (such as a message digest or

a digital signature). For example, there are many algorithms available to perform digital signatures, which differ vastly from one another in terms of implementation. However, at a broad level, all provide the same abstracted functionality of a digital signature. Therefore, there will be only a single javasecurity signature class in JCA, which represents all the possible variations of the digital signature algorithms. Another class, called **provider**, does the actual implementation of these algorithms. The provider class may be supplied by a number of vendors. This concept is shown in Fig. 8.3.

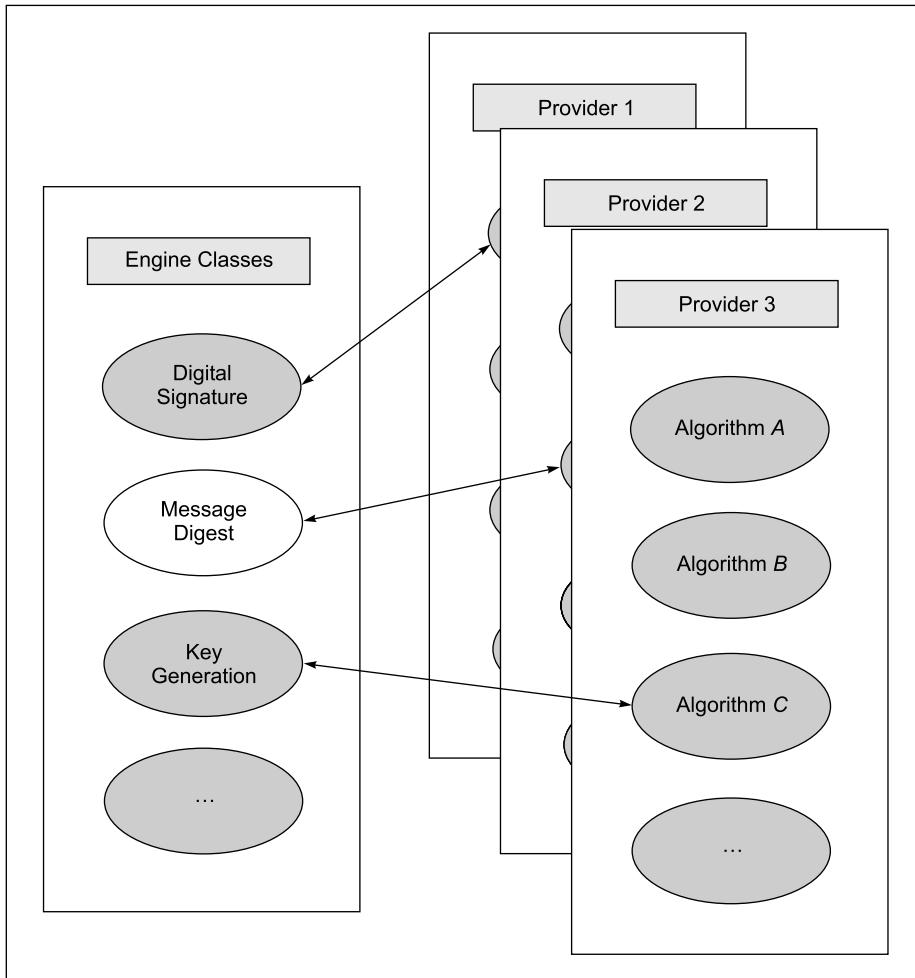


Fig. 8.3 Relation between engine and provider classes

As we can see, an application developer need not at all worry about the provider classes. An application programmer has to make the appropriate calls to the engine classes. The relationship between the engine classes and the provider classes is established through parameter files, which need not be considered while developing an application using JCA. More specifically, we specify the provider classes in a properties file that has a pre-determined name and location. When the Java Virtual Machine (JVM) begins execution, it consults this property file and loads the appropriate provider classes in the memory.

This concept can be shown in another fashion, as illustrated in Fig. 8.4.

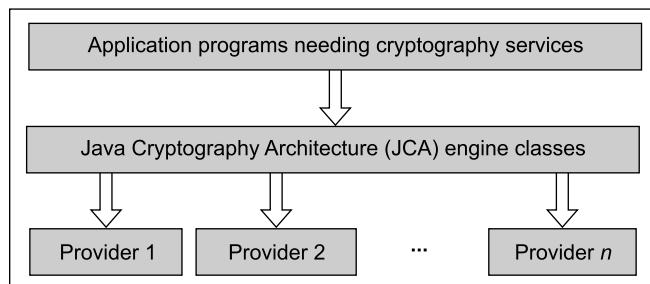


Fig. 8.4 JCA engine classes and providers

2. Key Management in JCA

One important question in any cryptographic system is how the keys used in any cryptographic operation are created and managed. Java 1.1 provided a utility called **JavaKey**. However, this utility had not matured enough, and had several problems. The main concern here was that it used to store the private and public keys of a user in the same unprotected database. Consequently, the designers of Java decided to come up with an improved utility for key creation, storage, and management.

As we had discussed earlier, Java 2 came with a new utility called **Keytool**. Keytool stores the public and private keys separately, and protects them with passwords. The database used by the keytool to store the keys is called **keystore**. Usually, the keystore is a simple computer file with a *.keystore* extension, in the user's home directory. Let us list down a few important services provided by keytool:

- Creation of key pairs and self-signed certificates
- Export certificates
- Issue Certificate Signing Requests (CSR) to be sent to a Certification Authority (CA) for requesting a certificate
- Import other people's certificates for signature verification

For example, we can type *keytool-genkey* as a command to generate a key pair. Interestingly, the keystore database can be accessed even programmatically. For this, keystore is treated as a class, and used in an application program.

3. JCA Features

Having understood the basic concepts behind the JCA architecture, we will now discuss some of the features offered by JCA in terms of cryptographic capabilities.

As we might have realized by now, application developers are usually interested only in using the engine classes for performing the desired cryptographic operations. Each engine class has a public interface, i.e. a set of methods, which specify the operations that the engine class can perform. This is true for most Object Oriented (OO) systems these days, anyway. However, none of the engine classes has a public constructor. Instead, every engine class provides a `getInstance (?)` method, which accepts the name of the desired algorithm as an argument and returns an instance of the appropriate class.

Let us now look at an example of creating a message digest using the SHA-1 algorithm, written using JCA. The code contains detailed comments for those not familiar with the Java syntax, as shown in Fig. 8.5.

```

public class CreateDigest {
    public static void main (String args [ ]) {
        try {
            // Create an output file for storing the message digest when it is created.
            FileOutputStream fos = new FileOutputStream ("sample");

            // Create an object of the class MessageDigest. The getInstance method
            // creates this instance and stores it in the object md. We use the SHA-1
            // algorithm. It is ok to write it as SHA instead of SHA-1.
            MessageDigest md = MessageDigest.getInstance ("SHA");

            // Create an output object of the standard Java type output stream. This will
            // be used later in our program. Associate it with the output file.
            ObjectOutputStream oos = new ObjectOutputStream (fos);

            // Specify the input string over which the message digest is to be created.
            String data = "This is an input string for digesting";

            // Transform the string format into byte (binary) format.
            byte buffer [ ] = data.getBytes ();

            // Call the update method of the message digest object. This method adds
            // the specified input data to the digest, over which finally the digest will be
            // calculated.
            md.update (buffer);

            // Write the original data to the output file.
            oos.writeObject (data);

            // Calculate and write the message digest to the output file.
            oos.writeObject (md.digest ());
        }catch (Exception e) {
            System.out.println (e);
        }
    }
}

```

Fig. 8.5 Example of message-digest creation in Java using JCA

Let us summarize the steps involved in creating the message digest, as shown in the figure. The call to the *getInstance ()* method finds and loads a message-digest object that implements the SHA-1 message-digest algorithm. After we create our input string to be digested, we pass that data to the *update()* method of the message-digest object, and then write it to the output file. Finally, we call the *digest()* method to create the message digest, and add it to the same file.

Similar functionalities exist for digital signature and other cryptographic functionalities in JCA. In every case, the separation between the engine classes and the provider classes is carefully maintained and managed.

8.2.3 Java Cryptography Extension (JCE)

1. The Politics of Cryptography

The cryptographic functionality of encryption of data falls in the category of Java Cryptography Extension (JCE). This might actually seem quite strange. Why put the functionalities such as message digests and digital signatures in one package (JCA) and the likes of encryption in another package (JCE)? Well, there is an important reason behind this.

When JCA and JCE were designed, the US government used to put strict export restrictions on cryptographic software. Since Sun Microsystems is located in the US, the JCE software developed by them falls in the category. The reason for the restriction is that the US wanted to be able to prevent outside terrorist and anti-national elements from using strong cryptography (e.g. 128-bit encryption) for illegal means. Thus, earlier JCE could be used only within the US and Canada. However, this stance of the US government is now changed. Other countries want to be able to freely use this technology. At the other extreme, some people want the strong cryptographic technologies to be banned even within the US. As per the current state of affairs, the US government now allows strong cryptography to be used outside of the US and Canada. However, this was not the case, and that is why originally JCE was separated from JCA.

Even in the days of restrictions, things were not so simple. Companies had found loopholes to create their own *clones* of JCE as third-party implementations. Of course, for the sake of completeness, it must be pointed out that JCE was not the only cryptography software that came under the restrictions imposed by the US government. Several other cryptographic software applications as well as the algorithms themselves were restricted. Moreover, many algorithms are patented as well. This means that the users of the algorithm must pay a licensing fee to the patent-holders. For instance, RSA Data Security Inc. holds patents in the US on many algorithms based on RSA encryption and digital signature. Similarly, Ascom System AG (Switzerland) holds the patent for the IDEA algorithm. Therefore, if the application developers reside in a country where these patent rules apply, the application developers as well as the end users must pay a licensing fee to the patent holders, depending on the clauses in the patent document.

Coming back to the original discussion of the historical restrictions on JCE, Java application developers who wanted to use JCE had to observe the following points.

- They had to procure JCE separately from the JDK. The official JCE developed by Sun Microsystems could be obtained only by the citizens of the US and Canada. People residing in other countries could procure third-party implementations of JCE.
- Electronic documentation of JCE was also supposed to follow the same guidelines as above. (However, in practice, this was never the case. This clause was violated to a great extent).
- The JCE APIs and any applications developed using JCE could not be used outside the US or Canada. An interesting situation was: what if the developers hosted the application within the US or Canada, and allowed applets to be downloaded on the browser clients outside the US or Canada (which, in turn, used cryptography)? This could be highly possible in this age of Internet. Therefore, this was also restricted.

2. JCE Architecture

The architecture of JCE follows the same pattern as that of JCA. It is also based on the concept of the engine classes and the provider classes. There is only one difference. JCE also comes with one implementation of the engine classes. This implementation is the default implementation, provided by Sun Microsystems. Since the architecture of JCE is very similar to that of JCA, we will not discuss it any further.

We will conclude our discussion with an example demonstrating an encryption process using JCE. This is shown in Fig. 8.6. As before, the example contains a lot of comments to help us understand what is going on inside the code.

```

public class EncryptionDemo {
    public static void main (String args [ ] ) {
        try {
            // The KeyGenerator class provided by JCE can be used to generate
            // symmetric (secret) keys. We specify which algorithm we will use with
            // this symmetric key for actual encryption. Here, we specify it as DES.
            KeyGenerator kg = KeyGenerator.getInstance ("DES");

            // The Cipher class is used to instantiate an object of the specified
            // encryption algorithm class. We can also specify the mode and padding
            // scheme to be used during the encryption process. In this case, we indicate
            // that we want to use the DES encryption algorithm in the Cipher Block
            // Chaining (CBC) mode with padding as specified in PKCS#5 standard.
            Cipher c = Cipher.getInstance ("DES/CBC/PKCS5Padding");

            // The generateKey function generates a symmetric key, using the
            // parameters discussed above. The key is stored in a variable called key.
            Key key = kg.generateKey ( );

            // JCE demands that once the key is generated, we must execute an init ()
            // method against the Cipher object created earlier. This method takes two
            // parameters. The first parameter specifies if we want to perform
            // encryption or decryption. The second parameter specifies which key to
            // use in that operation.
            c.init (Cipher.ENCRYPT_MODE, key);

            // Now we specify the plain text, which we want to encrypt. We also
            // transform it into a byte array.
            byte plaintext [ ] = "I am plain text. Please encrypt me.".getBytes ();

            // Execute the doFinal () method, which performs the actual encryption or
            // decryption (in this case, it is encryption). It accepts the plain text as the
            // input parameter, and returns cipher text. Also, this method is a part of the
            // Cipher object (note the prefix c.).
            byte ciphertext [ ] = c.doFinal (plaintext);
        } catch (Exception e) {
            e.printStackTrace ();
        }
    }
}

```

Fig. 8.6 Example of encryption in Java using JCE

The comments inside the code should have explained what is going on at each stage during the encryption process. We will not describe those steps again.

8.2.4 Conclusions

Both JCA and JCE are strong cryptographic architectures. They have been carefully planned and designed, so as to allow for future expansions as well as vendor-independence. However, the biggest problem with the use of Java cryptography was related to the licensing issues. Because of the US export laws, JCE did not come as a part of the core JDK, and it could also be procured outside the US and Canada easily. This was also why JCE was not a part of the Web-browser software.

Now that the restrictions have been lifted, application developers can use JCE freely. The biggest advantage of using JCE is that it is free.

8.3 CRYPTOGRAPHIC SOLUTIONS USING MICROSOFT .NET FRAMEWORK

8.3.1 Class Model

We take a look at the cryptography features provided by Microsoft in its .NET framework.

Like JCA and JCE, the .NET framework cryptographic object model was designed to allow the addition of new algorithms and implementations in an effortless manner. In the model, a category of algorithms such as *symmetric algorithms* is modeled as a single abstract base class. Individual algorithms are represented by the respective abstract algorithm classes. Finally, there is a concrete implementation class per abstract algorithm class. Figure 8.7 shows the idea.

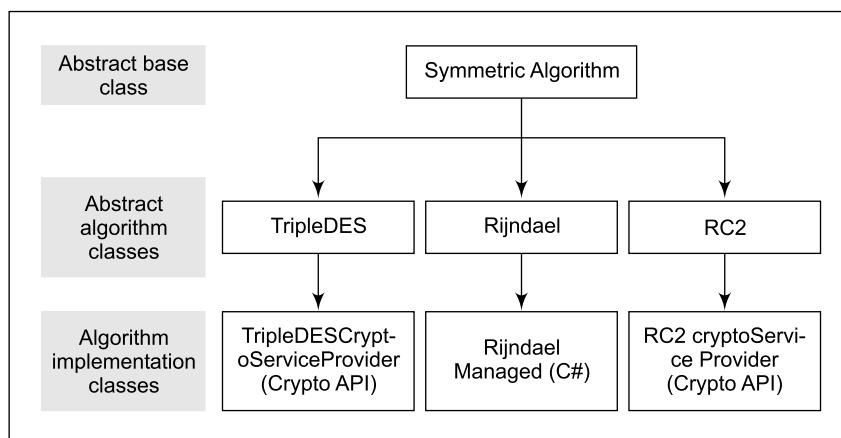


Fig. 8.7 The symmetric-key cryptographic object model in .NET

As we can see, `SymmetricAlgorithm` is the abstract base class. It is inherited by a number of abstract algorithm classes. We have shown three of them. Each represents an abstraction of a particular algorithm, such as DES. Finally, respective algorithm implementation classes are subclasses of the abstract-algorithm classes. As we can see, each can be potentially implemented differently.

- The abstract base class defines methods and properties common to all the algorithms in this class. For example, the `SymmetricAlgorithm` class defines a property by the name `LegalKeySizes`, which tells us the length of valid keys (in bits) for a cipher.
- The abstract-algorithm classes have two functions: (a) They expose algorithm-specific details (e.g. key sizes and block sizes) by implementing the properties defined by the abstract base class, as mentioned above. For example, for the `Rijndael` algorithm, the `LegalKeySizes` property can have one of the following values: 128, 192, and 256. (b) In addition, they define properties and methods that are specific to every implementation of the algorithm they represent, but do not apply to other algorithms. For example, for `TripleDES`, there is a method named `IsWeakKey`, which is specific only to that algorithm. The abstract algorithm class for DES will define this method, but the abstract algorithm class for other algorithms will not.

- The algorithm-implementation classes implement the algorithm to carry out the specified action. For example, the triple DES algorithm in .NET can be implemented as a class named TripleDESCryptoServiceProvider. In .NET, the convention is to add the name of the service provider to the implementation class. In this case, the provider of the implementation is Microsoft Cryptographic Service Provider (CSP), which ships with Microsoft Windows.

The way we have the SymmetricAlgorithm class for symmetric-key encryption algorithms, we have HashAlgorithm abstract base class for message digests, and the AsymmetricAlgorithm abstract base class for asymmetric-key encryption and digital signatures. We show some samples of these two abstract base classes and their hierarchies in Fig. 8.8 and Fig. 8.9.

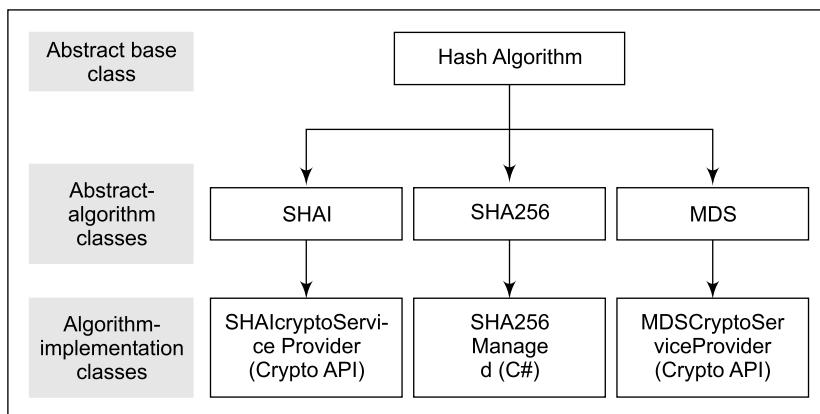


Fig. 8.8 The message-digest cryptographic object model in .NET

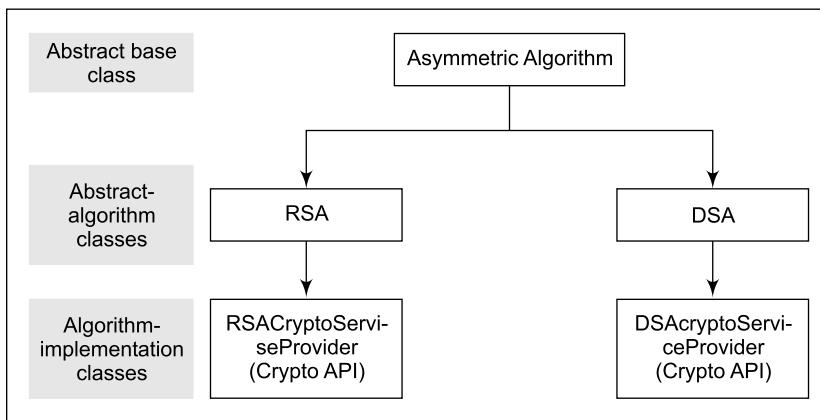


Fig. 8.9 The asymmetric-key cryptographic object model in .NET

8.3.2 Programmer's View

How does a programmer use the cryptographic class model in .NET? For this purpose, the .NET framework includes a configuration system for the various cryptographic classes. This defines a *default implementation type* for each abstract base class and each abstract algorithm class. Every abstract class

in the class model defines a static *Create ()* method. This method creates an instance of the default implementation for the abstract class. Using this feature, the programmer can simply ask for the *default implementation of the SHA-1 algorithm* like this:

```
SHA1 sha1 = SHA.Create();
```

Thus, the programmer is not bothered about the implementation details of the SHA algorithm. Of course, there are ways to directly invoke a specific implementation of an algorithm, whenever needed. We shall overlook that detail.

Once we get an object of the desired class (in this case, an object named *sha1* of class *SHA1*), we can call the appropriate method for the cryptographic operation. For example, to compute the message digest of a message, we can call a method named *ComputeHash*, as follows:

```
byte [ ] hashValue = sha1.ComputeHash (ourMessage);
```

This will compute a digest of our message stored inside a variable *ourMessage*, and store it in a byte array named *hashValue*. Instead of computing the digest of some text stored in a variable, we can compute the digest of a file on the disk, as follows:

```
FileStream inputFile = new FileStream ("C:\\atul\\myFile.txt", FileMode.Open);
SHA1 sha1 = SHA1.Create ();
byte [ ] hashValue = sha1.ComputeHash (inputFile);
inputFile.Close ();
```

■ 8.4 CRYPTOGRAPHIC TOOLKITS ■

Apart from the cryptographic solutions offered by companies such as Sun and Microsoft, there are a number of companies, which specialize in providing **cryptographic toolkits**, which can be used for developing cryptographic solutions. Conceptually, these cryptographic toolkits are pretty similar to the JCA/JCE or MS-CAPI offerings. This means that a toolkit would provide mechanisms for encryption, decryption, digital signatures, etc., in the form of APIs. However, the big difference is that since these companies focus solely on cryptography, their offerings are much more solid and proven. Examples of such companies are RSA Data Security Inc., Entrust, and Baltimore, etc. In India, a company called Odyssey also offers a toolkit. A typical cryptographic application built using such toolkits can be depicted as shown in Fig. 8.10.

Conceptually, this is similar to how JCA/JCE or .NET cryptography work. An application requiring cryptographic functionalities would call a toolkit-independent layer (similar to the engine classes), which, in turn, calls the toolkit layer. The middle layer is required to ensure that the applications are toolkit-independent. Thus, an application requiring encryption would need to call a generic method of the middle layer, such as *encrypt*. This method, in turn, would detect the toolkit in use, and call that toolkit's specific *encrypt* method. Since different toolkits provide the cryptographic functionalities differently both in terms of their external interfaces and internal im-

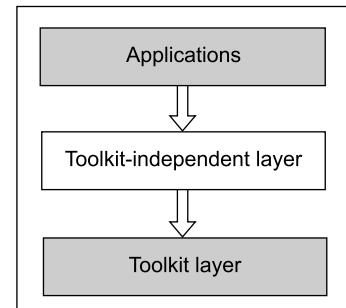


Fig. 8.10 Using cryptographic toolkits

plementations, the middle layer is required to ensure interoperability. If all toolkits adhered to a single, uniform standard, the middle layer would have been redundant.

However, everything is not so straightforward. Toolkits no doubt provide a very solid cryptographic infrastructure, but this does not come without problems.

- Using toolkits requires both client-and server-side licenses, which can be quite expensive (thousands of dollars).
- Also, how do we provide licenses to browser-based clients? After all, theoretically, any person in the world is a potential browser-based client!
- Toolkit-interoperability is also a major issue, because of which the middle layer is almost mandatory. Developing this middle layer can also be quite challenging, because of the interoperability issues involved therein.

We shall study how to deal with such situations in our case study separately in Appendix G.

■ 8.5 WEB SERVICES SECURITY ■

There are two entities involved in Web Services Communication, namely the Web Services Provider (Server) and Consumer (Client). The client uses Web Services information available in the form of a WSDL file to invoke the various Web Services. This means that unauthorized users can use Web Service. This does not stop here; unauthorized users may get access to the resources illegally. To prevent this, we require Web Services Security.

(a) Message Integrity Ensuring that messages have not been tampered.

(b) Non-Repudiation Ensuring that parties involved in the communication cannot deny the authenticity of their signature and the fact that they have originated a message.

(c) Authenticity/Identity Management To determine the identity of the clients to prevent illegal access.

(d) Authorization Determining the access rights once the identity is established.

(e) Confidentiality Protecting information during transmission.

What is WS-Security?

WS-Security is security standard that addresses security concerns when data is exchanged as a part of Web Services. WS-Security is a series of specifications that originated from vendors such as IBM, Microsoft, VeriSign, etc. WS-Security specification is an activity of *Web Services Interoperability Organization (WS-I Organization)* which is an industry-wide effort in standardization of Web Services Security.

WS-Security specifies two elements:

1. Enhancement to SOAP (Standard Object Access Protocol) to protect a message while in transit
2. Associating a security token with SOAP messages for authentication and authorization purposes

WS-Security is a series of specifications for digital signatures, encryption, XML signature, XML encryption, security tokens and foundation for various other standards. The diagram shows security standards overview which shows clear distinction between transport-level and message-level security elements.

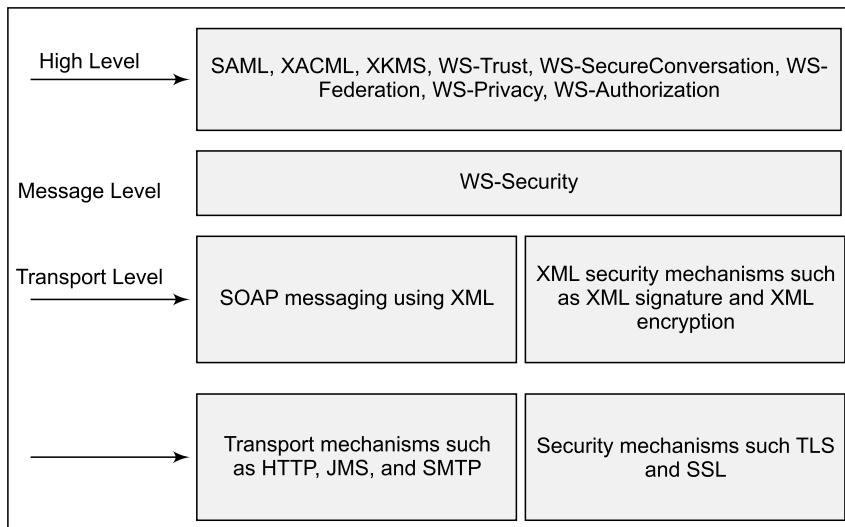


Fig. 8.11 Security Standards Overview

There are some primary security specifications as follows:

(a) XML Signature To ensure the *integrity* of an XML message. It is about how to compute, store, and verify digital signature of an entire XML document or the parts of an XML document.

(b) XML Encryption To ensure *confidentiality* of an XML message. Encryption of an entire XML message or its parts, maintaining the XML syntax. Encryption makes it very difficult to read the document. XML Signature and XML Encryption can be used simultaneously on the XML, i.e. on SOAP message.

(c) SAML Stands for *Security Assertion Markup Language*. It defines format and protocol to exchange identities. Used for interoperable and loosely coupled identity management.

(d) XACML Stands for *eXtensible Access Control Markup Language*. It defines format for authorization and access management.

WS-Security specification provides support for multiple signatures technologies, multiple encryption technologies and identity and access management using various security solutions such as security tokens.

Following are some of the security tokens supported by the WS-Security for authorization and authentication.

(i) Simple Tokens Username/clear password or Username/Password Digest.

(ii) Binary Tokens Such as X.509 certificates and Kerberos.

(iii) **XML Tokens** SAML assertions, XrML (eXtensible Rights Markup Language), and XCBF (XML Common Biometric Format).

When we apply these standards on a SOAP message, it would look as follows:

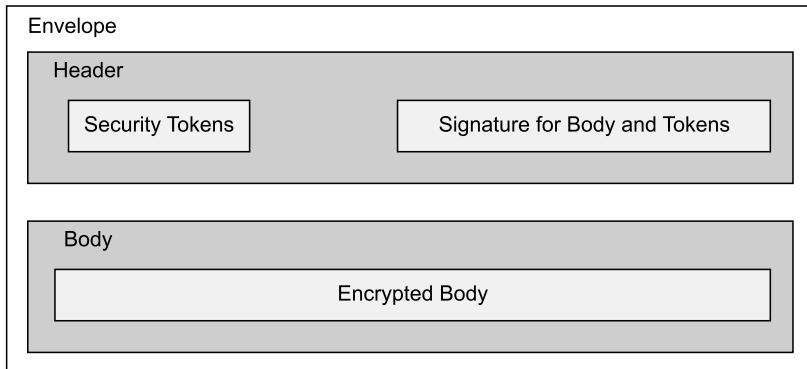


Fig. 8.12 Secured SOAP message

Let us have a brief look at WS-Security specification.

(a) WS-SecureConversation Specification that provides secure communication between Web Services using session keys. WS-SecureConversation works by defining and implementing a session key (encryption key) to be shared among all in a communication session. Session key is used to prevent outsiders from entering into the active communication.

(b) WS-Trust It defines standard interfaces for tasks such as security token creation, its management and exchange, distribution of credentials within different trust domains, etc.

(c) WS-Federation It helps in standardizing the disparate identities and identity-mechanism mechanisms. This specification helps in identity sharing, authorization, authentication, etc. This is true even if different mechanisms are used in identity management.

(d) WS-Authorization It defines specifications for authorization and works in conjunction with WS-SecureConversation and WS-Federation.

(e) WS-Privacy This specification describes model to define privacy policies about a Web Service which can be embedded into WS-Policy. WS-Trust can be used to evaluate user preferences and Web Service's privacy claims in order to obtain access to the proper users/systems.

■ 8.6 CLOUD SECURITY ■

The term **cloud computing** indicates *virtual servers* available on the Internet. That is, it is the usage of computing resources (hardware, software or both) that are not owned by the application owners, but are hosted in data centers owned by a third-party provider, in a consolidated manner. This frees the application owners of the computing resources from worrying about the underlying technology and implementation details. They can ask for more or less resources on demand, and it is the responsibility of the third-party provider to dynamically adjust to these requirements. Thus, application owners can

just concentrate on the application logic, and do not have to keep scaling up or down in terms of hardware infrastructure, personnel, or software licensing. Instead, they can *pay per use* to the third-party provider.

The term is used for referring to a platform and a type of application. Cloud computing makes the use of servers (physical or virtual) dynamically when the need arises. A cloud-computing platform dynamically makes provisions, configurations, reconfigurations, and de-provisions servers as and when needed.

The subject of **cloud security** is still evolving, since cloud computing itself is still evolving. Collectively, a set of policies, technologies, and approaches that are needed to protect data, applications and cloud infrastructure is the area of cloud security. At a very high level, cloud security involves two parties: the cloud provider/host and the cloud client/user. The cloud provider/host can provide the cloud platform as an infrastructure, as a service, or as an application. Security is needed in each case. Client's data and applications must be secure in cloud computing, especially because the client relies on the cloud provider much more than in a non-cloud application. Consequently, cloud security must deal with all aspects of securing data access, storage, application hosting and storage, user information, and authentication.

One important characteristic of cloud computing is the excessive use of virtualization. In other words, a layer of abstraction is created between the real hardware/network and what the user perceives as the real hardware/network. Hence, it becomes even more important to protect this virtual layer of abstraction.

In general, cloud security deals with identity management, physical and personal security, availability of data and applications, application security, and confidentiality. In addition, many national/international regulations mandate specific security measures to be applied for compliance reasons. This makes cloud security even more significant.



Summary

- Java cryptographic solutions are based on Java Cryptography Architecture (JCA) and Java Cryptography Extensions (JCE).
- JCA keeps the interface and implementation separate.
- JCA provides for plug-able architecture.
- JCA consists of engine classes and provider classes.
- JCE was created separately from JCA in order to abide by the US export rules for cryptographic software.
- JCE required licensing earlier, but not any more.
- .NET also provides security features similar to that of Java.
- .NET security has architecture similar to that of JCE.
- .NET security uses a three-layered class model for ease of modification and extension.
- Cryptographic toolkits can also be used for cryptography.

- Cryptographic toolkits are very solid and proven, but can be expensive.
- Operating systems can be monolithic, layered, or microkernel-based. The last one is possibly the most secure.
- Database control can be classified into two types: discretionary control and mandatory control.
- In discretionary control, the users of the database system have access rights, also called privileges.
- In mandatory control, each database object (e.g. table) has a classification level (e.g. top secret, secret, confidential, sensitive, unclassified), and each database user has a clearance level (e.g. top secret, secret, confidential, sensitive, unclassified).
- Database privileges can be divided into system privileges and object privileges.
- System privileges are related to the access of the database. They govern things such as the permission to connect to the database, the right to create tables and other objects, and database administration permissions.
- Object privileges are focused on a particular database object in question, e.g. a table or view, for instance.
- SQL provides rich features for database control and privilege enforcement.
- Privileges can be granted to or revoked from a database user.
- Privileges can apply to tables, columns, indexes, references, etc.
- Statistical databases can protect database information to a certain extent. They contain only summary information.



Key Terms and Concepts

- | | |
|--|---|
| <ul style="list-style-type: none">● Access rights● Clearance level● Database control● Engine classes● Interface● Java Cryptography Extensions (JCE)● Mandatory control● Monolithic operating systems● Privileges● Sequence guessing● Spoofing● SYN flooding | <ul style="list-style-type: none">● Classification level● Cryptographic toolkit● Discretionary control● Implementation● Java Cryptography Architecture (JCA)● Layered operating systems● Microkernel operating systems● Object privileges● Provider● Session hijacking● Statistical databases● System privileges |
|--|---|



PRACTICE SET

■ Multiple-Choice Questions

1. Java cryptography mechanisms are in the form of _____ and _____.
 - (a) JCP, JCA
 - (b) JCA, JCB
 - (c) JCA, JCE
 - (d) JCE, JCF
2. Out of JCA and JCE, _____ need(s) licensing.
 - (a) only JCA
 - (b) only JCE
 - (c) both JCA and JCE
 - (d) neither JCA nor JCE
3. JCA architecture is _____.
 - (a) static
 - (b) implementation-dependent
 - (c) not flexible
 - (d) plug-able
4. Digital signatures are a part of _____.
 - (a) only JCA
 - (b) only JCE
 - (c) both JCA and JCE
 - (d) neither JCA nor JCE
5. NET security architecture is _____ JCE.
 - (a) very different from
 - (b) quite similar to
 - (c) attached to
 - (d) none of the above
6. In .NET security framework, the _____ class is at the highest level.
 - (a) concrete
 - (b) abstract base
 - (c) abstract algorithm
 - (d) algorithm implementation
7. In the _____ attack, the attacker keeps on sending connection requests to the other side.
 - (a) SYN flooding
 - (b) spoofing
8. We can create a digital certificate using the Java _____.
 - (a) keytool
 - (b) keyalg
 - (c) JCA
 - (d) JCE
9. Web Services security standards are collectively known as _____.
 - (a) Sec
 - (b) WS-Sec
 - (c) WS-Security
 - (d) Web-Security
10. In Web services, message integrity is ensured by _____.
 - (a) digital signature
 - (b) message digest
 - (c) message signature
 - (d) XML signature
11. In Web services, confidentiality is ensured by _____.
 - (a) encryption
 - (b) decryption
 - (c) XML encryption
 - (d) XML cryptography
12. The _____ protocol is used to exchange identities.
 - (a) XML
 - (b) SAML
 - (c) WSDL
 - (d) SOAP
13. Security token creation is described in _____.
 - (a) WS-Federation
 - (b) WS-Authentication
 - (c) WS-Privacy
 - (d) WS-Trust

14. The privacy of a Web service is ensured by _____.
(a) WS-Federation
(b) WS-Authentication
(c) WS-Privacy
(d) WS-Trust
15. Modern applications demand _____.
(a) client security
(b) server security
(c) communication security
(d) cloud security

■ Exercises

1. Discuss JCE and JCA in brief.
2. What is the meaning of the term “provider architecture” in JCA?
3. What are engine classes in JCA?
4. What is the reason behind separating JCA and JCE?
5. Why is MS-CAPI likely to become popular?
6. What are the key aspects to be considered while using a cryptographic toolkit?
7. What is a Web service?
8. What is Web services security?
9. What are XML signatures?
10. What is XML encryption?
11. How is message transfer during a Web service call protected?
12. How are Web services themselves protected?
13. What is WS-Authentication?
14. What is cloud security?
15. Why is cloud security important?

■ Design/Programming Exercises

1. Key generation (public and private key pair) can be performed using Java. Write a program which can do this.
2. Write a program in Java, which performs a digital signature on a given text.
3. Write a C# program to perform cryptographic operations, such as digital signatures and encryption.
4. Implement the PKCS#5 standard of Password Based Encryption (PBE) using Java cryptography.
5. Sign some text in Java using RSA algorithm first, followed by DSA algorithm. Do you notice any difference while verifying the signatures?
6. Encrypt some text using AES algorithm in both Java and .NET.
7. Instead of some text, encrypt a file and verify that it can be decrypted using AES in Java and .NET.
8. What are the additional security features in J2EE as compared to plain Java?
9. Create a Web service using Java.
10. Implement SSL on a Java Web service.
11. Where are the security parameters passed in a Web service call? Investigate.
12. Study more about SAML and where it is used.
13. Write a utility in Java which can take a given private key, the corresponding public key certificate and a message and sign and verify that message.
14. Write a utility in Java which can take a given private key, the corresponding public-key certificate and a message and encrypt and decrypt that message using RSA algorithm.
15. What are the trends in cloud security related to user’s data encryption? Study them.



NETWORK SECURITY, FIREWALLS, AND VIRTUAL PRIVATE NETWORKS (VPN)

■ 9.1 INTRODUCTION ■

Network-layer security is a key aspect of the Internet-based security mechanisms. Originally, people were concentrating only on application level security. However, new security requirements demand that even the lower level data units should be protected. With this view in mind, network-security mechanisms have emerged and are being used quite extensively in real life.

This chapter discusses TCP/IP protocol suite in brief, with specific attention to the IP and TCP protocols. This is important for understanding the concepts in network security. We then examine the technology of firewalls. Firewalls are widely used by organizations to protect their internal networks from outside attacks. The chapter discusses the various types organizations and architectures of firewalls. It also takes a look at the various issues involved therein.

The chapter then focuses on network level security, with a thorough description of the IPSec protocol. IPSec protocol has many sub-protocols. We discuss them in great detail. Finally, we study what a Virtual Private Network (VPN) means and how it works.

■ 9.2 BRIEF INTRODUCTION TO TCP/IP ■

9.2.1 Basic Concepts

As we know, the Internet is based on the Transmission Control Protocol/Internet protocol (TCP/IP) protocol suite. It is very important to know the basics of TCP/IP before we study how we can provide security at the network layer. We have already discussed this in brief earlier. However, in the context of network level security, a comprehensive understanding of TCP and IP is mandatory.

Let us first take a look at the various layers in the TCP/IP protocol suite. As we have studied before, the TCP/IP protocol suite contains five main layers: application, transport, network (or Internet), data link and physical. Unlike the OSI protocol suite, there are no presentation and session layers in TCP/IP. The various layers and protocols in the TCP/IP suite are shown in Fig. 9.1. Note that the presentation and session layers are not present in TCP/IP, but are shown for the comparison of TCP/IP with the OSI model.

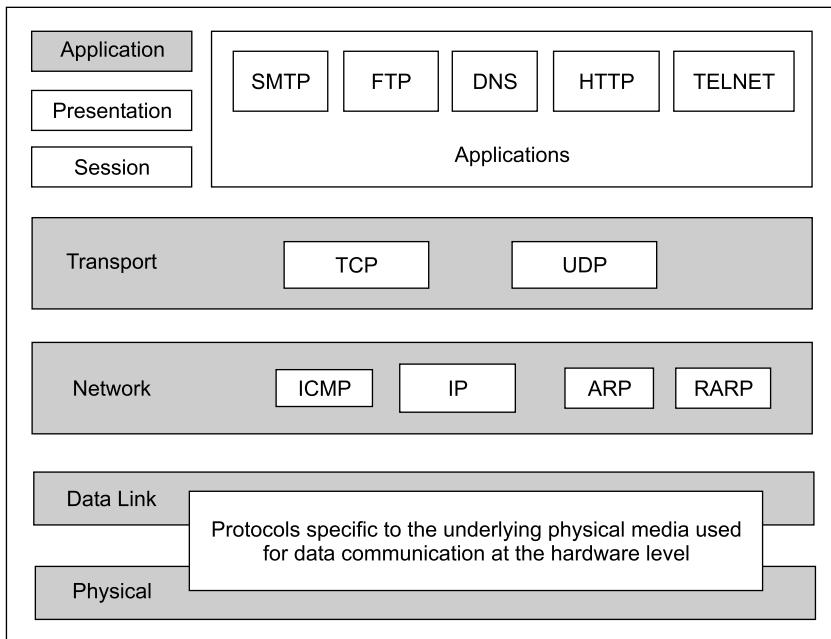


Fig. 9.1 Layers in the TCP/IP protocol suite

Let us now think about the significance of these layers. Figure 9.2 shows the idea. The data unit initially created at the application layer (i.e. by an application, such as email, Web browser, etc.) is called a *message*. A message is actually broken down into *segments* by the transport layer, which is not shown here. Note that the transport layer of TCP/IP contains two protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is more often used, therefore, we shall concentrate on it. For the scope of our current text, the same discussion would also apply to UDP. The transport layer then adds its own header to the segment and gives it to the network layer. The network layer adds the IP header to this block and gives the result to the data-link layer. The data-link layer adds the frame header and gives it to the physical layer for transmission. At the physical layer, the actual bits are transmitted as voltage pulses. An opposite process happens at the destination end, where each layer removes the previous layer's header and finally the application layer receives the original message.

9.2.2 TCP Segment Format

We must now examine what we mean by saying that the transport (TCP), network (IP) and data link (frame) layers add headers to the received data block. For instance, when the transport layer adds the TCP header to the original message, it not only appends the header fields to the original message, but

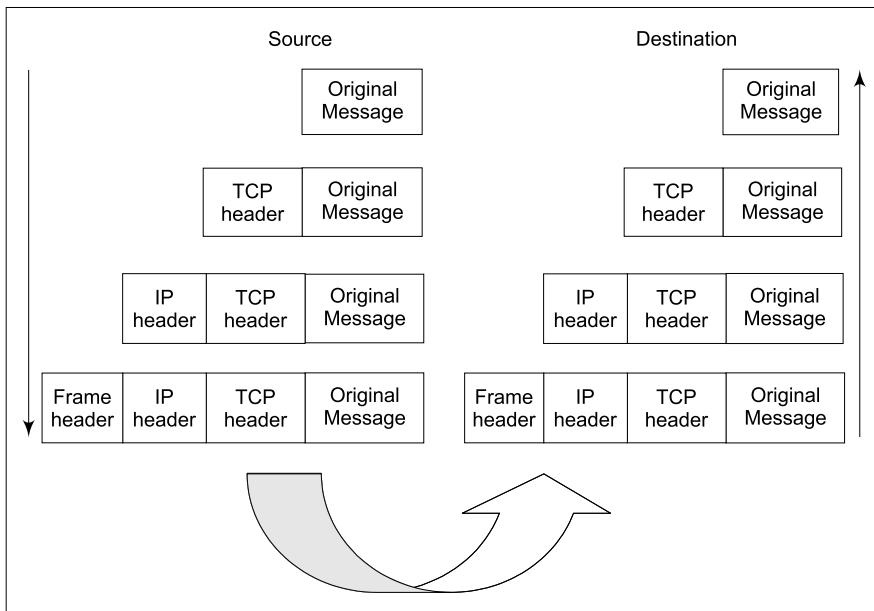


Fig. 9.2 Message transfer from the source to the destination at different TCP/IP layers

also performs some processing, such as calculating the checksum for error detection, etc. After the headers are added, a TCP segment looks as shown in Fig. 9.3. As we can see, a TCP segment consists of a header of size 20 to 60 bytes, followed by the actual data. The header consists of 20 bytes if the TCP packet does not contain any options. Otherwise, the header consists of 60 bytes. That is, a maximum of 40 bytes are reserved for options. Options can be used to convey additional information to the destination. However, we shall ignore them, as they are not very frequently used.

Let us briefly discuss the header fields inside a TCP segment.

(a) Source Port Number This 2-byte number signifies the port number of the source computer, corresponding to the application that is sending this TCP segment.

(b) Destination Port Number This 2-byte number signifies the port number of the destination computer, corresponding to the application that is expected to receive this TCP segment.

(c) Sequence Number This 4-byte field defines the number assigned to the first byte of the data portion contained in this TCP segment. TCP is a connection-oriented protocol. For ensuring a correct delivery, each byte to be transmitted from the source to the destination is numbered in an increasing sequence. The sequence number field tells the destination host, which byte in this sequence comprises the first byte of the TCP segment. During the TCP connection establishment phase, both the source as well as the destination generate different unique random numbers. For instance, if this random number is 3130 and the first TCP packet is carrying 2000 bytes of data, then the sequence number field for that packet would contain 3132 (bytes 3130 and 3131 are used in connection establishment). The second segment would then have a sequence number of 5132 (3132 + 2000), and so on.

(d) Acknowledgement Number If the destination host receives a segment with sequence number X correctly, it sends $X + 1$ as the acknowledgement number back to the source. Thus, this 4-byte

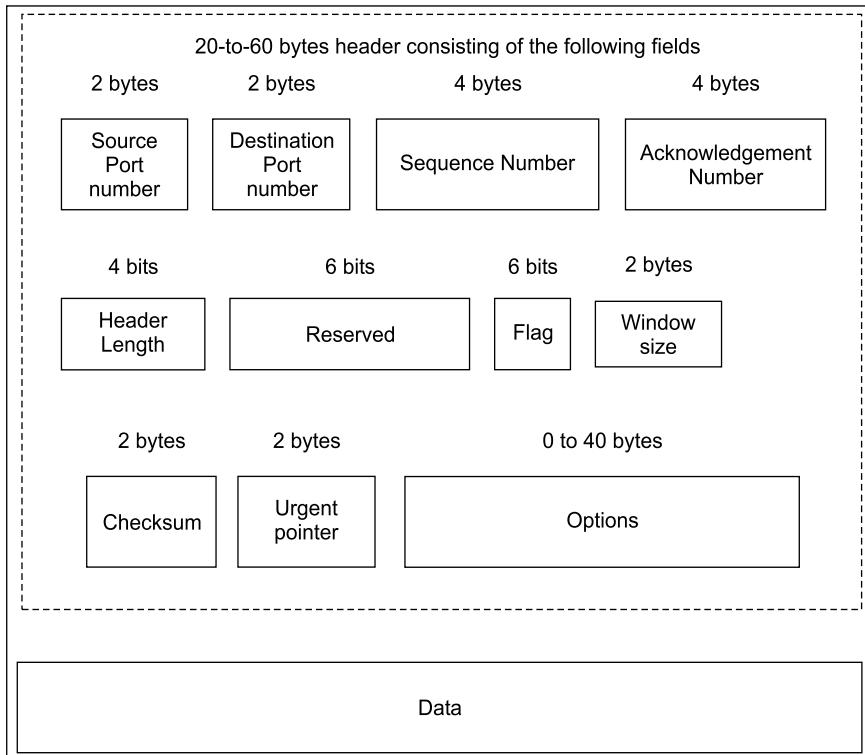


Fig. 9.3 TCP segment format

number defines the sequence number that the source is expecting from the destination as a receipt of the correct delivery.

(e) Header Length This 4-bit field specifies the number of four-byte words in the TCP header. As we know, the header length can be between 20 and 60 bytes. Therefore, the value of this field can be between 5 (because $5 \times 4 = 20$) and 15 (because $15 \times 4 = 60$).

(f) Reserved This 6-byte field is reserved for future use and is currently unused.

(g) Flag This 6-bit field defines six different control flags, each one of them occupying one bit. Out of the six flags, two are most important. The SYN flag indicates that the source wants to establish a connection with the destination. Therefore, this flag is used when a TCP connection is being established between two hosts. Similarly, the other flag of importance is the FIN flag. If the bit corresponding to this flag is set then it means that the sender wants to terminate the current TCP connection.

(h) Window Size This field determines the size of the sliding window that the other party must maintain.

(i) Checksum This 16-bit field contains the checksum for facilitating the error detection and correction.

(j) Urgent Pointer This field is used in situations where data in a TCP segment is more important or urgent than other data in the same TCP connection. However, a discussion of such situations is beyond the scope of the current text.

9.2.3 IP Datagram Format

The TCP header plus the original message is now passed to the IP layer. The IP layer treats this whole package of TCP header + original message as its *original message* and adds its own header to it. This results into the creation of an IP datagram. The format of an IP datagram is shown in Fig. 9.4.

Version (4 bits)	HLEN (4 bits)	Service Type (8 bits)	Total Length (16 bits)	
Identification (16 bits)		Flags (3 bits)	Fragmentation Offset (13 bits)	
Time to live (8 bits)	Protocol (8 bits)	Header Checksum (16 bits)		
Source IP address (32 bits)				
Destination IP address (32 bits)				
Data				
Options				

Fig. 9.4 IP datagram

An IP datagram is a variable-length datagram. A message can be broken down into multiple datagrams and a datagram in turn can be fragmented into different fragments, as we shall see. A datagram can contain a maximum of 65,536 bytes. A datagram is made up of two main parts: the header and the data. The header consists of anywhere between 20 and 60 bytes and essentially contains information about routing and delivery. The data portion contains the actual data to be sent to the recipient. The header is like an envelope: it contains information *about* the data. The data is analogous to the letter *inside* the envelope. Let us examine the fields of a datagram in brief.

(a) Version This field currently contains a value 4, which indicates **IP version 4 (IPv4)**. In future, this field would contain 6 when **IP version 6 (IPv6)** becomes the standard.

(b) Header Length (HLEN) Indicates the size of the header in a multiple of four-byte words. When the header size is 20 bytes as shown in the figure, the value of this field is 5 (because $5 \times 4 = 20$) and when the option field is at the maximum size, the value of HLEN is 15 (because $15 \times 4 = 60$).

(c) Service Type This field is used to define service parameters such as the priority of the datagram and the level of reliability desired.

(d) Total Length This field contains the total length of the IP datagram. Because it is two bytes long, an IP datagram cannot be more than 65,536 bytes ($2^{16} = 65,536$).

(e) Identification This field is used in the situations when a datagram is fragmented. As a datagram passes through different networks, it might be fragmented into smaller sub-datagrams to match the physical datagram size of the underlying network. In these situations, the sub-datagrams are sequenced using the identification field, so that the original datagram can be reconstructed from them.

(f) Flags This field corresponds to the earlier field (identification). It indicates whether a datagram can be fragmented in the first place—and if it can be fragmented, whether it is the first or the last fragment or it can be a middle fragment, etc.

(g) Fragmentation Offset If a datagram is fragmented, this field is useful. It is a pointer that indicates the offset of the data in the original datagram before fragmentation. This is useful when reconstructing a datagram from its fragments.

(h) Time to Live We know that a datagram travels through one or more routers before reaching its final destination. In the case of network problems, some of the routes to the final destination may not be available because of many reasons such as hardware failure, link failure or congestion. In that case, the datagram may be sent through a different route. This can continue for a long time if the network problems are not resolved quickly. Soon, there could be many datagrams traveling in different directions through lengthy paths, trying to reach their destinations. This can create congestion and the routers may become too busy, thus bringing at least parts of the Internet to a virtual halt. In some cases, the datagrams can continue to travel in a loop in between, without reaching the final destination and in fact, coming back to the original sender. To avoid this, the datagram sender initializes this field (that is, *Time to live*) to some number. As the datagram travels through routers, this field is decremented each time. If the value in this field becomes zero or negative, it is immediately discarded. No attempt is made to forward it to the next hop. This avoids a datagram traveling for an infinite amount of time through various routers and therefore, helps avoid network congestion. After all the other datagrams have reached the destination, the TCP protocol operating at the destination will find out this missing datagram and will have to request for its retransmission. Thus, IP is not responsible for the error-free, timely and in-sequence delivery of the entire message – it is done by TCP.

(i) Protocol This field identifies the transport protocol running on top of IP. After the datagram is constructed from its fragments, it has to be passed on to the upper layer software piece. This could be TCP or UDP. This field specifies which piece of software at the destination node the datagram should be passed on to.

(j) Source Address This field contains the 32-bit IP address of the sender.

(k) Destination Address This field contains the 32-bit IP address of the final destination.

(l) Options This field contains optional information such as routing details, timing, management and alignment. For instance, it can store the information about the exact route that the datagram has taken. When it passes through a router, the router puts in its id and optionally, also the time when it passed through that router, in one of the slots in this field. This helps tracing and fault detection of datagrams. However, most of the time, the space in this field is not sufficient for all these details, therefore, it is not used very often.

This brief introduction to TCP/IP would suffice for the scope of the current text.

■ 9.3 FIREWALLS ■

9.3.1 Introduction

The dramatic rise and progress of the Internet has opened possibilities that no one would have thought of. We can connect any computer in the world to any other computer, no matter how far the two are

located from each other. This is undoubtedly a great advantage for individuals and corporate as well. However, this can be a nightmare for network support staff, which is left with a very difficult job of trying to protect the corporate networks from a variety of attacks. At a broad level, there are two kinds of attacks:

- Most corporations have large amounts of valuable and confidential data in their networks. Leaking of this critical information to competitors can be a great setback.
- Apart from the danger of the insider information leaking out, there is a great danger of the outside elements (such as viruses and worms) entering a corporate network to create havoc.

We can depict this situation as shown in Fig. 9.5.

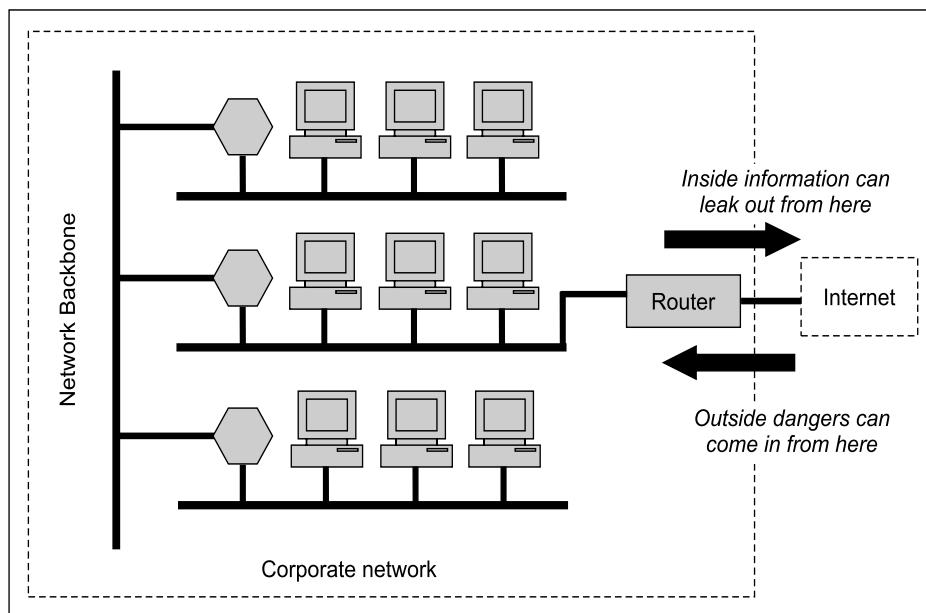


Fig. 9.5 Threats from inside and outside a corporate network

As a result of these dangers, we must have mechanisms which can ensure that the inside information remains inside and also prevent the outsider attackers from entering inside a corporate network. As we know, encryption of information (if implemented properly) renders its transmission to the outside world redundant. That is, even if confidential information flows out of a corporate network, if it is in encrypted form, outsiders cannot make any sense of it. However, encryption does not work in the other direction. Outside attackers can still try to break inside a corporate network. Consequently, better schemes are desired to achieve protection from outside attacks. This is where a **firewall** comes into picture.

Conceptually, a firewall can be compared with a sentry standing outside an important person's house (such as the nation's president). This sentry usually keeps an eye on and physically checks every person that enters into or comes out of the house. If the sentry senses that a person wishing to enter the president's house is carrying a knife, the sentry would not allow the person to enter. Similarly, even

if the person does not possess any banned objects, but somehow looks suspicious, the sentry can still prevent that person's entry.

A firewall acts like a sentry. If implemented, it guards a corporate network by standing between the network and the outside world. All traffic between the network and the Internet in either direction must pass through the firewall. The firewall decides if the traffic can be allowed to flow or whether it must be stopped from proceeding further. This is shown in Fig. 9.6.

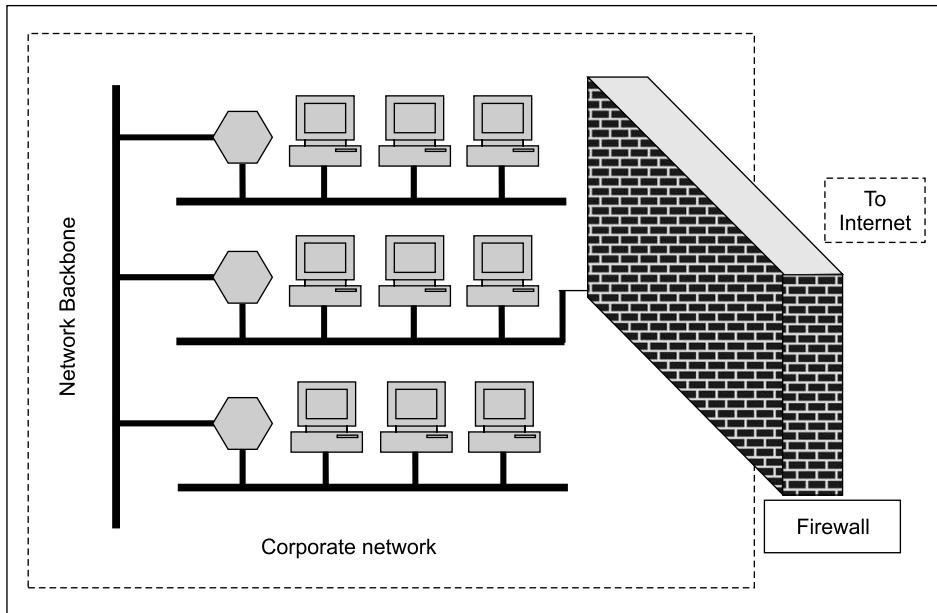


Fig. 9.6 Firewall

Of course, technically, a firewall is a specialized version of a router. Apart from the basic routing functions and rules, a router can be configured to perform the firewall functionality, with the help of additional software resources.

The characteristics of a good firewall implementation can be described as follows.

- All traffic from inside to outside and vice versa, must pass through the firewall. To achieve this, all the access to the local network must first be physically blocked and access only via the firewall should be permitted.
- Only the traffic authorized as per the local security policy should be allowed to pass through.
- The firewall itself must be strong enough, so as to render attacks on it useless.

9.3.2 Types of Firewalls

Based on the criteria that they use for filtering traffic, firewalls are generally classified into two types, as shown in Fig. 9.7.

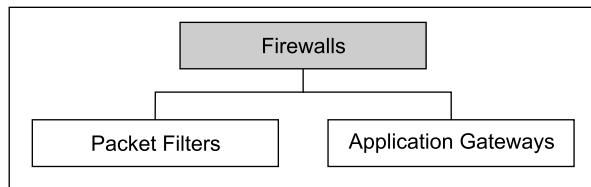


Fig. 9.7 Types of firewalls

Let us discuss these two types of firewalls one by one.

1. Packet Filters

As the name suggests, a **packet filter** applies a set of rules to each packet and based on the outcome, decides to either forward or discard the packet. It is also called as **screening router** or **screening filter**. Such a firewall implementation involves a router, which is configured to filter packets going in either direction (from the local network to the outside world and vice versa). The filtering rules are based on a number of fields in the IP and TCP/UDP headers, such as source and destination IP addresses, IP protocol field (which identifies if the protocol in the upper transport layer is TCP or UDP), TCP/UDP port numbers (which identify the application which is using this packet, such as email, file transfer or World Wide Web).

The idea of a packet filter is shown in Fig. 9.8.

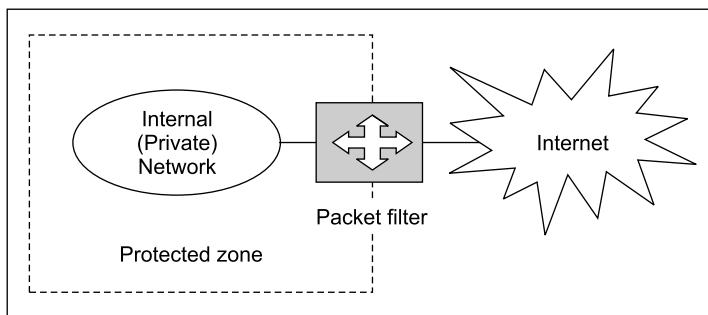


Fig. 9.8 Packet filter

Conceptually, a packet filter can be considered as a router that performs three main actions, as shown in Fig. 9.9.

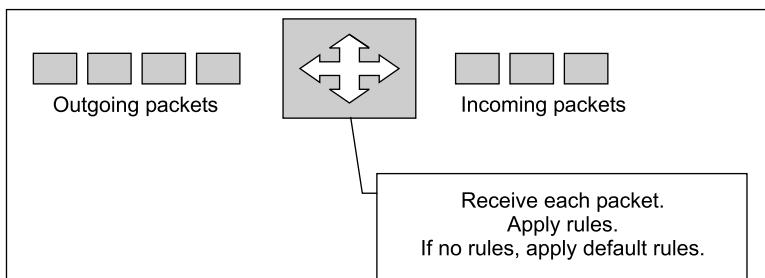


Fig. 9.9 Packet filter operation

A packet filter performs the following functions.

- Receive each packet as it arrives.
- Pass the packet through a set of rules, based on the contents of the IP and transport header fields of the packet. If there is a match with one of the set rules, decide whether to accept or discard the packet based on that rule. For example, a rule could specify: disallow all incoming traffic from an IP address 157.29.19.10 (this IP address is taken just as an example) or disallow all traffic that uses UDP as the higher (transport) layer protocol.
- If there is no match with any rule, take the default action. The default can be *discard all packets* or *accept all packets*. The former policy is more conservative, whereas the latter is more open. Usually, the implementation of a firewall begins with the default *discard all packets* option and then rules are applied one-by-one to enforce packet filtering.

The chief advantage of the packet filter is its simplicity. The users need not be aware of a packet filter at all. Packet filters are very fast in their operating speed. However, the two disadvantages of a packet filter are the difficulties in setting up the packet filter rules correctly and lack of support for authentication.

Figure 9.10 shows an example where a router can be converted into a packet filter by adding the filtering rules in the form of a table. This table decides which of the packets should be allowed (forwarded) or discarded.

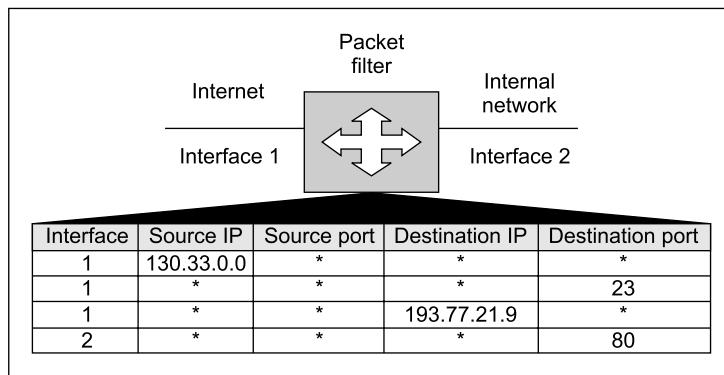


Fig. 9.10 Example of packet filter table

The rules specified in the packet filter work as follows:

- Incoming packets from network 130.33.0.0 are not allowed. They are blocked as a security precaution.
- Incoming packets from any external network on the TELNET server port (number 23) are blocked.
- Incoming packets intended for a specific internal host 193.77.21.9 are blocked.
- Outgoing packets intended for port 80 (HTTP) are banned. That is, this organization does not want to allow its employees to send requests to the external world (i.e. the Internet) for browsing the Internet.

Attackers can try and break the security of a packet filter by using the following techniques.

(a) IP Address Spoofing An intruder outside the corporate network can attempt to send a packet towards the internal corporate network, with the source IP address set equal to one of the IP addresses of the internal users. This is shown in Fig. 9.11. This attack can be defeated by discarding all the packets that arrive at the incoming side of the firewall, with the source address equal to one of the internal addresses.

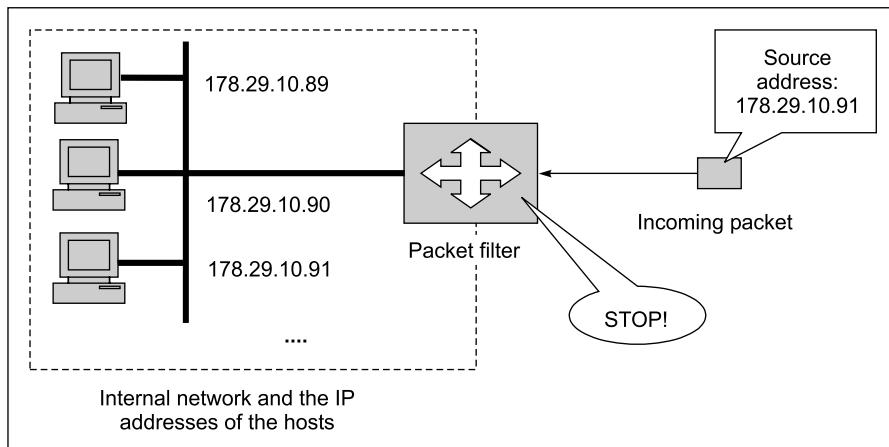


Fig. 9.11 Packet filter defeating the IP address spoofing attack

(b) Source Routing Attacks An attacker can specify the route that a packet should take as it moves along the Internet. The attacker hopes that by specifying this option, the packet filter can be fooled to bypass its normal checks. Discarding all packets that use this option can thwart such an attack.

(c) Tiny Fragment Attacks IP packets pass through a variety of physical networks, such as Ethernet, Token Ring, X.25, Frame Relay, ATM, etc. All these networks have a pre-defined maximum frame size (called as the Maximum Transmission Unit or MTU). Many times, the size of the IP packet is greater than this maximum size allowed by the underlying network. In such cases, the IP packet needs to be fragmented, so that it can be accommodated inside the physical frame and carried further. An attacker might attempt to use this characteristic of the TCP/IP protocol suite by intentionally creating fragments of the original IP packet and sending them. The attacker feels that the packet filter can be fooled, so that after fragmentation, it checks only the first fragment and does not check the remaining fragments. This attack can be foiled by discarding all the packets where the (upper layer) protocol type is TCP and the packet is fragmented (refer to *identification* and *protocol* fields of an IP packet discussed earlier to understand how we can implement this).

An advanced type of packet filter is called as **dynamic packet filter** or **stateful packet filter**. A dynamic packet filter allows the examination of packets based on the current state of the network. That is, it adapts itself to the current exchange of information, unlike the normal packet filters, which have routing rules hard coded. For instance, we can specify a rule with the help of a dynamic packet filter as follows:

Allow incoming TCP packets only if they are responses to the outgoing TCP packets that have gone through our network.

Note that the dynamic packet filter has to maintain a list of the currently open connections and outgoing packets in order to deal with this rule. Hence, it is called as *dynamic* or *stateful*. When such a rule is in effect, the logical view of the packet filtering can be illustrated as shown in Fig. 9.12.

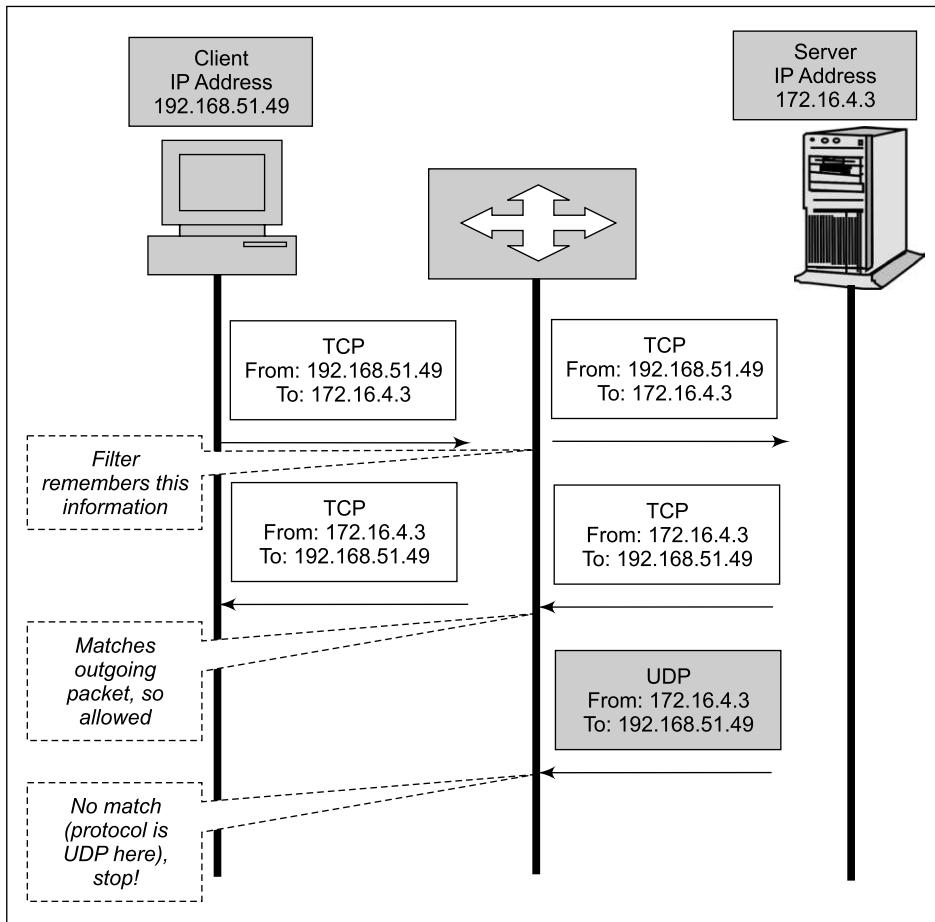


Fig. 9.12 Dynamic packet filter technology

As shown in the figure, firstly, an internal client sends a TCP packet to an external server, which the dynamic packet filter allows. In response, the server sends back a TCP packet, which the packet filter examines and realizes that it is a response to the internal client's request. Therefore, it allows that packet in. However, next, the external server sends a new UDP packet, which the filter does not allow, because previously, the exchange of the client and the server packets happened using the TCP protocol. However, this packet is based on the UDP protocol. Since this is against the rule that was set up earlier, the filter drops the packet.

2. Application Gateways

An **application gateway** is also called a **proxy server**. This is because it acts like a proxy (i.e. deputy or substitute) and decides about the flow of application level traffic. The idea is shown in Fig. 9.13.

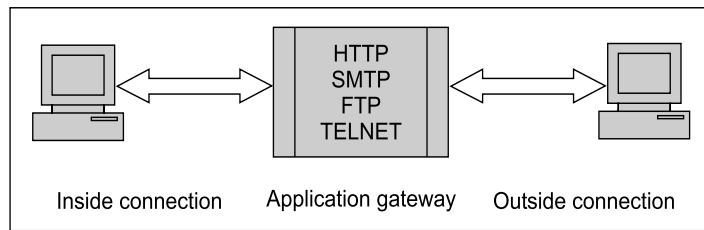


Fig. 9.13 Application gateway

Application gateways typically work as follows.

- An internal user contacts the application gateway using a TCP/IP application, such as HTTP or TELNET.
- The application gateway asks the user about the remote host with which the user wants to set up a connection for actual communication (i.e. its domain name or IP address, etc.). The application gateway also asks for the user id and the password required to access the services of the application gateway.
- The user provides this information to the application gateway.
- The application gateway now accesses the remote host on behalf of the user and passes the packets of the user to the remote host. Note that there is a variation of the application gateway, called as **circuit gateway**, which performs some additional functions as compared to those performed by an application gateway. A circuit gateway, in fact, creates a new connection between itself and the remote host. The user is not aware of this and thinks that there is a direct connection between itself and the remote host. Also, the circuit gateway changes the source IP address in the packets from the end user's IP address to its own. This way, the IP addresses of the computers of the internal users are hidden from the outside world. This is shown in Fig. 9.14. Of course, both the connections are shown with a single arrow to stress on the concept, in reality, both the connections are two-ways.

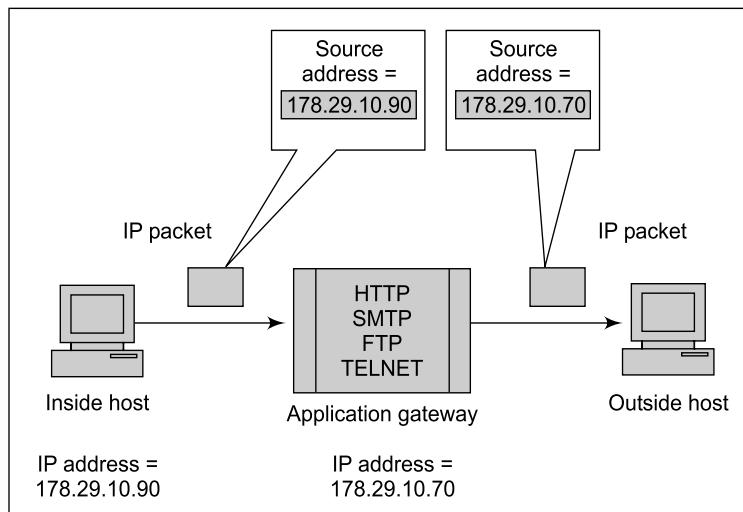


Fig. 9.14 Circuit-gateway operation

The SOCKS server is an example of the real-life implementation of a circuit gateway. It is a client-server application. The SOCKS client runs on the internal hosts and the SOCKS server runs on the firewall.

- (e) From here onwards, the application gateway acts like a proxy of the actual end user and delivers packets from the user to the remote host and vice versa.

Application gateways are generally more secure than packet filters, because rather than examining every packet against a number of rules, here we simply detect whether a user is allowed to work with a TCP/IP application or not. The disadvantage of application gateways is the overhead in terms of connections. As we noticed, there are actually two sets of connections now: one between the end user and the application gateway and another between the application gateway and the remote host. The application gateway has to manage these two sets of connections and the traffic going between them. This means that the actual communicating internal host is under an illusion, as illustrated in Fig. 9.15.

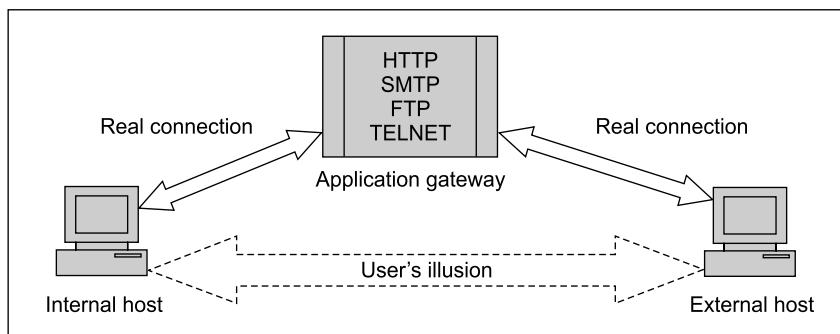


Fig. 9.15 Application gateway creates an illusion

An application gateway is also called **bastion host**. Usually, a bastion host is a very key point in the security of a network. How it functions is explained in more detail in the next section.

3. Network Address Translation (NAT)

One of the interesting jobs done by a firewall or proxy server is to perform **Network Address Translation (NAT)**. The number of people using the Internet from home, office or other places is increasing at a mind boggling rate. Earlier, users would access the Internet via an Internet Service Provider (ISP) for a short time and then disconnect. Thus, the ISP would have a set of IP addresses, from which it would dynamically allocate one IP address to every user for the duration the user was connected to the Internet. Once the user disconnected, the ISP would reallocate that same IP address to another user, who wanted to connect to the Internet now.

However, this situation changed dramatically as the number of people connecting to the Internet increased dramatically. Moreover, people started using the ADSL or cable connections to connect to the Internet, sing the *broadband* technology. Worse yet, people wanted multiple IP addresses for themselves, since they started creating small personal networks. This led to a serious problem of shortage of IP addresses.

NAT attempts to solve the problem of the shortage of IP addresses. NAT allows a user to have a large number of IP addresses internally, but only a single IP address externally. Only the external traffic needs the external address. The internal traffic can work with the internal addresses.

For NAT to be possible, the Internet authorities have specified that certain IP addresses must be used as only internal IP addresses. Others should be used only as external IP addresses. Thus, just by looking at an IP address, we can determine whether it is an internal or external IP address. Also, routers and hosts have no confusion, because of this classification. The internal (or private) IP addresses are listed in Fig. 9.16.

Any individual or organization can use any address within this range as an internal IP address, without needing to seek permission from anyone. Any address within this range is unique within that organization's network, but is obviously not unique outside of the organization's network. That does not matter, since the address is meant to be used in the context of, i.e. inside an organization's network anyway. Therefore, if a router receives a packet with destination address equal to an address that falls in one of the above the ranges, it does not forward it outside, since it knows that the address is internal.

When implemented in real life, a NAT configuration looks similar to what is shown in Fig. 9.17. As we can see, the router has two addresses: one external IP address and the other is an internal IP address. The external world (i.e. the rest of the Internet) knows the router based on the router's external address of 201.26.7.9, whereas the internal hosts refer to the router based on the router's internal address of 192.168.100.10. Also note how the internal hosts have internal IP addresses (192.168.x.x).

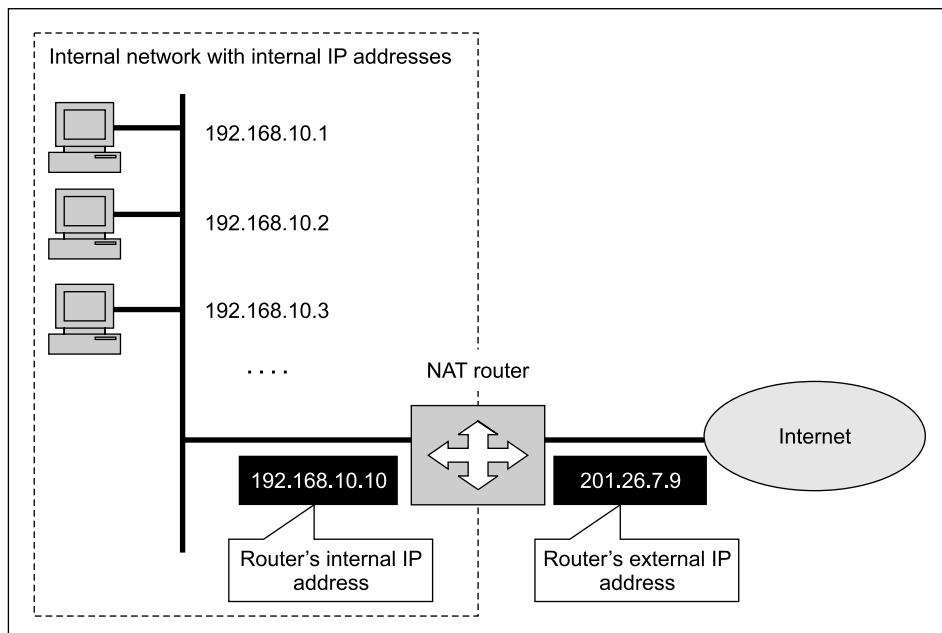


Fig. 9.17 NAT implementation example

This clearly means that the external world always sees only one IP address: the NAT router's external IP address. Thus:

Range of IP addresses		Total count
10.0.0.0	to	$10.255.255.255$
172.16.0.0	to	$172.31.255.255$
192.168.0.0	to	$192.168.255.255$

Fig. 9.16 Internal or private IP addresses

- For all *incoming* packets, regardless of which is actually the final destination in the internal network, the *destination address* field would always contain the NAT router's external address when the packet enters the network.
- For all *outgoing* packets, regardless of which is actually the original sender in the internal network, the *source address* field would always contain the NAT router's external address when the packet leaves the network.
- As a result, the NAT router has to perform the job of address translation. For this purpose, the NAT router does the following:
 - For all *incoming* packets, the NAT router replaces the *destination address* of the packet (which is set to the NAT router's external address) with the internal address of the final receiving host.
 - For all *outgoing* packets, the NAT router replaces the *source address* of the packet (which is set to the internal address of the original sender host) with the external address of the NAT router.

This concept is shown in Fig. 9.18.

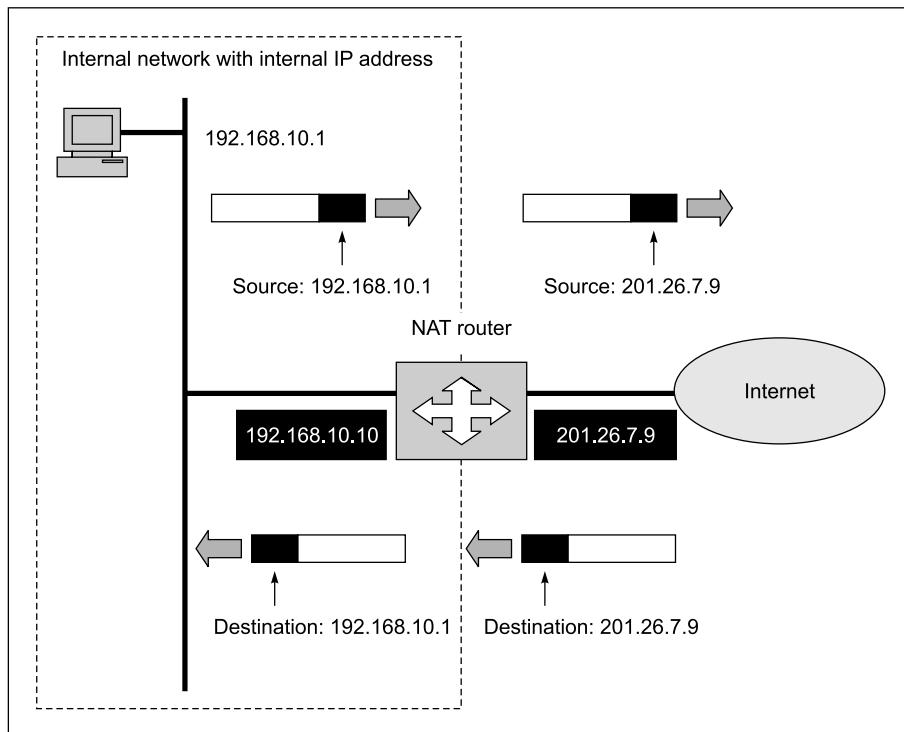


Fig. 9.18 NAT example

If we have studied this carefully, we would have realized that NAT for *outgoing packets* is pretty much straightforward. The NAT router simply has to replace the *source address* in the packet, which is the internal host's address with the external address of the NAT router. However, when it comes to *incoming packets*, how does the NAT router know what should be the actual internal host address? After all, if a network contains hundreds of hosts, the packet can be intended for any one of them!

For resolving this issue, the NAT router maintains a simple translation table, which maps the internal address of the host with the address of the external host to which the internal host is sending this packet. Thus, whenever an internal host sends a packet to an external host, the NAT makes an entry into the translation table. This entry contains the addresses of the internal host and that of the external host to which the packet is being sent over the Internet. Whenever a response comes back from any external host, the NAT router consults the translation table to see to which internal host the packet should be sent.

Let us consider an example to understand this.

- (a) Suppose an internal host (with address 192.168.10.1) wants to send a packet to an external host (with address 210.10.20.20). The internal host sends this packet on to the internal network, which reaches the NAT router. Currently, this packet contains *source address* = 192.168.10.1 and *destination address* = 210.10.20.20.
- (b) The NAT router adds an entry to the translation table, as follows:

Translation table	
Internal	External
192.168.10.1	210.10.20.20
...	...
...	...

- (c) The NAT router replaces the *source address* in the packet with its own address (i.e. 201.26.7.9) and sends the packet to the appropriate external host over the Internet, with the help of the usual routing mechanisms. Now, this packet contains *source address* = 201.26.7.9 and *destination address* = 210.10.20.20.
- (d) The external router processes the packet and sends a response back. Currently, this packet contains *source address* = 210.10.20.20 and *destination address* = 201.26.7.9.
- (e) The packet reaches the NAT router, as the destination address in the packet matches with that of the NAT router. The NAT router needs to find out whether this packet is meant for itself or for another internal host. Therefore, the NAT router consults its translation table to see if there is any entry for address 210.10.20.20 as the *External address*. In other words, the NAT router tries to find out if any host has sent a packet to and is expecting a response from an external host with address 210.10.20.20. It finds a match and comes to know that the internal host corresponding to this entry has an address of 192.168.10.1.
- (f) The NAT router replaces the *destination address* of the packet with that of the internal host for which it is destined, i.e. 192.168.10.1 and forwards the packet to this host.

This process is depicted in Fig. 9.19.

All this works fine, but we have another problem. With this scheme, only one internal host can communicate with any given external host at a given moment. Otherwise, the translation table will have multiple internal address entries for the same single external host. As a result, the NAT router will not be able to decide to which of these internal hosts a packet needs to be forwarded. In some cases, the NAT router has multiple external addresses. For example, if the NAT router has four external addresses, four internal hosts can access the same single external host now, each via a separate external NAT router address. However, there are two limitations in this approach:

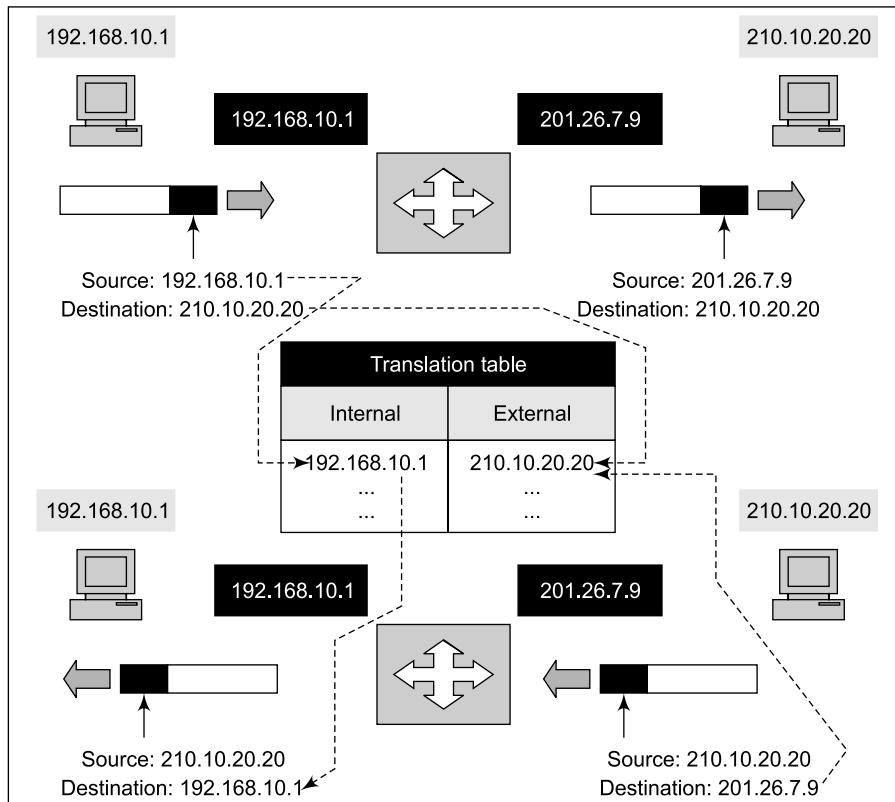


Fig. 9.19 Translation table in NAT

- (i) There is still a limitation on the number of internal users that can access the same external host simultaneously.
- (ii) A single internal host cannot access two different applications on the same single external host (e.g. HTTP and FTP) at the same time. This happens because there is no way to distinguish between one application and another. For a single internal-external host combination, our translation table has a single entry.

To resolve these issues, the translation table is modified to several new columns. The modified translation Table 9.1 gives further details.

Table 9.1 Modified NAT translation table

Internal address	Internal port	External address	External port	NAT port	Transport protocol
192.168.10.1	300	210.10.20.20	80	14000	TCP
192.168.10.1	301	210.10.20.20	21	14001	TCP
192.168.10.2	26601	210.10.20.20	80	14002	TCP
192.168.10.3	1275	207.21.1.5	80	14003	TCP

Let us now understand how these additional columns resolve our problems.

- The *Internal port* column signifies the port number used by the application program on the internal host. As per TCP/IP specifications, this port number is randomly chosen. However, this port number is important because when a response comes back from the other side corresponding to the user's request, the user's computer needs to know to which application program the response needs to be handed over. This is determined by this port number.
- The *External port* column signifies the port number used by the application program on the server side. This port number is always fixed for a given application and is known as **well-known port**. For example, an HTTP server always runs on port 80, an FTP server always runs on port 21, etc. This is why we see these numbers in this column.
- The *NAT port* is a sequentially incrementing number, generated by the NAT router. This column has absolutely nothing to do with the source or destination port numbers. Instead, it is merely used like a primary key column in the translation table when a response comes back from the external host. This is explained subsequently.
- The *Transport protocol* column is not relevant here.

Let us consider both situations: (a) Multiple applications on the same internal host wanting to access the same external host and (b) Multiple internal hosts wanting to access the same external host.

Case (a) is shown in the first two rows of our modified translation table. Internal host 192.168.10.1 wants to work with the HTTP server and the FTP server of an external host 210.10.20.20. The internal host creates two port numbers 300 and 301 dynamically to open two connections with port numbers 80 and 21 respectively of the external host. When the packets move from the internal host to the NAT router, the NAT router as usual replaces the *source address* from that of the internal host to the NAT router's address. In addition, it replaces the *source port numbers* in the packets to 14000 and 140001, respectively and adds all these details to the translation table. The packets are then sent out to the external host 210.10.20.20, as usual. When the HTTP server on this external host sends back a response to the NAT router, the NAT router sees that the *destination port number* in this incoming packet is 14000. Therefore, it knows from the translation table that this packet needs to be sent to an internal host with address 192.168.10.1 on port 300. Similarly, when a response comes back from the FTP server of the same external host, the NAT router sees that the destination port in the packet is now 14001, looks up the translation table and forwards the packet to the same internal host 192.168.10.2, but on port 301.

Based on this discussion, it should be easy to imagine how case (b) is handled. The third row in our table has an entry for an internal host with address 192.168.10.2 and port number 26601 wanting to send a packet to the same external host 210.10.20.20, again on port 80. The NAT router changes the *source port number* in the packet as before, adds an entry to the translation table and sends the packet to the external host. When the external host responds, the NAT router uses the *destination port number* in this incoming packet to map it to the appropriate internal host and port number combination (i.e. 192.168.10.2 and 26601) using the translation table and sends it to that host.

Just for the sake of completeness, we have shown another internal host sending a packet to yet another external host.

9.3.3 Firewall Configurations

In practical implementations, a firewall is usually a combination of packet filters and application (or circuit) gateways. Based on this, there are three possible configurations of firewalls, as shown in Fig. 9.20.

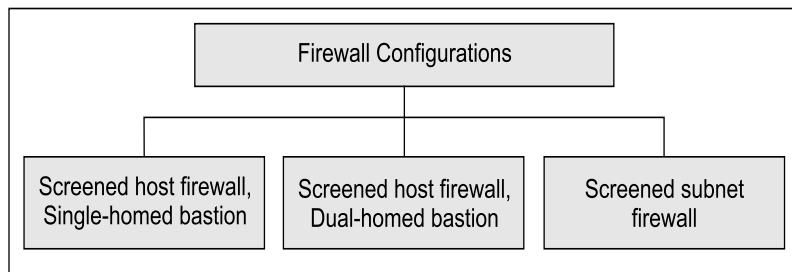


Fig. 9.20 Firewall configurations

Let us discuss these possible configurations as follows.

1. Screened Host Firewall, Single-Homed Bastion

In the **Screened host firewall, Single-homed bastion** configuration, a firewall set up consists of two parts: a packet-filtering router and an application gateway. Their purposes are as follows.

- The packet filter ensures that the incoming traffic (i.e. from the Internet to the corporate network) is allowed only if it is destined for the application gateway, by examining the *destination address* field of every incoming IP packet. Similarly, it also ensures that the outgoing traffic (i.e. from the corporate network to the Internet) is allowed only if it is originating from the application gateway, by examining the *source address* field of every outgoing IP packet.
- The application gateway performs authentication and proxy functions, as explained earlier.

This configuration is illustrated in Fig. 9.21.

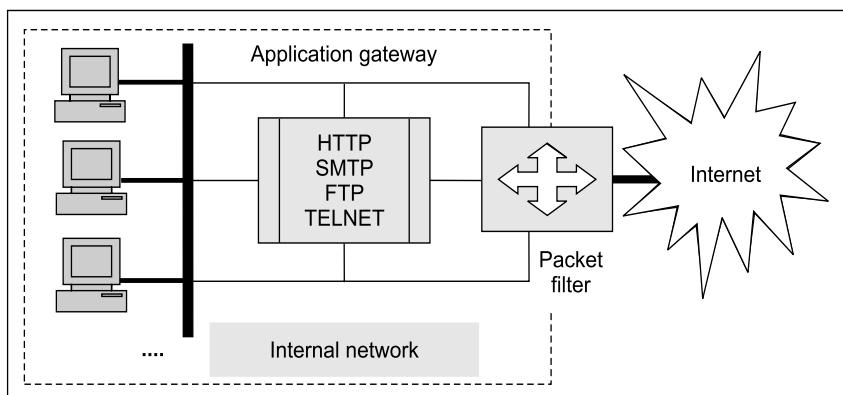


Fig. 9.21 Screened host firewall, Single-homed bastion

This configuration increases the security of the network by performing checks at both packet and application levels. This also gives more flexibility to the network administrators to define more granular security policies.

However, as we can see, one big disadvantage here is that the internal users are connected to the application gateway, as well as to the packet filter. Therefore, if the packet filter is somehow successfully attacked and its security compromised, then the whole internal network is exposed to the attacker.

2. Screened Host Firewall, Dual-Homed Bastion

To overcome the drawback of a *screened host firewall, single-homed bastion* configuration, another type of configuration, called **Screened host firewall, Dual-homed bastion**, exists. This configuration is an improvement over the earlier scheme. Here, direct connections between the internal hosts and the packet filter are avoided. Instead, the packet filter connects only to the application gateway, which, in turn, has a separate connection with the internal hosts. Therefore, now even if the packet filter is successfully attacked, only the application gateway is visible to the attacker. The internal hosts are protected. This is shown in Fig. 9.22.

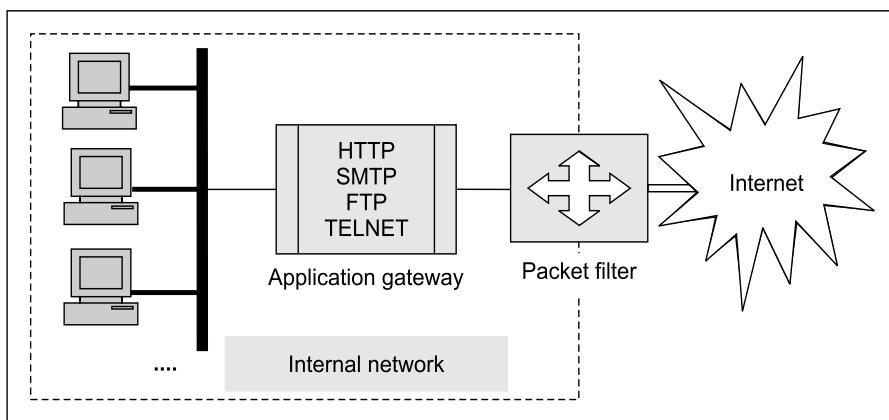


Fig. 9.22 Screened host firewall, Dual-homed bastion

Can we think of a scheme, which is even better than this?

3. Screened Subnet Firewall

The **Screened subnet firewall** offers the highest security among the possible firewall configurations. It is an improvement over the previous scheme of *screened host firewall, Dual-homed bastion*. Here, two packet filters are used, one between the Internet and the application gateway, as previously and another one between the application gateway and the internal network. This is shown in Fig. 9.23.

Now, there are three levels of security for an attacker to break into. This makes the life of the attacker very difficult. The attacker does not come to know about the internal network, unless she breaks into both the packet filters and the single application gateway standing between them.

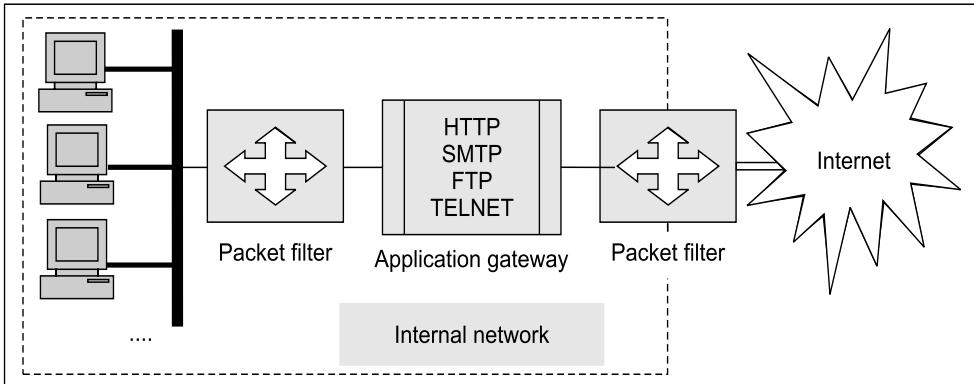


Fig. 9.23 Screened subnet firewall

9.3.4 Demilitarized Zone (DMZ) Networks

The concept of a **Demilitarized Zone (DMZ)** networks is quite popular in firewall architectures. Firewalls can be arranged to form a DMZ. DMZ is required only if an organization has servers that it needs to make available to the outside world (e.g. Web servers or FTP servers). For this, a firewall has at least three network interfaces. One interface connects to the internal private network; the second connects to the external public network (i.e. the Internet) and the third connects to the public servers (which form the DMZ network). The idea is illustrated in Fig. 9.24.

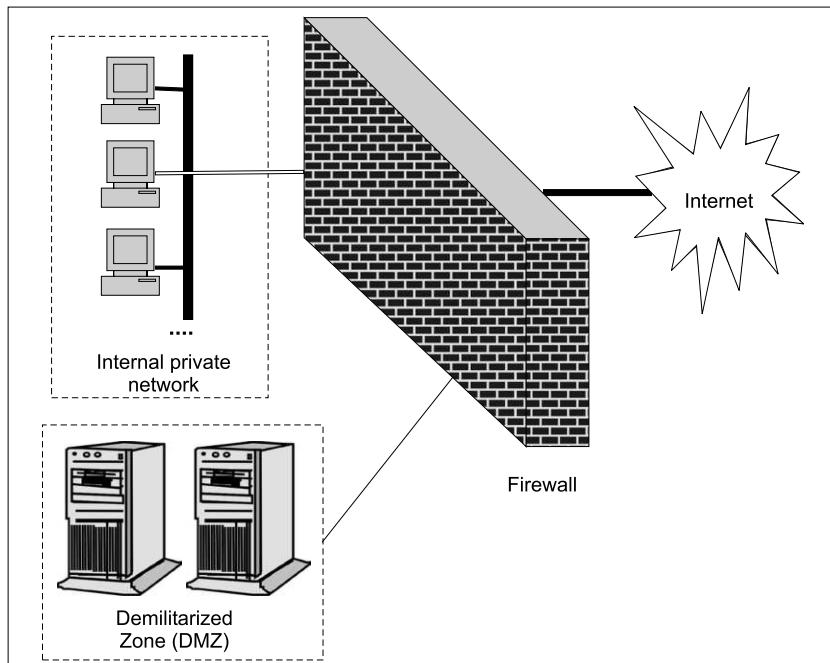


Fig. 9.24 Demilitarized (DMZ) network

The chief advantage of such a scheme is that the access to any service on the DMZ can be restricted. For instance, if the Web server is the only required service, we can limit the traffic in/out of the DMZ network to the HTTP and HTTPS protocols (i.e. ports 80 and 443, respectively). All other traffic can be filtered. More significantly, the internal private network is no way directly connected to the DMZ. So, even if an attacker can somehow manage to hack into the DMZ, the internal private network is safe and out of the reach of the attacker.

9.3.5 Limitations of Firewall

We must note that although a firewall is an extremely useful security measure for an organization, it does not solve all the practical security problems. The main limitations of a firewall can be listed as follows.

(a) Insider's Intrusion As we know, a firewall system is designed to thwart outside attacks. Therefore, if an inside user attacks the internal network in some way; the firewall cannot prevent such an attack.

(b) Direct Internet Traffic A firewall must be configured very carefully. It is effective only if it is the only entry-exit point of an organization's network. If, instead, the firewall is *one of the* entry-exit points, a user can bypass the firewall and exchange information with the Internet via the other entry-exit points. This can open up the possibilities of attacks on the internal network through those points. The firewall cannot, obviously, be expected to take care of such situations.

(c) Virus Attacks A firewall cannot protect the internal network from virus threats. This is because a firewall cannot be expected to scan every incoming file or packet for possible virus contents. Therefore, a separate virus detection and removal mechanism is required for preventing virus attacks. Alternatively, some vendors bundle their firewall products with anti-virus software, to enable both the features *out of the box*.

■ 9.4 IP SECURITY ■

9.4.1 Introduction

The IP packets contain data in plain text form. That is, anyone watching the IP packets pass by can actually access them, read their contents and even change them. We have studied higher-level security mechanisms (such as SSL, SHTTP, PGP, PEM, S/MIME, and SET) to prevent such kinds of attacks. Although these higher-level protocols enhance the protection mechanisms, there was a general feeling for a long time that why not secure IP packets themselves? If we can achieve this, then we need not rely only on the higher-level security mechanisms. The higher-level security mechanisms can then serve as additional security measures. Thus, we will have two levels of security in this scheme:

- First offer security at the IP packet level itself.
- Continue implementing higher-level security mechanisms, depending on the requirements.

This is shown in Fig. 9.25.

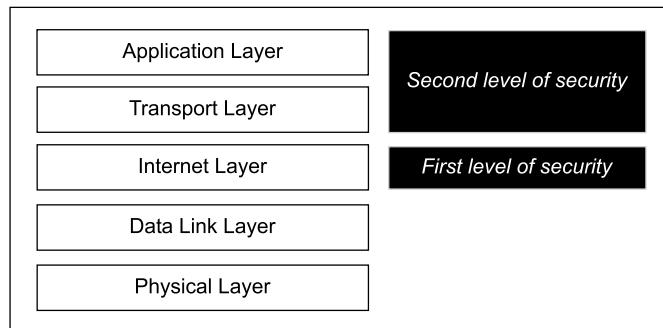


Fig. 9.25 Security at the Internet layer as well as the above layers

We have already discussed the higher-level security protocols. Our focus of discussion in this chapter is the first level of security (at the Internet layer).

In 1994, the Internet Architecture Board (IAB) prepared a report, called *Security in the Internet Architecture* (RFC 1636). This report stated that the Internet was a very open network, which was unprotected from hostile attacks. Therefore, said the report, the Internet needs better security measures, in terms of authentication, integrity and confidentiality. Just in 1997, about 150,000 Web sites were attacked in various ways, proving that the Internet was quite an unsafe place at times. Consequently, the IAB decided that authentication, integrity and encryption must be a part of the next version of the IP protocol, called as *IP version 6 (IPv6)* or *IP new generation (IPng)*. However, since the new version of IP was to take some years to be released and implemented, the designers devised ways to incorporate these security measures in the current version of IP, called as *IP version 4 (IPv4)* as well.

The outcome of the study and IAB's report is the protocol for providing security at the IP level, called as **IP Security (IPSec)**. In 1995, the Internet Engineering Task Force (IETF) published five security-based standards related to IPSec, as shown in Table 9.2.

Table 9.2 RFC Documents Related to IPSec

RFC Number	Description
1825	An overview of the security architecture
1826	Description of a packet authentication extension to IP
1827	Description of a packet encryption extension to IP
1828	A specific authentication mechanism
1829	A specific encryption mechanism

IPv4 *may* support these features, but IPv6 *must* support them. The overall idea of IPSec is to encrypt and seal the transport and application-layer data during transmission. It also offers integrity protection for the Internet layer. However, the Internet header itself is not encrypted, because of which the intermediate routers can deliver encrypted IPSec messages to the intended recipient. The logical format of a message after IPSec processing is shown in Fig. 9.26.

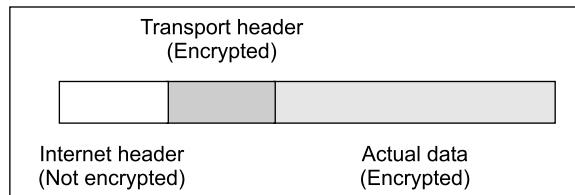


Fig. 9.26 Result of IPSec processing

Thus, the sender and the receiver look at IPSec as shown in Fig. 9.27 as another layer in the TCP/IP protocol stack. This layer sits in-between the transport and the Internet layers of the conventional TCP/IP protocol stack.

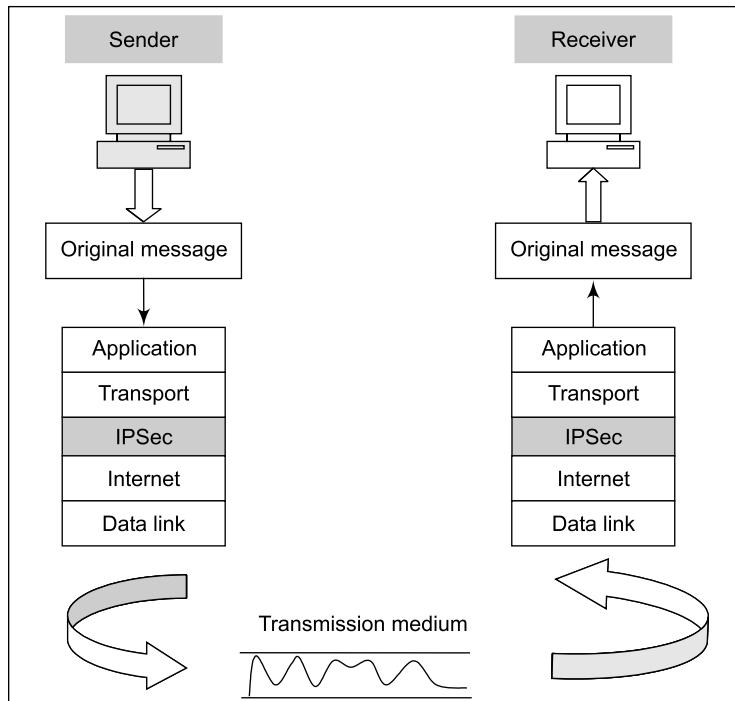


Fig. 9.27 Conceptual IPSec positioning in the TCP/IP protocol stack

9.4.2 IPSec Overview

1. Applications and Advantages

Let us first list the applications of IPSec.

(a) Secure Remote Internet Access Using IPSec, we can make a local call to our Internet Service Provider (ISP) so as to connect to our organization's network in a secure fashion from our home or hotel. From there, we can access the corporate network facilities or access remote desktops/servers.

(b) Secure Branch Office Connectivity Rather than subscribing to an expensive leased line for connecting its branches across cities/countries, an organization can set up an IPSec-enabled network to securely connect all its branches over the Internet.

(c) Set Up Communication with Other Organizations Just as IPSec allows connectivity between various branches of an organization, it can also be used to connect the networks of different organizations together in a secure and inexpensive fashion.

Following are the main advantages of IPSec.

- IPSec is transparent to the end users. There is no need for user training, key issuance or revocation.
- When IPSec is configured to work with a firewall, it becomes the only entry-exit point for all traffic; making it extra secure.
- IPSec works at the network layer. Hence, no changes are needed to the upper layers (application and transport).
- When IPSec is implemented in a firewall or a router, all the outgoing and incoming traffic gets protected. However, the internal traffic does not have to use IPSec. Thus, it does not add any overheads for the internal traffic.
- IPSec can allow traveling staff to have secure access to the corporate network.
- IPSec allows interconnectivity between branches/offices in a very inexpensive manner.

2. Basic Concepts

We must learn a few terms and concepts in order to understand the IPSec protocol. All these concepts are inter-related. However, rather than looking at these individual concepts straightaway, we shall start with the big picture. We will first take a look at the basic concepts in IPSec and then elaborate each of the concepts. In this section, we shall restrict ourselves to the broad overview of the basic concepts in IPSec.

3. IPSec Protocols

As we know, an IP packet consists of two portions: IP header and the actual data. IPSec features are implemented in the form of additional IP headers (called **extension headers**) to the standard, default IP headers. These extension IP headers follow the standard IP headers. IPSec offers two main services: authentication and confidentiality. Each of these requires its own extension header. Therefore, to support these two main services, IPSec defines two IP extension headers: one for authentication and another for confidentiality.

IPSec actually consists of two main protocols, as shown in Fig. 9.28.

These two protocols are required for the following purposes.

- The **Authentication Header (AH)** protocol provides authentication, integrity, and an optional anti-replay service. The IPSec AH is a header in an IP packet, which contains a cryptographic checksum (similar to a message digest or hash) for the contents of the packet. The AH is simply inserted between the IP header and any subsequent packet contents. No changes are required to the data contents of the packet. Thus, security resides completely in the contents of the AH.

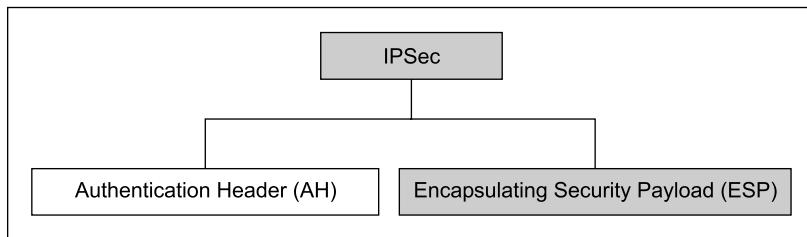


Fig. 9.28 IPSec protocols

- The **Encapsulating Security Payload (ESP)** protocol provides data confidentiality. The ESP protocol also defines a new header to be inserted into the IP packet. ESP processing also includes the transformation of the protected data into an unreadable, encrypted format. Under normal circumstances, the ESP will be *inside* the AH. That is, encryption happens first and then authentication.

On receipt of an IP packet that was processed by IPSec, the receiver processes the AH first, if present. The outcome of this tells the receiver if the contents of the packet are all right or whether they have been tampered with, while in transit. If the receiver finds the contents acceptable, it extracts the key and algorithms associated with the ESP and decrypt the contents.

There are some more details that we should know. Both AH and ESP can be used in one of the two *modes*, as shown in Fig. 9.29.

We shall later study more about these modes. However, a quick overview would help.

In the **tunnel mode**, an encrypted *tunnel* is established between two hosts. Suppose *X* and *Y* are two hosts, wanting to communicate with each other using the IPSec tunnel mode. What happens here is that they identify their respective proxies, say *P1* and *P2* and a *logical encrypted tunnel* is established between *P1* and *P2*. *X* sends its transmission to *P1*. The tunnel carries the transmission to *P2*. *P2* forwards it to *Y*. This is shown in Fig. 9.30.

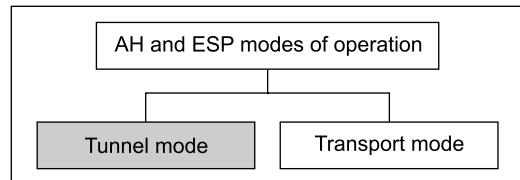


Fig. 9.29 AH and ESP modes of operation

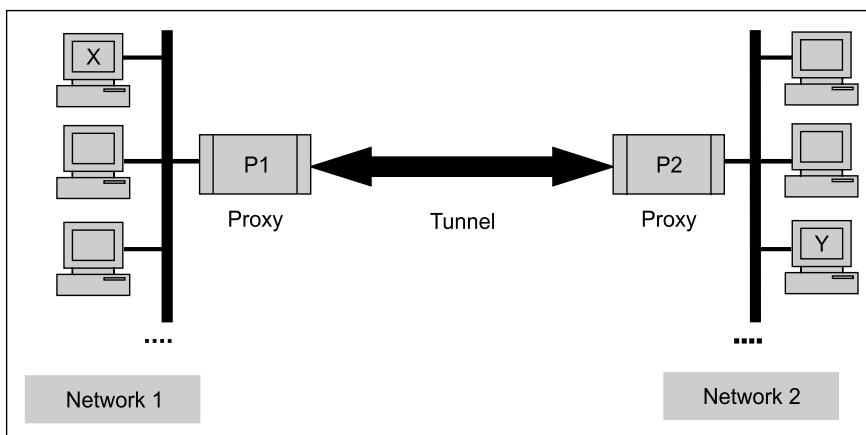


Fig. 9.30 Concept of tunnel mode

How do we implement this technically? As we shall see, we will have two sets of IP headers: internal and external. The internal IP header (which is encrypted) contains the source and destination addresses as X and Y , whereas the external IP header contains the source and destination addresses as $P1$ and $P2$. That way, X and Y are protected from potential attackers. This is shown in Fig. 9.31.

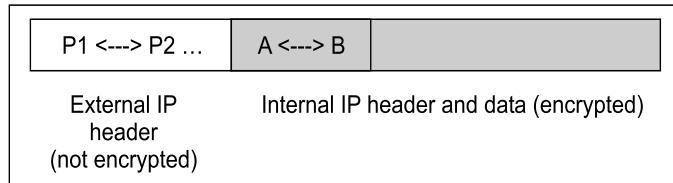


Fig. 9.31 Implementation of tunnel mode

- In the tunnel mode, IPSec protects the entire IP datagram. It takes an IP datagram (including the IP header), adds the IPSec header and trailer and encrypts the whole thing. It then adds new IP header to this encrypted datagram.

This is shown in Fig. 9.32.

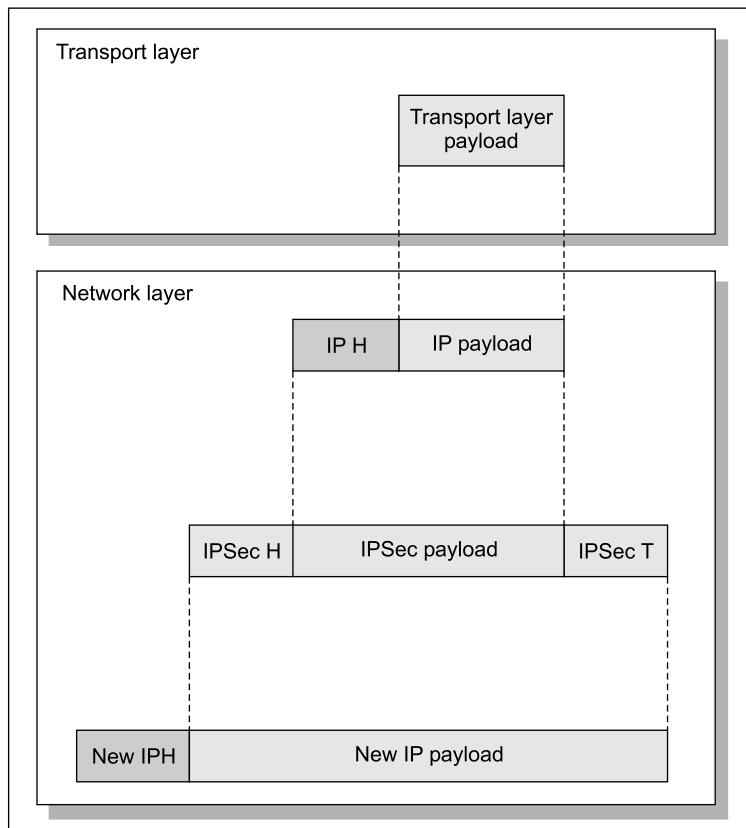


Fig. 9.32 IPSec tunnel mode

- In contrast, the transport mode does not hide the actual source and destination addresses. They are visible in plain text, while in transit. In the transport mode, IPSec takes the transport-layer payload, adds IPSec header and trailer, encrypts the whole thing and then adds the IP header. Thus, the IP header is not encrypted.

This is shown in Fig. 9.33.

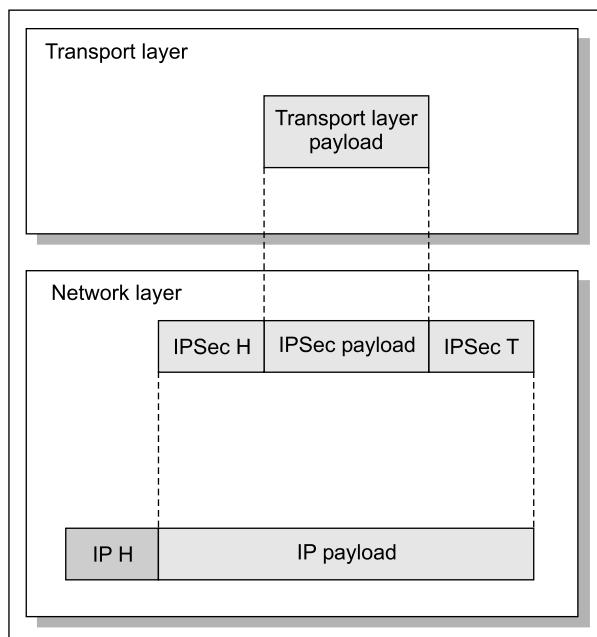


Fig. 9.33 IPSec transport mode

How to decide which mode should be used?

- We will notice that in the tunnel mode, the new IP header has information different from that is there in the original IP header. The tunnel mode is normally used between two routers, a host and a router or a router and a host. In other words, it is generally not used between two hosts, since the idea is to protect the original packet, including its IP header. It is as if the whole packet goes through an imaginary tunnel.
- The transport mode is useful when we are interested in a host-to-host (i.e. end-to-end) encryption. The sending host uses IPSec to authenticate and/or encrypt the transport layer payload and only the receiver verifies it.

4. The Internet Key Exchange (IKE) Protocol

Another supporting protocol is also used in IPSec. This protocol is used for the key management procedures and is called **Internet Key Exchange (IKE)** protocol. IKE is used to negotiate the cryptographic algorithms to be later used by AH and ESP in the actual cryptographic operations. The IPSec protocols are designed to be independent of the actual lower-level cryptographic algorithms. Thus, IKE is the initial phase of IPSec, where the algorithms and keys are decided. After the IKE phase, the AH and ESP protocols take over. This process is shown in Fig. 9.34.

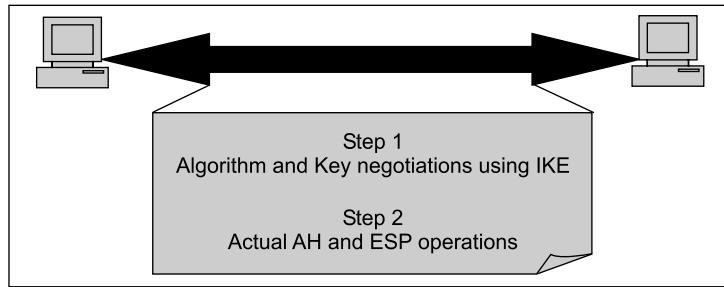


Fig. 9.34 Steps in IPSec operation

5. Security Association (SA)

The output of the IKE phase is a **Security Association (SA)**. SA is an agreement between the communicating parties about factors such as the IPSec protocol version in use, mode of operation (transport mode or tunnel mode), cryptographic algorithms, cryptographic keys, lifetime of keys, etc. By now, we would have guessed that the principal objective of the IKE protocol is to establish an SA between the communicating parties. Once this is done, both major protocols of IPSec (i.e. AH and ESP) make use of SA for their actual operation.

Note that if both AH and ESP are used, each communicating party requires two sets of SA: one for AH and one for ESP. Moreover, an SA is simplex, i.e. unidirectional. Therefore, at a second level, we need two sets of SA per communicating party: one for incoming transmission and another for outgoing transmission. Thus, if the two communicating parties use both AH and ESP, each of them would require four sets of SA, as shown in Fig. 9.35.

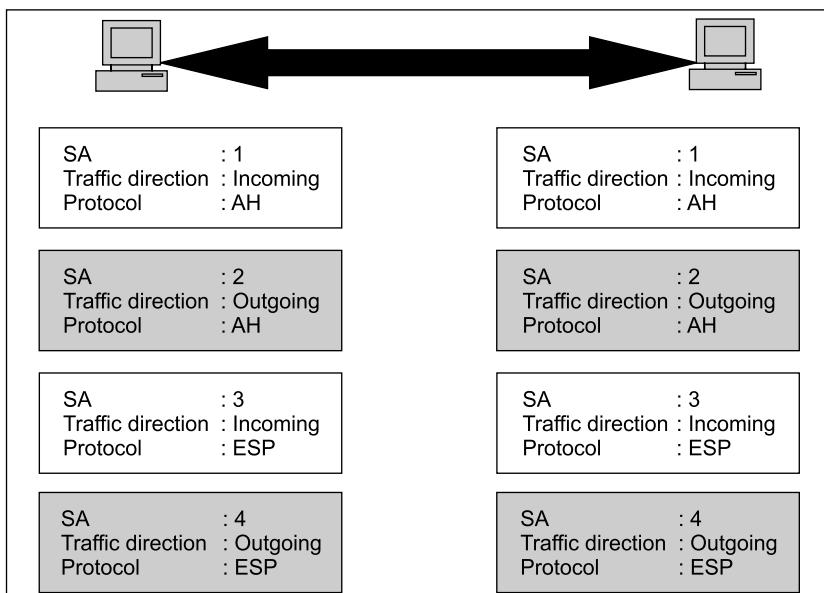


Fig. 9.35 Security association types and classifications

Obviously, both the communicating parties must allocate some storage area for storing the SA information at their end. For this purpose, a standard storage area called **Security Association Database (SAD)** is pre-defined and used by IPSec. Thus, each communicating party requires maintaining its own SAD. The SAD contains active SA entries. The contents of a SAD are shown in Table 9.3.

Table 9.3 SAD Fields

Field	Description
Sequence number counter	This 32-bit field is used to generate the sequence number field, which is used in the AH or ESP headers.
Sequence counter overflow	This flag indicates whether the overflow of the sequence number counter should generate an audible event and prevent further transmission of packets on this SA.
Anti-replay window	A 32-bit counter field and a bit map, which are used to detect if an incoming AH or ESP packet is a replay.
AH authentication	AH authentication cryptographic algorithm and the required key.
ESP authentication	ESP authentication cryptographic algorithm and the required key.
ESP encryption	ESP encryption algorithm, key, Initial Vector (IV).
IPSec protocol mode	Indicates which IPSec protocol mode (e.g. transport or tunnel) should be applied to the AH and ESP traffic.
Path Maximum Transfer Unit (PMTU)	The maximum size of an IP datagram that will be allowed to pass through a given network path without fragmentation.
Lifetime	Specifies the life of the SA. After this time interval, the SA must be replaced with a new one.

Having discussed the background of IPSec, let us now discuss the two main protocols in IPSec: AH and ESP.

9.4.3 Authentication Header (AH)

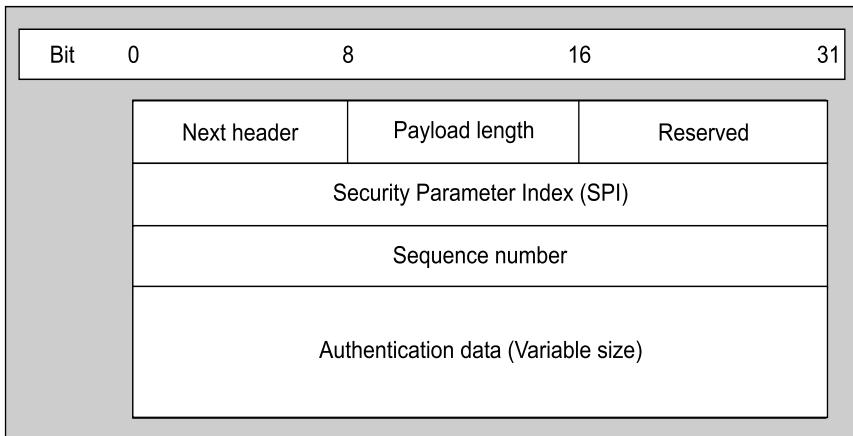
1. AH Format

The Authentication Header (AH) provides support for data integrity and authentication of IP packets. The data integrity service ensures that data inside IP packets is not altered during the transit. The authentication service enables an end user or a computer system to authenticate the user or the application at the other end and decide to accept or reject packets, accordingly. This also prevents the IP spoofing attacks. Internally, AH is based on the MAC protocol, which means that the two communicating parties must share a secret key in order to use AH. The AH structure is shown in Fig. 9.36.

Let us discuss the fields in the AH now, as shown in Table 9.4.

2. Dealing with Replay Attacks

Let us now study how AH deals with and prevents the replay attacks. To reiterate, in a replay attack, the attacker obtains a copy of an authenticated packet and later sends it to the intended destination. Since

**Fig. 9.36** Authentication Header (AH) format**Table 9.4** Authentication Header field descriptions

Field	Description
Next header	This 8-bit field identifies the type of header that immediately follows the AH. For example, if an ESP header follows the AH, this field contains a value 50, whereas if another AH follows this AH, this field contains a value 51.
Payload length	This 8-bit field contains the length of the AH in 32-bit words minus 2. Suppose that the length of the authentication data field is 96 bits (or three 32-bit words). With a three-word fixed header, we have a total of 6 words in the header. Therefore, this field will contain a value of 4.
Reserved	This 16-bit field is reserved for future use.
Security Parameter Index (SPI)	This 32-bit field is used in combination with the source and destination addresses as well as the IPSec protocol used (AH or ESP) to uniquely identify the Security Association (SA) for the traffic to which a datagram belongs.
Sequence number	This 32-bit field is used to prevent replay attacks, as discussed later.
Authentication data	This variable-length field contains the authentication data, called as the Integrity Check Value (ICV), for the datagram. This value is the MAC, used for authentication and integrity purposes. For IPv4 datagrams, the value of this field must be an integral multiple of 32. For IPv6 datagrams, the value of this field must be an integral multiple of 64. For this, additional padding bits may be required. The ICV is calculated generating a MAC using the HMAC digest algorithm.

the same packet is received twice, the destination could face some problems because of this. To prevent this, as we know, the AH contains a field called *sequence number*.

Initially, the value of this field is set to 0. Every time the sender sends a packet to the same sender over the same SA, it increments the value of this field by 1. The sender must not allow this value to circle back from $2^{32} - 1$ to 0. If the number of packets over the same increases this number, the sender must establish a new SA with the recipient.

On the receiver's side, there is some more processing involved. The receiver maintains a sliding window of size W , with the default value of $W = 64$. The right edge of the window represents the highest sequence number N received so far, for a valid packet. For simplicity, let us depict a sliding window with $W = 8$, as shown in Fig. 9.37.

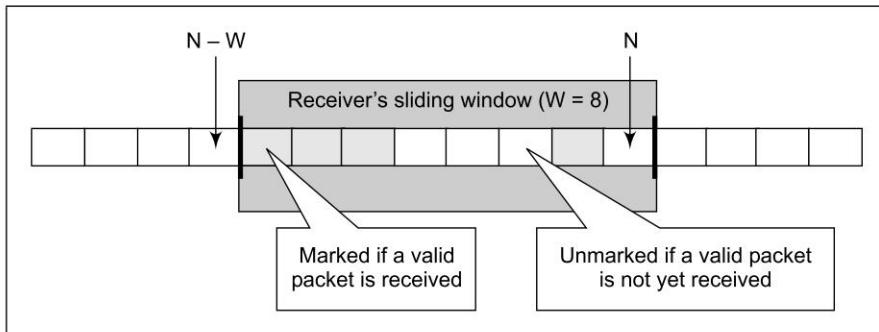


Fig. 9.37 Receiver's sliding window

Let us understand the significance of the receiver's sliding window and also see how the receiver operates on it.

As we can see, the following values are used:

- W : Specifies the size of the window. In our example, it is 8.
- N : Specifies the maximum highest sequence number so far received for a valid packet. N is always at the right edge of the window.

For any packet with a sequence number in the range from $(N - W + 1)$ to N that has been correctly received (i.e. successfully authenticated), the corresponding slot in the window is marked (see figure). On the other hand, any packet in this range, which is not correctly received (i.e. not successfully authenticated), the slot is unmarked (see figure).

Now, when a receiver receives a packet, it performs the following action depending on the sequence number of the packet, as shown in Fig. 9.38.

1. If the sequence number of the received packet falls within the window, and if the packet is new, its MAC is checked. If the MAC is successfully validated, the corresponding slot in the window is marked. The window itself does not move to the right hand side.
2. If the received packet is to the right of the window [i.e. the sequence number of the packet is $> N$], and if the packet is new, the MAC is checked. If the packet is authenticated successfully, the window is advanced to the right in such a way that the right edge of the window now matches with the sequence number of this packet. That is, this sequence number now becomes the new N .
3. If the received packet is to the left of the window [i.e. the sequence number of the packet is $< (N - W)$], or if the MAC check fails, the packet is rejected, and an audible event is triggered.

Fig. 9.38 Sliding window logic used by the receiver for each incoming packet

Note that the third action thwarts replay attacks. This is because if the receiver receives a packet whose sequence number is less than $(N - W)$, it concludes that someone posing as the sender is attempting to resend a packet sent by the sender earlier.

We must also realize that in extreme conditions, this kind of technique can make the receiver believe that a transmission is in error, even though it is not the case. For example, suppose that the value of W is 64 and that of N is 100. Now suppose that the sender sends a burst of packets, numbered 101 to 500. Because of network congestions and other issues, suppose that the receiver somehow receives a packet with sequence number 300 first. It would immediately move the right edge of the window to 300 (i.e. $N = 300$ now). Now suppose that the receiver next receives packet number 102. From our calculations, $N - W = 300 - 64 = 236$. Therefore, the sequence number of the packet just received (102) is less than ($N - W = 236$). Thus, our third condition in the earlier list would get triggered and the receiver would reject this valid packet and raise an alarm.

However, such situations are rare and with an optimized value of W , such situations can be avoided.

3. Modes of Operation

As we know, both AH and ESP can work in two modes: the transport mode and the tunnel mode. Let us now discuss AH in the context of these two modes.

(a) AH Transport Mode In the transport mode, the position of the Authentication Header (AH) is between the original IP header and the original TCP header of the IP packet. This is shown in Fig. 9.39.

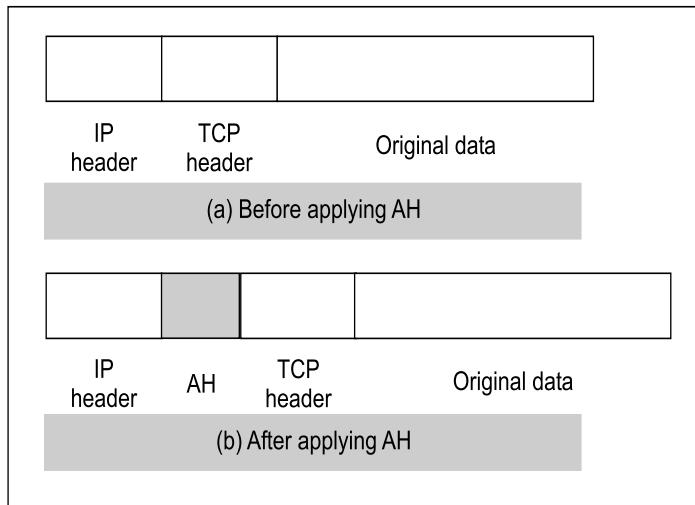


Fig. 9.39 AH transport mode

(b) AH Tunnel Mode In the tunnel mode, the entire original IP packet is authenticated and the AH is inserted between the original IP header and a new outer IP header. The inner IP header contains the ultimate source and destination IP addresses, whereas the outer IP header possibly contains different IP addresses (e.g. IP addresses of the firewalls or other security gateways). This is shown in Fig. 9.40.

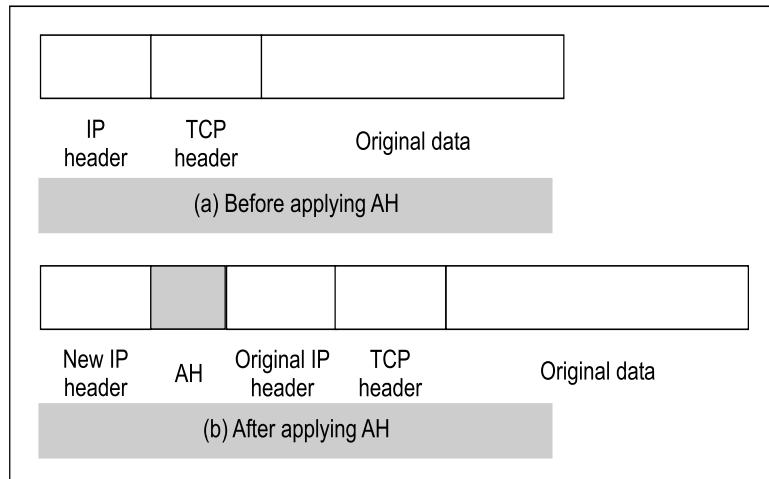


Fig. 9.40 AH tunnel mode

9.4.4 Encapsulating Security Payload (ESP)

1. ESP Format

The Encapsulating Security Payload (ESP) protocol provides confidentiality and integrity of messages. ESP is based on symmetric key cryptography techniques. ESP can be used in isolation or it can be combined with AH.

The ESP packet contains four fixed-length fields and three variable-length fields. Figure 9.41 shows the ESP format.

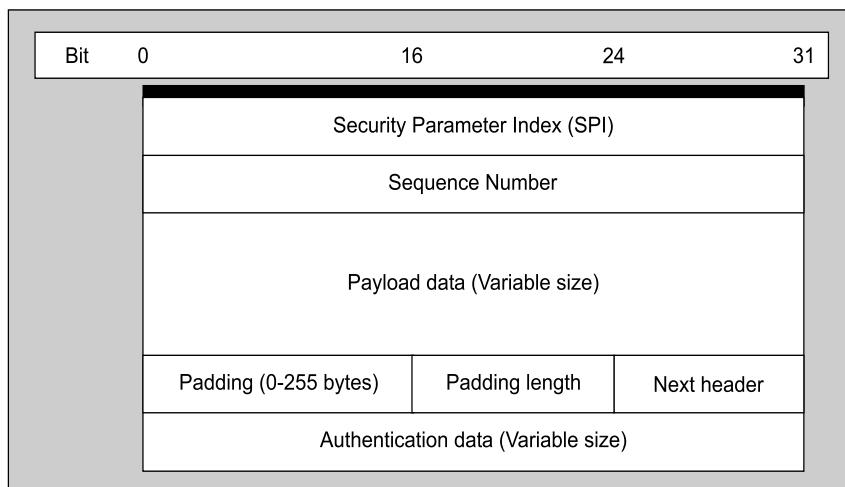


Fig. 9.41 Encapsulating Security Payload (ESP) format

Let us discuss the fields in the ESP now, as shown in Table 9.5.

Table 9.5 ESP field descriptions

Field	Description
Security Parameter Index (SPI)	This 32-bit field is used in combination with the source and destination addresses as well as the IPSec protocol used (AH or ESP) to uniquely identify the Security Association (SA) for the traffic to which a datagram belongs.
Sequence number	This 32-bit field is used to prevent replay attacks, as discussed earlier.
Payload data	This variable-length field contains the transport layer segment (transport mode) or IP packet (tunnel mode), which is protected by encryption.
Padding	This field contains the padding bits, if any. These are used by the encryption algorithm or for aligning the padding length field, so that it begins at the third byte within the 4-byte word.
Padding length	This 8-bit field specifies the number of padding bytes in the immediately preceding field.
Next header	This 8-bit field identifies the type of encapsulated data in the payload. For example, a value 6 in this field indicates that the payload contains TCP data.
Authentication data	This variable-length field contains the authentication data, called as the Integrity Check Value (ICV), for the datagram. This is calculated over the length of the ESP packet minus the Authentication Data field.

2. Modes of Operation

ESP, like AH, can operate in the transport mode or the tunnel mode. Let us discuss these two possibilities now.

(a) ESP Transport Mode Transport mode ESP is used to encrypt and optionally authenticate the data carried by IP (for example, a TCP segment). Here, the ESP is inserted into the IP packet immediately before the transport layer header (i.e. TCP or UDP) and an ESP trailer (containing the fields Padding, Padding length and Next header) is added after the IP packet. If authentication is also used, the ESP Authentication Data field is added after the ESP trailer. The entire transport layer segment and the ESP trailer are encrypted. The entire cipher text, along with the ESP header is authenticated. This is shown in Fig. 9.42.

We can summarize the operation of the ESP transport mode as follows.

- At the sender's end, the block of data containing the ESP trailer and the entire transport layer segment is encrypted and the plain text of this block is replaced with its corresponding cipher text to form the IP packet. Authentication is appended, if selected. This packet is now ready for transmission.
- The packet is routed to the destination. The intermediate routers need to take a look at the IP header as well as any IP extension headers, but not at the cipher text.
- At the receiver's end, the IP header plus any plain text IP extension headers are examined. The remaining portion of the packet is then decrypted to retrieve the original plain text transport layer segment.

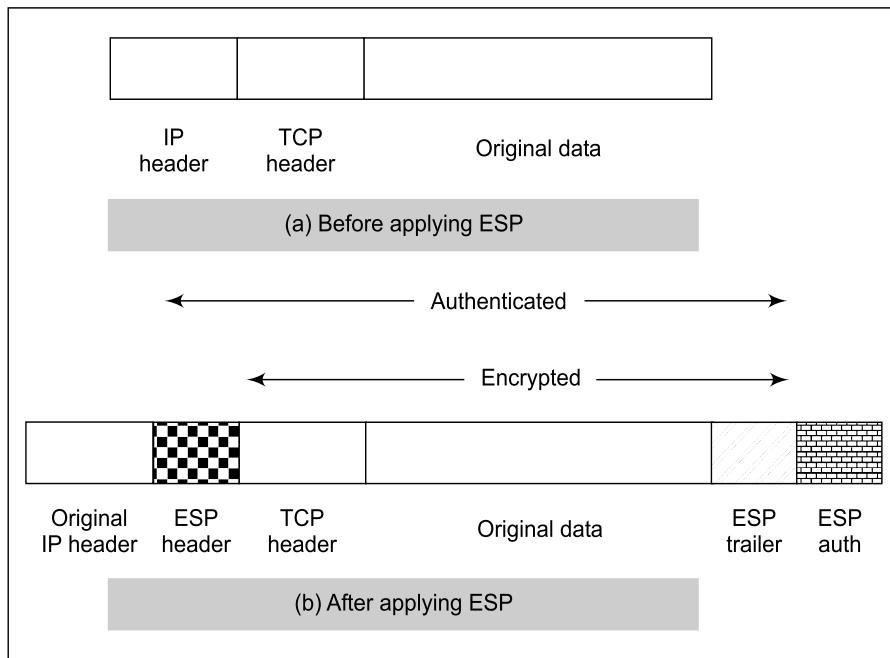


Fig. 9.42 ESP transport mode

(b) ESP Tunnel Mode The tunnel mode ESP encrypts an entire IP packet. Here, the ESP header is pre-fixed to the packet and then the packet along with the ESP trailer is encrypted. As we know, the IP header contains the destination address as well as intermediate routing information. Therefore, this packet cannot be transmitted as it is. Otherwise, the delivery of the packet would be impossible. Therefore, a new IP header is added, which contains sufficient information for routing. This is shown in Fig. 9.43.

We can summarize the operation of the ESP tunnel mode as follows.

- At the sender's end, the sender prepares an inner IP packet with the destination address as the internal destination. This packed is pre-fixed with an ESP header and then the packet and ESP trailer are encrypted and Authentication Data is (optionally) added. A new IP header is added to the start of this block. This forms the outer IP packet.
- The outer packet is routed to the destination firewall. Each intermediate router needs to check and process the outer IP header, along with any other outer IP extension headers. It need not know about the cipher text.
- At the receiver's end, the destination firewall processes the outer IP header plus any extension headers and recovers the plain text from the cipher text. The packet is then sent to the actual destination host.

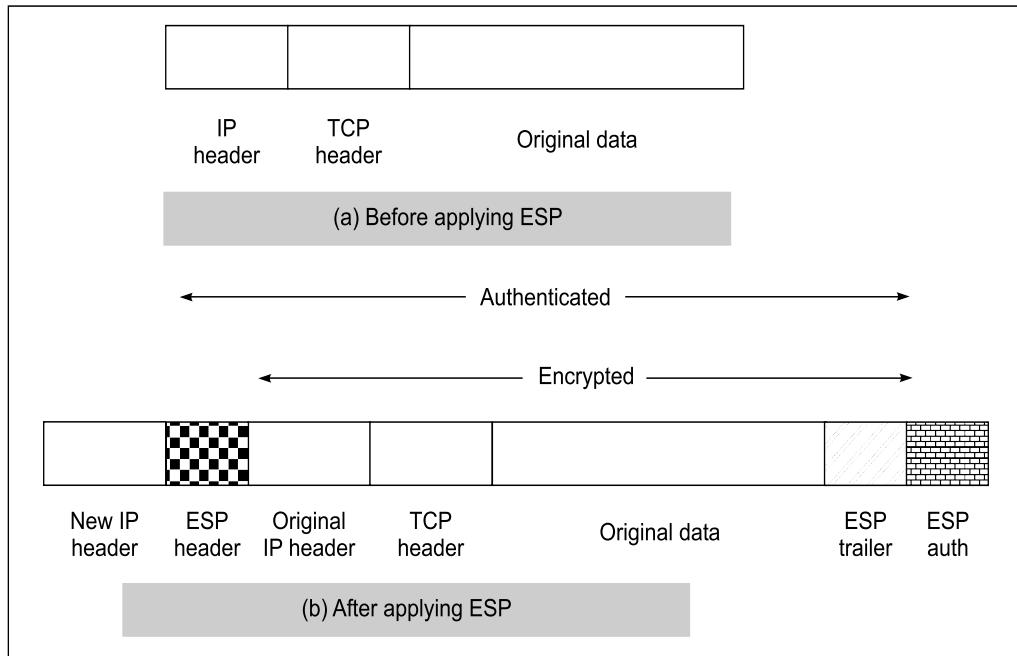


Fig. 9.43 ESP tunnel mode

9.4.5 IPSec Key Management

1. Introduction

Apart from the two core protocols (AH and ESP), the third most significant aspect of IPSec is key management. Without a proper key management set up, IPSec cannot exist. This key management in IPSec consists of two aspects: key agreement and distribution. As we know, we require four keys if we want to make use of both AH and ESP: two keys for AH (one for message transmissions, one for message receiving) and two keys for ESP (one for message transmissions, one for message receiving).

The protocol used in IPSec for key management is called as **ISAKMP/Oakley**. The **Internet Security Association and Key Management Protocol (ISAKMP)** protocol a platform for key management. It defines the procedures and packet formats for negotiating, establishing, modifying, and deleting SAs. ISAKMP messages can be transmitted via the TCP or UDP transport protocol. TCP and UDP port number 500 is reserved for ISAKMP.

The initial version of ISAKMP mandated the use of the Oakley protocol. Oakley is based on the Diffie–Hellman key exchange protocol, with a few variations. We will first take a look at Oakley and then examine ISAKMP.

2. Oakley Key Determination Protocol

The Oakley protocol is a refined version of the Diffie–Hellman key exchange protocol. We will not repeat the concepts of Diffie–Hellman, as we have already studied it in great detail. However, we will note here that Diffie–Hellman offers two desirable features:

- (a) Creation of secret keys as and when required
- (b) No requirement for any preexisting infrastructure

However, Diffie–Hellman also suffers from a few problems, as follows:

- No mechanism for authentication of the parties.
- Vulnerability to man-in-the-middle-attack.
- Involves a lot of mathematical processing. An attacker can take undue advantage of this by sending a number of hoax Diffie–Hellman requests to a host. The host can unnecessarily spend a large amount of time in trying to compute the keys, rather than doing any actual work. This is called as **congestion attack or clogging attack**.

The Oakley protocol is designed to retain the advantages of Diffie–Hellman and to remove its drawbacks. The features of Oakley are as follows.

It has features to defeat replay attacks.

- It implements a mechanism called cookies to defeat congestion attacks.
- It enables the exchange of Diffie–Hellman public key values.
- It provides authentication mechanisms to thwart man-in-the-middle attacks.

We have already discussed the Diffie–Hellman key exchange protocol in great detail. Here, we shall simply discuss the approaches taken by Oakley to tackle the issues with Diffie–Hellman.

(a) Authentication Oakley supports three authentication mechanisms: digital signatures (generation of a message digest and its encryption with the sender's private key), public key encryption (encrypting some information such as the sender's user id with the recipient's public key) and secret key encryption (a key derived by using some out-of-band mechanisms).

(b) Dealing with Congestion Attacks Oakley uses the concept of cookies to thwart congestion attacks. As we know, in this kind of attack, an attacker forges the source address of another legitimate user and sends a public Diffie–Hellman key to another legitimate user. The receiver performs modular exponentiation to calculate the secret key. A number of such calculations performed rapidly one after the other can cause congestion or clogging of the victim's computer. To tackle this, each side in Oakley must send a pseudo-random number, called as cookie, in the initial message, which the other side must acknowledge. This acknowledgement must be repeated in the first message of Diffie–Hellman key exchange. If an attacker forges the source address, she does not get the acknowledgement cookie from the victim and her attack fails. Note that at the most the attacker can force the victim to generate and send a cookie, but not to perform the actual Diffie–Hellman calculations.

The Oakley protocol provides for a number of message types. For simplicity, we shall consider only one of them, called as *aggressive key exchange*. It consists of three message exchanges between the two parties, say X and Y . Let us examine these three messages.

Message 1 To begin with, X sends a cookie and the public Diffie–Hellman key of X for this exchange, along with some other information. X signs this block with its private key.

Message 2 When Y receives *message 1*, it verifies the signature of X using the public key of X . When Y is satisfied that the message indeed came from X , it prepares an acknowledgement message for X , containing the cookie sent by X . Y also prepares its own cookie and Diffie–Hellman public key and along with some other information, it signs the whole package with its private key.

Message 3 Upon receipt of *message 2*, *X* verifies it using the public key of *Y*. When *X* is satisfied about it, it sends a message back to *Y* to inform that it has received *Y*'s public key.

(c) ISAKMP The ISAKMP protocol defines procedures and formats for establishing, maintaining and deleting SA information. An ISAKMP message contains an ISAKMP header followed by one or more payloads. The entire block is encapsulated inside a transport segment (such as TCP or UDP segment). The header format for ISAKMP messages is shown in Fig. 9.44.

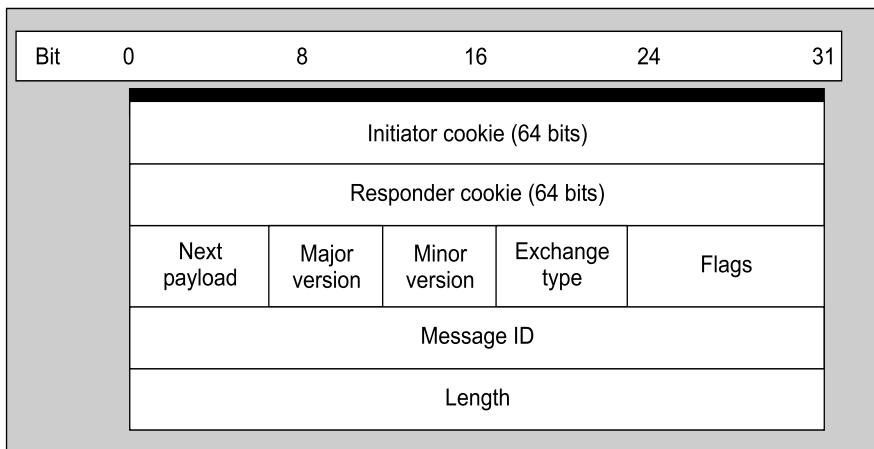


Fig. 9.44 ISAKMP header format

Let us discuss the fields in the ISAKMP header now, as shown in Table 9.6.

Table 9.6 ISAKMP header field descriptions

Field	Description
Initiator cookie	This 64-bit field contains the cookie of the entity that initiates the SA establishment or deletion.
Responder cookie	This 64-bit field contains the cookie of the responding entity. Initially, this field contains null when the initiator sends the very first ISAKMP message to the responder.
Next payload	This 8-bit field indicates the type of the first payload of the message (discussed later).
Major version	This 4-bit field identifies the major ISAKMP protocol version as used in the current exchange.
Minor version	This 4-bit field identifies the minor ISAKMP protocol version as used in the current exchange.
Exchange type	This 8-bit field indicates the type of exchange (discussed later).
Flags	This 8-bit field indicates the specific set of options for this ISAKMP exchange.
Message ID	This 32-bit field identifies the unique id for this message.
Length	This 32-bit field specifies the total length of the message, including the header and all the payloads in octets.

Let us quickly discuss the fields not explained yet.

(i) *Payload Types* ISAKMP specifies different *payload types*. For example, an *SA payload* is used to start establishment of an SA. The *proposal payload* contains information used during the SA establishment. The *key exchange payload* indicates for exchanging keys using mechanisms such as Oakley, Diffie-Hellman, RSA, etc. There are many other *payload types*.

(ii) *Exchange Types* There are five *exchange types* defined in ISAKMP. The *base exchange* allows the transmission of the key and authentication material. The *identity protection exchange* expands the *base exchange* to protect the identities of the user. The *authentication only exchange* is used to perform mutual authentication. The *aggressive exchange* attempts to minimize the number of exchanges at the cost of hiding the user's identities. The *information exchange* is used for one-way transmission of information for SA management.

■ 9.5 VIRTUAL PRIVATE NETWORKS (VPN) ■

9.5.1 Introduction

Until very recently, there has been a very clear demarcation between public and private networks. A public network, such as the public telephone system and the Internet, is a large collection of communicators who are generally unrelated with each other. In contrast, a private network is made up of computers owned by a single organization, which share information with each other. Local Area Networks (LAN), Metropolitan Area Networks (MAN) and Wide Area Networks (WAN) are examples of private networks. A firewall usually separates a private network from a public network.

Let us assume that an organization wants to connect two of its branch networks to each other. The trouble is that these branches are located quite a distance apart. One branch is in Delhi and the other branch is in Mumbai. Two solutions out of all the available ones seem logical:

- Connect the two branches using a personal network, i.e. lay cables between the two offices yourself or obtain a leased line between the two branches.
- Connect the two branches with the help of a public network, such as the Internet.

The first solution gives far more control and offers a sense of security as compared to the second solution. However, it is also quite complicated. Laying cables between two cities is not easy and is usually not permitted either. The second solution seems easier to implement, as there is no special infrastructure set up required. However, it also seems to be vulnerable to possible attacks. How nice it would be, if we could combine the two solutions!

Virtual Private Networks (VPN) offers such a solution. A VPN is a mechanism of employing encryption, authentication and integrity protection so that we can use a public network (such as the Internet) as if it is a private network (such as a physical network created and controlled by you). VPN offers high amount of security and yet does not require any special cabling on behalf of the organization that wants to use it. Thus, a VPN combines the advantages of a public network (cheap and easily available) with those of a private network (secure and reliable).

A VPN can connect distant networks of an organization or it can be used to allow traveling users to remotely access a private network (e.g. the organization's intranet) securely over the Internet.

A VPN is thus a mechanism to simulate a private network over a public network, such as the Internet. The term *virtual* signifies that it depends on the use of virtual connections. These connections are temporary and do not have any physical presence. They are made up of packets.

9.5.2 VPN Architecture

The idea of a VPN is actually quite simple to understand. Suppose an organization has two networks, *Network 1* and *Network 2*, which are physically apart from each other and we want to connect them using the VPN approach. In such a case, we set up two firewalls, *Firewall 1* and *Firewall 2*. The encryption and decryption are performed by the firewalls. The architectural overview is shown in Fig. 9.45.

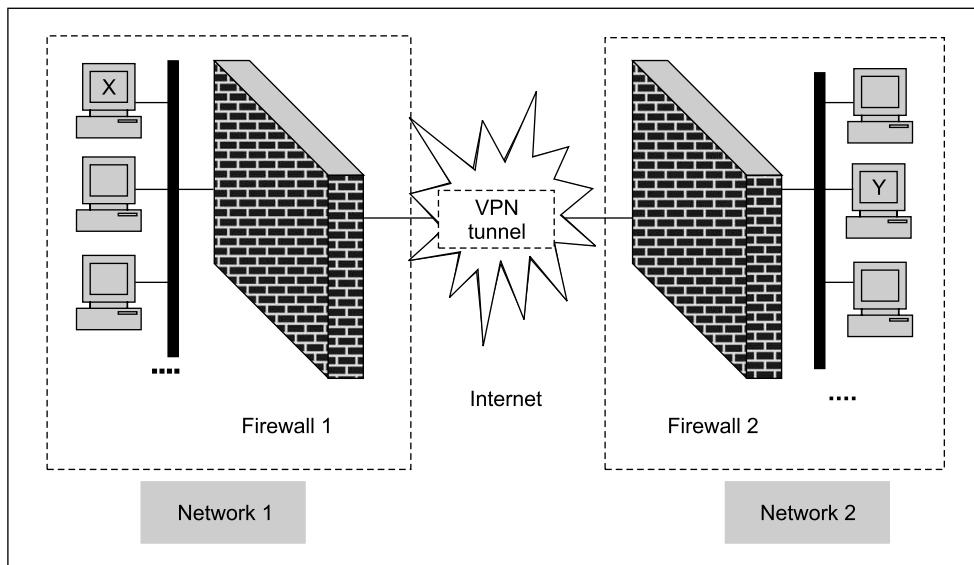


Fig. 9.45 VPN between two private networks

We have shown two networks, *Network 1* and *Network 2*. *Network 1* connects to the Internet via a firewall named *Firewall 1*. Similarly, *Network 2* connects to the Internet with its own firewall, *Firewall 2*. We shall not worry about the configuration of the firewall here and shall assume that the best possible configuration is selected by the organization. However, the key point here is that the two firewalls are *virtually* connected to each other via the Internet. We have shown this with the help of a *VPN tunnel* between the two firewalls.

With this configuration in mind, let us understand how the VPN protects the traffic passing between any two hosts on the two different networks. For this, let us assume that host *X* on *Network 1* wants to send a data packet to host *Y* on *Network 2*. This transmission would work as follows.

1. Host *X* creates the packet, inserts its own IP address as the source address and the IP address of host *Y* as the destination address. This is shown in Fig. 9.46. It sends the packet using the appropriate mechanism.

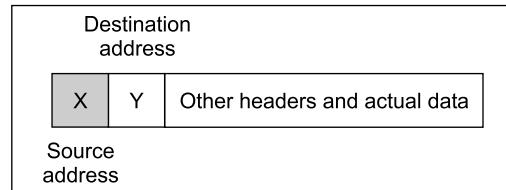


Fig. 9.46 Original packet

2. The packet reaches *Firewall 1*. As we know, *Firewall 1* now adds new headers to the packet. In these new headers, it changes the source IP address of the packet from that of host *X* to its own address (i.e. the IP address of *Firewall 1*, say *F1*). It also changes the destination IP address of the packet from that of host *Y* to the IP address of *Firewall 2*, say *F2*). This is shown in Fig. 9.47. It also performs the packet encryption and authentication, depending on the settings and sends the modified packet over the Internet.

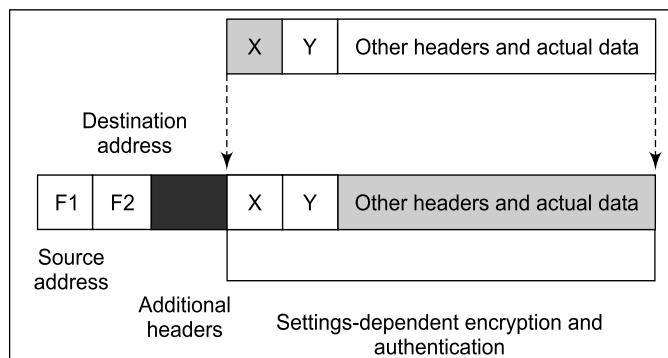


Fig. 9.47 Firewall 1 changes the packet contents

3. The packet reaches *Firewall 2* over the Internet, via one or more routers, as usual. *Firewall 2* discards the outer header and performs the appropriate decryption and other cryptographic functions as necessary. This yields the original packet, as was created by host *X* in Step 1. This is shown in Fig. 9.48. It then takes a look at the plain text contents of the packet and realizes that the packet is meant for host *Y* (because the destination address inside the packet specifies host *Y*). Therefore, it delivers the packet to host *Y*.

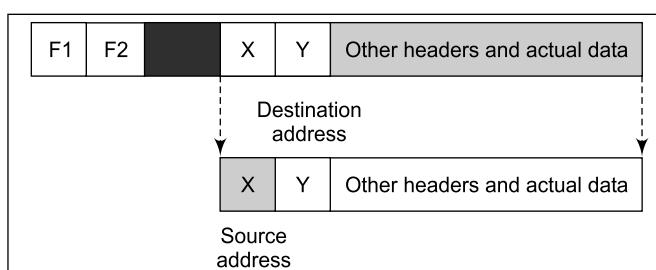


Fig. 9.48 Firewall 2 retrieves the original packet contents

There are three main VPN protocols. A detailed study of these protocols is beyond the scope of the current text. However, we shall briefly discuss them for the sake of completeness.

- The **Point to Point Tunneling Protocol (PPTP)** is used on Windows NT systems. It mainly supports the VPN connectivity between a single user and a LAN, rather than between two LANs.
- Developed by IETF, the **Layer 2 Tunneling Protocol (L2TP)** is an improvement over PPTP. L2TP is considered as the secure open standard for VPN connections. It works for both combinations: user-to-LAN and LAN-to-LAN. It can include the IPSec functionality as well.
- Finally, IPSec can be used in isolation. We have discussed IPSec in detail earlier.

■ 9.6 INTRUSION ■

9.6.1 Intruders

No matter how much secure a system is made, there would be attackers, who would constantly try to find their way. We call them **intruders**, because they try to intrude into the privacy of a network. Whether the network itself is private (e.g. a Local Area Network) or public (the Internet) does not matter. What matters is the intent of the attacker, of trying to intrude. It is generally said that the two most widely known threats to security are intruders and viruses. We shall concentrate on intruders here.

Intruders are said to be of three types, as explained below:

(a) Masquerader A user who does not have the authority to use a computer, but penetrates into a system to access a legitimate user's account is called as a **masquerader**. It is generally an external user.

(b) Misfeasor There are two possible cases for an internal user to be called a **misfeasor**:

- A legitimate user, who does not have access to some applications, data or resources accesses them.
- A legitimate user, who has access to some applications, data or resources misuses these privileges.

(c) Clandestine User An internal or external user who tries to work using the privileges of a supervisor user to avoid auditing information being captured and recorded is called as a **clandestine user**.

How do intruders try to attack? A simple example may be considered, where the attackers try to obtain the passwords of legitimate users, so as to impersonate them. Some of the popularly known methods of password guessing are as follows:

1. Try all possible short password combinations (2-3 characters).
2. Collect information about users, such as their full name, names of family members, their hobbies, etc.
3. Try default passwords that are provided by the supplier of a software product (e.g. Oracle comes with *scott* as the user name and *tiger* as the password).
4. Try words that people choose as passwords most often. Hacker bulletin boards maintain these lists. Also, try words from dictionary.
5. Try using phone numbers, dates of birth, social security numbers, bank account numbers, etc.
6. Tap the communication line between a user and the host network.

7. Use a Trojan Horse.
8. Try numbers on the vehicle license plates.

Regardless of how the intruder gets into a system, we need to first try and prevent it, if not, at least detect it and take an appropriate action.

9.6.2 Audit Records

One of the most important tools in intrusion detection is the usage of **audit records**, also called **audit logs**. Audit records are used to record information about the actions of users. Traces of illegitimate user actions can be found in these records, so as to detect intrusions so as to take appropriate actions.

Audit records can be classified into two categories: **Native audit records** and **Detection-specific audit records**. This is shown in Fig. 9.49.

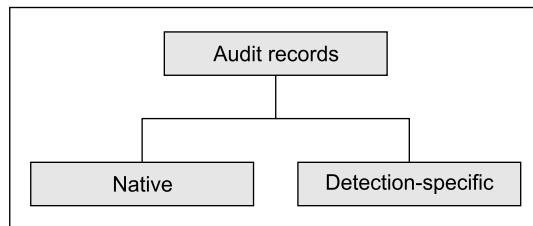


Fig. 9.49 Audit records classification

(a) Native Audit Records All multi-user operating systems have accounting software built-in. This software records information about all user actions.

(b) Detection-specific Audit Records This type of audit records facility collects information specific only to intrusion detection. This is more focused, but may duplicate information.

Regardless of the type of audit records, each such record contains information as shown in Table 9.7.

Table 9.7 Fields in an audit record

Field	Description
Subject	Information about who has initiated this action (e.g. terminal user, process, etc)
Action	Operation performed by the subject on an object (e.g. login, read, write, print, I/O, etc)
Object	Receiver of an action (e.g. a disk file, application program, a database record, etc)
Exception-condition	Exception if any, that resulted because of the subject's action
Resource-usage	Record of resource usage (e.g. CPU time, disk space, number of records written, number of files printed, etc)
Timestamp	Date and time information to as detailed level as possible

For example, if user Ram attempts to execute a program *payroll.exe*, the following audit records may get generated. Here we assume that Ram does not have the access rights to execute this program.

Ram	Execute	<SYSTEM CALL> EXECUTE	None	CPU = 0000001 16:17:10::101	24-10-2006 16:17:10::101
Ram	Execute	<SYSTEM CALL> EXECUTE	Access-violation	Records = 0	24-10-2006 16:17:10::102

As we can see, Ram attempted to execute a program for which he has no access.

9.6.3 Intrusion Detection

Intrusion prevention is almost impossible to achieve at all times. Hence, more focus is on **intrusion detection**.

Following factors motivate efforts on intrusion detection:

- (a) The sooner we are able to detect an intrusion, the quicker we can act. The hope of recovering from attacks and losses is directly proportional to how quickly we are able to detect an intrusion.
- (b) Intrusion detection can help collect more information about intrusions, strengthening the intrusion prevention methods.
- (b) Intrusion detection systems can act as good deterrents to intruders.

Intrusion detection mechanisms, also known as **Intrusion Detection Systems (IDS)** are classified into two categories: **Statistical anomaly detection** and **Rule-based detection**. This is shown in Fig. 9.50.

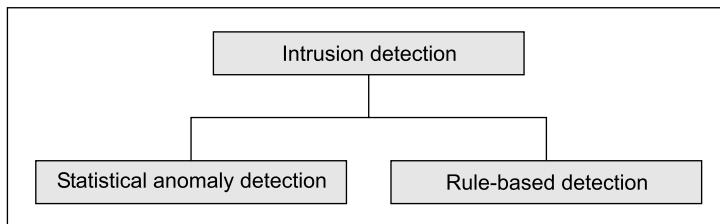


Fig. 9.50 Classification of intrusion detection

(a) Statistical Anomaly Detection In this type, behavior of users over time is captured as statistical data and processed. Rules are applied to test whether the user behavior was legitimate or not. This can be done in two ways:

(i) Threshold Detection In this type, thresholds are defined for all the users as a group and frequency of various events is measured against these thresholds.

(ii) Profile-based Detection In this type, profiles for individual users are created and they are matched against the collected statistics to see if any irregular patterns emerge.

(b) Rule-based Detection A set of rules is applied to see if a given behavior is suspicious enough to be classified as an attempt to intrude. This is also classified into two sub-types:

(i) *Anomaly Detection* Usage patterns are collected to analyze deviation from these usage patterns, with the help of certain rules.

(ii) *Penetration Identification* This is an expert system that looks for illegitimate behavior.

9.6.4 Distributed Intrusion Detection

Focus has started moving from intrusion detection on single systems to distributed systems, e.g. a LAN or a WAN. Following factors are important in this scheme of **distributed intrusion detection**:

- Different systems in the distributed system may record audit information in different formats. This needs to be uniformly processed.
- Typically one or a few nodes on the distributed system would be used to gather and analyze audit information. Hence, there should be provisions to securely send audit information from all other hosts to these hosts.

9.6.5 Honeypots

Modern intrusion detection systems make use of a novel idea, called as **honeypots**. A honeypot is a trap that attracts potential attackers. A honeypot is designed so as to do the following:

- Divert the attention of a potential intruder from critical systems
- Collect information about the intruder's actions
- Provide encouragement to the intruder so as to stay on for some time, allowing the administrators to detect this and swiftly act on it

Honeypots are designed with two important goals in mind:

- (a) Make them look like real-life systems. Put as much of real-looking (but fabricated) information into them as possible.
- (b) Do not allow legitimate users to know about or access them.

Naturally, anyone trying to access a honeypot is a potential intruder. Honeypots are armed with sensors and loggers, which alarm the administrators of any user actions.

■ CASE STUDY 1: IP SPOOFING ATTACKS ■

Points for Classroom Discussions

1. *What is the IP spoofing attack?*
2. *Why is it not easy to detect IP spoofing attacks?*

IP Spoofing attacks are more challenging than the DOS attacks, if the attacker intends to use the consequence of the attack for his own benefit. Kevin Mitnick launched such an attack successfully on Tsutomu Shimomura's home computer and the computer at the University of Southern California. Before describing the attack, the technical mechanism to achieve it is first described as follows:

1. The attacker creates an IP datagram (packet) to be sent to the server, just like any other normal packet. This is a SYN request.
2. The main difference between the packet created by the attacker and any other packet is that the attacker puts the *source address* as another computer's IP address. That is, the attacker *forges* or *spoofs* the source IP address.
3. As usual, the server responds back with a SYN ACK response, which travels to the computer with the forged IP address (i.e. not to the attacker).
4. The attacker has to somehow get hold of this SYN ACK response sent by the server and acknowledge it, so as to complete a connection with the server.
5. Once this is done, the attacker can try various commands on the server computer.

Kevin Metnick was a person who had a lot of *background* in hacking/attacking computer systems. Following are some of the incidents he was involved in.

- (a) In his early days, Kevin joined a group of hackers. Among their many attacks, the most notable was the one in which they changed a home phone to a pay phone! Thus, when the phone subscriber wanted to make a call, a message greeted her with a request to first pay 20 cents!
- (b) At the age of 17, Kevin and his friend actually entered a phone company's office and stole passwords from there! This resulted into a jail term for them.
- (c) In 1983, Kevin tried to break into a Pentagon computer over the ARPAnet, an act that was detected, causing another imprisonment for Kevin.

There were many such examples.

In the IP spoofing attack, Kevin performed the following tasks in a very intelligent fashion. Before we explain it, it should be noted that Tsutomu Shimomura had a trusted relationship between his home computer (X) and the computers at the University of Southern California (Y).

1. Kevin first flooded Tsutomu's home computer (X) with a series of SYN requests, causing it to virtually come to a halt.
2. Kevin then sent a SYN request to the main server (Y) at the University. In this packet, Kevin put the source address as X . That is, the source address was spoofed.
3. The server (Y) detected this as an attempt to establish a request for a TCP connection and responded back with a SYN ACK response. As expected, the SYN ACK response went back to Tsutomu's home computer (X), because that was the source address in the original SYN request of Step 2.
4. Kevin had flooded X in Step 1. So, X is not able to see Y 's response.
5. Kevin now guessed the sequence number used by Y in the SYN ACK response (after a few failures) and used that number in the acknowledgement of the SYN ACK message, which it sent to Y . That is, Kevin sent many acknowledgements (with different sequence numbers) of the SYN ACK message from Y , to Y .
6. In each case, Kevin immediately sent a command to add a wildcard entry to the trust relationship file maintained by Y , which made Y to trust anybody and everybody. Obviously, this command failed for all the unsuccessful attempts of Kevin to send an acknowledgement of the SYN ACK message. But it succeeded for the one acknowledgement, which succeeded. This opened up Y for all outside users!

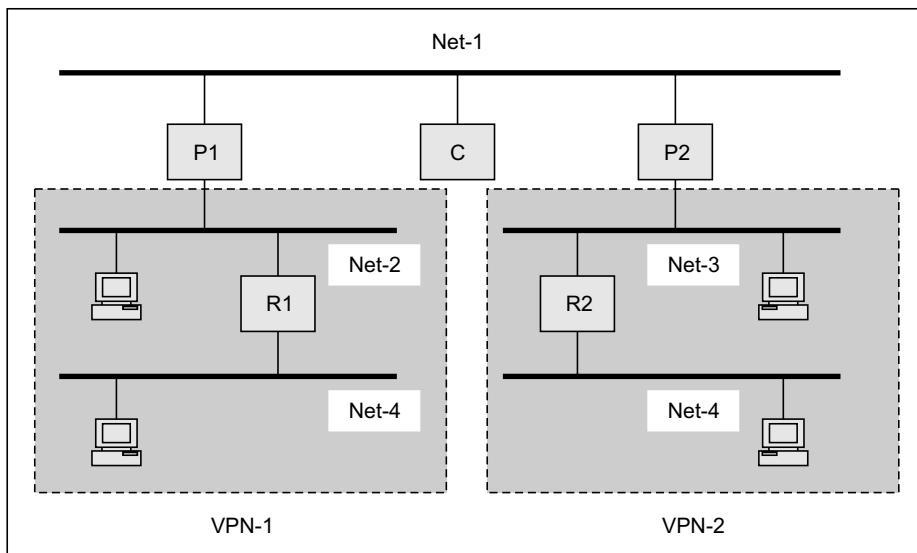
■ CASE STUDY 2: CREATING A VPN ■

Points for Classroom Discussions

1. Review the concepts of a VPN.
2. Discuss the pre-requisites of a VPN.
3. Create a VPN and test it, as explained below.

There are many companies/Web sites that offer free VPN software. An example is <http://sunsite.dk/vpnd>. This VPN software runs on Linux operating system. For the purpose of setting up the VPN, proceed as follows:

1. Use five layer-2 hubs, two routers (R1 and R2), two PCs running Linux (P1 and P2), which function as routers and five general purpose computers to form the intranet shown in the figure.



2. Download and compile a VPN software for Linux and install it on the two Linux computers (P1 and P2).
3. Configure the VPN software so that it encrypts all the communication between the two computers.
4. Run software on the *checker computer* (C), which will capture and display all packets traveling on network 1.
5. Do a *TELNET* from a computer in location 1 on to another computer in location 2. Verify that all data is encrypted.



Summary

- Corporate networks can be attacked from outside or internal information can be leaked out.
- Encryption cannot prevent outside attackers from attacking a network.
- A firewall should be placed between a corporate network and the outside world.
- A firewall is a special type of router, which applies rules for allowing or stopping traffic.
- A firewall stands like a sentry on the main door between the internal network and the external Internet.
- A firewall can be application gateway or packet filter.
- A packet filter examines every packet, applies rules and decides whether to allow the traffic to proceed or not.
- Packet filters are quite useful in applying more specific/granular rules.
- Dynamic packet filter (also called as stateful packet filter) adapts itself to the changing conditions.
- An application gateway works at the application layer. It decides whether to allow traffic for a certain application (e.g. HTTP or FTP). It does not apply granular rules such as *Stop the packet if the source IP address is x.x.x.x*, the way packet filter works.
- A circuit gateway creates a new connection between itself and the remote host.
- Firewall architectures combine the various types of firewalls in some combination.
- Screened subnet firewall is the strongest firewall architecture.
- Network Address Translation (NAT) allows a few IP addresses to be shared across many networks in the world, thus saving on the IP address space.
- Without NAT, the available range of IP addresses would have exhausted long back.
- NAT classifies certain IP addresses as internal, which have no recognition outside that network.
- Internal addresses are unique inside a network but are duplicated across different networks.
- A NAT router performs the job of translating between an internal and an external address.
- The NAT router has to maintain a translation table and use some intelligent tricks to perform address translation.
- IPSec provides security between the transport and the Internet layers.
- IPSec provides authentication and confidentiality services.
- A Demilitarized Zone (DMZ) firewall protects both the servers of an organization that are to be exposed to the external world, as well as the internal corporate network, which needs to be secured from the external world.
- IPSec protocol is used to apply security at the network layer.
- IPSec does not concern itself with the higher security mechanisms, such as SSL. IPSec can be implemented in addition to such protocols.
- IPSec can be implemented in tunnel mode or transport mode.
- In the tunnel mode, the entire IP datagram, including its original header, is encrypted by IPSec and a new IP header is added.

- In the transport mode, the IP datagram except its header is encrypted by IPSec.
- The tunnel mode creates a virtual tunnel between the two communicating computers (usually routers).
- IPSec makes use of two protocols: Authentication Header (AH) and Encapsulating Security Payload (ESP).
- The AH protocol provides authentication, integrity and an optional anti-replay service.
- The ESP protocol provides data confidentiality.
- The Internet Key Exchange (IKE) protocol is used to negotiate the cryptographic algorithms to be used later by AH and ESP.
- The output of IKE is called as Security Association (SA).
- A Virtual Private Network (VPN) is both *virtual* (it does not exist physically as a single-wired network) and *private* (provides features that make it look like a private network, although it runs on the open Internet).
- VPN is a very good facility for traveling staff, connecting offices in different cities/countries and linking up with other companies in an inexpensive fashion.
- VPN uses IPSec internally.
- VPN can be implemented as a Point to Point Tunneling Protocol (PPTP) on Windows or as a Layer 2 Tunneling Protocol (L2TP) as an open standard.
- Intrusions are almost impossible to prevent. Hence, an attempt is made to detect them.
- Intruders are classified into masquerader, misfeasor and clandestine user.
- Audit records are used to record information about the actions of users.
- Audit records can be classified into Native audit records and Detection-specific audit records.
- Intrusion Detection Systems (IDS) are classified into Statistical anomaly detection and Rule-based detection.
- In distributed intrusion detection, intrusions on multiple computers of a network need to be detected and recorded.
- A honeypot is a trap that attracts potential attackers.



Key Terms and Concepts

- | | |
|---|---|
| <ul style="list-style-type: none">● AH transport mode● Anomaly detection● Audit log● Authentication Header (AH)● Circuit gateway● Clogging attack● Demilitarized Zone (DMZ)● Distributed intrusion detection | <ul style="list-style-type: none">● AH tunnel mode● Application gateway● Audit records● Bastion host● Clandestine user● Congestion attack● Detection-specific audit records● Dynamic packet filter |
|---|---|

- Encapsulating Security Payload (ESP)
- ESP tunnel mode
- Firewall
- Internet Key Exchange (IKE)
- Intruder
- Intrusion detection
- IP packet spoofing
- ISAKMP/Oakley
- Masquerader
- Native audit records
- Packet filter
- Point to Point Tunneling Protocol (PPTP)
- Proxy server
- Screened host firewall, Dual-homed bastion
- Screened subnet firewall
- Security Association (SA)
- Source routing attacks
- Statistical anomaly detection
- Tiny fragment attacks
- Well-known port
- ESP transport mode
- Extension headers
- Honeypot
- Internet Security Association and Key Management Protocol (ISAKMP)
- Intrusion
- Intrusion prevention
- IP Security (IPSec)
- Layer 2 Tunneling Protocol (L2TP)
- Misfeasor
- Network Address Translation (NAT)
- Penetration identification
- Profile-based detection
- Rule-based detection
- Screened host firewall, Single-homed bastion
- Screening router
- Security Association Database (SAD)
- Stateful packet filter
- Threshold detection
- Virtual Private Network (VPN)



PRACTICE SET

■ Multiple-Choice Questions

1. Firewall should be situated _____ .
 - (a) inside a corporate network
 - (b) outside a corporate network
 - (c) between a corporate network and the outside world
 - (d) none of these
2. Firewall is a specialized form of a _____ .
 - (a) bridge
 - (b) disk
 - (c) printer
 - (d) router

3. A packet filter examines _____ packets.
 - (a) all
 - (b) no
 - (c) some
 - (d) alternate
 4. _____ adapts itself to the changing conditions.
 - (a) Stateless static filter
 - (b) Static packet filter
 - (c) Adaptive packet filter
 - (d) Stateful packet filter
 5. Application gateways are _____ than packet filters.
 - (a) less secure
 - (b) more secure
 - (c) equally secure
 - (d) slower
 6. In _____, direct connections between the internal hosts and the packet filter are avoided.
 - (a) Screened host firewall, Triple-homed bastion
 - (b) Screened host firewall, Single-homed bastion
 - (c) Screened host firewall, Null-homed bastion
 - (d) none of the above
 7. _____ allows reuse of IP addresses.
 - (a) Firewalls
 - (b) IPSec
 - (c) NAT
 - (d) VPN
 8. In order to allow multiple hosts to communicate with a single external host without compromising on the IP address range, the router needs to add details of the _____ to its translation tables.
 - (a) IP addresses
 - (b) port numbers
 - (c) protocol information
 - (d) external host
9. IPSec provides security at the _____ layer.
 - (a) application
 - (b) transport
 - (c) network
 - (d) data link
 10. ISAKMP/Oakley is related to _____.
 - (a) SSL
 - (b) SET
 - (c) SHTTP
 - (d) IPSec
 11. Key management in IPSec is done by _____.
 - (a) tunnel mode
 - (b) transport mode
 - (c) IKE
 - (d) ESP
 12. Encryption in IPSec is done by _____.
 - (a) tunnel mode
 - (b) transport mode
 - (c) IKE
 - (d) ESP
 13. In _____ the IP header of the original packet is also encrypted
 - (a) only tunnel mode
 - (b) only transport mode
 - (c) both tunnel mode and transport mode
 - (d) n mode
 14. Information about possible intruders can be obtained by examining the _____.
 - (a) router log
 - (b) host log
 - (c) IPSec entries
 - (d) audit log

■ Exercises

1. What can be the two main attacks on corporate networks?
 2. List the characteristics of a good firewall implementation.
 3. What are the three main actions of a packet filter?
 4. How is a circuit gateway different from an application gateway?
 5. What is the disadvantage of a Screened host firewall, Single-homed bastion?
 6. How is Screened host firewall, Dual-homed bastion different from Screened host firewall, Single-homed bastion?
 7. When is a Demilitarized Zone (DMZ) required? How is it implemented?
 8. What are the limitations of a firewall?
 9. What is the significance of tunnel mode?
 10. Explain how NAT works with an example.
 11. Why do we need to record port numbers in NAT?
 12. What is a VPN?
 13. Explain the AH and ESP protocols.
 14. Explain how audit logs work.
 15. What is a honeypot?

■ Design/Programming Exercises

1. Study at least one real-life firewall product. Study its features with reference to the theory introduced in this chapter.
 2. Try to download a free home firewall. Which of its features are annoying at times? Why?
 3. Can a firewall double up as an anti-virus product? Why?
 4. Configure the rules of a packet filter.
 5. Study how VPN is implemented in real life. Does it need digital certificates? Why?
 6. Implement NAT in software as a dummy exercise in Java. This NAT software should detect if a packet is outgoing or incoming and route it appropriately.
 7. Enhance the above solution to also take care of port numbers so as to allow multiple internal hosts communicate with a single external host.
 8. Study any one instance of real-life audit records. Is it sufficient to provide intrusion detection information?
 9. Would you like to enhance the audit records structure? How?
 10. Would you consider IPSec as a replacement for SSL? Why?
 11. Do you think it is easier to implement IPSec than SSL? Why?
 12. Would you ever propose leased line as a better approach than VPN? Why?
 13. Does a firewall always have to be a router? Why?
 14. What does it take for a host to become a firewall?
 15. What is the difference between software and hardware firewalls?



MATHEMATICAL BACKGROUND

■ A.1 INTRODUCTION ■

Some readers may like to know the mathematical background behind the various cryptography techniques. We outline a few key concepts that are essential for this purpose. However, we must point out that this is not mandatory for a reader who is not keen to study these details, instead, is content with the conceptual view of cryptography.

■ A.2 PRIME NUMBERS ■

A.2.1 Factoring

Prime numbers are very important in cryptography. A prime number is a positive integer, greater than 1, whose only factors are 1 and itself. That is, a prime number cannot be divided by any number other than 1 and itself. It should be obvious that 2, 3, 5, 7, 11, ... are prime numbers and 4, 6, 8, 10, 12, ... are not. There are an infinite number of prime numbers. Cryptography uses prime numbers heavily. Especially public key cryptography has its roots in prime number theory.

Figure A.1 shows a Java program for finding out whether any given integer is prime or not.

The reader is encouraged to modify the above program so as to automatically test for numbers from 2 to 1000 for primality. That is, the program should run a loop to test for all numbers between 2 and 1000 as to whether they are prime or not.

Two numbers are **relatively prime** when they have no factors in common other than 1. If the Greatest Common Divisor (GCD) of a and n is 1, it is written as $\gcd(a, n) = 1$. As we will note, the numbers 21 and 44 are relatively prime (because they have no factors in common), but the numbers 21 and 45 are not (because they have a factor 3 in common).

A.2.2 Euclid's Algorithm

A method of calculating the GCD of two numbers is by using the **Euclid's algorithm**. Let us write the C language representation of this algorithm, as shown in Fig. A.2.

```
// Java program to test whether a given number is prime
// Author: Atul Kahate

public class PrimeTest {
    public static void main (String [] args) {
        int numberToTest = 101;
        int m = 0;

        if (numberToTest <= 1) {
            System.out.println ("The number " + numberToTest + " is NOT prime");
            return;
        }

        for (int i=2; i<numberToTest - 1; i++) {
            m = numberToTest % i;
            if (m == 0) {
                System.out.println ("The number " + numberToTest + " is NOT prime");
                return;
            }
        }

        System.out.println ("The number " + numberToTest + " IS prime");
    }
}
```

Fig. A.1 Java program to test whether a number is prime or not

```
int gcd (int x, int y)
{
    int a;

    /* If the numbers are negative, make them positive */
    if (x < 0)
        x = -x;
    if (y < 0)
        y = -y;

    /* No point going ahead if the sum of the numbers is 0 */
    if ((x + y) == 0)
    {
        printf ("The sum of %d and %d is 0. ERROR!", x, y);
        return -1;
    }

    a = y;
    while (x > 0)
    {
        a = x;
        x = y% x;
        y = a;
    }

    return a;
}
```

Fig. A.2 C language representation of the Euclid's algorithm

Let us trace the algorithm for $x = 21$ and $y = 45$. The trace is shown in Fig. A.3.

x	y	A
21	45	NA
3	21	21
0	3	3

Fig. A.3 Trace of the Euclid's algorithm

As we can see, $\gcd(21, 45) = 3$ from the above table. By our earlier definition, therefore, 21 and 45 are not relatively prime.

A.2.3 Modular Arithmetic and Discrete Logarithms

Modular arithmetic is based on simple principles. *Modulo* is the remainder left after an integer division. For example, $23 \bmod 11 = 12$, because 12 is the remainder of the division $23 / 11$. Modular arithmetic then says that 23 and 11 are equivalent. That is, $23 \equiv 11 \pmod{12}$. In general, $a \circ b \pmod{n}$ if $a = b + kn$ for some integer k . If $a > 0$ and $0 < b < n$, then b is the remainder of the division a / n . Other names for these are: **residue** for b and **congruent** for a . The triple equal to sign (\circ) denotes **congruence**. Cryptography uses computation mod n very frequently.

Modular exponentiation is a one-way function used in cryptography. Solving it is easy. For example, consider $a^x \pmod{n}$, given the values of a , x and n . It is quite simple to solve. However, the inverse problem of modular exponentiation is that of finding the discrete logarithm of a number. This is quite tough. For instance, find x where $3^x \equiv b \pmod{17}$. As an example, if $3^x \circ 15 \pmod{17}$, then $x = 6$. For large numbers, solving this equation is quite difficult.

A.2.4 Testing for Primality

If p is an odd prime number, then the equation $x^2 \equiv 1 \pmod{p}$ has only two possible solutions, $x \equiv 1$ and $x \equiv -1$. We shall not discuss the proof of this.

A.2.5 Square Roots Modulo a Prime

If n is the result of the multiplication of two prime numbers, then the ability to find out the square root mod n is equivalent to the ability of factoring n . That is, if we know the prime factors of n , then we can easily calculate the square roots of a number mod n .

A.2.6 Quadratic Residues

If p is prime and $0 < a < p$ then a is a **quadratic residue** mod p , if:

$$x^2 \equiv a \pmod{p} \text{ for some } x$$

For instance, if $p = 7$, the quadratic residues are 1, 2 and 4, because:

$$1^2 = 1 \equiv 1 \pmod{7}$$

$$2^2 = 4 \equiv 4 \pmod{7}$$

$$3^2 = 9 \equiv 2 \pmod{7}$$

$$4^2 = 16 \equiv 2 \pmod{7}$$

$$5^2 = 25 \equiv 4 \pmod{7}$$

$$6^2 = 36 \equiv 1 \pmod{7}$$

■ A.3 FERMAT'S THEOREM AND EULER'S THEOREM ■

Two theorems that are significant to public key cryptography are **Fermat's Theorem** and **Euler's Theorem**.

A.3.1 Fermat's Theorem

This theorem states the following:

If p is prime and a is a positive integer not divisible by p then we have:

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

Let us have $a = 3$ and $p = 5$. Then, as per the above theorem, we have:

$$3^{5-1} \equiv 3^4 = 81 \equiv 1 \pmod{5}$$

Hence, the proof.

Another form of this theorem states that if p is prime and a is any positive integer then the following equation holds:

$$a^p \equiv a \pmod{p}$$

Let us consider a few examples:

1. Let $a = 3$ and $p = 5$, then we have:
 - (i) $a^p = 3^5 = 243$. Now, if we compute $243 \pmod{5}$, we will have a result of 3.
 - (ii) $a \pmod{p} = 3 \pmod{5} = 3$.
 Hence, we have: $3^5 \equiv 3 \pmod{5}$.
2. Let $a = 4$ and $p = 8$, then we have:
 - (i) $a^p = 4^8 = 65536$. Now, if we compute $65536 \pmod{8}$, we will have a result of 0.
 - (ii) $a \pmod{p} = 4 \pmod{8} = 4$.
 Now, the results of the above two steps are not matching. This is because p is not prime.
3. Let $a = 4$ and $p = 7$, then we have:
 - (i) $a^p = 4^7 = 16384$. Now, if we compute $16384 \pmod{7}$, we will have a result of 4.
 - (ii) $a \pmod{p} = 4 \pmod{7} = 4$.
 Hence, we have: $4^7 \equiv 4 \pmod{7}$.

A.3.2 Euler's Theorem

Before discussing Euler's theorem, we need to discuss the **Euler–Toient Function**. This function is written as $j(n)$, where $j(n)$ is the number of positive integers less than n and relatively prime to n .

For instance, if $n = 6$, the positive integers less than n are 1, 2, 3, 4, and 5. Of these, only 1 and 5 do not have any factors common with 6. Thus, $j(n) = j(6) = 2$. Note that the number 6 is not prime. Let us consider a prime number $n = 7$. Here, all the positive integers preceding it (i.e. 1 to 6) are relatively prime to it. In general, for a prime n , $j(n) = (n - 1)$.

Further, suppose p and q are two prime numbers. Then, for $n = pq$, we have

$$j(n) = j(pq) = j(p) \times j(q) = (p - 1) \times (q - 1)$$

Let $p = 3, q = 7$. Then, $n = p \times q = 21$.

Therefore, $\phi(n) = j(21) = \phi(3) \times \phi(7) = 2 \times 6 = 12$, where the 12 integers are 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20.

Based on this, Euler's theorem says that for every a and n , which are relatively prime, we have:

$$aj^{(n)} \equiv 1 \pmod{n}$$

Let $a = 3$ and $n = 10$. Then, we have:

$j(n) = j(10) = 4$, the 4 numbers being 1, 3, 7 and 9.

So, $aj^{(n)} = 3^4 = 81 \equiv 1 \pmod{10}$.

■ A.4 CHINESE REMAINDER THEOREM ■

The **Chinese Remainder Theorem** uses the prime factorization of a number n to solve a system of equations. In general, if the prime factors of n are $p_1 * p_2 * \dots * p_t$, then, the system of equations

$$x \bmod p_i = a_i$$

where $i = 1, 2, \dots, t$

has a unique solution x , provided $x < n$.

That is, for a number less than the product of a few primes, is uniquely identified by its residues moduli of those prime numbers.

For example, suppose we have two prime numbers as 5 and 7. Suppose that 16 is our number. Then, we have:

$$16 \bmod 5 = 1$$

$$16 \bmod 7 = 2$$

There is only one number smaller than $5 * 7$, i.e. 35, that has these residues: 16. These two residues can be used to uniquely determine the number. This can be proven by using the Java program shown in Fig. A.4.

```
// Java program to test Chinese remainder theorem basics
// Author: Atul Kahate

public class PrimeTest {
    public static void main (String [] args) {
        int k1 = 5, k2 = 7;

        for (int i=2; i<35; i++) {
            int n1 = i % k1;
            int n2 = i % k2;

            System.out.println ("Number = " + i + "Residues = " + n1 + " and " + n2);
        }
    }
}
```

Fig. A.4 Chinese remainder theorem test

Therefore, the Chinese Remainder Theorem suggests that for an arbitrary a , which is less than p , and b that is less than q (and where p and q are prime), there must be a unique number x , such that:

$$x < pq \text{ and}$$

$$x \equiv a \pmod{p} \text{ and } x \equiv b \pmod{q}$$

■ A.5 LEGENDRE SYMBOL ■

The **Legendre symbol**, written as $L(a, p)$, is defined when a is any integer and p is a prime number, whose value is greater than 2. It is equal to 0, 1 or -1 , as follows:

$$L(a, p) = 0 \quad \text{if } a \text{ is divisible by } p$$

$$L(a, p) = 1 \quad \text{if } a \text{ is a quadratic residue mod } p$$

$$L(a, p) = -1 \quad \text{if } a \text{ is not a quadratic residue mod } p$$

■ A.6 JACOBI SYMBOL ■

The **Jacobi symbol**, written as $J(a, n)$, is a general case of the Legendre symbol. It is defined for any integer a and any odd integer n . It can be defined in many ways, some of which are

1. $J(a, n)$ can be found only if n is odd
2. $J(0, n) = 0$
3. $J(a, n) = 0$, if n is prime and n divides a
4. $J(a, n) = 1$, if n is prime and a is a quadratic residue mod n
5. $J(a, n) = -1$, if n is prime and a is not a quadratic residue mod n

■ A.7 HASSE'S THEOREM ■

Hasse's Theorem says that if N is the number of points on an elliptic curve then, we have:

$$p + 1 - 2\sqrt{p} < N < p + 1 + 2\sqrt{p}$$

■ A.8 QUADRATIC RECIPROCITY THEOREM ■

The **Quadratic Reciprocity Theorem** says that if p and q are distinct prime numbers then the congruences:

$$x^2 \equiv q \pmod{p} \quad \text{and}$$

$$x^2 \equiv p \pmod{q}$$

are both solvable or are both unsolvable unless p and q leave a remainder 3 when divided by 4.

■ A.9 MASSEY–OMURA PROTOCOL ■

The **Massey–Omura Protocol** is a cryptographic protocol where the parties share an elliptic curve $E(A, B)$ over a point p , but none reveal their keys. It is significant that the order of the group $E(A, B) / \text{GF}(p)$ be known to all involved parties. Let N_p denote the order of the group.

The Massey–Omura cryptosystem is based on Shamir's three-pass protocol. Here, an encryption method is so used that after it is applied twice, the two encryptions do not need to be removed in the exact reverse of the order in which they were applied, but can be removed in any order. This allows one party to send an encrypted message, and the recipient can send it back encrypted again, and then the first party can remove her own encryption, sending it back to the recipient as if only the recipient had encrypted it.

To describe the system, assume that Bob and Alice use it.

1. Bob, and Alice, separately and secretly choose keys k_B and k_A respectively, such that $\gcd(k_A, N_p) = 1$ and $\gcd(k_B, N_p) = 1$;
2. Each secretly, and separately, compute $j_B = 1/k_B \pmod{N_p}$ and $j_A = 1/k_A \pmod{N_p}$, respectively.

Now suppose that Bob wants to securely send a message M to Alice. Then they proceed as follows through two *innings*:

First, let Q_M be a point on the curve associated with M (using the Koblitz embedding method).

Innings 1

Top: Bob computes $Q_1 = k_B * Q_M$ in $E(A, B)/\text{GF}(p)$, and sends Q_1 to Alice.

Bottom: Alice computes $Q_2 = k_A * Q_1$ in $E(A, B)/\text{GF}(p)$, and sends Q_2 to Bob.

Innings 2

Top: Bob computes $Q_3 = j_B * Q_2$ in $E(A, B)/\text{GF}(p)$ and sends Q_3 to Alice.

Bottom: Alice computes $Q_4 = j_A * Q_3$ in $E(A, B)/\text{GF}(p)$.

Now $Q_4 = Q_M$ and so Alice recovers M by reversing the Koblitz embedding.

A.10 COMPUTING THE INVERSE OF A MATRIX

(Reference: Chapter 2)

How do we compute the inverse of a matrix? It is a three-step process.

1. Replace the original elements of the matrix by the adjoint of those elements in the matrix.
2. Transpose the matrix.
3. Divide every element by the determinant of the original matrix.

For example, consider a matrix as follows:

$$\begin{array}{ccc} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{array}$$

The steps applied to this example are as follows.

1. To compute the adjoint of an element, we need to compute what is called a determinant. The determinant of a square matrix is a single number calculated by combining all the elements of the matrix. To compute the determinant, we need to:
 - (a) Eliminate the row and column in which the element is located. (In this case, 1st row and 1st column of our matrix).
 - (b) Multiply the cross-sectional values. (In this case, it means (18×19) and (21×2)).
 - (c) Subtract the results of these multiplications. (In this case, it means $(18 \times 19) - (21 \times 2) = 300$).

When we compute the adjoint of other elements, the matrix looks as follows:

$$\begin{array}{ccc} +300 & -357 & +6 \\ -313 & +313 & +0 \\ +267 & -252 & -51 \end{array}$$

2. Now we need to transpose the matrix, that is, write columns as rows. Thus, values in column 1 of the matrix (i.e. +300, -313, and +267) will now become the values in row 1, and so on. The result looks as follows:

$$\begin{array}{ccc} +300 & -313 & +267 \\ -357 & +313 & -252 \\ +6 & +0 & -51 \end{array}$$

3. The determinant of the original matrix is -939. In Hill cipher, we need to take a mod 26 of this value, i.e. $-939 \bmod 26 = -3$. But we will disregard this possibility. Therefore, the inverse of the matrix is:

$$\begin{array}{ccc} 300/-939 & -313/-939 & 267/-939 \\ -357/-939 & 313/-939 & -252/-939 \\ 6/-939 & 0 & -51/-939 \end{array}$$

As another example, consider the following matrix:

$$\begin{array}{ccc} 20 & 15 & 18 \\ 78 & 95 & 56 \\ 43 & 89 & 32 \end{array}$$

Adjoint of this matrix is

$$\begin{array}{ccc} -1944 & 1122 & -870 \\ -88 & -134 & 284 \\ 2857 & -1135 & 730 \end{array}$$

Determinant is 11226

Therefore, inverse is

$$\begin{array}{ccc} -1944/11226 & 1122/11226 & -870/11226 \\ -88/11226 & -134/11226 & 284/11226 \\ 2857/11226 & -1135/11226 & 730/11226 \end{array}$$

A.11 MATHEMATICS BEHIND CRYPTOGRAPHIC OPERATION MODES (Reference: Chapter 3)

This section describes the mathematics behind the various modes of cryptographic operations, as described in Chapter 3 of this book.

Cipher Block Chaining (CBC) Mode

In the Cipher Block Chaining (CBC) mode, each cipher text block is passed through the decryption algorithm. The result then gets XORed with the previous cipher text block to yield the original plain text block. To have a look at the mathematics, let us have:

$$C_j = E_k [C_{j-1} \text{ XOR } P_j]$$

Based on this:

$$D_k [C_j] = D_k [E_k (C_{j-1} \text{ XOR } P_j)]$$

$$D_k [C_j] = C_{j-1} \text{ XOR } P_j$$

$$C_{j-1} \text{ XOR } D_k [C_j] = C_{j-1} \text{ XOR } C_{j-1} \text{ XOR } P_j = P_j$$

To produce the first block of cipher text, we know that an Initialization Vector (IV) is used, which is XORed with the first plain text block. Hence, for decryption, the first cipher text block is XORed with the IV to get the first block of plain text.

B

Appendix B



NUMBER SYSTEMS

■ B.1 INTRODUCTION ■

A **number system** contains a set of numbers that have common characteristics. Let us discuss the common number systems that are used by humans (decimal) as well as computers (binary, octal, hexadecimal). In any number system, three aspects determine the value of each digit within a number:

1. The digit itself.
2. The position of the digit in that number.
3. The base of the number system.

The base of a number system is the number of different symbols used in that system. For example, in decimal number system the base is 10, because it uses 10 different symbols from 0 to 9.

■ B.2 DECIMAL NUMBER SYSTEM ■

As we know, the decimal number system contains 10 different symbols, 0 to 9. Therefore, its base is 10. Thus, a number in the decimal number system is actually represented by multiplying each digit by its weight and then by adding all the products. For example, the value of a number 4510 in decimal number system is calculated as shown in Fig. B.1.

Thousands position		Hundreds position		Tens position		Ones position		Total
4×10^3	+	5×10^2	+	1×10^1	+	0×1^0		--
4 x 1000	+	5 x 100	+	1 x 10	+	0 x 1		--
4000	+	500	+	10	+	0	=	4510

Fig. B.1 Representation of a decimal number

■ B.3 BINARY NUMBER SYSTEM ■

The binary number system contains only two distinct symbols (called binary digits or bits), 0 and 1. Therefore, its base is 2. We had used the increasing powers of 10 from right to left to represent decimal

numbers. Here, we use increasing powers of 2. Thus, the value of a binary number 1001 in decimal is calculated as equal to 9 as shown in Fig. B.2.

4 th position		3 rd position		2 nd position		1 st position		Total
1×2^3	+	0×2^2	+	0×2^1	+	1×2^0		--
1×8	+	0×4	+	0×2	+	1×1		--
8	+	0	+	0	+	0	=	9

Fig. B.2 Binary number representation

How can we convert a decimal number to its equivalent binary number? For that, we divide the number continuously by 2 till we get a quotient of 0. In every stage, we get 0 or 1 as the remainder. When we write these remainders in the reverse order, we get the equivalent binary number. This is as shown in Fig. B.3, for a decimal number 500.

Divisor	Quotient	Remainder
2	500	
2	250	0
2	125	0
2	62	1
2	31	0
2	15	1
2	7	1
2	3	1
2	1	1
	0	1



Fig. B.3 Decimal to binary conversion

Here, the number 500 is divided continuously by 2, each time noting down the remainder (0 or 1). Finally, when the quotient is 0, we stop the process and write down the remainders in the reverse order. As we can see, the binary equivalent of a decimal number 500 is 111110100.

■ B.4 OCTAL NUMBER SYSTEM ■

We know that the decimal number system has a base of 10 and that the binary number system has a base of 2. Similarly, the octal number system has a base of 8, as it represents 8 distinct symbols (0 to 7). The same principles, as discussed earlier are used for converting an octal number to decimal or vice versa. The only change is, now 8 is used for multiplying weights or dividing quotients successively. So, a number 432 in octal can be converted into its decimal equivalent as shown in Fig. B.4.

3 rd position		2 nd position		1 st position		Total
4×8^2	+	3×8^1	+	2×8^0		--
4 x 64	+	3 x 8	+	2 x 1		--
256	+	24	+	2	=	282

Fig. B.4 Octal to decimal conversion

Let us work backwards and convert the decimal number 282 thus obtained to see if we get back the original octal number 432, as shown in Fig. B.5. Note that we go on dividing the decimal number continuously by 8 now, until we obtain a quotient of 0. In each case, we note the remainder and in the end, write all the remainders in the reverse order to get the equivalent octal number.

Divisor	Quotient	Remainder
8	282	
8	35	2
8	4	3
	0	4

**Fig. B.5** Decimal to binary conversion

As the figure shows, the decimal number 282 is the same as octal number 432.

■ B.5 HEXADECIMAL NUMBER SYSTEM ■

Hexadecimal number system is actually a superset of the decimal number system. We know that the decimal number system has 10 distinct symbols, from 0 to 9. Hexadecimal number system has all these 10 symbols and has 6 more (A through F). Thus, the hexadecimal number system contains 16 different symbols from 0 to 9 and then A through F. The symbol A in hexadecimal is equivalent to the decimal number 10, the symbol B is equivalent to the decimal number 11 and so on, which means that the last symbol F in hexadecimal number system is equivalent of the decimal number 15.

We follow the same conversion logic as we used for binary and octal number systems. The base is now 16. Let us convert a hexadecimal number 683C into its decimal equivalent, as shown in Fig. B.6.

4 th position		3 rd position		2 nd position		1 st position		Total
6×16^3	+	8×16^2	+	3×16^1	+	$C \times 16^0$		--
6 x 4096	+	8 x 256	+	3 x 16	+	12 x 1		--
24576	+	2048	+	48	+	12	=	26684

Fig. B.6 Hexadecimal to decimal conversion

As before, let us convert this decimal number 26684 to see if we get back the hexadecimal number 683C using a decimal-to-hexadecimal conversion method, as shown in Fig. B.7.

Divisor	Quotient	Remainder
16	26684	
16	1667	C
16	104	3
16	6	8
	0	6



Fig. B.7 Decimal to hexadecimal conversion

We can see that we indeed obtain 683C in hexadecimal as equivalent of the decimal number 26684.

■ B.6 BINARY NUMBER REPRESENTATION ■

From the context of computers and data communications, the binary number system is most important, because it is used inside the computers to represent any alphabet, number, or symbol. Therefore, let us discuss some aspects related to binary numbers.

Unsigned Binary Number

All binary numbers are unsigned by default, which means that they are positive. This is true in the case of decimal numbers as well. When we write a decimal number 457, we mean that it is implicitly (+)457. Let us now try to figure out the maximum number representation capabilities of the binary number system. As before, let us take the simple example of decimal number system.

Suppose we have a single slot, where we can store one decimal digit. How many different values it can store? Of course, it can store a digit between 0 and 9, thus 10 different values. Instead, if we decide to store a binary digit there, how many different values can it take? Of course, it can take a value of 0 or 1, thus 2 different values, where 1 is the maximum.

Now let us assume that we have two slots. If we decide to write decimal digits in each of them, then we can store a minimum number of 00 and a maximum number of 99 in decimal in these slots. Similarly, in binary, we can write 00 as the minimum number and 11 as the maximum number. Since binary 11 is 3 in decimal, the maximum number that we can in binary is actually decimal 3.

If we note, a pattern is emerging. Each new slot can accommodate the maximum digit in a given number system. Thus, if we have three slots, we can have the maximum decimal number as 999 and the maximum binary number as 111 (which is 7 in decimal).

Let us tabulate these results only for binary numbers, as shown in Fig. B.8. We shall continue the list, as we are observing some commonalities as shown in the last column.

The last column indicates how we can find out the maximum number possible in any storage allocated for binary numbers. It is simply 2 raised to the number of bit positions allocated minus 1. Thus, when we have 8 slots or 8 bit positions, we can represent decimal numbers from 0 (minimum number) to 255 (maximum number), a total of 256 different numbers. Since a computer *byte* generally contains 8 bits, a byte can represent one of the 256 distinct values (i.e. a number between 0 and 255).

Available slots (bit positions)	Maximum number in binary	Maximum number in decimal	Another way to represent the same number
1	1	1	$2^1 - 1$
2	11	3	$2^2 - 1$
3	111	7	$2^3 - 1$
4	1111	15	$2^4 - 1$
5	11111	31	$2^5 - 1$
6	111111	63	$2^6 - 1$
7	1111111	127	$2^7 - 1$
8	11111111	255	$2^8 - 1$

Fig. B.8 Maximum binary numbers per bit positions allowed

This is how unsigned binary numbers are represented.

Signed Binary Numbers

When we want to represent signed binary numbers, we need an extra slot. This is similar to decimal number system. Suppose we want to store a decimal number -89 . Then we need three slots—two would not be enough. The only difference is that in case of the binary number system, even the sign is represented by a 0 (indicates the + sign) and 1 (indicates the – sign). Also, the leftmost bit represents the sign.

Thus, if we have a number 110, we are told that it is a signed number, we must interpret the leftmost bit (i.e. 1) as its sign, which is negative. Thus, only the two remaining bits (i.e. 10) should be considered as its value, which is 2 in decimal. When we append the negative sign, 110 in binary becomes -2 in decimal. This may sound very confusing. How do we know if we should treat 110 a signed or unsigned? If unsigned, the leftmost 1 is data, otherwise it is sign. But how do we know this fact? Well, it depends on the context. Unless we know whether we are dealing with unsigned binary numbers or signed binary numbers, we would simply not know this!



INFORMATION THEORY

■ C.1 INTRODUCTION ■

Claude Shannon first published the modern **Information Theory** in 1948. We discuss only the key aspects of this theory here.

■ C.2 ENTROPY AND UNCERTAINTY (EQUIVOCATION) ■

Information theory defines the **amount of information** as the minimum number of bits required to encode all the possible meanings of a message, given that all the possible meanings have an equal likelihood of occurring. For instance, to record the months in a year, we need 4 bits, as follows:

0000	January
0001	February
0010	March
0011	April
0100	May
0101	June
0110	July
0111	August
1000	September
1001	October
1010	November
1011	December

We say that the **entropy** of the message is slightly less than 4. We say *slightly less*, because the bit patterns 1100, 1101 and 1111 remain unused.

■ C.3 PERFECT SECRECY ■

A cryptosystem wherein the cipher text gives absolutely no information about the plain text (except its size) achieves **perfect secrecy**. According to Shannon, this is possible only if the number of possible

keys is greater than or equal to the number of possible messages. That is, the key must be equal to or longer than the message itself and no key should be reused. Thus, one time pad is the only candidate for perfect secrecy.

■ C.4 UNICITY DISTANCE ■

Unicity distance is the approximation of the amount of cipher text so that the sum of the real information (entropy) in the corresponding text, plus the entropy of the encryption key equals the number of cipher text bits used. Furthermore, cipher texts longer than this distance are fairly certain to decrypt to only one plain text. On the other hand, cipher texts shorter than this distance generally have multiple, equally valid decryptions. Therefore, they are more secure, as the cryptanalyst has to pick up the correct one.



REAL-LIFE TOOLS

■ D.1 INTRODUCTION ■

This appendix mentions some tools available on the Internet, which can aid in achieving security at the network or the Internet layer. The readers are advised to ensure that the concerned copyright/export rules are not violated in any manner. Also, the reader should first verify that the tool is right, suitable and not buggy.

■ D.2 AUTHENTICATION TOOLS ■

1. TIS International Firewall Toolkit (FWTK), available at <ftp://ftp.tis.com/pub/firewalls/toolkit/>
→ This is a useful tool for authentication, access control, proxy services, etc.
2. Kerberos
→ We have discussed this in detail. Have a look at <ftp://athena-dist.mit.edu/pub/kerberos> for more details.

■ D.3 ANALYSIS TOOLS ■

COPS Available at <ftp://coast.cs.purdue.edu/pub/tools/unix/cops>, it is developed by Dan Farmer. COPS stands for Computer Oracle and Password System. It checks Unix systems for common security problems.

Tiger Developed by Doug Schales, it is available at <ftp://coast.cs.purdue.edu/pub/tools/unix/tiger>. It is a set of scripts that scan a Unix system for problems.

SATAN Developed by Wietse Venema and Dan Farmer, this is available at <http://www.fish.com/~zen/satan/satan.html>, and stands for Security Administrator Tool for Analyzing Networks.

■ D.4 PACKET FILTERING TOOLS ■

A tool called as ipfilter, available at <http://coombs.anu.edu.au/~avalon>, is a TCP/IP filtering system for Unix.



WEB RESOURCES

■ E.1 INTRODUCTION ■

This appendix lists some of the good resources on the Internet, related to security topics. As before, the reader is requested to validate the accuracy and intentions of the contributors.

■ E.2 MAILING LISTS ■

List Name	How to join/URL	Details
bugtrq	Join by sending a mail with subject “subscribe bugtraq” to bugtraq-request@securityfocus.com , or visit www.securityfocus.com .	Contains list of security items, bug details and solutions.
CERT-advisory	Join by sending a mail to cert-advisory-request@cert.org , or visit www.cert.org .	Contains details of new security holes and fixes.
Firewalls	Join by sending a mail with subject “subscribe firewalls” to firewalls-request@lists.gnac.net , or visit www.lsits.gnac.net/firewalls .	Contains details such as the design, construction, maintenance, issues related to firewalls.

■ E.3 USENET GROUPS ■

Usenet Group Name	Details
alt.security	Discussions about computer and network security.
comp.admin.policy	Discussions about administrative policy issues, including security.
comp.protocols.tcp-ip	Discussions about TCP/IP and security issues therein.
comp.security.misc	Miscellaneous discussions about security topics and concerns.
comp.security.firewalls	Discussions about firewalls.
comp.virus	Discussions about computer viruses.
netscape.public.mozilla.security and netscape.public.mozilla.crypto	Details about JavaScript and SSL security issues.
sci.crypt	Miscellaneous security discussions.

■ E.4 USEFUL URLs ■

Site Name	URL
CERIAS	www.cerias.purdue.edu
CIAC	www.ciac.org/ciac
DigiCrime	www.digicrime.com
FIRST	www.first.org
IETF	www.ietf.org
Mozilla	www.mozilla.org
NIST	www.csrc.ncsl.nist.gov
Radius	www.crypto.radiusnet.net
RSA	www.rsasecurity.com
openSSL	www.openssl.org
W3C	www.w3c.org
Telstra	www.telstra.com
VeriSign	www.verisign.com

■ E.5 LIST OF IMPORTANT RFCS ■

An RFC (Request For Comment) is a formal specification regarding a particular technology or protocol. It goes through several reviews and is quite useful to go through, once we have a basic understanding of the technology/protocol. The following table lists some of the important RFC numbers, their brief description. The actual contents of the RFCs can be obtained from www.ietf.org.

RFC Number	Subject
1847	Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted
1958	Architectural Principles of the Internet
2119	Key words for use in RFCs to Indicate Requirement Levels
2268	A Description of the RC2(r) Encryption Algorithm
2311	S/MIME Version 2 Message Specification
2315	PKCS #7: Cryptographic Message Syntax Version 1.5
2409	The Internet Key Exchange (IKE)
2437	PKCS #1: RSA Cryptography Specifications Version 2.0
2437	PKCS #1: RSA Cryptography Specifications Version 2.0
2459	Internet X.509 Public Key Infrastructure Certificate and CRL Profile
2510	Internet X.509 Public Key Infrastructure Certificate Management Protocols
2510	Internet X.509 Public Key Infrastructure Certificate Management Protocols

2511	Internet X.509 Certificate Request Message Format
2527	Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework
2560	X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP
2587	Internet X.509 Public Key Infrastructure LDAPv2 Schema
2630	Cryptographic Message Syntax
2712	Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)
2807	XML Signature Requirements
2817	Upgrading to TLS Within HTTP/1.1
2818	HTTP—Over TLS
2875	Diffie-Hellman Proof-of-Possession Algorithms
2985	PKCS #9: Selected Object Classes and Attribute Types Version 2.0
2986	PKCS #10: Certification Request Syntax Specification Version 1.7
3029	Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols
3039	Internet X.509 Public Key Infrastructure Qualified Certificates Profile
3075	XML Signature Syntax and Processing
3161	Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)
draft-freier-ssl-version3-02.txt	The SSL Protocol Version 3.0
draft-ietf-pkix-ocspv2-02.txt	Online Certificate Status Protocol, version 2 draft-ietf-pkix-ocspv2-02.txt
draft-ietf-stime-ntpauth-02.txt	Public key Cryptography for the Network Time Protocol Version 2
draft-ietf-tls-56-bit-ciphersuites-01.txt	56-bit Export Cipher Suites For TLS
draft-ietf-tls-ecc-01.txt	ECC Cipher Suites for TLS



A BRIEF INTRODUCTION TO ASN, BER, DER

■ F.1 INTRODUCTION ■

Computers and networks vary from each other vastly in many respects, such as their architectures, operating systems, etc. This leads to a lot of incompatibilities, such as different ways of representing numbers, varying sizes of data types, etc. For instance, the x86 Intel chips order the bits in a byte from the right to the left, whereas Motorola does the opposite. Many computers now use ASCII, but still quite a few, mainly IBM Mainframes, still use the EBCDIC coding mechanism. Therefore, when applications want to communicate with other applications on different computers/networks, there could be a great amount of incompatibility and confusion. This problem is resolved by the Open Systems Interconnection (OSI) model, which governs the protocols to be used by computers while communicating with each other.

This is relevant to cryptography/security. Imagine two computers communicating with each other. Assume that the sending computer encrypts some text and sends it to the receiving computer. In order that the receiving computer is able to decrypt the text successfully, it must know not only the encryption algorithm and key used by the sending computer, but also the internal data format of the text. What if, for instance, the sending computer uses ASCII, but the receiving computer uses EBCDIC? Even after a successful decryption, the receiving computer would not be able to derive any meaning of the decrypted data. To avoid such problems, the sending and receiving computers must use a uniform *language* for the data being exchanged. This *language* is nothing but ASN.1, as we shall see. If the sending and the receiving computer both use ASN.1, then there would be no confusion.

■ F.2 ABSTRACT SYNTAX NOTATION (ASN.1) ■

Abstract Syntax Notation 1 (ASN.1) is used to describe the high-level format/structure of data to be transferred between the Application Layer and the Presentation Layer of the Open Systems Interconnection (OSI). Thus, the sending computer would transform the data to be sent into ASN.1 syntax and send it to the destination computer. Upon receiving the data, the destination computer would transform the data from the ASN.1 syntax back to the native format, make it available to the actual application. This concept is illustrated in Fig. F.1. We must point out that this is technically not 100% correct, as

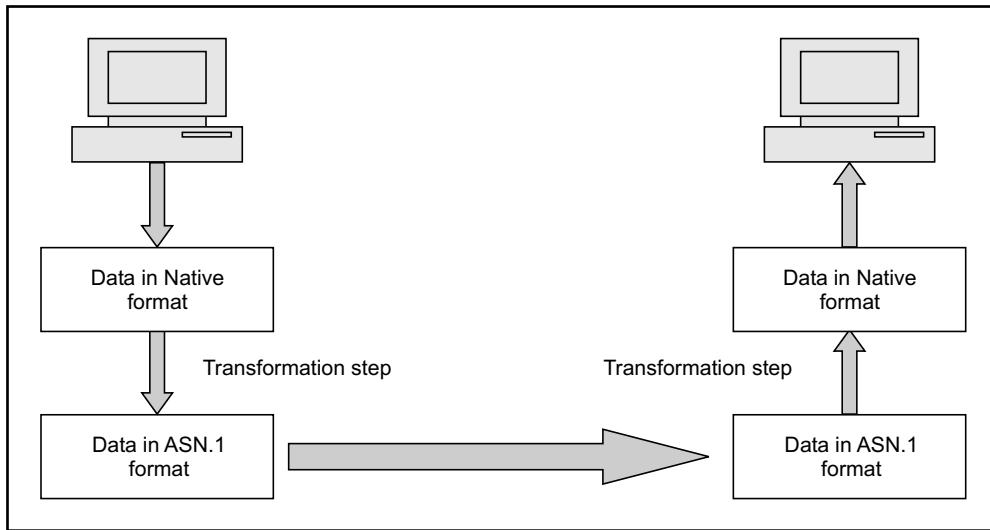


Fig. F.1 ASN.1 concept

ASN.1 data is not sent as it is, but is first transformed into binaries (as we shall study soon). But for the moment, we shall ignore this finer detail, worry only about the concept.

ASN.1 is meant to provide a mechanism using which the Presentation Layer can use a single standard encoding mechanism to exchange any arbitrary data structure with other computer systems, while the Application Layer can map this standard encoding into any type of representation or language that is appropriate for the end user. ASN.1 does not describe the content, meaning, or structure of the data, but only the way in which it is specified and encoded.

ASN.1 is defined jointly by ISO/IEC along with ITU. ASN.1 defines two important aspects: **type** and **value**. The types can be **primitive** or **constructed**.

Basic data types, such as strings (actual keyword is `PrintableString`), time (actual keyword is `GeneralizedTime`) are some of the primitive types.

Some other primitive data types are defined in Fig. F.2.

Data type	Size	Description
Integer	4 bytes	An integer value between 0 and $2^{32} - 1$
String	Variable	Zero or more characters
IP Address	4 bytes	32-bit IP address of a computer

Fig. F.2 Some ASN.1 primitive data types

Constructed types are created out of the primitive or other constructed types (similar to structures in the C language, or record types in COBOL). Examples of constructed types are SET and SEQUENCE.

Some constructed data types are listed in Fig. F.3.

Data type	Description
Sequence	Combination of similar/dissimilar data types—similar to structures in the C language, or records in any programming language.
Sequence of	Combination of same data types of simple or sequence types—similar to the concept of arrays in programming languages.

Fig. F.3 ASN.1 constructed data types

■ F.3 ENCODING WITH BER AND DER ■

The encoding rules define how an ASN.1 value should be encoded into a stream of octets (bytes). Two sets of rule have gained prominence: **Basic Encoding Rules (BER)** and **Distinguished Encoding Rules (DER)**. BER provides one or more ways to represent values of each ASN.1 object into an octet. DER provides a unique way to do so. That is the major difference between them. The way this process works is as shown in Fig. F.4, which is a logical extension of the ASN.1 formatting.

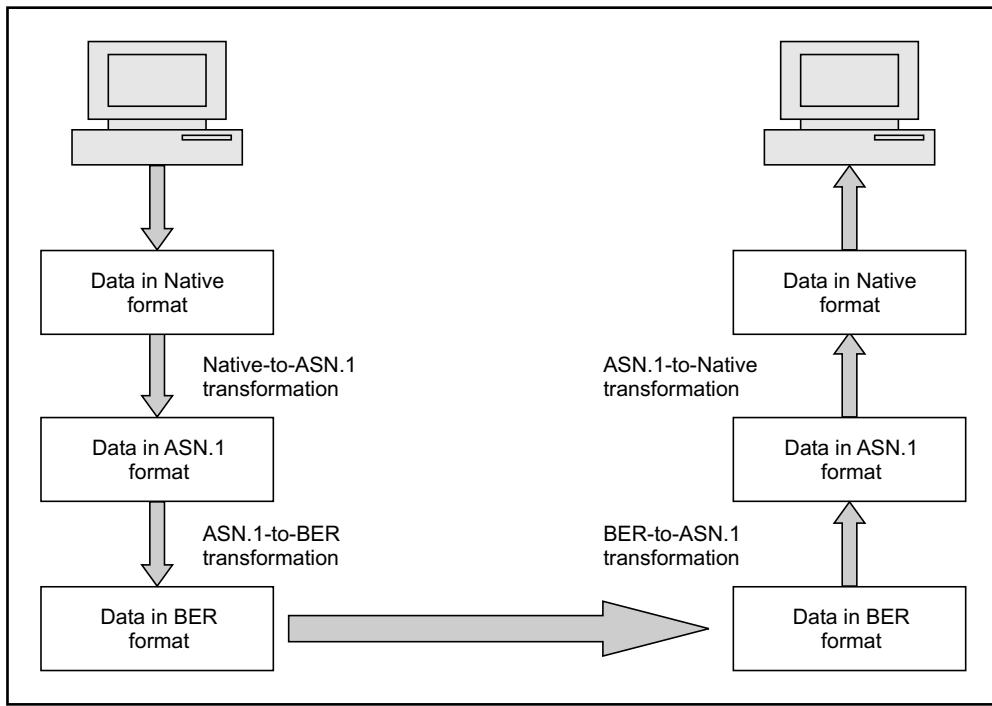


Fig. F.4 BER concept

This idea can be best illustrated with an example. Let us consider a digital certificate, which is stored in PKCS#7 format. We shall see how this looks like, in the ASN.1 format (which is understandable to the human eyes) and also in the BER format (which is pure binary, and therefore, understandable to computers only). Figure F.5 shows how the digital certificate looks in the ASN.1 format and Fig. F.6 shows how it looks in the BER format (only a portion of it is shown for brevity).

```

Object: SEQUENCE[C] = 6 elements
    Object: SET[C] = 1 elements
        Object: SEQUENCE[C] = 2 elements
            Object: OBJECT ID = countryName
            Object: PrintableString = "IN"
    Object: SET[C] = 1 elements
        Object: SEQUENCE[C] = 2 elements
            Object: OBJECT ID = organizationName
            Object: PrintableString = "Personal"
    Object: SET[C] = 1 elements
        Object: SEQUENCE[C] = 2 elements
            Object: OBJECT ID = organizationalUnitName
            Object: PrintableString = "security"
    Object: SET[C] = 1 elements
        Object: SEQUENCE[C] = 2 elements
            Object: OBJECT ID = userid
            Object: PrintableString = "Atul"
    Object: SET[C] = 1 elements
        Object: SEQUENCE[C] = 2 elements
            Object: OBJECT ID = commonName
            Object: PrintableString = "Atul Kahate"
    Object: SET[C] = 1 elements
        Object: SEQUENCE[C] = 2 elements
            Object: OBJECT ID = emailAddress
            Object: IA5String = "akahate@indiatimes.com"

```

Fig. F.5 Portion of a digital certificate stored in ASN.1 format

```

E0NlcnRpZmljYXRRIIE1hbmfFnZXlwHhcNMIDlwMjExMTIyNjM0WhcNMDMwMjExMTIy
NjM0WJB5MQswCQYDVQQGEwJJTjEOMAwGA1UEChMFaWZsZXgxDDAKBgNVBAAsTA3Br
aTEWMBQGCgmSJomT8ixkAQETBnNlcnZlcjETMBEGA1UEAxMKZ2lyaNlcnZlcjEf
MB0GCSqGSIb3DQEJARYQc2VydmyQGlmbGV4LmNvbTCBnzANBgkqhkiG9w0BAQEF
AAOBjQAwgYkCgYEArIsLROwrlrVxu/Mie8q0rUCQ5GtqMBWeJtuJM0vn2Qk5XaWc
8y1nJ/zc90v7qSx33X/sW5aRJph1ApOvPARQhK9PAyPhCcCIUEOvUYnxFmu8YE9U
Tz2p9wiUkgN+Uehlr2EMWDRaB7wctb4eyuNmyeUlryNy2d8ujDxP2ls1CzHkCAwEA
AaOBgzCBgDARBglhgkBhvhCAQEEBAMCBAwDgYDVR0PAQH/BAQDAgXgMB0GA1Ud

```

Fig. F.6 Portion of a digital certificate stored in BER format

As we have noted, what happens is that the digital certificate is first transformed into the ASN.1 format by the sending computer, and then sent as BER format (because computers can deal only with binaries). The receiving computer receives the BER data, transforms it back into ASN.1, and gives it back to the application program.

When sending a variable, the BER standard specifies what are its data type (in binary format, corresponding to the data types discussed earlier), length and value. That is, the BER standard specifies that each data item that is to be sent should be encoded to contain three pieces or fields of information: tag, length, and value, which are described in Fig. F.7.

BER field	Description
Tag	This one-byte field defines the type of data, which contains three sub-fields, namely, class, format and number. This is simply the numeric representation (encoding) of the data types discussed earlier. For instance, an integer is represented as 00000010, string is represented as 00000100, etc.
Length	This field specifies the length of the item in binary form. So, if this field contains 11, the length of the data item defined by the tag field is 3.
Value	This field contains the actual value of the data.

Fig. F.7 BER format contents



REFERENCES

- Adams, Carlisle, *Understanding Public Key Infrastructure*, New Riders publishing, 1999.
- Ahuja, Vijay, *Network and Internet security*, AP Professional, 1996.
- Amor, Daniel, *The E-business (r)evolution*, Prentice Hall 2000.
- Anderson, Ross *Security Engineering*, John Wiley and Sons. 2001.
- Atkins, Derek et al., *Internet Security Professional Reference (2nd Edition)*, Techmedia, 1997.
- Black, Uyless, *Internet Security Protocols*, Pearson Education Asia, 2000.
- Burnett, Steve and Paine, Steven, *RSA security's official guide to cryptography*, Tata McGraw-Hill, 2001.
- Comer, Douglas., *Internetworking with TCP/IP, Volume 1*, Prentice Hall, India, 1999.
- Comer, Douglas, *Computer Networks and Internets*, Prentice Hall, 2000.
- Comer, Douglas, *The Internet Book*, Prentice Hall, India, 1999.
- Davis, Carlton, *IPSec-Securing VPNs*, Tata McGraw-Hill, 2001.
- Dennin, Dorothy, *Information Warfare and Security*, Pearson Education Asia, 1999.
- Dournaee, Blake, *XML Security*, Tata McGraw-Hill, 2002.
- Forouzan, Behrouz, *Data Communications and Networking*, Tata McGraw-Hill, 2002.
- Forouzan, Behrouz, *TCP/IP* Tata McGraw-Hill, 2002.
- Garfinkel, Simson and Spafford, Gene, *Web Security, Privacy and Commerce*, O'Reilly, 2002.
- Godbole, Achyut and Kahate, Atul, *Web Technologies-TCP/IP to Internet Application Architectures*, Tata McGraw-Hill, 2003.
- Gralla, Preston, *How the Internet Works*, Techmedia, 2000.
- Hall, Eric, *Internet Core Protocols*, O'Reilly, 2000.

- Howard, Michael and LeBlanc, David, *Writing secure code*, WP Press, 2002.
- Kalakota, Ravi, *Frontiers of Electronic Commerce*, Addison Wesley, 2000.
- Kaufman, Charlie et al., *Network Security*, Pearson Education Asia, 2002
- Kosiur, David, *Understanding Electronic Commerce*, Microsoft Press, 1997.
- Krutz, Ronald and Dean Vines, Russell, *The CISSP Prep Guide*, John Wiley and Sons, 2001.
- Many, *Professional WAP*, Wrox, 2000.
- Minoli, Daniel and Minoli, Emma, *Web Commerce Technology Handbook*, Tata McGraw-Hill, 1999.
- Moulton, Pete, *The Telecommunications Survival Guide*, Pearson Education, 2001.
- Naik, Dilip, *Internet Standards and Protocols*, Microsoft Press, 2001.
- Nanavati, Samir et al., *Biometrics*, Pearson Education Asia, 2002.
- Nash, Andrew et al., *PKI-Implementing and Managing E-Security*, Tata McGraw-Hill, 2000.
- Oaks, Scott, *Java Security*, O'Reilly, 2001.
- Orfali, Robert, Harkey Dan, Edwards, Jerry, *The essential client/server survival guide*, Galgotia, 2000.
- Pistoia, Marco et al., *Java 2 Network Security*, Pearson Education Asia, 2001.
- Ramachandran, Jay, *Designing Security Architecture Solutions*, John Wiley and Sons, 2002.
- Richard Stevens, W, *TCP/IP illustrated, Volume 1*, Addison Wesley, 1999.
- Schneider, Gary and Perry, James, *Electronic Commerce*, Thomson Learning, 2001.
- Schneier, Bruce, *Applied Cryptography*, John Wiley and Sons, 2001.
- Scott, Charlie et al., *Virtual Private Networks*, O'Reilly, 2000.
- Smith, Richard, *Internet Cryptography*, Pearson Education Asia, 1999.
- Stallings, William, *Network Security Essentials*, Pearson Education Asia, 2002.
- Stallings, William, *Cryptography and Network Security*, Pearson Education Asia, 2000.
- Swaminatha, Tara and Elden, Charles, *Wireless Security and Privacy*, Pearson Education Asia, 2003.
- Tanenbaum, Andrew, *Computer Networks*, Prentice Hall, India, 1995.
- Tanenbaum, Andrew, *Modern Operating Systems*, Pearson Education, 2002.
- Winfield Treese and Stewart, Lawrence, *Designing Systems for Internet Commerce*, Addison Wesley, 1999.
- Zwicky, Elizabeth et al., *Building Internet firewalls*, O'Reilly, 2000.



INDEX

- 1-factor authentication, 357
- 2-factor authentication, 357
- 3-D Secure, 296-299
- 3G security, 324
- Access Control List (ACL), 11
- Access control, 8
- Active attacks, 15
- Advanced Encryption Standard (AES), 130-141
- Algorithm mode, 80
- Algorithm type, 80
- Algorithm, 51
- Application gateway, 429-431
- Asymmetric key cryptography, 53, 158
- Audit record, 462
- Authentication Header (AH), 443, 448-452
- Authentication token, 356-366
- Authentication, 8, 342
- Availability, 8
- Based CRL, 228
- Bastion host, 431
- Bell-LaPadula model, 6
- Biometric authentication, 372-374
- Birthday attack, 167
- Block cipher, 81
- Blowfish, 127-130
- Book cipher, 51
- Brute-force attack, 38
- Bucket-brigade attack, 60
- Caesar cipher, 36
- Certificate Revocation List (CRL), 226
- Certificate revocation, 225
- Certificate Signing Request (CSR), 212
- Certificate-based authentication, 366-372
- Certification Authority (CA), 206
- Chain of trust, 220
- Chaining mode, 82
- Chosen cipher-text attack, 71
- Chosen plain-text attack, 70
- Chosen-text attack, 71
- Cipher Block Chaining (CBC), 84, 85
- Cipher Feedback (CFB), 84, 87
- Cipher text, 35
- Cipher-text only attack, 68
- Clear text, 33
- Cloud security, 414
- Collision, 167
- Confidentiality, 8
- Confusion, 83
- Congestion attack, 456
- Counter (CTR), 89
- Cross-certification, 224
- Cryptanalysis, 32, 38
- Cryptographic toolkit, 410
- Cryptography, 32
- Cryptology, 33
- Cycling attack, 155
- Data Encryption Standard (DES), 94-108
- Decryption algorithm, 51
- Decryption, 51
- Delta CRL, 227
- Demilitarized Zone (DMZ), 439
- Denial of Service (DOS), 16

- Dictionary attack, 238
Diffie-Hellman Key Exchange, 56
Diffusion, 83
Digital certificate, 205
Digital envelope, 159
Digital Signature Algorithm (DSA), 189
Digital Signature Standard (DSS), 189
Digital signature, 162
DNS spoofing, 25
DNSSec, 27
DomainKeys Identified Mail (DKIM), 318
Double DES, 104-107
- Electronic Code Book (ECB), 84
ElGamal cryptography, 156
Elgamal Digital Signature, 194
Email security, 299
Encapsulating Security Payload (ESP), 444, 452-455
Encryption algorithm, 51
Encryption, 51
eXtensible Access Control Markup Language (XACML), 412
- Fabrication, 9, 14
Factorization attack, 156
Firewall, 423-440
- GSM security, 322
- Hash, 165
Hash-based Message Authentication Code (HMAC), 185-189
Hill cipher, 45
Homophonic substitution cipher, 40
Honeypot, 464
- Integrity, 8
Interception, 9, 14
International Data Encryption Algorithm (IDEA), 108-115
Internet Association and Key Management Protocol (ISAKMP), 455, 457-458
Internet Key Exchange (IKE), 446
Interruption, 11, 15
Intruder, 461
Intrusion Detection Systems (IDS), 463
- Intrusion, 461-464
IP security (IPSec), 440, 458
- Java Cryptography Architecture (JCA), 401-405
Java Cryptography Extension (JCE), 401, 405-407
- Kerberos, 374-380
Key agreement, 148
Key Distribution Center (KDC), 380-381
Key exchange, 54
Key pair, 63
Key range, 65
Key size, 66
Key wrapping, 159
Key, 51
Knapsack algorithm, 193
Known plain-text attack, 70
- Low decryption exponent attack, 156
- Macro virus, 19
Man-in-the-middle attack, 60
Masquerade, 16
MD5, 169-177
Meet-in-the-middle attack on DES, 105
Message Authentication Code (MAC), 183-184
Message digest, 164
Modification, 10, 15
Mono-alphabetic cipher, 39
Multi-factor authentication, 357
- Network Address Translation (NAT), 431
Non-repudiation, 8
- One-time pad, 50
Online Certificate Status Protocol (OCSP), 231
Output Feedback (OFB), 84, 88
- Packet filter, 426-429
Packet sniffing, 23
Packet spoofing, 23
Passive attacks, 15
Password Based Encryption (PBE), 238
Password encryption, 352
Password, 343-356
Pharming, 25

- Phishing, 23
PKIX model, 236
Plain text, 33
Plain-text attack, 155
Playfair cipher, 42
Polyalphabetic substitution cipher, 40
Polygram substitution cipher, 40
Pretty Good Privacy (PGP), 305-313
Privacy Enhanced Mail (PEM), 302-305
Private key management, 234
Private key, 63
Product cipher, 36
Proxy server, See *Application gateway*
Public key cryptography Standards (PKCS), 212, 238
Public key cryptography, 149
Public Key Infrastructure (PKI), 204
Public key, 63

Rail-fence technique, 47
RC4, 116-118
RC5, 118-126
Reference monitor, 6
Registration Authority (RA), 208
Replay attack, 16, 391
Revealed decryption exponent attack, 156
Root CA, 220
RSA algorithm, 149, 151-156

S/MIME, 313-318
Screened host firewall, Dual-homed bastion, 438
Screened host firewall, Single-homed bastion, 437
Screened subnet firewall, 438
Secret key, 63
Secure Electronic Transaction (SET), 283-296
Secure Hyper Text Transfer Protocol (SHTTP), 282
Secure Socket Layer, 271-282
Security Assertion Markup Language (SAML), 412
Security Association (SA), 447
Security handshake pitfalls, 381-390

Security policy, 7
Self-signed certificate, 223
Session hijacking, 391
SHA-1, 177-180
SHA-3, 183
SHA-512, 180-183
Short-message attack, 155
Simple Certificate Validation Protocol (SCVP), 231
Single Sign On (SSO), 390
Steganography, 64
Stream cipher, 81
Substitution, 35
Symmetric key cryptography, 53, 92, 158

Transport Layer Security (TLS), 282
Transport mode, 446
Transposition, 35
Triple DES, 107-108
Trojan Horse, 19
Trusted system, 6
Trusted third party, 63
Tunnel mode, 444

Unconcealed message attack, 156

Vernam cipher, 50
Vigenere cipher, 40
Virtual Private Network (VPN), 458-461
Virus, 18

Web Services security, 411
Wi-Fi Protected Access (WPA), 330
Wired Equivalent Privacy (WEP), 327
Wireless Application Protocol (WAP) security, 319
Worm, 19
WS-Security, 411

XML digital signature, 245, 412
XML encryption, 245, 412
XML Key Management Specification (XKMS), 248