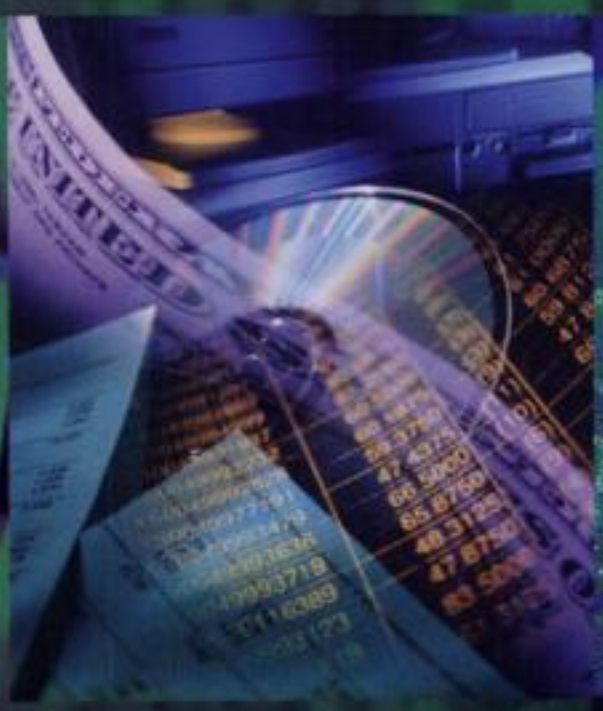
Cryptography and Network Security



ATUL KAHATE

Tata McGraw-Hill

Tata McGraw-Hill

Eighth reprint 2006
RALQCDDKRZQCA

This edition can be exported from India only by the publishers,
Tata McGraw-Hill Publishing Company Limited

ISBN 0-07-049483-5

Published by Tata McGraw-Hill Publishing Company Limited,
7 West Patel Nagar, New Delhi 110 008, typeset in Baskerville
at Tej Composers, WZ 391, Madipur, New Delhi 110 063, and printed at
Sai Printo Pack, Okhla Industrial Area, Phase II, New Delhi 110 020

Cover Design: B&M Design, Noida

Cover: SDR

The McGraw-Hill Companies

Contents

Chapter 1

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 fundamental point: 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, 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 of the types of attacks. There are certain theoretical concepts associated with attacks, and there is a practical side to it as well. We discuss all these aspects.

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

1.2 THE NEED FOR SECURITY

Most initial 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 id 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 permissions

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 that the basic security measures were not quite enough.

Furthermore, the Internet took the world by storm, and 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, and payment details such as credit card information travel across the Internet to the merchant's 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 receives the credit card details and validates them so as to process the order and later obtain payments, the merchant 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 300,000 such cards, the total cost of such an attack is about $6 million! Had the merchant in the example just discussed employed proper security measures, he would have saved so much money and bother.

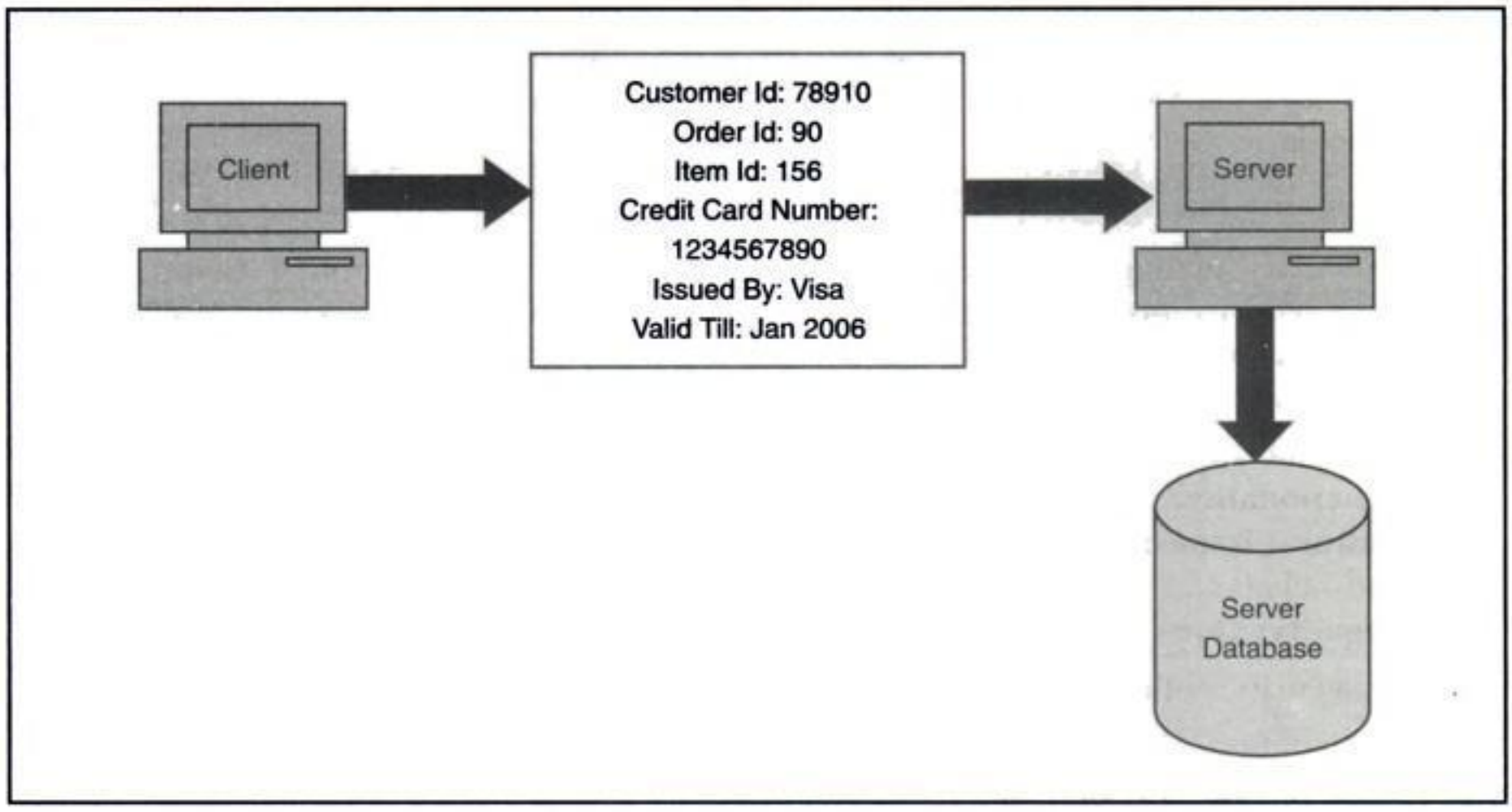


Fig. 1.1 *Example of information traveling from a client to a server over the Internet*

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 an another example in 1999, a Swedish hacker broke into Microsoft's Hotmail Website, and created a mirror site. This site allowed anyone to enter any Hotmail user's email id, and read 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 figure 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!

1.3 SECURITY APPROACHES

1.3.1 Security Models

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

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

- **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.

1.3.2 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 efforts does this security implementation cost?
- **Functionality:** What is the mechanism of providing security?
- **Cultural issues:** Does the policy gel well with 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, 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 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. Example of compromising the confidentiality of a message is shown in Fig. 1.2. Here, the user of computer A sends a message to the user of computer B. (Actually, from here 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 user A, B, etc.). Another user C gets access to this message, which is not desired, and therefore, defeats the purpose of confidentiality. 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 as **interception.**

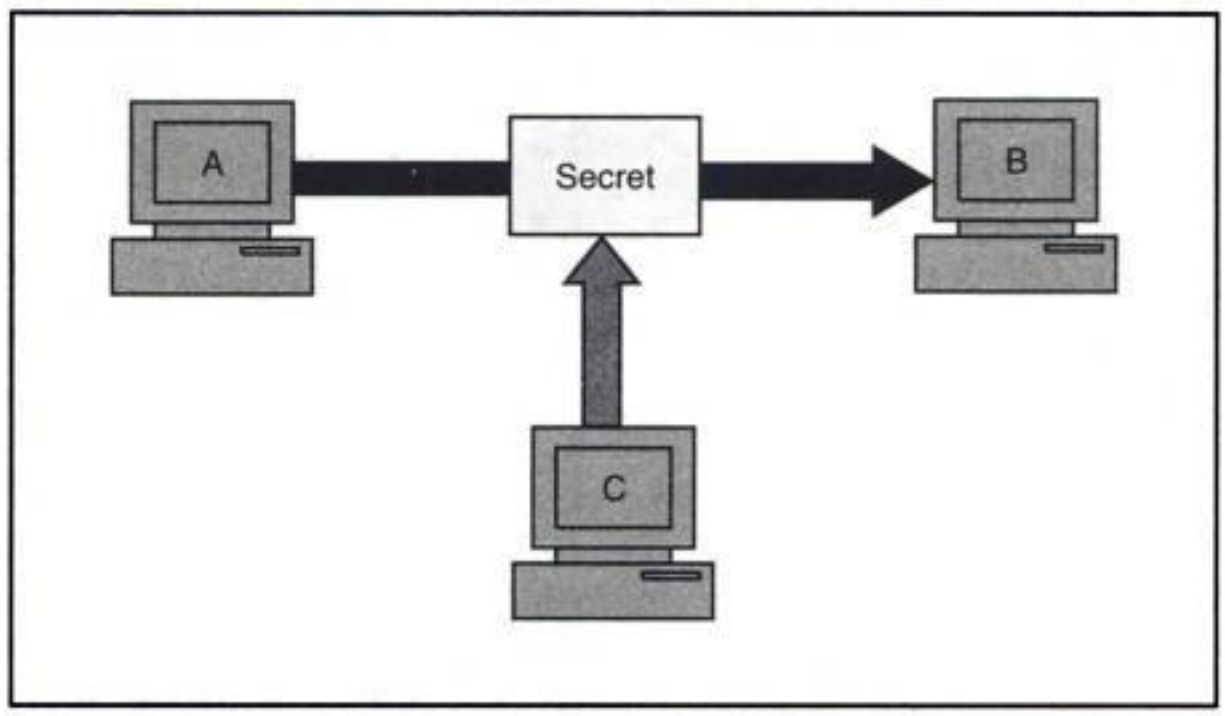


Fig. 1.2 *Loss of confidentiality*

Note 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 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. 1.3. This type of attack is called as **fabrication**.

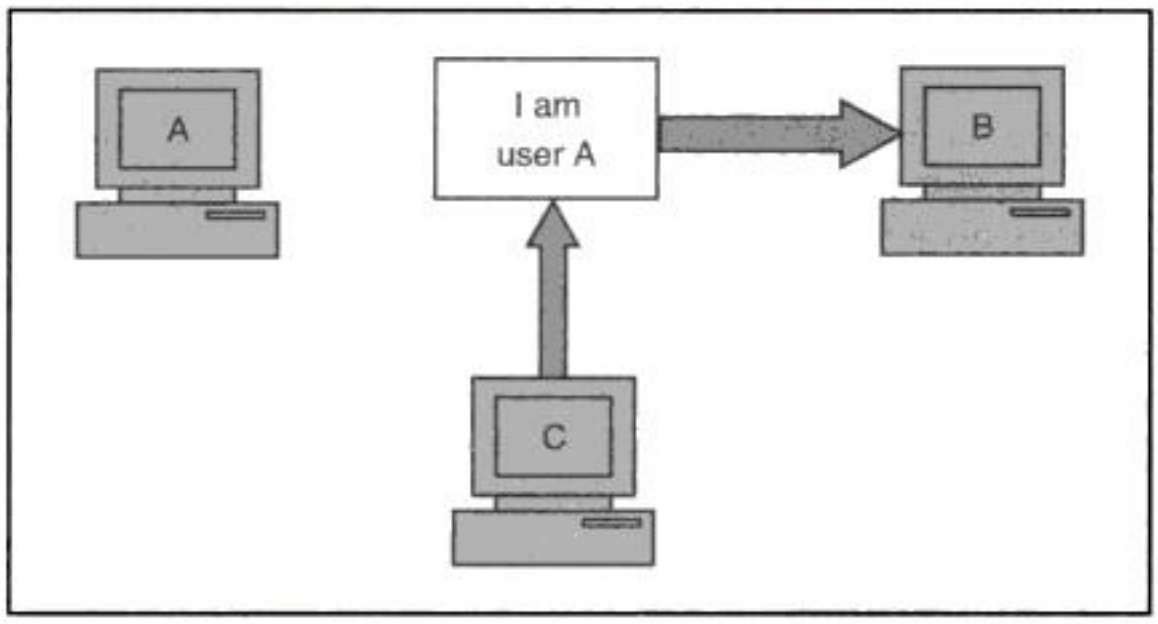


Fig. 1.3 *Absence of authentication*

Note Fabrication is possible in the 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 cheque for $100 to pay for the goods bought from the US. However, when you see your next account statement, you are startled to see that the cheque resulted in a payment of $1000! This is the case for loss of message integrity. Conceptually, this is shown in Fig. 1.4. 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 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 as **modification**.

Note 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 clain. that she never sent the funds transfer instruction to the bank! Thus, A repudiates, or denies,

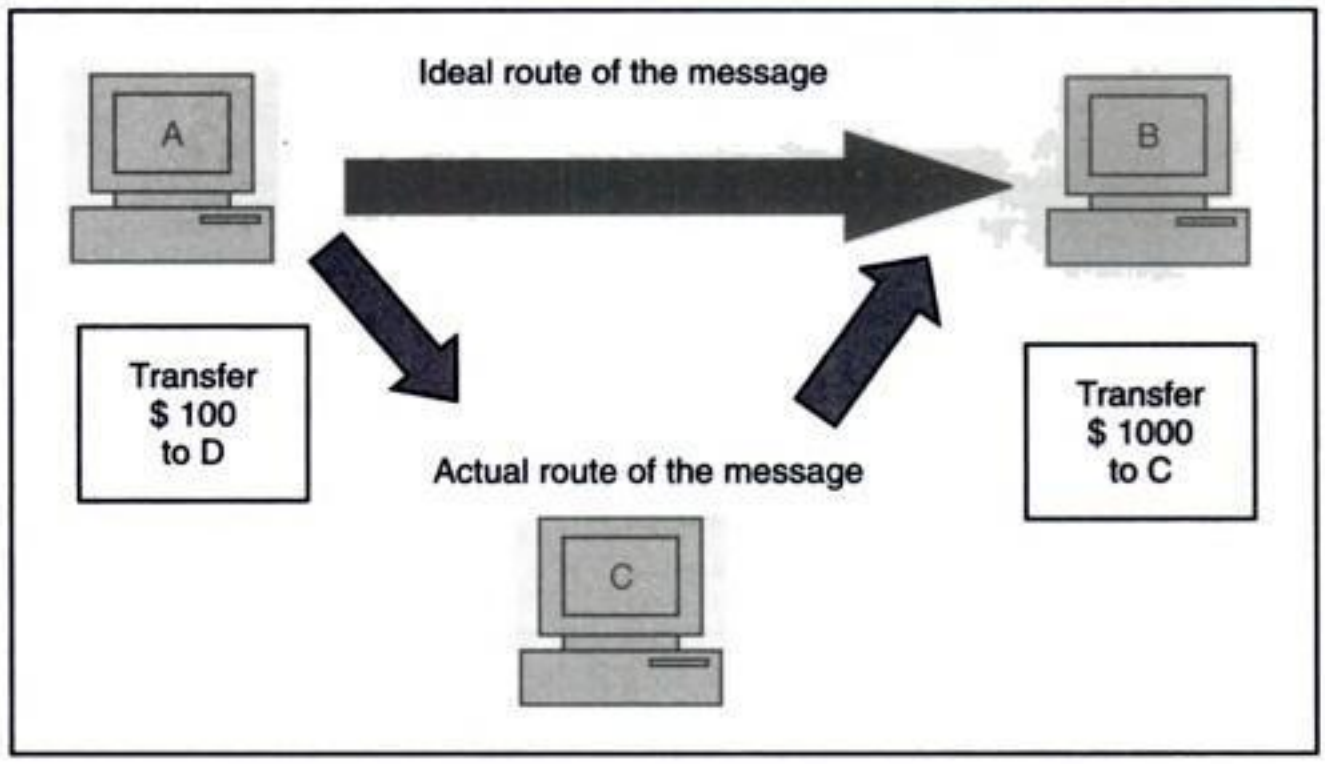


Fig. 1.4 *Loss of integrity*

her funds transfer instruction. The principle of *non-repudiation* defeats such possibilities of denying something, having done it.

Note 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.

Note 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. 1.5. This would defeat the principle of availability. Such an attack is called as **interruption**.

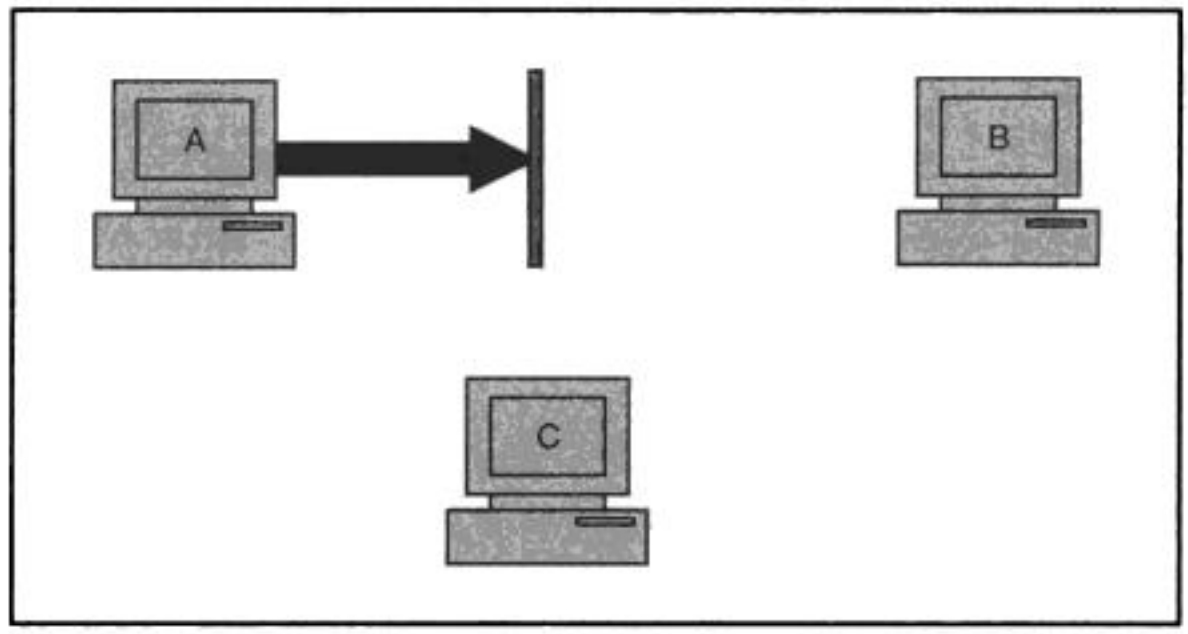


Fig. 1.5 *Attack on availability*

Note **Interruption puts the availability of resources in danger.**

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

1.5 TYPES OF ATTACKS

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.

1.5.1 Theoretical Concepts

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

- Interception—Discussed in the context of *confidentiality*, earlier.
- Fabrication—Discussed in the context of *authentication*, earlier.
- Modification—Discussed in the context of *integrity*, earlier.
- Interruption—Discussed in the context of *availability*, earlier.

These attacks are further grouped into two types: **passive attacks** and **active attacks**, as shown in Fig. 1.6.

Let us discuss these two types of attacks now.

1. *Passive attacks*

Passive attacks are those, wherein the attacker indulges in evesdropping 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.

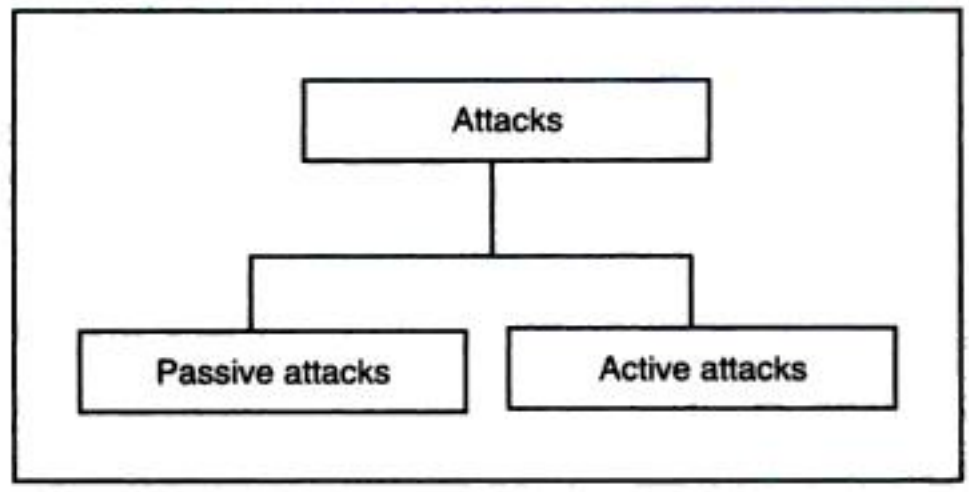


Fig. 1.6 *Types of attacks*

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

Figure 1.7 shows further classification of passive attacks into two sub-categories. These categories are **release of message contents** and **traffic analysis**.

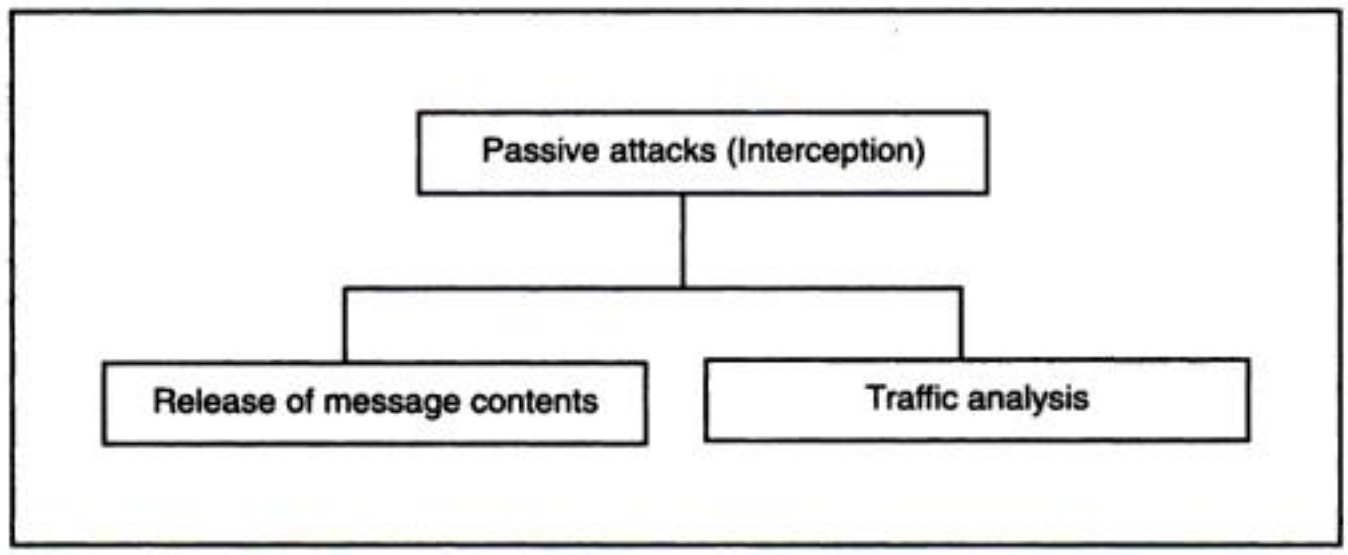


Fig. 1.7 *Passive attacks*

Release of message contents is quite simple to understand. When we send a confidential email message to our friend, we desire that only 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 *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 the 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.

2. Active attacks

Unlike *passive attacks*, the *active attacks* are based on modification of the original message in some manner, or on 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.

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

- Interruption attacks are called as **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. 1.8.

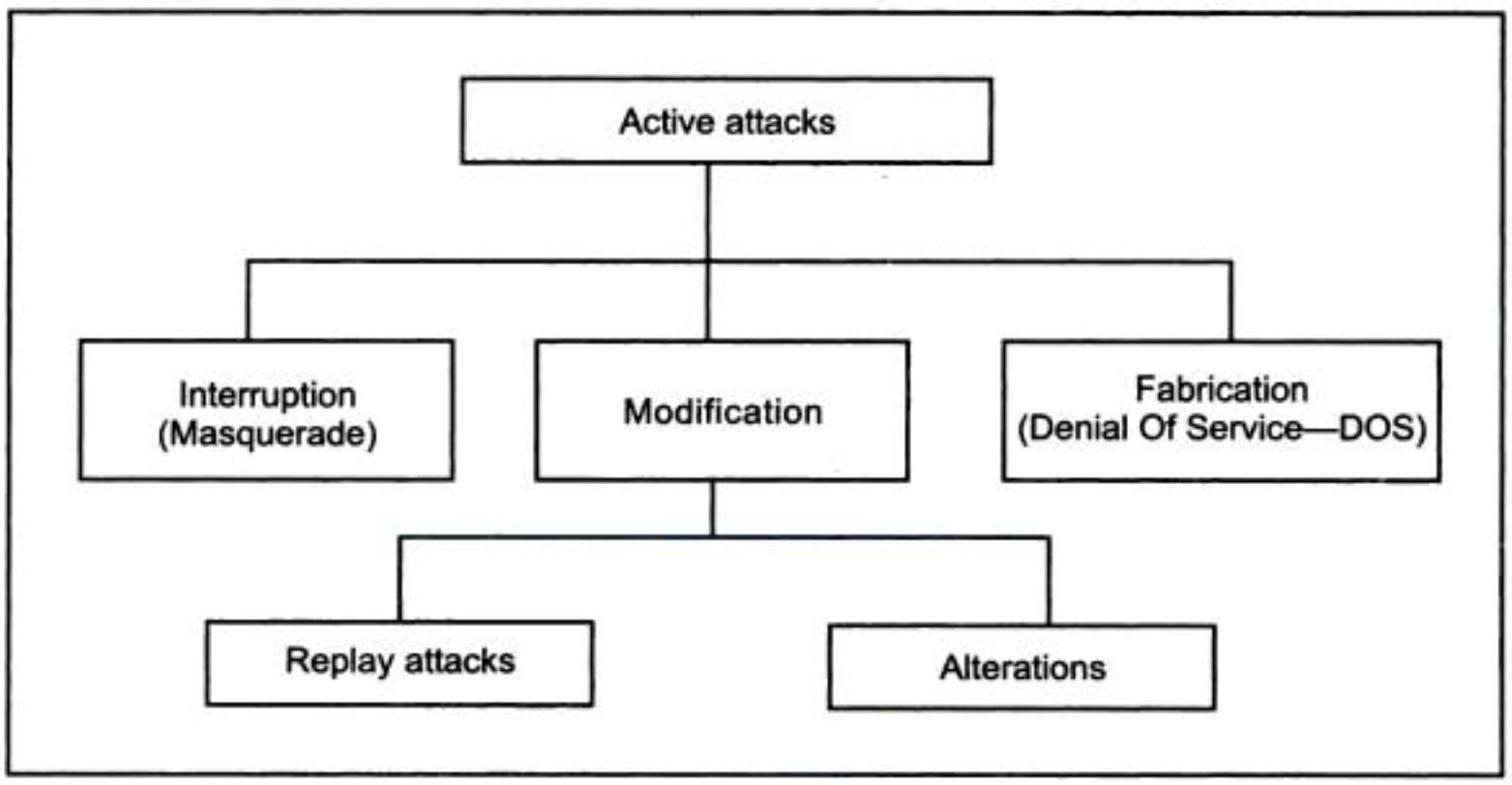


Fig. 1.8 *Active attacks*

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 a *replay attack*, a user captures a sequence of events, or some data units, and resends 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 one after the other in quick succession, so as to flood the network and deny other legitimate users an access to the network.

1.5.2 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. 1.9.

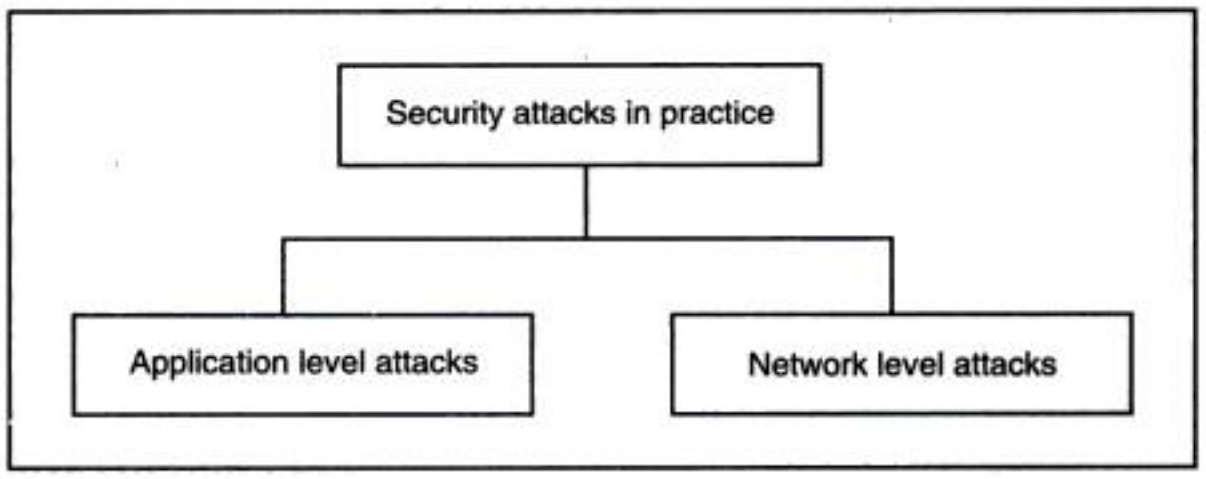


Fig. 1.9 *Practical side of attacks*

Let us discuss these now.

- **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.
- **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 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.

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

1. *Virus*

One can launch an application-level attack or a network level attack using a **virus**.

Note 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. 1.10. 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.

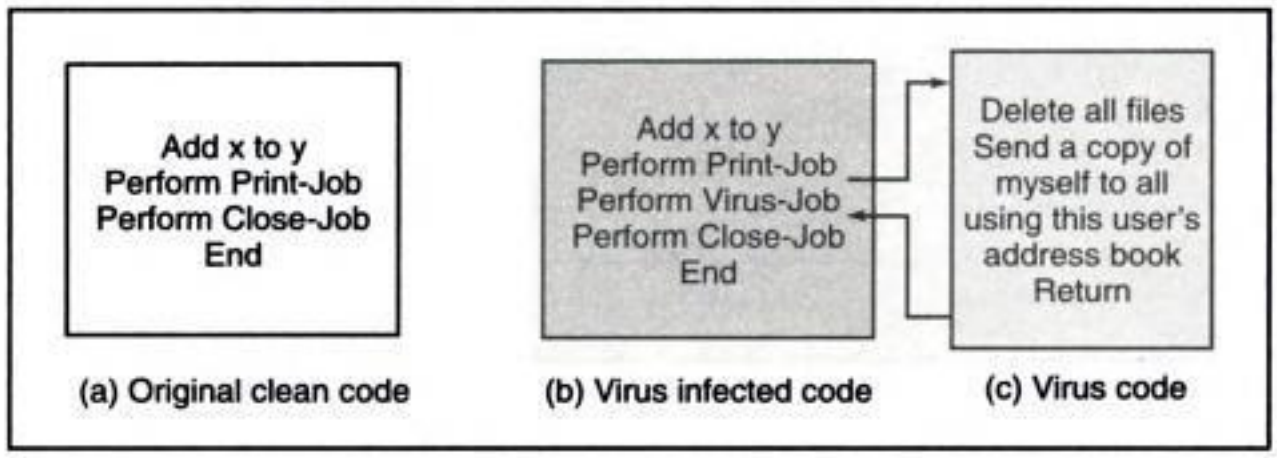


Fig. 1.10 *Virus*

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

Note A virus can be repaired, and its damage can be controlled by using good backup procedures.

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. 1.11. The replication grows so much that ultimately the computer or the network on which the worm resides, becomes very slow, finally 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.

Note 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. The main purpose of a virus is to make some sort of modifications to the target computer or network, whereas a Trojan horse attempts to reveal confidential information to an attacker. The name (Trojan horse) is due to the Greek soldiers, who hid inside a large

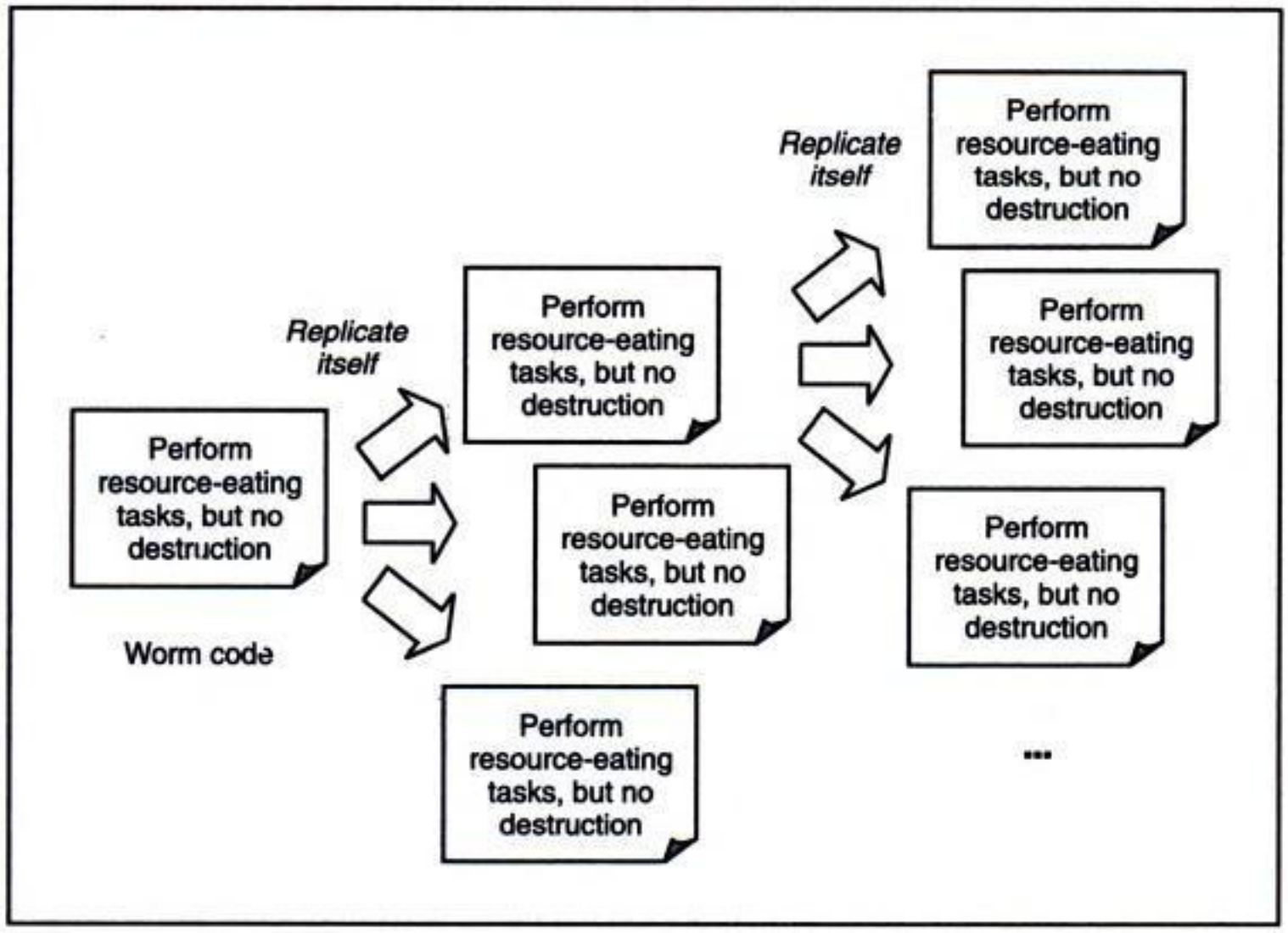


Fig. 1.11 *Worm*

hollow horse, which was pulled by Troy citizens, unaware of its *contents*. Once the Greek soldiers entered the city of Troy, they opened the gates for the rest of Greek soldiers.

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 captures these details, and sends this information to the attacker without the knowledge of the user who had entered the id and password. The attacker can then merrily use the user id and password to gain access to the system. This is shown in Fig. 1.12.

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

4. *Applets and ActiveX controls*

Applets and **ActiveX** controls were born due to the technological development of the World Wide Web (WWW) application (usually referred to simply as the *Web*) of the Internet. In its simplest form, the Web consists of communication between client and server computers using a communications protocol called as **Hyper Text Transfer Protocol (HTTP)**. The client uses a software called **Web browser**. The server runs a program called **Web server**. In its simplest form, a browser sends a HTTP request for a **Web page** to a Web server. The Web server locates this Web page (actually a computer file) and sends it back to the Web browser, again using HTTP. The Web browser interprets the contents of that file, and shows the results on the screen to the user. This is shown in Fig. 1.13. Here, the client sends a request for a Web page called as www.yahoo.com/info, which the server sends back to the client.

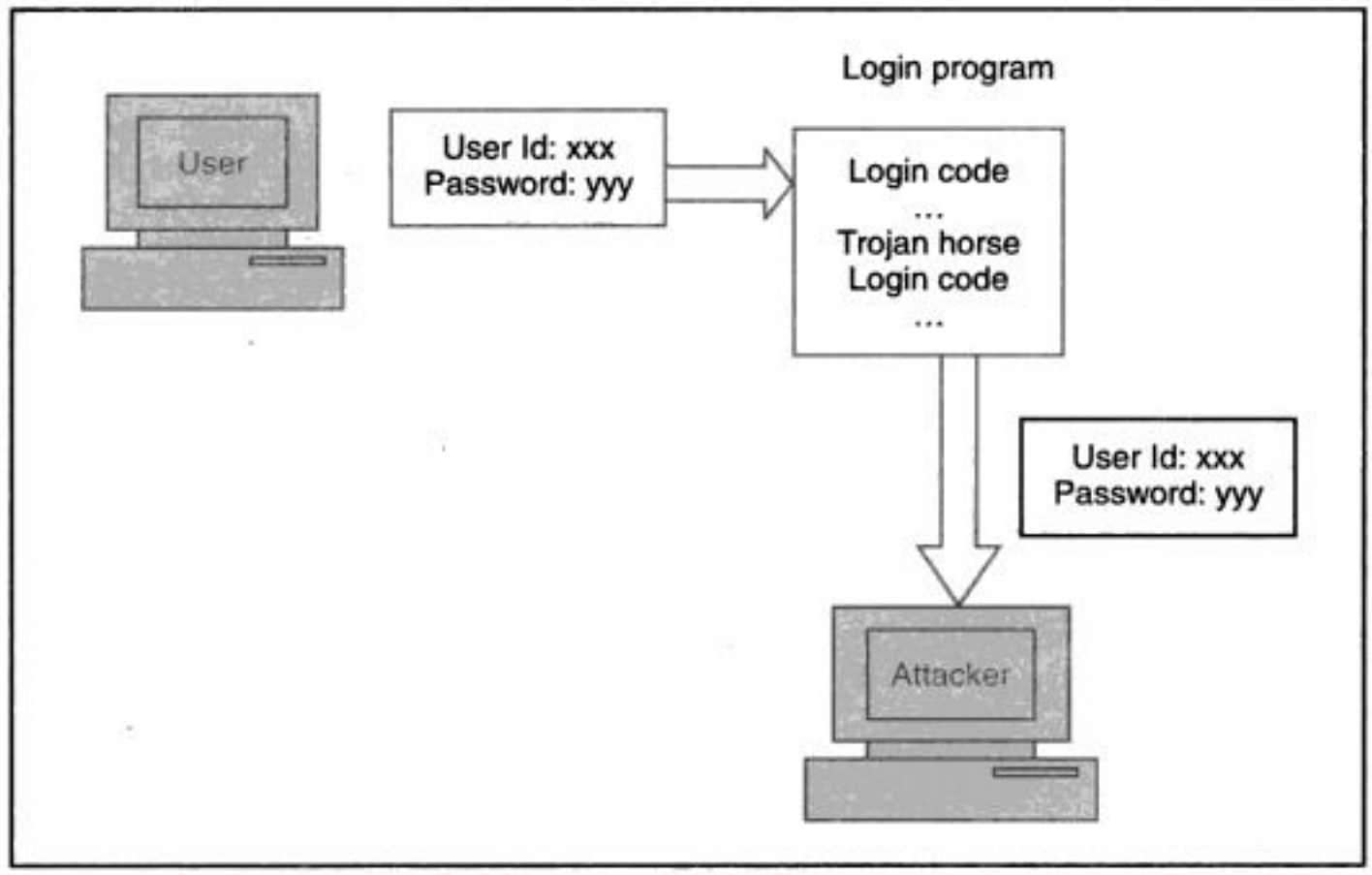


Fig. 1.12 *Trojan horse*

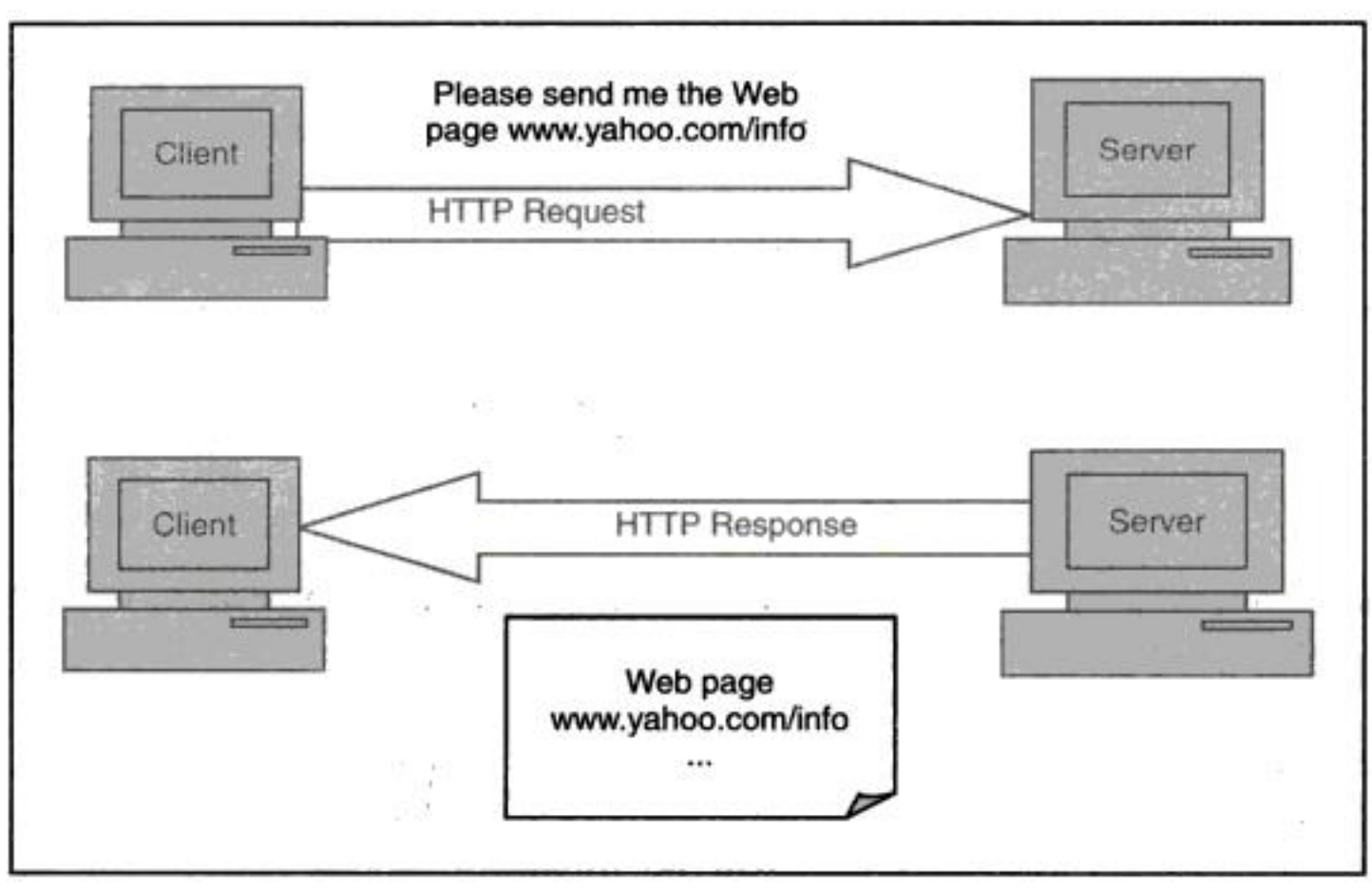


Fig. 1.13 *Example of HTTP interaction between client and server*

Many Web pages contain small programs that get downloaded on to the client along with the Web page itself. These programs then execute inside the browser. Sun Microsystems provides **Java applets** for this purpose, and Microsoft's technology makes use of **ActiveX controls** for the same purpose. Both are essentially small programs that get downloaded along with a Web page and then execute on the client. This is shown in Fig. 1.14. Here, the server sends an applet along with the Web page to the client.

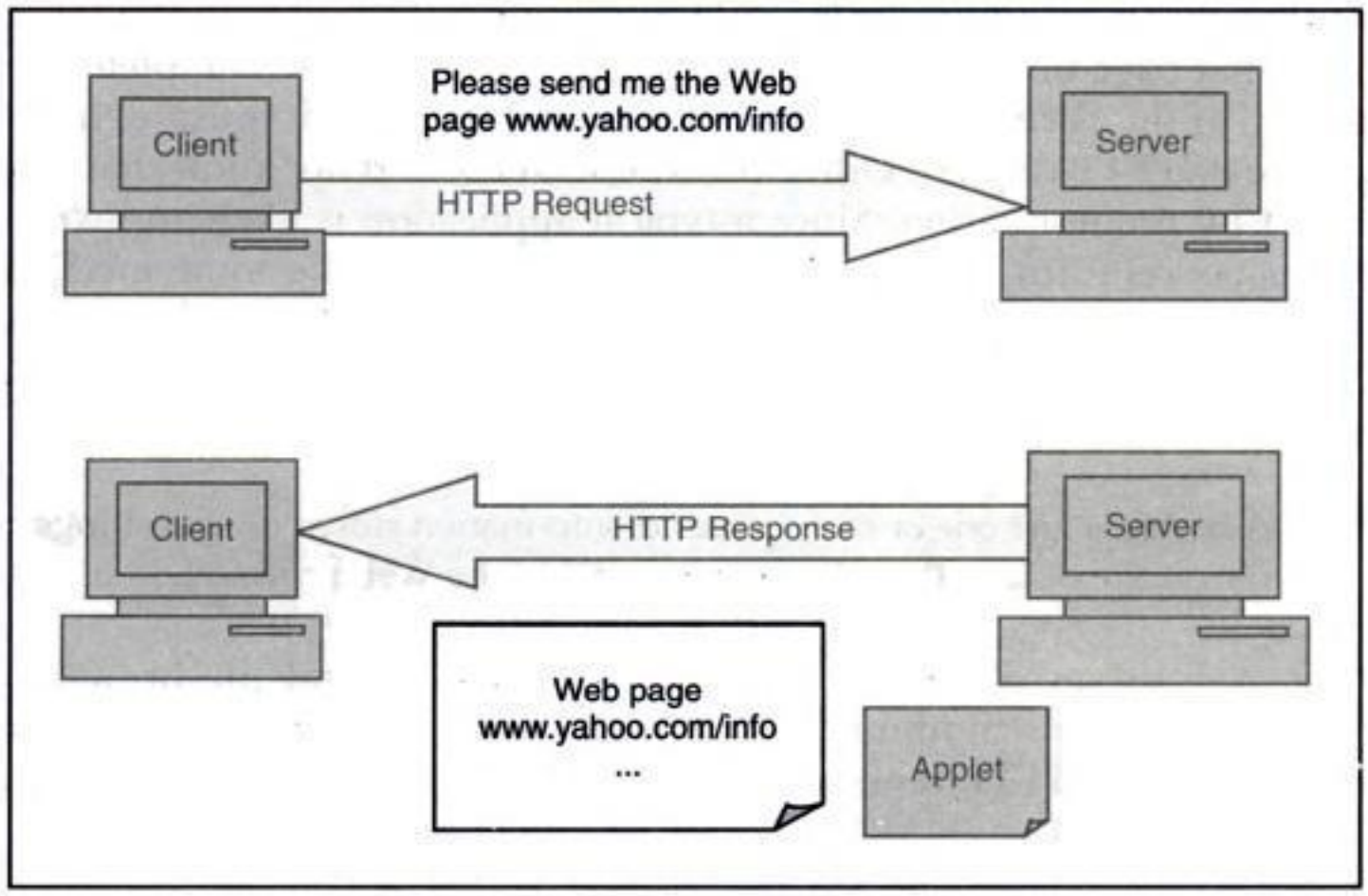


Fig. 1.14 *Applet sent back along with a Web page*

Usually, these programs (applets or ActiveX controls) are used to either perform some processing on the client side, or to automatically and periodically request for information from the Web server using a technology called as *client pull*. For instance, a program can get downloaded on to the client along with the Web page showing the latest stock prices on a stock exchange, and then periodically issue HTTP requests for pulling the updated prices, to the Web server. After obtaining this information, the program could display it on the user's screen.

These apparently innocuous programs can sometimes cause havocs. What if such a program performs a virus-like activity by deleting files on the user's hard disk, or by stealing some personal information, or by sending junk emails to all the users whose addresses are contained in the user's address book?

To prevent these attacks, Java applets have strong security checks as to what they can do, and what they cannot. ActiveX controls have no such restrictions. Moreover, a new version of applets called as **signed applets** allows accesses similar to ActiveX. Of course, a number of checks have been in place to ensure that neither applets nor ActiveX controls can do a lot of damage, and even if they somehow manage to do it, it can be detected. However, at least in theory, they pose some sort of security risks.

Java applets (from Sun Microsystems) and ActiveX controls (from Microsoft Corporation) are small client-side programs that might cause security problems, if used by attackers with a malicious intention.

5. *Cookies*

Cookies were born as a result of a specific characteristic of the Internet. The Internet uses HTTP protocol, which is **stateless**. Let us understand what it means, and what are its implications.

Suppose that the client sends an HTTP request for a Web page to the server. The Web server locates that page on its disk, sends it back to the client, and completely forgets about this interaction! If the client wants to continue this interaction, it must identify itself to the server in the next HTTP request. Otherwise, the server would not know that this same client had sent a HTTP request earlier. Since a typical application is likely to involve a number of interactions between the client and the server, there must be some mechanism for the client to identify itself to the server each time it sends a HTTP request to the server. For this, *cookies* are used. Cookies are perhaps the most popular mechanism of maintaining the *state information* (i.e. identifying a client to a server).

Note A cookie is just one or more pieces of information stored as text strings in a text file on the disk of the client computer (i.e. the Web browser).

Actually, a Web server sends the Web browser a cookie and the browser stores it on the hard disk of the client computer. The browser then sends a copy of the cookie to the server during the next HTTP request. This is used for identification purposes as shown in Figs. 1.15(a) and 1.15(b).

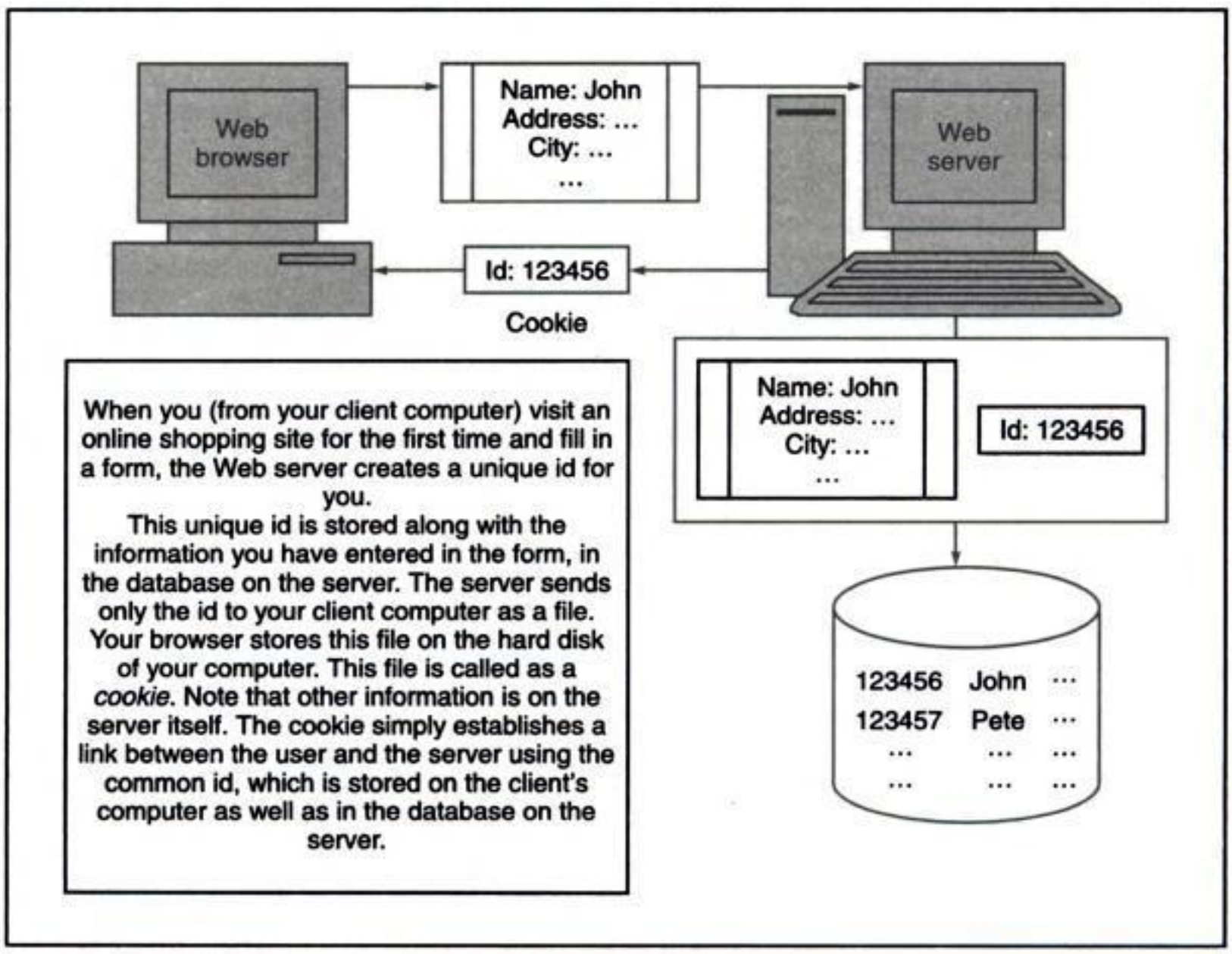


Fig. 1.15(a) *Creation of cookies*

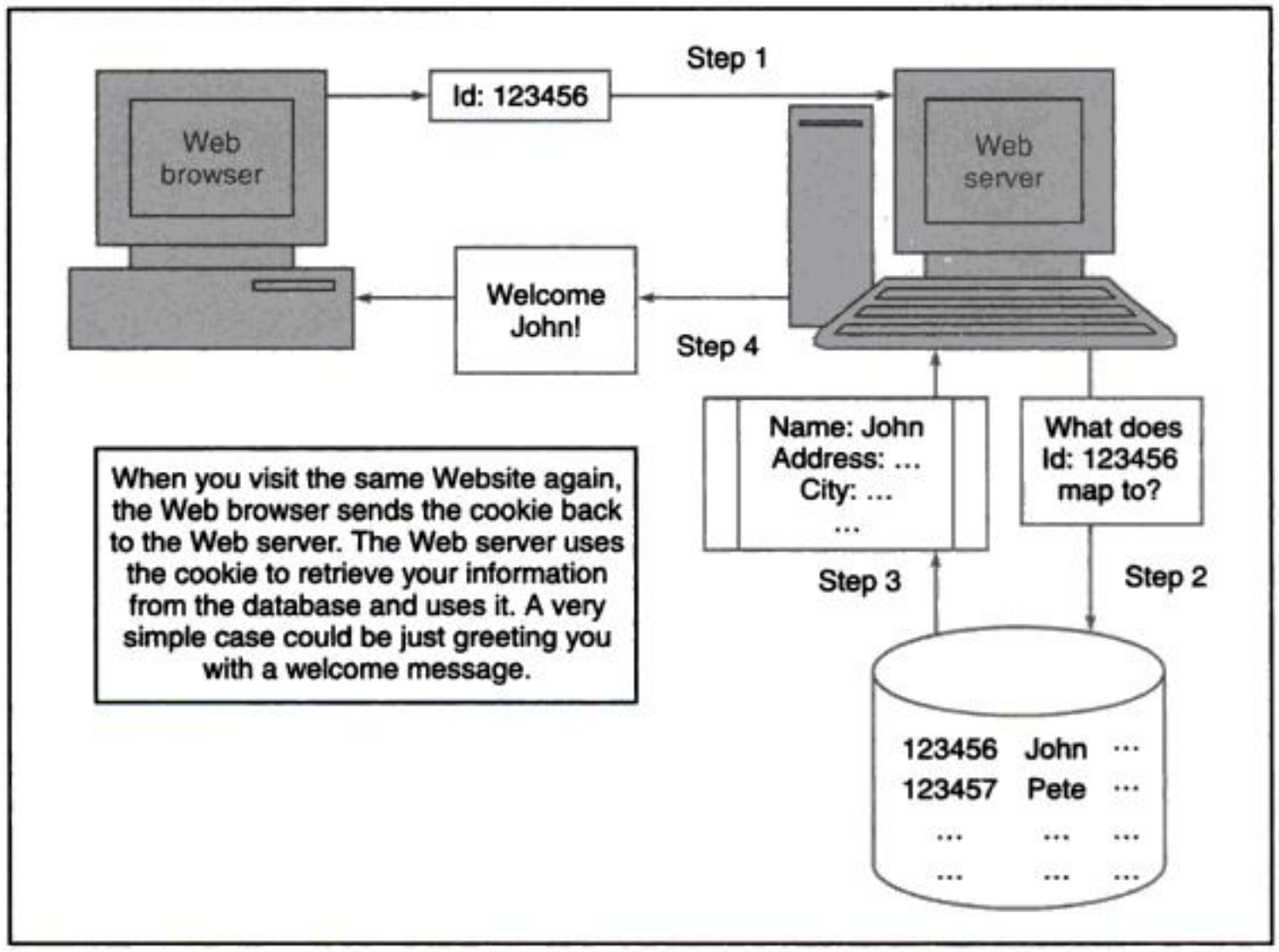


Fig. 1.15(b) *Usage of cookies*

This works as follows:

(a) When you interact with a Website for the first time, the site might want you to *register* yourself. Usually, this means that the Web server sends a page to you wherein you have a form to enter your name, address and other details such as date of birth, interests, etc.

(b) When you complete this form and send it to the server with the help of your browser, the server stores this information into its database. Additionally, it also creates a unique id for you. It stores this id along with your information in the database (as shown in the Fig. 1.15) and also sends the id back to you in the form of a cookie.

(c) The next time you interact with the server, you do not have to enter any information such as your name and address. Your browser would automatically send your id (i.e. the cookie) along with the HTTP request for a particular page to the server (as shown in the Fig. 1.15).

(d) The server now takes this id, tries to find a match in its database, and having found it, knows that you are a registered user. Accordingly, it sends you the next page. As illustrated in the Fig. 1.15, it could be a simple welcome message. In practical situations, this could be used for many other purposes.

People perceive that cookies are dangerous. Actually, this is generally not true. Cookies can do little, if any, harm to you. Firstly, the Web server that originally created a cookie can only access the cookie. Secondly, cookies can contain only text-based information. Thirdly, the user can refuse accepting cookies.

6. JavaScript, VBScript and JScript

A Web page is constructed using a special language called as **Hyper Text Markup Language (HTML)**. It is a tag-based language. A tag begins with the symbol <> and it ends with </ >. Between these boundaries of the tags, the actual information to be displayed on the user's computer is mentioned. As an example, let us consider how the tag pair <B> and </B> can be used to change the text font to boldface. This is shown in Fig. 1.16.

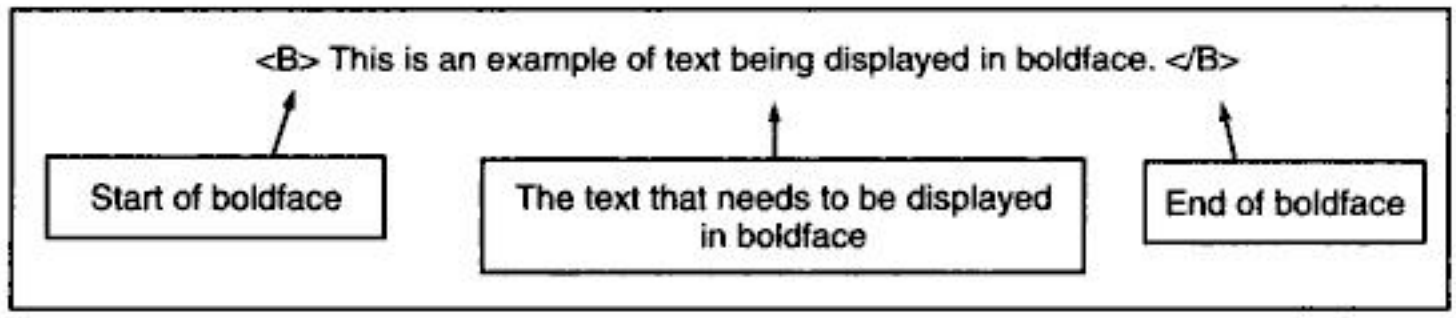


Fig. 1.16 *Example of the <B> and </B> HTML tags to display the specified text in boldface*

When a browser comes across this portion of a HTML document, it realizes that the portion of the text embedded within the <B> and </B> tags needs to be displayed in boldface. Therefore, it displays this text in boldface, as shown in Fig. 1.17.

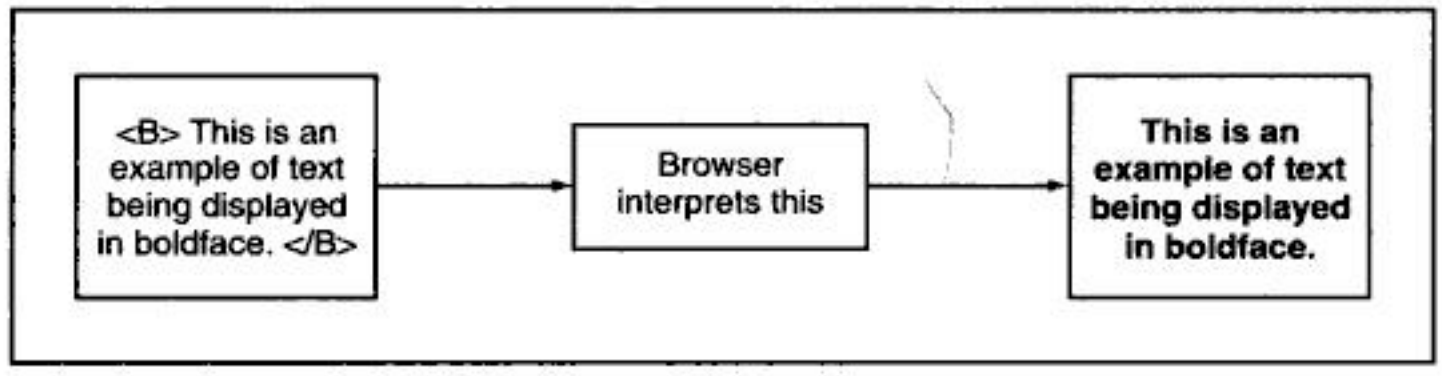


Fig. 1.17 *Output resulting from the use of the <B> and </B> HTML tags to display the specified text in boldface*

In addition to HTML tags, a Web page can contain **client-side scripts**. These are small programs written in scripting languages like JavaScript, VBScript or JScript, which are executed inside the Web browser on the client computer. For instance, let us assume that a user visits the Website of an online bookshop. Suppose that the Website mandates that the user must place an order for at least three books. Then, the Web page can contain a small JavaScript program, which can ensure that this condition is met before the user can place the order. Otherwise, the JavaScript program would not allow the user to proceed. Note that HTML cannot be used for this purpose, as its sole purpose is to display text on the client computer in a pre-specified format. To perform dynamic actions, such as the one discussed here, we need scripts.

These scripts can be dangerous at times. Since these scripts are small programs, they can perform a lot of actions on the client's computer. Of course, there are restrictions as to what a scripting program can and cannot do. However, still incidents of security breaches have been reported, whose blame was put to such scripting languages.

1.5.3 Java Security

1. *Introduction*

For Java to become successful, it needed to avoid the security problems that had plagued other models of software distribution. Therefore, the early design of Java focused mainly on these concerns. Consequently, Java was designed in such a way that Java programs are considered safe as they cannot install, execute, or propagate viruses, and because the program itself cannot perform any action that is harmful to the user's computer.

As we know, one of the key attributes of Java is the ability to download Java programs over a network and execute these programs on a different computer within the context of a Java-enabled browser. Different developers were attracted to Java with different expectations. As a result, they brought different ideas about Java security. Simply put, if we expect Java to be free from introducing viruses, any release of Java should satisfy our requirements. (However, if we require special functionalities such as digital signatures, authentication and encryption in our programs, we need to use at least release 1.1 of Java).

Interestingly, Java security discussions are centered on the idea of Java's applet based security model. This security is contained inside Java-enabled browsers. This model is envisaged for use on the Internet.

2. *The Java sandbox*

Java's security model is closely associated with the idea of a **sandbox** model. A sandbox model allows a program to be hosted and executed, but there are some restrictions in place. The developer/end user may decide to give the program access to certain resources. However, in general, they want to make sure that the program is confined to its sandbox. The overall execution of a Java program on the Internet is as shown in Fig. 1.18, and explained below.

The chief job of the Java sandbox is to protect a number of resources, and it performs this task at a number of levels, as described below.

- A sandbox in which a program can access the CPU, the screen, the keyboard and mouse, and its own memory. This is the basic sandbox. It contains just enough resources for a program to execute.
- A sandbox in which a program can access the CPU, and its memory as well as access the Web server from which it was downloaded. This is often considered as the default state for the sandbox.
- A sandbox in which program can access the CPU, its memory, its Web server and a set of resources (files, computers, etc.) that are local.
- An open sandbox, in which the program can access whatever resources the host machine can.

3. *Java application security*

Let us discuss the broad level aspects of Java security, and how they relate to each other.

- *The bytecode verifier* The bytecode verifier ensures that Java class files obey the rules of the Java programming language. The bytecode verifier ensures memory protection for all Java programs. However, not all files are required to go through bytecode verification.

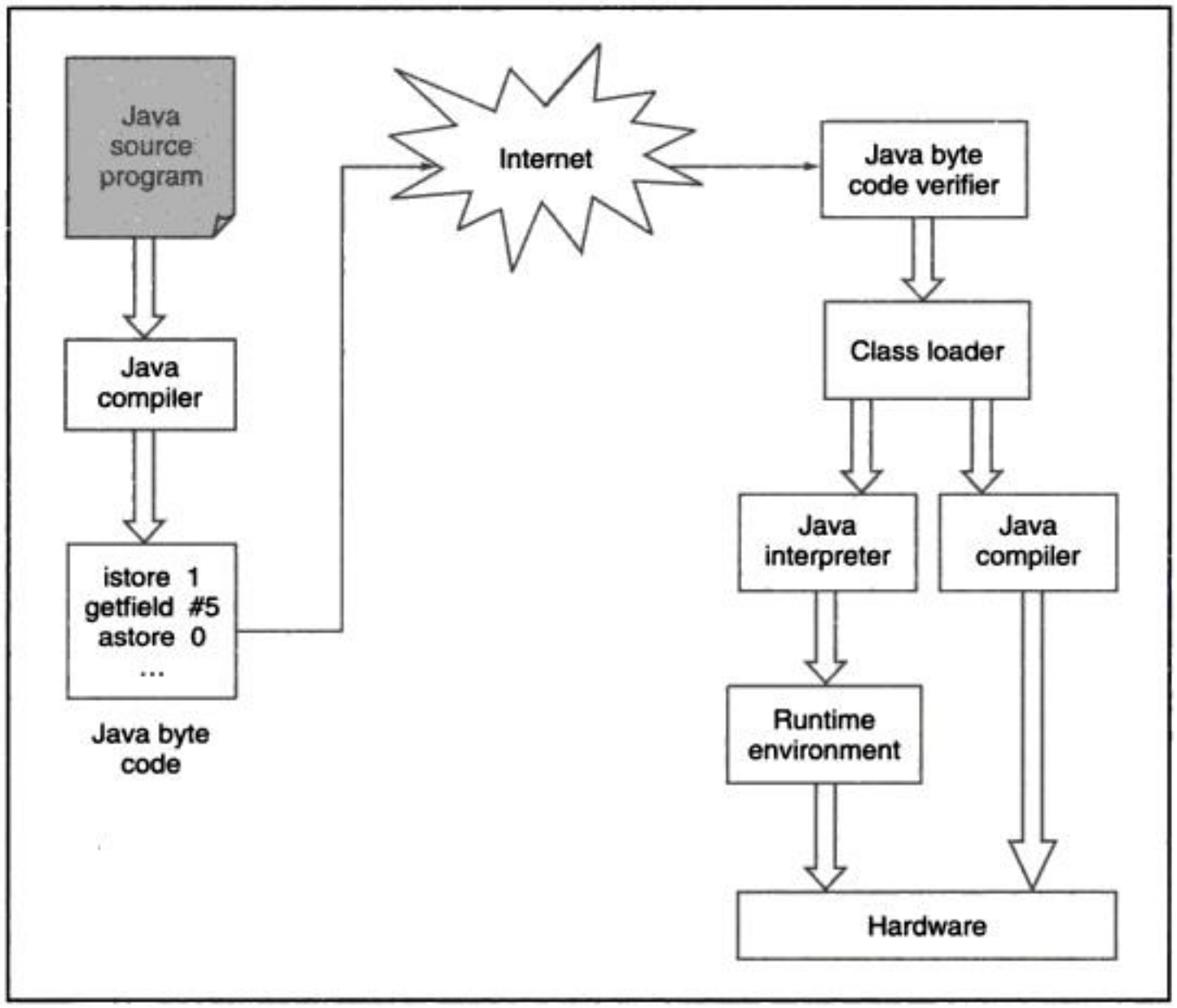


Fig. 1.18 *Steps in the execution of a Java program on the Internet*

- *The class loader* Class loaders load classes that are located in Java's default path (called as CLASSPATH). In Java 1.2, the class loaders also take up the job of loading classes that are not found in the CLASSPATH.
- *The access controller* In Java 1.2, the access controller allows (or prevents) access from the core Java API to the operating system.
- *The security manager* The security manager is the chief interface between the core Java API and the operating system. It has the ultimate responsibility for allowing or disallowing access to all operating system resources. The security manager uses the access controller for many of these decisions.
- *The security package* The security package (that is, classes in the java.security package) helps in authenticating signed Java classes.
- *The key database* The key database is a set of keys used by the security manager and access controller to validate the digital signature that comes along with a signed class file. In the Java architecture, it is contained within the security package, although it may be an external file or database as well.

4. Built-in Java application security

From version 1.2, the Java platform itself comes with a security model built for the applications it runs. Here, the classes that are found in the CLASSPATH may have to go through

a security check. This allows running of the application code in a sandbox defined by a user or an administrator. The following points are salient:

- Access methods are strictly adhered to
- A program cannot access arbitrary memory location
- Entities that are declared as *final* must not be changed
- Variables may not be used before they are initialized
- Array bounds must be checked during all array accesses
- Objects cannot arbitrarily cast into other object types

To illustrate this, consider a C program shown in Fig. 1.19. As we can see, the program simply declares a character pointer, and without allocating any memory, accepts user input in that pointer. This can cause havoc, if an attacker finds intelligent ways to exploit such code. This is not possible in Java.

```
#include <stdio.h>

void main ()
{
        char *p;

        printf ("Enter a string: ");
        gets (p);

        printf ("You entered: %s", p);
}
```

Fig. 1.19 *C program using a pointer without initializing it*

1.5.4 Specific Attacks

On the Internet, computers exchange messages with each other in the form of small groups of data, called as 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 as **snooping**) and (b) **Packet spoofing**. Since the protocol used in this communication is called as 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 a false source address) would inadvertently send replies back to this forged address (called as **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.

Another attack, which is similar to these attacks, is the **DNS spoofing** attack. As we know, using the **Domain Name System (DNS)**, people can identify Websites 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 as 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.

1. 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

2. The attacker (Say Trudy) manages to hack and replace the IP address of Bob with her own (say 100.20.20.20) in the DSN 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

3. 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.
4. 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 as **DNSSec (Secure DNS)** is being used to thwart such attacks. However, unfortunately it is not widely used.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Chapter 10 contains a number of case studies in the area of cryptography and network security. It discusses how the concepts learnt in the earlier chapters can be put into actual practice. It also covers a few real-life security attacks that have happened, and how they have been dealt with. This presents the viewpoints of the attackers as well as those of the attacked party.

Several appendices then examine the various theoretical and practical sides of cryptography and network security.

Chapter Summary

- *Network and Internet security has gained immense prominence in the last few years, as conducting business using these technologies has become very crucial.*
- *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 an 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.*
- *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, Java applets, and ActiveX controls can practically cause attacks on a computer system.*

Key Terms and Concepts

- Access Control List (ACL)
- ActiveX control
- Application level attack
- Availability
- Active attack
- Alteration of message
- Authentication
- Confidentiality

- Denial Of Service (DOS) attack
- Integrity
- Interruption
- Masquerade
- Network level attack
- Passive attack
- Replay attack
- Traffic analysis
- Virus
- Fabrication
- Interception
- Java applet
- Modification
- Non-repudiation
- Release of message contents
- Signed Java applet
- Trojan horse
- Worm

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. Allowing certain users specific accesses comes in the purview of _____.
   (a) confidentiality (b) authentication
   (c) integrity (d) access control
5. If a computer system is not accessible, the principle of _____ is violated.
   (a) confidentiality (b) authentication
   (c) availability (d) access control
6. The four primary security principles related to a message are _____.
   (a) confidentiality, authentication, integrity and non-repudiation
   (b) confidentiality, access control, non-repudiation and integrity
   (c) authentication, authorization, non-repudiation and availability
   (d) availability, access control, authorization and authentication
7. 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
8. The _____ attack is related to confidentiality.
   (a) interception (b) fabrication
   (c) modification (d) interruption

9. The _____ attack is related to authentication.
   (a) interception (b) fabrication
   (c) modification (d) interruption
10. The _____ attack is related to integrity.
   (a) interception (b) fabrication
   (c) modification (d) interruption
11. The _____ attack is related to availability.
   (a) interception (b) fabrication
   (c) modification (d) interruption
12. In _____ attacks, there is no modification to message contents.
   (a) passive (b) active
   (c) both of the above (d) none of the above
13. In _____ attacks, the message contents are modified.
   (a) passive (b) active
   (c) both of the above (d) none of the above
14. Interruption attacks are also called as _____ attacks.
   (a) masquerade (b) alteration
   (c) denial of service (d) replay attacks
15. DOS attacks are caused by _____.
   (a) authentication (b) alteration
   (c) fabrication (d) replay attacks
16. Virus is a computer _____.
   (a) file (b) program
   (c) database (d) network
17. A worm _____ modify a program.
   (a) does not (b) does
   (c) may or may not (d) may
18. A _____ replicates itself by creating its own copies, in order to bring the network to a halt.
   (a) virus (b) worm
   (c) Trojan horse (d) bomb
19. Applets and ActiveX controls are _____ side programs.
   (a) client (b) server
   (c) database (d) none of the above
20. ActiveX controls are _____ secure as compared to applets.
   (a) more (b) equally
   (c) far more (d) less

Review Questions

1. Find out more examples of security attacks reported in the last few years.
2. What are the key principles of security?
3. 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).

4. 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?)
5. In real life, how is the message integrity ensured? (*Hint*: On what basis is a check honored or dishonored?)
6. 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).
7. What is access control? How different is it from availability?
8. Why are some attacks called as passive? Why are others called active?
9. Discuss any one passive attack.
10. What is masquerade? Which principle of security is breached because of that?
11. What are replay attacks? Give an example of replay attacks.
12. What is denial of service attack?
13. What is a worm? What is the significant difference between a worm and a virus?
14. Find out more about some recent worms that were floating around the world over the Internet.
15. Write a small virus-like program in plain English language that accepts a file name and changes every character in the file to an asterisk.
16. Read more about computer viruses, and their principle of working in detail.
17. What is a Trojan horse? What is the principle behind it?
18. What are Java applets?
19. Discuss ActiveX controls, and contrast them with applets.
20. Find out more about applets and ActiveX controls technology.

Design/Programming Exercises

1. Write an ActiveX control that creates a file on the user's disk, when downloaded, and deletes the same thereafter.
2. Write an ASP program that creates a cookie with a value 'Welcome to Internet' on the user's computer.
3. Write an ASP program that reads the cookie value of the above exercise, and displays its value on the user's browser.
4. 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. Do you notice anything special?
5. 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?

Chapter 2

Cryptographic 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 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 ready 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.

In the early days, cryptography used to be 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. 2.1.

Hi Amit

Hope you are doing fine. How about meeting at the train station this Friday at 5 pm? Please let me know if it is ok with you.

Regards.

Atul

Fig. 2.1 *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.

Note Clear text or plain text signifies a message that can be understood by the sender, the recipient, and also by anyone else who gets an 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 that I am interested in knowing 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. Meena), I 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 earlier shown in Fig. 2.1 is confidential for some reason. Therefore, I do not want anyone else to understand what I have written even if 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 alphabet. 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. 2.2. 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. 2.2 *A scheme for codifying messages by 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. 2.3.

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

Fig. 2.3 *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 as **cipher text**. Cipher means a code or a secret message.

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

Let us now write our original email message and the resulting cipher text by using the alphabet-replacing scheme, as shown in Fig. 2.4. This will clarify the idea further.

As shown in Fig. 2.5, there are two primary ways in which a plain text message can be codified to obtain the corresponding cipher text: **Substitution** and **Transposition**.

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

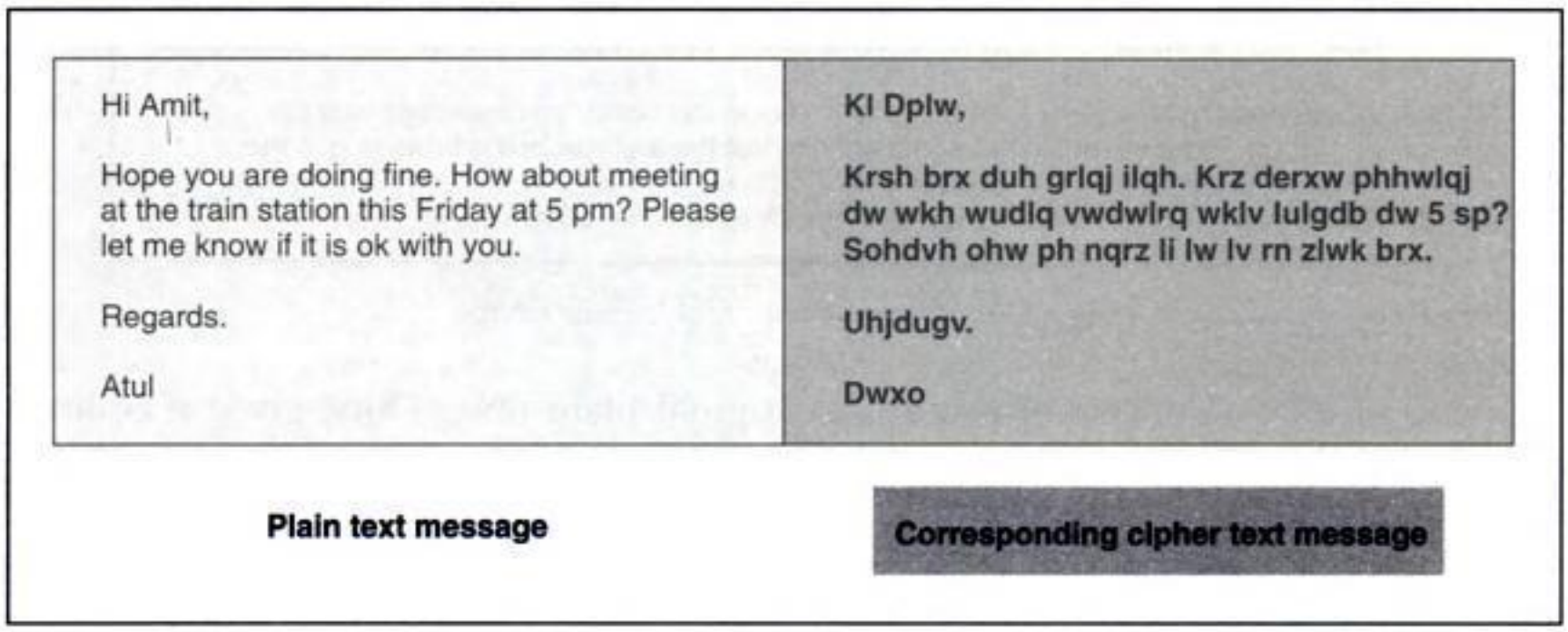


Fig. 2.4 *Example of a plain text message being transformed into cipher text*

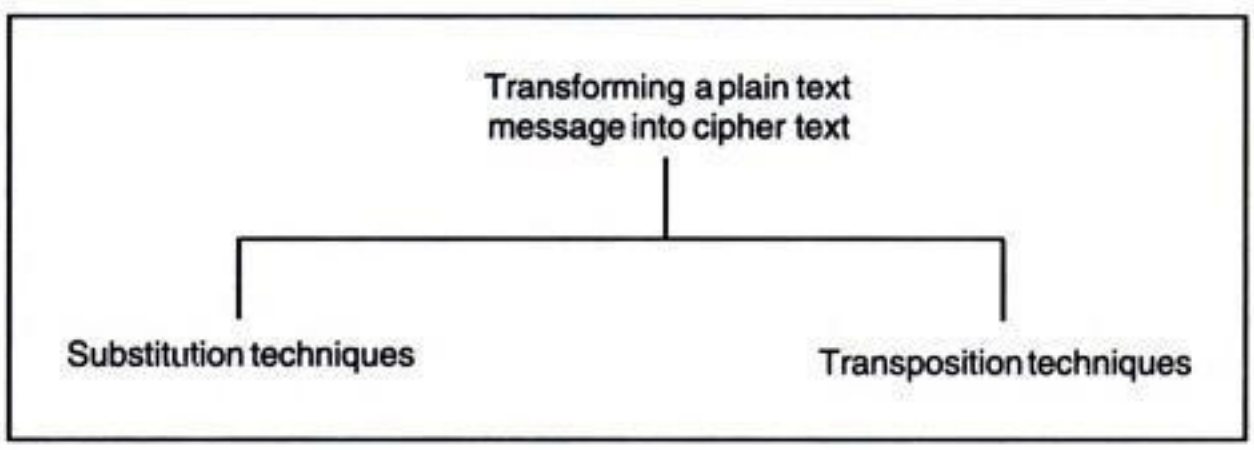


Fig. 2.5 *Techniques for transforming plain text to cipher text*

2.3 SUBSTITUTION TECHNIQUES

2.3.1 Caesar Cipher

The scheme illustrated in Fig. 2.2 was first proposed by Julius Caesar, and is termed as **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. Caesar Cipher is a special case of substitution techniques 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.

Note 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 Caesar Cipher can be summarized as shown in Fig. 2.6.

1. Read each alphabet in the cipher text message, and search for it in the second row of the Fig. 2.2.
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. 2.6 *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. 2.7.

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

Fig. 2.7 *Example of breaking caesar cipher*

2.3.2 Modified Version of Caesar Cipher

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 life 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 Caesar Cipher, as shown in Fig. 2.8.

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. 2.8 *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 in Fig. 2.8. Since each alphabet in the plain text can be potentially replaced by any of the other 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 Table 2.1.

Table 2.1 *Attempts to break modified Caesar cipher text using all possibilities*

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

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 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 as 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 as a **Brute-force attack**. The process of trying to break any cipher

text message to obtain the original plain text message itself is called as **Cryptanalysis**, and the person attempting a cryptanalysis is called as a **cryptanalyst**.

Cryptanalyst is a person who attempts to break a cipher text message to obtain the original plain text message. The process itself is called as 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 even the modified Caesar Cipher tough 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 she is assured of a 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 $(26 \times 25 \times 24 \times 23 \times \ldots 2)$ 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 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 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, 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 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 the replacement alphabet set in case of the simple substitution techniques is fixed (e.g. replace A with D, B with E, etc.), whereas 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 **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. 2.9. This is true in spite the first four characters of the two blocks of text (HELL) being the same. This shows that in Polygram Substitution Cipher, the replacement of plain text happens block-by-block, rather than character-by-character.

Polygram Substitution Cipher technique replaces one block of plain text with a 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 Vigenere Cipher and the Beaufort Cipher are examples of Polyalphabetic Substitution Cipher.

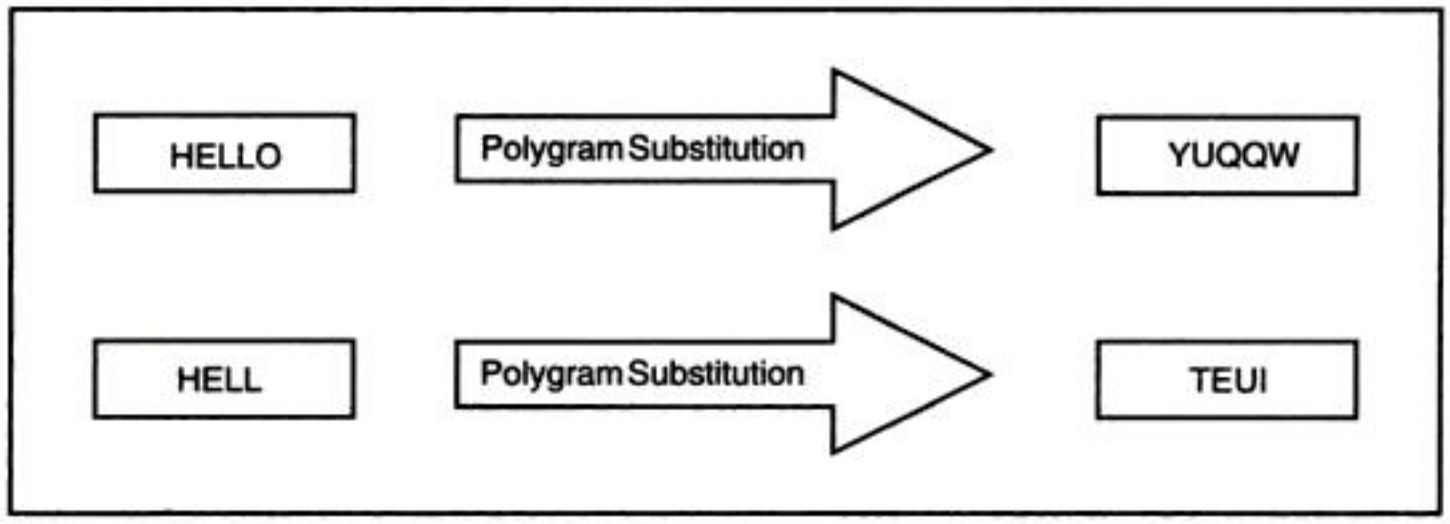


Fig 2.9 *Polygram substitution*

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 as the **period** of the cipher.

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: they also perform some permutation over the plain text alphabets.

2.4.1 Rail Fence Technique

The **Rail Fence Technique** is an example of transposition. It uses a simple algorithm as shown in Fig. 2.10.

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. 2.10 *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. 2.11.

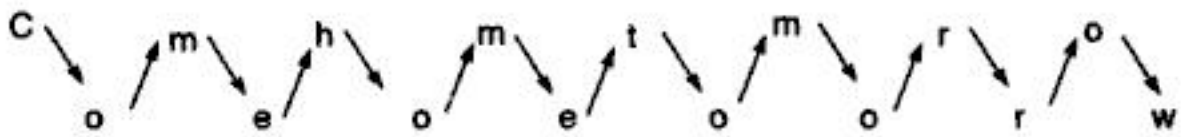
As the figure shows, the plain text message *Come home tomorrow* transforms into *Cmhmtmrooeoeoorw* with the help of Rail Fence Technique.

Note Rail fence technique involves writing plain text as 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***

1. 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.



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

Fig. 2.11 *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. 2.12, which we shall call as **Simple Columnar Transposition Technique**.

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.

Fig. 2.12 *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. 2.13.

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 (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		

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.

Fig. 2.13 *Example of simple columnar transposition technique*

Note 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 rounds 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 do it more than once. That adds considerably more complexity for the cryptanalyst.

The basic algorithm used in this technique is shown in Fig. 2.14.

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. 2.14 *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. 2.15.

As the Fig. 2.15 shows, multiple rounds or iterations add more complexity to the cipher text produced by the basic Simple Columnar Transposition Technique. More the number of iterations, more complex is the cipher text thus produced.

Note 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**, also called as **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 Vernam Cipher is described in Fig. 2.16.

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 ***oeochemmormorwot*** in round 2.
7. Continue like this if more number of iterations is desired, otherwise stop.

Fig. 2.15 *Example of simple columnar transposition technique with multiple rounds*

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. 2.16 *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 *UQXTUYFR* as shown in Fig. 2.17.

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 as the Vernam Machine.

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

	H	O	W	A	R	E	Y	O	U
1. Plain text	7	14	22	0	17	4	24	14	20
	+								
2. One-time pad	13	2	1	19	25	16	0	17	23
	N	C	B	T	Z	Q	A	R	X
3. Initial Total	20	16	23	19	42	20	24	39	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. 2.17 *Example of vernam cipher*

2.4.4 Book Cipher/Running Key Cipher

The idea used in **Book Cipher**, also called as **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 as **encryption**. Figure 2.18 illustrates the idea.

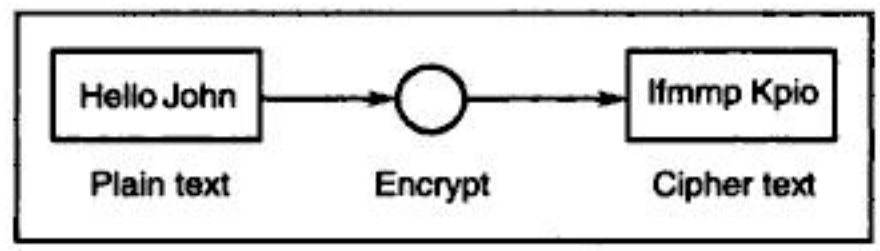


Fig. 2.18 *Encryption*

The reverse process of transforming cipher text messages back to plain text messages is called as **decryption**. Figure 2.19 illustrates the idea.

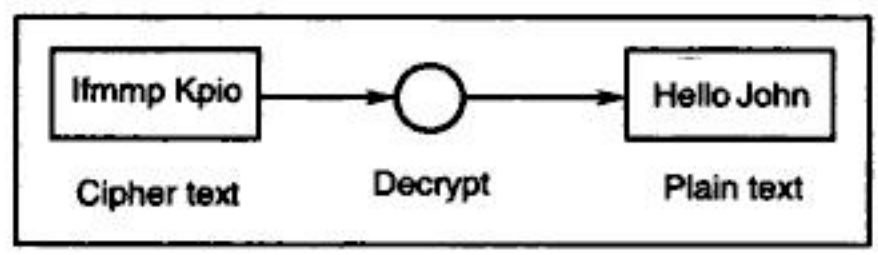


Fig. 2.19 *Decryption*

Note Decryption is exactly 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 cipher text 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 Fig. 2.20.

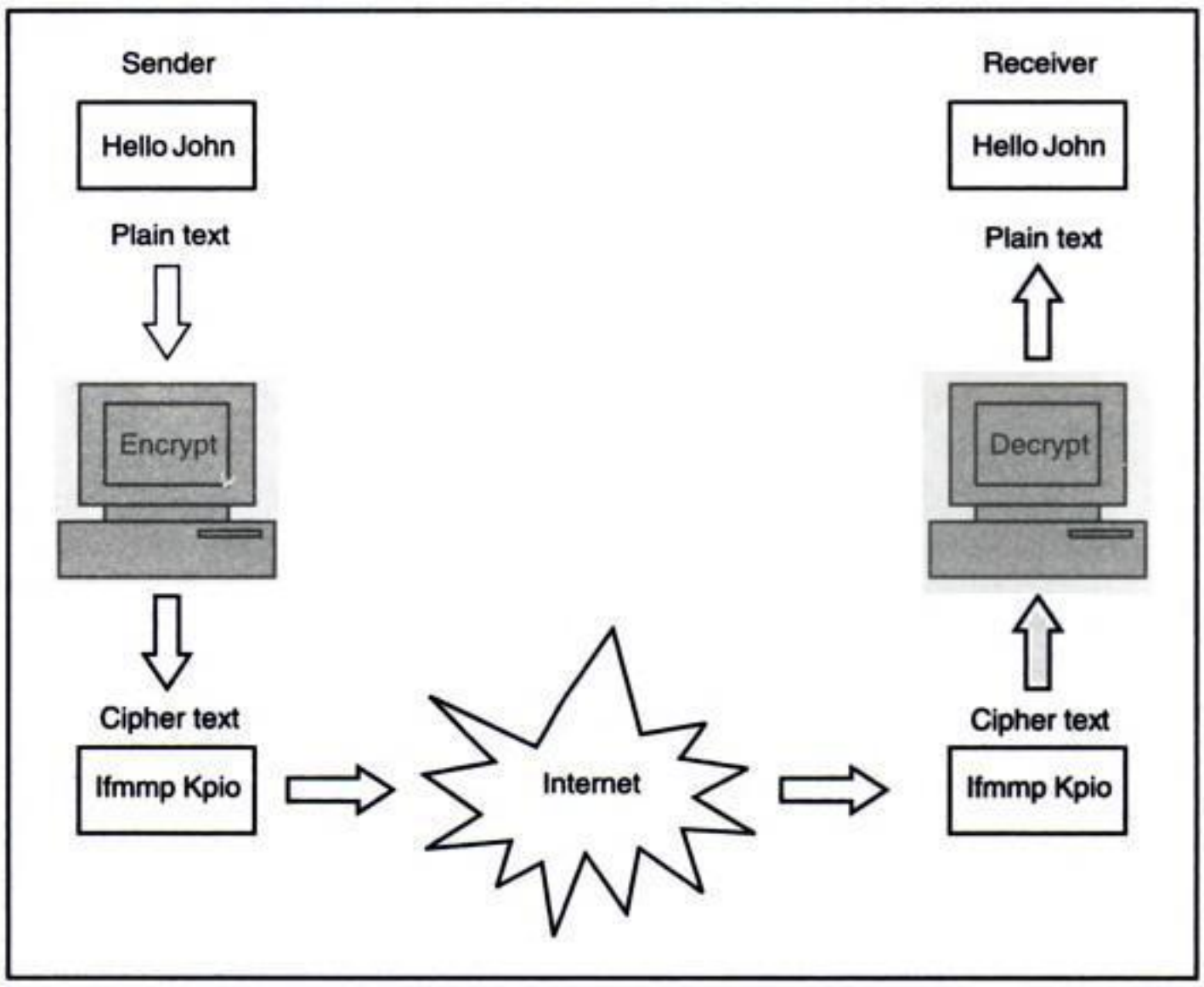


Fig. 2.20 *Encryption and decryption in the real world*

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 not be able 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 mean-

ingful communication to take place. The algorithm basically takes one text as input and produces another as the output.

The second aspect of performing encryption and encryption 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.

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

This is shown in Fig. 2.21.

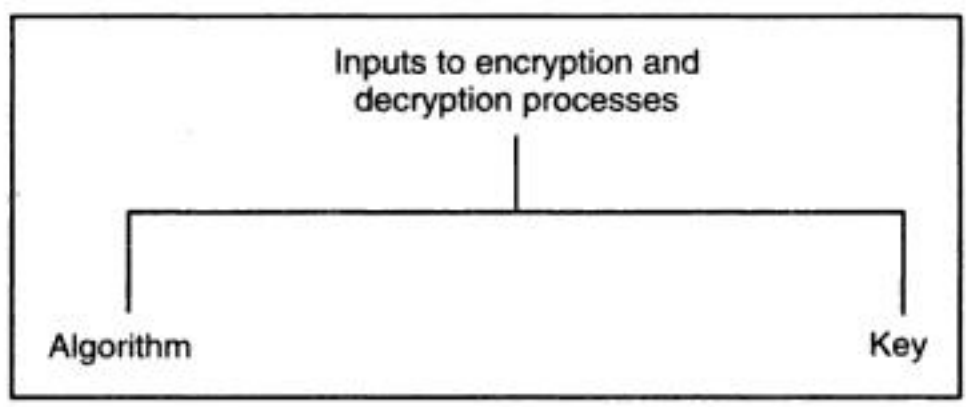


Fig. 2.21 *Aspects of encryption and decryption*

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.

Note 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 as **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 as **Asymmetric Key Cryptography**. This is shown in Fig. 2.22.

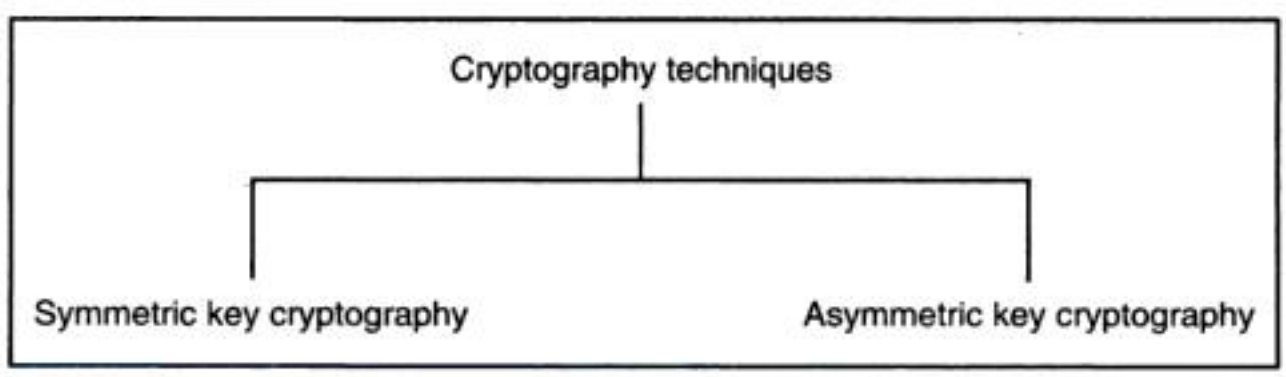


Fig. 2.22 *Cryptography techniques*

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. 2.23.

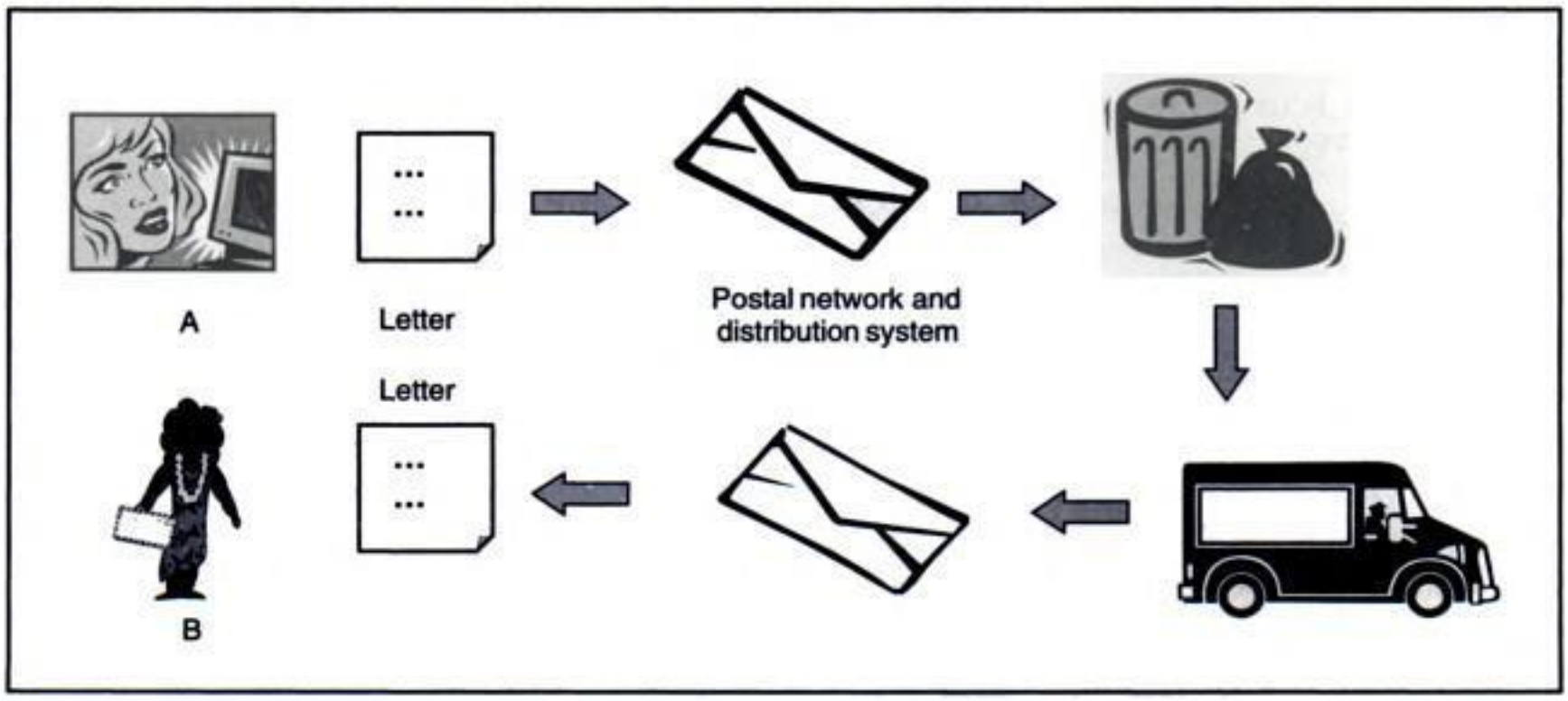


Fig. 2.23 *Simplest way to send a confidential letter*

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 reseal 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 full proof.

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 send 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 full-proof 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 full-proof, 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 as *symmetric key operation* (when used in the context of cryptography, this operation is called as **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 much 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. 2.24. It must also somehow send the respective lock-opening keys to B and C.

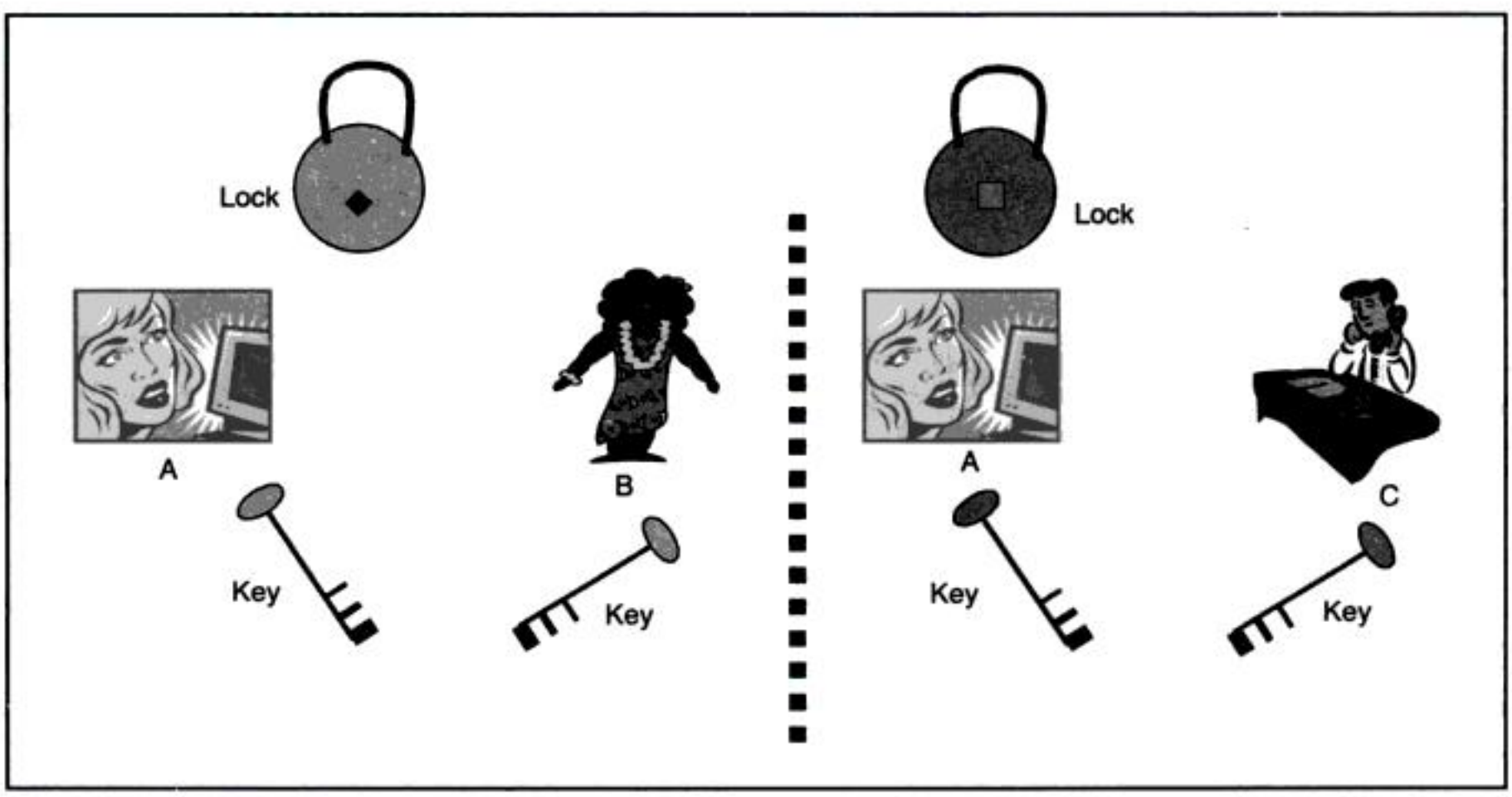


Fig. 2.24 *Use of separate locks and keys per communication pair*

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 Table 2.2 to see if any pattern emerges.

Table 2.2 *Number of parties and correspondingly number of lock-and-key pairs required*

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)

We can see that:

- 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 as 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 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. 2.25.

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

Fig. 2.25 *Diffie-Hellman key exchange algorithm*

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. 2.26.

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:

Note Diffie-Hellman key exchange algorithm gets it 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:
$K2 = A^y \bmod n$.

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.

   Let n = 11, g = 7.

2. Alice chooses another large random number x, and calculates A such that:
   $A = g^x \bmod n$

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

3. Alice sends the number A to Bob.

   Alice sends 2 to Bob.

4. Bob independently chooses another large random integer y and calculates B such that:
   $B = g^y \bmod n$

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

5. Bob sends the number B to Alice.

   Bob sends 4 to Alice.

6. A now computes the secret key K1 as follows:
   $K1 = B^x \bmod n$

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

7. 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. 2.26 *Example of Diffie-Hellman key exchange*

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. Hence 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 do 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. *Problems with the algorithm*

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

Diffie-Hellman key exchange algorithm can fall pray to the **man-in-the-middle attack** (or to be politically correct, *woman-in-the-middle attack*), also called as **bucket brigade attack**. The way this happens is as follows.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Alice	Tom	Bob
K1 = B^x mod n	K1 = B^x mod n	K2 = A^y mod n
= 4^3 mod 11	= 8^8 mod 11	= 9^9 mod 11
= 64 mod 11	= 16777216 mod 11	= 387420489 mod 11
= 9	= 5	= 5
	K2 = A^y mod n	
	= 2^6 mod 11	
	= 64 mod 11	
	= 9	

Fig. 2.32 *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 9 with Alice and 5 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!

However, not everything is lost. If we think deeply and try to come up with the scheme that will still work, an alternatives emerges: namely an **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 (K1) that can seal the lock, and sends the lock and key K1 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 (K2), which is obtained by B from T along with the lock and key K1, only which can open the lock. It is guaranteed that no other key, and of course, including the one used by A (i.e. K1) for locking, can open the lock. Since one key (K1) is used for locking, and *another, different* key (K2) 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 K1 and K2). One key (i.e. K1) can be used for locking, and only the corresponding other key (i.e. K2) from the key pair can be used for unlocking. Thus B can send the lock and key K1 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

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

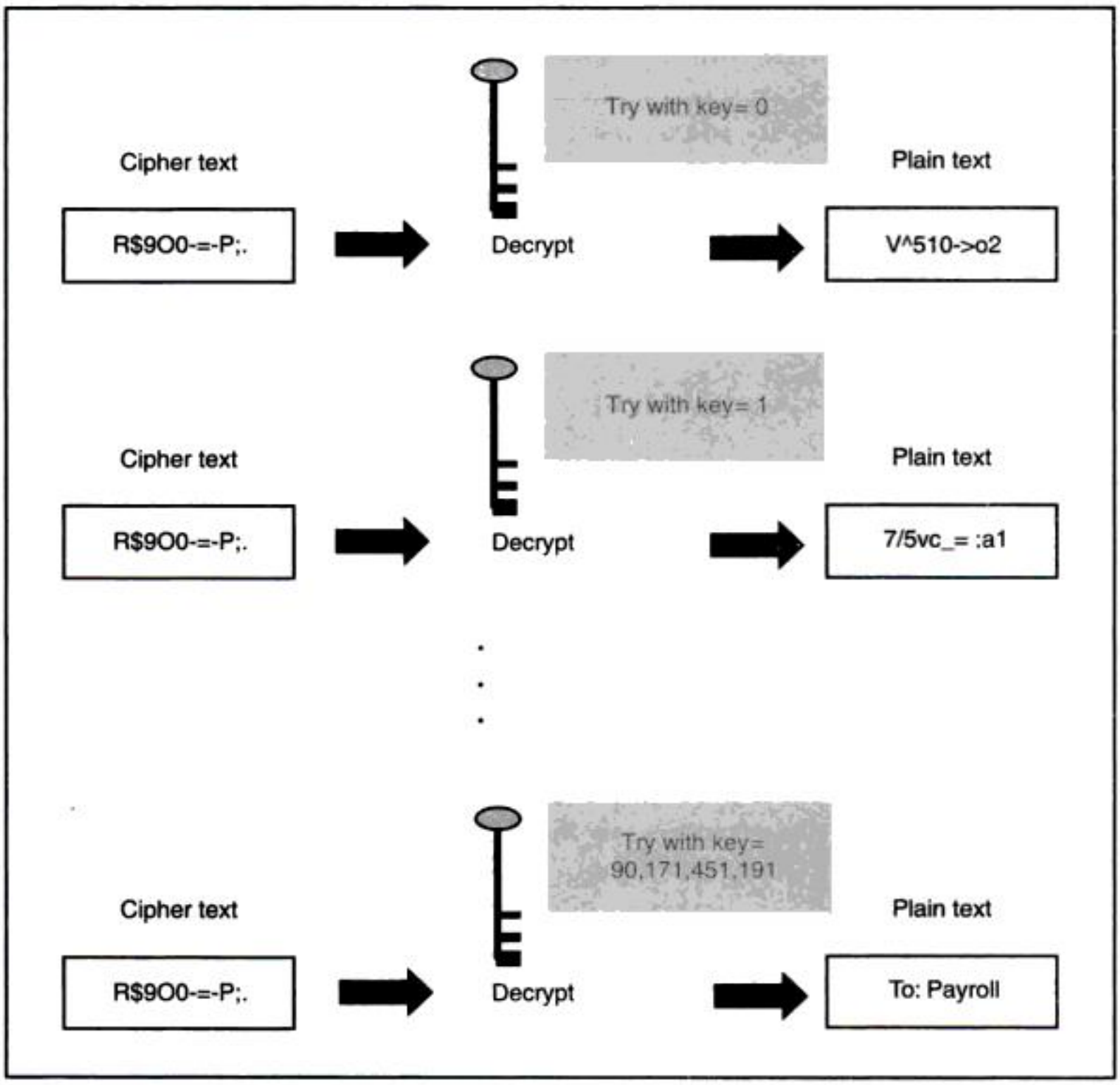


Fig. 2.35 *Brute force attack*

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 is using the binary number system. Thus, our key might be of size 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 it should 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

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Key Terms and Concepts

- Asymmetric Key Cryptography
- Brute-force attack
- Caesar Cipher
- Clear text
- Cryptanalyst
- Decryption
- Encryption
- Homophonic Substitution Cipher
- Man-in-the-middle attack
- One-Time Pad
- Polygram Substitution Cipher
- Running Key Cipher
- Simple Columnar Transposition Technique with multiple rounds
- Transposition Cipher
- Book Cipher
- Bucket brigade attack
- Cipher text
- Cryptanalysis
- Cryptography
- Decryption algorithm
- Encryption algorithm
- Key
- Mono-alphabetic Cipher
- Plain text
- Rail Fence Technique
- Simple Columnar Transposition Technique
- Substitution Cipher
- Symmetric Key Cryptography
- Vernam Cipher

Multiple-choice Questions

1. The language that we commonly use can be termed as _____.
   (a) pure text (b) simple text
   (c) plain text (d) normal text
2. The codified language can be termed as _____.
   (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

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Chapter 3

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, 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

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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 as permutation techniques).

Stream cipher relies only on confusion. Block cipher uses both confusion and diffusion. We leave the question of why this is the case, to the reader.

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.

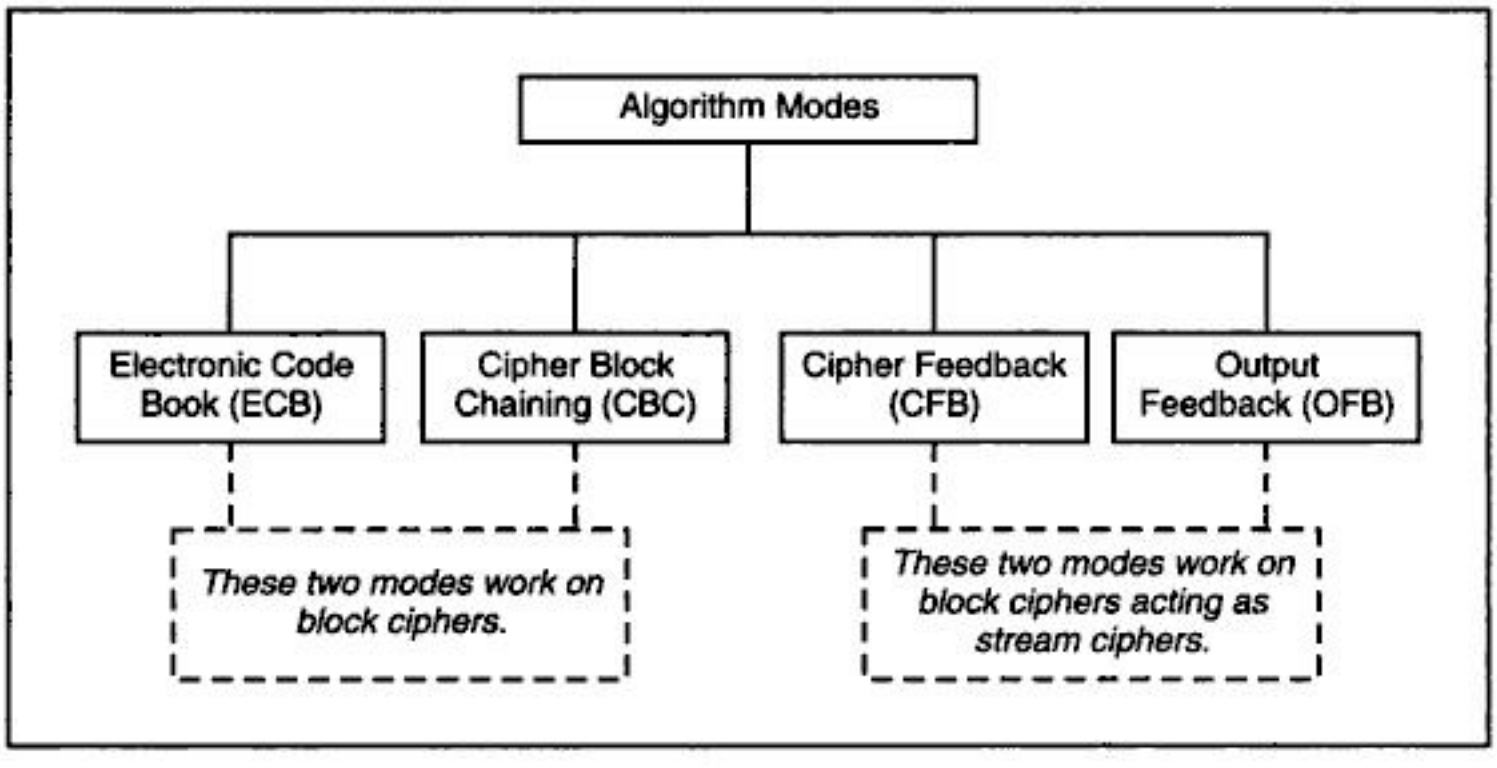


Fig. 3.5 *Algorithm modes*

We shall discuss the algorithm modes in brief now.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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 need 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.

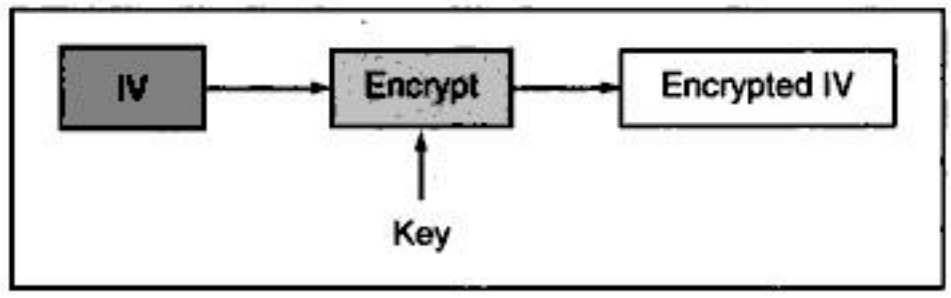


Fig. 3.10 *CFB—Step 1*

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.

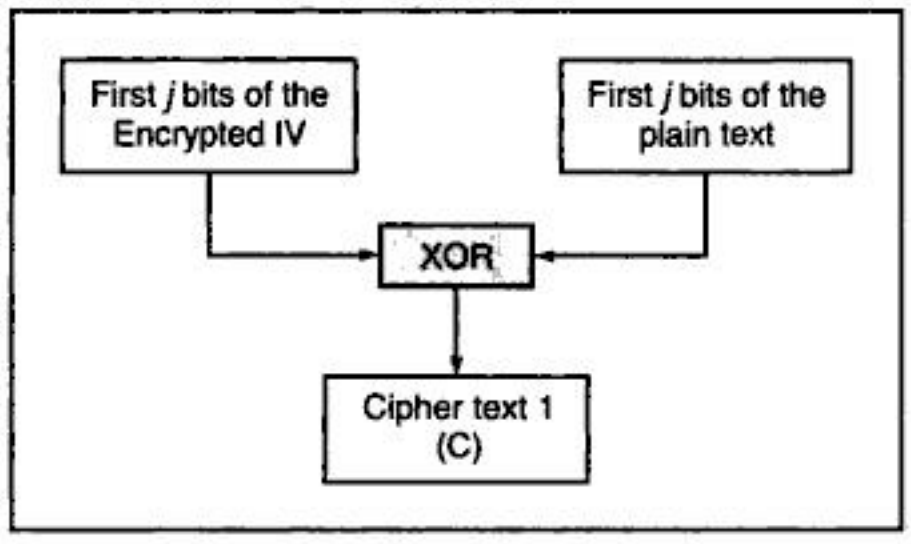


Fig. 3.11 *CFB—Step 2*

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.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

3.4 DATA ENCRYPTION STANDARD (DES)

3.4.1 Background and History

The **Data Encryption Standard (DES)**, also called as 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 as 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 as *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 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. The basic idea is shown in Fig. 3.16.

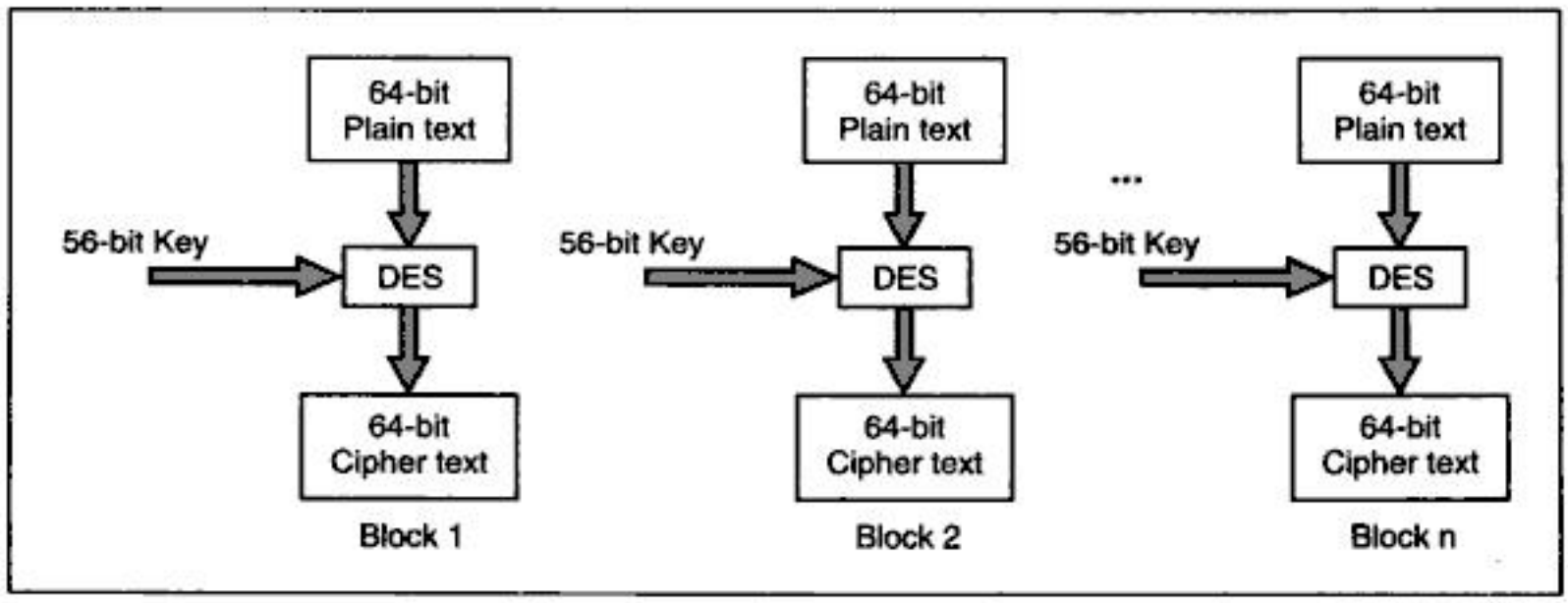


Fig. 3.16 *The conceptual working of DES*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Step 2: Expansion Permutation

Recall that after Initial Permutation, we had two 32-bit plain text areas, called as 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. 3.24.

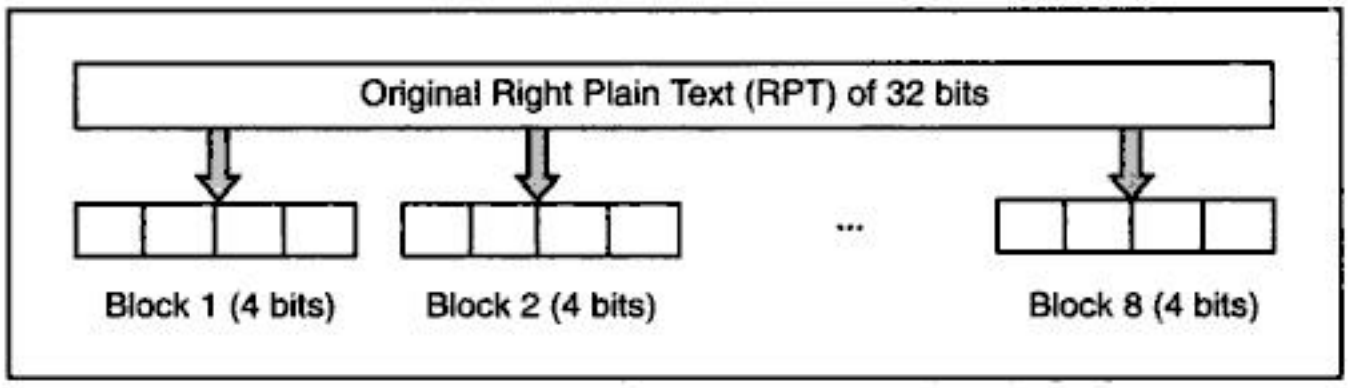


Fig. 3.24 *Division of 32-bit RPT into eight 4-bit blocks*

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. 3.25. Note that the first input bit 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.

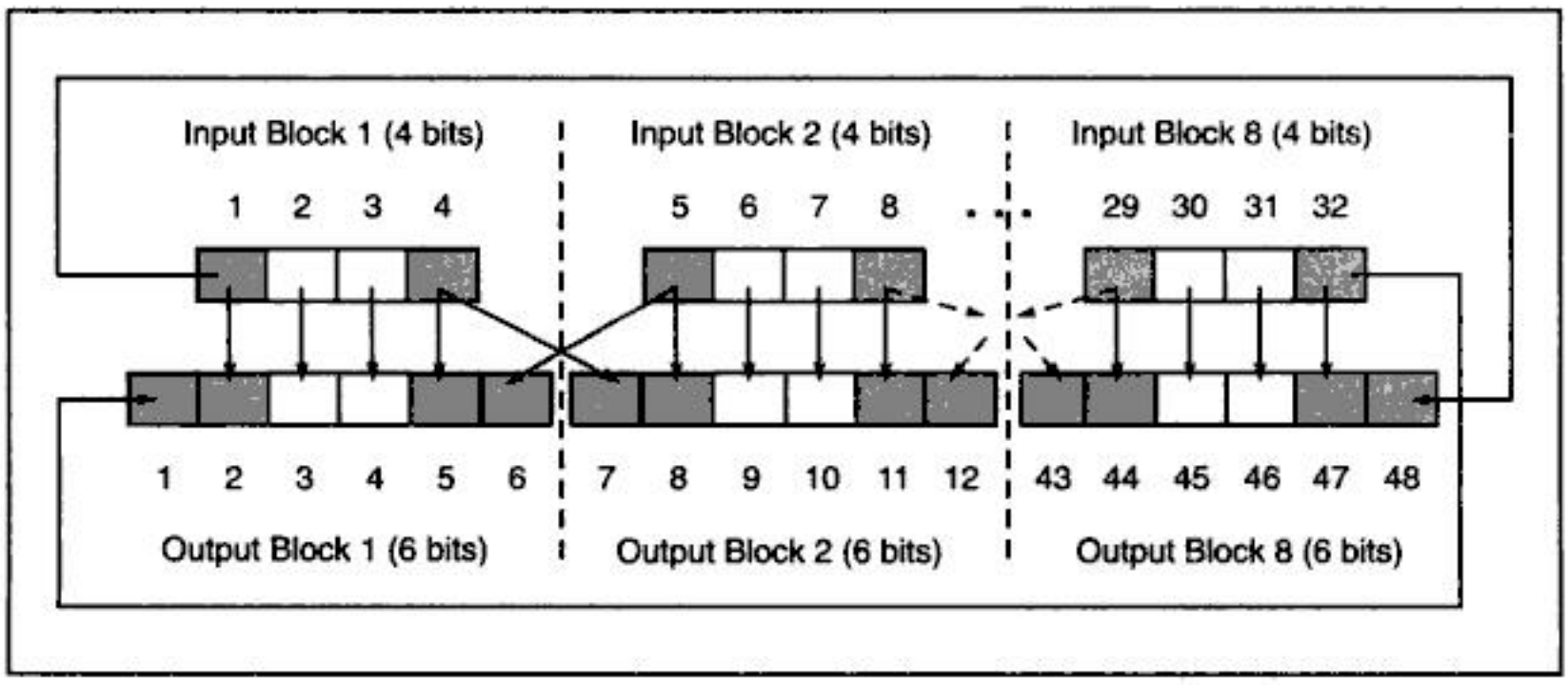


Fig. 3.25 *RPT expansion permutation process*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

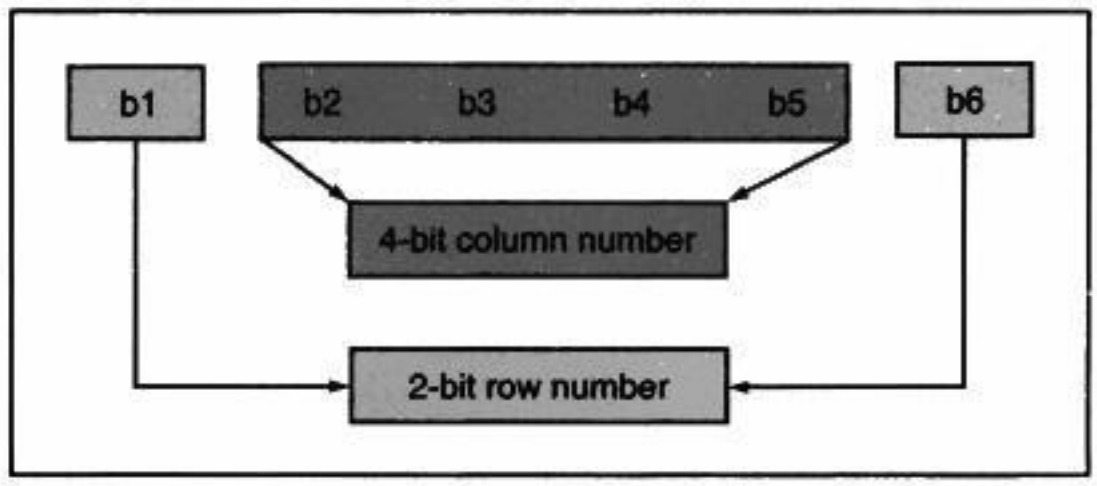


Fig. 3.30 *Selecting an entry in a S-box based on the 6-bit input*

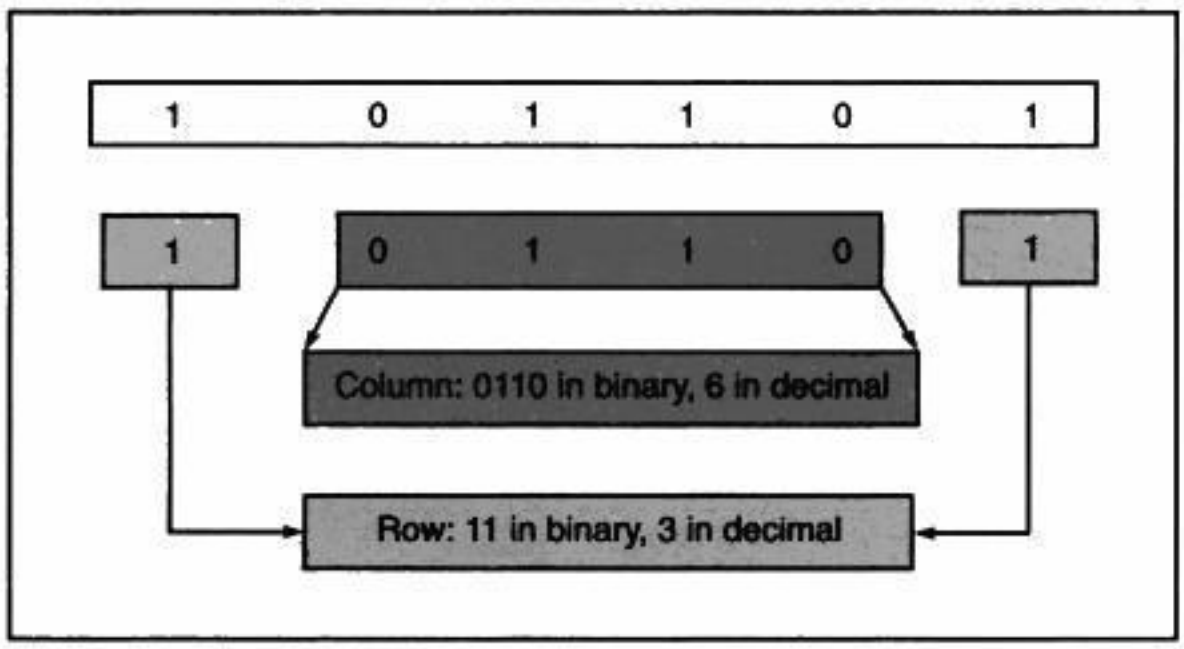


Fig. 3.31 *Example of selection of S-box output based on the input*

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 as **P-box Permutation**. The P-box is shown in Fig. 3.32. For example, a 16 in the first block indicates that the bit at position 16 of the original input moves to bit at position 1 in the 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.

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. 3.32 *P-box permutation*

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

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Now, the aim of the cryptanalyst, who is armed with the knowledge of P and C, is to obtain the values of K1 and K2. What would the cryptanalyst do?

Step 1: For all possible values (2^{56}) of key K1, 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 56-bit key, which makes the task a lot harder). Refer to Fig. 3.38.

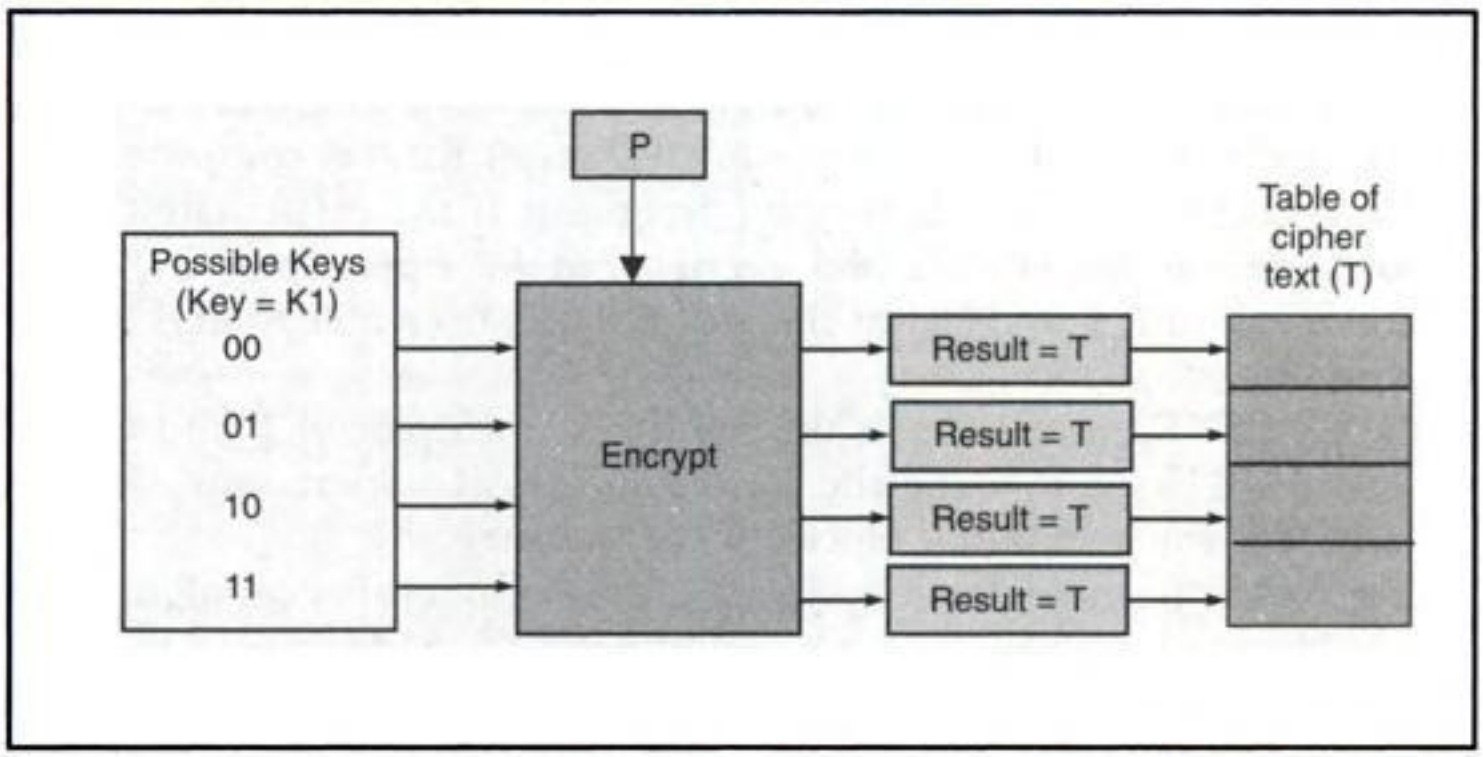


Fig. 3.38 *Conceptual view of the cryptanalyst's encrypt operation*

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, 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. 3.39.

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 equation:

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

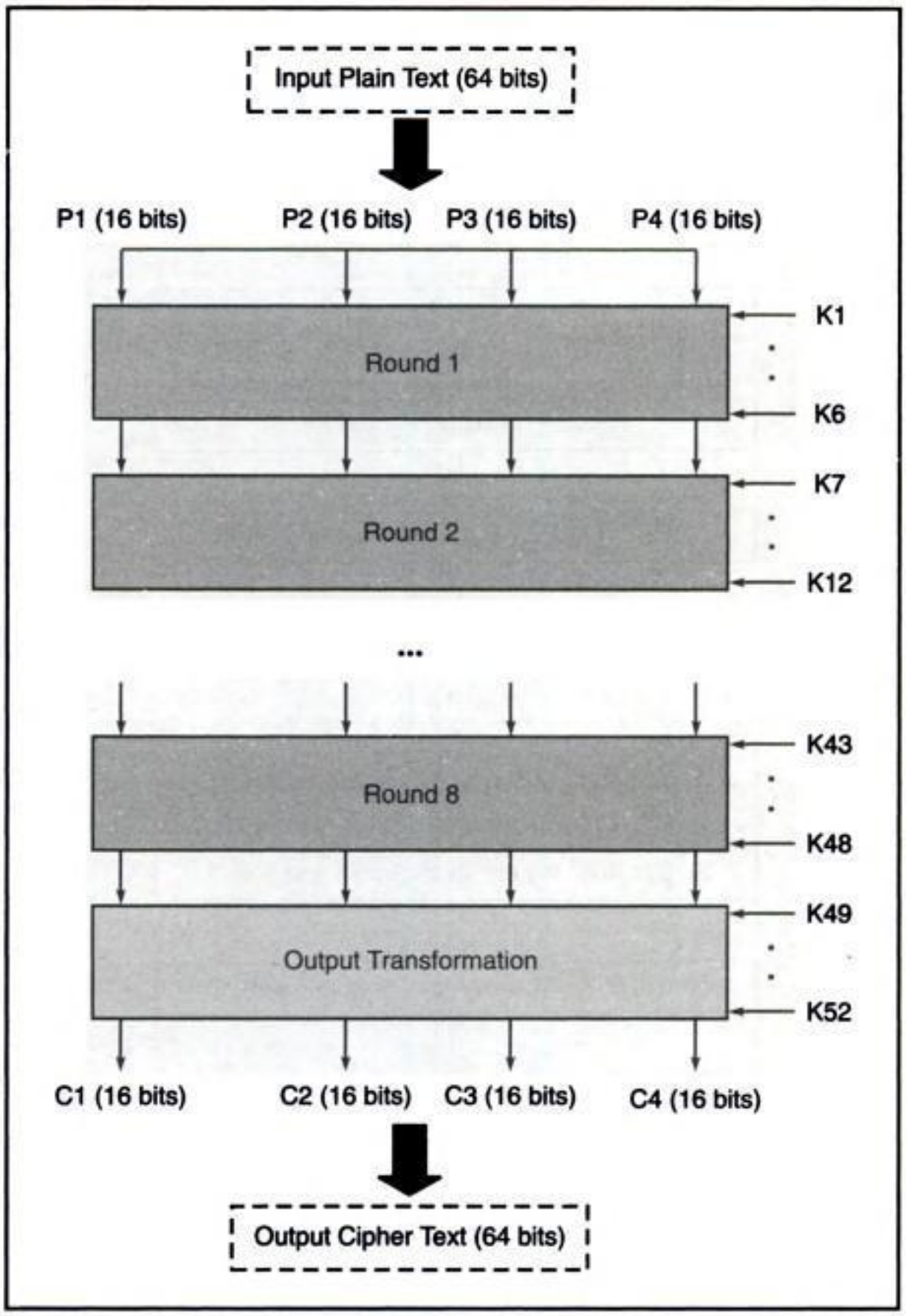


Fig. 3.42 *Broad level steps in IDEA*

described as shown in Fig. 3.43. 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 mod b* is the remainder of the division a/b. Thus, for example, *5 mod 2* is 1 (because the remainder of 5/2 is 1), and *5 mod 3* is 2 (because the remainder of 5/3 is 2)].

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

ing 65 bits? 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. 3.47.

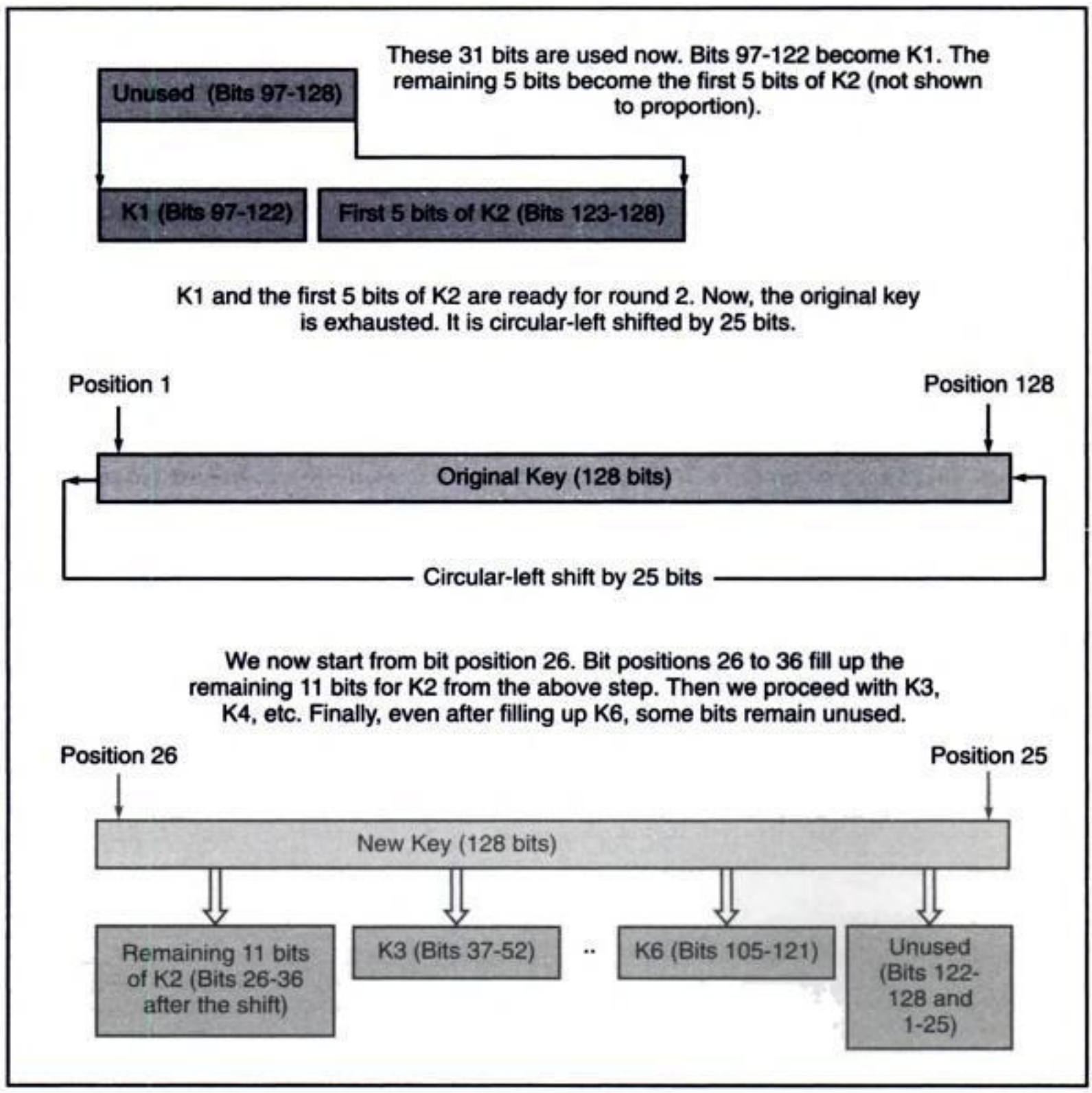


Fig. 3.47 *Circular-left key shift and its use in sub-key generation for round 2*

We can now imagine that the unused bits of the second *round* (i.e. bit positions 122-128 and 1-25) will firstly be used in *round* 3, and then a circular-left shift of 25 bits will be performed on the last key shown in the above figure. This will mean that the new key will now start with bit position number 51 (original start of 26th bit + circular-left shift of 25 bits). Also, the end bit now will be bit number 50 (original end of 25th bit + circular-left shift of 25 bits). This key will now be used for the rest of the *round* 3. Again, its last portions

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

working of RC5 using Fig. 3.50. 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 text block with size 64-bits. The same principles of 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 sub-keys (we shall later see how they are generated)

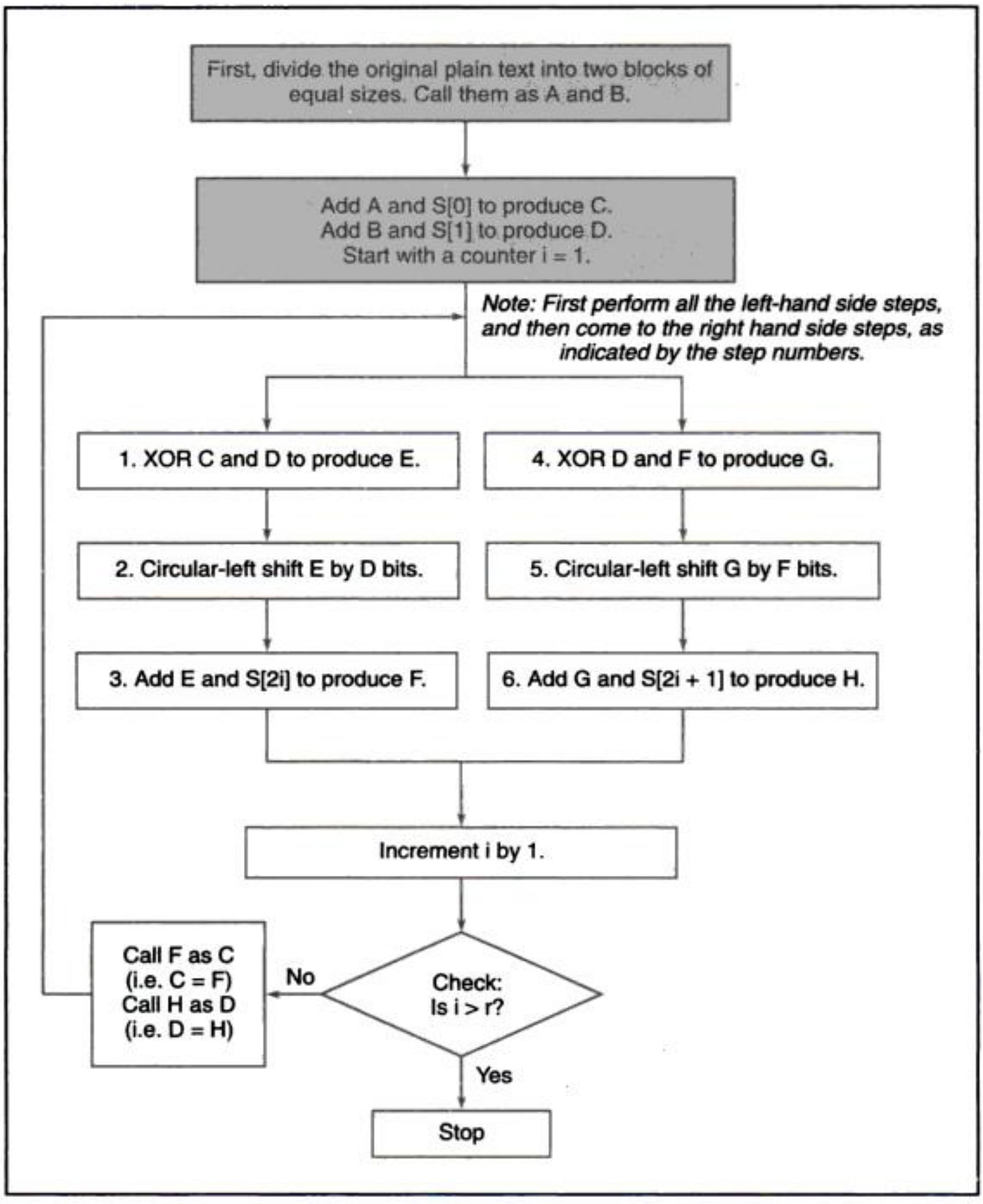


Fig. 3.50 *Encryption using RC5*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
For i = 1 to 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. 3.60 *Mathematical representation of RC5 decryption*

6. *Sub-key creation*

Let us now examine the sub-key creation process in RC5. This is a two-step process.

1. In the first step, the sub-keys (denoted by S[0], S[1], ...) are generated.
2. The original key is called as L. In the second step, the sub-keys (S[0], S[1], ...) are mixed with the corresponding sub-portions of the original key (i.e. L[0], L[1], ...).

This is shown in Fig. 3.61.

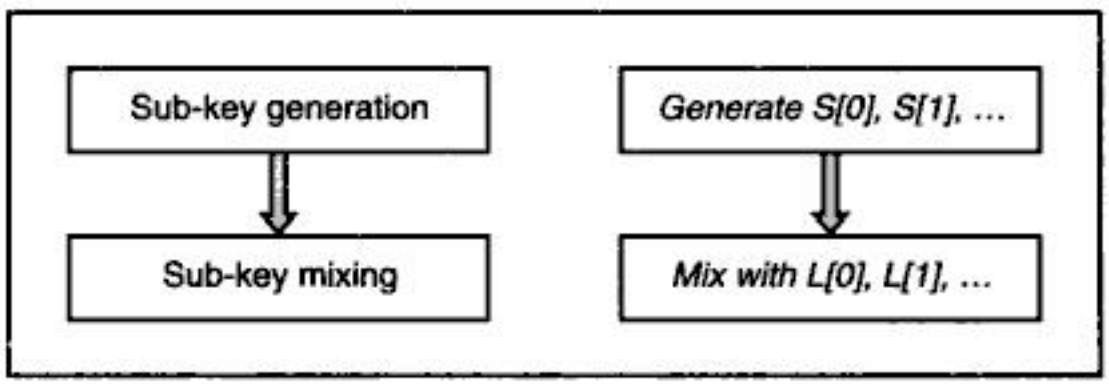


Fig. 3.61 *Sub-key generation process*

Let us understand these two steps.

Step 1: Sub-key generation

In this step, two constants P and Q are used. The array of sub-keys to be generated is called as S. The first sub-key S[0] is initialised with the value of P.

Each next sub-key (i.e. S[1], S[2], ...) 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 sub-keys S[0], S[1], ... S[25].

This process is illustrated in Fig. 3.62.

The mathematical form of this sub-key generation is shown in Fig. 3.63.

Step 2: Sub-key mixing

In the sub-key mixing stage, the sub-keys S[0], S[1], ... are mixed with the sub-portions of the original key, i.e. L[0], L[1], ... L[c]. Note that c is the last sub-key position in the original key. This process is shown mathematically in Fig. 3.64.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

3.8 ADVANCED ENCRYPTION STANDARD (AES)

3.8.1 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 as the **Advanced Encryption Standard (AES)**. Many proposals were submitted, and after a lot of debate, an algorithm called as **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 as 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 short listed in August 1999. In October 2000, Rijndael was announced as the final selection for AES.

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.

3.8.2 Operation

In Rijndael, the plain text blocks and keys can be arranged in variable sizes (16, 24 or 32 bytes). The logical view of such a block (of either plain text or key) is shown in Fig. 3.66.

Rijndael consists of 10, 12 or 14 rounds. Each round contains four steps. Each operation performs a specific function. This idea is shown in Fig. 3.67.

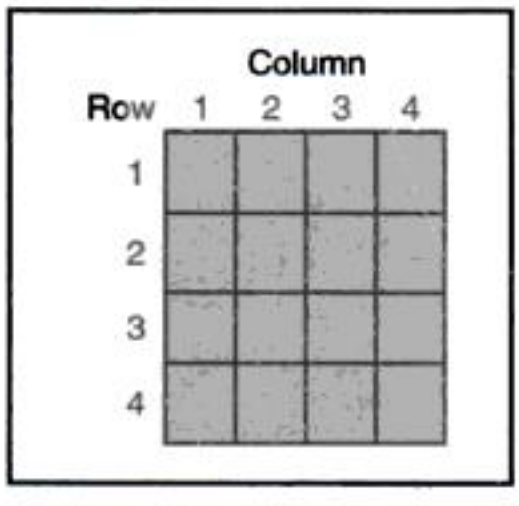


Fig. 3.66 *Initial plain text/key block*

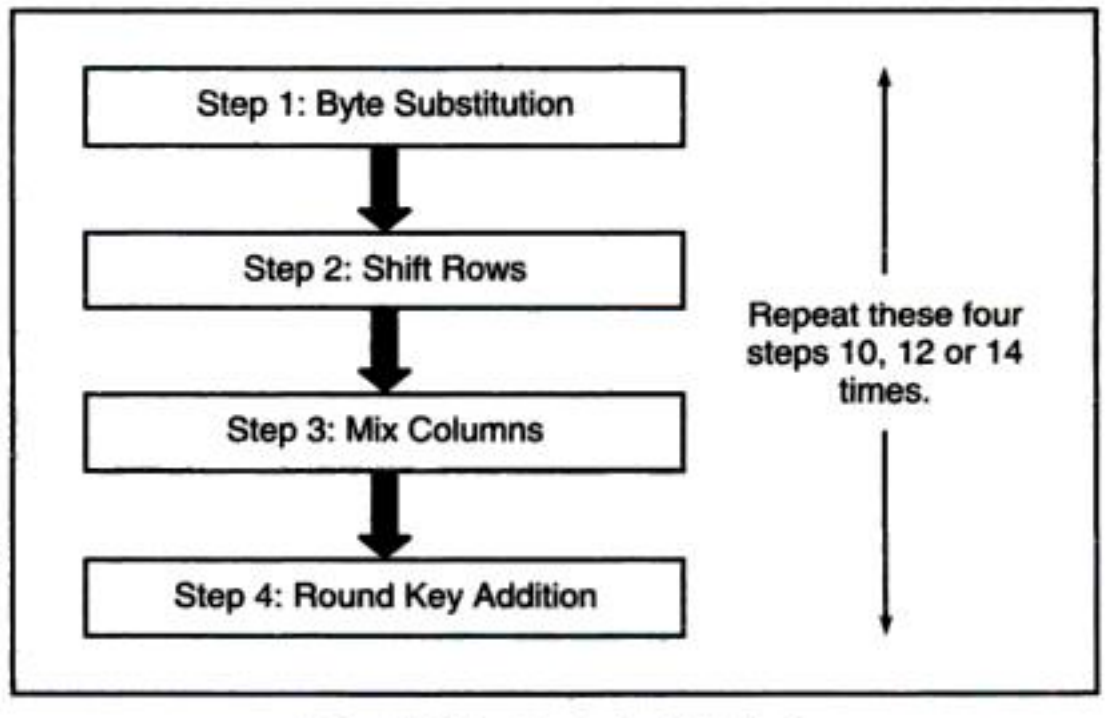


Fig. 3.67 *Steps in Rijndael*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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

Review Questions

1. Distinguish between stream and block ciphers.
2. Discuss the idea of algorithm modes.
3. What is the idea behind meet-in-the-middle attack?
4. How can the same key be reused in triple DES?
5. Distinguish between differential and linear cryptanalysis.
6. Find out more about the vulnerabilities of DES from the Internet.

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.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

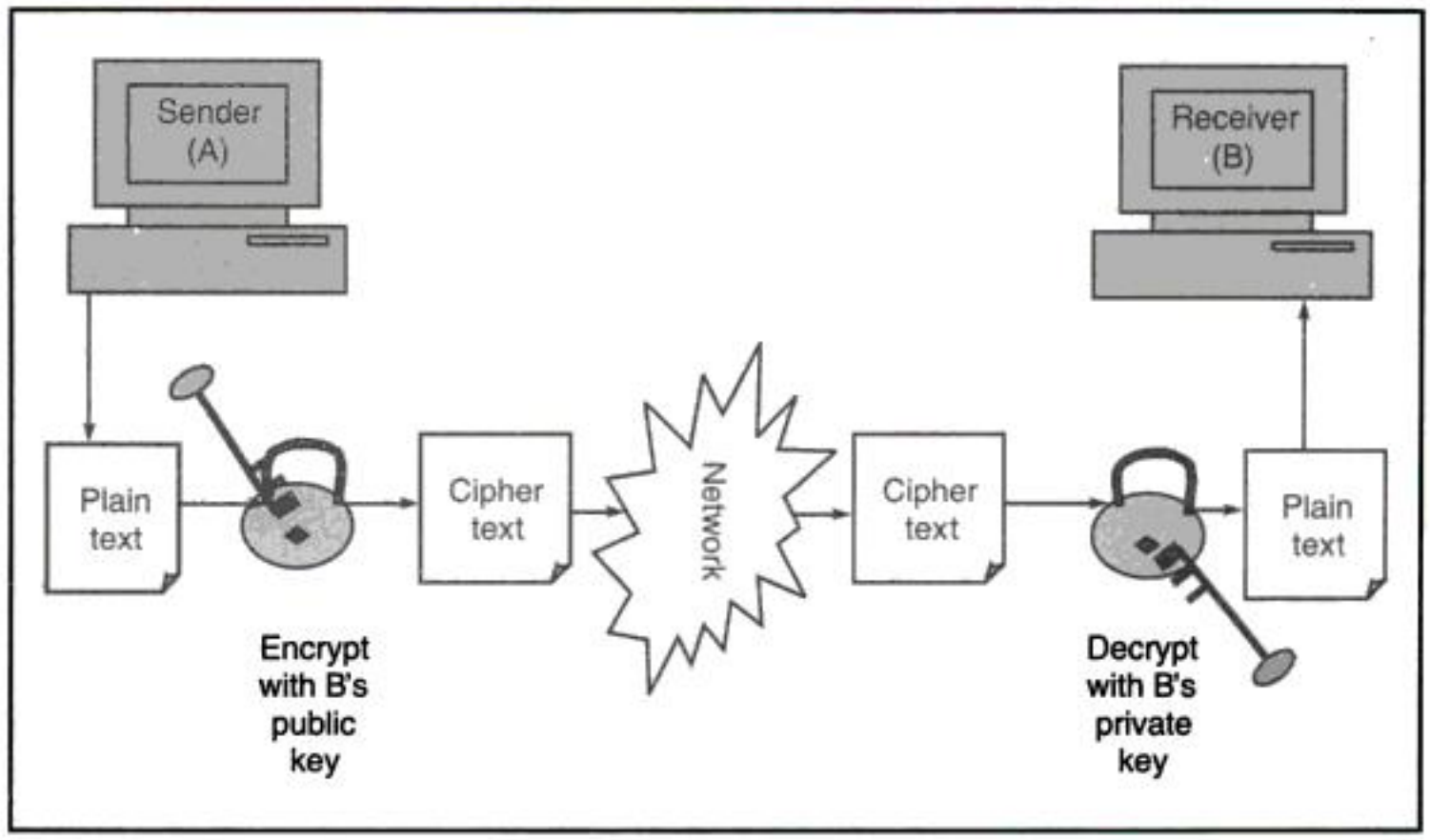


Fig. 4.1 *Asymmetric key cryptography*

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 bank can perform the decryption, as it alone knows its private key. This concept is shown in Fig. 4.2. 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.

Note A prime number is 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, ... are non-prime numbers. A quick observation can also be made that a prime number above 2 must be an odd number

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

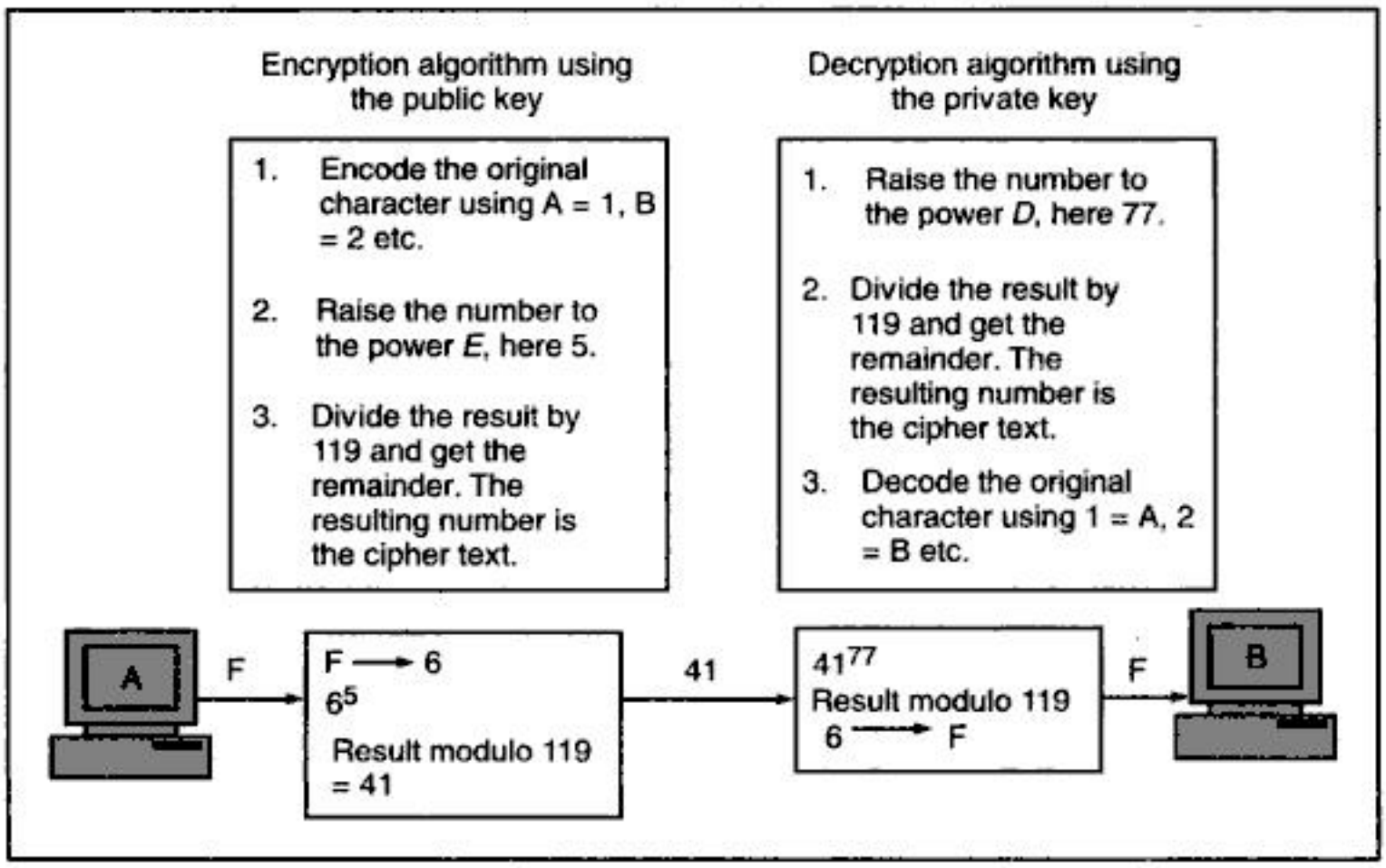


Fig. 4.5 *Example of the RSA algorithm*

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, 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.

4.5 SYMMETRIC AND ASYMMETRIC KEY CRYPTOGRAPHY TOGETHER

4.5.1 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

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

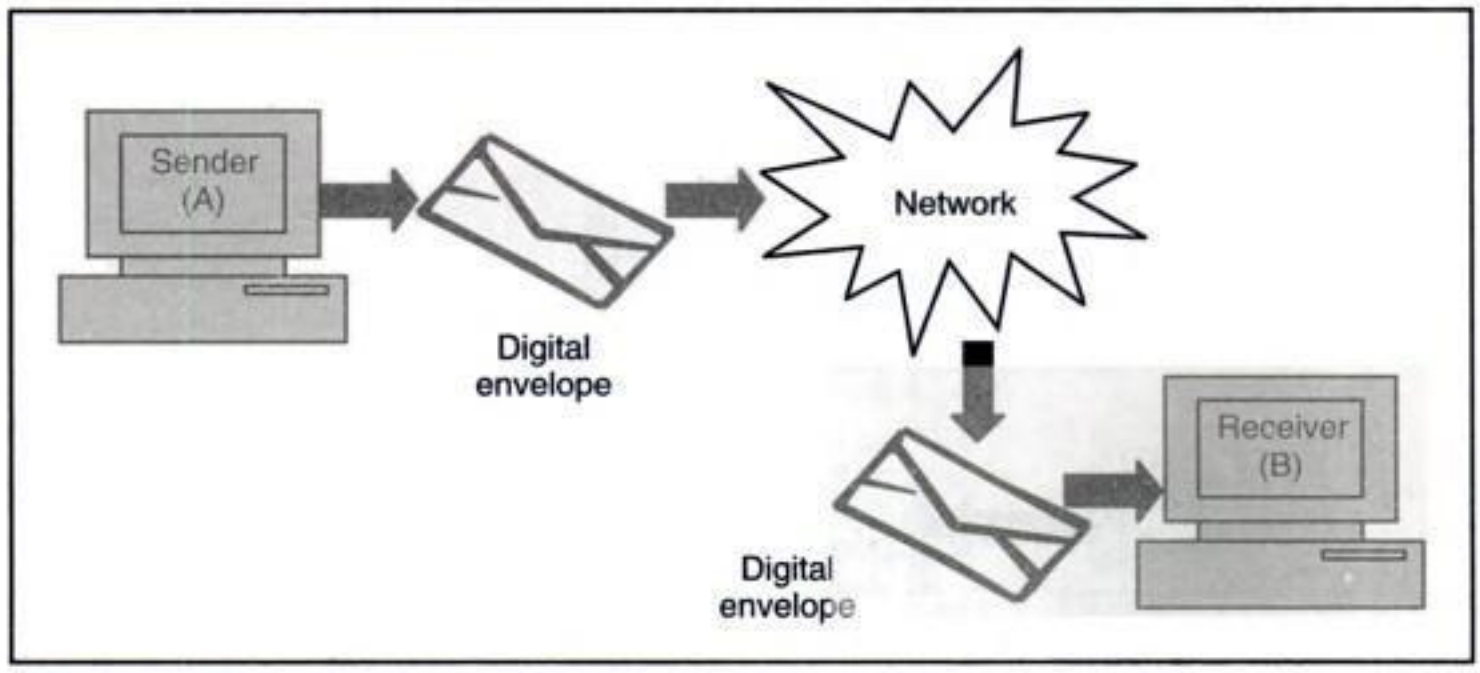


Fig. 4.9 *Digital envelope reaches B over the network*

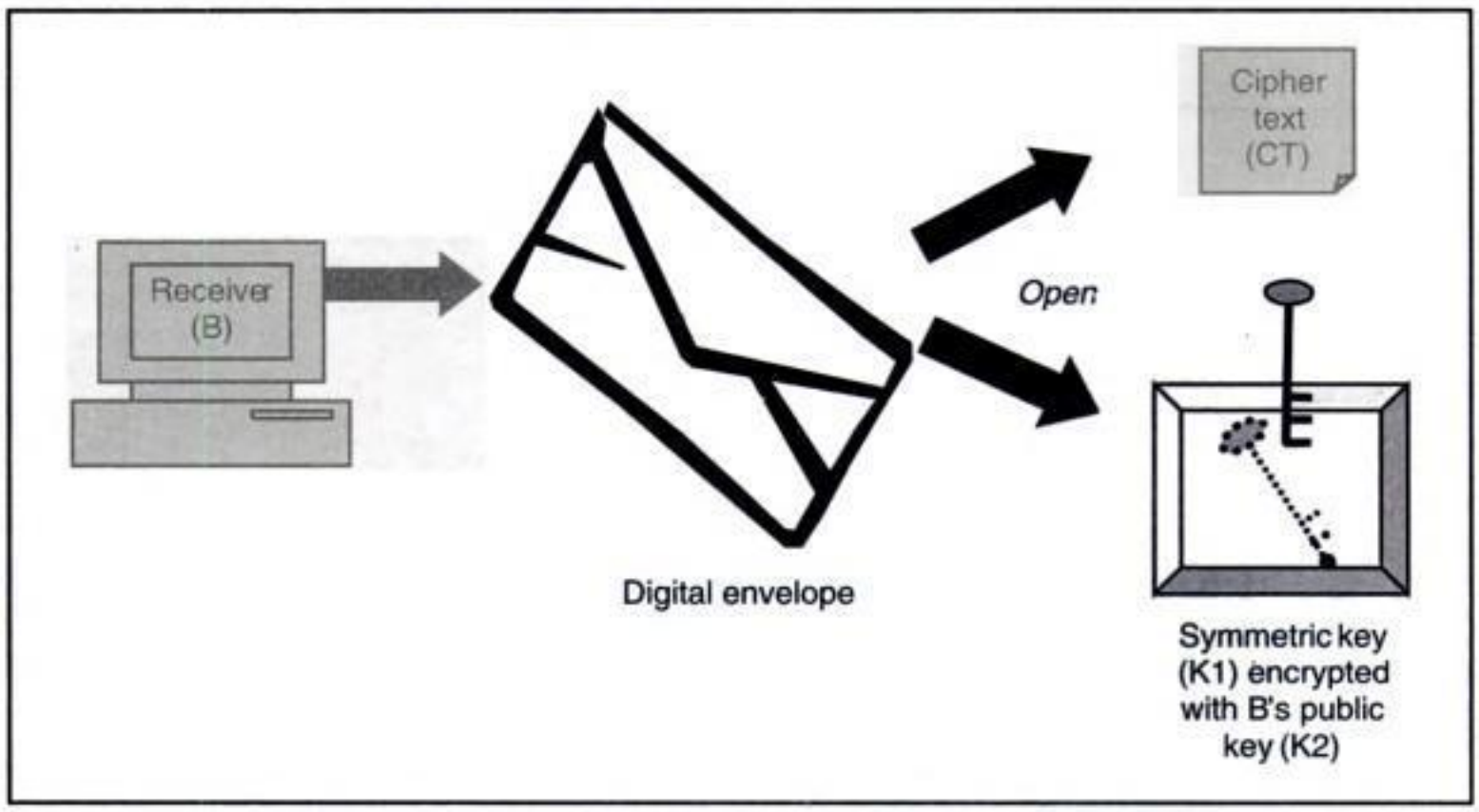


Fig. 4.10 *B opens the digital envelope using her private key*

(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 not also consume a lot of space.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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.

4.6.2 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 as **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. 4.15. 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 as *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

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

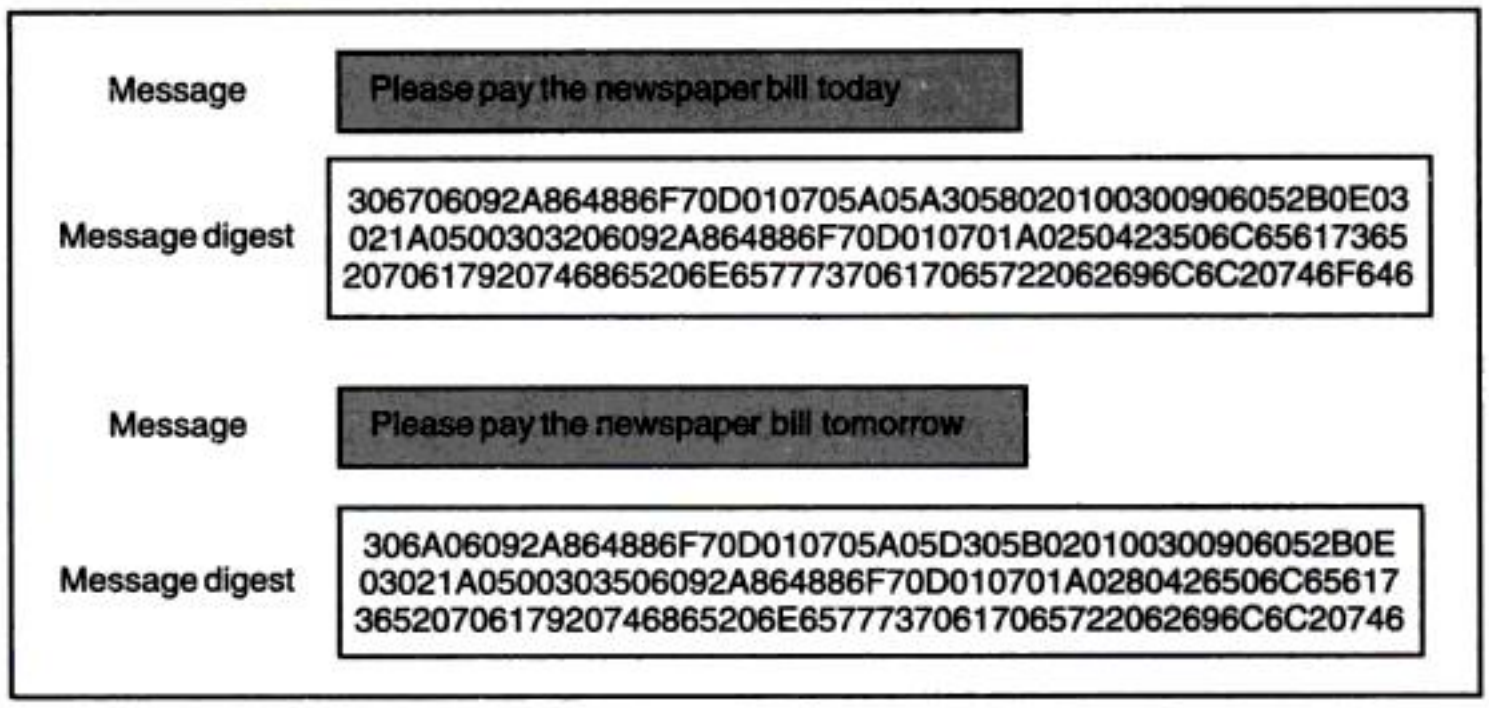


Fig. 4.21 *Message digest example*

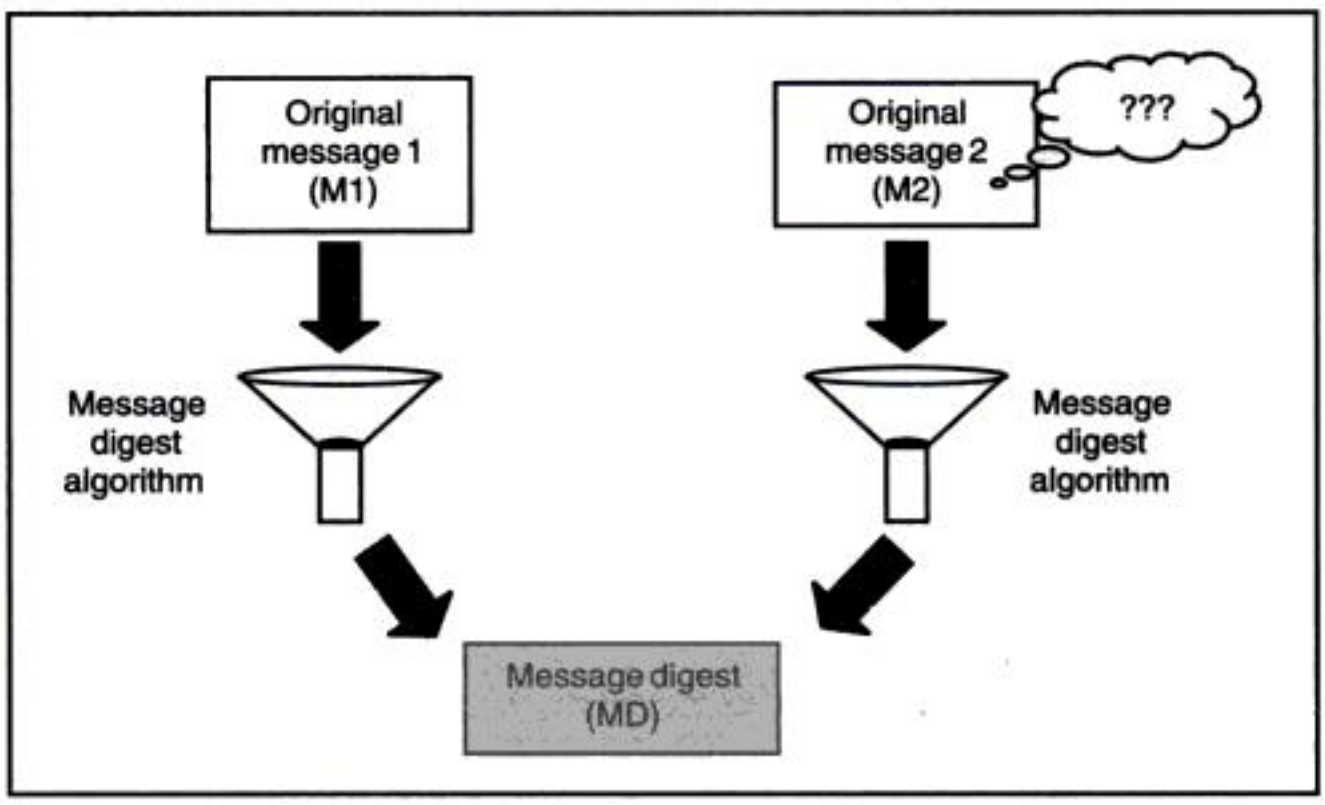


Fig. 4.22 *Message digests should not reveal anything about the original message*

4.6.3 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 as **MD**. He soon came up with its next version, **MD2**. Rivest 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.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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:

3. $w = s^{-1} \bmod q$
   $u1 = (H(m) * w) \bmod q$
   $u2 = (rw) \bmod q$
   $v = ((g^{u1} * y^{u2}) \bmod p) \bmod q$

If v = r, the signature is said to be verified. Otherwise, it is rejected.

4.7 KNAPSACK ALGORITHM

Actually, Ralph Merkle and Martin Hellman developed the first algorithm for public key encryption, called as 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, ..., Mn are the given values and S is the sum, find out bi so that:

S = b1M1 + b2M2 + ... + 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. 4.48.

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. 4.48 *Knapsack example*

4.8 SOME OTHER ALGORITHMS

Let us discuss some other public key algorithms.

4.8.1 Elliptic Curve Cryptography (ECC)

1. *Introduction*

RSA is the most prominent algorithm used in public key cryptography techniques for encryption and digital signatures. Over the years, the key lengths for RSA have been increasing. This puts considerable burden on RSA. Another public key cryptography technique is gaining popularity in the last few years. It is called as **Elliptic Curve Cryptography (ECC)**.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

End

For instance, consider the following example:

To find 7^5 mod 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 mod 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).

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

digital certificates, but also of the 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. 5.11. As shown, the user gets a screen, which informs the user that his digital certificate is ready, and that he should download it from the CA's site.

Fig. 5.11 *CA informs the user that the certificate is ready and can be downloaded*

After the user opts for downloading this certificate, he gets to see the following screen, as shown in Fig. 5.12. 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.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

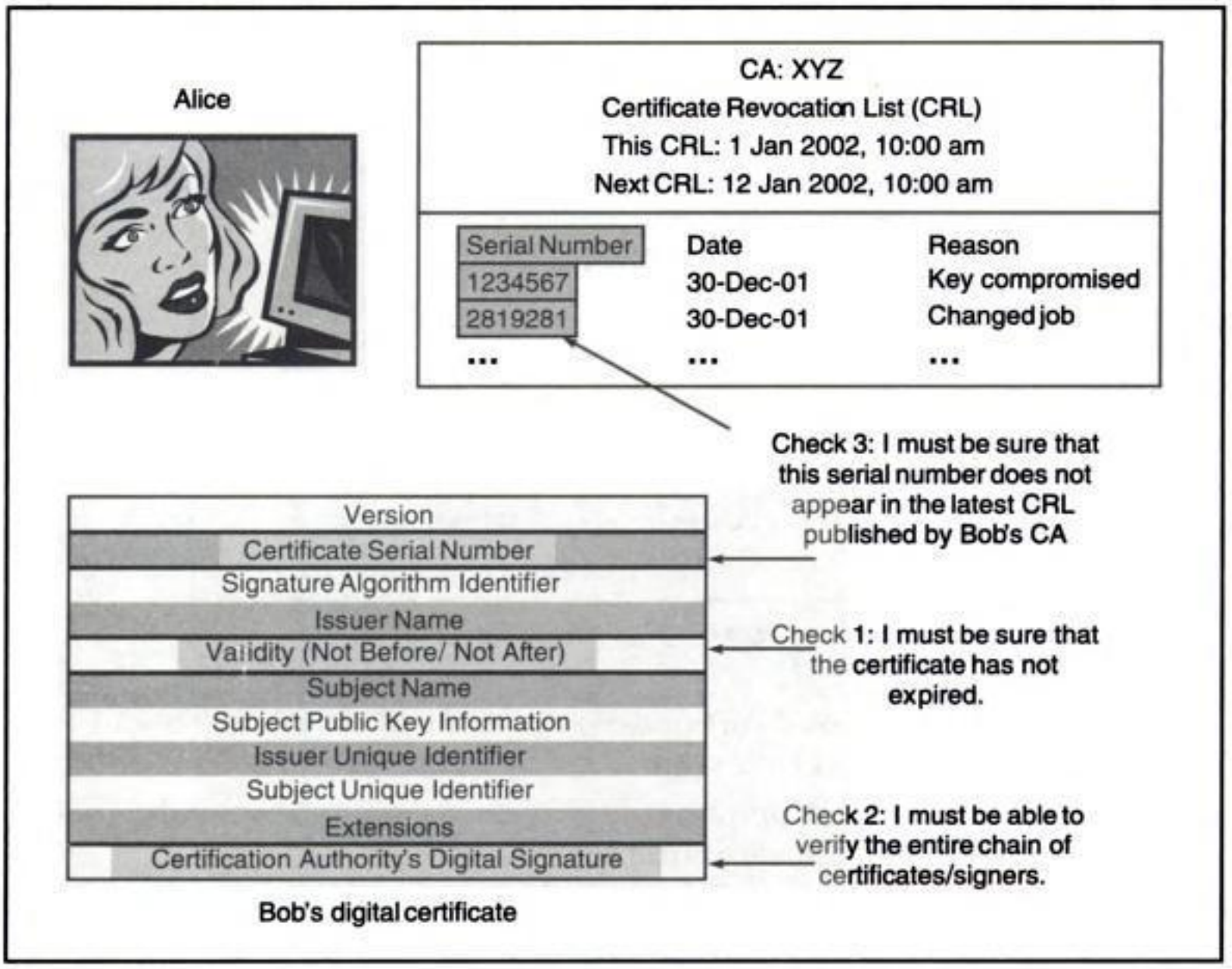


Fig. 5.26 *Validating a certificate and CRL's role in the validation process*

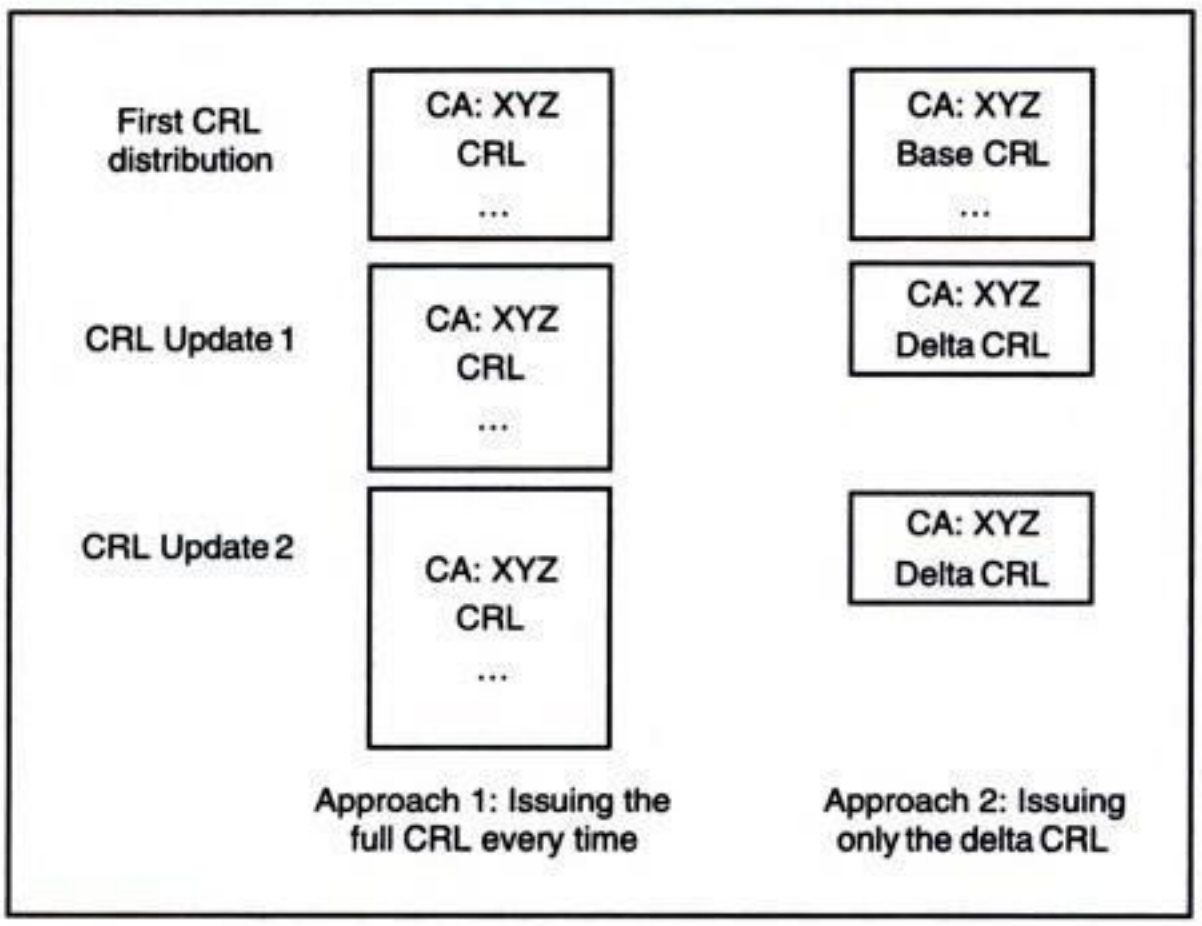


Fig. 5.27 *Delta CRL*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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 center stage 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. 5.38.

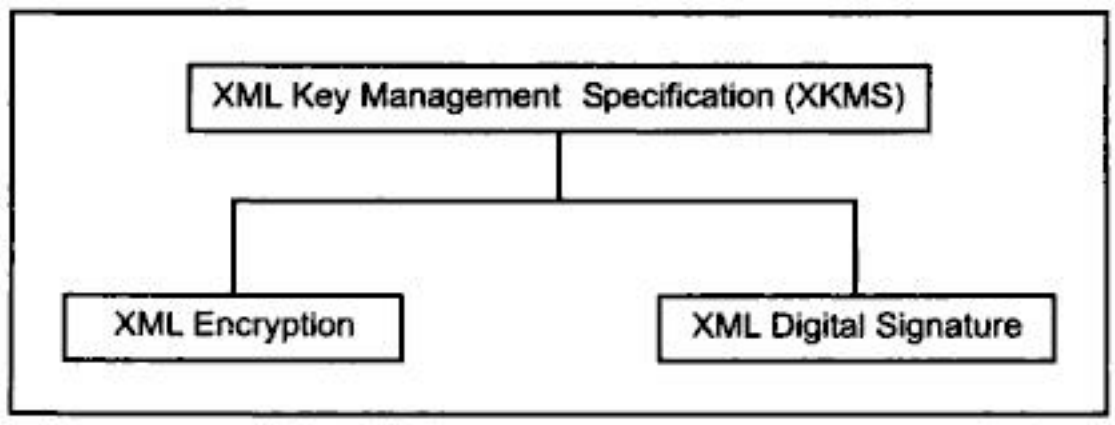


Fig. 5.38 *XML and security*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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.

Chapter Summary

- *Digital certificates solve the problem of key exchange.*
- *Digital certificates can be compared to a person's driving license or passport.*
- *A digital certificate binds a user with the user's public key.*
- *A Certification Authority (CA) can issue digital certificates.*
- *X.509 protocol is used to specify the structure of digital certificates.*
- *Since a CA can have great load, it can offload some of its tasks to a Registration Authority (RA).*
- *Root CA uses self-signed certificates.*
- *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.*
- *CRL is offline check.*
- *OCSP and SCVP are online checks.*
- *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.*
- *PKIX model deals with the issues related to PKI.*
- *PKCS standards touch upon the various aspects of the PKI technology.*
- *XML security is now becoming an important concept.*

Key Terms and Concepts

- Attribute certificate
- Base CRL
- Certificate Management Protocol (CMP)
- Certificate Signing Request (CSR)
- Certification Authority hierarchy
- Cross-certification
- Dictionary attack
- Lightweight Directory Access Protocol (LDAP)
- Proof Of Possession (POP)
- Public Key Cryptography Standards (PKCS)
- Authority Revocation List (ARL)
- Certificate directory
- Certificate Revocation List (CRL)
- Certification Authority (CA)
- Chain of trust
- Delta CRL
- Digital certificate
- Online Certificate Status Protocol (OCSP)
- Psuedo-random number
- Public Key Infrastructure X.509 (PKIX)

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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, 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 transmission medium to the other end.

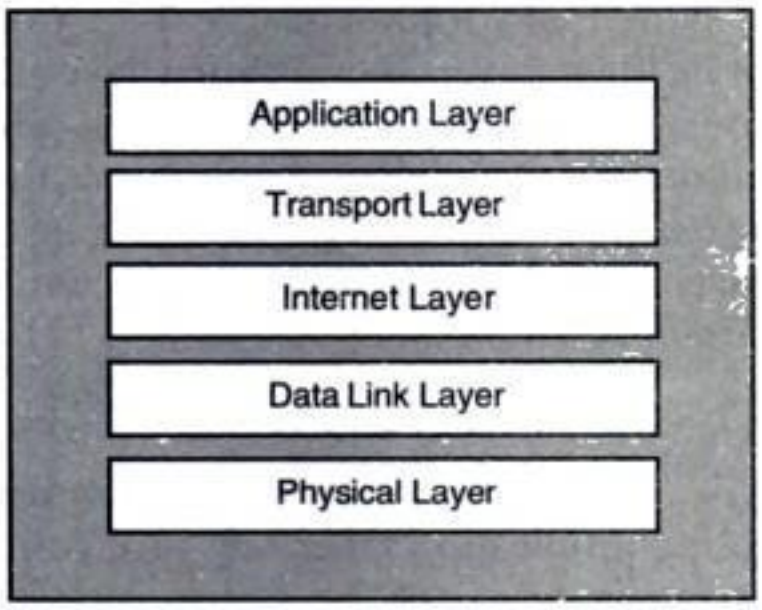


Fig. 6.5 *TCP/IP layers*

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. 6.6. Here, we assume that X is the browser and Y is the Web server.

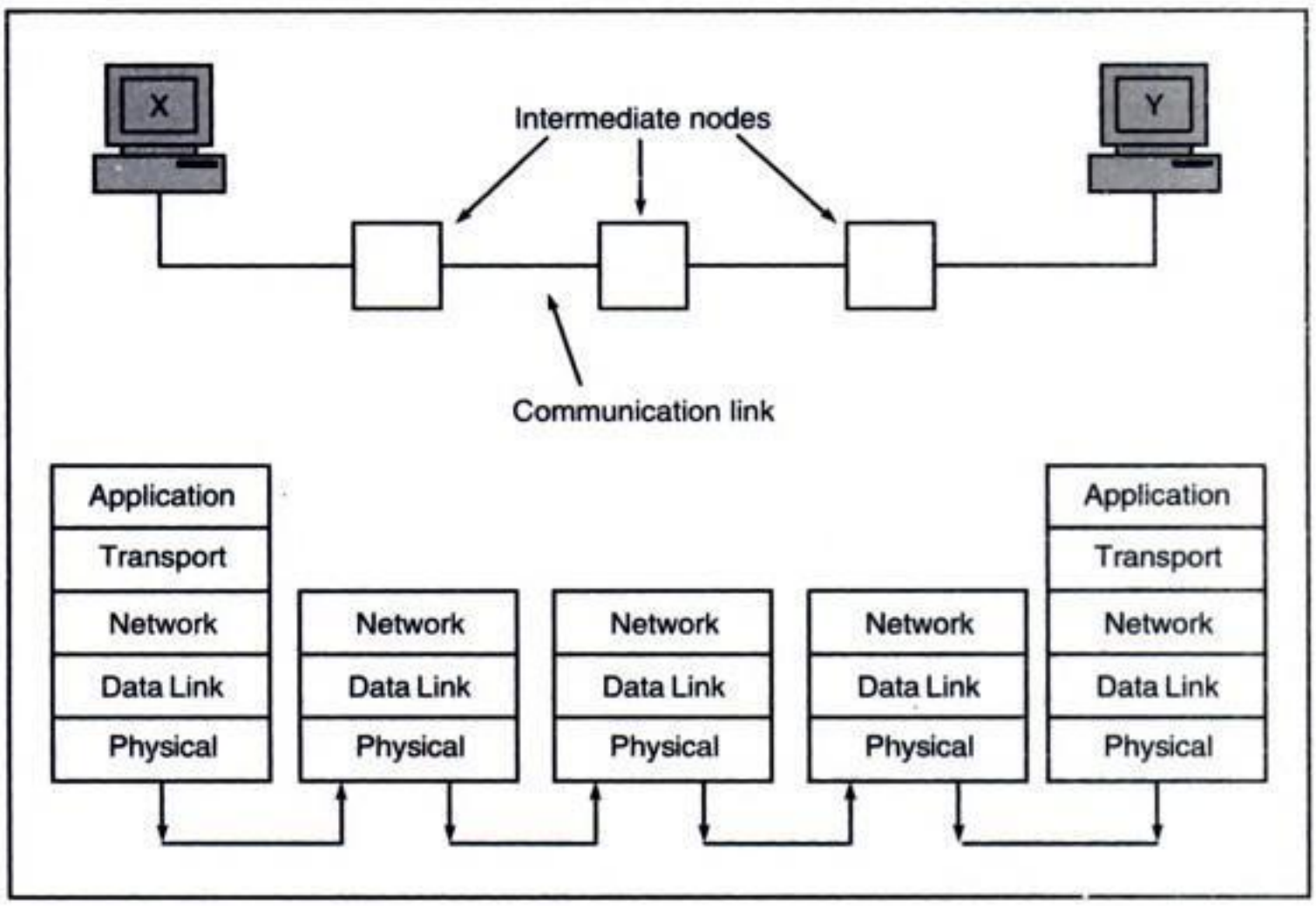


Fig. 6.6 *Communication using TCP/IP*

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).

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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 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).

In 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 the server's 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 recipient of all these messages. This phase contains three steps, as shown in Fig. 6.14. These steps are: **Certificate, Client key exchange,** and **Certificate verify**.

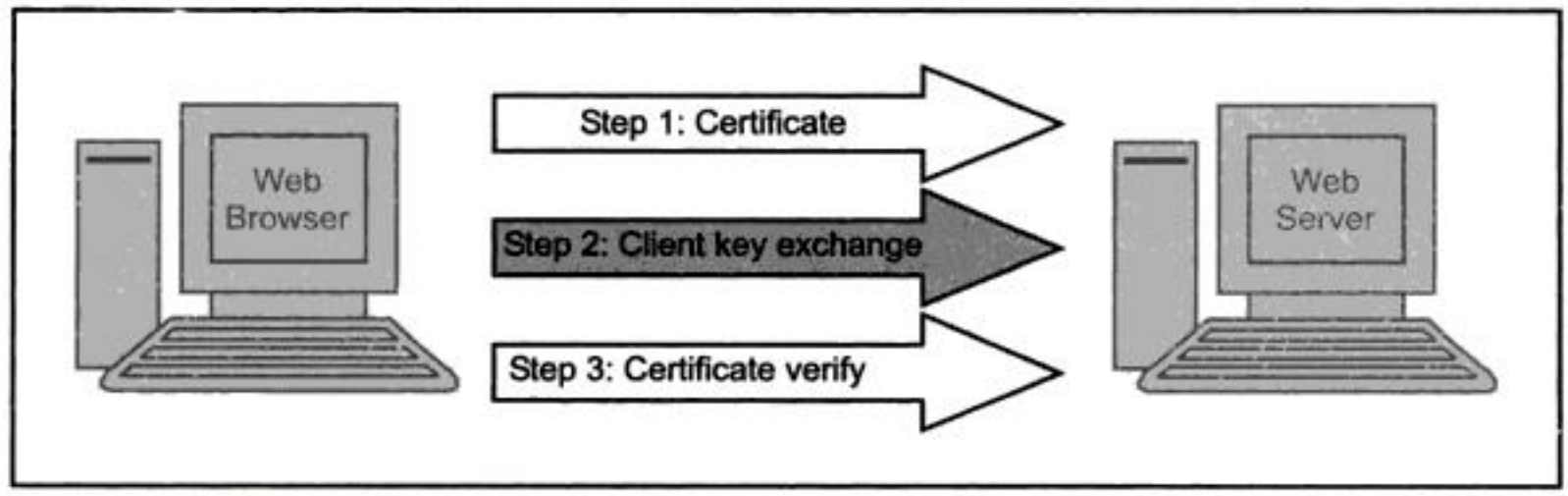


Fig. 6.14 *Phase 3 of SSL Handshake protocol: Client authentication and key exchange*

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 do 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* it 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

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Step 2: Time Stamping Request

Now, the client sends the message digest calculated in step 1 to the Time Stamp Authority (TSA) for getting it time stamped, as shown in Fig. 6.22. This is called as a Time Stamping Request.

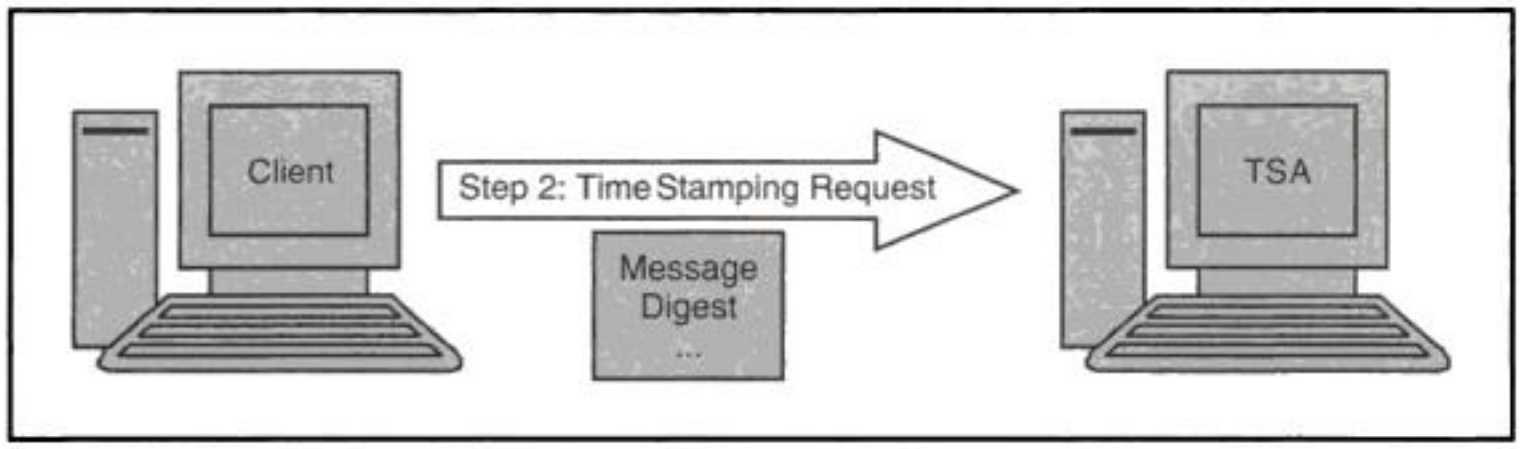


Fig. 6.22 *Time Stamping Request*

Step 3: Time Stamping Response

In response to the client's request, the TSA might decide to grant or reject the time stamp. If it decides to accept the request and process it, it signs the client's request together with the time stamp by the TSA private key. Regardless, it returns a Time Stamping Response back to the client. This is shown in Fig. 6.23.

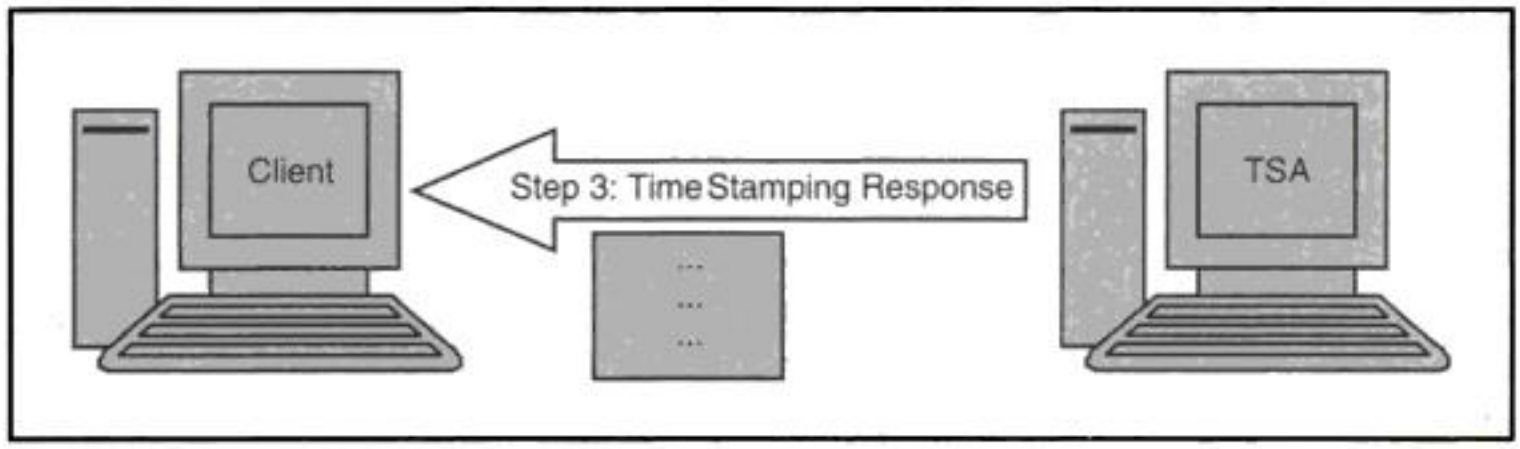


Fig. 6.23 *Time Stamping Response*

The PKIX model mandates a number of requirements on a TSA. The TSA must use a trustworthy time source. It must time stamp a message digest. It must not include any identification of the requesting entity (client) in the timestamp.

6.5 SECURE ELECTRONIC TRANSACTION (SET)

6.5.1 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.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

in Fig. 6.44. Of course, it uses the TCP/IP protocol underneath. That is, SMTP runs on top of TCP/IP (in the application layer).

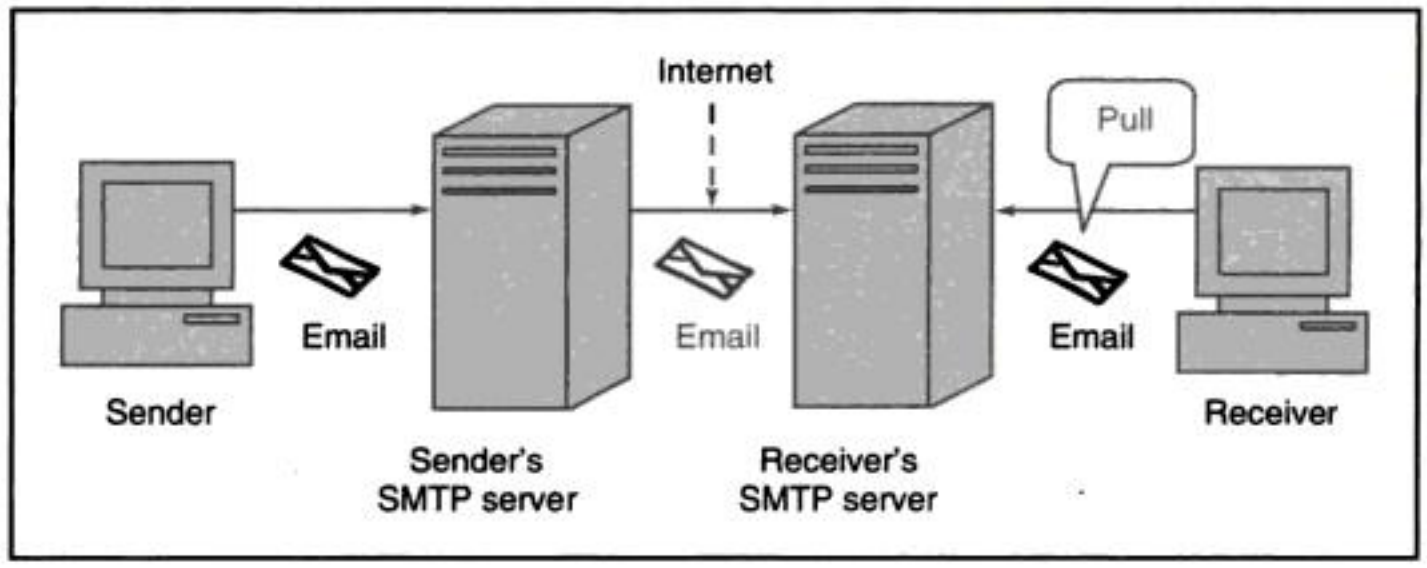


Fig. 6.44 *Email using the SMTP protocol*

The basic phases of an email communication consist 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.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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 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. 6.55.

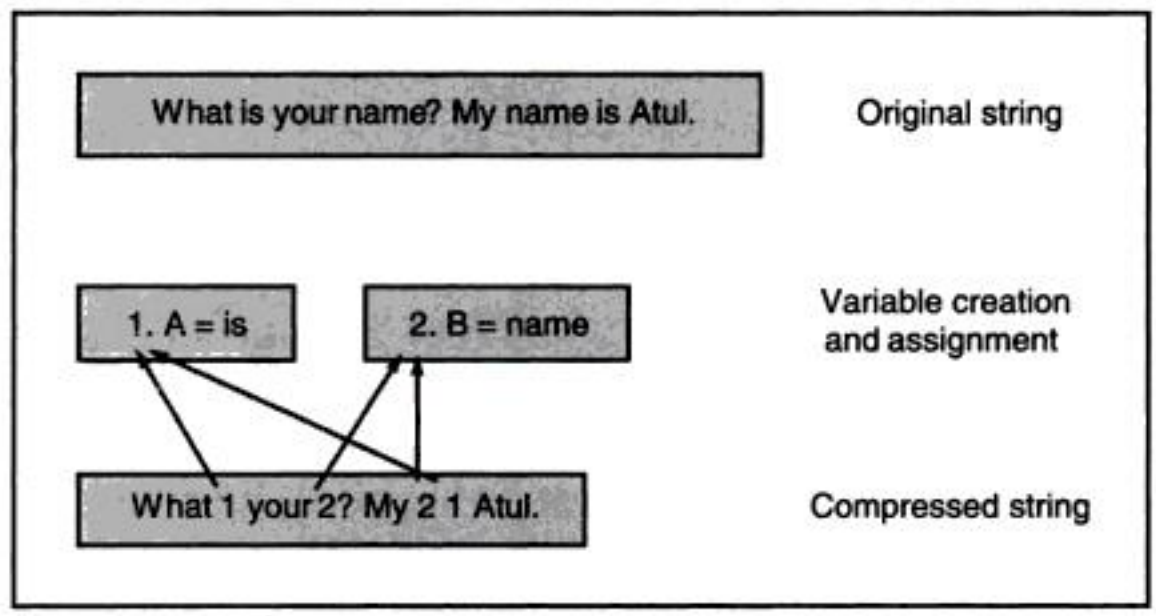


Fig. 6.55 *Lempel-Ziv algorithm, as used by the ZIP program*

As we can see, the resulting string *What 1your 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 outputs 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. 6.56.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Note 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 secretes. 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.

Note 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 Passwords

1. *How it works?*

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.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Application architecture

Let us now think about the architecture of our application. This involves thinking about the portions that are required on the client-side, and those required on the server-side. Clearly, the user would perform the cryptographic operations such as message digests, digital signatures and encryption on the client-side (i.e. on the browser machine). We must verify the results of these operations on the server-side, such as checking if a digital signature was correct, or to try and decrypt an encrypted message.

Client-side cryptographic processing: On the client-side (i.e. the browser side), we have decided to use MS-CAPI for cryptographic functionalities. In order to make use of the MS-CAPI services, pure HTML pages are not sufficient. HTML pages (i.e. a Web page written in the HTML language) can be used for displaying text on the browser in a desired format. However, it cannot perform any client-side processing. Therefore, we must have some other mechanisms to utilize the services of MS-CAPI on the client-side.

As we know, the most widely used approaches for client-side programming are: client-side scripting (e.g. JavaScript or VBScript), Java applets or browser plug-ins (ActiveX controls). Unfortunately, JavaScript cannot directly access MS-CAPI. Therefore, that option is ruled out. We must now decide between applets and plug-ins. Applets are safer to use. But the drawback with applets is that they are quite slow, both because they are written in Java, and also because they must be downloaded every time from the Web server to the browser when a cryptographic operation is to be performed. Consequently, we shall go for the approach of browser plug-ins.

A plug-in is downloaded only once—for the very first time it is referenced in a Web page. After that, it resides inside the browser and can be reused directly without requiring a fresh download. Plug-ins do create some psychological doubts in the mind of the users, such as the possibility of performing malicious operations on the user's computer (e.g. introducing a virus or overwriting the disk contents). However, to guard against such doubts, we shall digitally sign the plug-ins, to create confidence in the mind of the end users. When a signed plug-in is downloaded from the Web server to the browser, the user is shown a message that the plug-in is signed, and whether the user wants to trust the organization that has signed the plug-in.

Server-side cryptographic processing: Server-side cryptographic functionalities are pretty much straightforward. We shall provide a dedicated cryptographic API, which can perform the cryptographic operations. The business application of funds transfer would need to call the functions provided by our cryptographic API. The cryptographic API, in turn, will call the appropriate toolkit API.

Thus, the overall cryptographic operations can be summarized as shown in Fig. 10.2.

In order to understand how this works, we shall consider all the desired cryptographic functionalities one-by-one, in the following sections.

Message digest

As we know, the creation of message digest involves the usage of an algorithm such as MD5 or SHA-1. We will provide a method *create_digest ( )* in the plug-in, which, in turn, will call the message digest function provided by MS-CAPI. Let us understand how this works, step-by-step.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

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.

10.9 SECRET SPLITTING

Note Points for classroom discussions

1. *What is the need of secret splitting?*
2. *What would happen if we split a secret and one of the persons knowing the secret leaves the organization? Can the original secret still be recovered? Can the person leaving the organization break the secret before leaving?*
3. *Is the concept of secret splitting the same as XML signatures (signing only a portion of a document)?*

Many times, it is important to *split* a secret in such a fashion that the individual pieces of the secret make no sense. For example, suppose that in a bank, the account details are

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

As we can see, gcd (21, 45) = 3 from the above table.

```
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.1 *C language representation of the Euclid's algorithm*

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 mod 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, $a \equiv b \pmod{n}$ 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 (≡) 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 $a^x \equiv b \pmod{n}$. As an example, if $3^x \equiv 15 \pmod{17}$, then x = 6. For large numbers, solving this equation is quite difficult.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Cryptography and Network Security

Security is one of the most significant concerns of any organisation. This book clearly explains the concepts and practical issues behind Cryptography and Network Security.

The book has a lot of visual appeal, a large number of illustrations (412) are used to aid understanding. Each chapter is followed by multiple-choice questions, review questions and design/programming exercises.

Salient Features

- Practice of security using JAVA and Microsoft Toolkit technologies is demonstrated and practical implementation issues are discussed.
- Lucid explanation of technologies of Encryption, Decryption, Symmetric and Asymmetric Key Cryptography.
- Detailed analysis and explanation of all major Cryptographic algorithms, including Data Encryption Standard (DES), International Data Encryption Algorithm (IDEA), RC5, Blowfish, Advanced Encryption Standard (AES), Rivest Shamir Adleman Algorithm (RSA), Digital Signature Algorithm (DSA).
- Discussion on Organizational Security arrangements — Firewalls, Virtual Private Networks (VPN).
- Coverage of Digital Certificates, Digital Signatures, Public Key Infrastructure (PKI) and Extensible Markup Language (XML) Security.
- Focus on Internet Security with coverage of Secure Socket Layer (SSL), Secure Hyper Text Transfer Protocol (SHTTP), Time Stamping Protocol (TSP), Secure Electronic Transaction (SET), 3-D Secure, Pretty Good Privacy (PGP), Privacy-Enhanced Electronic Mail (PEM), Secure Multi-Purpose Internet Mail Extensions (S/MIME).
- Trends in Wireless Security: Wireless Application Protocol (WAP), Global System for Mobile Telecommunication (GSM), Third Generation Services (3G).
- Coverage of authentication mechanisms and Single Sign On (SSO) techniques.
- A selection of case studies highlighting the various security issues including Denial of Service (DOS) attacks, Voting over Internet and Online Banking transactions.

Atul Kahate is Project Manager, i-flex solutions, Pune. He has written a number of books on computers and cricket.
He is the co-author of *Web Technologies: TCP/IP to Internet Application Architectures*, published by Tata McGraw-Hill.

Tata McGraw-Hill

Visit us at: www.tatamcgrawhill.com
http://www.mhhe.com/kahate/cryptography

The McGraw-Hill Companies

ISBN 0-07-049483-5
9780070494831