Table of Contents

MOD 2
12
MOD 2
10

Reviewers' Comments

A comprehensive book covering all aspects of network security and cryptography. It is written in a style that the beginners in this area will find easy to understand.

Prof. S. V. Raghvan
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

The book is written in a lucid style, and covers all aspects of network security and cryptography well. It will be a good textbook for an advanced undergraduate elective.

Prof. Dheeraj Sanghi
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

The book will be a good reference for anyone who works with network security. It has a wide coverage of the subject and will be very useful as a textbook.

Dr. S. K. Ghosh
School of Information Technology
Indian Institute of Technology, Kharagpur

The book is excellent for the beginners. I can quote this book as "everything at one place" on Cryptography and Network Security.

Prof. B. B. Amberkar
Department of Computer Science and Engineering
NIT, Warangal

This book is really a comprehensive package for two different subjects, cryptographic foundations and network security put together in one. It also discusses a few real time applications on RFIDs, Web security services and electronic payments. The book will prove to be handy for researchers working in information security areas.

Prof. B. Majhi
Department of Computer Science and Engineering
NIT, Rourkela

I am fairly certain that those who read this book will be able to extend their knowledge base in Network Security domain. The book provides exhaustive opportunities to experiment and demonstrate intricate concepts in Cryptography with seamless transition. The book is well planned and specifically targeted at a wide array of audience ranging from beginners to advanced users.

Dr. H. S. Johal
Department of CS/IT
LPU, Phagwara

This book is perhaps the most comprehensive text on the subject, bringing together theoretical concepts, practical concerns, and examples in a field that continues to grow in importance. This book is highly recommended for both the student and the practioner who seek to endeavour in this great and fast moving field of network security.

Tony Antony, Data Center Group,
Cisco Systems Inc. Richardson, TX, USA

Chapter **1**

Introduction

Computer security is all about studying cyber *attacks* with a view to defending against them. We begin by investigating the principal *motives* behind such attacks. We then present a short survey of the attacks that have received the most press in recent times. These include pharming and phishing attacks together with assorted malware and denial of service attacks.

Understanding what makes systems vulnerable to these attacks is an important first step in avoiding or preventing them. We examine different classes of *vulnerabilities* including those caused by poorly written or configured software. We then sample diverse *defence strategies*. Access control, authentication, and data protection techniques are introduced. We distinguish between measures for *preventing* versus *detecting* intrusions.

In the last section, we highlight eight *guiding principles* of sound security practice. Although these are targeted at the security engineer, they also prove to be useful to other security professionals, system administrators, and IT managers.

1.1 CYBER ATTACKS

Here we discuss some common cyber attacks encountered.

1.1.1 Motives

Before we discuss the common attacks encountered, it is appropriate to ask, "What are the main goals of an attacker?" The sheer thrill of mounting a successful cyber attack has been motivation enough for *hackers* (Table 1.1). Most hackers were (and still are) young adults, often teens, who had dropped out of school but were otherwise intelligent and focused. Many of the "traditional" hackers seem to be obsessive programmers. They seem to be adept at circumventing limitations to achieve a challenging but often forbidden objective.

In addition, there is a tribe of *script kiddies* – juveniles who use scripts and attack kits designed by others (these can be freely downloaded from the Internet). Their activities do not require any special programming skills or advanced knowledge of computer systems.

Other perpetrators of cyber attacks include company insiders, often *disgruntled employees* who wish to even scores with their employers. There is also a serious threat from *cyber terrorists* who espouse extreme religious or political causes. Cyber terrorism is one weapon in their impressive arsenal which may include biological, chemical, and nuclear weapons. Their goals are to cripple the information/communication systems of the financial and business institutions of their "enemies."

In more recent times, the primary motivation for launching cyber attacks has shifted to financial gain. We next discuss some of the main motives of launching cyber attacks.

Table 1.1 *Notable cyber attacks*

Year	*Event*
1988	Robert Morris, Jr., a 23-year-old Cornell graduate student, released a worm that overran Arpanet, incapacitating almost 6000 computers, congesting government and university systems. He was fined $10,000 and sentenced to 3 years probation.
1991	31-year-old David L. Smith created the worm "Melissa," which infected thousands of computers causing damage of approximately $1.5 billion. This virus sent copies of itself to the first 50 names of the recipient's address book. He received a 20-month jail term.
2001	"Anna Kournikova" virus. Promising photos of the tennis star mailed itself to the every person in the victim's address book. Investigators were apprehensive that the virus was created with a toolkit enabling the rookies to create a virus.
2008	The headquarters of the Obama and McCain presidential campaigns were hacked.

Theft of sensitive information. Many organizations store and communicate sensitive information. Information on *new products* being designed or revenue sources can be hugely advantageous to a company's competitors. Likewise, details of military installations or precise *military plans* can be of immense value to a nation's adversaries. *Political espionage* targeted at government ministries and national intelligence can lay bare many sensitive operations planned for the future.

Besides corporations, banks, the military, intelligence, etc., the individual too has increasingly been a target. Leakage of personal information such as credit card numbers, passwords, and even personal spending habits are common and are collectively referred to as *identity theft*. Such information is advertised on certain websites and may be purchased for a small fee.

Disruption of service. Interruption or disruption of service is launched against an organization's servers so they are rendered *unavailable* or *inaccessible*. In recent times, there have been unconfirmed reports of such attacks being launched by business rivals of e-commerce websites. The goal here appears to be "my competitor's loss is my gain."

In 2001, there were a series of such attacks that targeted the websites of Yahoo, Microsoft, etc. in a short span of time. These appear to be headline-grabbing, but they could also have been "Proof of Concept" in nature. They were meant to alert corporates and others of the dangers of this class of attacks.

Illegal access to or use of resources. The goal here is to obtain free access or service to paid services. Examples of this include free access to online digital products such as magazine or journal articles, free talk time on someone else's account, free use of computing power on a supercomputer, etc. In each case, the attacker is able to *circumvent controls* that permit access to only paid subscribers of such services.

1.1.2 Common Attacks

The space of cyber attacks is large and expanding. In this section, we will introduce some of the high-profile attacks. The more subtle details of these and other attacks are dealt with throughout this text.

One set of attacks are those that attempt to retrieve personal information from an individual. There are various methods to accomplish this. Two of the best known attacks in this category are *phishing* and *pharming* attacks. The former lures its victims to a fake website – an on-line bank, for example. The fake site has the look and feel of the authentic bank with which the victim has an account. The victim is then induced to reveal sensitive information such as his/her login name and password, which are then passed on to the fake website.

Personal information may also be leaked out from credit cards, smart cards, and ATM cards through a variety of *skimming attacks*. There are a rich set of techniques to enable these attacks ranging from fake terminals to sophisticated *side channel attacks*. The latter attempt to deduce sensitive information from lost or stolen smart cards through advanced power and timing measurements conducted on them. Finally, leakage of information may also take place through *eavesdropping* or *snooping* on the link between two communicating parties.

One means of intruding into a computer system is through *password-guessing* attacks, which are a special case of *dictionary attacks*. Many of the attacks mentioned thus far are forms of *identity theft*. The ultimate goal of the attacker is to *impersonate* his/her victim. The attacker can then perform unauthorized logins (break-ins), make on-line purchases, initiate banking transactions, etc., all under the assumed identity of the victim.

The common name employed for an attacker's interruption or disruption of the computing services of the victim is *Denial of Service* (DoS). These attacks exhaust the computing power, memory capacity, or communication bandwidth of their targets so they are rendered unavailable. One version of this attack causes website defacement. At various times, the websites of high-profile targets such as the American president or various government ministries have been targeted. To prevent an alarm being raised, an advanced version of the DoS attack on a web server, for example, does not necessarily paralyze it. Instead, it slows down the web server so that its response time to requests from the outside world is unacceptably high.

An important class of attacks is that caused by various types of *malware*. *Worms* and *viruses* are malware that replicate themselves. A virus typically infects a file, so a virus spreads from one file to another. A worm is usually a stand-alone program that infects a computer, so a worm spreads from one computer to another. Worms and viruses use various spreading techniques and media – e-mail, Internet messages, web pages, Bluetooth, and MMS are some of the propagation vectors.

A *trojan* is a kind of malware that masquerades as a utility but has other insidious goals such as the modification of files, data theft, etc. *Spyware*, installed on a machine, can be used to monitor user activity and as a key logger to recover valuable information such as passwords from user keystrokes. Figure 1.1 summarizes some of the most common attacks.

1.1.3 Vulnerabilities

Behind every attack is a vulnerability of some type or the other. But what exactly is a vulnerability? A *vulnerability* is a *weakness* or lacuna in a procedure, protocol, hardware, or software within an organization that has the potential to cause damage. The understanding of security vulnerabilities is the key in helping us understand attacks better and, more importantly, in defending against them.

There are at least four important vulnerability classes in the domain of security:

(i) *Human Vulnerabilities:* These are vulnerabilities induced by human behaviour or action. For example, the user clicks on a link in an e-mail message received from a questionable source. By so doing, the user can be directed to a site controlled by the attacker as in a *phishing attack* or *a cross-site scripting attack*.

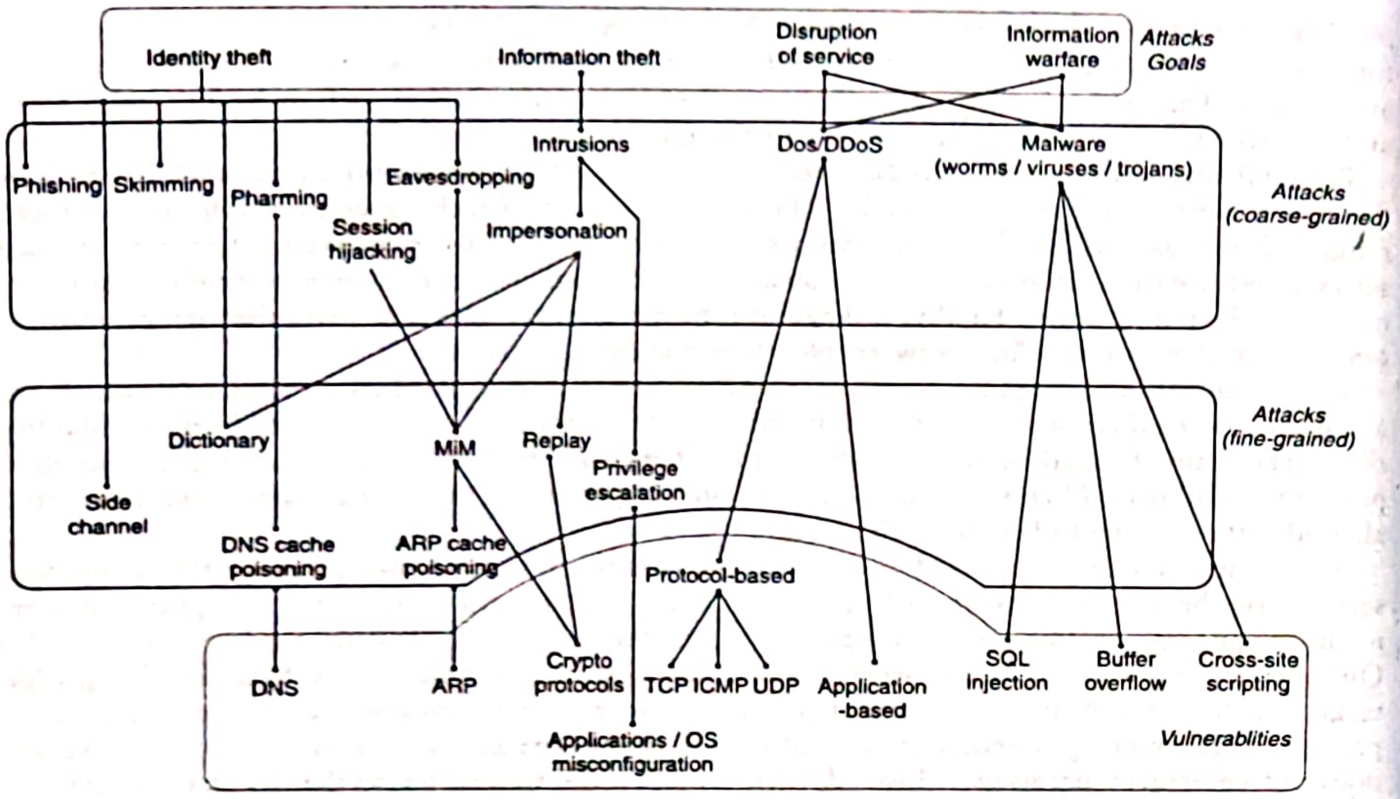


Figure 1.1 *Common attacks and vulnerabilities*

Similarly clicking on an e-mail attachment may open up a document causing a macro to be executed. The macro may be designed to infect other files on the system and/or spread the infected e-mail to other e-mail addresses harvested from the victim's inbox.

In both these cases, the human vulnerability consists of clicking on a link or attachment in an e-mail from a possibly unknown source. The link or attachment may have lured the victim by a flashy or tantalizing message suggesting quick money, erotica, etc., blinding him/her to the fact that the message came from an unknown source. It is actions like this that make a phishing attack or an e-mail virus so very successful.

(ii) *Protocol Vulnerabilities:* A number of networking protocols including TCP, IP, ARP, ICMP, UDP, DNS, and various protocols used in local area networks (LANs) have features that have been used in unanticipated ways to craft assorted attacks. *Pharming attacks* and various *hijacking attacks* are some examples. There are tools available on-line to facilitate some of these attacks. One such tool subverts the normal functioning of the ARP protocol to sniff passwords from a LAN.

There are a number of vulnerabilities in the design of *security protocols* that lead to *replay* or *man-in-the-middle* attacks. These attacks, in turn, lead to identity theft, compromise of secret keys, etc. Vulnerabilities in network protocols are often related to aspects of their design though they may also be the result of poor implementation or improper deployment.

(iii) *Software Vulnerabilities:* This family of vulnerabilities is caused by sloppily written system or application software. In many cases, the crux of the problem seems to be the code that is all too trusting of user input. Here are some examples:

- A program declares an array of 100 elements. It populates the array with input from the user. However, the program does not check the length of the user input string. If the latter is greater than 100 bytes, the buffer will overflow. There are hundreds of instances where this vulnerability, referred to as *buffer overflow*, has been exploited to inject and execute malicious worm code.
- A web server accepts input from a user's browser. The user is expected to type a simple text string containing his/her name or password. Instead, the user enters a part of one or more statements in the browser scripting language, Javascript. The server software does not perform sufficient *validation of user input* received by it. This is an example of a *cross-site scripting vulnerability* and may be exploited to devise an attack by the same name.
- A server accepts input from a user's browser or through some other interface. Once again, a simple text string such as a user name or password is expected. Instead, the user types part of an SQL query. Structured Query Language or SQL is popularly used to query, modify, insert, or delete information in a relational database. Again, the server does not validate the user input. Consequently, the database receives a query which is semantically quite different from what it normally receives. The insufficient validation of user input by the server results in an *SQL Injection vulnerability*. This vulnerability has been exploited, for example, to steal customer credit card information residing on database servers.

(iv) *Configuration Vulnerabilities:* These relate to configuration settings on newly installed applications, files, etc. Read-write-execute permissions on files may be too generous and susceptible to abuse. The privilege level assigned to a process may be higher than what it should be to carry out a task. This privilege may be misused during some point in its execution leading to what are commonly called "privilege escalation" attacks. Besides misconfiguration of software and services, security appliances such as firewalls may be incorrectly or incompletely configured with possibly devastating effect.

Vulnerability identification is a necessary first step in designing attacks. The term, "exploit," is often used as a noun in the security literature. An *exploit* is an instance of a particular attack on a computer system that leverages a specific vulnerability or set of vulnerabilities. The next step beyond vulnerability identification is *plugging* them or creating *patches* so that they can no longer be exploited. We next turn out attention to defence strategies and counter measures.

1.2 DEFENCE STRATEGIES AND TECHNIQUES

1.2.1 Access Control—Authentication and Authorization

The first defence strategy to prevent intrusions is *access control*. This implies the existence of a *trusted third party* that mediates access to a protected system. The trusted third party is typically implemented in software and may be a part of the operating system and/or the application.

The first step in access control is to permit or deny entry into the system. This involves some form of *authentication* – a process whereby the *subject* or *principal* (the party attempting to login) establishes that it is indeed the entity it claims to be. One form of authentication is the humble password. The principal first enters his/her login name. By prompting him/her to enter his/her password, the system implicitly challenges the principal to prove his/her identity. In this simple case, knowledge of the secret password constitutes "proof of identity."

After successful authentication, a subject is logged into the system. The subject may need to access several resources such as files. It is the job of the *access controller* to answer *authorization*-related questions such as

"Is *Rajeev* allowed to *write* into file, *CS649Grades* ?"

Note that there are at least three parameters to such an access control decision:

- the subject or principal, *Rajeev*,
- the object or resource, *CS649Grades*, and
- the access mode or operation, *write*.

To further highlight the difference between the important concepts of authentication and authorization, consider the example of a university library. The rule for getting the entry into the library is

— Any member of the faculty, staff, or student can enter the library by producing a valid university ID card.

The ID card is the *mechanism for authentication*. In this case, it is a contactless smart card, which is read electronically by an overhead card reader. It activates a turnstile allowing the card owner entry.

The following rules pertain to reading and checking out of library books and periodicals once a user is permitted to enter the library premises:

— Any member of the university community is permitted to read any book or access any computer while in the library.

— Only faculty members and students can check out books.

— Only the instructor, TAs, and students enrolled in a particular course may check out books on reserve for that particular course for a 3-hour period.

— Only faculty and graduate students are permitted to check out journals for a period of 1 day.

The problem of permitting library entry is analogous to authentication, while that of permitting book or journal check-out is analogous to authorization. Note that authentication is more coarse-grained and it appears first. The authorization decisions made in response to subject requests for various resources are fine-grained and only made after authentication has been successfully completed.

An important application of access control is to network traffic from the external insecure Internet into the protected environment of an organization. Clear-cut rules governing the entry of all such traffic must be formulated. A device called a *firewall* sits at the perimeter of the organization and it is configured to filter packets based on the source/destination addresses and port numbers.

1.2.2 Data Protection data

The data in transit or in storage need to be protected. But what does protection imply? In many contexts, it implies *data confidentiality* – the data should not be readable by an intruder. Another dimension to data protection is the preservation of *data integrity*. This implies that the data in transit should not be tampered with or modified – either inadvertently or by malicious design – without being detected.

Cryptographic techniques are among the best known ways to protect both, the confidentiality and integrity of data. Cryptography is the science of disguising data and is the subject of the first part of this book. The *encryption* operation is performed by the sender on a message to disguise it prior to sending it, while a *decryption* operation is performed on the disguised message in order to recover the original message. Similarly, there are a number of *integrity check* techniques that may be used – the most secure of these is the *cryptographic checksum*.

In the most basic case, the encryption and decryption operations both use the same *secret key* known only to the sender and receiver. This prevents an eavesdropper from decrypting the encrypted message. Likewise, the computation of the cryptographic checksum uses a secret shared by the sender and receiver. The sender computes the checksum as a "one-way function" of the message and secret. It transmits the message and checksum. The receiver also computes the checksum. If the computed checksum matches that received, the receiver concludes that there is no error in the received message.

1.2.3 Prevention and Detection

Access control and message encryption are preventive strategies. Authentication keeps intruders out, while authorization limits what can be done by those who have been allowed in. Encryption prevents intruders from eavesdropping on messages. The cryptographic checksum, on the other hand, detects tampering of messages.

In the important domain of software security, code testing is used to detect vulnerabilities. *Blackbox testing* is employed when the source code of a program is not easily available. In this case, carefully crafted inputs are fed to a running program. The response of the running program to these inputs is carefully observed. For example, if a program expects a person's name as input, a string of 300 characters is the input. The goal here is to determine whether the software has been carefully designed to handle unexpected or malicious input. For greater assurance of secure software, *whitebox testing* should be employed. Here, the security engineer has access to source code and can perform more elaborate testing by exercising different control paths in the source code.

If we have intrusion preventive techniques in place, why do we need intrusion detection at all? The fact of the matter is that intrusion prevention may not always be practical or affordable. Also, where deployed, it may not always be effective. For example, it may not always be easy to keep out DoS traffic since it is often difficult to distinguish between legitimate traffic and attack packets. In such cases, we look for *anomalous behaviour* – radical departure from the norm. Continuous *monitoring* of network logs and operating system logs are a good starting point.

Intrusion detection systems also look for certain *patterns of behaviour*. For example, multiple instances of a given worm often exhibit a characteristic bit pattern called a *worm signature*. Many anti-virus products are signature-based. Other give-aways of certain malware are a peculiar sequence of system calls made or patterns of file access.

1.2.4 Response, Recovery, and Forensics

Once an attack or infection has been detected, *response measures* should be swiftly taken. These include shutting down all or part of the system. During a worm epidemic, the infected part of the system should be *quarantined* and necessary *patches* applied. Many intrusion attempts leave fingerprints just like a criminal does at the crime site. *Cyber forensics* is an emerging discipline with a set of tools that help trace back the perpetrators of cyber crime.

Table 1.2 defines some of the most widely used terms in cyber security parlance.

1.3 GUIDING PRINCIPLES

We enunciate several ideas which taken together constitute some of the folk principles of modern security practice. This list of principles is in no way exhaustive, but it does deserve serious consideration.

1. Security is as much (or more) a human problem than a technological problem and must be addressed at different levels.

In many discussions on security, the triad of *people, processes*, and *technologies* is often highlighted. At the highest level, security should be addressed by top-level management in large organizations. Robust *security policies* should be formulated and a comprehensive implementation strategy outlined by a dedicated team of security specialists, possibly headed by a Chief Information Security Officer (CISO).

Some of the *mechanisms* used to implement high-level policies are in the realm of technology. Security engineers have a key role to play in designing techniques and products to protect organizations from the various cyber attacks studied in the previous section. The thrust of this book is on technological solutions to security, but it should be borne in mind that security is more of a human problem than a technological one.

System administrators handle day-to-day operations. They should be proactive in crucial security practices such as *patch application*. One of the key tasks of a system administrator is to *configure systems* and *applications*. Their job also involves setting user/group permissions to various system resources such as files, configuring firewalls, sifting through system logs for signs of an intrusion, and processing alerts.

Table 1.2 *Definitions of commonly used terms in security parlance*

- *Security policy* is the set of rules and practices that regulate how an organization manages and protects its computing and communication resources from unauthorized use or misuse.
- *A security mechanism* is a technique or device used to implement a security policy.
- A *vulnerability* is a weakness or flaw in the architecture, implementation, or operational procedures of a system that could be exploited to cause loss or failure.
- Exploitation of a vulnerability with malicious intent leads to a cyber *attack*.
- *Access control* is the process of preventing unauthorized access to a computing or communication resource.
- *Authorization* involves granting a specific entity or process the permission to access restricted data or perform a restricted operation.
- *Auditing* is the process of collecting and analyzing relevant information in order to ensure compliance with security policies laid out for an organization.

One or more of the following are implicit when we talk about a secure connection or session between two parties:

- *Entity authentication* is the process of verifying that the entity being communicated with is indeed the entity it claims to be.
- *Message authentication* is the process of verifying the source or origin of the message.
- *Confidentiality* is the protection of data from disclosure to an unauthorized party or process.
- *Integrity* is the assurance that data has not been modified, tampered with, or made inconsistent in any way.
- *Non-repudiation* offers a guarantee against repudiation or denial by a party of the fact that it created or sent a particular message.

The final link in the security chain is the rank and file within an organization. The employees within an organization should be educated on various do's and don'ts through periodically updated security *awareness programs*. In summary, a healthy combination of enlightened security policy and procedures, backed by enforcement, aided by technology, coupled with diligent participation of administrators and employees, and presided over by an empowered CISO is the surest insurance against cyber attacks.

2. Security should be factored in at inception, not as an afterthought.

This might sound obvious but scores of designers have paid scant heed to such an advice. For example, security was never on the horizon for *protocol designers* several decades ago. This is hardly surprising. Functionality, correctness, performance, and reliability hogged the attention of designers then. No one then had anticipated that those protocols would be abused by attackers in so many creative ways!

Less forgiving is the fact that *application software* (web software, for example) developed today continues to be often vulnerable to numerous attacks such as cross-site scripting and SQL injection attacks. The solution lies, at least in part, in integrating *secure coding practices* into the software curriculum in our colleges and universities.

In general, security should be factored in early on during the design phase of a new product and then carried forward right through implementation and testing. The product could be a networking protocol, a new version of an operating system, a piece of application software, or the architectural layout of computing infrastructure for an enterprise.

3. Security by obscurity (or by complexity) is often bogus.

There have been a number of cryptographic algorithms proposed by a small set of people. Their use was mandated in newly standardized protocols, but their details were not made public. History is replete with case studies of many such algorithms that have had serious vulnerabilities. The flaws are exposed over time after the protocols have been widely deployed, attracting closer attention from the hacker community. Note that hacking, per se, should not be always portrayed in a negative light for there is a vibrant community of *ethical hackers* whose goal is to break software/protocols/algorithms so that they can be fixed before things get out of hand.

It is the ethical hacker community at least, if not the public at large, who should be able to study new protocols and algorithms prior to widespread adoption. One such example was the procedure followed for selecting an algorithm in the late 1990s for the new secret key cryptography standard – AES was finally chosen after much public scrutiny and debate. (Details of AES are covered in Chapter 9.) As another example, *open source software* is usually freely available. Public review of its security features can make or break its reputation.

4. Always consider the "Default Deny" policy for adoption in access control.

The subjects in an access control policy could be people, packets, or even user input. One policy is the "Default Permit," i.e., grant the subject's request unless the subject is on a *blacklist* or it has certain blacklisted attributes. The dual of this policy is the "Default Deny" policy. In this case, the subject's request is denied unless it is on a *whitelist*.

Clearly, whitelisting is the more conservative approach. With whitelisting, the access controller may reject a legitimate subject whose name has been mistakenly excluded from the whitelist but that is the price to be paid for greater security. Blacklisting, on the other hand, may accept a bad guy because his name or attributes were mistakenly excluded from the blacklist. The *tradeoffs* between blacklisting and whitelisting should be carefully examined (see Principle 8). However, in general, prudent security design should seriously consider adoption of the "Default Deny" policy.

5. An entity should be given the least amount/level of permissions/privileges to accomplish a given task.

Role-based access control (RBAC) has influenced a variety of software platforms ranging from operating systems to database management systems. The principal idea in RBAC is that the *mapping between roles and permissions* is paramount. The role played by an individual at a given point in time determines the rights or privileges the individual has. Conferring higher privilege to an individual than what is warranted by his/her current role could compromise the system. It is like handing over the keys to your home safe to the person who has just arrived to shampoo your home carpet! Is the right to open your safe necessary to get the shampooing job done?

Privilege escalation in its different manifestations has caused many security breaches in computer systems. The problem often lies in sloppy or incomplete *configuration management*. In publicly accessible servers within an organization such as the web and e-mail servers, unnecessary services hosted by them can open the door to malware, which can compromise those servers. The latter are then used as a springboard to spread to the internal machines in that organization.

6. Use 'Defence in depth' to enhance security of an architectural design.

This principle is used in many high-security installations and has been recently introduced in some airports. A passenger's ticket is checked before entering the airport terminal building. This is followed by verification of travel documents and inspection of check-in baggage at the airline counter. Next comes a security check (physical) and a further check of the boarding pass, travel documents, and check-in baggage before entering the boarding area (main concourse).

Defence in depth is applicable to cyber security as well. Consider designing the *firewall architecture* for a mid-to-large size enterprise. Every packet from the outside (Internet) should be intercepted by at least two firewalls. The firewalls may be from two different vendors and would, preferably, have been configured by two different system administrators. They may, and typically do, have some overlapping functionality. Because of differences in the hardware/software design and in configuration, what escapes Firewall 1 may be caught by Firewall 2 and vice versa.

7. Identify vulnerabilities and respond appropriately.

We have already seen a large number of vulnerability types. Vulnerabilities in software or protocols are well researched. But equally important are lacunae/shortcomings in policy, procedures, and operations. How many organizations are geared to implement policies regarding the entry of visitors' laptops and PDAs? Or do they even have such policies in place? Such mobile devices and Bluetooth-enabled gadgets may transmit malware to unsuspecting stations within the organization. Likewise, USB-enabled PCs may be victims of viruses residing on USB flash drives. Often, these organizations have elaborate security infrastructure in the form of firewalls and intrusion detection systems. They securely guard the high-profile main entrance but blissfully ignore the security requirements of the less conspicuous side and rear doors.

Vulnerability detection and response brings to mind fast-spreading Internet scanning worms. An excellent example is the Code Red worm unleashed in July, 2001. Within a few weeks, a patch was created and released for the vulnerable IIS server. Sadly, many servers remained unpatched, aiding the rapid spread of the worm over a 24-hour period.

Is the effort involved in identifying and plugging vulnerabilities justified? Security analysts are often accused of being paranoid. The other side of vulnerability identification is *risk assessment*. Formally,

$$\textit{Risk} = \textit{Assets} \times \textit{Vulnerabilities} \times \textit{Threat}$$

If the assets impacted by a vulnerability are of low value and/or the threat perception (probability that a vulnerability is successfully exploited) is small, then the associated risk is low. In such cases, it may not make economic sense to address such vulnerabilities.

8. Carefully study the tradeoffs involving security before making any.

Engineering design often involves making tradeoffs – cost versus performance, functionality versus chip area, etc. The previous principle highlighted an important tradeoff – *security versus cost.* We elaborate on this further.

Consider, for example, the area of electronic payment involving small purchases (say Rs. 10 or less). Such payments, called micropayments, may be made for digital goods such as on-line newspaper articles. Payment schemes use some form of cryptography. The cryptographic overheads of these schemes, in terms of computation cost, can be high. Can we use cheaper (lower overhead) cryptography for micropayments? The downside here is that such cryptography is not as secure. But given the transaction amount, the risk of fraud is probably acceptable. In this case, we may be justified in trading off increased security for lower cost.

Besides security versus cost, *security versus performance* is a tradeoff often encountered. Almost all techniques studied in this book to prevent or detect various cyber attacks involve extra computations, which incur not merely a financial cost but also a performance penalty.

Thanks to heightened fears of terrorism in the wake of 9/11, there has been increased frisking and scanning at airports. Many airports require passengers to check in several hours before flight departure. *Human convenience* is being sacrificed at the altar of enhanced flight security. This has some parallels with cyber security. Think of the minor inconveniences you may have to put up with for greater security.

- The system will not allow you to log in today unless you change your password. (The system expects you to do so at least once a month.)
- The system will not accept your new password since it is less than 8 characters long and/or it does not contain any non-alphanumeric character.
- You try to download a utility program from a certain website on to your smartphone, but the newly installed version of the operating system on your smartphone forbids downloads of files not digitally signed.

Making tradeoffs is not always easy. The marketing folks in a major telecom company might wish customers had all features enabled by default on their latest smartphone to be launched, but the security folks may have completely different ideas! The security team understands that many of the jazzy features could easily open the door to trojans and other malware. In summary, security versus cost, security versus performance, and *security versus convenience/flexibility* are among the many tradeoffs a security engineer may have to grapple with.

SELECTED REFERENCES

www.sans.org and www.cert.org/ maintain up-to-date information of various vulnerabilities and cyber attacks. www.mitre.org/work/cybersecurity.html also contains useful material on this subject. All three sites have a number of very useful white papers related to cyber defence. The official website of the Indian Computer Emergency Team is first.org/members/teams/cert-in/.

There have been many texts on the subject of network security in recent times. Some of the more widely used are [STAL10], [GOLL06], and [STAM06].

OBJECTIVE-TYPE QUESTIONS

1.1. The motivation of an ethical hacker is
(a) financial gain
(b) the thrill of hacking
(c) the desire to identify vulnerabilities so they can be patched before they are publicly exposed
(d) a religious/political/ideological cause

1.2. Which of the following attacks is/are likely to result in identity theft?
(a) Phishing attack
(b) Denial of service attack
(c) Dictionary attack
(d) Virus infection

1.3. The buffer overflow attack is caused by
(a) a vulnerability in the design of a networking protocol
(b) a vulnerability in the implementation of a networking protocol
(c) a vulnerability in human behaviour
(d) a vulnerability in software

1.4. A counter-measure to eavesdropping on the communication link is the use of
(a) a cryptographic checksum
(b) encryption
(c) a login name and password
(d) a fake identity

1.5. The following is used when the source code for a piece of software is unavailable:
(a) blackbox testing
(b) whitebox testing
(c) regression testing
(d) unit testing

1.6. A good example of defence in depth is
(a) the use of exhaustive software testing
(b) the detection of worm and virus signatures in incoming packets
(c) the use of encryption in conjunction with a cryptographic checksum
(d) the use of multiple firewalls in an organization

1.7. The choice of whether to use low-overhead cryptography versus heavy-duty cryptography in a micropayment scheme represents a tradeoff between
(a) security and cost
(b) cost and performance
(c) security and convenience
(d) security and performance

EXERCISES

1.1. In the conventional military sense, it is occasionally suggested that "a strong offense is the best defence." Does this make sense in the context of cyber security and cyber defence? Explain your answer.

1.2. Some of the counter-measures against cyber attacks are proactive and some are reactive. Identify three examples of proactive measures and three examples of reactive measures. When are proactive measures more appropriate and when are reactive measures more appropriate?
1.3. Propose any two counter-measures against phishing attacks. Compare them in respect to effectiveness, user acceptance, practicality, and any other relevant metric.
1.4. The term *data integrity* may mean quite different things depending on the context in which it is used. What is the difference between the integrity of data in communication versus the integrity of data stored in a relational database?
1.5. What is the difference between reliable software and secure software? How would the testing methodology and testing details be different if software were tested for security versus being testing for reliability?
1.6. Are there situations where the "Default Accept" or "Default Permit" policy is more appropriate than the "Default Deny" policy for access control? Explain.
1.7. You are asked to design a "Computer Security Awareness Program" for general employees in a telecom company. What would the different modules of such a course be? What would the contents and duration of each module be? Assume that the targeted audience use computers, e-mail, etc. in their professional day-to-day activities. Would your course change if you were to target employees of a bank instead? If so, how and why?

ANSWERS TO OBJECTIVE-TYPE QUESTIONS

1.1 (c)	**1.2** (a)(c)	**1.3** (d)	**1.4** (b)
1.5 (a)	**1.6** (d)	**1.7** (a)(d)	

Chapter 3

Mathematical Background for Cryptography

3.1 MODULO ARITHMETIC

Let d be an integer and let n be a positive integer. Let q and r be the quotient and remainder obtained from dividing d by n. The relationship between d, n, q, and r is

$$d = n * q + r, \quad 0 \le r < n \tag{3.1}$$

Note that r is a non-negative integer less than n. d and n are the dividend and the divisor, respectively. We say "d is equal to r modulo n" if the remainder obtained from dividing d by n is r. This is expressed as

$$r \equiv d \ (mod \ \mathrm{n}) \tag{3.2}$$

For a given value of n and r, there are an infinite number of (d, q) pairs that satisfy Eq. 3.1.

Example 3.1

Let $n = 10$ and $r = 3$.

Then 13, 23, 33, etc. all satisfy Eq. (3.1) with quotients 1, 2, 3, etc. In fact, each element of the set below satisfies Eq. (3.2).

$$\{. . . -37, -27, -17, -7, 3, 13, 23, 33, 43, . . .\}$$

Any two numbers in the above set are said to be congruent modulo 10 and the set itself is referred to as a *congruence class*. It is helpful to visualize the "modulo n relationship" using Fig. 3.1. The integers are laid out along a spiral with n integers on a single "circle". Starting with 0, we encounter the positive integers in sequence as we traverse the spiral clockwise, while the negative numbers are encountered as we traverse the spiral in the anti-clockwise direction. The set of elements along a given radius constitute one of the congruence classes modulo n. There are n congruence classes mod n. It is convenient to represent a class by the smallest non-negative integer in that class.

Two distinct integers, a and b, that are congruent modulo n map to the same radius in the spiral. Counting from a to b involves one or more revolutions. It follows that:

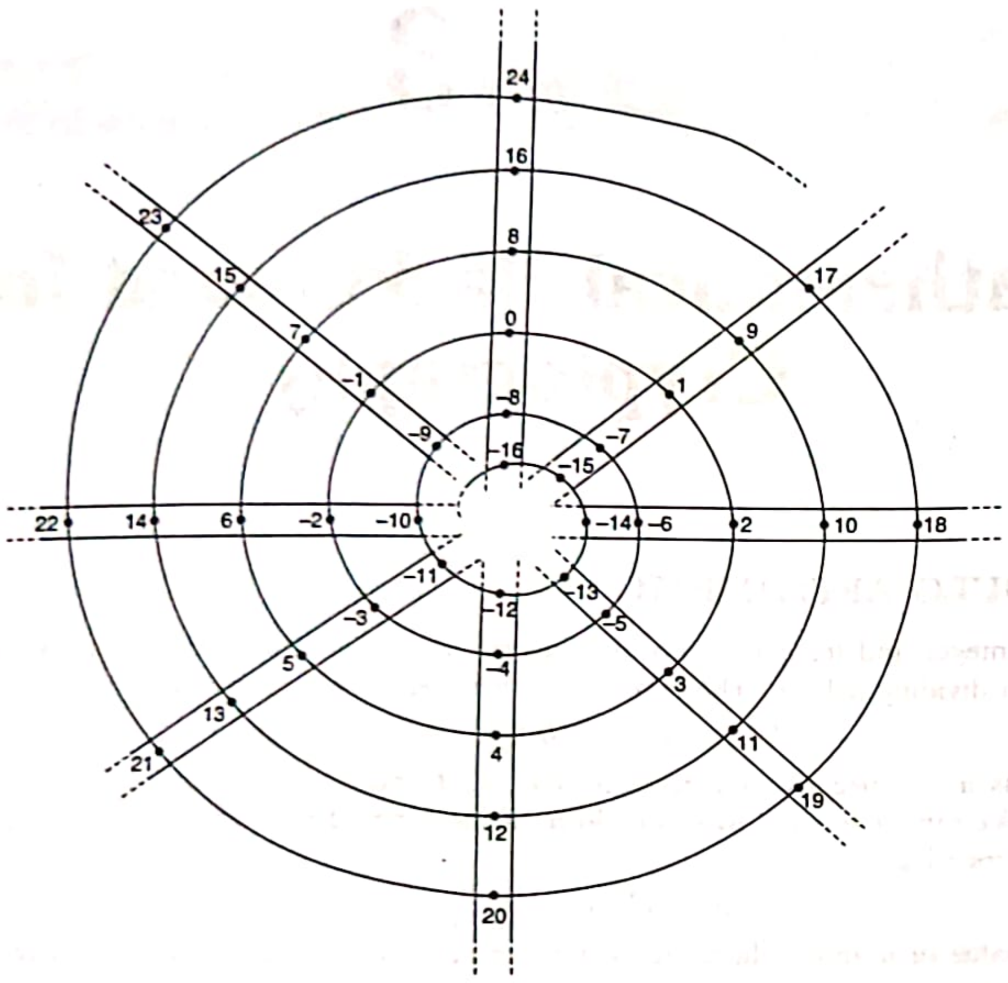


Figure 3.1 *Equivalence (congruence) classes modulo 8*

Fact: If two integers are *congruent modulo n*, then they differ by an integral multiple of n. Algebraically, if

$$a \bmod n = r \quad \text{and} \quad b \bmod n = r,$$

then

$$a = n^*q_1 + r \quad \text{and}$$
$$b = n^*q_2 + r$$

where q_1 and q_2 are integers.
Subtracting, we get

$$a - b = n\,(q_1 - q_2)$$

Since q_1 and q_2 are integers, a and b differ by an integral multiple of n.

Many useful *properties of modulo arithmetic* can be proved using the above fact.

1. $(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$
2. $(a - b) \bmod n = ((a \bmod n) - (b \bmod n)) \bmod n$
3. $(a * b) \bmod n = ((a \bmod n) * (b \bmod n)) \bmod n$

The proof for the three properties is similar. We prove Property 3 here.

Let

$$a = n^*q_1 + r_1 \quad \text{and} \quad b = n^*q_2 + r_2.$$

So

$$a^*b = n(n^*q_1{}^*q_2 + q_1{}^*\ r_2 + q_2{}^*\ r_1) + r_1{}^*\ r_2$$

We can add/subtract any multiple of n from a number without changing its value modulo n. So, the LHS of Property 3 is

$$(a\ ^*\ b) \bmod n = (r_1{}^*\ r_2) \bmod n$$

Substituting for a and b, the RHS of Property 3 is

$$\begin{aligned}&(((n^*q_1 + r_1) \bmod n)\ ^*\ ((n^*q_2 + r_2) \bmod n)) \bmod n\\ &= ((r_1 \bmod n)\ ^*\ (r_2 \bmod n)) \bmod n\\ &= (r_1{}^*\ r_2) \bmod n\end{aligned}$$

Example 3.2

We verify Property 1 for $n = 8$, $a = 27$, and $b = 34$.

The LHS of Property 1 is

$$\begin{aligned}&(27 + 34) \bmod 8\\ &= 61 \bmod 8\\ &= 5\end{aligned}$$

The RHS of Property 1 is

$$\begin{aligned}&((27 \bmod 8) + (34 \bmod 8)) \bmod 8\\ &= (3 + 2) \bmod 8\\ &= 5\end{aligned}$$

Where do we use these properties? In cryptography, we often have to perform computations such as multiplying a large number of terms, each term itself being a very large number. For example, we may have to multiply 50 integers, each about 1000 digits long.

$$(a_1\ ^*\ a_2{}^*\ a_3 \ldots\ ^*\ a_{50}) \bmod n$$

In the worst case, the size of $a_1\ ^*\ a_2$ will be 2000 digits, the size of $a_1\ ^*\ a_2{}^*\ a_3$ will be 3000 digits. Property 3, however, tells us that we could "reduce modulo n" each intermediate product before multiplying by the next term. For example, we could

compute the product	$a_1\ ^*\ a_2$
reduce, i.e., compute	$b = (a_1\ ^*\ a_2) \bmod n$
compute the product	$b\ ^*\ a_3$
reduce, i.e., compute	$(b\ ^*\ a_3) \bmod n$

Notice, that by so doing, we are restricting the size of each intermediate result. In particular, if n is roughly 1000 digits, then the length of the intermediate results after a multiplication and a reduction is no more than 2000 and 1000 digits, respectively.

3.2 THE GREATEST COMMON DIVISOR

For simplicity, we deal with non-negative numbers only in this sub-section. Given two integers, a and b, we say that a divides b, denoted $a|b$, if there exists an integer $x \geq 1$ such that $a * x = b$. a is said to be a *divisor* of b.

Definition: If $a|b$ and $a|c$, and there exists no $a' > a$ such that $a'|b$ and $a'|c$, then a is referred to as the *greatest common divisor* of b and c, denoted $a = gcd(b, c)$.

Example 3.3

2, 3, and 6 are each common divisors of both 24 and 78. The largest integer that divides both is 6, so $gcd(24, 78) = 6$.

Definition: If $gcd(b, c) = 1$, we say that b and c are *relatively prime* or *co-prime*.

Definition: An integer is prime if it is co-prime with all positive integers less than it.

Example 3.4

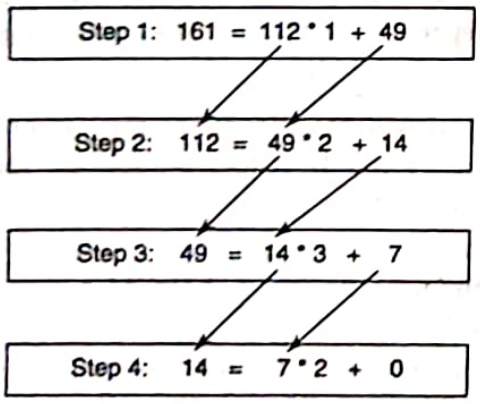
The integers 14 and 9 are co-prime but neither is a prime number.

3.2.1 Euclid's Algorithm

Euclid's algorithm is used to find the gcd of two integers, b and c. Without loss of generality let $b > c$. The first step in the algorithm is to divide b by c explicitly showing the quotient, q, and remainder, r.

$$b = c^* q + r$$

In each subsequent step, a similar equation is written in which the new dividend (leftmost number) and new divisor (to the right of the equal sign) are respectively the divisor and remainder from the previous step. We illustrate this with an example that computes gcd(161, 112).



1) 49, 14, 7 divided by gcd(b,c)

2) Above '0' is the gcd(b,c)

We note that the sequence of remainders (the rightmost numbers) keeps decreasing. We have defined a remainder to be a non-negative integer. So, the sequence is finite. But must it end with 0? If it did not, we could continue the process of division. So, the sequence of divisions continues until a remainder of 0 is encountered.

The following are two key observations about the above procedure.

(a) *gcd*(b, c) divides each non-zero remainder above.

Consider dividing each term of the equation in Step 1 by *gcd*(b, c). Since *gcd*(b, c)|b and *gcd*(b, c)|c, we get integer quotients in the first two terms. If *gcd*(b, c) does not divide the remainder, r in Step 1, the result of the division of the last term will have a fractional component. This is a contradiction since it implies that an integer is equal to an integer plus a fractional part. This argument can be repeated for Steps 2, 3, etc., in turn. This proves Statement (a).

(b) The remainder just above the zero in the procedure described above is *gcd*(b, c).

Let $g = gcd(b, c)$. There are only two possibilities – either the remainder = g or it is a multiple of g, say k_1*g, where $k_1 > 1$. Also, the divisor in the penultimate step is a multiple of g, say k_2*g. The last step then becomes

$$k_2*g = k_1*g * q', \text{ where } q' \text{ is the quotient in the last step.}$$

So

$$k_2 = k_1 * q'$$

Now $gcd(k_1, k_2) = 1$. If not, since $gcd(k_1, k_2)*g$ divides both the remainder and divisor in the penultimate step and, by extension, all previous remainders and divisors, it follows that $gcd(b, c) > g$ – a contradiction.

Since $\quad k_2 = k_1 * q', \quad gcd(k_1, k_2) = k_1$

Since $\quad gcd(k_1, k_2) = 1$ and $gcd(k_1, k_2) = k_1, \quad k_1 = 1$.

We conclude that the remainder in the penultimate step is indeed *gcd*(b, c).

This sequence of steps to compute the gcd of two integers is called *Euclid's algorithm*. The procedure naturally leads to the following conclusion:

GCD Theorem: Given two integers b and c, there exist two integers x and y such that

$$b * x + c * y = gcd(b, c)$$

We demonstrate this by an example.

Example 3.5

$$\text{Let } b = 161 \quad \text{and} \quad c = 112.$$

From Step 3 of the previous example,

$$7 = 49 - 14 * 3$$

Substituting for 14 from Step 2, we get

$$7 = 49 - (112 - 49 * 2) * 3$$

or

$$7 = 49 * 7 + 112 * (-3)$$

Substituting for 49 from Step 1, we get

$$7 = (161 - 112 * 1) * 7 + 112 * (-3)$$

or

$$161 * 7 + 112 * (-10) = 7$$

The following is a useful corollary of the GCD theorem.

Corollary 1: If b and c are relatively prime, then there exist integers x and y such that

$$b * x + c * y = 1$$

b, a%b

In cryptography, we often need to compute *multiplicative inverses modulo a prime* number. Corollary 1 can be used to obtain the inverse of c modulo a prime integer, b. Since c^*y differs from 1 by an integral multiple of b, $c^*y \equiv 1$ (*mod b*). It follows that y (reduced modulo b) is actually the inverse of c *modulo b*.

The formal procedure to obtain the inverse of c modulo b is called the *Extended Euclidean Algorithm* and is presented next. This algorithm is essentially an automated version of Example 5. It assumes that b and c are relatively prime.

```
ComputeInverse(b, c)          // Computes inverse of c mod b
{
    old1 = 1          new1 = 0
    old2 = 0          new2 = 1
    b' = b            c' = c
    r = 2
    while (r > 1) {
        q = $\frac{b'}{c'}$                    // compute quotient
        r = b' % c'                  // compute remainder
        temp1 = old1 – new1 * q
        old1 = new1
        new1 = temp1

        temp2 = old2 – new2 * q
        old2 = new2
        new2 = temp2

        b' = c'                  // update dividend
        c' = r                   // update divisor

        // At this point new1 * b + new2 * c = r
    }
    return new2
}
```

Figure 3.2 *Extended Euclidean Algorithm*

Example 3.6

We use the Extended Euclidean Algorithm (Fig. 3.2) to compute the inverse of 12 modulo 79 ($b = 79$ and $c = 12$).

Table 3.1 tracks the values of b', c', q, r, *old1*, *new1*, *old2*, and *new2* at the end of each iteration. The invariant, $new1 \times b + new2 \times c = r$ is maintained across all iterations. At the end of the last iteration $r = 1$ and the invariant is

$$(-5) \times 79 + 33 \times 12 = 1.$$

or

$$12 * 33 = 1 + 5 * 79 \equiv 1 \text{ (mod 79)}$$

Thus, the inverse of 12 modulo 79 is 33.

Table 3.1 *Illustrating the Extended Euclidean Algorithm – Finding inverse of 12 modulo 79*

Iteration	b′	c′	q	r	old1	new1	old2	new2	Invariant new1 × b + new2 × c = r
—	79	12	—	2	1	0	0	1	—
1	12	7	6	7	0	1	1	−6	$1 \times 79 + (-6) \times 12 = 7$
2	7	5	1	5	1	−1	−6	7	$(-1) \times 79 + 7 \times 12 = 5$
3	5	2	1	2	−1	2	7	−13	$2 \times 79 + (-13) \times 12 = 2$
4	2	1	2	1	2	−5	−13	33	$(-5) \times 79 + 33 \times 12 = 1$

3.3 USEFUL ALGEBRAIC STRUCTURES

3.3.1 Groups

A group is the most basic algebraic structure used in cryptography.

Definition: A *group* is a pair $\langle G, *\rangle$, where G is a set and $*$ is a binary operation such that the following hold:

Closure	If a and b are elements of G, then so is $a*b$
Associativity	If a, b, and c are elements of G then $a*(b*c) = (a*b)*c$
Identity element	There exists an element I in G such that for all b in G, $I * b = b = b * I$
Inverse	For each element b in G, there exists exactly one element c in G such that $b*c = c*b = I$ (c is referred to as the inverse of b).

A group may be infinite – an example is the set of all integers with regular addition. The groups of interest to us in cryptography are finite groups. An example of a finite group is the set $\{0, 1, \ldots n-1\}$ with the operation "addition modulo n," where n is a positive integer. The set $\{0, 1, \ldots n-1\}$ is denoted by Z_n and the operation, "addition modulo n" is denoted by $+_n$. It is easy to verify that $\langle Z_n, +_n\rangle$ is a group. The identity element of this group is 0 and the inverse of an element b is $-b$.

Example 3.7

$\langle Z_n - \{0\}, \underset{n}{*}\rangle$ is a group only if n is a prime number. Here, $\underset{n}{*}$ denotes multiplication modulo n.

Closure and associativity hold in such a structure. Also, 0 is the identity element. But, does each element have an inverse?

If n is prime, then by Corollary 1, for any b in Z_n, we can find integers c and d such that $b * c + n * d = 1$. So, $b * c \equiv 1$ *mod* n. Hence, $b^{-1} \equiv c \bmod n$.

If n is non-prime, let f be one of its non-zero factors. Suppose f has an inverse, say h. So, $f * h = 1 \bmod n$ or $f * h = 1 + k * n$ (for some integer k). Now dividing throughout by f, we end up equating a fraction to an integer – a contradiction. Hence, some elements in Z_n have no inverses. So $\langle Z_n - \{0\}, \underset{n}{*}\rangle$ is not a group if n is non-prime.

We conclude that $\langle Z_n - \{0\}, \underset{n}{*}\rangle$ is a group if and only if n is prime.

Notation: Let Z_n^* denote the set

$$\{i \mid 0 < i < n \quad \text{and} \quad gcd(i, n) = 1\}$$

i.e., Z_n^* is the set of all integers modulo n that are relatively prime to n.

Example 3.8

$$Z_5^* = \{1, 2, 3, 4\} \quad \text{and} \quad Z_6^* = \{1, 5\}$$

It can be shown that $\langle Z_n^*, *_n \rangle$ is a group.

Definition: The order of a group, $\langle G, * \rangle$ is the number of elements in G.

Definition: The *Euler Totient Function*, denoted by $\varphi(n)$, is the order of $\langle Z_n^*, *_n \rangle$. So,

$$\varphi(5) = 4$$
$$\varphi(6) = 2$$

Definition: $\langle G', * \rangle$ is a *sub-group* of $\langle G, * \rangle$ if $\langle G', * \rangle$ satisfies the group properties enumerated earlier and G' is a sub-set of G.

Example 3.9

Consider $Z_5^* = \{1, 2, 3, 4\}$. The different sub-groups of $\langle Z_5^*, *_5 \rangle$ are $\langle \{1\}, *_5 \rangle$ and $\langle \{1, 4\}, *_5 \rangle$ in addition to $\langle Z_5^*, *_5 \rangle$.

Lagrange's Theorem: The order of any subgroup of a group divides the order of the parent group.

This theorem tells us that there are constraints on the number of elements in any subgroup of a given group. Consider $\langle Z_5^*, *_5 \rangle$. Since its order is 4, no sub-group of it can have order = 3. Its three subgroups have orders 1, 2, and 4 (see Example 3.9).

Let $\langle G, * \rangle$ be a finite group and let g be an element of G. We use g^i to denote the element obtained by performing the operation $*$ on g, i times.

$$g * g * g \ldots * g \qquad i \text{ times}$$

Consider the elements

$$\begin{aligned} &g \\ &g^2 = g * g \\ &g^3 = g * g^2 \\ &g^4 = g * g^3 \\ &\ldots \end{aligned}$$

Each element in the above list is in G since the elements of $\langle G, * \rangle$ are closed under $*$. So, the list must repeat at some point. In general, if elements in the i-th and j-th positions of the list are identical, then so must elements in the $(i-1)$-th and $(j-1)$-th positions. [This is because an element in the $(i-1)$-th position is obtained from an element in the i-th position by multiplication by g^{-1}.]

We conclude that the first element to repeat in the list is g itself. The element immediately preceding g is the identity, I. Let S denote the set of unique elements from the start element to the first occurrence of I in the above list. Then, we note the following:

- $\langle S, * \rangle$ is a subgroup of $\langle G, * \rangle$. We say that $\langle S, * \rangle$ is a subgroup generated by g and denote it as $\langle g \rangle$.
- The order of $\langle g \rangle$ divides the order of $\langle G, * \rangle$ by Lagrange's Theorem.

Let k denote the order of $\langle g \rangle$. So, $g^k = I$. By Lagrange's Theorem, k divides $|G|$. So, there exists an integer j such that $j \times k = |G|$. So, $g^{|G|} = (g^k)^j = I^j = I$. This leads to the following corollary of Lagrange's Theorem:

Fact: Let G be a group. Let g be an element in G and I be the identity in G. Then, $g^{|G|} = I$.

We next apply this result to the group, $\langle Z^*_n, *_n\rangle$. Recall that $\Phi(n)$ denotes the order of the group, $\langle Z^*_n, *_n\rangle$.

Fact: $\forall m \in Z^*_n$, $m^{\Phi(n)} \bmod n = 1$.

A more general result follows easily from the above fact.

Euler's Theorem: If m and n are relatively prime, $m^{\Phi(n)} \bmod n = 1$.

A special case of Euler's Theorem is the following version of Fermat's Little Theorem when n is prime.

Fermat's Little Theorem: Let p be prime and let m be a non-zero integer that is not a multiple of p. Then, $m^{p-1} \bmod p = 1$.

Definition: A group $\langle G, *\rangle$ is *cyclic* if there is at least one element g in it such that $\langle g\rangle$ is $\langle G, *\rangle$. We refer to such an element of $\langle G, *\rangle$ as a *generator* of $\langle G, *\rangle$.

Example 3.10

Consider $\langle Z^*_{13}, *_{13}\rangle$. In this group,

$$\begin{array}{lll}
2^1 \bmod 13 = 2 & 2^5 \bmod 13 = 6 & 2^9 \bmod 13 = 5 \\
2^2 \bmod 13 = 4 & 2^6 \bmod 13 = 12 & 2^{10} \bmod 13 = 10 \\
2^3 \bmod 13 = 8 & 2^7 \bmod 13 = 11 & 2^{11} \bmod 13 = 7 \\
2^4 \bmod 13 = 3 & 2^8 \bmod 13 = 9 & 2^{12} \bmod 13 = 1
\end{array}$$

Hence, 2 is a generator of $\langle Z^*_{13}, *_{13}\rangle$. However, 3 is not a generator of $\langle Z^*_{13}, *_{13}\rangle$.

Now consider $\langle Z^*_8, *_8\rangle = \langle\{1, 3, 5, 7\}, *_8\rangle$. In this case,

$$3^2 \bmod 8 = 5^2 \bmod 8 = 7^2 \bmod 8 = 1$$

We conclude that $\langle Z^*_{13}, *_{13}\rangle$ is a cyclic group with at least one generator = 3. On the other hand, $\langle Z^*_8, *_8\rangle$ is not a cyclic group since no element in it "generates" all the elements of $\langle Z^*_8, *_8\rangle$.

The group, $\langle Z^*_p, *_p\rangle$, where p is prime, is of special interest in cryptography. We next enumerate some of its useful properties.

Fact: The group, $\langle Z^*_p, *_p\rangle$, where p is prime, is cyclic.

Fact: The number of generators in $\langle Z^*_p, *_p\rangle$ is $\Phi(p - 1)$.

We can verify that the group, $\langle Z^*_{13}, *_{13}\rangle$ has $\Phi(12) = 4$ generators. These are 2, 6, 11, and 7.

Fact: Let p be prime and let $p_1, p_2, \ldots, p_k$ be the distinct prime factors of $p - 1$. Then, g is a generator of $\langle Z^*_p, *_p\rangle$ if and only if

$$g^{(p-1)/pi} \neq 1 \bmod p \qquad \text{for all } p_i, \qquad 1 \le i \le k$$

Example 3.11

We check whether 7 and 3 are generators of $\langle Z^*_{13}, *_{13}\rangle$.

The distinct prime factors of 12 are 2 and 3. So, $p_1 = 2$ and $p_2 = 3$. To test if 7 is a generator of $\langle Z^*_{13}, *_{13}\rangle$, we compute

$$7^{12/2} \bmod 13 = 7^6 \bmod 13$$
$$\equiv -1$$

and

$$7^{12/3} \bmod 13 = 7^4 \bmod 13$$
$$\equiv 9$$

So, 7 passes the test and is a generator. On the other hand,

$$3^{12/2} \bmod 13 = 3^6 \bmod 13$$
$$\equiv 1$$

Hence, 3 fails the test and is not a generator of $\langle Z_{13}^*, *_{13}\rangle$.

3.3.2 Rings

A ring is a triplet $\langle R, +, *\rangle$, where + and * are binary operations and R is a set satisfying the following properties:

- $\langle R, +\rangle$ is a *commutative group*. The *additive identity* is designated 0.
- For all x, y, and z in R,
  - $x * y$ is also in R. (In other words, the set R is *closed under* *).
  - $x * (y * z) = (x * y) * z$. (In other words, * is an *associative* operation).
  - $x * (y + z) = x * y + x * z = (y + z) * x$. (We say that * *distributes* over +).

A few other properties (or non-properties) of a ring are worthy of note.

(1) All rings that we use have a *multiplicative identity* designated as 1.
(2) Unlike the + operation, the operation * does not need to be commutative. If * is commutative, the ring is said to be a *commutative ring*.
(3) While each element, x, in R has an additive inverse (denoted by $-x$), an element need not have a multiplicative inverse (denoted by x^{-1}, if it exists).

Example 3.12

The *set of non-negative integers* with the regular definitions of + and * do not constitute a ring since none of the elements (except 0) has an additive inverse. On the other hand, it can be easily verified that the *set of all integers* with + and * does form an infinite ring. This structure has the additional property of being a commutative ring since the * operation in this case is commutative.

The set of 2 × 2 matrices populated by reals forms a ring. However, this ring is non-commutative.

The set of integers modulo n or, more precisely, $\langle Z_n, +_n, *_n\rangle$ is a finite, commutative ring.

Polynomial Rings: Let $Z_p[x]$ be the set of all polynomials in x with coefficients belonging to Z_p. We restrict attention to prime values of p. We define addition of two polynomials in the usual way except that addition of coefficients is modulo p.

For illustration, consider two polynomials in $Z_3[x]$,

$$a(x) = 2x^4 + x^3 + 2x + 1 \quad \text{and} \quad b(x) = x^5 + x^4 + 2x$$

- $a(x) + b(x)$ is computed below.

$$\begin{array}{rcllll} a(x) & = & & 2x^4 + x^3 & & + 2x + 1 \\ b(x) & = & x^5 + x^4 & & & + 2x \\ \hline a(x) + b(x) & = & x^5 & + x^3 & & + x + 1 \\ \hline \end{array}$$

Multiplication is performed similar to regular long-hand multiplication except that the addition and multiplication of coefficients are performed using modulo p arithmetic.

- $a(x) * b(x)$ is computed below.

$$
\begin{array}{rcrrrrrrrrrr}
a(x) & = & & & & & & 2x^4 & +\,x^3 & & +\,2x & +\,1 \\
b(x) & = & & & & & x^5 & +\,x^4 & & & +\,2x & \\
\hline
 & & & & & & 4x^5 & +\,2x^4 & & +\,4x^2 & +\,2x & \\
 & & & 2x^8 & +\,x^7 & & +\,2x^5 & +\,x^4 & & & & \\
 & & 2x^9 & +\,x^8 & & +\,2x^6 & +\,x^5 & & & & & \\
\hline
a(x) * b(x) & = & 2x^9 & & +\,x^7 & +\,2x^6 & +\,x^5 & & & +\,x^2 & +\,2x & \\
\hline
\end{array}
$$

Fact: $Z_p[x]$ with the above-defined operations of polynomial addition and multiplication comprise a ring.

3.3.3 Fields

A field, $\langle R, +, * \rangle$, is a commutative ring with the following additional properties:

- R has a multiplicative identity (denoted by 1) distinct from 0 (the additive identity).
- Each element, x, in R (except for 0) has an inverse element in R, denoted by x^{-1}, such that

$$x*x^{-1} = x^{-1}*x = 1$$

The above properties imply that $\langle R, + \rangle$ and $\langle R - \{0\}, * \rangle$ are each commutative groups.

Example 3.13

- The set of all real numbers with regular addition and multiplication comprises an infinite field.
- However, the set of integers does not comprise a field since no non-zero element has a multiplicative inverse except for 1.
- Every non-zero element in Z_n has a multiplicative inverse if n is prime. It follows that $\langle Z_n, +_n, *_n \rangle$ is a field if and only if n is prime.

It can be shown that the cardinality of any finite field has the form p^m where p is prime and m is a positive integer. It can be shown that all fields of a given cardinality are isomorphic[1]. A field of size p^m is commonly denoted by $GF(p^m)$ or $F(p^m)$. An example of a field for $m = 1$ is precisely $\langle Z_p, +_p, *_p \rangle$. Fields of cardinality p^m, $m > 1$, are conveniently represented using polynomials of degree $m - 1$ over Z_p which we discuss next.

3.3.3.1 *Polynomial Fields*

We created the field, $\langle Z_p, +_p, *_p \rangle$ by restricting attention to the field of integers modulo the prime number p. In a similar way, we could consider a subset of the polynomial ring $Z_p[x]$ by "reducing" a polynomial in $Z_p[x]$ modulo a "prime" polynomial. A prime or irreducible polynomial in $Z_p[x]$ is one which has no factors in $Z_p[x]$ other than itself and 1. We explain what is meant by the reduction of a polynomial modulo a given irreducible polynomial after providing examples of prime polynomials.

[1] Two fields, A and B, are isomorphic if there exists a 1-1 and onto mapping between the elements of the two fields such that substitution of each element of A by the corresponding element of B in the addition and multiplication tables of A yields the corresponding tables of B.

The factors (if any) of each polynomial over Z_2 of degree 3 are listed below.

x^3	$= (x)\ (x)\ (x)$
$x^3 + 1$	$= (x + 1)(x^2 + x + 1)$
$x^3 + x$	$= x(x + 1)^2$
$x^3 + x^2$	$= x^2(x + 1)$
$x^3 + x + 1$	
$x^3 + x^2 + 1$	
$x^3 + x^2 + x$	$= x(x^2 + x + 1)$
$x^3 + x^2 + x + 1$	$= (x + 1)^3$

We see that there are only two polynomials of degree 3 over Z_2 that cannot be factorized into lower degree polynomials. It turns out that for any $m > 1$ and any prime p, there is at least one irreducible polynomial over Z_p of degree m. We next study how a field of size p^m may be generated using any one (of possibly many) irreducible polynomials of degree m over Z_p.

We represent the elements of a field of size p^m using all polynomials over Z_p of degree $m - 1$ or less. Note that there are exactly p^m such polynomials.

The operation + is identical to that described for addition of two elements in a polynomial ring over Z_p.

The operation * is also performed as in the case for polynomial rings. However, the degree of the product polynomial could exceed $m - 1$. In that case, the product is divided by the irreducible polynomial of degree m. The remainder obtained upon division is the required result. Note that the remainder is a polynomial of degree $m - 1$ or less over Z_p. Thus, multiplication of two elements in the set yields an element of the set as it should to satisfy the property of closure with respect to multiplication.

Let us choose the irreducible polynomial, $x^3 + x^2 + 1$ to construct a field, GF(2^3) of $2^3 = 8$ elements. Suppose that the elements $x^2 + 1$ and x^2 are to be multiplied. The first step is multiplication in the polynomial ring over Z_2. This yields $x^4 + x^2$. The next step is reduction modulo $x^3 + x^2 + 1$ shown below using polynomial division.

$$\begin{array}{rl}
 & x + 1 \\ \cline{2-2}
x^3 + x^2 + 1 \big) & x^4 \qquad\quad + x^2 \\
 & x^4 + x^3 \qquad\quad + x \\ \cline{2-2}
 & \qquad\ x^3 + x^2 + x \\
 & \qquad\ x^3 + x^2 \qquad + 1 \\ \cline{2-2}
 & \qquad\qquad\qquad x + 1
\end{array}$$

The remainder, $x + 1$, is the product of the two polynomials, $x^2 + 1$ and x^2. Table 3.2 shows the multiplication table for all field elements of GF(2^3) (represented by the set of all polynomials of degree 2 and lower over Z_2 using the irreducible polynomial, $x^3 + x^2 + 1$). For ease of viewing, we represent each polynomial by a binary string of its coefficients. For example, $x^2 + 1$ is represented as 101. The product of two field elements, say $x^2 + 1$ (the string 101) and $x + 1$ (the string 011) is found in the cell which lies on row 101 and column 011.

Table 3.2 Multiplication table for elements in GF(2^3) using irreducible polynomial $x^3 + x^2 + 1$

*	000	001	010	011	100	101	110	111
000	000	000	000	000	000	000	000	000
001	000	**001**	010	011	100	101	110	111
010	000	010	100	110	101	111	**001**	011
011	000	011	110	101	**001**	010	111	100
100	000	100	101	**001**	111	011	010	110
101	000	101	111	010	011	110	100	**001**
110	000	110	**001**	111	010	100	011	101
111	000	111	011	100	110	**001**	101	010

Finally, all the shaded cells in Table 3.2 contain the value 001 (or the multiplicative identity). To find the multiplicative inverse of a non-zero element, say 110, proceed along row 110 until you hit the shaded cell, then read out the column number of the shaded cell. From Table 3.2, the inverse of 110 is 010.

The multiplicative inverse of an element is obtained using a straightforward generalization of the Extended Euclidean Algorithm (Section 3.2.1).

The fields of interest to us in cryptography are GF(p) and GF(2^m). These are respectively referred to as prime fields and binary fields. They are used extensively in Elliptic Curve Cryptography (Chapter 9).

3.4 CHINESE REMAINDER THEOREM

The Chinese Remainder Theorem is used in proving a number of results in cryptography. Consider the factorization of an integer, N

$$N = n_1 * n_2 \ldots * n_k$$

where $n_1, n_2, \ldots$ are pairwise relatively prime, i.e., $\gcd(n_i, n_j) = 1$, $1 \le i, j \le k$, $i \ne j$. Consider the mapping

$$f: \; Z_n \to Z_{n_1} \times Z_{n_2} \times \ldots \times Z_{n_k} \quad \text{defined by}$$

$f(x) = (x \bmod n_1,\ x \bmod n_2, \ldots,\ x \bmod n_k)$, $0 \le i < N$

Example 3.14

Let $N = 30$. Choose $n_1 = 6$ and $n_2 = 5$.
$f(i)$, $0 \le i < 30$ is shown below.

$f(0) = (0, 0)$,	$f(1) = (1, 1)$,	$f(2) = (2, 2)$,	$f(3) = (3, 3)$,
$f(4) = (4, 4)$,	$f(5) = (5, 0)$,	$f(6) = (0, 1)$,	$f(7) = (1, 2)$,
$f(8) = (2, 3)$,	$f(9) = (3, 4)$,	$f(10) = (4,0)$,	$f(11) = (5, 1)$,
$f(12) = (0, 2)$,	$f(13) = (1, 3)$,	$f(14) = (2, 4)$,	$f(15) = (3, 0)$,
$f(16) = (4, 1)$,	$f(17) = (5, 2)$,	$f(18) = (0, 3)$,	$f(19) = (1, 4)$,
$f(20) = (2, 0)$,	$f(21) = (3, 1)$,	$f(22) = (4, 2)$,	$f(23) = (5, 3)$,
$f(24) = (0, 4)$,	$f(25) = (1, 0)$,	$f(26) = (2, 1)$,	$f(27) = (3, 2)$,
$f(28) = (4, 3)$,	$f(29) = (5, 4)$		

It is straightforward to compute $f(x)$ given an x. However, given a tuple in $Z_{n_1} \times Z_{n_2} \times \ldots \times Z_{n_k}$, how do we reverse-map it to Z_N? At a more fundamental level, given a tuple $(x_1, x_2, \ldots, x_k) \in Z_{n_1} \times Z_{n_2} \times \ldots \times Z_{n_k}$, does an $x \in Z_N$ even exist such that $f(x) = (x_1, x_2, \ldots x_k)$? We next show that such a reverse mapping does indeed exist.

Define

$$a_i = \frac{N}{n_i},\ 1 \le i \le k.$$

Let α_i denote the inverse of a_i in the modulo n_i sense, i.e.,

$$\alpha_i \times a_i \equiv 1 \pmod{n_i},\quad 1 \le i \le k.$$

Then, given a tuple $(x_1, x_2, \ldots x_k) \in Z_{n_1} \times Z_{n_2} \times \ldots \times Z_{n_k}$, compute

$$x = (x_1 \times a_1 \times \alpha_1 + x_2 \times a_2 \times \alpha_2 \ldots + x_k \times a_k \times \alpha_k) \bmod N$$

We claim that $f(x) = (x_1, x_2, \ldots x_k)$. To verify the claim, we note that

$$(x_1 \times a_1 \times \alpha_1 + x_2 \times a_2 \times \alpha_2 \ldots + x_k \times a_k \times \alpha_k) \bmod n_i = x_i$$

For a given n_i, the term, $x_i \times a_i \times \alpha_i \bmod n_i = x_i$ since, by definition, a_i and α_i are inverses modulo n_i. The other terms have the form, $x_j \times a_j \times \alpha_j \bmod n_i$, $i \ne j$. They are each zero. This is so since, by construction, each a_j, $j \ne i$, has n_i as a factor.

Example 3.15

Let $N = 210$ and let $n_1 = 5$, $n_2 = 6$, $n_3 = 7$.
Compute $f^{-1}(3, 5, 2)$, i.e., given $x_1 = 3$, $x_2 = 5$, $x_3 = 2$, compute x.
We have

$$a_1 = N/n_1 = 42,$$
$$a_2 = N/n_2 = 35 \quad \text{and}$$
$$a_3 = N/n_3 = 30.$$

$$\alpha_1 = 42^{-1} \pmod 5 = 3 \checkmark$$
$$\alpha_2 = 35^{-1} \pmod 6 = 5 \quad \text{and}$$
$$\alpha_3 = 30^{-1} \pmod 7 = 4.$$

So,

$$\begin{aligned} x &= (x_1 \times a_1 \times \alpha_1 + x_2 \times a_2 \times \alpha_2 + x_3 \times a_3 \times \alpha_3) \pmod N \\ &= (3 * 42 * 3 + 5 * 35 * 5 + 2 * 30 * 4) \pmod{210} \\ &= 1493 \pmod{210} \\ &= 23 \end{aligned}$$

SELECTED REFERENCES

Number Theory and Discrete Structures are vast subjects and there are a large number of texts in these areas. The relevant mathematical background for a first course in cryptography introduced in this text has been covered in this chapter. Further material on Number Theory may be found in [JONE98], [ROSE04], [BURT02], or [SILV06]. Further material on algebraic structures may be found in [GALL09], [DURB04], and [HERS96].

OBJECTIVE-TYPE QUESTIONS

3.1 (403*6000*5981*378) mod 9 equals
(a) 7 (b) 3 (c) 0 (d) 4

3.2 The smallest positive integer satisfying
$x \bmod 501 = 10$ and
$x \bmod 502 = 10$ is
(a) $x = 251{,}502$ (b) $x = 512$ (c) $x = 511$ (d) none of these

3.3 If $\gcd(a,b) = x$ and $\gcd(b,c) = y$, then $\gcd(a,c)$ is
(a) xy (b) $\gcd(x,y)$ (c) $\dfrac{xy}{\gcd(x,y)}$ (d) none of these

3.4 The number of generators in the group $\langle Z_7^*, *_7\rangle$ is
(a) 1 (b) 2 (c) 3 (d) 4

3.5 The number of subgroups of the group $\langle Z_7^*, *_7\rangle$ is
(a) 3 (b) 4 (c) 5 (d) 6

3.6 Which of the following is/are an invalid size for a finite field?
(a) 100 (b) 89 (c) 289 (d) 133

3.7 The triplet, $\langle Z_n, +_n, *_n\rangle$ for non-prime n is
(a) a commutative ring (b) a non-commutative ring
(c) a ring with a multiplicative inverse (d) a field

EXERCISES

3.1 Does the following hold for any choice of a, b, x, and n?

$$x^{a+b} \equiv x^{(a+b) \bmod n} \pmod n$$

3.2 Compute gcd(6622, 645).

3.3 Do each of the following inverses exist? If yes, what are they? If no, explain why not.

$$102^{-1} \pmod{411}$$
$$77^{-1} \pmod{411}$$

3.4 Check whether 14 is a generator of the group $\langle Z_{457}^*, *_{457}\rangle$.

3.5 Can you think of a finite group that has
(a) no generator?
(b) only a single generator?
(c) |G| generators (where |G| is the cardinality of the group)?
(d) |G| – 1 generators?

3.6 Can you think of a ring of size 9 (i.e., there are 9 elements in the ring). You should clearly define the ring operations, + and *.
Is that ring also a field? Why or why not?

3.7 Re-do the multiplication table (Table 3.2) using the irreducible polynomial $x^3 + x + 1$ instead.

3.8 List all irreducible polynomials over Z_2 of degree 4.

3.9 Consider the field GF(2^4). Let field multiplication be performed modulo the irreducible polynomial $x^4 + x + 1$. Compute each of the following
(a) (1100) + (1001)
(b) (1011) * (0111)
(c) $(1101)^{-1}$

3.10 The notion of a trace is used to speed up certain cryptographic operations studied in Chapter 9. Consider the binary field, GF(2^m). The *trace* of an element, x, in the field, GF(2^m), is defined as

$$Tr(x) = x + x^2 + x^4 \ldots + x^{2^{m-1}}$$

Prove that $Tr(x) = 0$ or 1 for *any* x in the field.

3.11 An integer, n, $0 \le n < 210$, satisfies the following set of congruences:

$$n \bmod 5 = 4$$
$$n \bmod 6 = 3$$
$$n \bmod 7 = 2$$

What is n?

3.12 Use the Chinese Remainder Theorem to obtain all square roots of 1 mod 840. Also, list all the square roots of –1 mod 840.

ANSWERS TO OBJECTIVE-TYPE QUESTIONS

3.1 (c)	3.2 (a)	3.3 (d)	3.4 (b)
3.5 (b)	3.6 (a)(d)	3.7 (a)	

Chapter 4

Basics of Cryptography

In this chapter, we introduce the basics of encryption and decryption. Early ciphers such as the Caesar, Vignere, and Hill are studied. We then study the building blocks of modern secret key ciphers – the transposition and substitution cipher. Some basic properties of ciphers and different classes of attacks are also introduced.

4.1 PRELIMINARIES

Cryptography is the science of disguising messages so that only the intended recipient can decipher the received message. Cryptography is the lynchpin of data security – besides providing for message *confidentiality*, it also helps in providing message *integrity*, *authentication*, and *digital signatures*. Without it, e-banking, e-trading, and e-commerce would simply not be a reality.

The original message or document to be transferred is called *plaintext* and its disguised version is called *ciphertext*. It is often (but not always) the case that the plaintext and ciphertext are binary strings of the same length. The process of disguising the original plaintext is called *encryption* and the process of recovering the original plaintext from the ciphertext is called *decryption*.

Encryption involves the use of an *encryption function* or algorithm, denoted by E, and an *encryption key*, e. Likewise, decryption involves the use of a *decryption function* denoted by D, and a *decryption key*, d. These operations are summarized below.

$$c = E_e(p) \quad \text{and}$$
$$p = D_d(c)$$

Here, p denotes a *block* of plaintext. It is encrypted by the sender to produce ciphertext denoted by c. The second operation is performed by the receiver on the ciphertext to recover the plaintext.

In the early days of cryptography, the actual algorithms for decryption were kept secret. More recently, decryption algorithms for wireless communication were shrouded in mystery. "Ethical hackers" reverse-engineered the code and discovered numerous bugs which were brought to the attention of the cryptographic community. Had the decryption algorithms been placed in the public domain prior to their incorporation in standards, they would have been exposed to scrutiny and the bugs in them would have been identified early on. Fortunately, today such examples are the exception rather than the rule.

Kerckhoff's Principle: The secrecy should be in the key used for decryption, not in the decryption or encryption algorithms.

4.1.1 Secret versus "Public" Key Cryptography

There are two types of cryptography in widespread use – secret key cryptography and public key cryptography. They are defined as follows:

- In *secret key cryptography*, both sender and receiver share a common secret – the same secret is used for encryption as well as decryption. So $e = d$ in the equation above. Hence, this form of cryptography is also referred to as *symmetric key cryptography.*
- In *public key cryptography*, two distinct keys forming a key pair are used – the encryption key or *public key* and the decryption key or *private key*. The public key of a user is used to encrypt messages to that user. It is intended to be known to the outside world. The corresponding private key, however, should not be revealed to anyone. It is the private key of the recipient that is used to decrypt the message.

  Note that the private key here has no relation with the "secret key" used in secret key cryptography. Because the public and private keys are distinct, this form of cryptography is also referred to as *asymmetric key cryptography*.

Let us see how the two types of cryptography may be employed when Alka intends to send a confidential message to Brijesh.

If Alka and Brijesh share a secret key, k, then she encrypts the message using the common secret. The encrypted message received by Brijesh is decrypted using the same secret. The secret key operations are summarized below.

Operation performed by Alka	$c = E_k(p)$
Operation performed by Brijesh:	$p = D_k(c)$

On the other hand, Alka may wish to use public key cryptography. Assuming that Brijesh has a public key–private key pair, she would encrypt her message using his public key, B.pu. Brijesh then decrypts the message using the corresponding private key, B.pr. Assuming that Brijesh keeps his private key securely, he and only he can decrypt the message received from Alka. The public key–private key operations are summarized below.

Operation performed by Alka:	$c = E_{B.pu}(p)$
Operation performed by Brijesh:	$p = D_{B.pr}(c)$

There are several cryptographic algorithms widely used today. The two best known ones for secret key cryptography are the *Data Encryption Standard* (DES) covered in Chapter 5 and the *Advanced Encryption Standard* (AES) covered in Chapter 9. Other examples of secret key algorithms include Blowfish and RC4. The most widely used public key algorithms are *RSA* (named after its originators) and *Elliptic Curve Cryptography* (ECC). These are dealt with in Chapters 6 and 9, respectively.

Choosing a particular cryptographic algorithm depends upon a variety of factors. These include ease of implementation (in hardware and/or software), hardware requirements (number of gates, memory), performance characteristics, and security. For military or high-value financial applications, security is of paramount importance. But what exactly does secure mean?

4.1.2 Types of Attacks

At a very high level, a cryptographic algorithm is *secure* if a cryptanalyst (a person with expertise in breaking ciphers) is unable to

(a) obtain the corresponding plaintext from a given ciphertext
(b) deduce the secret key or the private key

How would the attacker proceed to realize the above objectives? He could accumulate copious amounts of ciphertext. He would then look for patterns in the ciphertext in an attempt to reconstruct some plaintext and/or deduce the key. Such an attack which exclusively uses ciphertext is referred to as a *"known ciphertext" attack.*

Occasionally, all or part of some plaintext blocks are predictable or may be guessed. A cryptanalyst may then build a repertoire of corresponding plaintext, ciphertext pairs with the intention of deducing the key. Such an attack is referred to as a *"known plaintext" attack*. It may even be possible for a shrewd attacker to carefully choose pieces of plaintext and then induce the sender to encrypt such text. An attack on a cryptographic scheme which makes use of pairs of attacker-chosen plaintext and the corresponding ciphertext is referred to as a *"chosen plaintext" attack*.

The most obvious, though compute-intensive, attack with known plaintext is a *brute force* attempt at obtaining the key by trying all possible key values.
Let (p_1, c_1), (p_2, c_2), … (p_m, c_m) be plaintext–ciphertext pairs.

```
for (each potential key value, $k$ in the key space)
{
        $proceed$ = true;
        $i = 1$;

        while ($proceed$ = = true   &&   $i \le m$) {
                if ($c_i \neq E_k(p_i)$)
                        $proceed$ = false;
                $i$ ++ ;
        }

        if ($i = m+1$)
                print (" Key Value is $k$ ");
}
```

4.2 ELEMENTARY SUBSTITUTION CIPHERS

4.2.1 Monoalphabetic Ciphers

The most basic cipher is a substitution cipher. For ease of understanding, we consider English text in all the examples in this chapter. Let Σ denote the set of alphabets, {A, B, . . . Z}. A monoalphabetic substitution cipher defines a permutation of the elements in Σ. There are 26! permutations; so, there are 26! possible monoalphabetic substitution ciphers.

The simplest substitution cipher is one that replaces each alphabet in a text by the alphabet k positions away (in the modulo 26 sense). For $k = 3$, the substitutions are

D for **A**, E for **B**,… A for **X**, B for **Y** etc.

Such a scheme is referred to as a *Caesar cipher*. A sample plaintext and the corresponding ciphertext for $k = 3$ is

Plaintext:	**WHAT**	**IS**	**THE**	**POPULATION**	**OF**	**MARS**
Ciphertext:	ZKDW	LV	WKH	SRSXODWLRQ	RI	PDUV

One approach to attacking such a cipher is to compute the frequencies of the different alphabets occurring in the ciphertext. Numerous studies have been conducted on the frequency distribution

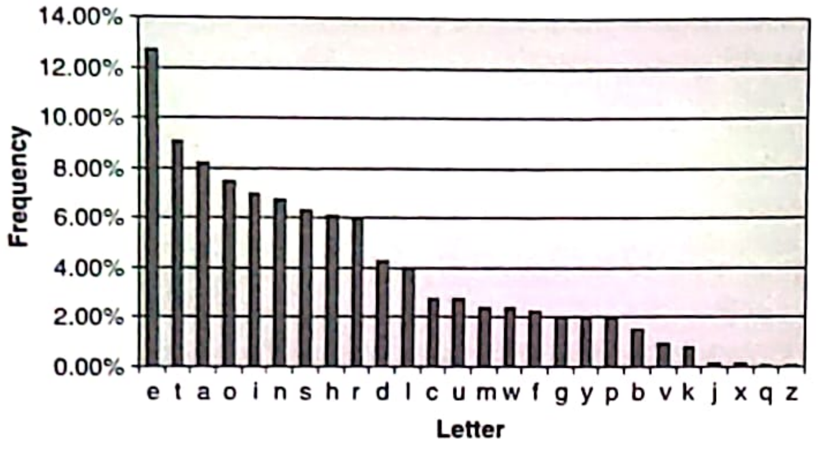


Figure 4.1 *Average frequency of letters in English text*

of the alphabets in regular text (see Fig. 4.1). For example, it is known that the most frequently occurring letters in English are E, T, and A (12.7%, 9.1%, and 8.2%, respectively).

Given a string of ciphertext, we could, as a first cut, substitute the three most frequently occurring letters in the ciphertext for the three most frequently occurring letters in "regular" English text. We could use other rules such as "The letters **R** and **N** never occur consecutively" or "The letter **Q** is usually followed by a **U**." We could write a program that performs various substitutions on the ciphertext and also enforces rules on the occurrence or non-occurrence of consecutive letters. It is possible to use insights on statistical properties of letters in English text to guess the plaintext by studying the ciphertext alone. This is the basis of a known ciphertext attack.

In substitution ciphers, like the Caesar cipher, each letter is always substituted for another unique letter. Such ciphers are said to be *monoalphabetic*. We next study polyalphabetic ciphers.

4.2.2 Polyalphabetic Ciphers

In a *polyalphabetic* cipher, the ciphertext corresponding to a particular character in the plaintext is not fixed. It may depend on, for example, its position in the block. We next study two examples of such ciphers.

The Vigenere Cipher

The *Vigenere cipher* is a polyalphabetic cipher that uses a multi-digit key, $k_1, k_2, \ldots, k_m$. Here, $k_1, k_2, \ldots k_m$ are each integers. The plaintext is split into non-overlapping blocks, each containing m consecutive characters. Then the first letter of each block is replaced by the letter k_1 positions to its right, the second letter of each block is replaced by the letter k_2 positions to its right, and so on.

Plaintext:	*W i s h i n g*	*Y o u*	*M u c h*	*S u c c e s s*
+				
Key:	04 19 03 22 07 12 05	11 04 19	03 22 07 12	05 11 04 19 03 22 07
=				
Ciphertext:	A B V D P Y L	J S N	P Q J T	X F G V H O Z

The first letter in the above text is W. The corresponding key is 04. This means that the ciphertext is the letter 4 positions ahead (in the modulo 26 sense). The key length = 8, i.e., the keystring repeats after every 8 characters. There are four occurrences of the letter "s" in the above text (for simplicity we do not distinguish between upper and lower case letters). However, each occurrence of "s" is encrypted as a different character in the ciphertext – "$\mathbb{V}$", "$\mathbb{X}$", "$\mathbb{O}$," and "$\mathbb{Z}$".

The Hill Cipher

The *Hill cipher* is another polyalphabetic cipher proposed by Lester Hill. As in the Vigenere cipher, the plaintext is broken into blocks of size m. However, the key in the Hill cipher is an $m \times m$ matrix of integers between 0 and 25. Unlike the Caesar and Vigenere ciphers, each character in the ciphertext is a function of all the characters in that block.

Let $p_1, p_2, \ldots p_m$ be the numeric representation of the characters in the plaintext and let $c_1, c_2, \ldots c_m$ represent the corresponding characters in the ciphertext. To compute the ciphertext, we map each alphabet to an integer. We use the mapping, A → 0, B → 1, . . . Z → 25. The relationship between a block of plaintext and its ciphertext is expressed by

$$\begin{aligned}
c_1 &= p_1\, k_{11} + p_2\, k_{21} + \ldots + p_m\, k_{m1} \mod 26\\
c_2 &= p_1\, k_{12} + p_2\, k_{22} + \ldots + p_m\, k_{m2} \mod 26\\
&\ldots\\
&\ldots\\
c_m &= p_1\, k_{1m} + p_2\, k_{2m} + \ldots + p_m\, k_{mm} \mod 26
\end{aligned}$$

This can be conveniently written as

$$c = \mathbf{p}\,\mathrm{K} \tag{4.1}$$

Here, c and p are row vectors corresponding to the plaintext and ciphertext, respectively, and K is the $m \times m$ matrix comprising the key.

At the receiver end, the plaintext can be recovered from the ciphertext by using

$$\mathbf{p} = \mathbf{c}\,\mathrm{K}^{-1} \tag{4.2}$$

Note that not all $m \times m$ matrices have inverses. Decryption will fail if K is singular.

Example 4.1

Consider a Hill cipher using a block size of 2 (m = 2).
Let

$$\mathrm{K} = \begin{pmatrix} 3 & 7 \\ 15 & 12 \end{pmatrix}$$

Let a block of plaintext be (H I). The numeric equivalent of this block is (7 8).
We obtain the corresponding ciphertext using Eq. (4.1). This yields

$$\begin{aligned}
\mathrm{c} &= (7\;\;8) * \begin{pmatrix} 3 & 7 \\ 15 & 12 \end{pmatrix}\\
&= (11\;\;15)\\
&\equiv (\mathbb{L}\;\;\mathbb{P})
\end{aligned}$$

It may be verified that

$$\mathrm{K}^{-1} = \begin{pmatrix} 10 & 5 \\ 7 & 9 \end{pmatrix}$$

Using Eq. (4.2) and the value of K^{-1}, we can verify that the plaintext corresponding to tne ciphertext ($\mathbb{L}$ $\mathbb{P}$) is indeed (H I).

A B [illegible]

How secure are the above substitution ciphers? We saw that the Caesar cipher is susceptible to ciphertext-only attacks that exploit the frequency of characters or character strings in English text. Are similar attacks possible with the Vigenere and Hill ciphers?

In the Vigenere and Hill ciphers, two occurrences of the same character string in the plaintext will not, in general, result in identical character strings in the ciphertext output. However, in the Vigenere cipher, if two identical character strings occur in distinct blocks but their start positions in the two blocks are the same, then their corresponding ciphertext outputs will be the same. In the case of the Hill cipher, if two complete blocks are identical, then their ciphertexts will also be identical.

While known ciphertext attacks need some luck, a *known plaintext attack* is relatively straightforward. In the case of the Vigenere cipher, a single block of plaintext and its corresponding ciphertext are all it takes to deduce the key.

With the Hill cipher, given *m* blocks of plaintext–ciphertext pairs, it may be possible to deduce the key matrix, K. A determined attacker could launch a *chosen plaintext attack* by choosing *m* plaintext blocks as below.

```
1 0 0 0 . . . 0
0 1 0 0 . . . 0
. . .

0 0 0 0 . . . 1
```

Using the first block of plaintext and its corresponding ciphertext, the attacker would obtain the first row of K since $c = p$ K. From the second block of plaintext and its corresponding ciphertext, he/she would obtain the second row of K, and so on.

One-time Pad

The most secure cipher is a one-time pad. Both parties that wish to communicate agree beforehand (through a secure off-line channel) to an *arbitrarily long, random, and non-repeating sequence* of characters. The one-time pad performs encryption in much the same way as the Vigenere cipher. If, for example, the *i*-th entry in the pad is E (corresponding to the integer 4) and the *i*-th character in the plaintext is G, then the corresponding character in the ciphertext is four characters ahead of G, i.e., K.)

Plain G ↓ Pad E(4) ↓ Cipher 'K'

In the Vigenere cipher, the message size may be several times the key size, so each character in the key is used repeatedly. Moreover, multiple messages are encrypted with the same key thus giving the cryptanalyst many leads. On the other hand, a one-time pad, by definition, is assumed to have no repeating subsequences. It is completely random (at least in theory) and is *not re-used*. Hence, given a ciphertext *n* characters long, there is equal likelihood of it being produced from any plaintext string of *n* characters. For this reason, a one-time pad is the most secure cipher ever devised.

There are, however, some serious limitations in the use of the one-time pad. This includes the secure communication of the pad (key) itself. The key is as large as the message – safe storage of such a large key is non-trivial. Generating such a large random pad (it should not be pseudo-random or have repeating subsequences) is a major challenge.

The substitution cipher is one of the building blocks of modern symmetric ciphers. Another building block is the transposition cipher, which we introduce next.

Limitations
1. Comm of Key/pad.
2. Safe storage of key.
3. Generation of key.

4.3 ELEMENTARY TRANSPOSITION CIPHERS

A transposition cipher shuffles, rearranges, or permutes the bits in a block of plaintext. Unlike a substitution cipher, the number of 0's and 1's in a block does not change after the shuffling.

For simplicity, we work with characters (letters) rather than bits. Imagine a block of plaintext arranged in a matrix row by row as below.

Plaintext: *Begin Operation at Noon*

$$\begin{pmatrix} b & e & g & i \\ n & o & p & e \\ r & a & t & i \\ o & n & a & t \\ n & o & o & n \end{pmatrix}$$

Let us rearrange the *rows* as follows
Row 1 → 3, Row 2 → 5, Row 3 → 2, Row 4 → 1 Row 5 → 4.
The resulting matrix is

$$\begin{pmatrix} o & n & a & t \\ r & a & t & i \\ b & e & g & i \\ n & o & o & n \\ n & o & p & e \end{pmatrix}$$

We now rearrange the *columns* as follows
Column 1 → 4 , Column 2 → 3, Column 3 → 1, Column 4 → 2.
The resulting matrix is

$$\begin{pmatrix} a & t & n & o \\ t & i & a & r \\ g & i & e & b \\ o & n & o & n \\ p & e & o & n \end{pmatrix}$$

The ciphertext thus generated is

A T N O T I A R G I E B O N O N P E O N

To decrypt the message, the recipient would have to cast the cipher text in a 5 × 4 matrix, reverse the column shuffles, and then reverse the row shuffles.

An adversary (or spy) could search ciphertext for "interesting" keywords such as "Attack" or "Operation." The letters of the words in a message are not modified as in a substitution cipher but are strewn about in a block of the ciphertext. For example, with a combination of guesswork, luck, and limited prior information, a spy might be able to deduce that the planned start time of an attack is 11:15 pm upon receiving the following ciphertext.

1 1 K C T A T A M M O C P M 5 1 C E N E

This is the ciphertext using the row and column shuffling as in the example above. The corresponding plaintext is

Commence Attack 11 15 pm

4.4 OTHER CIPHER PROPERTIES

Conf
Ciphertext
+ Key

Diff
Plaintext
ciphertext

4.4.1 Confusion and Diffusion

In 1949, Claude Shannon first proposed the ideas of confusion and diffusion in the operation of a cipher. Confusion is the property of a cipher whereby it provides no clue regarding the *relationship between the ciphertext and the key*. This relationship is obfuscated to the point where given a plaintext p, a sequence of keys $k_1, k_2, \ldots k_i$, and the corresponding ciphertexts, $E_{k1}(p)$, $E_{k2}(p)$, $\ldots E_{k_i}(p)$, it is near impossible to deduce the value of a new, arbitrarily chosen key, k_j, used to create the ciphertext, $E_{k_j}(p)$.

Confusion reigns supreme with a cipher if, for any plaintext, p, if even a single bit in a key, k, is changed to produce k', then roughly half the bits in the ciphertexts $E_k(p)$ and $E_{k'}(p)$ are different. Moreover, the positions of the flipped bits in the block are random.

While confusion is concerned with the relationship between the key and the ciphertext, *diffusion* is concerned with the *relationship between the plaintext and the corresponding ciphertext*. Diffusion holds if the statistics of a block of the plaintext is irretrievably dissipated or scattered across the block of its ciphertext. Thus, changing a single bit in a block of the plaintext will have the effect of changing each bit of the block of ciphertext with probability 0.5. If this were not the case, i.e., if the probability were closer to 0 or 1, the cryptanalyst could derive clues that might help in mounting a known plaintext attack.

Practically speaking, a strong substitution function enhances confusion while transposition is used to enhance diffusion. To get the benefit of both confusion and diffusion, both substitutions and transpositions are combined to create *product ciphers*, which is the principal design template for contemporary symmetric block ciphers. A generic product cipher is introduced in the next chapter.

4.4.2 Block Ciphers and Stream Ciphers

With block ciphers, the plaintext is split into fixed size chunks called *blocks*, and each block is encrypted separately. Typically all blocks in the plaintext are encrypted using the same key. Block ciphers include DES, AES, RSA, and ECC.

Block sizes used in secret key cryptography are usually smaller – 64 bits in DES and 128 bits in AES. The block size in RSA is much larger – 768 or more bits, while the block size in ECC is about 200 bits. If two blocks of plaintext within a message are identical, their corresponding ciphertexts are identical. This statement, however, is only partially true. In the next two chapters, we will study ways in which two identical blocks of plaintext are transformed into different blocks of ciphertext.

Unlike block ciphers, stream ciphers typically operate on bits. The one-time pad is an example of a stream cipher. Practical stream ciphers typically generate a *pseudo-random keystream* as a function of a fixed length key and a per-message bit string. The key is known to both the sender and the receiver. The per-message string could be a message sequence number. Alternatively, it could be a random number generated by the sender and transmitted to the receiver along with the encrypted message. The ciphertext is itself obtained by performing an ⊕ operation between the plaintext and the keystream.

An example of a stream cipher is RC4 used in the wireless LAN protocol, IEEE 802.11. Stream ciphers are usually faster than block ciphers and use less complicated circuits. However, RC4 and some other stream ciphers have been shown to be vulnerable to attack. Their use has not been as widespread as block ciphers.

SELECTED REFERENCES

A comprehensive reference for cryptographic algorithms and protocols is [SCHN96]. A more rigorous treatment is found in [MENE01] and [STIN05].

OBJECTIVE-TYPE QUESTIONS

4.1 Kerckhoff's Principle states that the secrecy of a cryptosystem should be a consequence of
- (a) the secrecy of the encryption algorithm
- (b) secrecy of the decryption algorithm
- [✓ (c) secrecy of the encryption key]
- (d) secrecy of the decryption key

4.2 Secret key cryptography is synonymous with
- (a) symmetric key cryptography
- (b) asymmetric key cryptography
- (c) private key cryptography
- (d) quantum cryptography

4.3 To encrypt a message from Alka to Brijesh using public key cryptography, the following is needed:
- (a) Alka's private key
- (b) Alka's public key
- (c) Brijesh' private key
- (d) Brijesh' public key

4.4 Assuming the same key is used, two occurrences of the same plaintext character are encrypted as identical output symbols in which of the following
- (a) Caesar cipher
- (b) Vigenere cipher
- (c) Hill cipher
- (d) One-time pad

4.5 The one-time pad is susceptible to a
- (a) known ciphertext attack
- (b) known plaintext attack
- (c) chosen plaintext attack
- (d) none of the above

4.6 A product cipher is constructed using a combination of
- (a) symmetric and asymmetric ciphers
- (b) substitution and transposition ciphers
- (c) monoalphabetic and polyalphabetic ciphers
- (d) stream and block ciphers

4.7 For the same key, a single bit change in a block of plaintext should result in
- (a) a change in exactly half the bits in the block of ciphertext
- (b) a change in half the bits in the block of ciphertext (on average)
- (c) a change in most of the bits in the block of ciphertext
- (d) a change in a region of ciphertext different from the affected region of plaintext

EXERCISES

4.1 Design known plaintext attacks to obtain the key used in the
- (a) Vigenere cipher
- (b) Hill cipher

4.2 Consider a Hill cipher with $m = 3$ (block size = 3) with key K shown below

$$\begin{pmatrix} 25 & 3 & 7 \\ 5 & 9 & 21 \\ 11 & 8 & 13 \end{pmatrix}$$

(a) What is the ciphertext corresponding to the plaintext = (V O W)?
(b) What is the plaintext corresponding to the ciphertext = (T Q X)?

4.3 The inverse of the key matrix, K in the Hill cipher should exist, otherwise it will not be possible to decrypt a given block of ciphertext. Show that this translates to the following necessary condition.

$$\gcd(\det K, 26) = 1$$

det K in the above equation is the determinant of the key matrix, K. All addition and multiplication operations in computing the determinant are modulo 26.

4.4 Which of the two operations – substitution or transposition – causes diffusion? Which causes confusion? Explain your answer.

4.5 A block of plaintext arranged in a matrix in row-major form (i.e., row by row) is shown below on the left. It is encrypted by performing a sequence of row transpositions, followed by column transpositions followed again by row transpositions. The encrypted block appears on the right.

$$\begin{pmatrix} s & e & c & u & r \\ e & y & o & u & r \\ n & e & t & w & o \\ r & k & n & o & w \end{pmatrix} \rightarrow \begin{pmatrix} w & n & r & k & o \\ r & c & s & e & u \\ r & o & e & y & u \\ o & t & n & e & w \end{pmatrix}$$

What is the sequence of transpositions that were performed?

4.6 Consider a block of plaintext arranged in a matrix in row-major form (i.e., row by row). Can *any* permutation (re-arrangement) of the elements in the block be realized *only* by some sequence of row and/or column transpositions (possibly interleaved)? If yes, give an informal but convincing proof. If no, provide a counter-example.

4.7 What is the number of possible substitution functions that may be defined on a *b*-bit block? (Assume that a *b*-bit block of plaintext results in a *c*-bit block of ciphertext.) What is the number of transposition functions?

4.8 A cryptosystem has the following set of plaintexts, ciphertexts, and keys:

$$\mathcal{P} = \{0, 1, 2, 3\},\ \mathcal{C} = \{A, B, C, D\} \text{ and } \mathcal{K} = \{K_0, K_1, K_2, K_3\}.$$

The probabilities of the various plaintexts are

$$\Pr(0) = 0.3,\ \Pr(1) = 0.25,\ \Pr(2) = 0.10,\ \Pr(3) = 0.35$$

The usage probabilities of the keys are

$$\Pr(K_0) = 0.2,\ \Pr(K_1) = 0.35\ ,\ \Pr(K_2) = 0.25\ ,\ \Pr(K_3) = 0.2.$$

The decryption function is defined in the table below [for example, $E_{K1}(3) = A$].

	0	1	2	3
K_0	B	A	D	C
K_1	D	B	C	A
K_2	A	C	D	B
K_3	C	A	B	D

(a) Compute the conditional probability, Pr(1|C).
(b) What is the probability of occurrence of the ciphertext, A?

(c) A cryptosystem preserves *perfect secrecy* if and only if

$$(\forall p \in P)(\forall c \in C)\ [\Pr(p|c) = \Pr(p)]$$

Does the cryptosystem as defined provide perfect secrecy? If not, why not?
Are there any necessary conditions on the decryption function (represented by the table) for perfect secrecy? If so, what are they?
Are the above conditions sufficient? Explain.

4.9 Does a one-time pad achieve perfect secrecy? Why or why not?

4.10 State one advantage of a block cipher vis-à-vis a stream cipher.
State one drawback of a stream cipher vis-à-vis a block cipher.

4.11 Could you think of an application for which a block cipher is more appropriate and an application where a stream cipher is more appropriate? Justify your answers.

ANSWERS TO OBJECTIVE-TYPE QUESTIONS

4.1 (d), also (c) in the case of secret key cryptography

4.2 (a)	4.3 (d)	4.4 (a)	4.5 (d)
4.6 (b)	4.7 (b)		

Chapter 5

Secret Key Cryptography

There are two types of secret key ciphers – stream ciphers and block ciphers. In this chapter, we focus on block ciphers. We first introduce the product cipher. This provides a template upon which most modern day secret key ciphers are based. We then study the Data Encryption Standard (DES), which is one of the most widely used block ciphers for secret key cryptography. A new standard for secret key cryptography is the Advanced Encryption Standard. A study of this is deferred to Chapter 9. After studying DES, we explore various modes of operation of block ciphers. Applications of secret key ciphers are then discussed. Finally, we examine attacks launched on block ciphers. In particular, we focus on linear cryptanalysis.

[illegible] n bit [illegible]

5.1 PRODUCT CIPHERS

In Chapter 4, elementary substitution and transposition (or permutation) ciphers were introduced. Modern day secret-key ciphers are typically synthesized using the *Substitution Box* (S-Box) and the *Permutation Box* (P-Box) as described in this section.

An S-box is a device that takes as input a (binary) string of length m and returns a (binary) string[1] of length n. While it is often the case that $m = n$, this need not always be so. In DES, for example, $m > n$. An S-box is easily implemented using a *table* (or array) of 2^m rows with each row containing an n-bit value. The input to the S-box is used to index the table which returns the n-bit output of the S-Box.

A P-Box performs a permutation or *re-arrangement* of the bits in the input. A permutation is more restrictive than a substitution. For example, the number of zeros in the output of the P-Box is equal to the number of zeros in its input while an S-box imposes no such restriction.

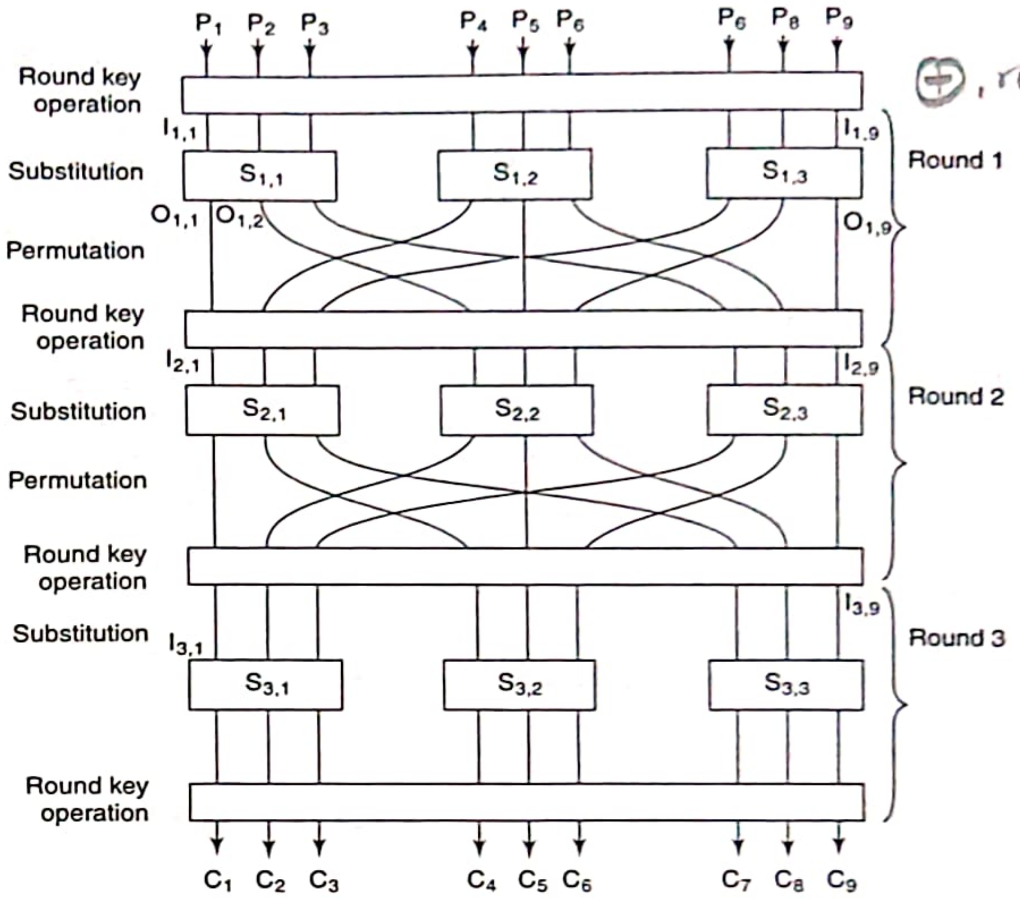
A P-Box or S-box by itself is not sufficiently powerful to create a secure cipher. However, it has been shown that by *cascading* P-Boxes and S-Boxes alternately, the strength of a cipher can be greatly increased. Such a cipher is referred to as a *product cipher*. There are actually three operations that take place in sequence as shown in Fig. 5.1:

(1) an operation involving a function of the encryption key

(2) a substitution and

(3) a permutation

[1]For simplicity, we work with binary strings for both input and output though, in general, the strings could be alphanumeric, decimal, etc.



$\oplus$, round key.

Figure 5.1 *Three-round SPN network*

These operations are repeated over many *rounds* or iterations. Of the three operations, the first is the only one that involves the encryption key. It is usually an $\oplus$ of the input to this step and the "round" key. Each *round key* is a function of the bits in the encryption key.

As mentioned earlier, the S-box is usually implemented as a table. If the block size of the cipher is b, the size of the table that implements a $b \times b$ S-box is $b \times 2^b$ bits. Thus, the table size increases exponentially with the number of inputs. As an example, for $b = 64$, the size of the table is 2^{70} bits which is a thousand billion billion bits!

To save table space, a single S-box is broken into multiple S-boxes as shown in each round of Fig. 5.1. If s is the number of S-boxes, the number of inputs to each S-box is b/s. Each S-box is now implemented using a table of size $(b/s)2^{b/s}$ bits. Thus, the total size of all the S-boxes is $b \times 2^{b/s}$ bits. For a block size of 64, the use of eight S-boxes (each with 8 inputs) would bring down the storage requirements to about 16,000 bits.

What is the contribution of the S-boxes? Assuming they are well-designed, their main contribution is to inject *non-linearity* into the design of the cipher. Non-linearity implies the absence of a linear relationship between any subset of bits in the plaintext, cipher text, and key. We return to this subject in greater detail in Section 5.6.

Finally, the third step in each round or iteration is a permutation. A P-Box re-orders the inputs that it receives. By so doing, it diffuses or spreads contiguous bits of the input across the entire block. Without the P-Box, the first *b/s* bits of the output would be a function of the first *b/s* bits of the input, the second *b/s* bits of the output would be a function of the second *b/s* bits of the input and so on. Without P-Boxes, a single bit change in the plaintext would have only local effect leaving most bits unchanged. The absence of diffusion makes cryptanalysis of a cipher much easier.

The secret key cipher should be designed so that, for the same key, a single bit change in the plaintext should invert each bit in the ciphertext with probability 0.5. Likewise, for the same plaintext, a single bit change in the key should invert each bit in the ciphertext with probability 0.5.

5.2 DES CONSTRUCTION

In this section, we study the construction of DES. DES is the successor to a cipher called Lucifer designed by cryptographers at IBM in the 1960's. It was first published in March 1975 and was chosen by the U.S. National Bureau of Standards or NBS (later re-named National Institute of Standards and Technology or NIST) as the standard cipher for secret key cryptography in January 1977.

5.2.1 Fiestel Structure

The DES block size is 64 bits. DES uses either 56 or 128 bit keys. A single block of plaintext is transformed into ciphertext after passing through the following stages [Fig. 5.2(a)]:

- an initial permutation .
- 16 *rounds* of a given function
- a 32-bit left-right swap and
- a final permutation

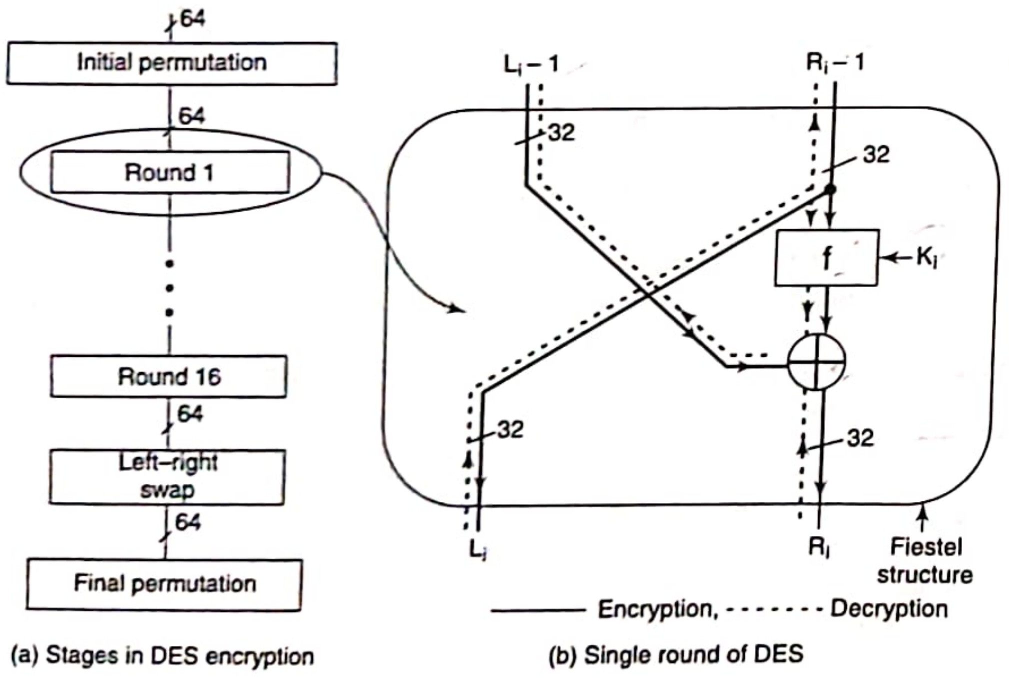


Figure 5.2 *DES operations*

Each of the 16 rounds is functionally identical. Such an *iterative design* is economical from the perspective of chip area (in hardware implementations) or code size (in software implementations). The structure of each DES round is explained below.

Let L_{i-1} and R_{i-1} be the left and right halves of the input to round *i*. As shown in Fig. 5.2(b)

$$L_i = R_{i-1} \tag{5.1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \tag{5.2}$$

The function *f* is applied at each round and is referred to as the "round" function. Each round uses a round key, which is one of the inputs to *f*. Each round key is derived from the DES key.

The process of *decryption* involves obtaining L_{i-1} and R_{i-1} from L_i and R_i. Execution proceeds from bottom to top and is summarized by the following equations derived from Eqs (5.1) and (5.2):

$$R_{i-1} = L_i$$

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

Note that decryption too involves computing $f(\ )$, not $f^{-1}(\ )$! The implication of this fact is that the function *f* does not even have to be invertible [see dashed lines in Fig. 5.2(b)]. The structure of such a cipher is attributed to Horst Feistel (one of the key designers of DES). A cipher that has such a structure is referred to as a *Feistel cipher*.

5.2.2 Round Function

A round function [Fig. 5.2(b)] involves four operations:

- Expansion
- $\oplus$ with the round key
- Substitution
- Permutation

The input to the round function is R_{i-1}, a 32-bit quantity [Fig. 5.2(b)]. This is first expanded into 48 bits by repeating some bits and interchanging their positions. The 48-bit quantity is then $\oplus$ed with the round key, K_i (which is different for each round). The bits in a round key are a function of the bits in the original 56-bit key.

The result of the $\oplus$ operation is divided into eight 6-bit chunks. Each chunk is substituted by a 4-bit chunk. A total of 8 different S-boxes provide the eight substitutions. An S-box is implemented using a 4 × 16 array. Each row of the array is a permutation of the numbers 0 through 15. Two bits of the *i*-th chunk serve as a row index into the *i*-th table (Fig. 5.3) and the remaining four bits serve as a column index. The output of the S-box is simply the 4-bit string pointed to by the row and column indices.

Much thought has gone into different aspects of the design of DES. The number of rounds (why 16 and not less?), the S-boxes, and the permutations have each been chosen to thwart a number of potential attacks of a sophisticated nature, which the designers of DES seemed to have anticipated.

5.3 MODES OF OPERATION

DES and other block-oriented secret key schemes can be employed in different modes – each with its own advantages and disadvantages. The most straightforward approach is to encrypt each block separately (Fig. 5.4). This mode of operation is referred to as *ECB* or *Electronic Code Book Mode*.

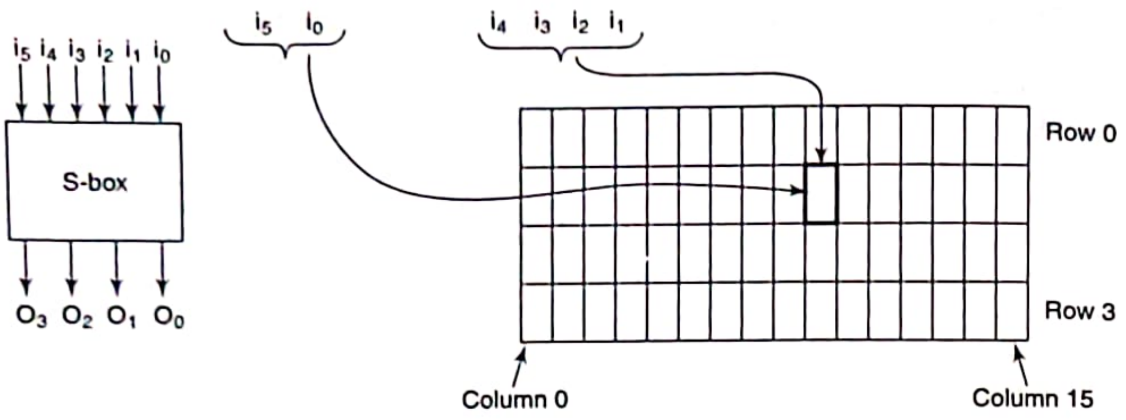


Figure 5.3 *S-Box implementation using table lookup*

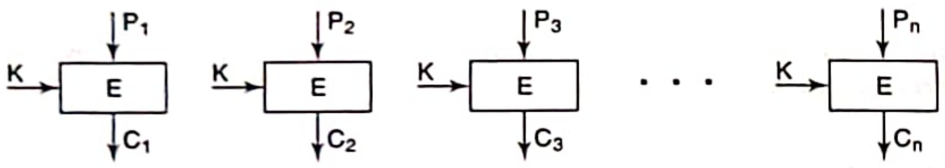


Figure 5.4 *ECB mode of operation*

One drawback of the ECB mode is that identical blocks of plaintext will be encrypted as identical blocks of ciphertext. Suppose, for example, an eavesdropper notices that blocks 23 and 95 of the ciphertext are equal. If, in addition, he/she knows (or can guess) the value of block 23 of plaintext, he/she can then deduce the content of block 95 of the plaintext. Another drawback of this mode of operation is that a re-ordering of blocks by an attacker, while in transit, will not be detected by the receiver.

The above drawbacks can be overcome by combining the previous block of ciphertext with the current block of plaintext before performing the encryption. In Fig. 5.5(a), C_{i-1} is ⊕ed with P_i and then encrypted to produce C_i. This has the effect of "randomizing" the input to the encryption box so that two identical blocks of plaintext will, with high probability, map to different ciphertext values. This mode of operation is referred to as *Cipher Block Chaining* (CBC).

To jump-start the process of encryption, an *initialization vector*, IV, is used (Fig. 5.5). This is usually a random *b*-bit string, where *b* is the block size. The initialization vector should be known to (or agreed upon by) both sender and receiver. To perform decryption, reverse the arrows in Fig. 5.5(a) and also replace the E-box by a D-box (decryption). Decryption is shown in Fig. 5.5(b).

One drawback of the CBC mode is that, if a block of ciphertext is received in error, that block and the next will both be decrypted to incorrect blocks of plaintext. Thus, the error propagates to the next block unlike in the case of the ECB mode. A feature of both ECB and CBC is that a block is encrypted in its entirety. In some applications, it may be desirable to encrypt and transmit only part of a block. For example, a sender might produce bytes intermittently. In real-time applications, it may be unacceptable to wait for an entire block of plaintext to be produced and only then encrypted and sent. The next mode addresses this issue.

Instead of transmitting ciphertext blocks of size, *b*, *Cipher Feedback Mode* (CFB), encrypts and transmits sub-blocks of size *s*. This mode uses a shift register of size *b* (Fig. 5.6). The steps in encryption are the following:

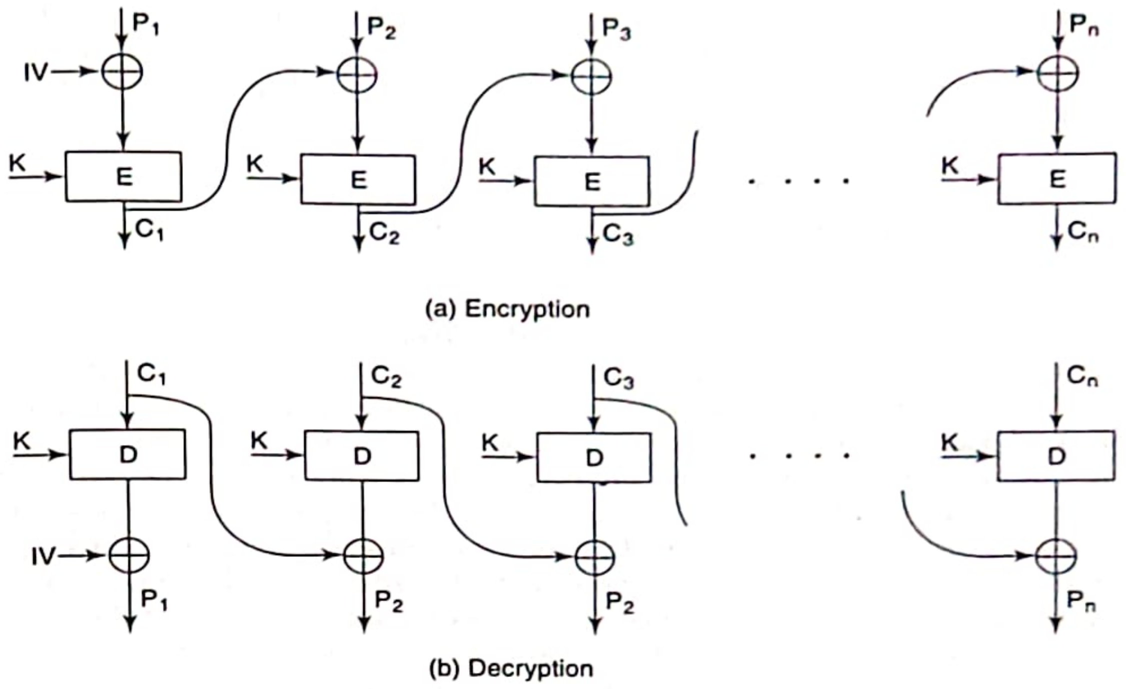


Figure 5.5 *CBC mode of operation*

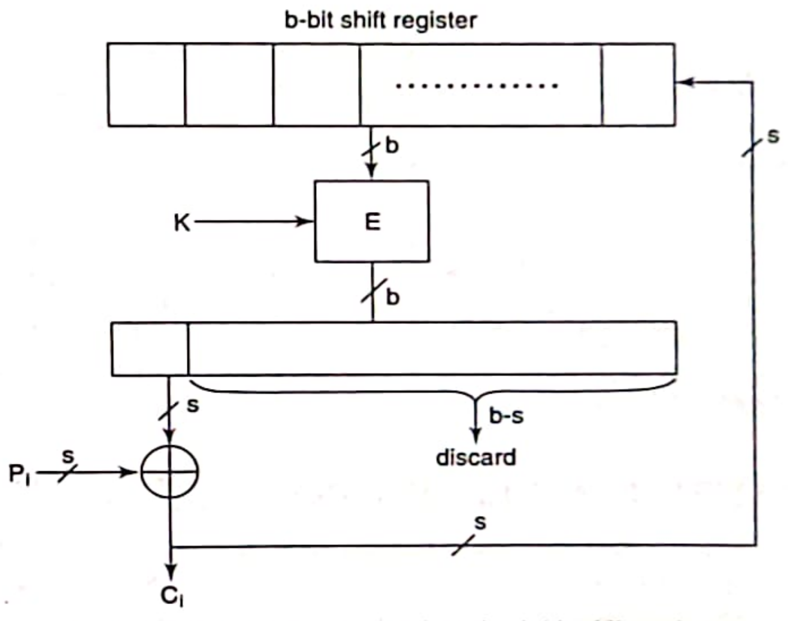

b-bit shift register
b
s
K
E
b
s
b-s
discard
Pi
s
s
Ci
Note: Initially the IV is loaded into the b-bit shift register

Figure 5.6 *CFB mode of operation*

(1) The shift register is initially loaded with the initialization vector (IV).
(2) The contents of the shift register are encrypted with the cipher key.

(3) The most significant s bits of the b-bit output are then ⊕ ed with s bits of plaintext to create the next s-bit chunk of ciphertext. The remaining $b - s$ bits, of the output are discarded.
(4) The shift register is shifted left by s bits. In the process, the leftmost s bits are lost.
(5) Then, the s bits of ciphertext are inserted into the vacated (rightmost s bits of the shift register).

Decryption is easily derived from the scheme for encryption. Note that, unlike in the other modes, decryption is performed using the encryption function. One drawback of this mode is that the number of encryption operations per b-bit block is now b/s as compared to just a single encryption in the ECB or CBC modes.

A mode that is used in some security protocols is the *Counter Mode*. A b-bit counter is initialized to a *random value*. This value is encrypted with the secret key and then ⊕ ed with the first block of plaintext. The counter is then incremented, the incremented value is encrypted and ⊕ ed with the next block of plaintext to create the next block of ciphertext and so on (Fig. 5.7). Care should be taken to ensure that the initial value of the counter is truly random and are not re-used. Otherwise, secret key encryption using this mode may be compromised by a known plaintext attack. There are several advantages of the Counter Mode of operation. First, blocks of plaintext can be encrypted in *any order* unlike CBC and CFB modes. Also, multiple plaintext blocks can be processed in *parallel*, thereby speeding up encryption. Indeed the encryption of the counter values can be performed even before the plaintext is available for encryption. Once the plaintext is available, computing the ciphertext is only a matter of performing the inexpensive ⊕ of encrypted counter values (which were pre-computed) and the blocks of plaintext.

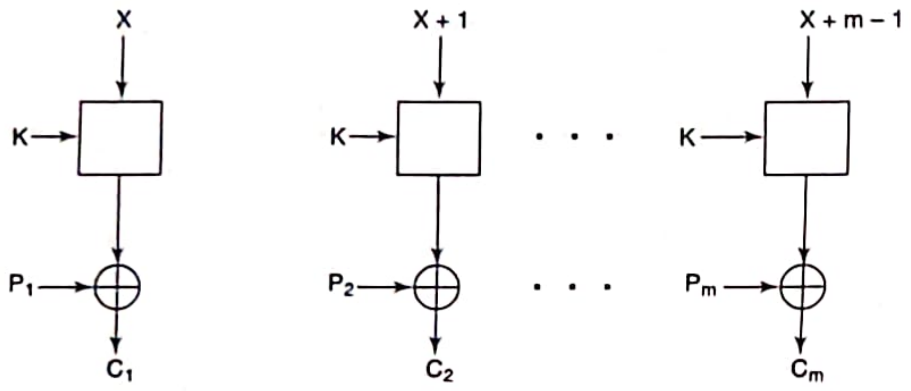


Figure 5.7 *Counter mode of operation*

5.4 MAC AND OTHER APPLICATIONS

The principal use of secret key cryptographic schemes such as DES is to provide message *confidentiality*. It is also used to protect the privacy of stored documents by encrypting them with a key known to the owner of the document.

Secret key cryptography is also used for authentication. There are two types of authentication – *entity authentication* and *message authentication*. Entity authentication involves making sure that the party you are establishing connection with is indeed the party you intend to communicate with. We study the protocols for entity authentication using secret and public key cryptography in Chapter 11.

Message authentication involves making sure that *each* message received is indeed from the party that has participated in the establishment of that connection/session. Message authentication is handled on a per-message basis. Message authentication and message integrity can both be provided by a keyed checksum called a *MAC* or *Message Authentication Code*.

A MAC is a fixed-length, *one-way function* of both a *message and a secret* shared by the sender and receiver. For each message to be sent, the sender computes a MAC, which is appended to the message. On receipt of the message and MAC, the recipient computes the same function on the message and secret shared with the sender. This process is called *MAC verification* and succeeds if the MAC computed by the recipient and that received by him/her match. Necessary properties of a MAC include the following:

(1) If even a single bit of the message is corrupted, the MAC for the new message should be quite different from the MAC computed on the original message.
(2) It should be computationally infeasible to deduce the secret knowing one or more <message, MAC> pairs.
(3) It should be computationally infeasible to generate a MAC for any new message without knowledge of the secret even if an attacker has knowledge of one or more <message, MAC> pairs.

Property 1 is used to verify the integrity of a received message. Properties 2 and 3 imply that if verification succeeds, the message recipient can be sure that the received MAC was created by none other than the party with whom the recipient shares the secret key. We conclude by studying the use of DES in generating a MAC.

Imagine encrypting each message block using *DES in CBC mode* with the encryption key being the secret shared by sender and receiver (Fig. 5.5). The IV used here is agreed upon beforehand or it could be chosen by the sender and sent to the receiver together with the message and the MAC. All blocks of the ciphertext except for the last are discarded. The last block is used as the MAC for that message. Note that the MAC is a function of the entire message and the shared secret. Properties 1–3 above are guaranteed if a secret key cipher such as DES is employed.

5.5 ATTACKS

One attack on DES is a *known plaintext attack*. The attacker gathers a number of plaintext–ciphertext pairs obtained by the use of the same key. With 56-bit DES, for example, each of the 2^{56} possible keys are applied to a block of plaintext to determine which key creates the correct ciphertext. This is repeated for other plaintext–ciphertext pairs using the keys that were successful in all previous iterations until a single consistently successful key is found.

To investigate the insecurity of 56-bit DES, RSA Security Lab sponsored a series of contests to find, in minimum time, the key used to encrypt a bunch of text. In the third such contest held in January 1999, a combined team comprising the Electronic Frontier Foundation (EFF) and Distributed.net (a non-profit organization) broke all previous records.

EFF bagged the top slot in an earlier contest in July 1998 using *DES Cracker* – a $250,000 machine built with 1500 chips specifically designed for cracking DES. The January 1999 contest saw DES cracker being used in conjunction with about *10,000 PCs* connected to the Internet. The idle CPU power of these widely dispersed PCs was harnessed to run a special client in the background. A number of key servers were used to distribute the keys. This combined effort was able to obtain the DES key in only *22 hours* (down from 56 hours using the DES Cracker alone in the earlier contest).

Given the vulnerability of 56-bit DES, is it possible to enhance the security of DES by doing a double encryption with two different keys?

In the case of *Double DES*, the ciphertext, $\mathcal{C}_i$, is obtained from the plaintext, $\mathcal{P}_i$, using

$$\mathcal{C}_i = \mathrm{E}_{k1}(\mathrm{E}_{k2}\ (\mathcal{P}_i)) \qquad i = 1, 2, \ldots n \tag{5.3}$$

Since the number of key combinations is $2^{56} \times 2^{56}$, it appears that a brute force attack on Double DES would take time proportional to 2^{112}. However, the following attack based on known plaintext–ciphertext pairs shows that the time to attack Double DES is still proportional to 2^{56}!

Suppose the following known plaintext–ciphertext pairs are available:
$(\mathcal{P}_1, \mathcal{C}_1), (\mathcal{P}_2, \mathcal{C}_2), \ldots (\mathcal{P}_n, \mathcal{C}_n)$. From Eq. 5.3, it follows that

$$\mathrm{D}_{k1}(\mathcal{C}_i) = \mathrm{E}_{k2}(\mathcal{P}_i) \qquad i = 1, 2, \ldots n \tag{5.4}$$

We first consider the pair $(\mathcal{P}_1, \mathcal{C}_1)$. We create two tables. The first table lists all pairs k', $\mathrm{D}_{k'}(\mathcal{C}_1)$ for all possible 2^{56} values of k'. We also create a table that lists all pairs k'', $\mathrm{E}_{k''}(\mathcal{P}_1)$. The first table is sorted on the column, $\mathrm{D}_{k'}(\mathcal{C}_1)$ and the second table is sorted on $\mathrm{E}_{k''}(\mathcal{P}_1)$. We scan both tables looking for a match of values in columns $\mathrm{D}_{k'}(\mathcal{C}_1)$ and $\mathrm{E}_{k''}(\mathcal{P}_1)$. When a match does occur, we note that the corresponding values in the columns k' and k'' could correspond to the two encryption keys used to obtain $\mathcal{C}_1$ from $\mathcal{P}_1$ in Eq. (5.3). There may be several such matches, in which case the corresponding k' and k'' values are potential candidates for the encryption keys.

We repeat this experiment for the other known plaintext–ciphertext pairs in turn, each time considering only key pairs that satisfy Eq. (5.4) for all previous plaintext–ciphertext pairs. The actual encryption keys are the ones that appear as candidates for each of the known plaintext–ciphertext pairs.

Creating these tables and scanning them take time proportional to the number of distinct 56-bit keys or 2^{56}. Sorting the table of size n takes $O(n \log n)$ time. Since $\log n$ grows much more slowly compared to n, the $\log n$ term is usually ignored. So the *time* to find the encryption keys in Double DES is *still proportional to* 2^{56} as in Single DES! To enhance the security of DES, a commonly used strategy is the use of Triple DES or 3-DES.

3-DES employs three successive encryptions with three distinct keys. In practice, only two distinct keys are used to obtain the ciphertext as follows

$$\mathcal{C} = \mathrm{E}_{k1}\ (\mathrm{D}_{k2}\ (\mathrm{E}_{k1}\ (\mathcal{P}))) \tag{5.5}$$

The time complexity of a brute force attack in this case is proportional to $2^{56 \cdot 2} = 2^{112}$.

Differential cryptanalysis [BIHA91], pioneered by Biham and Shamir, is a chosen plaintext attack. It examines the transformations that two related plaintexts undergo during encryption by each key from a set of candidate keys. Based on this, each key is assigned a probability of being the real key. Again, two plaintexts are chosen that differ in the same bit positions as the earlier chosen plaintexts and the process is repeated. Biham and Shamir [BIHA93] show that the key can be obtained by inspecting about 2^{47} plaintext–ciphertext pairs – an improvement over 2^{56} in a brute-force attack.

Linear cryptanalysis, pioneered by Matsui [MATS93], tries to find approximate linear relationships between key bits, bits of plaintext, and bits of ciphertext. It requires about 2^{43} plaintext–ciphertext pairs. In practice, however, these schemes using cryptanalytic techniques do not perform as well as the embarrassingly parallel brute-force attacks described earlier. For example, a linear cryptanalytic attack on 46-bit DES took 10 days and it took an additional 40 days to generate the known plaintext–ciphertext pairs [MATS93]! Nevertheless, a solid understanding of cryptanalytic attacks provides valuable insights into the design of a secure secret key cipher.

5.6 LINEAR CRYPTANALYSIS

5.6.1 Preliminaries

The basic idea in linear cryptanalysis is to identify *linear relationships* between bits in the plaintext, ciphertext, and key that hold "universally" for every choice of plaintext and key. Formally, we attempt to identify equations of the following form:

$$P_{i_1} \ldots \oplus P_{i_p} \oplus C_{i_1} \ldots \oplus C_{i_c} \oplus K_{i_1} \oplus \ldots \oplus K_{i_k} = 0 \tag{5.6}$$

OR

$$P_{i_1} \oplus \ldots \oplus P_{i_p} \oplus C_{i_1} \ldots \oplus C_{i_c} \oplus K_{i_1} \oplus \ldots \oplus K_{i_k} = 1 \tag{5.7}$$

Here, $C_{i_1} \ldots C_{i_c}$, $P_{i_1} \ldots P_{i_p}$, and $K_{i_1} \ldots K_{i_k}$ are, respectively, the subsets of bits in the ciphertext, C_i obtained by encrypting plaintext, $\mathcal{P}_i$ using key, K_i.

For well-designed ciphers, there should be NO such relationships! The existence of one or more such relationships provides the attacker with valuable clues in connection with some of the bits of the key. It is the S-boxes that make it hard to establish such relationships since S-boxes are the only *non-linear* component in the cipher. For an ideal S-box, all equations of the form Eq. (5.6) or Eq. (5.7) should hold with probability ½. In practice, however, we may be able to find bits in the plaintext, ciphertext, and key that satisfy Eq. (5.6) with probability *sufficiently different* from ½. The more the deviation from ½, the higher is the chance of a successful cryptanalytic attack. We call the deviation of the probability from ½ as the *bias* and denote it as β.

There are two steps in carrying out linear cryptanalysis:

- The first step is identifying linear relationships similar to that described by Eq. (5.6).
- The second step employs known plaintext–ciphertext pairs to obtain all or part of the key.

We describe these steps in the next two subsections.

5.6.2 Step 1: Identifying Linear Relationships

To obtain relationships expressed in Eqs (5.6) or (5.7) that hold with a high probability bias, we repeatedly combine linear expressions of the following form:

$$X_1 = P_{i_1} \oplus \ldots \oplus P_{i_p} \oplus I_{i_1} \oplus \ldots \oplus I_{i_l} \oplus K'_{i_1} \oplus \ldots \oplus K'_{i_k}$$

with expressions of the form

$$X_2 = I_{j_1} \oplus \ldots \oplus I_{j_l} \oplus K'_{j_1} \oplus \ldots \oplus K'_{j_K}$$

to create expressions of the form $X_1 \oplus X_2$.

Here $P_{i_1} \ldots P_{i_p}$ are bits of the plaintext, $I_{i_1} \ldots I_{i_l}$ and $I_{j_1} \ldots I_{j_l}$ are *inputs to S-boxes*. $K'_{i_1} \ldots K'_{i_k}$ and $K'_{j_1} \ldots K'_{j_K}$ are bits *of the round keys*.

In the treatment of linear cryptanalysis, we often need to estimate the bias of $X_1 \oplus X_2$ given the biases of the random, Boolean variables, X_1 and X_2.

Let X_1 and X_2 be two *independent*, binary random variables. Prob$[X_1 = 0]$ and Prob$[X_2 = 0]$ are known. Let β_1 and β_2 be the biases of X_1 and X_2, respectively.

$$\begin{aligned}
\text{Prob } [X_1 \oplus X_2 = 0] &= \text{Prob } [(X_1 = 0 \text{ and } X_2 = 0) \text{ or } (X_1 = 1 \text{ and } X_2 = 1)] \\
&= (½ + \beta_1)\,(½ + \beta_2) + (½ - \beta_1)\,(½ - \beta_2) \\
&= ½ + 2\beta_1\beta_2
\end{aligned} \tag{5.8}$$

Hence the bias of $X_1 \oplus X_2$ is $2\beta_1\beta_2$. This result is a special case of what is called the *Piling-up Lemma*.

We next illustrate the first step in linear cryptanalysis with an example of a toy three-round cipher. As shown in Fig. 5.8, the block size of this cipher is 9 bits. Each round involves a substitution followed by a permutation and a round key addition step. In addition, there is a round key addition step preceding the first round.

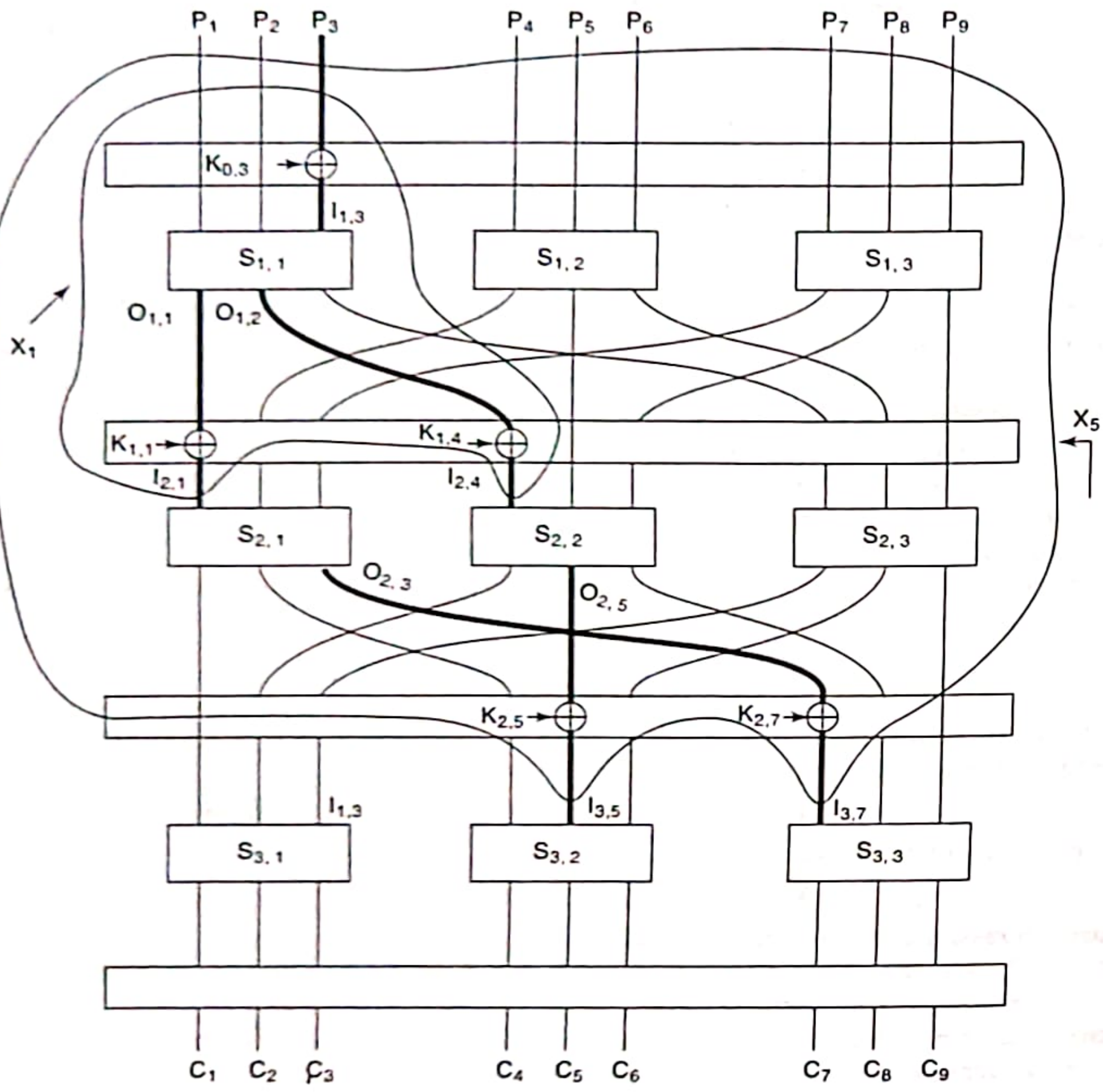


Figure 5.8 *Deriving a high-bias linear expression*

We use the notation $I_{j,1}\ I_{j,2}\ \ldots\ I_{j,9}$ to denote the inputs to the S-boxes in round j. Likewise, $O_{j,1}$ $O_{j,2}\ \ldots\ O_{j,9}$ are the outputs of the S-boxes in round j. Finally, $K_{j,1}\ K_{j,2}\ \ldots\ K_{j,9}$ are the bits of the round key in round j. Note that round key addition performs an $\oplus$ of each bit of the input with the corresponding bit of the round key bit. So, for example, $P_3 \oplus K_{0,3} = I_{1,3}$ and $O_{2,3} \oplus K_{2,7} = I_{3,7}$.

Each of the S-boxes in Fig. 5.8 realizes the same substitution shown in Table 5.1.

Table 5.1 *S-Box definition and bias computation*

Substitution $I_1 I_2 I_3$ | $O_1 O_2 O_3$	$I_3 \oplus O_1 \oplus O_2 \overset{?}{=} 0$	$I_1 \oplus I_2 \oplus O_2 \overset{?}{=} 0$	$I_2 \oplus I_3 \oplus O_1 \oplus O_3 \overset{?}{=} 0$
0 0 0 → 0 0 0	✓	✓	✓
0 0 1 → 0 1 0	✓	×	×
0 1 0 → 0 0 1	✓	×	✓
0 1 1 → 1 0 0	✓	×	×
1 0 0 → 1 1 1	✓	✓	✓
1 0 1 → 1 0 1	✓	×	×
1 1 0 → 1 1 0	✓	×	✓
1 1 1 → 0 1 1	✓	×	×
	Bias = 8/8 – ½ = ½	Bias = 2/8 – ½ = – ¼	Bias = 4/8 – ½ = 0

Example 5.1

We attempt to obtain a random variable that has a "high" bias. The random variable will be a linear combination of subsets of

- plaintext bits,
- bits in the round keys, and
- input bits to the S-Boxes in Round 3.

It is helpful to start by considering random variables comprising different combinations of inputs and outputs to/from a *single S-Box*. Two such combinations are $I_3 \oplus O_1 \oplus O_2$ and $I_1 \oplus I_2 \oplus O_2$. Given the substitution function appearing in the leftmost column of Table 5.1, their computed values (0 or 1) are shown in each row of Table 5.1. Accordingly, their biases (towards 0) are ½ and –¼, respectively. Since the first of these expressions has a high bias, we use it to construct a random variable. In terms of the inputs and outputs of $S_{1,1}$ (leftmost box of Round 1), our first random variable, X_1 is

$$X_1 = I_{1,3} \oplus O_{1,1} \oplus O_{1,2} \tag{5.9}$$

It can be expressed in terms of a plaintext bit, P_3 and inputs to the S-Boxes in Round 2 ($I_{2,1}$ and $I_{2,4}$). Since $P_3 \oplus K_{0,3} = I_{1,3}$, $O_{1,1} \oplus K_{1,1} = I_{2,1}$ and $O_{1,2} \oplus K_{1,4} = I_{2,4}$ (see Fig. 5.8), it follows that

$$X_1 = P_3 \oplus I_{2,1} \oplus I_{2,4} \oplus K_{0,3} \oplus K_{1,1} \oplus K_{1,4} \tag{5.10}$$

We next attempt to identify a high-bias random variable, X_2, that combines $I_{2,1}$ and some outputs of $S_{2,1}$ (leftmost S-Box in Round 2). One such combination is $I_{2,1}$ and $O_{2,3}$. From Table 5.1, it may be verified that X_2 has a bias = ¼. From Fig. 5.8, $O_{2,3} \oplus K_{2,7} = I_{3,7}$. This allows X_2 to be expressed as

$$X_2 = I_{2,1} \oplus K_{2,7} \oplus I_{3,7} \tag{5.11}$$

Let $X_3 = X_1 \oplus X_2$. Since the bias of X_1 is ½ and that of X_2 is ¼, the bias of X_3 = 2(½)(¼) = ¼ from the Piling-up Lemma. Substituting from Eqs 5.10 and 5.11, we get

$$X_3 = P_3 \oplus I_{2,4} \oplus I_{3,7} \oplus K_{0,3} \oplus K_{1,1} \oplus K_{1,4} \oplus K_{2,7} \tag{5.12}$$

We next search for a high-bias random variable, X_4, that combines $I_{2,4}$ and some outputs of $S_{2,2}$.

One such combination is $I_{2,4}$ and $O_{2,5}$ with bias = ¼. Once again, from Fig. 5.8, $O_{2,5} \oplus K_{2,5} = I_{3,5}$. This allows X_4 to be expressed as

$$X_4 = I_{2,4} \oplus K_{2,5} \oplus I_{3,5} \tag{5.13}$$

Let $X_5 = X_3 \oplus X_4$. The bias of X_5 is 2(¼)(¼) = 1/8. Also, from Eqs 5.12 and 5.13,

$$X_5 = P_3 \oplus I_{3,5} \oplus I_{3,7} \oplus K_{0,3} \oplus K_{1,1} \oplus K_{1,4} \oplus K_{2,5} \oplus K_{2,7} \tag{5.14}$$

We now summarize what has been accomplished in the preceding example. We built random variables involving the inputs and outputs of three S-Boxes, $S_{1,1}$, $S_{2,1}$, and $S_{2,2}$. We combined these random variables into a single one, X_5 with a bias = 1/8. As desired, X_5 involves bits of the plaintext, round keys, and inputs to the S-Boxes in Round 3. Figure 5.8 shows the step-wise combining of these variables. In general then, we derive a biased linear expression, denoted by X defined by

$$X = P_{i_1} \oplus \ldots \oplus P_{i_p} \oplus I_{i_1} \oplus \ldots \oplus I_{i_l} \oplus \mathcal{K} \tag{5.15}$$

where $P_{i_1} \ldots P_{i_p}$ are bits of the plaintext, $I_{i_1} \ldots I_{i_l}$ are inputs to S-boxes of the last stage and $\mathcal{K}$ represents the sum of selected bits in the round keys of different stages (except for the last stage).

The next task is to use known (plaintext, ciphertext) pairs to deduce certain bits in the secret key.

5.6.3 Step 2: Using Known Plaintext–Ciphertext Pairs

In this step, we focus attention on obtaining bits of the key used in the last round. We use the following strategy:

Consider all S-Boxes in the last stage that have at least one input included in the linear expression derived in Step 1 [Eq. (5.15)]. In the context of Example 5.1, the relevant S-Boxes are $S_{3,2}$ and $S_{3,3}$. They have inputs $I_{3,5}$ and $I_{3,7}$, respectively [see Eq. (5.14) and Fig. (5.8)]. We then focus on the following:

- The outputs of the relevant S-Boxes in the last stage (for Example 5.1, these are $O_{3,4}$, $O_{3,5}$, $O_{3,6}$, and $O_{3,7}$, $O_{3,8}$, $O_{3,9}$).
- The bits of the last round key that will be ⊕ ed with the above S-Box outputs (for Example 5.1, these are $K_{3,4}$, $K_{3,5}$, $K_{3,6}$, $K_{3,7}$, $K_{3,8}$, and $K_{3,9}$).
- The bits of the ciphertext affected by the above ⊕ operation (for Example 5.1, these are C_4, C_5, C_6, C_7, C_8, and C_9).

We refer to each of the above sets of S-Box outputs, round key bits, and ciphertext bits as the *relevant* variables. Given the set of (plaintext, ciphertext) pairs, we then run the procedure outlined in Fig. 5.9.

Suppose the number of known plaintext–ciphertext pairs is π. For an incorrect choice of final round key bits, $\mathcal{K}'$, the value in score[$\mathcal{K}'$] would be expected to be close to $\pi/2$. However, for the correct choice of $\mathcal{K}'$, the value of score[$\mathcal{K}'$] would be quite different from $\pi/2$. The extent of deviation from $\pi/2$ is a function of the bias of the linear expression in Eq. (5.15). This assumes that π is sufficiently large to begin with. Matsui [MATS93] has shown that, for the attack to succeed, the number of known (plaintext, ciphertext) pairs should be around $\frac{1}{\beta^2}$.

We summarize Step 2 by completing Example 5.1 of the last subsection.

```
Let $\mathcal{K}'$ denote the sequence of relevant bits in the key for the last round.
// This procedure attempts to find the true value of $\mathcal{K}'$.
// score[ ] is used later to determine the most likely value of $\mathcal{K}'$.
for (each plaintext–ciphertext pair, $(\mathcal{P}_i, \mathcal{C}_i)$)  {
        extract the relevant bits of $\mathcal{C}_i$
        for ( $\mathcal{K}'$ = 00...0  to  11...1 )   {
                perform the $\oplus$ of the relevant bits of $\mathcal{C}_i$ and $\mathcal{K}'$
                        to obtain the output values of the relevant S-Boxes
                        in the last round
                use the definition of the inverse S-Box to obtain
                        the inputs to the relevant S-Boxes
                extract the values of the inputs to the S-Boxes that also
                        appear in the linear expression [Eq. (5.15)]
                extract the values in $\mathcal{P}_i$ that also
                        appear in the linear expression [Eq. (5.15)]
                obtain the value of $\mathcal{K}$ by setting $X = 0$ in Eq. (5.15)
                        and by substituting from $\mathcal{P}_i$ for the
                        plaintext bits in Eq. (5.15)
                        and substituting for relevant inputs to S-boxes
                if ($\mathcal{K}$ = 1 )  score[$\mathcal{K}'$] $\leftarrow$ score[$\mathcal{K}'$] + 1
        }
}
```

Figure 5.9 *Procedure for identifying bits of the round key*

Example 5.2

The linear expression derived in Example 5.1 (with bias $\beta = 1/8$) is

$$X_5 = P_3 \oplus I_{3,5} \oplus I_{3,7} \oplus K_{0,3} \oplus K_{1,1} \oplus K_{1,4} \oplus K_{2,5} \oplus K_{2,7}$$

$$= P_3 \oplus I_{3,5} \oplus I_{3,7} \oplus \mathcal{K}$$

The relevant S-Boxes in the last round are $S_{3,2}$ and $S_{3,3}$. The relevant bits of the last round key, $\mathcal{K}'$, are $K_{3,4}$, $K_{3,5}$, $K_{3,6}$, $K_{3,7}$, $K_{3,8}$, and $K_{3,9}$. The relevant bits of the ciphertext are C_4, C_5, C_6, C_7, C_8, and C_9. For each given (plaintext–ciphertext) pair, we iterate overall 2^6 possible values of $\mathcal{K}'$. We set $X_5 = 0$, so $\mathcal{K}$ is computed from

$$\mathcal{K} = P_3 \oplus I_{3,5} \oplus I_{3,7}$$

We update score[ ] and choose the most likely value of $\mathcal{K}'$ based on the final values in score[ ] as described above.

SELECTED REFERENCES

[FIPS 46-3] defines the mathematical steps in DES required to transform plaintext into ciphertext. NIST special publication [NIST800-38] is an excellent source for various cipher modes. [BIHA91] and [BIHA93] are pioneering works on differential cryptanalysis, while [MATS93] introduces linear cryptanalysis. [HEYS01] is a well-written tutorial on both types of cryptanalysis. Material on the EFF DES Cracker may be found in [EFF].

OBJECTIVE-TYPE QUESTIONS

5.1 A Substitution Box provides
- (a) diffusion only
- (b) confusion only
- (c) both diffusion and confusion
- (d) neither confusion nor diffusion

5.2 The number of rounds in 56-bit DES and 128-bit DES are respectively
- (a) 12 and 12
- (b) 12 and 16
- (c) 16 and 16
- (d) 16 and 20

5.3 The block size in 56-bit DES and 128-bit DES are respectively
- (a) 64 and 64 bits
- (b) 64 and 128
- (c) 128 and 128
- (d) 128 and 256

5.4 A MAC provides
- (a) message integrity
- (b) message confidentiality
- (c) message authentication
- (d) message non-repudiation

5.5 A MAC may be implemented using DES in
- (a) ECB mode
- (b) CBC mode
- (c) CFB mode
- (d) counter mode

5.6 If the biases of three binary random variables, X_1, X_2, and X_3 are respectively – ¼, ¼, and 1/8, then the bias of $X_1 \oplus X_2 \oplus X_3$ is
- (a) –1/32
- (b) 1/8
- (c) 1/16
- (d) –1/16

5.7 Attacks based on linear cryptanalysis use
- (a) chosen ciphertext only
- (b) chosen plaintext
- (c) known ciphertext only
- (d) known plaintext–ciphertext pairs

EXERCISES

5.1 Consider a function whose range and domain are both 64-bit binary strings. f is defined as: $f(x) = E_k(x)$, $0 \leq x \leq 2^{64} - 1$. Here, E denotes 56-bit DES encryption. Is this mapping onto [i.e., for an arbitrary 64-bit string y, is it always true that there exists a plaintext, p, such that $E_k(x) = y$]? Explain your answer.

5.2 The function f used in each round of DES is invertible. Yes or No? Explain.

5.3 Do you agree with the statement that an increase in key size of 1 bit doubles the security of DES? Justify your answer.

5.4 (a) Let K be a 56-bit DES key that encrypts p_1 as c_1, i.e., $E_K(p_1) = c_1$. What is the probability that there exists another key, K', such that $E_{K'}(p_1) = c_1$?

(b) Let K be a 56-bit DES key that encrypts p_1 as c_1 and p_2 as c_2, i.e. $E_K(p_1) = c_1$ and $E_K(p_2) = c_2$. What is the probability that there exists another key, K', such that $E_{K'}(p_1) = c_1$ and $E_{K'}(p_2) = c_2$?

5.5 3-DES, defined as $E_{K1}(D_{K2}(E_{K1}(p)))$ was proposed to enhance the security of single DES. Why do you think 3-DES is defined this way? Would its security be enhanced if it were defined as $E_{K1}(E_{K2}(E_{K3}(p)))$?

5.6 Refer to Figure 5.5 which shows message encryption in CBC mode. Suppose the positions of the XOR gate and the encryption box were swapped, what effect would this have on security?

5.7 Which of the following is/are true and why? The Initialization Vector (IV) in CBC mode should be

- a constant known only to sender and receiver
- a non-secret constant
- a random variable known only to sender and receiver
- a non-secret random variable

5.8 State one advantage of the counter mode over CBC mode and one possible drawback of the counter mode.

5.9 A single bit error occurs in exactly one block of ciphertext during transmission. How will this affect the recovery of plaintext in each of the following modes?

ECB, CBC, CFB, Counter.

5.10 Comment on the choice of the following substitution function for an $n \times n$ S-Box in an SPN.

$$S(x) = (x + i) \bmod n, \ 1 < i < n$$

5.11 Consider a three-round SPN as shown in Fig. 5.1. Write a linear expression involving bits in the plaintext, round keys and inputs to the third stage S-Boxes with maximum possible bias. Highlight the paths involving the variables in the figure. Assume that the S-boxes are identical and that the mapping of S-box inputs to outputs is

0 0 0 → 0 0 0
0 0 1 → 0 1 0
0 1 0 → 0 0 1
0 1 1 → 1 0 0
1 0 0 → 1 1 1
1 0 1 → 1 0 1
1 1 0 → 1 1 0
1 1 1 → 0 1 1

5.12 Write a program to implement 56-bit DES.

5.13 Write a program to cryptanalyze *i*-stage DES (i = 1 or 2, for example). Specifically, you should write two programs.

The first program should generate a random key and a small number of plaintexts. Using the generated key, it should compute the ciphertext for each generated plaintext using *i*-stage DES.

Given only the plaintext–ciphertext pairs, the second program should then try to obtain the key.

ANSWERS TO OBJECTIVE-TYPE QUESTIONS

5.1 (b)	5.2 (c)	5.3 (a)	5.4 (a)(c)
5.5 (b)	5.6 (a)	5.7 (d)	