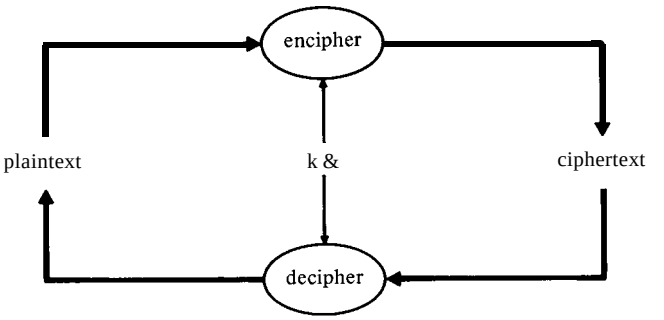
1 Introduction

1.1 CRYPTOGRAPHY

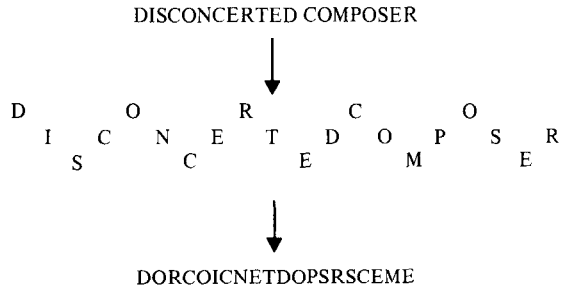
Cryptography is the science and study of secret writing. A **cipher** is a secret method of writing, whereby **plaintext** (or **cleartext)** is transformed into **ciphertext** (sometimes called a **cryptogram).** The process of transforming plaintext into ciphertext is called **encipherment** or **encryption;** the reverse process of transforming ciphertext into plaintext is called **decipherment** or **decryption.** Both encipherment and decipherment are controlled by a cryptographic **key** or keys (see Figure 1.1).

There are two basic types of ciphers: transpositions and substitutions. **Transposition ciphers** rearrange bits or characters in the data. With a "rail-fence" cipher, for example, the letters of a plaintext message are written down in a

FIGURE 1.1 Secret writing.



pattern resembling a rail fence, and then removed by rows. The following illustrates this pattern:



The key to the cipher is given by the depth of the fence, which in this example is 3.

Substitution ciphers replace bits, characters, or blocks of characters with substitutes. A simple type of substitution cipher shifts each letter in the English alphabet forward by K positions (shifts past Z cycle back to A); K is the key to the cipher. The cipher is often called a **Caesar cipher** because Julius Caesar used it with $K = 3$. The following illustrates Caesar's method:

```
IMPATIENT WAITER
    ↓
LPSDWLHQW ZDLWHU.
```

A **code** is a special type of substitution cipher that uses a "code book" as the key. Plaintext words or phrases are entered into the code book together with their ciphertext substitutes, as shown next:

Word	Code
BAKER	1701
FRETTING	5603
GUITARIST	4008
LOAFING	3790
.	.
.	.
.	.

```
LOAFING BAKER
     ↓
3790 1701
```

The term code is sometimes used to refer to any type of cipher.

In computer applications, transposition is usually combined with substitution. The Data Encryption Standard (DES), for example, enciphers 64-bit blocks using a combination of transposition and substitution (see Chapter 2).

Cryptanalysis is the science and study of methods of breaking ciphers. A cipher is **breakable** if it is possible to determine the plaintext or key from the ciphertext, or to determine the key from plaintext-ciphertext pairs. There are three basic methods of attack: ciphertext-only, known-plaintext, and chosen-plaintext.

Under a **ciphertext-only attack**, a cryptanalyst must determine the key solely from intercepted ciphertext, though the method of encryption, the plaintext language, the subject matter of the ciphertext, and certain probable words may be

known. For example, a message describing the location of a buried treasure would probably contain words such as BURIED, TREASURE, NORTH, TURN, RIGHT, MILES.

Under a **known-plaintext attack**, a cryptanalyst knows some plaintext-ciphertext pairs. As an example, suppose an enciphered message transmitted from a user's terminal to the computer is intercepted by a cryptanalyst who knows that the message begins with a standard header such as "LOGIN". As another example, the cryptanalyst may know that the *Department* field of a particular record contains the ciphertext for *Physics;* indeed, the cryptanalyst may know the *Department* field of every record in the database. In some cases, knowledge of probable words allows a close approximation to a known-plaintext attack. Encrypted programs are particularly vulnerable because of the regular appearance of keywords—e.g. **begin**, **end**, **var**, **procedure**, **if**, **then**. Even if the exact position of encrypted keywords is unknown, a cryptanalyst may be able to make reasonable guesses about them. Ciphers today are usually considered acceptable only if they can withstand a known-plaintext attack under the assumption that the cryptanalyst has an arbitrary amount of plaintext-ciphertext pairs.

Under a **chosen-plaintext attack**, a cryptanalyst is able to acquire the ciphertext corresponding to selected plaintext. This is the most favorable case for the cryptanalyst. A database system may be particularly vulnerable to this type of attack if users can insert elements into the database, and then observe the changes in the stored ciphertext. Bayer and Metzger [Baye76] call this the **planted record problem**.

Public-key systems (defined in Section 1.3) have introduced a fourth kind of attack: a **chosen-ciphertext attack**. Although the plaintext is not likely to be intelligible, the cryptanalyst may be able to use it to deduce the key.

A cipher is **unconditionally secure** if, no matter how much ciphertext is intercepted, there is not enough information in the ciphertext to determine the plaintext uniquely. We shall give a formal definition of an unconditionally secure cipher in Section 1.4. With one exception, all ciphers are breakable given unlimited resources, so we are more interested in ciphers that are computationally infeasible to break. A cipher is **computationally secure**, or **strong**, if it cannot be broken by systematic analysis with available resources.

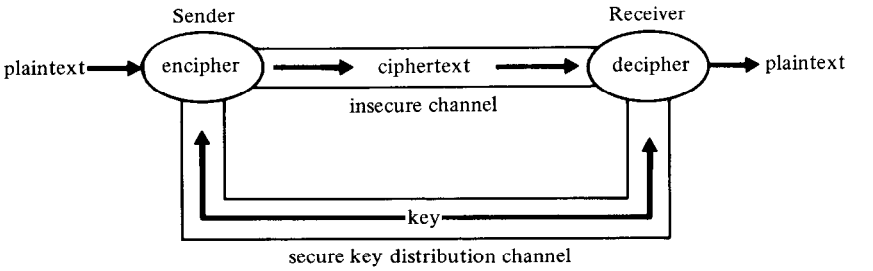
The branch of knowledge embodying both cryptography and cryptanalysis is called **cryptology**.

1.2 DATA SECURITY

Classical cryptography provided secrecy for information sent over channels where eavesdropping and message interception was possible. The sender selected a cipher and encryption key, and either gave it directly to the receiver or else sent it indirectly over a slow but secure channel (typically a trusted courier). Messages and replies were transmitted over the insecure channel in ciphertext (see Figure 1.2). Classical encryption schemes are described in Chapter 2.

Modern cryptography protects data transmitted over high-speed electronic

FIGURE 1.2 Classical information channel.

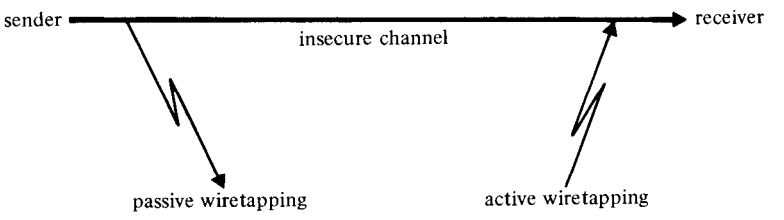


lines or stored in computer systems. There are two principal objectives: **secrecy** (or **privacy**), to prevent the unauthorized disclosure of data; and **authenticity** or **integrity**), to prevent the unauthorized modification of data.

Information transmitted over electronic lines is vulnerable to passive wiretapping, which threatens secrecy, and to active wiretapping, which threatens authenticity (see Figure 1.3). **Passive wiretapping (eavesdropping)** refers to the interception of messages, usually without detection. Although it is normally used to disclose message contents, in computer networks it can also be used to monitor traffic flow through the network to determine who is communicating with whom. Protection against disclosure of message contents is provided by enciphering transformations, which are described in Chapter 2, and by the cryptographic techniques described in Chapter 3. Protection against traffic flow analysis is provided by controlling the endpoints of encryption; this is discussed in Chapter 3.

Active wiretapping (tampering) refers to deliberate modifications made to the message stream. This can be for the purpose of making arbitrary changes to a message, or of replacing data in a message with replays of data from earlier messages (e.g., replacing the amount field of a transaction "CREDIT SMITH'S ACCOUNT WITH $10" with the amount field of an earlier transaction "CREDIT JONES'S ACCOUNT WITH $5000"). It can be for the purpose of injecting false messages, injecting replays of previous messages (e.g., to repeat a credit transaction), or deleting messages (e.g., to prevent a transaction "DEDUCT $1000 FROM SMITH'S ACCOUNT"). Encryption protects against message

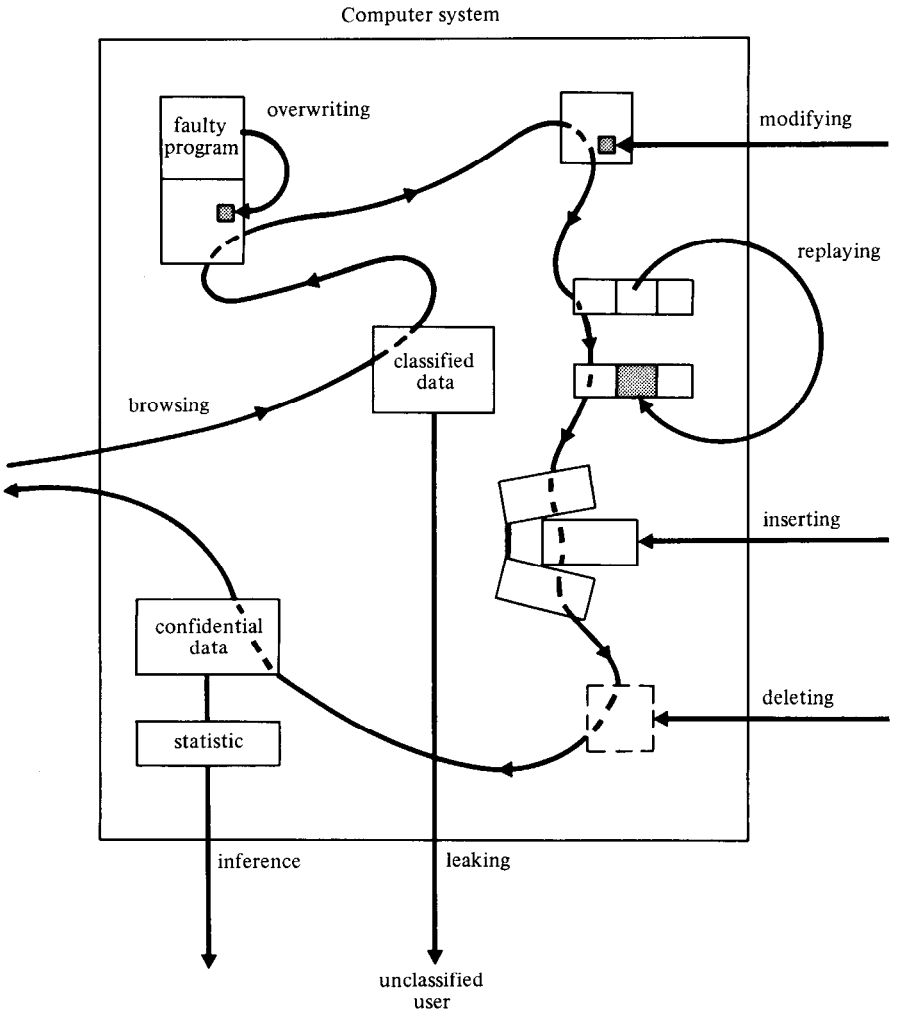
FIGURE 1.3 Threats to secure communication.



modification and injection of false messages by making it infeasible for an opponent to create ciphertext that deciphers into meaningful plaintext. Note, however, that whereas it can be used to detect message modification, it cannot prevent it.

Encryption alone does not protect against replay, because an opponent could simply replay previous ciphertext. Cryptographic techniques for protecting against this problem are discussed in Chapter 3. Although encryption cannot prevent message deletion, the cryptographic techniques discussed in Chapter 3 can detect deletions of blocks or characters within a message stream. Deletion of entire messages can be detected with communication protocols that require message acknowledgment.

FIGURE 1.4 Threats to data stored in computer systems.



Data in computer systems is vulnerable to similar threats (see Figure 1.4). Threats to secrecy include browsing, leakage, and inference. **Browsing** refers to searching through main memory or secondary storage for information (e.g., confidential data or proprietary software programs). It is similar to eavesdropping on communication channels, but there are two important differences. On the one hand, information stored in computer systems has a longer lifetime; in this sense, browsing poses a more serious threat than eavesdropping. On the other hand, information transmitted over electronic lines is vulnerable to tapping even when access to the system is denied. Browsing is possible only if the user has access to the system and to unauthorized regions of memory. Access controls, described in Chapter 4, can prevent this.

Cryptography protects against browsing by making the information unintelligible. It can supplement access controls and is especially useful for protecting data on tapes and discs which, if stolen, can no longer be protected by the system. Cryptography cannot, however, protect data from disclosure while it is being processed in the clear. Access controls are needed for this purpose, and these controls must include procedures that clear memory between use to ensure that confidential data is not inadvertently exposed. If access is not controlled, encrypted data can also be vulnerable to **ciphertext searching** (e.g., finding employees making identical salaries by searching for records with identical ciphertext salaries); cryptographic solutions to this problem are described in Chapter 3.

Leakage refers to the transmission of data to unauthorized users by processes with legitimate access to the data. A compiler, for example, could leak a proprietary software program while it is being compiled. An income tax program could leak confidential information about a user. A file editor could leak classified military data to a user without a security clearance. Cryptography and access controls must be supplemented with information flow controls, discussed in Chapter 5, to control information dissemination.

Inference refers to the deduction of confidential data about a particular individual by correlating released statistics about groups of individuals. For example, if Smith is the only non-Ph.D. faculty member in a Physics department, Smith's salary can be deduced by correlating the average salary of all faculty in the department with the average salary of all Ph.D. faculty in the department. Although cryptography and access controls can protect the data records from browsing, they do not provide a mathematical framework for determining which statistics can be released without disclosing sensitive data. Inference controls, discussed in Chapter 6, address this problem.

Threats to authenticity include tampering and accidental destruction. **Tampering** with data in computer systems is analogous to active wiretapping on communication channels, but differs from it in the same ways browsing differs from passive wiretapping. Like active wiretapping, tampering can be for the purpose of making arbitrary changes to data (e.g., changing the *Salary* field of an employee record from $20,000 to $25,000). It can be for the purpose of replaying data stored previously in a record (e.g., to restore a previous balance in an accounting record), or replaying data stored in some other record (e.g., to make the *Salary* field of an employee record the same as that of a higher paid employee). It can also be for the

purpose of overwriting data with nonsense (e.g., overwriting a cryptographic key so that encrypted data becomes inaccessible). Finally, it can be for the purpose of inserting records (e.g., adding a dummy employee record to the payroll file) or deleting files or records (e.g., to remove a bad credit report). Cryptographic techniques can help protect against these threats by making it possible to detect false or replayed ciphertext. But it cannot prevent them. Access controls are essential for the reliable operation of the system. Backup is vital for recovery.

Accidental destruction refers to the unintentional overwriting or deletion of data. Unintentional overwriting is caused by faulty software (e.g., because an array subscript is out-of-range). Cryptography cannot protect against this threat. Access controls, implemented in language processors and in hardware, provide error confinement by preventing programs from writing into the memory regions of other programs or into system tables. Unintentional deletion is caused by software or hardware failure (e.g., a disk head crash), and by user mistakes (e.g., inadvertently deleting lines of a file during an editing session). Backup is needed to recover from accidental as well as deliberate destruction. Many text editors have automatic backup facilities so that an earlier version of a file is easily recovered; some have facilities for undoing each editing command.

Computer systems are vulnerable to another problem: **masquerading**. If an intruder can gain access to a system under another user's account, then the intruder can access the user's data files and all other information permitted to the user. Similarly, if a program can spoof legitimate users logging into the system into believing that they are conversing with the system, the program might be able to obtain confidential information from these users (e.g., their login passwords). Protection against masquerading requires that the system and user be able to mutually authenticate each other. Such strategies that use encrypted passwords are described in Chapter 3. "Digital signatures" provide a more general means of authenticating users or processes; they are introduced in Section 1.3.3.

Data security is the science and study of methods of protecting data in computer and communications systems. It embodies the four kinds of controls studied in this book: cryptographic controls, access controls, information flow controls, and inference controls. It also embodies procedures for backup and recovery.

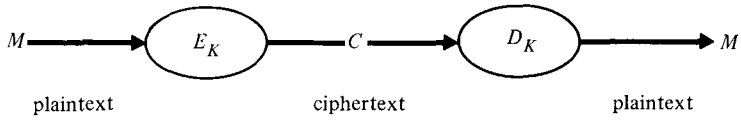
1.3 CRYPTOGRAPHIC SYSTEMS

This section describes the general requirements of all cryptographic systems, the specific properties of public-key encryption, and digital signatures.

A **cryptographic system** (or **cryptosystem** for short) has five components:

1. A **plaintext message space**, $\mathcal{M}$.
2. A **ciphertext message space**, $\mathcal{C}$.
3. A **key space**, $\mathcal{K}$.
4. A family of **enciphering transformations**, E_K: $\mathcal{M} \rightarrow \mathcal{C}$, where $K \in \mathcal{K}$.
5. A family of **deciphering transformations**, D_K: $\mathcal{C} \rightarrow \mathcal{M}$, where $K \in \mathcal{K}$.

FIGURE 1.5 Cryptographic system.



Each **enciphering transformation** E_K is defined by an **enciphering algorithm** E, which is common to every transformation in the family, and a **key** K, which distinguishes it from the other transformations. Similarly, each **deciphering transformation** D_K is defined by a **deciphering algorithm** D and a key K. For a given K, D_K is the inverse of E_K; that is, $D_K(E_K(M)) = M$ for every plaintext message M. In a given cryptographic system, the transformations E_K and D_K are described by parameters derived from K (or directly by K). The set of parameters describing E_K is called the **enciphering key,** and the set of parameters describing D_K the **deciphering key**. Figure 1.5 illustrates the enciphering and deciphering of data.

Cryptosystems must satisfy three general requirements:

1. The enciphering and deciphering transformations must be efficient for all keys.
2. The system must be easy to use.
3. The security of the system should depend only on the secrecy of the keys and not on the secrecy of the algorithms E or D.

Requirement (1) is essential for computer applications; data is usually enciphered and deciphered at the time of transmission, and these operations must not be bottlenecks. Requirement (2) implies it must be easy for the cryptographer to find a key with an invertible transformation. Requirement (3) implies the enciphering and deciphering algorithms must be inherently strong; that is, it should not be possible to break a cipher simply by knowing the method of encipherment. This requirement is needed because the algorithms may be in the public domain or known to a cryptanalyst, whence knowing K reveals E_K and D_K. Note, however, the converse need not hold; that is, knowing E_K or D_K need not reveal K. This is because the enciphering key describing E_K or the deciphering key describing D_K could be derived from K by a one-way (irreversible) transformation (see Section 1.5.3). This technique is used in public-key systems (see Section 1.3). We shall assume the algorithms E and D are public knowledge.

There are specific requirements for secrecy and authenticity. Secrecy requires that a cryptanalyst not be able to determine plaintext data from intercepted ciphertext. Formally, there are two requirements:

Secrecy requirements

1. It should be computationally infeasible for a cryptanalyst to systematically determine the deciphering transformation D_K from intercepted ciphertext C, even if the corresponding plaintext M is known.

2. It should be computationally infeasible for a cryptanalyst to systematically determine plaintext M from intercepted ciphertext C.

Requirement (1) ensures that a cryptanalyst cannot systematically determine the deciphering transformation (guessing may be possible). Thus, the cryptanalyst will be unable to decipher C or other ciphertext enciphered under the transformation E_K. Requirement (2) ensures that a cryptanalyst cannot systematically determine plaintext without the deciphering transformation. Both requirements should hold regardless of the length or number of ciphertext messages intercepted.

Secrecy requires only that the transformation D_K (i.e., the deciphering key) be protected. The transformation E_K can be revealed if it does not give away D_K. Figure 1.6 illustrates. The straight line shows the intended flow through the system, while the bent line shows the undesired flow that results from successful attacks.

Data authenticity requires that a cryptanalyst not be able to substitute a false ciphertext C' for a ciphertext C without detection. Formally, the two requirements are:

Authenticity requirements

1. It should be computationally infeasible for a cryptanalyst to systematically determine the enciphering transformation E_K given C, even if the corresponding plaintext M is known.
2. It should be computationally infeasible for a cryptanalyst to systematically find ciphertext C' such that $D_K(C')$ is valid plaintext in the set $\mathcal{M}$.

Requirement (1) ensures that a cryptanalyst cannot systematically determine the enciphering transformation. Thus the cryptanalyst will be unable to encipher a different plaintext message M', and substitute the false ciphertext $C' = E_K(M')$ for C. Requirement (2) ensures that a cryptanalyst cannot find ciphertext C' that deciphers into meaningful plaintext without the enciphering transformation. Numerical data is particularly vulnerable to ciphertext substitution because all values may be meaningful. Both requirements should hold regardless of the amount of ciphertext intercepted.

Authenticity requires only that the transformation E_K (i.e., the enciphering

FIGURE 1.6 Secrecy.

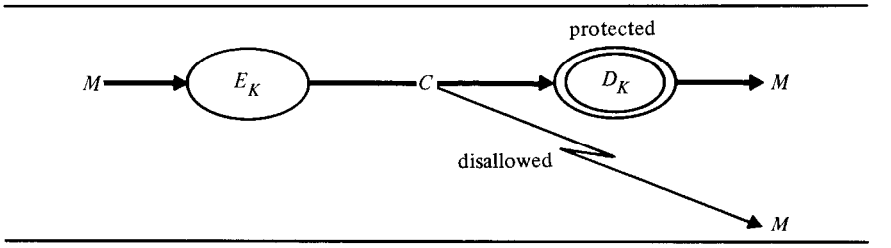
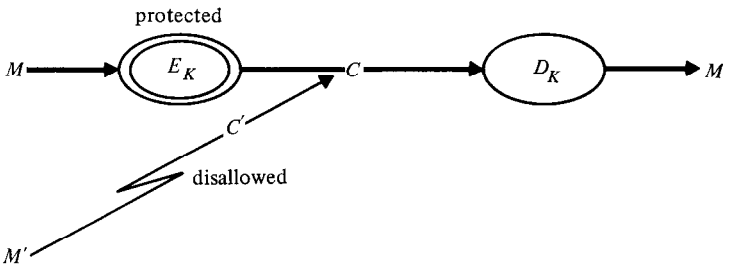


FIGURE 1.7 Authenticity.



key) be protected. The transformation D_K could be revealed if it does not give away E_K. Figure 1.7 illustrates.

Simmons classifies cryptosystems as symmetric (one-key) and asymmetric (two-key) [Simm79]. In **symmetric** or **one-key** cryptosystems the enciphering and deciphering keys are the same (or easily determined from each other). Because we have assumed the general method of encryption is known, this means the transformations E_K and D_K are also easily derived from each other. Thus, if both E_K and D_K are protected, both secrecy and authenticity are achieved. Secrecy cannot be separated from authenticity, however, because making either E_K or D_K available exposes the other. Thus, all the requirements for both secrecy and authenticity must hold in one-key systems.

One-key systems provide an excellent way of enciphering users' private files. Each user A has private transformations E_A and D_A for enciphering and deciphering files (see Figure 1.8). If other users cannot access E_A and D_A, then both the secrecy and authenticity of A's data is assured.

One-key systems also provide an excellent way of protecting information transmitted over computer networks. This is the classical information channel where the sender and receiver share a secret communication key (see Figure 1.2). If both parties are mutually trustworthy, they can be assured of both the secrecy and authenticity of their communications.

Until recently, all cryptosystems were one-key systems. Thus, one-key sys-

FIGURE 1.8 Single-key encryption of private files.

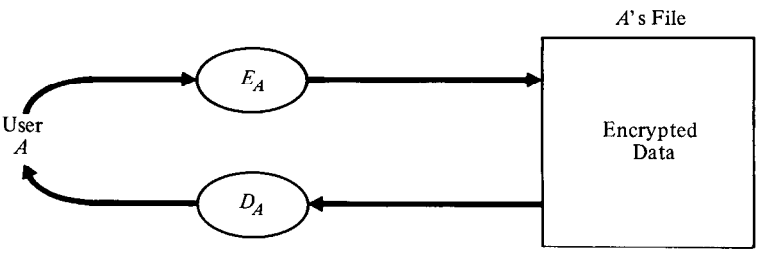
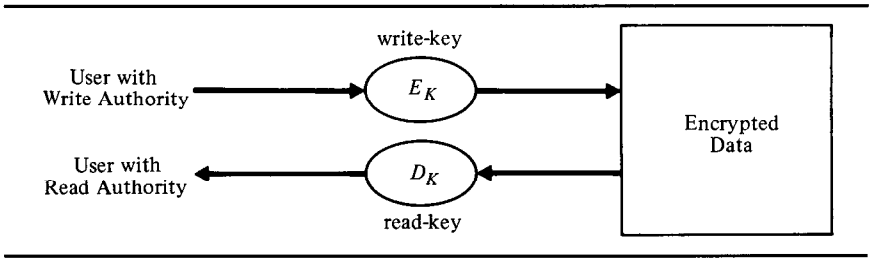


FIGURE 1.9 File encryption with separate Read/Write keys.



tems are also usually referred to as **conventional** (or **classical**) systems. The DES is a conventional system.

In **asymmetric** or **two-key** cryptosystems the enciphering and deciphering keys differ in such a way that at least one key is computationally infeasible to determine from the other. Thus, one of the transformations E_K or D_K can be revealed without endangering the other.

Secrecy and authenticity are provided by protecting the separate transformations—D_K for secrecy, E_K for authenticity. Figure 1.9 illustrates how this principle can be applied to databases, where some users have read-write authority to the database, while other users have read authority only. Users with read-write authority are given both D_K and E_K, so they can decipher data stored in the database or encipher new data to update the database. If E_K cannot be determined from D_K, users with read-only authority can be given D_K, so they can decipher the data but cannot update it. Thus D_K is like a **read-key**, while E_K is like a **write-key** (more precisely, the deciphering key describing D_K is the read-key, and the enciphering key describing E_K the write-key).

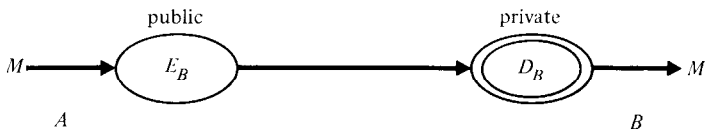
Note that this does not prevent a user with read-only authority (or no access authority) from destroying the data by overwriting the database with nonsense. It only prevents that user from creating valid ciphertext. To protect the data from such destruction, the system must be secured by access controls, so that no user can write into the database without the write-key E_K. The system need not, however, control read access to the data, because the data cannot be deciphered without the read-key D_K.

1.3.1. Public-Key Systems

The concept of two-key cryptosystems was introduced by Diffie and Hellman in 1976 [Diff76]. They proposed a new method of encryption called **public-key encryption**, wherein each user has both a public and private key, and two users can communicate knowing only each other's public keys.

In a public-key system, each user A has a **public enciphering transformation** E_A, which may be registered with a public directory, and a **private deciphering transformation** D_A, which is known only to that user. The private transformation D_A is described by a **private key**, and the public transformation E_A by a **public key**

FIGURE 1.10 Secrecy in public-key system.



derived from the private key by a one-way transformation. It must be computationally infeasible to determine D_A from E_A (or even to find a transformation equivalent to D_A).

In a public-key system, secrecy and authenticity are provided by the separate transformations. Suppose user A wishes to send a message M to another user B. If A knows B's public transformation E_B, A can transmit M to B in secrecy by sending the ciphertext $C = E_B(M)$. On receipt, B deciphers C using B's private transformation D_B, getting

$$D_B(C) = D_B(E_B(M)) = M .$$

(See Figure 1.10.) The preceding scheme does not provide authenticity because any user with access to B's public transformation could substitute another message M' for M by replacing C with $C' = E_B(M')$.

For authenticity, M must be transformed by A's own private transformation D_A. Ignoring secrecy for the moment, A sends $C = D_A(M)$ to B. On receipt, B uses A's public transformation E_A to compute

$$E_A(C) = E_A(D_A(M)) = M .$$

(See Figure 1.11.) Authenticity is provided because only A can apply the transformation D_A. Secrecy is not provided because any user with access to A's public transformation can recover M.

Now, we had previously defined a transformation D_A as a function from the ciphertext space $\mathcal{C}$ to the message space $\mathcal{M}$. To apply D_A to plaintext messages, D_A must instead map $\mathcal{M}$ to $\mathcal{C}$. Furthermore, to restore the original message, E_A must be the inverse of D_A; that is, E_A must be a function from $\mathcal{C}$ to $\mathcal{M}$ such that $E_A(D_A(M)) = M$.

To use a public-key system for both secrecy and authenticity, the ciphertext space $\mathcal{C}$ must be equivalent to the plaintext space $\mathcal{M}$ so that any pair of transformations E_A and D_A can operate on both plaintext and ciphertext messages. Furthermore, both E_A and D_A must be mutual inverses so that $E_A(D_A(M)) = D_A(E_A(M)) = M$. These requirements are summarized in Table 1.1.

FIGURE 1.11 Authenticity in public-key system.

TABLE 1.1 Requirements for public-key transformations.

Secrecy	Authenticity	Both
E_A: $\mathcal{M} \rightarrow \mathcal{C}$	D_A: $\mathcal{M} \rightarrow \mathcal{C}$	E_A: $\mathcal{M} \rightarrow \mathcal{M}$
D_A: $\mathcal{C} \rightarrow \mathcal{M}$	E_A: $\mathcal{C} \rightarrow \mathcal{M}$	D_A: $\mathcal{M} \rightarrow \mathcal{M}$
$D_A(E_A(M)) = M$	$E_A(D_A(M)) = M$	$D_A(E_A(M)) = M$
		$E_A(D_A(M)) = M$

To achieve both secrecy and authenticity, the sender and receiver must each apply two sets of transformations. Suppose A wishes to send a message M to B. First A's private transformation D_A is applied. Then A enciphers the result using B's public enciphering transformation E_B, and transmits the doubly transformed message $C = E_B(D_A(M))$ to B. B recovers M by first applying B's own private deciphering transformation D_B, and then applying A's public transformation E_A to validate its authenticity, getting

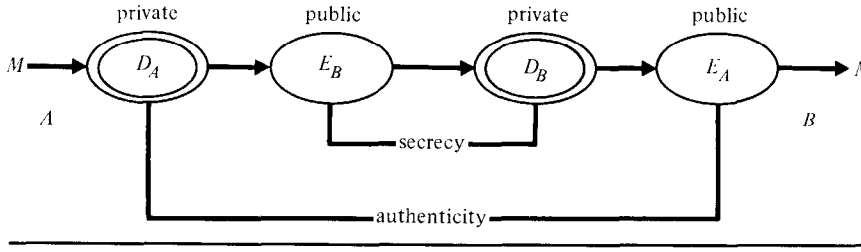
$$\begin{aligned} E_A(D_B(C)) &= E_A\big(D_B\big(E_B(D_A(M))\big)\big) \\ &= E_A(D_A(M)) \\ &= M . \end{aligned}$$

(See Figure 1.12.)

Only one of the public-key encryption methods discussed in Chapter 2 can be used for both secrecy and authenticity. This is the scheme invented by Rivest, Shamir, and Adleman of MIT (referred to as the "RSA scheme"). The RSA scheme is based on the difficulty of factoring large numbers (see Section 2.7). The schemes based on the difficulty of solving "knapsack problems" can be used for either secrecy or authenticity but not both (see Section 2.8). McEliece's [McE178] scheme based on error correcting codes (not discussed in this book) is also restricted to secrecy.

Simmons [Simm81] shows how a public-key authenticity system can be used to verify the identity of individuals seeking entrance to secured areas (computer room, nuclear reactor site, etc.). Each individual permitted to enter the area is given an ID card containing descriptive information such as name and social security number, identifying information such as voiceprint or handprint, and access

FIGURE 1.12 Secrecy and authenticity in public-key system.



control information such as the time of day when entrance is permitted. The information is encrypted under the private key of the central authority issuing the card. The corresponding public key is distributed to all areas where entry is controlled. The individual enters the restricted area through a special facility where the identifying information is taken and checked against the information stored on the individual's card.

1.3.2 Digital Signatures

A **digital signature** is a property private to a user or process that is used for signing messages. Let B be the recipient of a message M signed by A. Then A's signature must satisfy these requirements:

1. B must be able to **validate** A's signature on M.
2. It must be impossible for anyone, including B, to **forge** A's signature.
3. In case A should disavow signing a message M, it must be possible for a judge or third party to **resolve** a dispute arising between A and B.

A digital signature, therefore, establishes **sender authenticity**; it is analogous to an ordinary written signature. By condition (2), it also establishes data authenticity.

Public-key authentication systems provide a simple scheme for implementing digital signatures. Because the transformation D_A is private to A, D_A serves as A's digital signature. The recipient B of a message M signed by A (i.e., transformed by D_A) is assured of both sender and data authenticity. It is impossible for B or anyone else to forge A's signature on another message, and impossible for A to disclaim a signed document (assuming D_A has not been lost or stolen). Because the inverse transformation E_A is public, the receiver B can readily validate the signature, and a judge can settle any disputes arising between A and B. Summarizing,

1. A signs M by computing $C = D_A(M)$.
2. B validates A's signature by checking that $E_A(C)$ restores M.
3. A judge resolves a dispute arising between A and B by checking whether $E_A(C)$ restores M in the same way as B.

Whereas conventional systems such as the DES provide data authenticity, they do not in themselves provide sender authenticity. Because the sender and receiver share the same key, the receiver could forge the sender's signature, and it would be impossible for a judge to settle a dispute.

It is possible to implement digital signatures in conventional systems using a trusted third party S. The following approach was suggested by Merkle [Merk80]. Each user A registers a pair of private transformations E_A and D_A with S, where $E_A(D_A(M)) = M$ for every message M. To send a signed message M to B, A computes $C = D_A(M)$, and transmits C to B. To check the validity of C and obtain M, B sends C to S. S computes $E_A(C) = M$ and returns M to B enciphered under

B's private transformation. (Other methods of implementing digital signatures in conventional systems are described in Rabin [Rabi78], Needham and Schroeder [Need78], Popek and Kline [Pope79], and Smid [Smid79].)

There are difficulties with both the conventional and public-key approach if signature keys are lost or stolen. This problem is addressed in Chapter 3.

There are many applications for digital signatures. For example, if customer *A*'s bank receives an electronic message requesting the withdrawal of $100,000, the bank must be certain the request came from *A*; if *A* later disavows the message, the bank must be able to prove to a third party that the message originated with *A*.

In the preceding example, secrecy is desired as well as authenticity, because the customer would like the transaction to be confidential. In some applications, sender and data authenticity are desirable in the absence of secrecy. Simmons [Simm79] describes a system developed at Sandia Laboratories for nuclear test ban treaty verification where authentication is required but secrecy cannot be tolerated. Each nation is allowed to install a seismic observatory in the other nation (the host) to determine whether it is complying with a requirement to stop all underground testing of nuclear weapons. The observatory transmits the data gathered back to a monitor in the nation owning the observatory. There are three requirements:

1. The monitor must be certain the information reported back has originated from the observatory and has not been tampered with by the host; thus both sender and data authenticity are essential.
2. The host nation must be certain the information channel is not being used for other purposes; thus it must be able to read all messages transmitted from the observatory.
3. Neither the monitor nor the host should be able to create false messages that appear to have originated from the observatory. If a dispute arises between the monitor and host about the authenticity of a message, a third party (e.g., the United Nations or NATO) must be able to resolve the dispute.

All three requirements are satisfied in a public-key authentication system, where the observatory uses a private transformation (unknown even to the host) to sign all messages transmitted to the monitor. Both the monitor and host have access to the corresponding public transformation.

Merkle [Merk80] describes two applications of signatures for software protection. The first involves distributing network software to the individual nodes of a network. If the software is signed, the nodes can check the validity of the software before execution. The second involves running privileged programs in operating systems. The system (preferably hardware) could refuse to execute any program in privileged mode that is not properly signed by a program verifier, making it impossible for someone to substitute a program that could run in privileged mode and wreak havoc in the system. This idea could be extended to all programs, with the system refusing to execute any code that has not been signed

by some authority. Note that these applications do not require a method of resolving disputes. They could, therefore, be implemented in a conventional system, where the sender and receiver share a common key.

1.4 INFORMATION THEORY

In 1949, Shannon [Shan49] provided a theoretical foundation for cryptography based on his fundamental work on information theory [Shan48]. He measured the theoretical secrecy of a cipher by the uncertainty about the plaintext given the received ciphertext. If, no matter how much ciphertext is intercepted, nothing can be learned about the plaintext, the cipher achieves perfect secrecy.

With one exception, all practical ciphers leave some information about the plaintext in the ciphertext. As the length of the ciphertext increases, the uncertainty about the plaintext usually decreases, eventually reaching 0. At this point, there is enough information to determine the plaintext uniquely, and the cipher is, at least in theory, breakable.

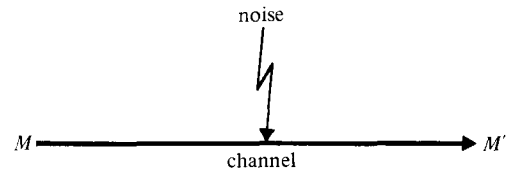
Most ciphers are theoretically breakable with only a few hundred bits of plaintext. But this does not mean these ciphers are insecure, because the computational requirements to determine the plaintext may exceed available resources. Thus, the important question is not whether a cipher is unconditionally secure, but whether it is computationally secure in the sense of being infeasible to break.

This section reviews information theory and its application to cryptography. Information theory also applies to the problem of controlling information dissemination; this application is discussed in Chapter 5. Section 1.5 discusses computational complexity and its application to cryptography.

Information theory addresses two related problems: the "noisy channel problem" and the secrecy problem. In the noisy channel problem, a sender transmits a message M over a noisy channel to a receiver (see Figure 1.13). If a distorted message M' is received, then the receiver would like to recover M. To make this possible, the sender adds redundant bits (called error control codes) to M in such a way that transmission errors can be corrected (or at least detected so that the receiver can request retransmission).

The noisy channel problem is analogous to the secrecy problem in cryptographic systems—the noise corresponding to the enciphering transformation, the

FIGURE 1.13 Noisy channel.



received message M' to ciphertext. Although the role of the cryptanalyst is similar to the role of the receiver in the noisy channel problem, the role of the sender is quite different because the objective is to make message recovery infeasible. (See [Simm79] for more discussion of this.)

1.4.1. Entropy and Equivocation

Information theory measures the **amount of information** in a message by the average number of bits needed to encode all possible messages in an optimal encoding. The *Sex* field in a database, for example, contains only one bit of information because it can be encoded with one bit (*Male* can be represented by "0", *Female* by "1"). If the field is represented by an ASCII character encoding of the character strings "MALE" and "FEMALE", it will take up more space, but will not contain any more information. The *Salary* field in a database, however, contains more than one bit of information, because there are more possibilities, and these possibilities cannot all be encoded with one bit. In computer systems, programs and text files are usually encoded with 8-bit ASCII codes, regardless of the amount of information in them. As we shall see shortly, text files can be compressed by about 40% without losing any information.

The amount of information in a message is formally measured by the entropy of the message. The entropy is a function of the probability distribution over the set of all possible messages. Let $X_1, \ldots, X_n$ be n possible messages occurring with probabilities $p(X_1), \ldots, p(X_n)$, where $\sum_{i=1}^{n} p(X_i) = 1$. The **entropy** of a given message is defined by the weighted average:

$$H(X) = -\sum_{i=1}^{n} p(X_i) \log_2 p(X_i) .$$

We shall write this as the sum taken over all messages X:

$$\begin{aligned} H(X) &= -\sum_{X} p(X) \log_2 p(X) \\ &= \sum_{X} p(X) \log_2 \left(\frac{1}{p(X)}\right) . \end{aligned} \tag{1.1}$$

Example:
Suppose there are two possibilities: *Male* and *Female,* both equally likely; thus $p(Male) = p(Female) = 1/2$. Then

$$\begin{aligned} H(X) &= \frac{1}{2}(\log_2 2) + \frac{1}{2}(\log_2 2) \\ &= \frac{1}{2} + \frac{1}{2} = 1 , \end{aligned}$$

confirming our earlier observation that there is 1 bit of information in the *Sex* field of a database. ■

Intuitively, each term $\log_2 (1/p(X))$ in Eq. (1.1) represents the number of bits needed to encode message X in an optimal encoding—that is, one which minimizes the expected number of bits transmitted over the channel. The weighted average $H(X)$ gives the expected number of bits in optimally encoded messages.

Because $1/p(X)$ decreases as $p(X)$ increases, an optimal encoding uses short codes for frequently occurring messages at the expense of using longer ones for infrequent messages. This principle is applied in Morse code, where the most frequently used letters are assigned the shortest codes.

"Huffman codes" [Huff52] are optimal codes assigned to characters, words, machine instructions, or phrases. Single-character Huffman codes are frequently used to compact large files. This is done by first scanning the file to determine the frequency distribution of the ASCII characters, next finding the optimal encoding of the characters, and finally replacing each character with its code. The codes are stored in a table at the beginning of the file, so the original text can be recovered. By encoding longer sequences of characters, the text can be compacted even further, but the storage requirements for the table are increased. A character encoding of the text file for this chapter using the *Compact* program on UNIX† reduced its storage requirements by 38%, which is typical for text files.‡ Machines with variable-length instruction sets use Huffman codes to assign short codes to frequently used instructions (e.g., LOAD, STORE, BRANCH).

The following examples further illustrate the application of Eq. (1.1) to determine the information content of a message.

Example:

Let $n = 3$, and let the 3 messages be the letters A, B, and C, where $p(\mathrm{A}) = 1/2$ and $p(\mathrm{B}) = p(\mathrm{C}) = 1/4$. Then

$$\log_2\left(\frac{1}{p(\mathrm{A})}\right) = \log_2 2 = 1$$

$$\log_2\left(\frac{1}{p(\mathrm{B})}\right) = \log_2 4 = 2$$

$$\log_2\left(\frac{1}{p(\mathrm{C})}\right) = \log_2 4 = 2\,,$$

and

$$H(X) = \left(\frac{1}{2}\right)\log_2 2 + 2\left[\left(\frac{1}{4}\right)\log_2 4\right] = 0.5 + 1.0 = 1.5\ .$$

An optimal encoding assigns a 1-bit code to A and 2-bit codes to B and C. For example, A can be encoded with the bit 0, while B and C can be encoded with two bits each, 10 and 11. Using this encoding, the 8-letter sequence ABAACABC is encoded as the 12-bit sequence 010001101011 as shown next:

† UNIX is a trademark of Bell Labs.

‡ Tom Sederberg wrote a program to determine the net reduction in space for this chapter when sequences of n characters are encoded. For $n = 2$, the reduction was again about 38% (the increase in table size compensating for the decrease in text space); for $n = 3$, it dropped to about 25%.

A	B	A	A	C	A	B	C
0	10	0	0	11	0	10	11

The average number of bits per letter is 12/8 = 1.5.

The preceding encoding is optimal; the expected number of bits per letter would be at least 1.5 with any other encoding. Note that B, for example, cannot be encoded with the single bit 1, because it would then be impossible to decode the bit sequence 11 (it could be either BB or C). Morse code avoids this problem by separating letters with spaces. Because spaces (blanks) must be encoded in computer applications, this approach in the long run requires more storage. ■

Example:
Suppose all messages are equally likely; that is, $p(X_i) = 1/n$ for $i = 1, \ldots, n$. Then

$$H(X) = n[\left(\frac{1}{n}\right)\log_2 n] = \log_2 n.$$

Thus, $\log_2 n$ bits are needed to encode each message. For $n = 2^k$, $H(X) = k$ and k bits are needed to encode each possible message. ■

Example:
Let $n = 1$ and $p(X) = 1$. Then $H(X) = \log_2 1 = 0$. There is no information because there is no choice. ■

Given n, $H(X)$ is maximal for $p(X_1) = \ldots = p(X_n) = 1/n$; that is, when all messages are equally likely (see exercises at end of chapter). $H(X)$ decreases as the distribution of messages becomes more and more skewed, reaching a minimum of $H(X) = 0$ when $p(X_i) = 1$ for some message X_i. As an example, suppose X represents a 32-bit integer variable. Then X can have at most 32 bits of information. If small values of X are more likely than larger ones (as is typical in most programs), then $H(X)$ will be less than 32, and if the exact value of X is known, $H(X)$ will be 0.

The entropy of a message measures its **uncertainty** in that it gives the number of bits of information that must be learned when the message has been distorted by a noisy channel or hidden in ciphertext. For example, if a cryptanalyst knows the ciphertext block "Z$JP7K" corresponds to either the plaintext "MALE" or the plaintext "FEMALE", the uncertainty is only one bit. The cryptanalyst need only determine one character, say the first, and because there are only two possibilities for that character, only the distinguishing bit of that character need be determined. If it is known that the block corresponds to a salary, then the uncertainty is more than one bit, but it can be no more than $\log_2 n$ bits, where n is the number of possible salaries.

Public-key systems used for secrecy only are vulnerable to a ciphertext-only

attack if there is not enough uncertainty in the plaintext. To see why, consider a ciphertext $C = E_A(M)$, where E_A is a public enciphering transformation and M is a plaintext message in a set of n possible messages $M_1, \ldots, M_n$. Even if it is computationally infeasible to determine the private deciphering transformation D_A, it may be possible to determine M by computing $C_i = E_A(M_i)$ for $i = 1, 2, \ldots$ until $C = C_i$, whence $M = M_i$. This type of attack would work, for example, if M is known to be an integer salary less than \$100,000 because there would be at most 100,000 messages to try. The attack can be prevented by appending a random bit string to a short message M before enciphering; this string would be discarded on deciphering. Of course, if authenticity is used with secrecy—that is, $C = E_A(D_B(M))$, the cryptanalyst, lacking D_B, cannot search the plaintext space this way. Conventional systems are not vulnerable to this attack because the enciphering (and deciphering) key is secret.

For a given language, consider the set of all messages N characters long. The **rate of the language** for messages of length N is defined by $r = H(X)/N$; that is, the average number of bits of information in each character. For large N, estimates of r for English range from 1.0 bits/letter to 1.5 bits/letter. The **absolute rate** of the language is defined to be the maximum number of bits of information that could be encoded in each character assuming all possible sequences of characters are equally likely. If there are L characters in the language, then the absolute rate is given by $R = \log_2 L$, the maximum entropy of the individual characters. For English, $R = \log_2 26 = 4.7$ bits/letter. The actual rate of English is thus considerably less than its absolute rate. The reason is that English, like all natural languages, is highly redundant. For example, the phrase "occurring frequently" could be reduced by 58% to "crng frq" without loss of information. By deleting vowels and double letters, mst ids cn b xprsd n fwr ltrs, bt th xprnc s mst nplsnt.

Redundancy arises from the structure of the language. It is reflected in the statistical properties of English language messages in the following ways [Shan51]:

1. *Single letter frequency distributions.* Certain letters such as E, T, and A occur much more frequently than others.
2. *Digram frequency distributions.* Certain digrams (pairs of letters) such as TH and EN occur much more frequently than others. Some digrams (e.g., QZ) never occur in meaningful messages even when word boundaries are ignored (acronyms are an exception).
3. *Trigram distributions.* The proportion of meaningful sequences decreases when trigrams are considered (e.g., BB is meaningful but BBB is not). Among the meaningful trigrams, certain sequences such as THE and ING occur much more frequently than others.
4. N-*gram distributions.* As longer sequences are considered, the proportion of meaningful messages to the total number of possible letter sequences decreases. Long messages are structured not only according to letter sequences within a word but also by word sequences (e.g., the phrase PROGRAMMING LANGUAGES is much more likely than the phrase LANGUAGES PROGRAMMING).

Programming languages have a similar structure, reflected in the statistical properties of programs [Turn73]. Here there is more freedom in letter sequences (e.g., the variable name QZK is perfectly valid), but the language syntax imposes other rigid rules about the placement of keywords and delimiters.

The rate of a language (entropy per character) is determined by estimating the entropy of N-grams for increasing values of N. As N increases, the entropy per character decreases because there are fewer choices and certain choices are much more likely. The decrease is sharp at first but tapers off quickly; the rate is estimated by extrapolating for large N. (See [Shan51,Cove78].)

The **redundancy** of a language with rate r and absolute rate R is defined by $D = R - r$. For $R = 4.7$ and $r = 1$, $D = 3.7$, whence the ratio D/R shows English to be about 79% redundant; for $r = 1.5$, $D = 3.2$, implying a redundancy of 68%. We shall use the more conservative estimate $r = 1.5$ and $D = 3.2$ in our later examples.

The uncertainty of messages may be reduced given additional information. For example, let X be a 32-bit integer such that all values are equally likely; thus the entropy of X is $H(X) = 32$. Suppose it is learned that X is even. Then the entropy is reduced by one bit because the low order bit must be 0.

Given a message Y in the set $Y_1, \ldots, Y_m$, where $\sum_{i=1}^{m} p(Y_i) = 1$, let $p_Y(X)$ be the conditional probability of message X given message Y [this is sometimes written $P(X \mid Y)$], and let $p(X, Y)$ be the joint probability of message X and message Y; thus,

$$p(X, Y) = p_Y(X)p(Y) .$$

The **equivocation** is the conditional entropy of X given Y:

$$H_Y(X) = -\sum_{X,Y} p(X, Y) \log_2 p_Y(X) ,$$

which we shall write as

$$H_Y(X) = \sum_{X,Y} p(X, Y) \log_2 \left(\frac{1}{p_Y(X)}\right) \qquad \textbf{(1.2a)}$$

or

$$H_Y(X) = \sum_{Y} p(Y) \sum_{X} p_Y(X) \log_2 \left(\frac{1}{p_Y(X)}\right) . \qquad \textbf{(1.2b)}$$

Example:

Let $n = 4$ and $p(X) = 1/4$ for each message X; thus $H(X) = \log_2 4 = 2$. Similarly, let $m = 4$ and $p(Y) = 1/4$ for each message Y. Now, suppose each message Y narrows the choice of X to two of the four messages as shown next, where both messages are equally likely:

Y_1: X_1 or X_2, $\quad Y_2$: X_2 or X_3
Y_3: X_3 or X_4, $\quad Y_4$: X_4 or X_1 .

Then for each Y, $p_Y(X) = 1/2$ for two of the X's and $p_Y(X) = 0$ for the remaining two X's. Using Eq. (1.2b), the equivocation is thus

$$H_Y(X) = 4[\ \left(\frac{1}{4}\right) 2[\ \left(\frac{1}{2}\right) \log_2 2]\ ] = \log_2 2 = 1\ .$$

Thus knowledge of Y reduces the uncertainty of X to one bit, corresponding to the two remaining choices for X. ■

1.4.2. Perfect Secrecy

Shannon studied the information theoretic properties of cryptographic systems in terms of three classes of information:

1. Plaintext messages M occurring with prior probabilities $p(M)$, where $\sum_M p(M) = 1$.
2. Ciphertext messages C occurring with probabilities $p(C)$, where $\sum_C p(C) = 1$.
3. Keys K chosen with prior probabilities $p(K)$, where $\sum_K p(K) = 1$.

Let $p_C(M)$ be the probability that message M was sent given that C was received (thus C is the encryption of message M). **Perfect secrecy** is defined by the condition

$$p_C(M) = p(M)\ ;$$

that is, intercepting the ciphertext gives a cryptanalyst no additional information.

Let $p_M(C)$ be the probability of receiving ciphertext C given that M was sent. Then $p_M(C)$ is the sum of the probabilities $p(K)$ of the keys K that encipher M as C:

$$p_M(C) = \sum_{E_K(M) = C} p(K)$$

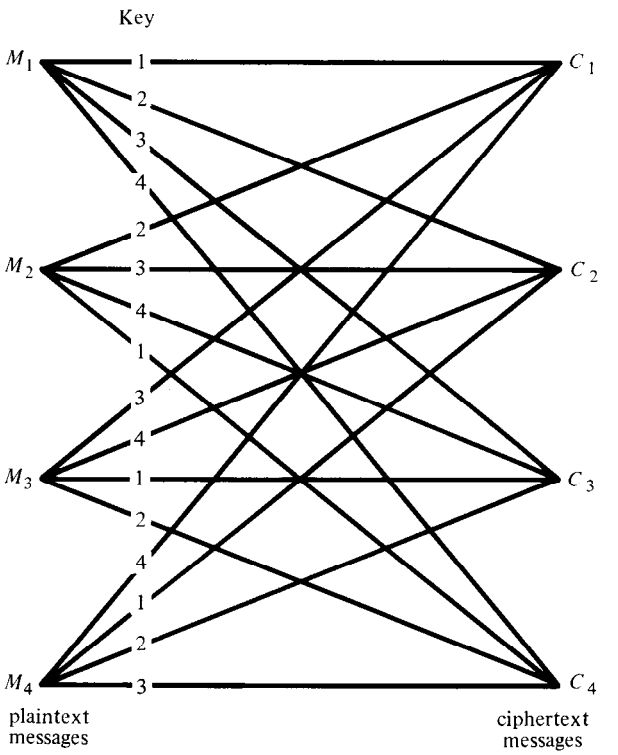
Usually there is at most one key K such that $E_K(M) = C$ for given M and C, but some ciphers can transform the same plaintext into the same ciphertext under different keys.

A necessary and sufficient condition for perfect secrecy is that for every C,

$$p_M(C) = p(C) \text{ for all } M\ .$$

This means the probability of receiving a particular ciphertext C given that M was sent (enciphered under some key) is the same as the probability of receiving C given that some other message M' was sent (enciphered under a different key). Perfect secrecy is possible using completely random keys at least as long as the messages they encipher. Figure 1.14 illustrates a perfect system with four messages, all equally likely, and four keys, also equally likely. Here $P_C(M) = P(M) = 1/4$, and $p_M(C) = p(C) = 1/4$ for all M and C. A cryptanalyst intercepting one of the ciphertext messages C_1, C_2, C_3, or C_4 would have no way of

FIGURE 1.14 Perfect secrecy (adapted from [Shan49]).



determining which of the four keys was used and, therefore, whether the correct message is M_1, M_2, M_3, or M_4.

Perfect secrecy requires that the number of keys must be at least as great as the number of possible messages. Otherwise there would be some message M such that for a given C, no K deciphers C into M, implying $P_C(M) = 0$. The cryptanalyst could thereby eliminate certain possible plaintext messages from consideration, increasing the chances of breaking the cipher.

Example:
Suppose the 31-character ciphertext

$$C = \text{LZWJWAKFGGLZWJDSFYMSYWTMLXJWFUZ}$$

was produced by a Caesar cipher (see Section 1.1), where each letter in the alphabet is shifted forward by K positions, $0 \leq K \leq 25$. Because the number of possible keys is smaller than the number of possible English sentences of length 31, perfect secrecy is not achieved. The cipher is easily broken by trying all 26 keys as shown in Figure 1.15. The plaintext message is

FIGURE 1.15 Solution of substitution cipher.

Key	Message
0:	L Z W J W A K F G G L Z W J D S F Y M S Y W T M L X J W F U Z
1:	K Y V I V Z J E F F K Y V I C R E X L R X V S L K W I V E T Y
2:	J X U H U Y I D E E J X U H B Q D W K Q W U R K J V H U D S X
3:	I W T G T X H C D D I W T G A P C V J P V T Q J I U G T C R W
4:	H V S F S W G B C C H V S F Z O B U I O U S P I H T F S B Q V
5:	G U R E R V F A B B G U R E Y N A T H N T R O H G S E R A P U
6:	F T Q D Q U E Z A A F T Q D X M Z S G M S Q N G F R D Q Z O T
7:	E S P C P T D Y Z Z E S P C W L Y R F L R P M F E Q C P Y N S
8:	D R O B O S C X Y Y D R O B V K X Q E K Q O L E D P B O X M R
9:	C Q N A N R B W X X C Q N A U J W P D J P N K D C O A N W L Q
10:	B P M Z M Q A V W W B P M Z T I V O C I O M J C B N Z M V K P
11:	A O L Y L P Z U V V A O L Y S H U N B H N L I B A M Y L U J O
12:	Z N K X K O Y T U U Z N K X R G T M A G M K H A Z L X K T I N
13:	Y M J W J N X S T T Y M J W Q F S L Z F L J G Z Y K W J S H M
14:	X L I V I M W R S S X L I V P E R K Y E K I F Y X J V I R G L
15:	W K H U H L V Q R R W K H U O D Q J X D J H E X W I U H Q F K
16:	V J G T G K U P Q Q V J G T N C P I W C I G D W V H T G P E J
17:	U I F S F J T O P P U I F S M B O H V B H F C V U G S F O D I
18:	T H E R E I S N O O T H E R L A N G U A G E B U T F R E N C H
19:	S G D Q D H R M N N S G D Q K Z M F T Z F D A T S E Q D M B G
20:	R F C P C G Q L M M R F C P J Y L E S Y E C Z S R D P C L A F
21:	Q E B O B F P K L L Q E B O I X K D R X D B Y R Q C O B K Z E
22:	P D A N A E O J K K P D A N H W J C Q W C A X Q P B N A J Y D
23:	O C Z M Z D N I J J O C Z M G V I B P V B Z W P O A M Z I X C
24:	N B Y L Y C M H I I N B Y L F U H A O U A Y V O N Z L Y H W B
25:	M A X K X B L G H H M A X K E T G Z N T Z X U N M Y K X G V A

M = THERE IS NO OTHER LANGUAGE BUT FRENCH.†

Because only one of the keys ($K = 18$) produces a meaningful message, we have:

$$p_C(M) = 1$$
$$p_C(M') = 0, \text{ for every other message } M'$$
$$p_M(C) = p(18) = \frac{1}{26}$$
$$p_{M'}(C) = 0, \text{ for every other message } M' . \blacksquare$$

Example:

With a slight modification to the preceding scheme, we can create a cipher having perfect secrecy. The trick is to shift each letter by a random amount. Specifically, K is given by a stream $k_1 k_2 \ldots$, where each k_i is a random

† From S. Gorn's Compendium of Rarely Used Cliches.

integer in the range [0, 25] giving the amount of shift for the ith letter. Then the 31-character ciphertext C in the preceding example could correspond to any valid 31-character message, because each possible plaintext message is derived by some key stream. For example, the plaintext message

THIS SPECIES HAS ALWAYS BEEN EXTINCT.†

is derived by the key stream

18, 18, 14, 17, 4, ...

Though most of the 31-character possible plaintext messages can be ruled out as not being valid English, this much is known even without the ciphertext. Perfect secrecy is achieved because interception of the ciphertext does not reveal anything new about the plaintext message.

The key stream must not repeat or be used to encipher another message. Otherwise, it may be possible to break the cipher by correlating two ciphertexts enciphered under the same portion of the stream (see Section 2.4.4). ■

A cipher using a nonrepeating random key stream such as the one described in the preceding example is called a **one-time pad**. One-time pads are the only ciphers that achieve perfect secrecy. Implementation of one-time pads and approximations to one-time pads is studied in Chapters 2 and 3.

1.4.3 Unicity Distance

Shannon measured the secrecy of a cipher in terms of the **key equivocation** $H_C(K)$ of a key K for a given ciphertext C; that is, the amount of uncertainty in K given C. From Eq. (1.2b), this is

$$H_C(K) = \sum_C p(C) \sum_K p_C(K) \log_2 \left(\frac{1}{p_C(K)}\right) ,$$

where $p_C(K)$ is the probability of K given C. If $H_C(K)$ is 0, then there is no uncertainty, and the cipher is theoretically breakable given enough resources. As the length N of the ciphertext increases, the equivocation usually decreases.

The **unicity distance** is the smallest N such that $H_C(K)$ is close to 0; that is, it is the amount of ciphertext needed to uniquely determine the key. A cipher is **unconditionally secure** if $H_C(K)$ never approaches 0 even for large N; that is, no matter how much ciphertext is intercepted, the key cannot be determined. (Shannon used the term "ideal secrecy" to describe systems that did not achieve perfect secrecy, but were nonetheless unbreakable because they did not give enough information to determine the key.)

Most ciphers are too complex to determine the probabilities required to derive the unicity distance. Shannon showed, however, it is possible to approxi-

† Also from S. Gorn's Compendium of Rarely Used Cliches.

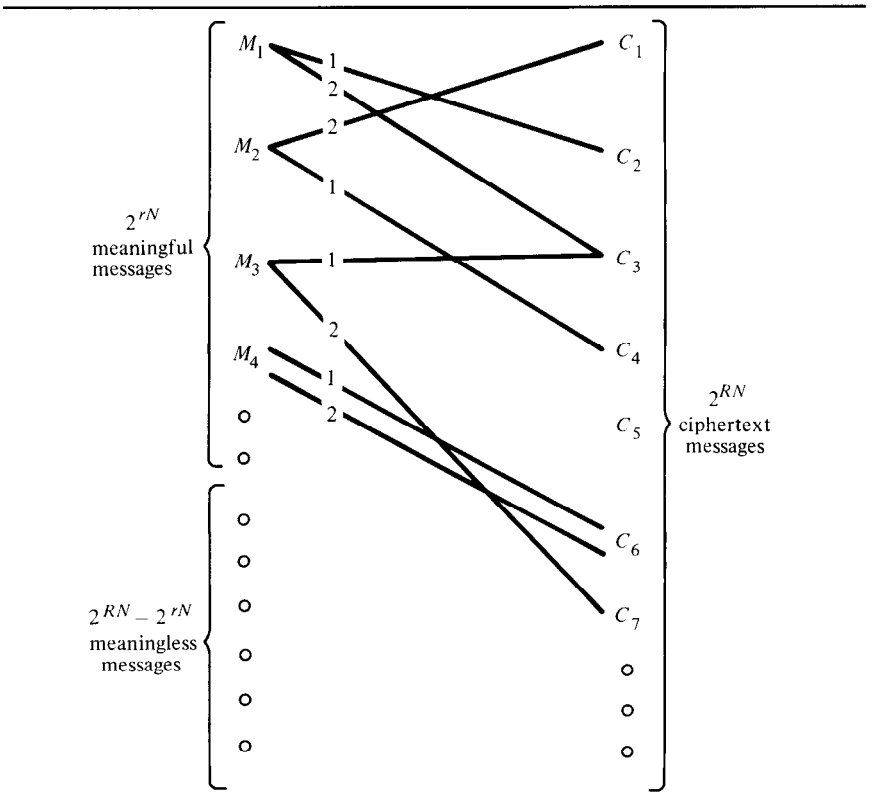
mate it for certain ciphers using a random cipher model. Hellman [Hell77] also derived Shannon's result using a slightly different approach.

Following Hellman, we assume each plaintext and ciphertext message comes from a finite alphabet of L symbols. Thus there are 2^{RN} possible messages of length N, where $R = \log_2 L$ is the absolute rate of the language. The 2^{RN} messages are partitioned into two subsets: a set of 2^{rN} **meaningful messages** and a set of $2^{RN} - 2^{rN}$ **meaningless messages**, where r is the rate of the language. All meaningful messages are assumed to have the same prior probability $1/2^{rN} = 2^{-rN}$, while all meaningless messages are assumed to have probability 0.

We also assume there are $2^{H(K)}$ keys, all equally likely, where $H(K)$ is the key entropy (number of bits in the key). The prior probability of all keys is $p(K) = 1/2^{H(K)} = 2^{-H(K)}$.

A **random cipher** is one in which for each key K and ciphertext C, the decipherment $D_K(C)$ is an independent random variable uniformly distributed over all 2^{RN} messages, both meaningful and not. Intuitively, this means that for a given K and C, $D_K(C)$ is as likely to produce one plaintext message as any other. Actually the decipherments are not completely independent because a given key must uniquely encipher a given message, whence $D_K(C) \neq D_K(C')$ for $C \neq C'$.

FIGURE 1.16 Random cipher model (adapted from [Hell 77]).



Consider the ciphertext $C = E_K(M)$ for given K and M. A **spurious key decipherment** or **false solution** arises whenever encipherment under another key K' could produce C; that is, $C = E_{K'}(M)$ for the same message M, or $C = E_{K'}(M')$ for another meaningful message M'. Figure 1.16 shows two spurious key decipherments, one from the third ciphertext and one from the sixth. A cryptanalyst intercepting one of these ciphertexts would be unable to break the cipher since there would be no way of picking the correct key. We are not concerned with decipherments that produce meaningless messages, because the cryptanalyst can immediately reject these solutions.

Now, for every correct solution to a particular ciphertext, there are $(2^{H(K)} - 1)$ remaining keys, each of which has the same probability q of yielding a spurious key decipherment. Because each plaintext message is equally likely, the probability of getting a meaningful message and, therefore, a false solution is given by

$$q = \frac{2^{rN}}{2^{RN}} = 2^{(r-R)N} = 2^{-DN},$$

where $D = R - r$ is the redundancy of the language. Letting F denote the expected number of false solutions, we have

$$F = (2^{H(K)} - 1)\, q = (2^{H(K)} - 1)\, 2^{-DN} \cong 2^{H(K)-DN} . \tag{1.3}$$

Because of the rapid decrease in the exponential with increasing N,

$$\log_2 F = H(K) - DN = 0$$

is taken as the point where the number of false solutions is sufficiently small the cipher can be broken. Thus

$$N = \frac{H(K)}{D} \tag{1.4}$$

is the unicity distance—the amount of text necessary to break the cipher.

If for given N, the number of possible keys is as large as the number of meaningful messages, then $H(K) = \log_2(2^{RN}) = RN$; thus

$$H(K) - DN = (R - D)N = rN \neq 0 ,$$

and the cipher is theoretically unbreakable. This is the principle behind the one-time pad.

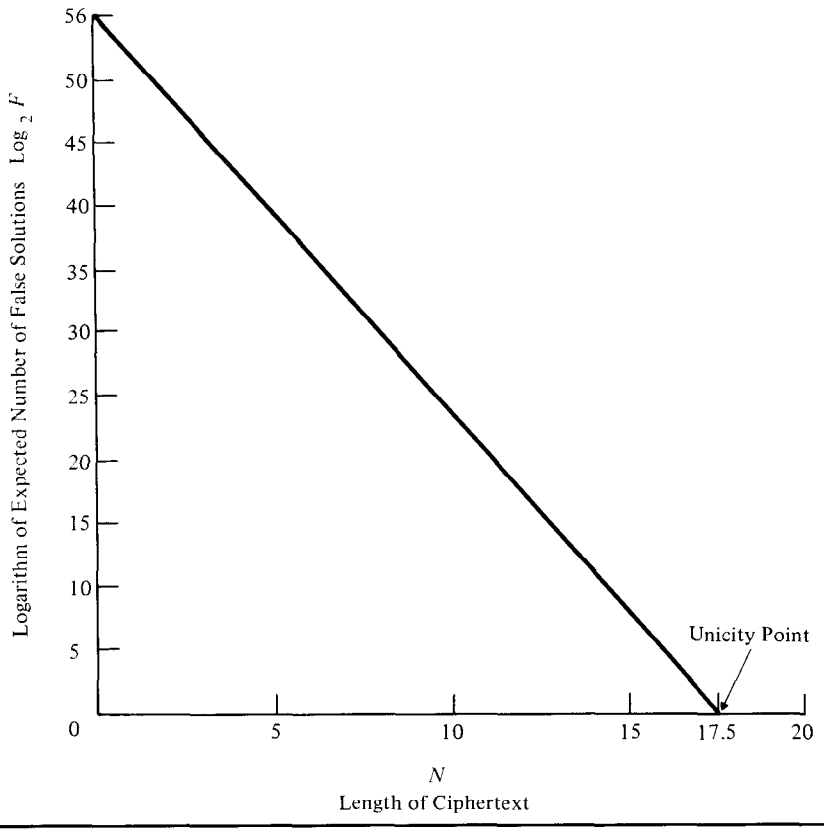
Example:

Consider the DES, which enciphers 64-bit blocks (8 characters) using 56-bit keys. The DES is a reasonably close approximation to the random cipher model. Figure 1.17 shows F as a function of N for English language messages, where $H(K) = 56$ and $D = 3.2$ in Eq. (1.3). The unicity distance is thus

$$N = \frac{56}{3.2} = 17.5 \text{ characters},$$

or a little over two blocks. Doubling the key size to 112 bits would double the unicity distance to 35 characters. ■

FIGURE 1.17 Unicity distance for DES.



Example:
Consider a simple substitution cipher that shifts every letter in the alphabet forward by K positions, $0 \leq K \leq 25$. Then $H(K) = \log_2 26 = 4.7$ and the unicity distance is

$$N = \frac{4.7}{3.2} = 1.5 \text{ characters.}$$

This estimate does not seem plausible, however, because no substitution cipher can be solved with just one or two characters of ciphertext. There are two problems with the approximation. First, the estimate $D = 3.2$ applies only to reasonable long messages. Second, the cipher is a poor approximation to the random cipher model. This is because most ciphertexts are not produced by meaningful messages (e.g., the ciphertext QQQQ is produced only by the meaningless messages AAAA, BBBB, . . . , ZZZZ), whence the decipherments are not uniformly distributed over the entire message space. Nevertheless, shifted ciphers can generally be solved with just a few characters of ciphertext. ■

The random cipher model gives a conservative estimate of the amount of ciphertext needed to break a cipher. Thus a particular cipher will have a unicity distance of at least $H(K)/D$. In practice, $H(K)/D$ is a good approximation even for simple ciphers. We shall derive the unicity distance of several ciphers in Chapter 2. The interested reader can read more about the unicity distances of classical ciphers in Deavours [Deav77].

The unicity distance gives the number of characters required to uniquely determine the key; it does not indicate the computational difficulty of finding it. A cipher may be computationally infeasible to break even if it is theoretically possible with a relatively small amount of ciphertext. Public-key systems, for example, can be theoretically broken without any ciphertext at all. The cryptanalyst, knowing the public key and the method of generating key pairs, can systematically try all possible private keys until the matching key is found (see Brassard [Bras79a, Bras80]). This strategy is computationally infeasible, however, for large key spaces (e.g., with 2^{200} keys). The DES can also be broken by exhaustive search of the key space in a known-plaintext attack (by trying all keys until one is found that enciphers the plaintext into the matching ciphertext). Nevertheless, the best known strategies for breaking the DES are extremely time-consuming. By contrast, certain substitution ciphers discussed in the next chapter use longer keys and have much greater unicity distances than DES. These ciphers are often relatively simple to solve, however, when enough ciphertext is intercepted.

Equation (1.4) shows that the unicity distance N is inversely proportional to the redundancy D. As D approaches 0, an otherwise trivial cipher becomes unbreakable. To illustrate, suppose a 6-digit integer M is enciphered as 351972 using a Caesar-type shifted substitution cipher with key K, where $0 \leq K \leq 9$, and that all possible 6-digit integers are equally likely. Then a cryptanalyst cannot determine which of the following integers is the value of M:

Key	*Integer*
0	351972
1	240861
2	139750
.	.
.	.
.	.
9	462083

The reason the cipher cannot be solved is that the language has no redundancy; every digit counts.

Because of the inherent redundancy of natural languages, many ciphers can be solved by statistical analysis of the ciphertext. These techniques use frequency distributions of letters and sequences of letters, ciphertext repetitions, and probable words. Although a full discussion of these techniques is beyond the scope of this book, Chapter 2 describes how a few simple ciphers can be broken using frequency distributions. (For more depth in this area, see [Konh81].)

Protection against statistical analysis can be provided by several means. One way, suggested by Shannon, is by removing some of the redundancy of the lan-

guage before encryption. In computer systems, for example, Huffman codes could be used to remove redundancy by compressing a file before encryption.

Shannon also proposed two encryption techniques to thwart attacks based on statistical analysis: confusion and diffusion. **Confusion** involves substitutions that make the relationship between the key and ciphertext as complex as possible. **Diffusion** involves transformations that dissipate the statistical properties of the plaintext across the ciphertext. Many modern ciphers such as the DES and public-key schemes provide confusion and diffusion through complex enciphering transformations over large blocks of data. These ciphers can also be operated in a "chaining mode", where each ciphertext block is functionally dependent on all preceding blocks; this diffuses the plaintext across the entire ciphertext (see Chapter 3).

1.5 COMPLEXITY THEORY

Computational complexity provides a foundation for analyzing the computational requirements of cryptanalytic techniques, and for studying the inherent difficulty of solving ciphers. It also provides a foundation for studying the inherent difficulty of proving security properties about arbitrary systems (see Chapter 4), and for analyzing the computational difficulty of protecting confidential data released in the form of statistics (Chapter 6).

1.5.1 Algorithm Complexity

The strength of a cipher is determined by the computational complexity of the algorithms used to solve the cipher. The computational complexity of an algorithm is measured by its time (T) and space (S) requirements, where T and S are expressed as functions of n, and n characterizes the size of the input. A function $f(n)$ is typically expressed as an "order-of-magnitude" of the form $O(g(n))$ (called "big O" notation), where $f(n) = O(g(n))$ means there exist constants c and n_0 such that

$$f(n) \leq c\,|\,g(n)\,| \quad \text{for } n \geq n_0\,.$$

As an example, suppose $f(n) = 17n + 10$. Then $f(n) = O(n)$ because $17n + 10 \leq 18n$ for $n \geq 10$ [i.e., $g(n) = n$, $c = 18$, and $n_0 = 10$]. If $f(n)$ is a polynomial of the form

$$f(n) = a_t n^t + a_{t-1} n^{t-1} + \cdots + a_1 n + a_0$$

for constant t, then $f(n) = O(n^t)$; that is, all constants and low-order terms are ignored.

Measuring the time and space requirements of an algorithm by its order-of-magnitude performance has the advantage of being system independent; thus, it is unnecessary to know the exact timings of different instructions or the number of bits used to represent different data types. At the same time, it allows us to see

TABLE 1.2 Classes of algorithms.

Class	Complexity	Number of operations for $n = 10^6$	Real time
Polynomial			
Constant	$O(1)$	1	1 μsec
Linear	$O(n)$	10^6	1 second
Quadratic	$O(n^2)$	10^{12}	10 days
Cubic	$O(n^3)$	10^{18}	27,397 years
Exponential	$O(2^n)$	10^{301030}	10^{301016} years

how the time and space requirements grow as the size of the input increases. For example, if $T = O(n^2)$, doubling the size of the input quadruples the running time.

It is customary to classify algorithms by their time (or space) complexities. An algorithm is **polynomial** (more precisely, polynomial time) if its running time is given by $T = O(n^t)$ for some constant t; it is **constant** if $t = 0$, **linear** if $t = 1$, **quadratic** if $t = 2$, and so forth. It is **exponential** if $T = O(t^{h(n)})$ for constant t and polynomial $h(n)$.

For large n, the complexity of an algorithm can make an enormous difference. For example, consider a machine capable of performing one instruction per microsecond (μsec); this is 10^6 instructions per second, or 8.64×10^{10} instructions per day. Table 1.2 shows the running times of different classes of algorithms for $n = 10^6$, where we have ignored all constants and rounded to 10^{11} instructions per day. At $T = O(n^3)$ execution of the algorithm becomes computationally infeasible on a sequential machine. It is conceivable, however, that a configuration with 1 million processors could complete the computation in about 10 days. For $T = O(2^n)$ execution of the algorithm is computationally infeasible even if we could have trillions of processors working in parallel.

Many ciphers can be solved by exhaustively searching the entire key space, trying each possible key to ascertain whether it deciphers into meaningful plaintext or some known plaintext. If $n = 2^{H(K)}$ is the size of the key space, then the running time of this strategy is $T = O(n) = O(2^{H(K)})$. Thus, the time is linear in the number of keys, but exponential in the key length. This is why doubling the length of the keys used for DES from 56 bits to 112 bits can have a dramatic impact on the difficulty of breaking the cipher, even though it increases the unicity distance only by a factor of 2.

1.5.2. Problem Complexity and NP-Completeness

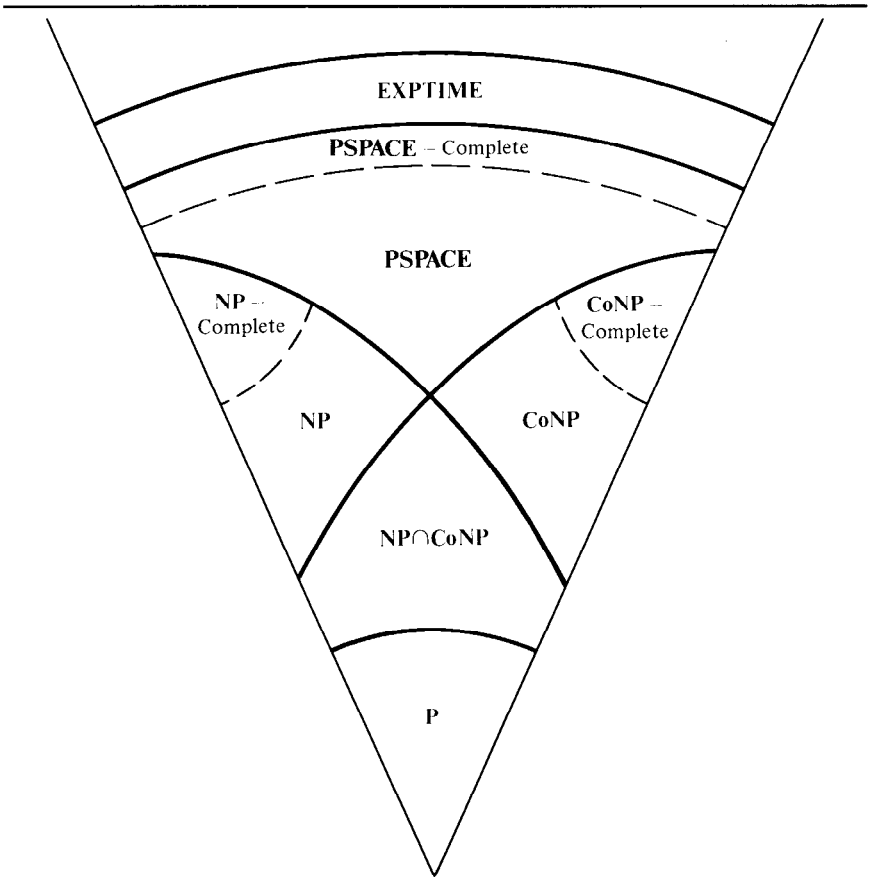
Complexity theory classifies a problem according to the minimum time and space needed to solve the hardest instances of the problem on a Turing Machine (or some other abstract model of computation). A Turing Machine (TM) is a finite state machine with an infinite read-write tape (e.g., see [Gare79,Aho74,Mins67] or the description in Section 4.7.2 for details). A TM is a "realistic" model of

computation in that problems that are polynomial solvable on a TM are also polynomial solvable on real systems and vice versa.

Problems that are solvable in polynomial time are called **tractable** because they can usually be solved for reasonable size inputs. Problems that cannot be systematically solved in polynomial time are called **intractable** or simply "hard", because as the size of the input increases, their solution becomes infeasible on even the fastest computers. Turing [Turi36] proved that some problems are so hard they are **undecidable** in the sense that it is impossible to write an algorithm to solve them. In particular, he showed the problem of determining whether an arbitrary TM (or program) halts is undecidable. Many other problems have been shown to be undecidable by proving that if they could be solved, then the "halting problem" could be solved (see Section 4.7.2 for an example).

Figure 1.18 shows several important complexity classes and their possible relationships (their exact relationships are unknown). The class **P** consists of all problems solvable in polynomial time.

FIGURE 1.18 Complexity classes.



The class **NP** (nondeterministic polynomial) consists of all problems solvable in polynomial time on a nondeterministic TM. This means if the machine guesses the solution, it can check its correctness in polynomial time. Of course, this does not really "solve" the problem, because there is no guarantee the machine will guess the right answer.

To systematically (deterministically) solve certain problems in **NP** seems to require exponential time. An example of such a problem is the "knapsack problem": given a set of n integers $A = \{a_1, \ldots, a_n\}$ and an integer S, determine whether there exists a subset of A that sums to S.† The problem is clearly in **NP** because for any given subset, it is easy to check whether it sums to S. Finding a subset that sums to S is much harder, however, as there are 2^n possible subsets; trying all of them has time complexity $T = O(2^n)$. Another example of a problem that seems to have exponential time complexity is the "satisfiability problem", which is to determine whether there exists an assignment of values to a set of n boolean variables $v_1, \ldots, v_n$ such that a given set of clauses over the variables is true.

The class **NP** includes the class **P** because any problem polynomial solvable on a deterministic TM is polynomial solvable on a nondeterministic one. If all **NP** problems are polynomial solvable on a deterministic TM, we would have **P** = **NP**. Although many problems in **NP** seem much "harder" than the problems in **P** (e.g., the knapsack problem and satisfiability) no one has yet proved **P** ≠ **NP**.

Cook [Cook71] showed the satisfiability problem has the property that every other problem in **NP** can be reduced to it in polynomial time. This means that if the satisfiability problem is polynomial solvable, then every problem in **NP** is polynomial solvable, and if some problem in **NP** is intractable, then satisfiability must also be intractable. Since then, other problems (including the knapsack problem) have been shown to be equivalent to satisfiability in the preceding sense. This set of equivalent problems is called the **NP-complete** problems, and has the property that if any one of the problems is in **P**, then all **NP** problems are in **P** and **P** = **NP**. Thus, the **NP**-complete problems are the "hardest" problems in **NP**. The fastest known algorithms for systematically solving these problems have worst-case time complexities exponential in the size n of the problem. Finding a polynomial-time solution to one of them would be a major breakthrough in computer science.

A problem is shown to be **NP**-complete by proving it is **NP**-hard and in **NP**. A problem is **NP-hard** if it cannot be solved in polynomial time unless **P** = **NP**. To show a problem A is **NP**-hard, it is necessary to show that some **NP**-complete problem B is polynomial-time reducible to an instance of A, whence a polynomial-time algorithm for solving A would also solve B. To show A is in **NP**, it is necessary to prove that a correct solution can be proved correct in polynomial time.

The class **CoNP** consists of all problems that are the complement of some problem in **NP**. Intuitively, problems in **NP** are of the form "determine whether a

† The integers represent rod lengths, and the problem is to find a subset of rods that exactly fits a one-dimensional knapsack of length n.

solution exists," whereas the complementary problems in **CoNP** are of the form "show there are no solutions." It is not known whether **NP = CoNP**, but there are problems that fall in the intersection **NP** ∩ **CoNP**. An example of such a problem is the "composite numbers problem": given an integer n, determine whether n is composite (i.e., there exist factors p and q such that $n = pq$) or prime (i.e., there are no such factors). The problem of finding factors, however, may be harder than showing their existence.

The class **PSPACE** consists of those problems solvable in polynomial space, but not necessarily polynomial time. It includes **NP** and **CoNP**, but there are problems in **PSPACE** that are thought by some to be harder than problems in **NP** and **CoNP**. The **PSPACE**-complete problems have the property that if any one of them is in **NP**, then **PSPACE = NP**, or if any one is in **P**, then **PSPACE = P**. The class **EXPTIME** consists of those problems solvable in exponential time, and includes **PSPACE**. The interested reader is referred to [Gare79,Aho74] for a more complete treatment of complexity theory.

1.5.3. Ciphers Based on Computationally Hard Problems

In their 1976 paper, Diffie and Hellman [Diff76] suggested applying computational complexity to the design of encryption algorithms. They noted that **NP**-complete problems might make excellent candidates for ciphers because they cannot be solved in polynomial time by any known techniques. Problems that are computationally more difficult than the problems in **NP** are not suitable for encryption because the enciphering and deciphering transformations must be fast (i.e., computable in polynomial time). But this means the cryptanalyst could guess a key and check the solution in polynomial time (e.g., by enciphering known plaintext). Thus, the cryptanalytic effort to break any polynomial-time encryption algorithm must be in **NP**.

Diffie and Hellman speculated that cryptography could draw from the theory of **NP** complexity by examining ways in which **NP**-complete problems could be adapted to cryptographic use. Information could be enciphered by encoding it in an **NP**-complete problem in such a way that breaking the cipher would require solving the problem in the usual way. With the deciphering key, however, a shortcut solution would be possible.

To construct such a cipher, secret "trapdoor" information is inserted into a computationally hard problem that involves inverting a one-way function. A function f is a **one-way function** if it is easy to compute $f(x)$ for any x in the domain of f, while, for almost all y in the range of f, it is computationally infeasible to compute $f^{-1}(y)$ even if f is known. It is a **trapdoor one-way function** if it is easy to compute f^{-1} given certain additional information. This additional information is the secret deciphering key.

Public-key systems are based on this principle. The trapdoor knapsack schemes described in Section 2.8 are based on the knapsack problem. The RSA scheme described in Section 2.7 is based on factoring composite numbers.

The strength of such a cipher depends on the computational complexity of the problem on which it is based. A computationally difficult problem does not necessarily imply a strong cryptosystem, however. Shamir gives three reasons [Sham79]:

1. Complexity theory usually deals with single isolated instances of a problem. A cryptanalyst often has a large collection of statistically related problems to solve (e.g., several ciphertexts generated by the same key).
2. The computational complexity of a problem is typically measured by its worst-case or average-case behavior. To be useful as a cipher, the problem must be hard to solve in almost all cases.
3. An arbitrarily difficult problem cannot necessarily be transformed into a cryptosystem, and it must be possible to insert trapdoor information into the problem in such a way that a shortcut solution is possible with this information and only with this information.

Lempel [Lemp79] illustrates the first deficiency with a block cipher for which the problem of finding an n-bit key is **NP**-complete when the plaintext corresponding to one block of ciphertext is known. But given enough known plaintext, the problem reduces to solving n linear equations in n unknowns. The cipher is described in Section 2.8.4.

Shamir [Sham79] proposes a new complexity measure to deal with the second difficulty. Given a fraction r such that $0 \leq r \leq 1$, the **percentile complexity** $T(n, r)$ of a problem measures the time to solve the easiest proportion r of the problem instances of size n. For example, $T(n, 0.5)$ gives the **median complexity**; that is, at least half of the instances of size n can be solved within time $T(n, 0.5)$. The problem of deciding whether a given integer is prime has median complexity $O(1)$ because half of the numbers have 2 as a factor, and this can be tested in constant time.

With respect to the third difficulty, Brassard [Bras79b] shows it may not be possible to prove that the cryptanalytic effort to invert a trapdoor one-way function is **NP**-complete. If the function satisfies a few restrictions, then a proof of **NP**-completeness would imply **NP = CoNP**.

1.6 NUMBER THEORY

This section summarizes the concepts of number theory needed to understand the cryptographic techniques described in Chapters 2 and 3. Because we are primarily interested in the properties of modular arithmetic rather than congruences in general, we shall review the basic theorems of number theory in terms of modular arithmetic, emphasizing their computational aspects. We shall give proofs of these fascinating theorems for the benefit of readers unfamiliar with them. Readers

familiar with these results can go on to the next chapter. For a comprehensive treatment of this material, see, for example [LeVe77,Nive72,Vino55].

1.6.1 Congruences and Modular Arithmetic

Given integers a, b, and $n \neq 0$, a, is **congruent** to b modulo n, written†

$$a \equiv_n b$$

if and only if

$$a - b = kn$$

for some integer k; that is n divides $(a - b)$, written

$$n \mid (a - b) .$$

For example, $17 \equiv_5 7$, because $(17 - 7) = 2 * 5$.

If $a \equiv_n b$, then b is called a **residue** of a modulo n (conversely, a is a residue of b modulo n). A set of n integers $\{r_1, \ldots, r_n\}$ is called a **complete set of residues** modulo n if, for every integer a, there is exactly one r_i in the set such that $a \equiv_n r_i$. For any modulus n, the set of integers $\{0, 1, \ldots, n - 1\}$ forms a complete set of residues modulo n.

We shall write

$$a \bmod n$$

to denote the residue r of a modulo n in the range $[0, n - 1]$. For example, $7 \bmod 3 = 1$. Clearly,

$$a \bmod n = r \quad \text{implies} \quad a \equiv_n r,$$

but not conversely. Furthermore,

$$a \equiv_n b \quad \text{if and only if} \quad a \bmod n = b \bmod n;$$

thus, congruent integers have the same residue in the range $[0, n - 1]$.

Note that this definition of mod is somewhat different from the definition in some programming languages, such as PASCAL, where a mod n gives the remainder in dividing a by n. Whereas the range of our mod is $[0, n - 1]$, the range of PASCAL's is $[-(n - 1), n - 1]$. For example, $-2 \bmod 26 = -2$ in PASCAL rather than 24.

Like the integers, the integers mod n with addition and multiplication form a **commutative ring**. This means the laws of associativity, commutativity, and distributivity hold. Furthermore, computing in modular arithmetic (i.e., reducing each intermediate result mod n) gives the same answer as computing in ordinary integer arithmetic and reducing the result mod n. This is because reduction mod n is a

† We shall reserve the more familiar notation using "mod n" for modular arithmetic.

homomorphism from the ring of integers to the ring of integers mod n as shown next:

Theorem 1.1. Principle of modular arithmetic:
Let a_1 and a_2 be integers, and let *op* be one of the binary operators $+$, $-$, or $*$. Then reduction mod n is a homomorphism from the integers to the integers mod n (see Figure 1.19); that is,

$$(a_1 \; op \; a_2) \bmod n = [(a_1 \bmod n) \; op \; (a_2 \bmod n)] \bmod n.$$

Proof:
We can write

$$a_1 = k_1 n + r_1$$
$$a_2 = k_2 n + r_2$$

where $r_1, r_2 \in [0, n-1]$. For addition, we have

$$\begin{aligned}(a_1 + a_2) \bmod n &= [(k_1 n + r_1) + (k_2 n + r_2)] \bmod n \\ &= [(k_1 + k_2)n + (r_1 + r_2)] \bmod n \\ &= [r_1 + r_2] \bmod n \\ &= [(a_1 \bmod n) + (a_2 \bmod n)] \bmod n \; .\end{aligned}$$

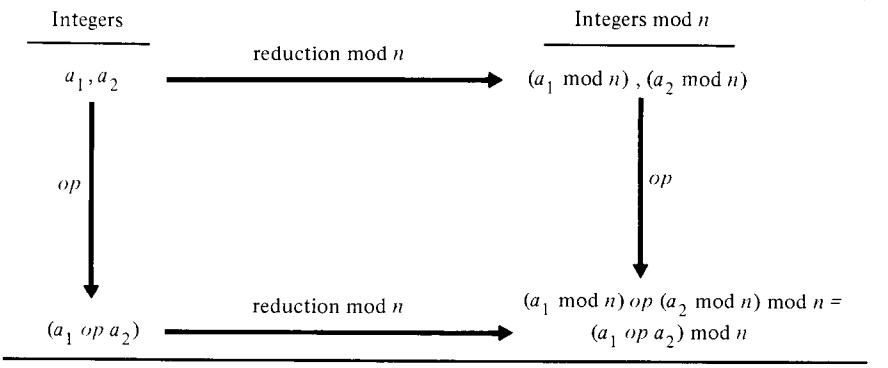
Subtraction is similar. For multiplication,

$$\begin{aligned}(a_1 * a_2) \bmod n &= [(k_1 n + r_1) * (k_2 n + r_2)] \bmod n \\ &[(k_1 k_2 n + r_1 k_2 + r_2 k_1)n + r_1 r_2] \bmod n \\ &[r_1 * r_2] \bmod n \\ &[(a_1 \bmod n) * (a_2 \bmod n)] \bmod n \; . \; \blacksquare\end{aligned}$$

The preceding theorem shows that evaluating $(a_1 \; op \; a_2)$ in modular arithmetic gives the same result as evaluating it in ordinary integer arithmetic and reducing the result mod n.

The principle of modular arithmetic is behind the familiar principle of "casting out 9's". For an integer a, a mod 9 is the sum of the digits of a (mod 9).

FIGURE 1.19 Principle of modular arithmetic.



Example:
The following illustrates how this principle is applied to check the result of a computation (135273 + 261909 + 522044):

Integer Arithmetic		Mod 9 Arithmetic
1 3 5 2 7 3		3
2 6 1 9 0 9	→	0
+ 5 2 2 0 4 4		+ 8
9 1 9 2 2 6	→	2 . ■

Note that the principle of modular arithmetic also applies to exponentiations of the form e^t, where $0 \leq t \leq n-1$; that is, computing e^t in mod n arithmetic is equivalent to computing e^t and reducing the result mod n. This is because exponentiation is equivalent to repeated multiplications:

$$e^t \bmod n = \left[\prod_{i=1}^{t} (e \bmod n)\right] \bmod n.$$

Example:
Consider the expression

$3^5 \bmod 7.$

This can be computed by raising 3 to the power 5 and then reducing the result mod 7 as shown next:

1. Square 3: $3 * 3 = 9$
2. Square the result: $9 * 9 = 81$
3. Multiply by 3: $81 * 3 = 243$
4. Reduce mod 7: $243 \bmod 7 = 5$,

where we have used the method of repeated squaring and multiplication to reduce the number of multiplications. Alternatively, the intermediate results of the computation can be reduced mod 7 as shown next:

1. Square 3: $3 * 3 \bmod 7 = 2$
2. Square the result: $2 * 2 \bmod 7 = 4$
3. Multiply by 3: $4 * 3 \bmod 7 = 5$. ■

If $t \geq n$, reducing t mod n may not give the same result; that is, $(e^{t \bmod n}) \bmod n$ may not equal $e^t \bmod n$. For example, $(2^{5 \bmod 3}) \bmod 3 = 1$, but $2^5 \bmod 3 = 2$.

Computing in modular arithmetic has the advantage of restricting the range of the intermediate values. For a k-bit modulus n (i.e., $2^{k-1} \leq n < 2^k$), the value of any addition, subtraction, or multiplication will be at most $2k$ bits. This means that we can, for example, perform exponentiations of the form $a^z \bmod n$ using large numbers without generating enormous intermediate results.

Because some of the encryption algorithms discussed later in this book are

FIGURE 1.20 Fast exponentiation.

Algorithm *fastexp* (a, z, n)

```
begin "return x = a^z mod n"
  a1 := a; z1 := z;
  x := 1;
  while z1 ≠ 0 do "x(a1^z1 mod n) = a^z mod n"
    begin
      while z1 mod 2 = 0 do
        begin "square a1 while z1 is even"
          z1 := z1 div 2;
          a1 := (a1 * a1) mod n
        end;
      z1 := z1 − 1;
      x  := (x * a1) mod n "multiply"
    end;
  fastexp := x
end
```

based on exponentiation mod n, we give here a fast exponentiation algorithm. The algorithm, shown in Figure 1.20, uses repeated squaring and multiplication. The notation for this algorithm and other algorithms in this book is based on PASCAL control structures. The operator "div" denotes integer division with truncation. Because $z1 = 0$ when the algorithm terminates, the loop invariant "$x(a1^{z1} \bmod n) = a^z \bmod n$" implies that $x = a^z \bmod n$. Suppose a, z, and n are k-bit integers. Letting $(z_{k-1}, \ldots, z_1, z_0)$ denote the binary representation of z, the algorithm processes the bits in the order $z_0, z_1, \ldots, z_{k-1}$ (i.e., from low order to high order), squaring when the bits are 0, and multiplying and squaring when they are 1. In a hardware implementation of the algorithm, these bits could be accessed directly, omitting the computations "$z1 \bmod 2$", "$z1$ div 2", and "$z1 - 1$".

Let T be the running time of the algorithm. Because each 0-bit gives rise to one multiplication and each 1-bit gives rise to two multiplications (except for the leftmost 1-bit, which gives rise to one multiplication), the number of multiplications is bounded by

$$k + 1 \leq T \leq 2k + 1,$$

where $k = \lfloor \log_2 z \rfloor$ is the length of z in bits ("$\lfloor \quad \rfloor$" denotes the floor—i.e., round down to nearest integer); this is linear in the length of z. The expected number of multiplications for all z of length k is $1.5k + 1$. By comparison, a naive algorithm performs $z - 1$ multiplications, which is exponential in the length of z.

1.6.2. Computing Inverses

Unlike ordinary integer arithmetic, modular arithmetic sometimes permits the computation of multiplicative inverses; that is, given an integer a in the range

$[0, n-1]$, it may be possible to find a unique integer x in the range $[0, n-1]$ such that

$$ax \bmod n = 1 .$$

For example, 3 and 7 are multiplicative inverses mod 10 because $21 \bmod 10 = 1$. It is this capability to compute inverses that makes modular arithmetic so appealing in cryptographic applications.

We will now show that given $a \in [0, n-1]$, a has a unique inverse mod n when a and n are relatively prime; that is when $gcd(a, n) = 1$, where "gcd" denotes the greatest common divisor. We first prove the following lemma:

Lemma 1.1:
If $gcd(a, n) = 1$, then $(ai \bmod n) \neq (aj \bmod n)$ for each i, j such that $0 \leq i < j < n$.

Proof:
Assume the contrary. Then $n \mid a(i-j)$. Since $gcd(a, n) = 1$, $(i-j)$ must be a multiple of n. But this is impossible because both i and j are smaller than n. ■

This property implies that each $ai \bmod n$ $(i = 0, \ldots, n-1)$ is a distinct residue mod n, and that the set $\{ai \bmod n\}_{i=0,\ldots,n-1}$ is a permutation of the complete set of residues $\{0, \ldots, n-1\}$. For example, if $n = 5$ and $a = 3$:

$$3 * 0 \bmod 5 = 0$$
$$3 * 1 \bmod 5 = 3$$
$$3 * 2 \bmod 5 = 1$$
$$3 * 3 \bmod 5 = 4$$
$$3 * 4 \bmod 5 = 2 .$$

This property does not hold when a and n have a common factor, as shown next:

$$2 * 0 \bmod 4 = 0$$
$$2 * 1 \bmod 4 = 2$$
$$2 * 2 \bmod 4 = 0$$
$$2 * 3 \bmod 4 = 2 .$$

Lemma 1.1 implies a has a unique inverse when $gcd(a, n) = 1$:

Theorem 1.2:
If $gcd(a, n) = 1$, then there exists an integer x, $0 < x < n$, such that $ax \bmod n = 1$.

Proof:
Because the set $\{ai \bmod n\}_{i=0,\ldots,n-1}$ is a permutation of $\{0, 1, \ldots, n-1\}$, $x = i$, where $ai \bmod n = 1$, is a solution. ■

Theorem 1.2 shows the existence of an inverse, but does not give an algorithm for finding it. We shall now review some additional properties of congruences related

to reduced residues and the Euler totient function, showing how these properties lead to the construction of an algorithm for computing inverses.

The **reduced set of residues** mod n is the subset of residues $\{0, \ldots, n-1\}$ relatively prime to n.† For example, the reduced set of residues mod 10 is $\{1, 3, 7, 9\}$. If n is prime, the reduced set of residues is the set of $n-1$ elements $\{1, 2, \ldots, n-1\}$; that is, it is the complete set of residues except for 0. Note that 0 is never included in the reduced set of residues.

The **Euler totient function** $\phi(n)$ is the number of elements in the reduced set of residues modulo n. Equivalently, $\phi(n)$ is the number of positive integers less than n that are relatively prime to n.

For a given modulus n, let $\{r_1, \ldots, r_{\phi(n)}\}$ be the reduced set of residues. The reduced set of residues has the property that every integer relatively prime to n is congruent modulo n to some member r_j of the set. Thus if $gcd(a, n) = 1$ for some integer a, then for each r_i $[1 \leq i \leq \phi(n)]$, $gcd(ar_i, n) = 1$ and $ar_i \bmod n = r_j$ for some r_j. By the same reasoning as for a complete set of residues, the set $\{ar_i \bmod n\}_{i=1,\ldots,\phi(n)}$ is, therefore, a permutation of $\{r_1, \ldots, r_{\phi(n)}\}$.

For prime p, the number of integers less than p that are relatively prime to p is trivially given by $\phi(p) = p - 1$. For the product of two primes p and q we have:

Theorem 1.3:
For $n = pq$ and p, q prime,

$$\phi(n) = \phi(p)\phi(q) = (p-1)(q-1) .$$

Proof:
Consider the complete set of residues modulo n: $\{0, 1, \ldots, pq-1\}$. All of these residues are relatively prime to n except for the $p-1$ elements $\{q, 2q, \ldots, (p-1)q\}$, the $q-1$ elements $\{p, 2p, \ldots, (q-1)p\}$, and 0. Therefore,

$$\begin{aligned}\phi(n) &= pq - [(p-1) + (q-1) + 1] = pq - p - q + 1 \\ &= (p-1)(q-1) . \quad \blacksquare\end{aligned}$$

Example:
Let $p = 3$ and $q = 5$. Then $\phi(15) = (3-1)(5-1) = 2 * 4 = 8$, and there are 8 elements in the reduced set of residues modulo 15: $\{1, 2, 4, 7, 8, 11, 13, 14\}$. ■

In general, for arbitrary n, $\phi(n)$ is given by

$$\phi(n) = \prod_{i=1}^{t} p_i^{e_i - 1}(p_i - 1) ,$$

where

†Strictly speaking, any complete set of residues relatively prime to n is a reduced set of residues; here we are only interested in the set of residues in the range $[1, n-1]$.

$$n = p_1^{e_1} p_2^{e_2} \cdots p_t^{e_t}$$

is the prime factorization of n (i.e., the p_i are distinct primes, and e_i gives the number of occurrences of p_i).

Example:
For $n = 24 = 2^3 3^1$,

$$\phi(24) = 2^2(2 - 1)3^0(3 - 1) = 8;$$

the reduced set of residues is $\{1, 5, 7, 11, 13, 17, 19, 23\}$. ■

The following are two important results of number theory:

Theorem 1.4. Fermat's Theorem:
Let p be prime. Then for every a such that $gcd(a, p) = 1$,

$$a^{p-1} \bmod p = 1 .$$

The proof follows from Euler's generalization: ■

Theorem 1.5. Euler's Generalization:
For every a and n such that $gcd(a, n) = 1$,

$$a^{\phi(n)} \bmod n = 1 .$$

Proof:
Let $\{r_1, \ldots, r_{\phi(n)}\}$ be the reduced set of residues modulo n such that $0 < r_i < n$ for $1 \leq i \leq \phi(n)$. Then $\{ar_1 \bmod n, \ldots, ar_{\phi(n)} \bmod n\}$ is a permutation of $\{r_1, \ldots, r_{\phi(n)}\}$. Therefore,

$$\prod_{i=1}^{\phi(n)} (ar_i \bmod n) = \prod_{i=1}^{\phi(n)} r_i$$

giving

$$(a^{\phi(n)} \bmod n) \prod_{i=1}^{\phi(n)} r_i = \prod_{i=1}^{\phi(n)} r_i ,$$

which, by cancellation, implies $a^{\phi(n)} \bmod n = 1$. ■

Euler's generalization of Fermat's theorem gives us an algorithm for solving an equation

$$ax \bmod n = 1 ,$$

where $gcd(a, n) = 1$. This solution is given by

$$x = a^{\phi(n)-1} \bmod n . \qquad \textbf{(1.5)}$$

If n is prime, this is simply

$$x = a^{(n-1)-1} \bmod n = a^{n-2} \bmod n .$$

Example:
Let $a = 3$ and $n = 7$. Then

$$x = 3^5 \bmod 7 ,$$

which we saw earlier is 5. This checks, because $3 * 5 \bmod 7 = 1$. ■

If $\phi(n)$ is known, the inverse x of a (mod n) can be computed using Eq. (1.5) and the algorithm *fastexp* given in Figure 1.20. Alternatively, x can be computed using an extension of Euclid's algorithm for computing the greatest common divisor [Knut69]. With this approach, it is not necessary to know $\phi(n)$.

An iterative version of Euclid's algorithm is given in Figure 1.21. The algorithm is extended to solve for x as shown in Figure 1.22. The algorithm computes $gcd(a, n)$ by computing $g_{i+1} = g_{i-1} \bmod g_i$ for $i = 1, 2, \ldots$ until $g_i = 0$, where $g_0 = n$, $g_1 = a$, and "$g_i = u_i n + v_i a$" is the loop invariant. When $g_i = 0$, $g_{i-1} = gcd(a, n)$. If $gcd(a, n) = 1$, then $g_{i-1} = 1$ and $v_{i-1}a - 1 = u_{i-1}n$, giving $v_{i-1}a \equiv_n 1$. Thus, $x = v_{i-1}$ is an inverse of a mod n. Now, x will be in the range $-n < x < n$. If x is negative, $x + n$ gives the solution in the range $0 < x < n$. Knuth shows the average number of divisions performed by the algorithm is approximately (.843 ln (n) + 1.47).

Example:
The following illustrates the execution of the algorithm to solve the equation "$3x \bmod 7 = 1$":

i	g_i	u_i	v_i	y
0	7	1	0	
1	3	0	1	2
2	1	1	−2	3
3	0			

Because $v_2 = -2$ is negative, the solution is $x = -2 + 7 = 5$. ■

Note that the implementation of the algorithm can use three local variables: *glast*, *g*, and *gnext* to represent g_{i-1}, g_i, and g_{i+1}, respectively, for all i (similarly for v). Note also that the variable u is not essential to the computation and, therefore, can be omitted. We have presented the algorithm in the preceding form for clarity.

The algorithm is easily extended to find solutions to general equations

$$ax \bmod n = b$$

when $gcd(a, n) = 1$. First, the solution x_0 to "$ax \bmod n = 1$" is found. Now

$$ax_0 \bmod n = 1 \text{ implies } abx_0 \bmod n = b ,$$

whence

$$x = bx_0 \bmod n$$

FIGURE 1.21 Euclid's algorithm for computing the greatest common divisor.

Algorithm *gcd(a, n)*

```
begin
  g_0 := n;
  g_1 := a;
  i   := 1;
  while g_i ≠ 0 do
    begin
      g_{i+1} := g_{i-1} mod g_i;
      i := i + 1
    end;
  gcd := g_{i-1}
end
```

FIGURE 1.22 Euclid's algorithm extended to compute inverses.

Algorithm *inv (a, n)*

```
begin "Return x such that ax mod n = 1, where 0 < a < n"
  g_0 := n; g_1 := a;
  u_0 := 1; v_0 := 0;
  u_1 := 0; v_1 := 1;
  i   := 1;
  while g_i ≠ 0 do "g_i = u_i n + v_i a"
    begin
      y     := g_{i-1} div g_i;
      g_{i+1} := g_{i-1} - y * g_i;
      u_{i+1} := u_{i-1} - y * u_i;
      v_{i+1} := v_{i-1} - y * v_i;
      i     := i + 1
    end;
  x := v_{i-1};
  if x ≥ 0 then inv := x else inv := x + n
end
```

is the unique solution to "$ax \bmod n = b$" in the range $[1, n-1]$. This leads to the formulas:

Solve "$ax \bmod n = b$" when $gcd(a, n) = 1$:

$$x = [b * inv(a, n)] \bmod n \tag{1.6}$$

$$x = [b * fastexp(a, \phi(n) - 1, n)] \bmod n \tag{1.7}$$

If $gcd(a, n) \neq 1$, the equation "$ax \bmod n = b$" will either have no solution or will have more than one solution in the range $[1, n-1]$ as described by the following theorem.

Theorem 1.6:
Let $g = gcd(a, n)$. If $g \mid b$ (i.e., $b \bmod g = 0$) the equation

$$ax \bmod n = b$$

will have g solutions of the form

$$x = \left[\left(\frac{b}{g}\right)x_0 + t\left(\frac{n}{g}\right)\right] \bmod n \qquad \text{for } t = 0, \ldots, g - 1 ,$$

where x_0 is the solution to

$$\left(\frac{a}{g}\right)x \bmod \left(\frac{n}{g}\right) = 1 ;$$

otherwise it will have no solution.

Proof:
If "$ax \bmod n = b$" has a solution in the range $[1, n - 1]$, then $n \mid (ax - b)$. Because $g \mid n$ and $g \mid ax$, it follows that $g \mid b$ must also hold. Now, the equation

$$\left(\frac{a}{g}\right)x \bmod \left(\frac{n}{g}\right) = 1$$

has a unique solution x_0 in the range $[1, (n/g) - 1]$. This implies that $x_1 = (b/g)x_0 \bmod (n/g)$ is a solution of

$$\left(\frac{a}{g}\right)x \bmod \left(\frac{n}{g}\right) = \left(\frac{b}{g}\right)$$

in the range $[1, (n/g) - 1]$. Therefore, $(a/g)x_1 - (b/g) = kn$ for some integer k. Multiplying by g we get $ax_1 - b = kn$, which implies that x_1 is a solution of "$ax \bmod n = b$". Now, any x in the range $[1, n - 1]$ such that $x \equiv_{(n/g)} x_1$ is also a solution of "$ax \bmod n = b$". Therefore, all solutions of "$ax \bmod n = b$" are given by

$$x = x_1 + t\left(\frac{n}{g}\right) , \quad t = 0, \ldots, g - 1 . \quad \blacksquare$$

Example:
Consider the equation

$$6x \bmod 10 = 4.$$

Because $g = gcd(6, 10) = 2$, and 2 divides 4, there are 2 solutions. We first compute the solution x_0 to the equation

$$\left(\frac{6}{2}\right)x \bmod \left(\frac{10}{2}\right) = 1,$$

that is,

$$3x \bmod 5 = 1$$

getting $x_0 = 2$. This gives

$$x_1 = \left(\frac{4}{2}\right)2 \bmod \left(\frac{10}{2}\right) = 4 \bmod 5 = 4 .$$

The solutions are thus:

$$t = 0\text{: } x = 4$$

$$t = 1\text{: } x = [4 + \left(\frac{10}{2}\right)] \bmod 10 = 9 . \quad \blacksquare$$

Figure 1.23 gives an algorithm for printing the solutions described by Theorem 1.6. Division is denoted by "/" rather than "div" where the numerator is evenly divisible by the denominator.

An equation of the form "$ax \bmod n = b$" may also be solved using the prime factorization of n. Let

$$n = d_1 d_2 \ldots d_t$$

be the prime factorization of n, where

$$d_i = p_i^{e_i} \qquad (1 \leq i \leq t)$$

and the p_i are distinct primes. Thus, the d_i are pairwise relatively prime.

Let $f(x)$ be a polynomial in x. The following theorem shows that x is a solution to the equation $f(x) \bmod n = 0$ if and only if x is a common solution to the set of equations $f(x) \bmod d_i = 0$ for $i = 1, \ldots, t$.

Theorem 1.7:
Let $d_1, \ldots, d_t$ be pairwise relatively prime, and let $n = d_1 d_2 \ldots d_t$. Then

FIGURE 1.23 Solve linear equations.

Algorithm *solve* (*a, n, b*)

```
begin "print all solutions x to ax mod n = b"
  g := gcd(a, n);
  if (b mod g) = 0
    then begin
      print(g, "solutions follow");
      n0 := n/g;
      x0 := inv(a/g, n0);
      x1 := ((b/g) * x0) mod n;
      for t := 0 to g - 1 do
        begin
          x := (x1 + t * n0) mod n;
          print(x)
        end
      end
    else print("no solutions exist")
end
```

$f(x) \bmod n = 0$ if and only if
$f(x) \bmod d_i = 0 \quad (1 \leq i \leq t)$.

Proof:
Because the d_i are pairwise relatively prime, $n \mid f(x)$ if and only if $d_i \mid f(x)$ for $i = 1, \ldots, t$. ■

We can apply this result to solve equations of the form

$ax \bmod n = b$.

Writing this as $(ax - b) \bmod n = 0$, we find a common solution to the equations $(ax - b) \bmod d_i = 0$ or, equivalently, to the equations

$$ax \bmod d_i = b \bmod d_i \qquad (1 \leq i \leq t) .$$

We can construct a common solution x to $f(x) \bmod d_i = 0$ $(1 \leq i \leq t)$ from a set of independent solutions $x_1, \ldots, x_t$, where x_i is a solution to $f(x) \bmod d_i = 0$. Observe that every x congruent to x_i modulo d_i is a solution to $f(x) \bmod d_i = 0$. Therefore, x is a solution to $f(x) \bmod n = 0$ if $x \bmod d_i = x_i$ for $i = 1, \ldots, t$. The Chinese Remainder Theorem shows how a common solution x to the preceding system of equations can be computed.

Theorem 1.8. Chinese Remainder Theorem:
Let $d_1, \ldots, d_t$ be pairwise relatively prime, and let $n = d_1 d_2 \ldots d_t$. Then the system of equations

$$(x \bmod d_i) = x_i \qquad (i = 1, \ldots, t)$$

has a common solution x in the range $[0, n - 1]$.

Proof:
Now, for each $i = 1, \ldots, t$, $gcd(d_i, n/d_i) = 1$. Therefore, there exist y_i such that $(n/d_i)y_i \bmod d_i = 1$. Furthermore, $(n/d_i)y_i \bmod d_j = 0$ for $j \neq i$ because d_j is a factor of (n/d_i). Let

$$x = \left[\sum_{i=1}^{t} \left(\frac{n}{d_i}\right) y_i x_i \right] \bmod n .$$

Then x is a solution of "$x \bmod d_i = x_i$" $(1 \leq i \leq t)$ because

$$x \bmod d_i = \left(\frac{n}{d_i}\right) y_i x_i \bmod d_i = x_i . \quad ■$$

Example:
We shall show how the Chinese Remainder Theorem can be used to solve the equation "$3x \bmod 10 = 1$". We observe that $10 = 2 * 5$, so $d_1 = 2$ and $d_2 = 5$. We first find solutions x_1 and x_2, respectively, to the equations:

FIGURE 1.24 Find solution to system of equations using the Chinese Remainder Theorem.

Algorithm $crt(n, d_1, \ldots, d_t, x_1, \ldots, x_t)$

```
begin "return x ∈ [0, n − 1] such that x mod d_i = x_i (1 ≤ i ≤ t)"
  for i := 1 to t do
    y_i := inv ((n/d_i) mod d_i, d_i);
  x := 0;
  for i := 1 to t do
    x := [x + (n/d_i) * y_i * x_i] mod n;
  crt := x
end
```

$$3x \bmod 2 = 1$$
$$3x \bmod 5 = 1 .$$

This gives us $x_1 = 1$ and $x_2 = 2$. We then apply the Chinese Remainder Theorem to find a common solution x to the equations:

$$x \bmod 2 = x_1 = 1$$
$$x \bmod 5 = x_2 = 2 .$$

We find y_1 and y_2 such that

$$\left(\frac{10}{2}\right) y_1 \bmod 2 = 1, \text{ and}$$
$$\left(\frac{10}{5}\right) y_2 \bmod 5 = 1,$$

getting $y_1 = 1$ and $y_2 = 3$. We then have

$$x = \left[\left(\frac{10}{2}\right) y_1 x_1 + \left(\frac{10}{5}\right) y_2 x_2\right] \bmod 10$$
$$= [5 * 1 * 1 + 2 * 3 * 2] \bmod 10 = 7$$

Thus 7 is the inverse of 3 (mod 10). ■

An algorithm that computes the solution given by the Chinese Remainder Theorem is given in Figure 1.24. Note that an implementation of the algorithm can combine the two **for** loops and use a single local variable to represent the y_i. We have presented the algorithm this way to show its relation to the proof of Theorem 1.8.

1.6.3 Computing in Galois Fields

When the modulus is a prime p, every integer $a \in [1, p - 1]$ is relatively prime to p and, therefore, has a unique multiplicative inverse mod p. This means the set of

integers mod p, together with the arithmetic operations, is a **finite field** †, called the **Galois field** $\mathbf{GF}(p)$ after their discoverer Evariste Galois. Because division is possible, arithmetic mod p is more powerful than ordinary integer arithmetic. Real arithmetic is not generally applicable to cryptography because information is lost through round-off errors (the same holds for integer division, where information is lost through truncation). Many of the ciphers developed in recent years are based on arithmetic in $\mathbf{GF}(p)$, where p is a large prime.

Another type of Galois field with applications in cryptography is based on arithmetic mod q over polynomials of degree n. These fields, denoted $\mathbf{GF}(q^n)$, have elements that are polynomials of degree $n - 1$ (or lower) of the form

$$a = a_{n-1}x^{n-1} + \cdots + a_1x + a_0 \ ,$$

where the coefficients a_i are integers mod q. Each element a is a residue mod $p(x)$, where $p(x)$ is an irreducible polynomial of degree n (i.e., p cannot be factored into polynomials of degree less than n).

Arithmetic on the coefficients of the polynomials is done mod q. For example, the coefficients c_i in the sum $c = a + b$ are given by $c_i = (a_i + b_i) \bmod q$ $(0 \leq i < n)$. Because a and b are of degree $n - 1$, the sum $a + b$ is a polynomial of degree at most $n - 1$, whence it is already reduced mod $p(x)$. The product $a * b$ could be of degree greater than $n - 1$ (but at most $2n - 2$), however, so it must be reduced mod $p(x)$; this is done by dividing by $p(x)$ and taking the remainder.

Of particular interest in computer applications are the fields $\mathbf{GF}(2^n)$. Here the coefficients of the polynomials are the binary digits 0 and 1. Thus, an element a can be represented as a bit vector $(a_{n-1}, \ldots, a_1, a_0)$ of length n, and each of the possible 2^n bit vectors of length n corresponds to a different element in $\mathbf{GF}(2^n)$. For example, the bit vector 11001‡ corresponds to the polynomial $(x^4 + x^3 + 1)$ in $\mathbf{GF}(2^5)$. To avoid confusion with the notation $\mathbf{GF}(p)$, where p is a prime number, we shall not write $\mathbf{GF}(32)$ for $\mathbf{GF}(2^5)$, for example, even though 32 is not prime.

Computing in $\mathbf{GF}(2^n)$ is more efficient in both space and time than computing in $\mathbf{GF}(p)$. Let p be a prime number such that $2^{n-1} < p < 2^n$, whence the elements of $\mathbf{GF}(p)$ are also represented as bit vectors of length n (using the standard binary representation of the positive integers; e.g., 11001 corresponds to the integer $2^4 + 2^3 + 1 = 25$). We first observe that whereas all 2^n bit vectors correspond to elements of $\mathbf{GF}(2^n)$, this is not true for $\mathbf{GF}(p)$; in particular, the bit vectors representing the integers in the range $[p, 2^n - 1]$ are not elements of $\mathbf{GF}(p)$. Thus it is possible to represent more elements (up to twice as many!) in $\mathbf{GF}(2^n)$ than in $\mathbf{GF}(p)$ using the same amount of space. This can be important in cryptography applications, where the strength of a scheme usually depends on the size of the field. For comparable levels of security, $\mathbf{GF}(2^n)$ is more efficient in terms of space than $\mathbf{GF}(p)$.

We next observe that arithmetic is more efficient in $\mathbf{GF}(2^n)$ than in $\mathbf{GF}(p)$.

† A field is any integral domain in which every element besides 0 has a multiplicative inverse; the rational numbers form an infinite field.

‡ To simplify our notation, we shall write bit vectors as strings here and elsewhere in the book.

To see why, we shall briefly describe how the arithmetic operations are implemented in $\mathbf{GF}(2^n)$. We assume the reader has a basic understanding of integer arithmetic in digital computers.

We first consider operations over the binary coefficients of the polynomials. Recall that these operations are performed mod 2. Let u and v be binary digits. Then u and v can be added simply by taking the "exclusive-or" $u \oplus v$; that is,

$$(u + v) \bmod 2 = u \oplus v = \begin{cases} 0 & \text{if } u = v \text{ (both bits the same)} \\ 1 & \text{if } u \neq v . \end{cases}$$

Subtraction is the same:

$$\begin{aligned}(u - v) \bmod 2 &= (u + v - 2) \bmod 2 = (u + v) \bmod 2 \\ &= u \oplus v .\end{aligned}$$

The bits u and v can be multiplied by taking the boolean "and":

$$u * v = u \text{ and } v .$$

Now, let a and b be the bit vectors $a = (a_{n-1}, \ldots, a_0)$ and $b = (b_{n-1}, \ldots, b_0)$. In $\mathbf{GF}(2^n)$, a and b are added (or subtracted) by taking the $\oplus$ of each pair of bits. Letting $c = a + b$ (or $a - b$), we have $c = (c_{n-1}, \ldots, c_0)$, where

$$c_i = a_i \oplus b_i \quad \text{for } i = 0, \ldots, n - 1 .$$

The operator $\oplus$ is extended pairwise to bit strings, so we can also write $c = a \oplus b$ to denote the sum (or difference) of a and b.

Example:
Let $a = 10101$ and $b = 01100$. In $\mathbf{GF}(2^5)$, $c = a + b$ is computed as follows:

$$\begin{array}{l} a = 1\,0\,1\,0\,1 \\ b = 0\,1\,1\,0\,0 \\ c = 1\,1\,0\,0\,1 . \end{array}$$

By contrast, if we add the bit vectors a and b in $\mathbf{GF}(p)$ for $p = 31$, we must perform carries during the addition, and then divide to reduce the result mod 31:

Step 1. Add a and b:

$$\begin{array}{lll} a = & 1\,0\,1\,0\,1 & (21) \\ b = & 0\,1\,1\,0\,0 & (12) \\ c = 1 & 0\,0\,0\,0\,1 & (33) \end{array}$$

Step 2. Divide by 31 and keep the remainder:

$$c = \quad 0\,0\,0\,1\,0 \quad (\ 2) . \ \blacksquare$$

Multiplication of a and b in $\mathbf{GF}(2^n)$ is also easier than in $\mathbf{GF}(p)$. Here, however, the product $a * b$ must be divided by the irreducible polynomial $p(x)$ associated with the field. The product $d = a * b$ is represented by the polynomial sum:

$$d = \sum_{i=0}^{n-1} (a_i * b)x^i \bmod p(x) ,$$

where

$$a_i * b = \begin{cases} b = b_{n-1}x^{n-1} + \cdots + b_0 & \text{if } a_i = 1 \\ 0 \quad \text{otherwise} . \end{cases}$$

Example:
Let $a = 101$. If a is squared in **GF**(2^3) with irreducible polynomial $p(x) = x^3 + x + 1$ (1011 in binary), the product $d = a * a$ is computed as follows:

Step 1. Multiply $a * a$:

$$\begin{array}{r} 1\,0\,1 \\ 1\,0\,1 \\ \hline 1\,0\,1 \\ 0\,0\,0 \\ 1\,0\,1 \\ \hline 1\,0\,0\,0\,1 \end{array}$$

Step 2. Divide by $p(x) = 1011$:

$$\begin{array}{r} \quad\;\; 1\,0 \\ 1\,0\,1\,1 \overline{)\,1\,0\,0\,0\,1} \\ \quad\;\; 1\,0\,1\,1 \\ \hline \quad\;\; 1\,1\,1 = d . \end{array}$$

If a is squared in **GF**(p) for $p = 7$, the computation is similar, except the additions and subtractions in the multiply and divide steps require carries. ■

Example:
Let $a = 111$ and $b = 100$. The product $d = a * b$ is computed in **GF**(2^3) with irreducible polynomial $p(x) = 1011$ ($x^3 + x + 1$) as follows:

Step 1. Multiply $a * b$:

$$\begin{array}{r} 1\,1\,1 \\ 1\,0\,0 \\ \hline 0\,0\,0 \\ 0\,0\,0 \\ 1\,1\,1 \\ \hline 1\,1\,1\,0\,0 \end{array}$$

Step 2. Divide by $p(x) = 1011$:

$$\begin{array}{r} \quad\;\; 1\,1 \\ 1\,0\,1\,1 \overline{)\,1\,1\,1\,0\,0} \\ \quad\;\; 1\,0\,1\,1 \\ \hline \quad\;\; 1\,0\,1\,0 \\ \quad\;\; 1\,0\,1\,1 \\ \hline \quad\;\; 1 = d . \end{array}$$

Thus, 111 * 100 mod 1011 = 001, so 111 and 100 are inverses mod 1011 in **GF**(2^3) . ■

To divide b by a in **GF**(2^n) with modulus $p(x)$, we compute the inverse of a mod $p(x)$, denoted a^{-1}, and multiply b by a^{-1}. Because the algorithms developed in the preceding section for computing inverses apply to any finite field, we can apply them to compute inverses in **GF**(2^n). To do this, we observe that every bit vector of length n except for the 0-vector is relatively prime to $p(x)$ regardless of the irreducible polynomial $p(x)$. Thus, the number of residues relatively prime to $p(x)$ is given by $\phi(p(x)) = 2^n - 1$, where we have extended the meaning of the Euler totient function ϕ to polynomials. We can then use Eq. (1.5) to compute a^{-1}, where $a * a^{-1} \bmod p(x) = 1$, getting

$$a^{-1} = a^{\phi(p(x))-1} \bmod p(x) = a^{2^n-2} \bmod p(x) .$$

Alternatively, we can compute a^{-1} using the extended version of Euclid's algorithm shown in Figure 1.22:

$$a^{-1} = inv(a, p(x)) ,$$

where arithmetic is done in **GF**(2^n) .

Example:
Let a = 100 (x^2) and $p(x)$ = 1011 in **GF**(2^3).
The reader should verify that

$$\begin{aligned} a^{-1} &= 100^{2^3-2} \bmod 1011 = 100^6 \bmod 1011 \\ &= 111 , \end{aligned}$$

and

$$\begin{aligned} a^{-1} &= inv(100, 1011) \\ &= 111 . \end{aligned}$$ ■

(Davida [Davi72] describes an algorithm for computing inverses that is suitable for parallel implementation.)

To summarize, polynomial arithmetic in **GF**(2^n) is more efficient than integer arithmetic in **GF**(p) because there are no carries, and division by the modulus is never needed for addition or subtraction.

The cost of hardware (or software) to compute in **GF**(2^n) depends somewhat on the choice of modulus. Blakley (Blak80) shows how multiplication can be efficiently implemented in **GF**(2^n) with an irreducible trinomial of the form

$$p(x) = x^n + x + 1 .$$

The polynomial $p(x) = x^3 + x + 1$ in our examples is of this form. Most such trinomials are not irreducible; the following is a list of all irreducible ones through n = 127:

$$n = 1, 3, 4, 6, 7, 9, 15, 22, 28, 30, 46, 60, 63, 127.$$

(See, for example, [Zier68,Zier69] for a list of irreducible trinomials for $n > 127$.)

Multiplication is efficient when $p(x)$ is of this form, because the long string of 0's in $p(x)$ simplifies the reduction mod $p(x)$. To see how this works, let $d = a * b$, where $a = (a_{n-1}, \ldots, a_0)$ and $b = (b_{n-1}, \ldots, b_0)$. Before reduction mod $p(x)$, the product is given by the $(2n - 1)$-bit vector

$$(s_{n-1}, \ldots, s_2, s_1, c_{n-1}, \ldots, c_1, c_0),$$

where:

$$c_i = a_0 * b_i \oplus a_1 * b_{i-1} \oplus \ldots \oplus a_i * b_0, \qquad i = 0, \ldots, n-1$$

$$s_i = a_i * b_{n-1} \oplus a_{i+1} * b_{n-2} \oplus \ldots \oplus a_{n-1} * b_i, \qquad i = 1, \ldots, n-1.$$

This is illustrated next:

$$\begin{array}{cccccccc}
 & & & & b_{n-1} & \ldots & b_1 & b_0 \\
 & & & & a_{n-1} & \ldots & a_1 & a_0 \\
\hline
 & & & & a_0 b_{n-1} & \ldots & a_0 b_1 & a_0 b_0 \\
 & & & a_1 b_{n-1} & a_1 b_{n-2} & \ldots & a_1 b_0 & \\
 & & a_2 b_{n-1} & a_2 b_{n-2} & a_2 b_{n-3} & \ldots & & \\
 & \cdot & \cdot & \cdot & \cdot & \cdot & & \\
\cdot & \cdot & \cdot & \cdot & \cdot & & & \\
a_{n-1} b_{n-1} & \ldots & a_{n-1} b_2 & a_{n-1} b_1 & a_{n-1} b_0 & & & \\
\hline
s_{n-1} & \ldots & s_2 & s_1 & c_{n-1} & \ldots & c_1 & c_0
\end{array}.$$

Reducing by $p(x)$, we see that if bit $s_{n-1} = 1$, bit c_{n-1} and c_{n-2} are complemented; if bit $s_{n-2} = 1$, bits c_{n-2} and c_{n-3} are complemented; and so forth, giving the reduced product

$$d = (d_{n-1}, \ldots, d_0)$$

where:

$$d_{n-1} = c_{n-1} \oplus s_{n-1}$$

$$d_i = c_i \oplus s_i \oplus s_{i+1} \quad (0 < i < n-1)$$

$$d_0 = c_0 \oplus s_1.$$

Berkovits, Kowalchuk, and Schanning [Berk79] have developed a fast implementation of $\mathbf{GF}(2^{127})$ arithmetic using the previous approach with $p(x) = x^{127} + x + 1$.

The fields $\mathbf{GF}(2^n)$ are used in many error correcting codes, including "Hamming codes". These codes can be efficiently implemented using shift registers with feedback. (For more information on coding theory, see [Hamm80,Berl68, MacW78,Pete72,McEl77].) A cryptographic application for shift registers is described in Chapter 3.

EXERCISES

1.1 Decipher the following Caesar cipher using $K = 3$:

VRVRVHDPVWUHVV .

1.2 Let X be an integer variable represented with 32 bits. Suppose that the probability is $1/2$ that X is in the range $[0, 2^8 - 1]$, with all such values being equally likely, and $1/2$ that X is in the range $[2^8, 2^{32} - 1]$, with all such values being equally likely. Compute $H(X)$.

1.3 Let X be one of the six messages: A, B, C, D, E, F, where:

$$p(\mathrm{A}) = p(\mathrm{B}) = p(\mathrm{C}) = \frac{1}{4}$$

$$p(\mathrm{D}) = \frac{1}{8}$$

$$p(\mathrm{E}) = p(\mathrm{F}) = \frac{1}{16} .$$

Compute $H(X)$ and find an optimal binary encoding of the messages.

1.4 Prove that for $n = 2$, $H(X)$ is maximal for $p_1 = p_2 = 1/2$.

1.5 Prove that for any n, $H(X)$ is maximal for $p_i = 1/n$ $(1 \leq i \leq n)$.

1.6 Show that $H(X, Y) \leq H(X) + H(Y)$, where

$$H(X, Y) = \sum_{X, Y} p(X, Y) \log_2 \left(\frac{1}{p(X, Y)}\right) .$$

[*Hint:* $p(X) = \sum_{Y} p(X, Y)$, $p(Y) = \sum_{X} p(X, Y)$, and $p(X)p(Y) \leq p(X, Y)$, equality holding only when X and Y are independent.]

1.7 Show that $H(X, Y) = H_Y(X) + H(Y)$. Combine this with the result in the previous problem to show that $H_Y(X) \leq H(X)$; thus, the uncertainty about X cannot increase with the additional information Y. [*Hint:* $p(X, Y) = p_Y(X)p(Y)$.]

1.8 Let M be a secret message revealing the recipient of a scholarship. Suppose there was one female applicant, Anne, and three male applicants, Bob, Doug, and John. The probability of each applicant receiving the scholarship is given by: $p(\text{Anne}) = 1/2$ $p(\text{Bob}) = p(\text{Doug}) = p(\text{John}) = 1/6$. Compute $H(M)$. Letting S denote a message revealing the sex of the recipient, compute $H_S(M)$.

1.9 Let M be a 6-digit number in the range $[0, 10^6 - 1]$ enciphered with a Caesar-type shifted substitution cipher with key K, $0 \leq K \leq 9$. For example, if $K = 1$, $M = 123456$ is enciphered as 234567. Compute $H(M)$, $H(C)$, $H(K)$, $H_C(M)$, and $H_C(K)$, assuming all values of M and K are equally likely.

1.10 Consider the following ciphertexts:

XXXXX

VWXYZ
RKTIC
JZQAT

Which of the these ciphertexts could result from enciphering five-letter words of English using:

a. A substitution cipher, where each letter is replaced with some other letter, but the letters are not necessarily shifted as in the Caesar cipher (thus A could be replaced with K, B with W, etc.).
b. Any transposition cipher.

1.11 Suppose plaintext messages are 100 letters long and that keys are specified by sequences of letters. Explain why perfect secrecy can be achieved with keys fewer than 100 letters long. How long must the keys be for perfect secrecy?

1.12 Compare the redundancy of programs written in different languages (e.g., PASCAL, APL, and COBOL). Suggest ways in which the redundancy can be reduced (and later recovered) from a program.

1.13 Show that the cancellation law does not hold over the integers mod n with multiplication when n is not prime by showing there exist integers x and y such that $x * y \bmod n = 0$, but neither x nor y is 0.

1.14 Let n be an integer represented in base 10 as a sequence of t decimal digits $d_1\ d_2 \ldots d_t$. Prove that

$$n \bmod 9 = \left(\sum_{i=1}^{t} d_i \right) \bmod 9 .$$

1.15 For each equation of the form $(ax \bmod n = b)$ that follows, solve for x in the range $[0, n-1]$.

a. $5x \bmod 17 = 1$
b. $19x \bmod 26 = 1$
c. $17x \bmod 100 = 1$
d. $17x \bmod 100 = 10$

1.16 Find all solutions to the equation

$$15x \bmod 25 = 10$$

in the range $[0, 24]$.

1.17 Apply algorithm *crt* in Figure 1.24 to find an $x \in [0, 59]$ such that

$$x \bmod 4 = 3$$
$$x \bmod 3 = 2$$
$$x \bmod 5 = 4 .$$

1.18 Find a solution to the equation

$$13x \bmod 70 = 1$$

by finding x_1, x_2, and x_3 such that

$$13x_1 \bmod 2 = 1$$
$$13x_2 \bmod 5 = 1$$
$$13x_3 \bmod 7 = 1$$

and then applying algorithm *crt*.

1.19 Let $a = 100\ (x^2)$ in $\mathbf{GF}(2^3)$ with modulus $p(x) = 1011\ (x^3 + x + 1)$. Divide 1000000000000 by 1011 to show that

$$a^{-1} = 100^6 \bmod 1011 = 111\ .$$

Using algorithm *inv*, show that

$$a^{-1} = inv(100, 1011) = 111\ .$$

1.20 Find the inverse of $a = 011\ (x + 1)$ in $\mathbf{GF}(2^3)$ with $p(x) = 1011$.

REFERENCES

Aho74. Aho, A., Hopcroft, J., and Ullman, J., *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, Mass. (1974).

Baye76. Bayer, R. and Metzger, J. K., "On the Encipherment of Search Trees and Random Access Files," *ACM Trans. On Database Syst.* Vol. 1(1) pp. 37–52 (Mar. 1976).

Berk79. Berkovits, S., Kowalchuk, J., and Schanning, B., "Implementing Public Key Schemes," *IEEE Comm. Soc. Mag.* Vol. 17(3) pp. 2–3 (May 1979).

Berl68. Berlekamp, E. R., *Algebraic Coding Theory,* McGraw-Hill, New York (1968).

Blak80. Blakley, G. R., "One-Time Pads are Key Safeguarding Schemes, Not Cryptosystems," *Proc. 1980 Symp. on Security and Privacy,* IEEE Computer Society, pp. 108–113 (Apr. 1980).

Bras79a. Brassard, G., "Relativized Cryptography," *Proc. IEEE 20th Annual Symp. on Found. of Comp. Sci.,* pp. 383–391 (Oct. 1979).

Bras79b. Brassard, G., "A Note on the Complexity of Cryptography," *IEEE Trans. on Inform. Theory* Vol. IT-25(2) pp. 232–233 (Mar. 1979).

Bras80. Brassard, G., "A Time-Luck Tradeoff in Cryptography," *Proc. IEEE 21st Annual Symp. on Found. of Comp. Sci.,* pp. 380–386 (Oct. 1980).

Cook71. Cook, S. A., "The Complexity of Theorem-Proving Procedures," *Proc. 3rd Annual ACM Symp. on the Theory of Computing,* pp. 151–158 (1971).

Cove78. Cover, T. M. and King, R. C., "A Convergent Gambling Estimate of the Entropy of English," *IEEE Trans. on Infor. Theory* Vol. IT-24 pp. 413–421 (Aug. 1978).

Davi72. Davida, G. I., "Inverse of Elements of a Galois Field," *Electronics Letters* Vol. 8(21) (Oct. 19, 1972).

Deav77. Deavours, C. A., "Unicity Points in Cryptanalysis," *Cryptologia* Vol. 1(1) pp. 46–68 (Jan. 1977).

Diff76. Diffie, W. and Hellman, M., "New Directions in Cryptography," *IEEE Trans. on Info. Theory* Vol. IT-22(6) pp. 644–654 (Nov. 1976).

Gare79. Garey, M. R. and Johnson, D. S., *Computers and Intractability, A Guide to the Theory of NP-Completeness,* W. H. Freeman and Co., San Francisco, Calif. (1979).

Hamm80. Hamming, R. W., *Coding and Information Theory,* Prentice-Hall, Englewood Cliffs, N.J. (1980).

Hell77. Hellman, M. E., "An Extension of the Shannon Theory Approach to Cryptography," *IEEE Trans. on Info. Theory* Vol. IT-23 pp. 289–294 (May 1977).

Huff52. Huffman, D., "A Method for the Construction of Minimum-Redundancy Codes," *Proc. IRE* Vol. 40 pp. 1098–1101 (1952).

Knut69. Knuth, D., *The Art of Computer Programming; Vol. 2, Seminumerical Algorithms,* Addison-Wesley, Reading, Mass. (1969). (Exercise 4.5.2.15.)

Konh81. Konheim, A. G., *Cryptography: A Primer,* John Wiley & Sons, New York (1981).

Lemp79. Lempel, A., "Cryptology in Transition," *Computing Surveys* Vol. 11(4) pp. 285–303 (Dec. 1979).

LeVe77. LeVeque, W. J., *Fundamentals of Number Theory,* Addison-Wesley, Reading, Mass. (1977).

MacW78. MacWilliams, F. J. and Sloane, N. J. A., *The Theory of Error Correcting Codes,* North-Holland, New York (1978).

McEl77. McEliece, R., *The Theory of Information and Coding,* Addison-Wesley, Reading, Mass. (1977).

McEl78. McEliece, R., "A Public Key Cryptosystem Based on Algebraic Coding Theory," DSN Progress Rep. 42–44, Jet Propulsion Lab Calif. Inst. of Tech., Pasadena, Ca. (Jan., Feb. 1978).

Merk80. Merkle, R. C., "Protocols for Public Key Cryptosystems," pp. 122–133 in *Proc. 1980 Symp. on Security and Privacy,* IEEE Computer Society (Apr. 1980).

Mins67. Minsky, M., *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, N.J. (1967).

Need78. Needham, R. M. and Schroeder, M., "Using Encryption for Authentication in Large Networks of Computers," *Comm. ACM* Vol. 21(12) pp. 993–999 (Dec. 1978).

Nive72. Niven, I. and Zuckerman, H. A., *An Introduction to the Theory of Numbers,* John Wiley & Sons, New York (1972).

Pete72. Peterson, W. W. and Weldon, E. J., *Error Correcting Codes,* MIT Press, Cambridge, Mass. (1972).

Pope79. Popek, G. J. and Kline, C. S., "Encryption and Secure Computer Networks," *Computing Surveys* Vol. 11(4) pp. 331–356 (Dec. 1979).

Rabi78. Rabin, M., "Digitalized Signatures," pp. 155–166 in *Foundations of Secure Computation,* ed. R. A. DeMillo et al., Academic Press, New York (1978).

Sham79. Shamir, A., "On the Cryptocomplexity of Knapsack Systems," *Proc. 11th Annual ACM Symp. on the Theory of Computing,* pp. 118–129 (May 1979).

Shan48. Shannon, C. E., "A Mathematical Theory of Communication," *Bell Syst. Tech. J.* Vol. 27 pp. 379–423 (July), 623–656 (Oct.) (1948).

Shan49. Shannon, C. E., "Communication Theory of Secrecy Systems," *Bell Syst. Tech. J.* Vol. 28 pp. 656–715 (Oct. 1949).

Shan51. Shannon, C. E., "Predilection and Entropy of Printed English," *Bell Syst. Tech. J.,* Vol. 30 pp. 50–64 (Jan. 1951).

Simm79. Simmons, G. J., "Symmetric and Asymmetric Encryption," *Computing Surveys* Vol. 11(4) pp. 305–330 (Dec. 1979).

Simm81. Simmons, G. J., "Half a Loaf is Better Than None: Some Novel Message Integrity Problems," *Proc. 1981 Symp. on Security and Privacy,* IEEE Computer Society pp. 65–69, (April 1981).

Smid79. Smid, M., "A Key Notarization System for Computer Networks," NBS Special Pub. 500–54, National Bureau of Standards Washington, D.C. (Oct. 1979).

Turi36. Turing, A., "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proc. London Math. Soc. Ser. 2* Vol. 42, pp. 230–265 and Vol. 43, pp. 544–546 (1936).

Turn73. Turn, R., "Privacy Transformations for Databank Systems," pp. 589–600 in *Proc. NCC,* Vol. 42, AFIPS Press, Montvale, N.J. (1973).

Vino55. Vinogradov, I. M., *An Introduction to the Theory of Numbers,* Pergamon Press, Elmsford, N.Y. (1955).

Zier68. Zierler, N. and Brillhart, J., "On Primitive Trinomials (Mod 2)," *Info. and Control* Vol. 13 pp. 541–554 (1968).

Zier69. Zierler, N. and Brillhart, J., "On Primitive Trinomials (Mod 2)," *Info. and Control* Vol. 14 pp. 566–569 (1969).