interested in preserving the confidentiality of their communications. In the following sections, we'll cover the definition of a cipher and explore several common cipher types that form the basis of modern ciphers. It's important to remember that these concepts seem somewhat basic, but when used in combination, they can be formidable opponents and cause cryptanalysts many hours of frustration.

Codes vs. Ciphers

People often use the words *code* and *cipher* interchangeably, but technically, they aren't interchangeable. There are important distinctions between the two concepts. *Codes*, which are cryptographic systems of symbols that represent words or phrases, are sometimes secret, but they are not necessarily meant to provide confidentiality. A common example of a code is the "10 system" of communications used by law enforcement agencies. Under this system, the sentence "I received your communication and understand the contents" is represented by the code phrase "10-4." This code is commonly known by the public, but it does provide for ease of communication. Some codes are secret. They may convey confidential information using a secret codebook where the meaning of the code is known only to the sender and recipient. For example, a spy might transmit the sentence "The eagle has landed" to report the arrival of an enemy aircraft.

Ciphers, on the other hand, are always meant to hide the true meaning of a message. They use a variety of techniques to alter and/or rearrange the characters or bits of a message to achieve confidentiality. Ciphers convert messages from plaintext to ciphertext on a bit basis (that is, a single digit of a binary code), character basis (that is, a single character of an American Standard Code for Information Interchange (ASCII) message), or block basis (that is, a fixed-length segment of a message, usually expressed in number of bits). The following sections cover several common ciphers in use today.

An easy way to keep the difference between codes and ciphers straight is to remember that codes work on words and phrases, whereas ciphers work on individual characters and bits.

Transposition Ciphers

Transposition ciphers use an encryption algorithm to rearrange the letters of a plaintext message, forming the ciphertext message. The decryption algorithm simply reverses the encryption transformation to retrieve the original message.

In the challenge-response protocol example in Figure 6.1 earlier in this chapter, a simple transposition cipher was used to reverse the letters of the message so that *apple* became *elppa*. Transposition ciphers can be much more complicated than this. For example, you can use a keyword to perform a *columnar transposition*. In the following example, we're attempting to encrypt the message "The fighters will strike the enemy bases at noon" using the secret key *attacker*. Our first step is to take the letters of the keyword and number them in alphabetical order. The first appearance of the letter *A* receives the value 1; the second appearance is numbered 2. The next letter in sequence, *C*, is numbered 3, and so on. This results in the following sequence:

```
A T T A C K E R
1 7 8 2 3 5 4 6
```

Next, the letters of the message are written in order underneath the letters of the keyword:

```
A T T A C K E R
1 7 8 2 3 5 4 6
T H E F I G H T
E R S W I L L S
T R I K E T H E
E N E M Y B A S
E S A T N O O N
```

Finally, the sender enciphers the message by reading down each column; the order in which the columns are read corresponds to the numbers assigned in the first step. This produces the following ciphertext:

```
T E T E E F W K M T I I E Y N H L H A O G L T B O T S E S N H R R
N S E S I E A
```

On the other end, the recipient reconstructs the eight-column matrix using the ciphertext and the same keyword and then simply reads the plaintext message across the rows.

Substitution Ciphers

Substitution ciphers use the encryption algorithm to replace each character or bit of the plaintext message with a different character. The Caesar cipher discussed in the beginning of this chapter is a good example of a substitution cipher. Now that you've learned a little bit about cryptographic math, we'll take another look at the Caesar cipher. Recall that we simply shifted each letter three places to the right in the message to generate the ciphertext. However, we ran into a problem when we got to the end of the alphabet and ran out of letters. We solved this by wrapping around to the beginning of the alphabet so that the plaintext character *Z* became the ciphertext character *C*.

You can express the ROT3 cipher in mathematical terms by converting each letter to its decimal equivalent (where *A* is 0 and *Z* is 25). You can then add three to each plaintext letter to determine the ciphertext. You account for the wrap-around by using the modulo function discussed in the section "Cryptographic Mathematics." The final encryption function for the Caesar cipher is then this:

```
C = (P + 3) mod 26
```

The corresponding decryption function is as follows:

```
P = (C - 3) mod 26
```

As with transposition ciphers, there are many substitution ciphers that are more sophisticated than the examples provided in this chapter. Polyalphabetic substitution ciphers use multiple alphabets in the same message to hinder decryption efforts. One of the most notable examples of a polyalphabetic substitution cipher system is the Vigenère cipher. The Vigenère cipher uses a single encryption/decryption chart, as shown here:

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

Notice that the chart is simply the alphabet written repeatedly (26 times) under the master heading, shifting by one letter each time. You need a key to use the Vigenère system. For example, the key could be *secret*. Then, you would perform the following encryption process:

1. Write out the plaintext.
2. Underneath, write out the encryption key, repeating the key as many times as needed to establish a line of text that is the same length as the plaintext.
3. Convert each letter position from plaintext to ciphertext.
    a. Locate the column headed by the first plaintext character (*a*).
    b. Next, locate the row headed by the first character of the key (*s*).
    c. Finally, locate where these two items intersect, and write down the letter that appears there (*s*). This is the ciphertext for that

letter position.

4. Repeat steps 1 through 3 for each letter in the plaintext version.

Plaintext:	`a t t a c k a t d a w n`
Key:	`s e c r e t s e c r e t`
Ciphertext:	`s x v r g d s x f r a g`

Although polyalphabetic substitution protects against direct frequency analysis, it is vulnerable to a second-order form of frequency analysis called *period analysis*, which is an examination of frequency based on the repeated use of the key.

One-Time Pads

A *one-time pad* is an extremely powerful type of substitution cipher. One-time pads use a different substitution alphabet for each letter of the plaintext message. They can be represented by the following encryption function, where *K* is the encryption key used to encrypt the plaintext letter *P* into the ciphertext letter *C*:

```
C = (P + K) mod 26
```

Usually, one-time pads are written as a very long series of numbers to be plugged into the function.

> NOTE One-time pads are also known as *Vernam ciphers*, after the name of their inventor, Gilbert Sandford Vernam of AT&T Bell Labs.

The great advantage of one-time pads is that, when used properly, they are an unbreakable encryption scheme. There is no repeating pattern of alphabetic substitution, rendering cryptanalytic efforts useless. However, several requirements must be met to ensure the integrity of the algorithm.

- The one-time pad must be randomly generated. Using a phrase or a passage from a book would introduce the possibility that cryptanalysts could break the code.

- The one-time pad must be physically protected against disclosure. If the enemy has a copy of the pad, they can easily decrypt the enciphered messages.

NOTE You may be thinking at this point that the Caesar cipher, Vigenère cipher, and one-time pad sound very similar. They are! The only difference is the key length. The Caesar shift cipher uses a key of length one, the Vigenère cipher uses a longer key (usually a word or sentence), and the one-time pad uses a key that is as long as the message itself.

- Each one-time pad must be used only once. If pads are reused, cryptanalysts can compare similarities in multiple messages encrypted with the same pad and possibly determine the key values used.
- The key must be at least as long as the message to be encrypted. This is because each character of the key is used to encode only one character of the message.

TIP These one-time pad security requirements are essential knowledge for any network security professional. All too often, people attempt to implement a one-time pad cryptosystem but fail to meet one or more of these fundamental requirements. Read on for an example of how an entire Soviet code system was broken because of carelessness in this area.

If any one of these requirements is not met, the impenetrable nature of the one-time pad instantly breaks down. In fact, one of the major intelligence successes of the United States resulted when cryptanalysts broke a top-secret Soviet cryptosystem that relied on the use of one-time pads. In this project, code-named VENONA, a pattern in the way the Soviets generated the key values used in their pads was discovered. The existence of this pattern violated the first requirement of a one-time pad cryptosystem: the keys must be randomly generated without the use of any recurring pattern. The entire VENONA project was recently declassified and is publicly available on the National Security Agency website at https://www.nsa.gov/about/cryptologic-heritage/historical-figures-publications/publications/coldwar/assets/files/venona_story.pdf.

One-time pads have been used throughout history to protect extremely sensitive communications. The major obstacle to their widespread use is the difficulty of generating, distributing, and safeguarding the lengthy keys required. One-time pads can realistically be used only for short messages, because of key lengths.

Running Key Ciphers

Many cryptographic vulnerabilities surround the limited length of the cryptographic key. As you learned in the previous section, one-time pads avoid these vulnerabilities by using a key that is at least as long as the message. However, one-time pads are awkward to implement because they require the physical exchange of pads.

One common solution to this dilemma is the use of a *running key cipher* (also known as a *book cipher*). In this cipher, the encryption key is as long as the message itself and is often chosen from a common book. For example, the sender and recipient might agree in advance to use the text of a chapter from *Moby-Dick*, beginning with the third paragraph, as the key. They would both simply use as many consecutive characters as necessary to perform the encryption and decryption operations.

Let's look at an example. Suppose you wanted to encrypt the message "Richard will deliver the secret package to Matthew at the bus station tomorrow" using the key just described. This message is 66 characters in length, so you'd use the first 66 characters of the running key: "With much interest I sat watching him. Savage though he was, and hideously marred." Any algorithm could then be used to encrypt the plaintext message using this key. Let’s look at the example of modulo 26 addition, which converts each letter to a decimal equivalent, adds the plaintext to the key, and then performs a modulo 26 operation to yield the ciphertext. If you assign the letter *A* the value 0 and the letter *Z* the value 25, you have the following encryption operation for the first two words of the ciphertext:

Plaintext	R	I	C	H	A	R	D	W	I	L	L
Key	W	I	T	H	M	U	C	H	I	N	T
Numeric plaintext	17	8	2	7	0	17	3	22	8	11	11
Numeric key	22	8	19	7	12	20	2	7	8	13	19
Numeric ciphertext	13	16	21	14	12	11	5	3	16	24	4
Ciphertext	N	Q	V	O	M	L	F	D	Q	Y	E

When the recipient receives the ciphertext, they use the same key and then subtract the key from the ciphertext, perform a modulo 26 operation, and then convert the resulting plaintext back to alphabetic characters.

Block Ciphers

Block ciphers operate on “chunks,” or blocks, of a message and apply the encryption algorithm to an entire message block at the same time. The transposition ciphers are examples of block ciphers. The simple algorithm used in the challenge-response algorithm takes an entire word and reverses its letters. The more complicated columnar transposition cipher works on an entire message (or a piece of a message) and encrypts it using the transposition algorithm and a secret keyword. Most modern encryption algorithms implement some type of block cipher.

Stream Ciphers

Stream ciphers operate on one character or bit of a message (or data stream) at a time. The Caesar cipher is an example of a stream cipher. The one-time pad is also a stream cipher because the algorithm operates on each letter of the plaintext message independently. Stream ciphers can also function as a type of block cipher. In such operations there is a buffer that fills up to real-time data that is then encrypted as a block and transmitted to the recipient.

Confusion and Diffusion

Cryptographic algorithms rely on two basic operations to obscure plaintext messages—confusion and diffusion. *Confusion* occurs when the relationship between the plaintext and the key is so complicated that an attacker can't merely continue altering the plaintext and analyzing the resulting ciphertext to determine the key. *Diffusion* occurs when a change in the plaintext results in multiple changes spread throughout the ciphertext. Consider, for example, a cryptographic algorithm that first performs a complex substitution and then uses transposition to rearrange the characters of the substituted ciphertext. In this example, the substitution introduces confusion, and the transposition introduces diffusion.

Modern Cryptography

Modern cryptosystems use computationally complex algorithms and long cryptographic keys to meet the cryptographic goals of confidentiality, integrity, authentication, and nonrepudiation. The following sections cover the roles cryptographic keys play in the world of data security and examine three types of algorithms commonly used today: symmetric encryption algorithms, asymmetric encryption algorithms, and hashing algorithms.

Cryptographic Keys

In the early days of cryptography, one of the predominant principles was “security through obscurity.” Some cryptographers thought the best way to keep an encryption algorithm secure was to hide the details of the algorithm from outsiders. Old cryptosystems required communicating parties to keep the algorithm used to encrypt and decrypt messages secret from third parties. Any disclosure of the algorithm could lead to compromise of the entire system by an adversary.

Modern cryptosystems do not rely on the secrecy of their algorithms. In fact, the algorithms for most cryptographic systems are widely available for public review in the accompanying literature and on the internet. Opening algorithms to public scrutiny actually improves their security. Widespread analysis of algorithms by the computer security community allows practitioners to discover and correct potential security vulnerabilities and ensure that the algorithms they use to protect their communications are as secure as possible.

Instead of relying on secret algorithms, modern cryptosystems rely on the secrecy of one or more cryptographic keys used to personalize the algorithm for specific users or groups of users. Recall from the discussion of transposition ciphers that a keyword is used with the columnar transposition to guide the encryption and decryption efforts. The algorithm used to perform columnar transposition is well known —you just read the details of it in this book! However, columnar transposition can be used to securely communicate between parties as long as a keyword is chosen that would not be guessed by an outsider. As long as the security of this keyword is maintained, it doesn't matter that third parties know the details of the algorithm.

> NOTE Although the public nature of the algorithm does not compromise the security of columnar transposition, the method does possess several inherent weaknesses that make it vulnerable to cryptanalysis. It is therefore an inadequate technology for use in modern secure communication.

In the discussion of one-time pads earlier in this chapter, you learned that the main strength of the one-time pad algorithm is derived from the fact that it uses an extremely long key. In fact, for that algorithm, the key is at least as long as the message itself. Most modern cryptosystems do not use keys quite that long, but the length of the key is still an extremely important factor in determining the strength of the cryptosystem and the likelihood that the encryption will not be compromised through cryptanalytic techniques.

The rapid increase in computing power allows you to use increasingly long keys in your cryptographic efforts. However, this same computing power is also in the hands of cryptanalysts attempting to defeat the algorithms you use. Therefore, it's essential that you outpace adversaries by using sufficiently long keys that will defeat contemporary cryptanalysis efforts. Additionally, if you want to improve the chance that your data will remain safe from cryptanalysis some time into the future, you must strive to use keys that will outpace the projected increase in cryptanalytic capability during the entire time period the data must be kept safe. For example, the advent of quantum computing may transform cryptography, rendering current cryptosystems insecure, as discussed earlier in this chapter.

Several decades ago, when the Data Encryption Standard (DES) was created, a 56-bit key was considered sufficient to maintain the security of any data. However, there is now widespread agreement that the 56-bit DES algorithm is no longer secure because of advances in cryptanalysis techniques and supercomputing power. Modern cryptographic systems use at least a 128-bit key to protect data against prying eyes. Remember, the length of the key directly relates to the work function of the cryptosystem: the longer the key, the harder it is to break the cryptosystem.

Symmetric Key Algorithms

Symmetric key algorithms rely on a “shared secret” encryption key that is distributed to all members who participate in the communications. This key is used by all parties to both encrypt and decrypt messages, so the sender and the receiver both possess a copy of the shared key. The sender encrypts with the shared secret key and the receiver decrypts with it. When large-sized keys are used, symmetric encryption is very difficult to break. It is primarily employed to perform bulk encryption and provides only for the security service of confidentiality. Symmetric key cryptography can also be called *secret key cryptography* and *private key cryptography*. Figure 6.3 illustrates the symmetric key encryption and decryption processes.

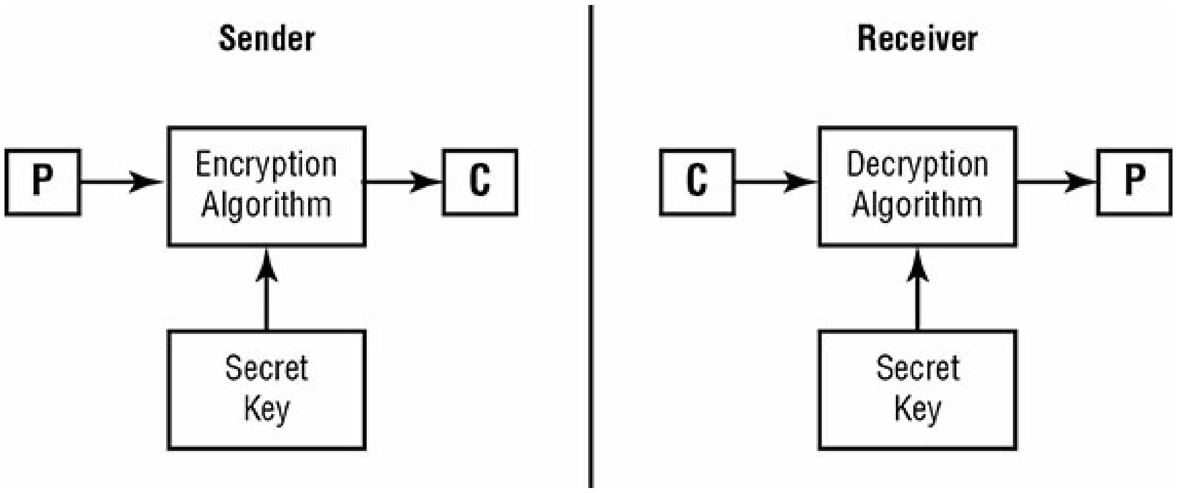

FIGURE 6.3 Symmetric key cryptography

The use of the term *private key* can be tricky because it is part of three different terms that have two different meanings. The term *private key* by itself always means the private key from the key pair of public key cryptography (aka asymmetric). However, both *private key cryptography* and *shared private key* refer to symmetric cryptography. The meaning of the word *private* is stretched to refer to two people sharing a secret that they keep confidential. (The true meaning of *private is that only a single person* has a secret that's kept confidential.) Be sure to keep these confusing terms straight in your studies.

Symmetric key cryptography has several weaknesses.

Key distribution is a major problem. Parties must have a secure method of exchanging the secret key before establishing communications with a symmetric key protocol. If a secure electronic channel is not available, an offline key distribution method must often be used (that is, out-of-band exchange).

Symmetric key cryptography does not implement nonrepudiation. Because any communicating party can encrypt and decrypt messages with the shared secret key, there is no way to prove where a given message originated.

The algorithm is not scalable. It is extremely difficult for large groups to communicate using symmetric key cryptography. Secure private communication between individuals in the group could be achieved only if each possible combination of users shared a private key.

Keys must be regenerated often. Each time a participant leaves the group, all keys known by that participant must be discarded.

The major strength of symmetric key cryptography is the great speed at which it can operate. Symmetric key encryption is very fast, often 1,000 to 10,000 times faster than asymmetric algorithms. By nature of the mathematics involved, symmetric key cryptography also naturally lends itself to hardware implementations, creating the opportunity for even higher-speed operations.

The section "Symmetric Cryptography" later in this chapter provides a detailed look at the major secret key algorithms in use today.

Asymmetric Key Algorithms

Asymmetric key algorithms, also known as *public key algorithms*, provide a solution to the weaknesses of symmetric key encryption. In these systems, each user has two keys: a public key, which is shared with all users, and a private key, which is kept secret and known only to the user. But here's a twist: opposite and related keys must be used in tandem to encrypt and decrypt. In other words, if the public key encrypts a message, then only the corresponding private key can decrypt it, and vice versa.

Figure 6.4 shows the algorithm used to encrypt and decrypt messages in a public key cryptosystem. Consider this example. If Alice wants to send a message to Bob using public key cryptography, she creates the message and then encrypts it using Bob's public key. The only possible way to decrypt this ciphertext is to use Bob's private key, and the only user with access to that key is Bob. Therefore, Alice can't even decrypt the message herself after she encrypts it. If Bob wants to send a reply to Alice, he simply encrypts the message using Alice's public key, and then Alice reads the message by decrypting it with her private key.

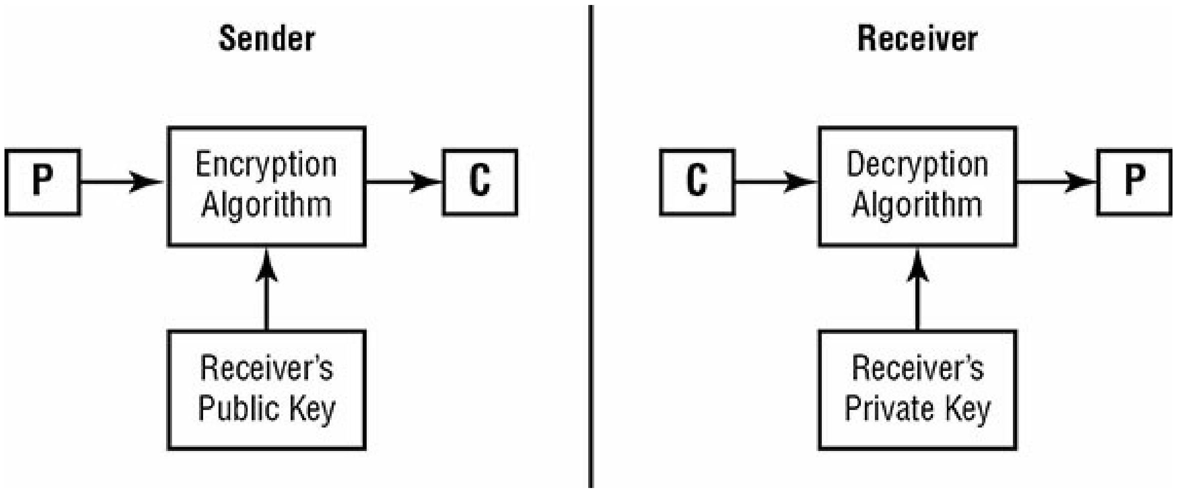

FIGURE 6.4 Asymmetric key cryptography

Real World Scenario

Key Requirements

In a class one of the authors of this book taught recently, a student wanted to see an illustration of the scalability issue associated with symmetric encryption algorithms. The fact that symmetric cryptosystems require each pair of potential communicators to have a shared private key makes the algorithm nonscalable. The total number of keys required to completely connect *n* parties using symmetric cryptography is given by the following formula:

$$\text{Number of Keys} = \frac{n(n-1)}{2}$$

Now, this might not sound so bad (and it's not for small systems), but consider the following figures. Obviously, the larger the population, the less likely a symmetric cryptosystem will be suitable to meet its needs.

Number of participants	**Number of symmetric keys required**	**Number of asymmetric keys required**
2	1	4
3	3	6
4	6	8
5	10	10
10	45	20
100	4,950	200
1,000	499,500	2,000
10,000	49,995,000	20,000

Asymmetric key algorithms also provide support for digital signature technology. Basically, if Bob wants to assure other users that a message with his name on it was actually sent by him, he first creates a message digest by using a hashing algorithm (you'll find more on hashing algorithms in the next section). Bob then encrypts that digest using his private key. Any user who wants to verify the signature simply decrypts the message digest using Bob's public key and then verifies that the decrypted message digest is accurate. Chapter 7 explains this process in greater detail.

The following is a list of the major strengths of asymmetric key cryptography:

The addition of new users requires the generation of only one public-private key pair. This same key pair is used to communicate with all users of the asymmetric cryptosystem. This makes the algorithm extremely scalable.

Users can be removed far more easily from asymmetric systems. Asymmetric cryptosystems provide a key revocation mechanism that allows a key to be canceled, effectively removing a user from the system.

Key regeneration is required only when a user's private key is compromised. If a user leaves the community, the system administrator simply needs to invalidate that user's keys. No other keys are compromised and therefore key regeneration is not required for any other user.

Asymmetric key encryption can provide integrity, authentication, and nonrepudiation. If a user does not share their private key with other individuals, a message signed by that user can be shown to be accurate and from a specific source and cannot be later repudiated.

Key distribution is a simple process. Users who want to participate in the system simply make their public key available to anyone with whom they want to communicate. There is no method by which the private key can be derived from the public key.

No preexisting communication link needs to exist. Two individuals can begin communicating securely from the moment they start communicating. Asymmetric cryptography does not require a preexisting relationship to provide a secure mechanism for data exchange.

The major weakness of public key cryptography is its slow speed of operation. For this reason, many applications that require the secure transmission of large amounts of data use public key cryptography to establish a connection and then exchange a symmetric secret key. The remainder of the session then uses symmetric cryptography. Table 6.1 compares the symmetric and asymmetric cryptography systems. Close examination of this table reveals that a weakness in one system is matched by a strength in the other.

TABLE 6.1 Comparison of symmetric and asymmetric cryptography systems

Symmetric	**Asymmetric**
Single shared key	Key pair sets
Out-of-band exchange	In-band exchange
Not scalable	Scalable
Fast	Slow
Bulk encryption	Small blocks of data, digital signatures, digital envelopes, digital certificates
Confidentiality	Confidentiality, integrity, authenticity, nonrepudiation

NOTE Chapter 7 provides technical details on modern public key encryption algorithms and some of their applications.

Hashing Algorithms

In the previous section, you learned that public key cryptosystems can provide digital signature capability when used in conjunction with a message digest. Message digests are summaries of a message's content (not unlike a file checksum) produced by a hashing algorithm. It's extremely difficult, if not impossible, to derive a message from an ideal hash function, and it's very unlikely that two messages will produce the same hash value. Cases where a hash function produces the same value for two different methods are known as *collisions*, and the existence of collisions typically leads to the deprecation of a hashing algorithm.

Chapter 7 provides details on contemporary hashing algorithms and explains how they are used to provide digital signature capability, which helps meet the cryptographic goals of integrity and nonrepudiation.