Chapter 6
Cryptography and Symmetric Key Algorithms

THE CISSP TOPICS COVERED IN THIS CHAPTER INCLUDE:

- ✓ **Domain 3.0: Security Architecture and Engineering**
  - 3.5 Assess and mitigate the vulnerabilities of security architectures, designs, and solution elements
    - 3.5.4 Cryptographic systems
  - 3.6 Select and determine cryptographic solutions
    - 3.6.1 Cryptographic life cycle (e.g., keys, algorithm selection)
    - 3.6.2 Cryptographic methods (e.g., symmetric)

Cryptography provides confidentiality, integrity, authentication, and nonrepudiation for sensitive information while it is stored (at rest), traveling across a network (in transit/in motion), and existing in memory (in use/in processing). Cryptography is an extremely important security technology that is embedded in many of the controls used to protect information from unauthorized visibility and use.

Over the years, mathematicians and computer scientists have developed a series of increasingly complex cryptographic algorithms designed to increase the level of protection provided to data. While cryptographers spent time developing strong encryption algorithms, malicious hackers and governments alike devoted significant resources to undermining them. This led to an “arms race” in cryptography and resulted in the development of the extremely sophisticated algorithms in use today.

This chapter looks at the basics of cryptographic communications and the fundamental principles of symmetric key (secret key) cryptosystems. The next chapter continues the discussion of cryptography by examining asymmetric key (public key) cryptosystems and the various techniques attackers use to defeat cryptography.

Cryptographic Foundations

The study of any science must begin with a discussion of the fundamental principles on which it is built. The following sections lay this foundation with a review of the goals of cryptography, an overview of the basic concepts of cryptographic technology, and a look at the major mathematical principles used by cryptographic systems.

Goals of Cryptography

Security practitioners use cryptographic systems to meet four fundamental goals: confidentiality, integrity, authentication, and nonrepudiation. Achieving each of these goals requires the satisfaction of a number of design requirements, and not all cryptosystems are intended to achieve all four goals. In the following sections, we'll examine each goal in detail and give a brief description of the technical requirements necessary to achieve it.

Confidentiality

Confidentiality ensures that data remains private in three different situations: when it is at rest, when it is in transit, and when it is in use.

Confidentiality is perhaps the most widely cited goal of cryptosystems—the preservation of secrecy for stored information or for communications between individuals and groups. Two main types of cryptosystems enforce confidentiality:

- *Symmetric cryptosystems* use a shared secret key available to all users of the cryptosystem.
- *Asymmetric cryptosystems* use individual pairs of public and private keys for each user of the system.

Both of these concepts are explored in the section "Modern Cryptography," later in this chapter.

When developing a cryptographic system for the purpose of providing confidentiality, you must think about the three different types of data that we discussed in Chapter 5, "Protecting Security of Assets":

- *Data at rest*, or stored data, resides in a fixed location awaiting access. Examples of data at rest include data stored on hard drives, backup tapes, cloud storage services, USB devices, and other storage media.
- *Data in transit*, data in motion, or data on the wire is data being transmitted across a network between two systems. Data in motion might be traveling on a corporate network, a wireless network, or the Internet.
- *Data in use* is data that is stored in the active memory of a computer system, where it may be accessed by a process running on that system.

TIP You should also know that data in transit is also commonly called data *on the wire*, referring to the network cables that carry data communications.

Each of these situations poses different types of confidentiality risks that cryptography can protect against. For example, data in transit may be susceptible to eavesdropping attacks, whereas data at rest is more susceptible to the theft of physical devices. Data in use may be accessed by unauthorized processes if the operating system does not properly implement process isolation.

Integrity

Integrity ensures that data is not altered without authorization. If integrity mechanisms are in place, the recipient of a message can be certain that the message received is identical to the message that was sent. Similarly, integrity checks can ensure that stored data was not altered between the time it was created and the time it was accessed. Integrity controls protect against all forms of alteration, including intentional alteration by a third party attempting to insert false information, intentional deletion of portions of the data, and unintentional alteration by faults in the transmission process.

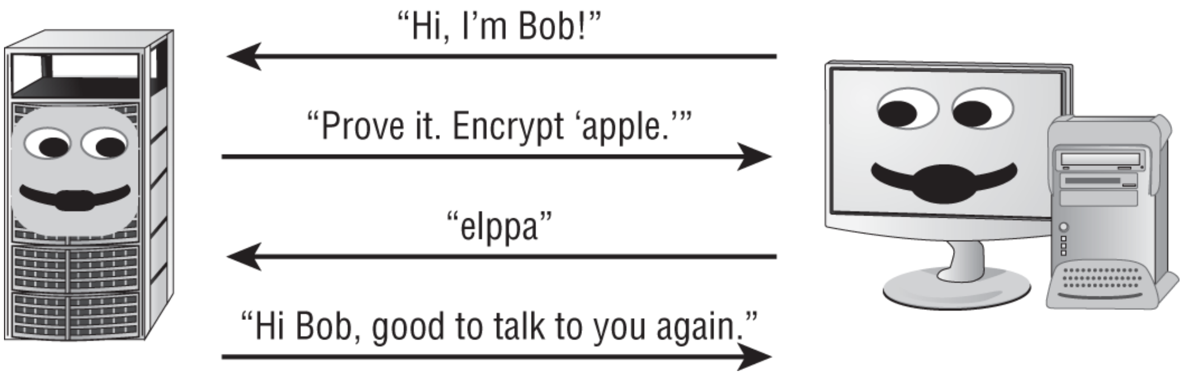
Message integrity is enforced through the use of encrypted message digests, known as *digital signatures*, created upon transmission of a message. The recipient of the message simply verifies that the message's digital signature is valid, ensuring that the message was not altered in transit. Integrity can be enforced by both public and secret key cryptosystems. This concept is discussed in detail in Chapter 7, "PKI and Cryptographic Applications." The use of cryptographic hash functions to protect file integrity is discussed in Chapter 21, "Malicious Code and Application Attacks."

Authentication

Authentication verifies the claimed identity of system users and is a major function of cryptosystems. For example, suppose that Bob wants to establish a communications session with Alice and they are both participants in a shared secret communications system. Alice might use a challenge-response authentication technique to ensure that Bob is who he claims to be.

Figure 6.1 shows how this challenge-response protocol would work in action. In this example, the shared-secret code used by Alice and Bob is quite straightforward—the letters of each word are simply reversed. Bob first contacts Alice and identifies himself. Alice then sends a challenge message to Bob, asking him to encrypt a short message using the secret code known only to Alice and Bob. Bob replies with the encrypted message. After

Alice verifies that the encrypted message is correct, she trusts that Bob himself is truly on the other end of the connection.



"Hi, I'm Bob!"
"Prove it. Encrypt 'apple.'"
"elppa"
"Hi Bob, good to talk to you again."

FIGURE 6.1 Challenge-response authentication protocol

Nonrepudiation

Nonrepudiation provides assurance to the recipient that the message was originated by the sender and not someone masquerading as the sender. It also prevents the sender from claiming that they never sent the message in the first place (also known as *repudiating* the message). Secret key, or symmetric key, cryptosystems (such as simple substitution ciphers) do not provide this guarantee of nonrepudiation. If Jim and Bob participate in a secret key communication system, they can both produce the same encrypted message using their shared secret key. Nonrepudiation is offered only by public key, or asymmetric, cryptosystems, a topic discussed in greater detail in Chapter 7.

Cryptography Concepts

As with any science, you must be familiar with certain terminology before studying cryptography. Let's take a look at a few of the key terms used to describe codes and ciphers. Before a message is put into a coded form, it is known as a *plaintext* message and is represented by the letter *P* when encryption functions are described. The sender of a message uses a cryptographic algorithm to *encrypt* the plaintext message and produce a *ciphertext* message, represented by the letter *C*. This message is transmitted by some physical or electronic means to the recipient. The recipient then uses a predetermined algorithm to *decrypt* the ciphertext message and retrieve the plaintext version. (For an illustration of this process, see Figure 6.3 later in this chapter.)

All cryptographic algorithms rely on *keys* to maintain their security. For the most part, a key is nothing more than a number. It's usually a very large binary number, but it's a number nonetheless. Every algorithm has a specific *key space*. The key space is the range of values that are valid for use as a key for a specific algorithm. A key space is defined by its *bit size*. Bit size is nothing more than the number of binary bits (0s and 1s) in the key. The key space is the range between the key that has all 0s and the key that has all 1s. Or to state it another way, the key space is the range of numbers from 0 to 2^n, where *n* is the bit size of the key. So, a 128-bit key can have a value from 0 to 2^{128} (which is roughly $3.40282367 \times 10^{38}$, a very big number!). It is absolutely critical to protect the security of secret keys and private keys. In fact, all of the security you gain from cryptography rests on your ability to keep the secret and private keys confidential.

Kerckhoffs's Principle

All cryptography relies on algorithms. An *algorithm* is a set of rules, usually mathematical, that dictates how encryption and decryption processes are to take place. Most cryptographers follow Kerckhoffs's principle, a concept that makes algorithms known and public, allowing anyone to examine and test them. Specifically, *Kerckhoffs's principle* (also known as Kerckhoffs's assumption) is that a cryptographic system should be secure even if everything about the system, except the key, is public knowledge. The principle can be summed up as “The enemy knows the system.”

A large number of cryptographers adhere to this principle, but not all agree. In fact, some believe that better overall security can be maintained by keeping both the algorithm and the key private. Kerckhoffs's adherents retort that the opposite approach includes the dubious practice of “security through obscurity” and believe that public exposure produces more activity and exposes more weaknesses more readily, leading to the abandonment of insufficiently strong algorithms and quicker adoption of suitable ones.

As you'll learn in this chapter and the next, different types of algorithms require different types of keys. In symmetric key (or secret key) cryptosystems, all participants use a single shared key. In public key cryptosystems, each participant has their own pair of public and private keys. Cryptographic keys are sometimes referred to as *cryptovariables*, particularly in U.S. government applications.

The art of creating and implementing secret codes and ciphers is known as *cryptography*. This practice is paralleled by the art of *cryptanalysis*—the study of methods to defeat codes and ciphers. Together, cryptography and cryptanalysis are commonly referred to as *cryptology*. Specific implementations of a code or cipher in hardware and software are known as *cryptosystems*.

Federal Information Processing Standard (FIPS) 140–3, “Security Requirements for Cryptographic Modules,” defines the hardware and software requirements for cryptographic modules that the federal government uses.

Cryptographic Mathematics

Cryptography is no different from most computer science disciplines in that it finds its foundations in the science of mathematics. To fully understand cryptography, you must first understand the basics of binary mathematics and the logical operations used to manipulate binary values. The following sections present a brief look at some of the most fundamental concepts with which you should be familiar.

Boolean Mathematics

Boolean mathematics defines the rules used for the bits and bytes that form the nervous system of any computer. You're most likely familiar with the decimal system. It is a base 10 system in which an integer from 0 to 9 is used in each place and each place value is a multiple of 10. It's likely that our reliance on the decimal system has biological origins—human beings have 10 fingers that can be used to count.

Boolean math can be very confusing at first, but it's worth the investment of time to learn how logical functions work. You need to know these concepts to truly understand the inner workings of cryptographic algorithms.

Similarly, the computer's reliance on the Boolean system has electrical origins. In an electrical circuit, there are only two possible states—on (representing the presence of electrical current) and off (representing the absence of electrical current). All computation performed by an electrical device must be expressed in these terms, giving rise to the use of Boolean computation in modern electronics. In general, computer scientists refer to the on condition as a *true* value and the off condition as a *false* value.

Logical Operations

The Boolean mathematics of cryptography uses a variety of logical functions to manipulate data. We'll take a brief look at several of these operations.

AND

The AND operation (represented by the ∧ symbol) checks to see whether two values are both true. Table 6.1 shows a truth table that illustrates all four possible outputs for the AND function. In this truth table, the first two columns, X and Y, show the input values to the AND function. Remember, the AND function takes only two variables as input. In Boolean math, there are only two possible values for each of these variables (0=FALSE and 1=TRUE), leading to four possible inputs to the AND function. The X ∧ Y column shows the output of the AND function for the input values shown in the two adjacent columns. It's this finite number of possibilities that makes it extremely easy for computers to implement logical functions in hardware. Notice in Table 6.1 that only one combination of inputs (where both inputs are TRUE) produces an output value of true.

Logical operations are often performed on entire Boolean words rather than single values. Take a look at the following example:

TABLE 6.1 AND operation truth table

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

```
X:      0 1 1 0 1 1 0 0
Y:      1 0 1 0 0 1 1 1
___________________________
X ∧ Y: 0 0 1 0 0 1 0 0
```

Notice that the AND function is computed by comparing the values of `X` and `Y` in each column. The output value is TRUE only in columns where both `X` and `Y` are true.

OR

The OR operation (represented by the ∨ symbol) checks to see whether at least one of the input values is true. Refer to the truth table in Table 6.2 for all possible values of the OR function. Notice that the only time the OR function returns a false value is when both of the input values are false.

TABLE 6.2 OR operation truth table

X	Y	X ∨ Y
0	0	0
0	1	1
1	0	1
1	1	1

We'll use the same example we used in the previous section to show you what the output would be if `X` and `Y` were fed into the OR function rather than the AND function:

```
X:      0 1 1 0 1 1 0 0
Y:      1 0 1 0 0 1 1 1
___________________________
X ∨ Y: 1 1 1 0 1 1 1 1
```

NOT

The NOT operation (represented by the ~ symbol) simply reverses the value of an input variable. This function operates on only one variable at a time. Table 6.3 shows the truth table for the NOT function.

TABLE 6.3 NOT operation truth table

X	~X
0	1
1	0

In this example, you take the value of X from the previous examples and run the NOT function against it:

```
X:   0 1 1 0 1 1 0 0
___________________________
~X: 1 0 0 1 0 0 1 1
```

Exclusive OR

The final logical function you'll examine in this chapter is perhaps the most important and most commonly used in cryptographic applications—the exclusive OR (XOR) function. It's referred to in mathematical literature as the XOR function and is commonly represented by the ⊕ symbol. The XOR function returns a true value when only one of the input values is true. If both values are false or both values are true, the output of the XOR function is false. Table 6.4 provides the truth table for the XOR operation.

TABLE 6.4 Exclusive OR operation truth table

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

The following operation shows the X and Y values when they are used as input to the XOR function:

```
X:      0 1 1 0 1 1 0 0
Y:      1 0 1 0 0 1 1 1
___________________________
X ⊕ Y: 1 1 0 0 1 0 1 1
```

Modulo Function

The *modulo* (mod) function is extremely important in the field of cryptography. Think back to the early days when you first learned division. At that time, you weren't familiar with decimal numbers and compensated by showing a remainder value each time you performed a division operation. Computers don't naturally understand the decimal system either, and these remainder values play a critical role when computers perform many mathematical functions. The modulo function is, quite simply, the remainder value left over after a division operation is performed.

The modulo function is just as important to cryptography as the logical operations are. Be sure you're familiar with its functionality and can perform simple modular math.

The modulo function is usually represented in equations by the abbreviation *mod*, although it's also sometimes represented by the % symbol. Here are several inputs and outputs for the modulo function:

```
8 mod 6 = 2
6 mod 8 = 6
10 mod 3 = 1
10 mod 2 = 0
32 mod 8 = 0
32 mod 26 = 6
```

We'll revisit this function in Chapter 7 when we explore the RSA public key encryption algorithm (named after Ron Rivest, Adi Shamir, and Leonard Adleman, its inventors).

One-Way Functions

A *one-way function* is a mathematical operation that easily produces output values for each possible combination of inputs but makes it impossible to retrieve the input values. Public key cryptosystems are all based on some sort of one-way function. In practice, however, it's never been proven that any specific known function is truly one-way. Cryptographers rely on functions that they believe are one-way, but it's always possible that they might be broken by future cryptanalysts.

Here's an example. Imagine you have a function that multiplies three numbers together. If you restrict the input values to single-digit numbers, it's a relatively straightforward matter to reverse-engineer this function and determine the possible input values by looking at the numerical output. For example, the output value 15 was created by using the input values 1, 3, and 5. However, suppose you restrict the input values to five-digit prime numbers. It's still quite simple to obtain an output value by using a computer or a good calculator, but reverse-engineering is not quite so simple. Can you figure out which three prime numbers were used to obtain the output value 10,718,488,075,259? Not so simple, eh? (As it turns out, the number is the product of the prime numbers 17,093; 22,441; and 27,943.) There are actually 8,363 five-digit prime numbers, so this problem might be attacked using a computer and a brute-force algorithm, but there's no easy way to figure it out in your head, that's for sure!

Nonce

Cryptography often gains strength by adding randomness to the encryption process. One method by which this is accomplished is through the use of a nonce. A *nonce* (from the phrase "number used once") is a random number that acts as a placeholder variable in mathematical functions. When the function is executed, the nonce is replaced with a random number generated at the moment of processing for one-time use. The nonce must be a unique number each time it is used. One of the more recognizable examples of a nonce is an initialization vector (IV), a random bit string that is the same length as the block size (the amount of data to be encrypted in each operation) and is XORed with the message. IVs are used to create unique ciphertexts every time the same message is encrypted using the same key.

Zero-Knowledge Proof

One of the benefits of cryptography is found in the mechanism to prove your knowledge of a fact to a third party without revealing the fact itself to that third party. This is often done with passwords and other secret authenticators.

The classic example of a *zero-knowledge proof* involves two individuals: Peggy and Victor. Peggy knows the password to a secret door located inside a circular cave, as shown in Figure 6.2. Victor would like to buy the password from Peggy, but he wants Peggy to prove that she knows the password before paying her for it. Peggy doesn't want to tell Victor the password for fear that he won't pay her later. The zero-knowledge proof can solve their dilemma.

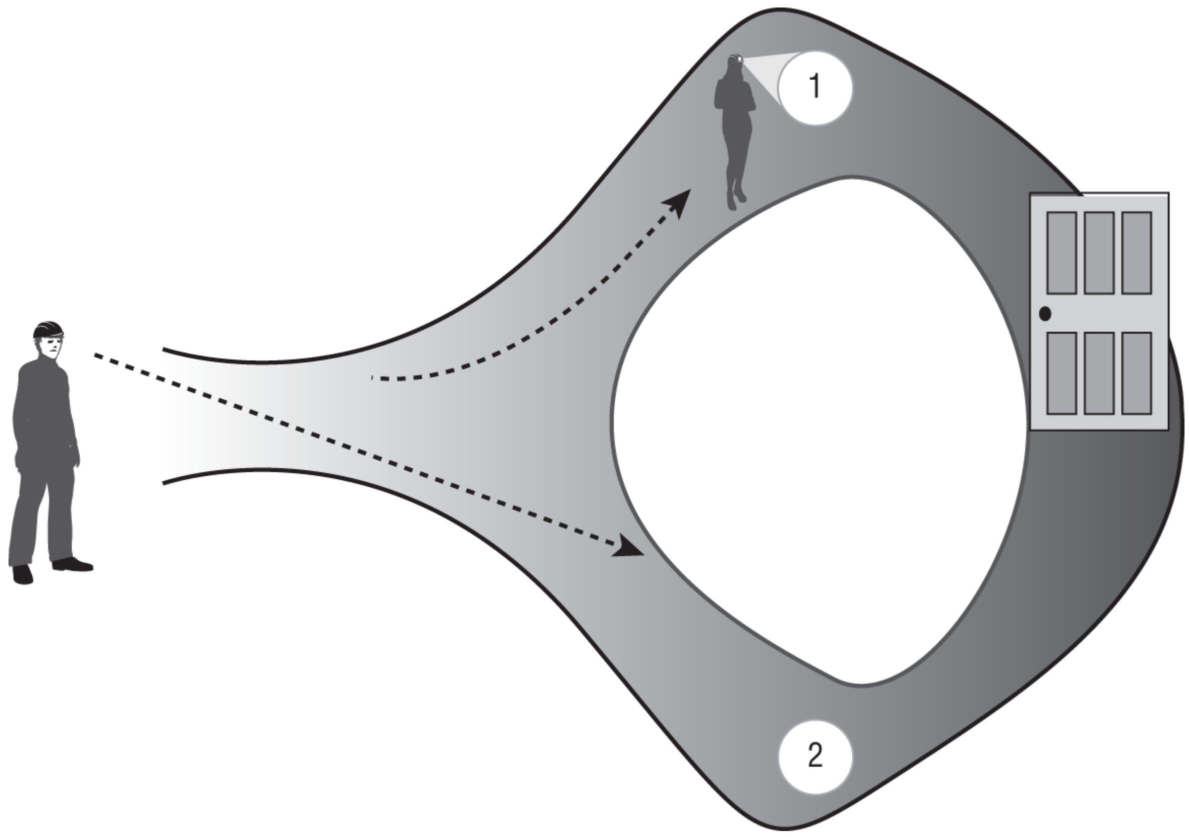


FIGURE 6.2 The magic door

Victor can stand at the entrance to the cave and watch Peggy depart down path 1. Peggy then reaches the door and opens it using the password. She then passes through the door and returns via path 2. Victor saw her leave down path 1 and return via path 2, proving that she must know the correct password to open the door.

Zero-knowledge proofs appear in cryptography in cases where one individual wants to demonstrate knowledge of a fact (such as a password or key) without actually disclosing that fact to the other individual. This may be done through complex mathematical operations, such as discrete logarithms and graph theory.

Split Knowledge

When the information or privilege required to perform an operation is divided among multiple users, no single person has sufficient privileges to compromise the security of an environment. This separation of duties and two-person control contained in a single solution is called *split knowledge*. The best example of split knowledge is seen in the concept of *key escrow*. In a key escrow arrangement, a cryptographic key is stored with a third party for safekeeping. When certain circumstances are met, the third party may use the escrowed key to either restore an authorized user's access or decrypt the material themselves. This third party is known as the recovery agent. In arrangements that use only a single-key escrow recovery agent, there exists an opportunity for fraud and abuse of this privilege, as the single recovery agent could unilaterally decide to decrypt the information. The *M of N Control* concept requires that a minimum number of agents (M) out of the total number of agents (N) work together to perform high-security tasks. So, implementing three of eight controls would require three people out of the eight with the assigned work task of key escrow recovery agent to work together to pull a single key out of the key escrow database (thereby also illustrating that M is always less than or equal to N).

Work Function

You can measure the strength of a cryptography system by measuring the effort in terms of cost and/or time using a *work function* or work factor. Usually the time and effort required to perform a complete brute-force attack against an encryption system is what the work function represents. The security and protection offered by a cryptosystem is directly proportional to the value of the work function/factor. The size of the work function should be matched against the relative value of the protected asset. The work function need be only slightly greater than the time value of that asset. In other words, all security, including cryptography, should be cost-effective and cost-efficient. Spend no more effort to protect an asset than it warrants, but be sure to provide sufficient protection. Thus, if information loses its value over time, the work function needs to be only large enough to ensure protection until the value of the data is gone.

In addition to understanding the length of time that the data will have value, security professionals selecting cryptographic systems must understand how emerging technologies may impact cipher-cracking efforts. For example, researchers may discover a flaw in a cryptographic algorithm next year that renders information protected with that algorithm insecure. Similarly, technological advancements in cloud-based parallel computing and quantum computing may make brute-force efforts much more feasible down the road.

Ciphers

Cipher systems have long been used by individuals and governments 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.” Semaphores (visual signaling) and Morse code are also examples of codes. These codes are commonly known by the public and 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 ASCII or Unicode message or another encoding), 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.

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

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. One of the earliest known substitution ciphers was used by Julius Caesar to communicate with Cicero in Rome while he was conquering Europe. Caesar knew that there were several risks when sending messages—one of the messengers might be an enemy spy or might be ambushed while en route to the deployed forces. For that reason, Caesar developed a cryptographic system now known as the *Caesar cipher*. The system is extremely simple. To encrypt a message, you simply shift each letter of the alphabet three places to the right. For example, *A* would become *D*, and *B* would become *E*. If you reach the end of the alphabet during this process, you simply wrap around to the beginning so that *X* becomes *A*, *Y* becomes *B*, and *Z* becomes *C*. For this reason, the Caesar cipher also became known as the ROT3 (or Rotate 3) cipher. The Caesar cipher is a substitution cipher that is mono-alphabetic.

NOTE Although the Caesar cipher uses a shift of 3, the more general shift cipher uses the same algorithm to shift any number of characters desired by the user. For example, the ROT12 cipher would turn an *A* into an *M*, a *B* into an *N*, and so on.

Here's an example of the Caesar cipher in action. The first line contains the original sentence, and the second line shows what the sentence looks like when it is encrypted using the Caesar cipher.

```
THE DIE HAS BEEN CAST
WKH GLH KDV EHHQ FDVW
```

To decrypt the message, you simply shift each letter three places to the left.

WARNING Although the Caesar cipher is easy to use, it's also easy to crack. It's vulnerable to a type of attack known as *frequency analysis*. The most common letters in the English language are *E*, *T*, *A*, *O*, *I*, *N*, *S*, *H, R, D, L, and U*. This is also known as "ETAOIN SHRDLU." An attacker seeking to break a Caesar-style cipher encoding message that was written in English merely needs to find the most common letters in the encrypted text and experiment with substitutions of these common letters to help determine the pattern.

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 (P) to determine the ciphertext (C). You account for the wrap-around by using the modulo function discussed in the section “Cryptographic Mathematics,” earlier in this chapter. 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 (such as a keyword) 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	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

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

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 plaintext could be LAUNCHNOW and the key could be *MILES*. 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 (*L*).
    b. Next, locate the row headed by the first character of the key (*M*).
    c. Finally, locate where these two items intersect, and write down the letter that appears there (*X*). This is the ciphertext for that letter position.
4. Repeat steps 1 through 3 for each letter in the plaintext version. The results are shown in Table 6.5.

TABLE 6.5 Using the Vigenère system

Stage of the process	**Letters**
Plaintext	L A U N C H N O W
Key	M I L E S M I L E
Ciphertext	X I F R U T V Z A

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*:

$$C = P \oplus K$$
$$P = C \oplus K$$

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 encryption key 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 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 plaintext values used. In fact, a common practice when using paper pads is to destroy the page of keying material after it is used to prevent reuse.
- 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.

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.

NOTE If you're interested in learning more about one-time pads, there is a great description with photos and examples at `www.cryptomuseum.com/crypto/otp/index.htm`.

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, newspaper, or magazine. 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, Table 6.6 shows the encryption operation for the first two words of the ciphertext.

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

TABLE 6.6 The encryption operation

Operation component	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
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. This section covers 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 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 random 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. Longer keys provide higher levels of security by increasing the size of the key space, rendering brute-force attacks more difficult.

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.

When the Data Encryption Standard (DES) was created in 1975, 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.

In addition to choosing keys that are long and will remain secure for the expected length of time that the information will remain confidential, you should also implement some other key management practices:

- Always store secret keys securely and, if you must transmit them over a network, do so in a manner that protects them from unauthorized disclosure.
- Select keys using an approach that has as much randomness as possible, taking advantage of the entire key space.
- Destroy keys securely when they are no longer needed.

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 (with “C” representing a ciphertext message and “P” representing a plaintext message).

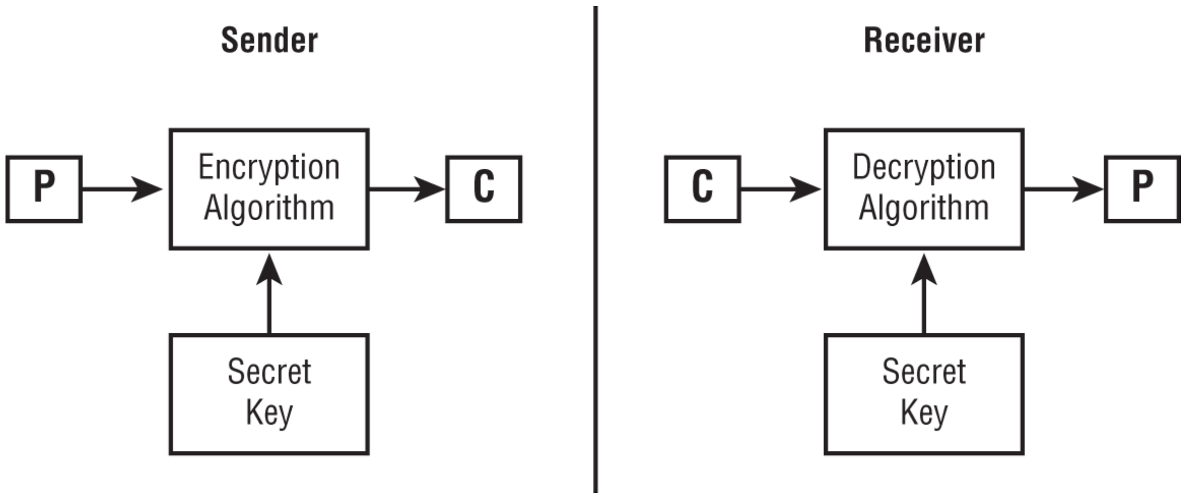


FIGURE 6.3 Symmetric key cryptography

TIP If you find yourself getting confused about the difference between symmetric and asymmetric cryptography, it may be helpful to remember that "same" is a synonym for "symmetric" and "different" is a synonym for asymmetric. In symmetric cryptography, the message is encrypted and decrypted with the same key, whereas in asymmetric cryptography, encryption and decryption use different (but related) keys.

In some cases, symmetric cryptography may be used with temporary keys that exist only for a single session. In those cases, the secret key is known as an *ephemeral key*. The most common example of this is the Transport Layer Security (TLS) protocol, which uses asymmetric cryptography to set up an encrypted channel and then switches to symmetric cryptography using an ephemeral key. You'll learn more about this topic in Chapter 7.

NOTE 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 secret key.

Keys must be regenerated often. Each time a participant leaves the group, all secret keys known by that participant must be discarded. In automated encryption systems, keys may be regenerated based on the length of time that has passed, the amount of data exchanged, or the fact that a session goes idle or is terminated.

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 and bulk encryption tasks.

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 provide a solution to the weaknesses of symmetric key encryption. *Public key algorithms* are the most common example of asymmetric algorithms. 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 from the key pair can decrypt it, and vice versa.

Figure 6.4 shows the algorithm used to encrypt and decrypt messages in a public key cryptosystem (with “C” representing a ciphertext message and “P” representing a plaintext message). 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 own private key.

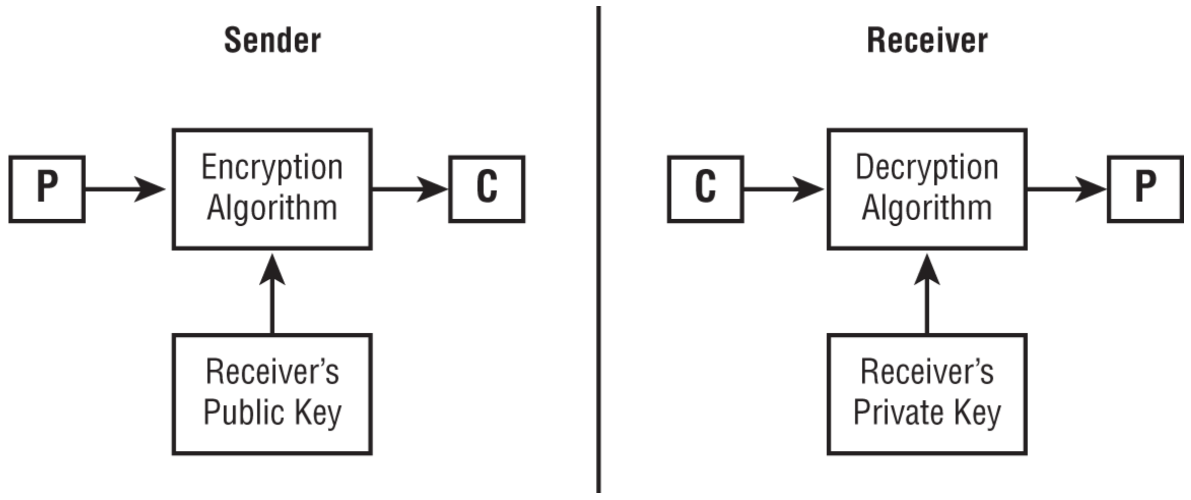


FIGURE 6.4 Asymmetric key cryptography

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 figures shown in Table 6.7. Obviously, the larger the population, the less likely a symmetric cryptosystem will be suitable to meet its needs.

TABLE 6.7 Symmetric and asymmetric key comparison

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 confidentiality, 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. Asymmetric cryptography may be used to create digital signatures that provide nonrepudiation, as discussed in Chapter 7.

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. This approach of combining symmetric and asymmetric cryptography is known as *hybrid cryptography*.

Table 6.8 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 6.8 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 certificates
Confidentiality	Confidentiality, integrity (via hashing), authenticity, nonrepudiation (via digital signatures)

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 (also known as hash values or fingerprints) are fixed-length 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 message digest value for two different messages 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.

Symmetric Cryptography

You've learned the basic concepts underlying symmetric key cryptography, asymmetric key cryptography, and hashing functions. In this section, we'll take an in-depth look at several common symmetric cryptosystems.

Block Cipher Modes of Operation

The symmetric key cryptography block cipher modes of operation describe the different ways that cryptographic algorithms may transform data to achieve sufficient complexity that offers protection against attack. The major modes of operation are:

- Electronic Codebook (ECB) mode
- Cipher Block Chaining (CBC) mode
- Cipher Feedback (CFB) mode
- Output Feedback (OFB) mode
- Counter (CTR) mode
- Galois/Counter mode (GCM)
- Counter with Cipher Block Chaining Message Authentication Code (CCM) mode

Electronic Codebook Mode

Electronic Codebook (ECB) mode is the simplest mode to understand and the least secure. Each time the algorithm processes a fixed block of data, it simply encrypts the block using the chosen secret key. This means that if the algorithm encounters the same block multiple times, it will produce the same encrypted block. If an enemy were eavesdropping on the communications, they could simply build a “codebook” of all the possible encrypted values. After a sufficient number of blocks were gathered, cryptanalytic techniques could be used to decipher some of the blocks and break the encryption scheme.

This vulnerability makes it impractical to use ECB mode on any but the shortest transmissions. In everyday use, ECB is used only for exchanging small amounts of data, such as keys and parameters used to initiate other cryptographic modes as well as the cells in a database.

Cipher Block Chaining Mode

In *Cipher Block Chaining (CBC)* mode, each block of unencrypted text is XORed with the block of ciphertext immediately preceding it before it is encrypted. The decryption process simply decrypts the ciphertext and reverses the XOR operation. CBC implements an initialization vector (IV) and XORs it with the first block of the message, producing a unique output every time the operation is performed. The IV must be sent to the recipient, perhaps by tacking the IV onto the front of the completed ciphertext in plain form or by protecting it with ECB mode encryption using the same key used for the message. One important consideration when using CBC mode is that errors propagate—if one block is corrupted during transmission, it becomes impossible to decrypt that block and the next block as well.

Cipher Feedback Mode

Cipher Feedback (CFB) mode is the streaming cipher version of CBC. In other words, CFB operates against data produced in real time. However, instead of breaking a message into blocks, it uses memory buffers of the same block size. As the buffer becomes full, it is encrypted and then sent to the recipients. Then the system waits for the next buffer to be filled as the new data is generated before it is in turn encrypted and then transmitted. Other than the change from preexisting data to real-time data, CFB operates in the same fashion as CBC. It uses an IV, and it uses chaining.

Output Feedback Mode

In *Output Feedback (OFB) mode*, ciphers operate in almost the same fashion as they do in CFB mode. However, instead of XORing an encrypted version of the previous block of ciphertext, OFB XORs the plaintext with a seed value. For the first encrypted block, an IV is used to create the seed value. Future seed values are derived by running the algorithm on the previous seed value. The major advantages of OFB mode are that there is no chaining function and transmission errors do not propagate to affect the decryption of future blocks.

Counter Mode

Counter (CTR) mode uses a stream cipher similar to that used in CFB and OFB modes. However, instead of creating the seed value for each encryption/decryption operation from the results of the previous seed values, it uses a simple counter that increments for each operation. As with OFB mode, errors do not propagate in CTR mode.

CTR mode allows you to break an encryption or decryption operation into multiple independent steps. This makes CTR mode well suited for use in parallel computing.

Galois/Counter Mode

Galois/Counter mode (GCM) takes the standard CTR mode of encryption and adds data authenticity controls to the mix, providing the recipient assurances of the integrity of the data received. This is done by adding *authentication tags* to the encryption process.

Counter with Cipher Block Chaining Message Authentication Code Mode

Similar to GCM, *Counter with Cipher Block Chaining Message Authentication Code mode (CCM)* combines a confidentiality mode with a data authenticity process. In this case, CCM ciphers combine the Counter (CTR) mode for confidentiality with the Cipher Block Chaining Message Authentication Code (CBC-MAC) algorithm for data authenticity.

Currently, CCM is used only with block ciphers that have a 128-bit block length (such as the AES algorithm) and require the use of a nonce that must be changed for each transmission.

> NOTE GCM and CCM modes both include data authenticity in addition to confidentiality. They are, therefore, known as authenticated modes of encryption. ECB, CBC, CFB, OFB, and CTR modes only provide confidentiality and are, therefore, known as unauthenticated modes.

Data Encryption Standard

The U.S. government published the Data Encryption Standard (DES) in 1977 as a proposed standard cryptosystem for all government communications. Because of flaws in the algorithm, cryptographers and the federal government no longer consider DES secure. It is widely believed that intelligence agencies routinely decrypt DES-encrypted information. DES was superseded by the Advanced Encryption Standard in December 2001. It is still important to understand DES because it is the building block of Triple DES (3DES), a stronger encryption algorithm discussed in the next section.

DES is a 64-bit block cipher that has five modes of operation: Electronic Codebook (ECB) mode, Cipher Block Chaining (CBC) mode, Cipher Feedback (CFB) mode, Output Feedback (OFB) mode, and Counter (CTR) mode. All of the DES modes operate on 64 bits of plaintext at a time to generate 64-bit blocks of ciphertext. The key used by DES is 56 bits long.

DES uses a long series of exclusive OR (XOR) operations to generate the ciphertext. This process is repeated 16 times for each encryption/decryption operation. Each repetition is commonly referred to as a *round* of encryption, explaining the statement that DES performs 16 rounds of encryption. Each round generates a new key that is then used as the input to subsequent rounds.

> NOTE As mentioned, DES uses a 56-bit key to drive the encryption and decryption process. However, you may read in some literature that DES uses a 64-bit key. This is not an inconsistency—there's a perfectly logical explanation. The DES specification calls for a 64-bit key. However, of those 64 bits, only 56 actually contain keying information. The remaining 8 bits are supposed to contain parity information to ensure that the other 56 bits are accurate. In practice, however, those parity bits are rarely used. You should commit the 56-bit figure to memory.

Triple DES

As mentioned in previous sections, the Data Encryption Standard's (DES) 56-bit key is no longer considered adequate in the face of modern cryptanalytic techniques and supercomputing power. However, an adapted version of DES, Triple DES (3DES), uses the same algorithm to produce encryption that is stronger but that is no longer considered adequate to meet modern requirements. For this reason, 3DES encryption should be avoided, although it may still be supported by some products.

> WARNING As of January 1, 2024, the U.S. federal government formally disallowed the use of 3DES for government data. The algorithm is still used in private industry.

There are several different variants of 3DES that each use different numbers of independent keys. The first two, DES-EDE3 and DES-EEE3, use three independent keys: K_1, K_2, and K_3. The difference between the two are the operations used, which are represented by the letter *E* for encryption and *D* for decryption. DES-EDE3 encrypts the data with K1, decrypts the resulting ciphertext with K2, and then encrypts that text with K3. DES-EDE3 can be expressed using the following notation, where `E(K,P)` represents the encryption of plaintext *P* with key *K*, and `D(K,C)` represents the decryption of ciphertext *C* with key *K*:

```
E(K1,D(K2,E(K3,P)))
```

DES-EEE3, on the other hand, encrypts the data with all three keys in sequential order, and may be represented as follows:

```
E(K1,E(K2,E(K3,P)))
```

TIP If you find yourself wondering why there is a decryption operation in the middle of EDE3 mode, that's an arcane artifact of the process used to create the algorithm and provide backward compatibility with DES. Encryption and decryption are reversible operations, so even though the decryption function is used, it can still be thought of as a round of encryption.

Mathematically, DES-EEE3 and DES-EDE3 should have an effective key length of 168 bits. However, known attacks against this algorithm reduce the effective strength to 112 bits.

NOTE This discussion raises an obvious question—what happened to Double DES (2DES)? You'll read in Chapter 7 that Double DES was tried but quickly abandoned when it was proven that an attack known as the meet-in-the-middle attack rendered it no more secure than standard DES.

International Data Encryption Algorithm

The International Data Encryption Algorithm (IDEA) block cipher was developed in response to complaints about the insufficient key length of the DES algorithm. Like DES, IDEA operates on 64-bit blocks of plaintext/ciphertext. However, it begins its operation with a 128-bit key. This key is broken up in a series of operations into 52 16-bit subkeys. The subkeys then act on the input text using a combination of XOR and modulus operations to produce the encrypted/decrypted version of the input message. IDEA is capable of operating in the same five modes used by DES: ECB, CBC, CFB, OFB, and CTR.

WARNING All of this material on key length block size and the number of rounds of encryption may seem dreadfully boring; however, it's important material, so be sure to brush up on it while preparing for the exam.

The IDEA algorithm was patented by its Swiss developers. However, the patent expired in 2012, and it is now available for unrestricted use. One popular implementation of IDEA is found in Phil Zimmermann's popular Pretty Good Privacy (PGP) secure email package. Chapter 7 covers PGP in further detail.

Blowfish

Bruce Schneier's Blowfish block cipher is another alternative to DES and IDEA. Like its predecessors, Blowfish operates on 64-bit blocks of text. However, it extends IDEA's key strength even further by allowing the use of variable-length keys ranging from a relatively insecure 32 bits to an extremely strong 448 bits. Obviously, the longer keys will result in corresponding increases in encryption/decryption time. However, time trials have established Blowfish as a much faster algorithm than both IDEA and DES. Also, Schneier released Blowfish for public use with no license required. Blowfish encryption is built into a number of commercial software products and operating systems. A number of Blowfish libraries are also available for software developers, and it is often used for Secure Shell (SSH) connections.

SKIPJACK

Developed by the U.S. National Security Agency (NSA), the algorithm code named SKIPJACK was approved for use by the U.S. government in Federal Information Processing Standard (FIPS) 185, the Escrowed Encryption Standard (EES) in 1994. SKIPJACK uses an 80-bit key and operates on 64-bit blocks of text. SKIPJACK was quickly embraced by the U.S. government and provided the cryptographic routines supporting the Clipper and Capstone encryption chips.

However, SKIPJACK had an added twist—it supports the escrow of encryption keys. At the time, two government agencies, the National Institute of Standards and Technology (NIST) and the U.S. Department of the Treasury, each held a portion of the information required to reconstruct a SKIPJACK key. When law enforcement authorities obtained legal authorization, they would contact the two agencies to obtain the pieces of the key, enabling them to decrypt communications between the affected parties.

SKIPJACK and the Clipper chip were not embraced by the cryptographic community at large because of its mistrust of the escrow procedures in place within the U.S. government.

Rivest Ciphers

Ron Rivest, of Rivest-Shamir-Adleman (RSA) Data Security, created a series of symmetric ciphers over the years known as the Rivest Ciphers (RC) family of algorithms. Several of these, RC4, RC5, and RC6, have particular importance today.

Rivest Cipher 4 (RC4)

RC4 is a stream cipher developed by Rivest in 1987 and very widely used during the decades that followed. It uses a single round of encryption and allows the use of variable-length keys ranging from 40 bits to 2,048 bits. RC4's adoption was widespread because it was integrated into the Wired Equivalent

Privacy (WEP), Wi-Fi Protected Access (WPA), Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols.

A series of attacks against this algorithm render it insecure for use today. WEP, WPA, and SSL no longer meet modern security standards for both this and other reasons. TLS no longer allows the use of RC4 as a stream cipher.

Rivest Cipher 5 (RC5)

RC5 is a block cipher of variable block sizes (32, 64, or 128 bits) that uses key sizes between 0 (zero) length and 2,040 bits. It is important to note that RC5 is not simply the next version of RC4. In fact, it is completely unrelated to the RC4 cipher. Instead, RC5 is an improvement on an older algorithm called RC2 that is no longer considered secure.

RC5 is the subject of brute-force cracking attempts. A large-scale effort leveraging massive community computing resources cracked a message encrypted using RC5 with a 64-bit key, but this effort took more than four years to crack a single message.

Rivest Cipher 6 (RC6)

RC6 is a block cipher that was developed as the next version of RC5. It uses a 128-bit block size and allows the use of 128-, 192-, or 256-bit symmetric keys. This algorithm was one of the candidates for selection as the Advanced Encryption Standard (AES) discussed in the next section, but it was not selected and is not widely used today.

Advanced Encryption Standard

In October 2000, NIST announced that the Rijndael (pronounced "rhine-doll") block cipher had been chosen as the replacement for DES. In November 2001, NIST released FIPS 197, which mandated the use of AES/Rijndael for the encryption of all sensitive but unclassified data by the U.S. government.

The Advanced Encryption Standard (AES) cipher allows the use of three key strengths: 128 bits, 192 bits, and 256 bits. AES only allows the processing of 128-bit blocks, but Rijndael exceeded this specification, allowing cryptographers to use a block size equal to the key length. The number of encryption rounds depends on the key length chosen:

- 128-bit keys require 10 rounds of encryption.
- 192-bit keys require 12 rounds of encryption.
- 256-bit keys require 14 rounds of encryption.

CAST

The CAST algorithms, named after their creators, Carlisle Adams and Stafford Tavares, are another family of symmetric key block ciphers that are integrated into some security solutions. The CAST algorithms use a Feistel network and come in two forms:

- CAST-128 uses either 12 or 16 rounds of Feistel network encryption with a key size between 40 and 128 bits on 64-bit blocks of plaintext.
- CAST-256 uses 48 rounds of encryption with a key size of 128, 160, 192, 224, or 256 bits on 128-bit blocks of plaintext.

The CAST-256 algorithm was a candidate for the Advanced Encryption Standard but was not selected for that purpose.

Comparison of Symmetric Encryption Algorithms

There are many symmetric encryption algorithms you need to be familiar with. Table 6.9 lists several common and well-known symmetric encryption algorithms along with their block size and key size.

TABLE 6.9 Symmetric encryption memorization chart

Algorithm Name	**Block size (Bits)**	**Key size (Bits)**
Advanced Encryption Standard (AES)	128	128, 192, 256
Rijndael	Variable	128, 192, 256
Blowfish	64	32–448
Data Encryption Standard (DES)	64	56
International Data Encryption Algorithm (IDEA)	64	128
Rivest Cipher 4 (RC4)	N/A (Stream cipher)	40–2,048
Rivest Cipher 5 (RC5)	32, 64, 128	0–2,040
Rivest Cipher 6 (RC6)	128	128, 192, 256
SKIPJACK	64	80
Triple DES (3DES)	64	112 or 168
CAST-128	64	40–128
CAST-256	128	128, 160, 192, 224, 256

Symmetric Key Management

Because cryptographic keys contain information essential to the security of the cryptosystem, it is incumbent upon cryptosystem users and administrators to take extraordinary measures to protect the security of the keying material. These security measures are collectively known as *key management practices*. They include safeguards surrounding the creation, distribution, storage, destruction, recovery, and escrow of symmetric (secret) keys.

Creation and Distribution of Symmetric Keys

As previously mentioned, one of the major problems underlying symmetric encryption algorithms is the secure distribution of the secret keys required to operate the algorithms. The three main methods used to exchange secret keys securely are offline distribution, public key encryption, and the Diffie–Hellman key exchange algorithm.

Offline Distribution The most technically simple (but physically inconvenient) method involves the physical exchange of key material. One party provides the other party with a sheet of paper or piece of storage media containing the secret key. In many hardware encryption devices, this key material comes in the form of an electronic device that resembles an actual key that is inserted into the encryption device. However, every offline key distribution method has its own inherent flaws. If keying material is sent through the mail, it might be intercepted. Telephones can be wiretapped. Papers containing keys might be inadvertently thrown in the trash or lost. The use of offline distribution is cumbersome for end users, particularly when they are located in geographically distant locations.

Public Key Encryption Many communicators want to obtain the speed benefits of secret key encryption without the hassles of key distribution. For this reason, many people use public key encryption to set up an initial communications link. Once the link is successfully established and the parties are satisfied as to each other's identity, they exchange a secret key over the secure public key link. They then switch communications from the public key algorithm to the secret key algorithm and enjoy the increased processing speed. In general, secret key encryption is thousands of times faster than public key encryption.

Diffie–Hellman In some cases, neither public key encryption nor offline distribution is sufficient. Two parties might need to communicate with each other, but they have no physical means to exchange key material, and there is no public key infrastructure in place to facilitate the exchange of a secret key. In situations like this, key exchange algorithms like the Diffie–Hellman algorithm prove to be extremely useful mechanisms. You'll find a complete discussion of Diffie–Hellman in Chapter 7.

Storage and Destruction of Symmetric Keys

Another major challenge with the use of symmetric key cryptography is that all of the keys used in the cryptosystem must be kept secure. This includes following best practices surrounding the storage of encryption keys:

- Never store an encryption key on the same system where encrypted data resides. This just makes it easier for the attacker!
- For sensitive keys, consider providing two different individuals with half of the key. They then must collaborate to re-create the entire key. This is known as the principle of *split knowledge* (discussed earlier in this chapter).

When a user with knowledge of a secret key leaves the organization or is no longer permitted access to material protected with that key, the keys must be changed, and all encrypted materials must be reencrypted with the new keys.

When choosing a key storage mechanism, you have three major options available to you:

- *Software-based storage mechanisms* store keys as digital objects on the system where they are used. For example, this might involve storing the key on the local file system. More advanced software-based mechanisms may use specialized applications to protect those keys, including the use of secondary encryption to prevent unauthorized access to the keys. Software-based approaches are generally simple to implement but introduce the risk of the software mechanism being compromised.
- *Hardware-based storage mechanisms* are dedicated hardware devices used to manage cryptographic keys. These may be personal devices, such as flash drives or smartcards that store a key used by an individual, or they may be enterprise devices, called *hardware security modules (HSMs)*, that manage keys for an organization. Hardware

approaches are more complex and expensive to implement than software approaches, but they offer added security.

- *Cloud-based storage mechanisms* take the HSM approach and implement it in the data centers of cloud service providers. Major cloud service providers, including AWS and Microsoft, offer HSM solutions for secure key management.

Key Escrow and Recovery

Cryptography is a powerful tool. Like most tools, it can be used for a number of beneficent purposes, but it can also be used with malicious intent. To gain a handle on the explosive growth of cryptographic technologies, governments around the world have floated ideas to implement key escrow systems. These systems allow the government, under limited circumstances such as a court order, to obtain the cryptographic key used for a particular communication from a central storage facility.

Two major approaches to key escrow have been proposed over the past decade:

Fair Cryptosystems In this escrow approach, the secret keys used in a communication are divided into two or more pieces, each of which is given to an independent third party. Each of these pieces is useless on its own, but they may be recombined to obtain the secret key. When the government obtains legal authority to access a particular key, it provides evidence of the court order to each of the third parties and then reassembles the secret key.

Escrowed Encryption Standard This escrow approach provides the government or another authorized agent with a technological means to decrypt ciphertext. It was the approach proposed for the Clipper chip.

It's highly unlikely that government regulators will ever overcome the legal and privacy hurdles necessary to implement key escrow on a widespread basis. The technology is certainly available, but the general public will likely never accept the potential government intrusiveness it facilitates.

There are, however, legitimate uses for key escrow within an organization. Key escrow and recovery mechanisms prove useful when an individual leaves the organization and other employees require access to their encrypted data, or when a key is simply lost. In these approaches, key *recovery agents (RAs)* have the ability to recover the encryption keys assigned to individual users. This is, of course, an extremely powerful privilege, as an RA could gain access to any user's encryption key. For this reason, many organizations choose to adopt a mechanism known as *M of N control* for key recovery. In this approach, there is a group of individuals of size N in an organization who are granted RA privileges. If they wish to recover an encryption key, a subset of at least M of them must agree to do so. For example, in an M-of-N control system where M=3 and N=12, there are 12 authorized recovery agents, of whom 3 must collaborate to retrieve an encryption key.

Cryptographic Life Cycle

With the exception of the one-time pad, all cryptographic systems have a limited life span. Moore's law, a commonly cited trend in the advancement of computing power, states that the processing capabilities of a state-of-the-art microprocessor will double approximately every two years. This means that, eventually, processors will reach the amount of strength required to simply guess the encryption keys used for a communication.

Security professionals must keep this cryptographic life cycle in mind when selecting an encryption algorithm and have appropriate governance controls in place to ensure that the algorithms, protocols, and key lengths selected are sufficient to preserve the integrity of a cryptosystem for however long it is necessary to keep the information it is protecting secret. Security professionals can use the following algorithm and protocol governance controls:

- Specifying the cryptographic algorithms (such as AES and RSA) acceptable for use in an organization

- Identifying the acceptable key lengths for use with each algorithm based on the sensitivity of information transmitted
- Enumerating the secure transaction protocols (such as TLS) that may be used

For example, if you're designing a cryptographic system to protect the security of business plans that you expect to execute next week, you don't need to worry about the theoretical risk that a processor capable of decrypting them might be developed a decade from now. On the other hand, if you're protecting the confidentiality of information that could be used to construct a nuclear bomb, it's virtually certain that you'll still want that information to remain secret 10 years in the future!

Summary

Cryptographers and cryptanalysts are in a never-ending race to develop more secure cryptosystems and advanced cryptanalytic techniques designed to circumvent those systems.

Cryptography dates back as early as Caesar and has been an ongoing topic of study for many years. In this chapter, you learned some of the fundamental concepts underlying the field of cryptography and gained a basic understanding of the terminology used by cryptographers.

This chapter also examined the similarities and differences between symmetric key cryptography (where communicating parties use the same key) and asymmetric key cryptography (where each communicator has a pair of public and private keys). You learned how hashing may be used to guarantee integrity and how hashes play a role in the digital signature process that guarantees nonrepudiation.

We then analyzed some of the symmetric algorithms currently available and their strengths and weaknesses. We wrapped up the chapter by taking a look at the cryptographic life cycle and the role of algorithm/protocol governance in enterprise security.

The next chapter expands this discussion to cover contemporary public key cryptographic algorithms. Additionally, some of the common cryptanalytic techniques used to defeat both types of cryptosystems will be explored.

Study Essentials

Understand the role that confidentiality, integrity, and nonrepudiation play in cryptosystems. Confidentiality is one of the major goals of cryptography. It protects the secrecy of data while it is both at rest and in transit. Integrity provides the recipient of a message with the assurance that data was not altered (intentionally or unintentionally) between the time it was created and the time it was accessed. Nonrepudiation provides undeniable proof that the sender of a message actually authored it. It prevents the sender from subsequently denying that they sent the original message.

Know how cryptosystems can be used to achieve authentication goals. Authentication provides assurances as to the identity of a user. One possible scheme that uses authentication is the challenge-response protocol, in which the remote user is asked to encrypt a message using a key known only to the communicating parties. Authentication can be achieved with both symmetric and asymmetric cryptosystems.

Be familiar with the basic terminology of cryptography. When a sender wants to transmit a private message to a recipient, the sender takes the plaintext (unencrypted) message and encrypts it using an algorithm and a key. This produces a ciphertext message that is transmitted to the recipient. The recipient then uses a similar algorithm and key to decrypt the ciphertext and re-create the original plaintext message for viewing.

Understand the difference between a code and a cipher and explain the basic types of ciphers. Codes are cryptographic systems of symbols that operate on words or phrases and are sometimes secret but don't always provide confidentiality. Ciphers, however, are always meant to hide the true meaning of a message. Know how the following types of ciphers work: transposition ciphers, substitution ciphers (including one-time pads), stream ciphers, and block ciphers.

Know the requirements for successful use of a one-time pad. For a one-time pad to be successful, the key must be generated randomly without any known pattern. The key must be at least as long as the message to be encrypted. The pads must be protected against physical disclosure, and each pad must be used only one time and then discarded.

Understand split knowledge. Split knowledge means that the information or privilege required to perform an operation is divided among multiple users. This ensures that no single person has sufficient privileges to compromise the security of the environment. M of N Control is an example of split knowledge used in key recovery and other sensitive tasks.

Understand work function (work factor). Work function, or work factor, is a way to measure the strength of a cryptography system by measuring the effort in terms of cost and/or time to decrypt messages. Usually, the time and effort required to perform a complete brute-force attack against an encryption system is what a work function rating represents. The security and protection offered by a cryptosystem is directly proportional to the value of its work function/factor.

Understand the importance of key security. Cryptographic keys provide the necessary element of secrecy to a cryptosystem. Modern cryptosystems utilize keys that are at least 128 bits long to provide adequate security.

Know the differences between symmetric and asymmetric cryptosystems. Symmetric key cryptosystems (or secret key cryptosystems) rely on the use of a shared secret key. They are much faster than asymmetric algorithms, but they lack support for the following: scalability, easy key distribution, and nonrepudiation. Asymmetric cryptosystems use public-private key pairs for communication between parties but operate much more slowly than symmetric algorithms.

Be able to explain the basic operational modes of symmetric cryptosystems. Symmetric cryptosystems operate in several discrete modes, including Electronic Codebook (ECB) mode, Cipher Block Chaining (CBC) mode, Cipher Feedback (CFB) mode, Output Feedback (OFB) mode, Counter (CTR) mode, Galois/Counter mode (GCM), and Counter with Cipher Block Chaining Message Authentication Code mode (CCM). ECB mode is considered the least secure and is used only for short messages. 3DES uses three iterations of DES with two or three different keys to increase the effective key strength to 112 or 168 bits, respectively.

Know the Advanced Encryption Standard (AES). The Advanced Encryption Standard (AES) uses the Rijndael algorithm and is the U.S. government standard for the secure exchange of sensitive but unclassified data. AES uses key lengths of 128, 192, and 256 bits and a fixed block size of 128 bits to achieve a much higher level of security than that provided by the older DES algorithm.

Written Lab

1. What is the major obstacle preventing the widespread adoption of one-time pad cryptosystems to ensure data confidentiality?
2. Encrypt the message "I will pass the CISSP exam and become certified next month" using columnar transposition with the keyword SECURE.
3. Decrypt the message "F R Q J U D W X O D W L R Q V B R X J R W L W" using the Caesar ROT3 substitution cipher.

Review Questions

1. Ryan is responsible for managing the cryptographic keys used by his organization. Which of the following statements are correct about how he should select and manage those keys? (Choose all that apply.)