Chapter 8

Encryption

Chapter Objectives

After reading this chapter and completing the exercises, you will be able to do the following:

- Explain the basics of encryption
- Discuss modern cryptography methods
- Select appropriate cryptography for your organization

Introduction

There are many aspects of computer and information security. *Encryption*, the process of scrambling a message or other information so that it cannot be easily read by someone intercepting the message, is one of the most critical parts to the security puzzle. If you have the best firewall, very tight security policies, hardened operating systems, virus scanners, intrusion detection software, antispyware, and every other computer security angle covered but send your data in raw plain text, then you simply are not secure.

In this chapter, you will obtain what can be termed a "manager's understanding" of *cryptography*—the art of writing in or deciphering secret code. It is important to understand that this chapter will not make you a cryptographer. In fact, reading several volumes on encryption would not accomplish that lofty goal. Rather, this chapter is designed to give you a basic overview of what encryption is, some idea of how it works, and enough information so that you can make intelligent decisions about what sorts of encryption to incorporate in your organization. You will learn the basic history of encryption and the fundamental concepts, and after you have completed the exercises at the end of the chapter, you'll have enough knowledge to at least be able to ask the right questions. Now, this does not mean we won't cover some technical details. We will. But the goal of this chapter is to give you a broad understanding of the relevant concepts.

It is also important to understand that some concepts in cryptography can be difficult. While the general ideas are readily understandable by most readers, you may have difficulty with some concepts. That is normal and should not be a concern. You may have to read some parts of the chapter a few times to fully get some concepts.

We will go into the actual process of some of the cryptography algorithms presented. For example, we will show you the process of DES and RSA. It is beyond the scope of this book to go in depth into every cryptographic algorithm available, but it is useful for you to see a few of them described in detail.

It is not critical for a security practitioner or aspiring security practitioner, to have in-depth knowledge of cryptographic algorithms. In this chapter, some of the concepts we cover, particularly in regard to asymmetric cryptography, may be difficult to grasp. It is acceptable if you don't get 100% of those concepts, particularly the math-related concepts, on your first reading of the material. Some topics in this chapter are likely to require a bit more study and effort than other material in this book. In Chapter 9, "Computer Security Technology," you will see some applications of cryptography, such as SSL/TLS, digital certificates, and virtual private networks.

Cryptography Basics

The aim of cryptography is not to hide the existence of a message but rather to hide its meaning—in a process known as *encryption*. To make a message unintelligible, it is scrambled according to a particular algorithm, which is agreed upon beforehand between the sender and the intended recipient. Thus, the recipient can reverse the scrambling protocol and make the message comprehensible. This reversal of the scrambling is referred to as *decryption*. The advantage of using encryption/decryption is that, for someone who doesn't know the scrambling protocol, the message is difficult to re-create.

There are two basic types of cryptography in use today: symmetric and asymmetric. *Symmetric* means the same key is used to encrypt the message and to decrypt the message. With *asymmetric* cryptography, a different key is used to encrypt the message than is used to decrypt the message. That may sound a bit odd, and you may be wondering how that is possible. Later in this chapter, we will explore exactly how it works. For now the important point is to understand the basic concept of symmetric and asymmetric cryptography.

History of Encryption

The idea of encryption is probably as old as written communication. The basic concept is actually fairly simple: Messages must be changed in such a way that they cannot be easily read by an enemy but so they can be easily decoded by the intended recipient. In this section, we will examine a few historical methods of encryption. It should be noted that these are very old methods, and they cannot be used for secure communication today. The methods discussed in this section would be easily cracked, even by an amateur. However, examining them is wonderful for conveying the concepts of cryptography without having to incorporate a great deal of math, which is required of the more complex encryption methods.

FYI: Cryptographers

Encryption is a very broad and complex subject area. Even amateur cryptographers typically have some mathematical training and have studied cryptographic methods for several years.

If you are interested in learning more about the history of cryptography than what we touch upon here, you may wish to read one of the many books written on the subject. A brief history of cryptography is provided in Table 8.1.

TABLE 8.1 History of Cryptography

Year	Cryptography
500 to 600 BC	Atbash
< 10 BC	Polybius Square
45 BC to 45 AD	Caesar Cipher
50–120 AD	Scytale
1553 AD	Vigenere' Cipher
1910 to 1940	Enigma
1976	DES and Diffie-Hellman
1977	RSA
1985	Elliptic Curve Cryptography
1993	Blowfish Cipher
1998	Rijndael Published
2001	Rijndael selected as AES

- **The Stanford University History of Cryptography website:** http://cs.stanford.edu/people/eroberts/courses/soco/projects/public-key-cryptography/history.html

- **Cryptography.org:** http://cryptography.org
- **SANS History of Cryptography:** https://www.sans.org/reading-room/whitepapers/vpns/paper/730

Understanding the simple methods described here and other methods listed on the aforementioned websites should give you a sense of how cryptography works as well as what is involved in encrypting a message. Regardless of whether you go on to study modern, sophisticated encryption methods, it is important for you to have some basic idea of how encryption works at a conceptual level. Having a basic grasp of how encryption works, in principle, will help you better understand the concepts of any encryption method you encounter in the real world. Khan Academy has an online cryptography course that is good for beginners.

The Caesar Cipher

One of the oldest encryption methods is the *Caesar cipher*. This method is purported to have been used by the ancient roman caesars—thus, the name. It is actually quite simple to do. You choose some number by which to shift each letter of a text. For example, if the text is:

A cat

and you choose to shift by two letters, then the message becomes:

C ecv

Or, if you choose to shift by three letters, it becomes:

D fdw

Julius Caesar was reputed to have used a shift of three to the right. However, you can choose any shifting pattern you wish. You can shift either to the right or to the left by any number of spaces you like. Because this is a very simple method to understand, it makes a good place to start our study of encryption. It is, however, extremely easy to crack. You see, any language has a certain letter and word frequency, meaning that some letters are used more frequently than others). In the English language, the most common single-letter word is *A*. The most common three-letter word is *the.* Those two rules alone could help you decrypt a Caesar cipher. For example, if you saw a string of seemingly nonsense letters and noticed that a three-letter word was frequently repeated in the message, you might easily surmise that this word was *the*—and the odds are highly in favor of this being correct. Furthermore, if you frequently noticed a single-letter word in the text, it is most likely the letter *A.* You now have found the substitution scheme for *A*, *T*, *H*, and *E.* You can now either translate all of those letters in the message and attempt to surmise the rest or simply analyze the substitute letters used for *A*, *T*, *H*, and *E* and derive the substitution cipher that was used for this message. Decrypting a message of this type does not even require a computer. It can be done in less than 10 minutes using pen and paper by someone with no background in cryptography. There are other rules that will help make cracking this code even easier. For example, in the English language the two most common two letter combinations are *ee* and *oo*. That gives you even more to work on.

Another reason this algorithm can be easily cracked is an issue known as *key space*—the number of possible keys that can be used. In this case, when applied to the English alphabet, there are only 26 possible keys since there are only 26 letters in the English alphabet. This means you could simply try each possible key (+1, +2, +3,…+26) until one works. Trying all possible keys is referred to as a *brute-force attack*.

The substitution scheme you choose (for example, +2, +1) is referred to as a *substitution alphabet* (for example, *B* substitutes for *A*, *U* substitutes for *T*). Thus, the Caesar cipher is also referred to as a *mono-alphabet substitution* method, meaning that it uses a single substitution for the encryption. There are other mono-alphabet algorithms, but the Caesar cipher is the most widely known.

Atbash

In ancient times, Hebrew scribes used the Atbash substitution cipher to encrypt religious works such as the book of Jeremiah. Applying this cipher is fairly simple: Just reverse the order of the letters of the alphabet. This is, by modern standards, a very primitive and easy-to-break cipher.

The Atbash cipher is a Hebrew code that substitutes the first letter of the alphabet for the last and the second letter for the second to the last, and so on. It simply reverses the alphabet. For example, in English, *A* becomes *Z*, *B* becomes *Y*, *C* becomes *X*, and so on. Of course, the Hebrews used a different alphabet, with *aleph* being the first letter and *tav* being the last letter. However, I will use English examples to demonstrate this:

Attack at dawn

becomes:

Zggzxp zg wzdm

As you can see, the *A* is the first letter in the alphabet and is switched with *Z*, the last letter in the alphabet. Then the *T* is the 19th letter (and the 7th from the end) and gets swapped with *G*, the 7th letter from the beginning. This process is continued until the entire message is enciphered.

To decrypt the message, you simply reverse the process, and *Z* becomes *A*, *B* becomes *Y*, and so on. This is obviously a rather simple cipher and not used in modern times. However, it illustrates the basic concept of cryptography: to perform some permutation on the plain text to render it difficult to read by those who don't have the key to unscramble the cipher text. The Atbash cipher, like the Caesar cipher, is a single substitution cipher, so each letter in the plain text has a direct, one-to-one relationship with each letter in the cipher text. This means that the same letter and word frequency issues that can be used to crack the Caesar cipher can be used to crack the Atbash cipher.

Multi-Alphabet Substitution

Eventually, a slight improvement on the Caesar cipher, called *multi-alphabet substitution*, was developed. In this scheme, you select multiple numbers by which to shift letters (that is, multiple substitution alphabets). For example, if you select three substitution alphabets (12, 22, 13), then:

A CAT

becomes:

C ADV

Notice that the fourth letter starts over with another +2, and you can see that the first *A* was transformed to *C* and the second *A* was transformed to *D*. This makes it more difficult to decipher the underlying text. While this is harder to decrypt than a Caesar cipher, it is not overly difficult. It can be done with simple pen and paper and a bit of effort. It can be cracked very quickly with a computer. In fact, no one would use such a method today to send any truly secure message, for this type of encryption is considered very weak.

At one time multi-alphabet substitution was considered quite secure. In fact, a special version of this, called a *Vigenère cipher*, was used in the 1800s and early 1900s. The Vigenère cipher was invented in 1553 by Giovan Battista Bellaso. It is a method of encrypting alphabetic text by using a series of different Caesar ciphers based on the letters of a keyword. Figure 8.1 shows the Vigenère cipher.

	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

FIGURE 8.1 Vigenère cipher.

Match the letter of your keyword on the top with the letter of your plain text on the left to find the cipher text. For example, using the chart shown in Figure 8.1, if you are encrypting the word *cat* and your key is *horse*, then the cipher text is *jok*.

Rail Fence

All the ciphers we have examined so far are substitution ciphers. Another approach to classic cryptography is the transposition cipher. The *rail fence* cipher may be the most widely known transposition

cipher. You simply take the message you wish to encrypt and alter each letter on a different row. So "attack at dawn" is written as:

A t c a d w

t a k t a n

Next, you write down the text reading from left to right as one normally would, thus producing:

atcadwtaktan

In order to decrypt the message, the recipient must write it out on rows:

A t c a d w

t a k t a n

Then the recipient reconstructs the original message. Most texts use two rows as examples; however, this can be done with any number of rows you wish to use.

Enigma

It is really impossible to have a discussion about cryptography and not talk about Enigma. Contrary to popular misconceptions, the Enigma is not a single machine but rather a family of machines. The first version was invented by German engineer Arthur Scherbius near the end of World War I. It was used by several different militaries, not just the Nazi Germans.

Some military texts encrypted using a version of Enigma were broken by Polish cryptanalysts Marian Rejewski, Jerzy Rozycki, and Henryk Zygalski. The three basically reverse engineered a working Enigma machine and used that information to develop tools for breaking Enigma ciphers, including one tool named the *cryptologic bomb*.

The core of the Enigma machine was the rotors, or disks that were arranged in a circle with 26 letters on them. The rotors were lined up. Essentially, each rotor represented a different single substitution cipher. You can think of the Enigma as a sort of mechanical poly-alphabet cipher. The operator of the Enigma machine would be given a message in plain text and then type that message into Enigma. For each letter that was typed in, Enigma would provide a different cipher text, based on a different substitution alphabet. The recipient would type in the cipher text, and as long as both Enigma machines had the same rotor settings, the recipient's machine would provide the correct plain text. Figure 8.2 is a picture of an Enigma machine.

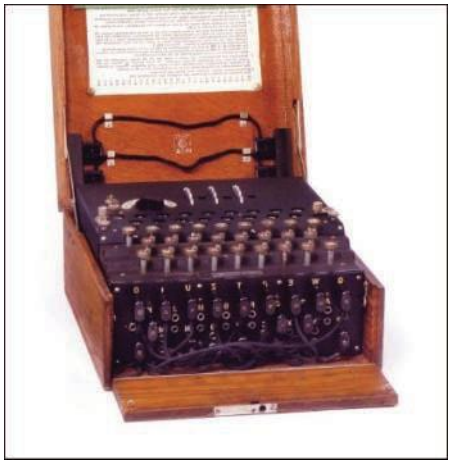

FIGURE 8.2 An Enigma machine.

There were actually several variations of the Enigma machine. The Naval Enigma machine was eventually cracked by British cryptographers working at the now famous Bletchley Park. Alan Turing and a team of analysts were able to eventually break the Naval Enigma machine. Many historians claim that this shortened World War II by as much as 2 years. This story is the basis for the 2014 movie *The Imitation Game*.

Binary Operations

Most symmetric ciphers make use of a binary operation called *exclusive or* (XOR). Before we examine modern symmetric ciphers, we will review basic binary math. Various operations on *binary numbers* (numbers made of only 0s and 1s) are well known to programmers and programming students. But for those who are not familiar with them, a brief explanation follows. When working with binary numbers, there are three operations not found in normal math: AND, OR, and XOR operations. Each is illustrated next.

AND

To perform the AND operation, you compare two binary numbers one place at a time. If both numbers have a 1 in both places, then the resultant number is a 1. If not, then the resultant number is a 0, as you see here:

```
1 1 0 1
1 0 0 1
_______
1 0 0 1
```

OR

The OR operation checks to see whether there is a 1 in either or both numbers in a given place. If so, then the resultant number is 1. If not, the resultant number is 0, as you see here:

```
1 1 0 1
1 0 0 1
_____________
1 1 0 1
```

XOR

The XOR operation impacts your study of encryption the most. It checks to see whether there is a 1 in a number in a given place but *not* in both numbers at that place. If it is in one number but not the other, then the resultant number is 1. If not, the resultant number is 0, as you see here:

```
1 1 0 1
1 0 0 1
_____________
0 1 0 0
```

XORing has a very interesting property in that it is reversible. If you XOR the resultant number with the second number, you get back the first number. And if you XOR the resultant number with the first number, you get the second number:

```
0 1 0 0
1 0 0 1
_____________
1 1 0 1
```

Binary encryption using the XOR operation opens the door for some rather simple encryption. Convert any message to binary numbers and then XOR that with some key, where the key is some string of digits (1s and 0s) that should be random (or at least as random as possible). Converting a message to a binary number is really a simple two-step process. First, convert a message to its ASCII code and then convert those codes to binary numbers. Each letter/number will generate an 8-bit binary number. Then you can use a random string of binary numbers of any given length as the key. Simply XOR your message with the key to get the encrypted text and then XOR it with the key again to retrieve the original message. This method is easy to use and great for computer science students; however, it does not work well for truly secure communications because the underlying letter and word frequency issues

remain. These issues provide valuable clues that even an amateur cryptographer can use to decrypt the message. However, this method does provide a valuable introduction to the concept of *single-key encryption*, which will be discussed in more detail in the next section. While simply XORing the text is not the method typically employed, single-key encryption methods are widely used today—and binary operations are often a part of the process.

Symmetric key cryptography often uses two processes: substitution and transposition. The *substitution* portion is accomplished by XORing the plain text message with the key. The *transposition* is done by swapping blocks of the text.

Modern Cryptography Methods

Modern cryptography methods, as well as computers, make cryptography a rather advanced science. What you have seen so far regarding cryptography is simply for educational purposes. As has been noted several times, you would not have a truly secure system if you implemented any of the previously mentioned encryption schemes. You may feel that this has been overstated in this text. However, it is critical that you have an accurate view of what encryption methods do and do not work. It is now time to discuss a few methods that are actually in use today.

Before we delve too deeply into this topic, let's start with some basic definitions you will need:

- **Key:** The bits that are combined with the plain text to encrypt it. In some cases this is random numbers; in other cases it is the result of some mathematical operation.
- **Plain text:** The unencrypted text.
- **Cipher text:** The encrypted text.
- **Algorithm:** A mathematical process for doing something.

Single-Key (Symmetric) Encryption

Basically, *single-key encryption* means that the same key is used to both encrypt and decrypt a message. This is also referred to as *symmetric key encryption*. There are two types of symmetric algorithms (or ciphers): stream and block. A block cipher divides the data into blocks (often 64-bit blocks, but newer algorithms sometimes use 128-bit blocks) and encrypts the data one block at a time. Stream ciphers encrypt the data as a stream of bits, one bit at a time.

Data Encryption Standard

Data Encryption Standard, or *DES*, was developed by IBM in the early 1970s and published in 1976. Yes, it is old, and it is no longer considered secure; however, it is worthy of study for two reasons. The first reason is that DES was the first modern symmetric cipher. The second reason is that the general

structure, often called a *Feistel function* or *Feistel cipher*, is still used in many modern algorithms. DES is a block cipher, which divides the plain text into 64-bit blocks and encrypts each block. The basic concept is as follows:

FYI: Block Ciphers and Stream Ciphers

When applying a key to plain text to encrypt it and produce the cipher text, you must also choose how to apply the key and the algorithm. In a block cipher, the key is applied to blocks (often 64 bits in size) at a time. This differs from a stream cipher that encrypts 1 bit at a time.

1. Data is divided into 64-bit blocks.
2. Each of those blocks is divided into two 32-bit halves.
3. One half is manipulated with substitution and XOR operations via a round function.
4. The two 32-bit halves are swapped.
5. This is repeated 16 times (16 rounds).

At the time DES was released, it was a marvelous invention. Even today the algorithm is still sound. However, the small key size, 56 bits, is not good enough to defend against brute-force attacks with modern computers. Many cryptography textbooks and university courses use DES as the basic template for studying all block ciphers. We will do the same and give this algorithm more attention than most of the others in this chapter.

For those new to security and cryptography, the brief facts listed earlier are enough. However, for those who want to delve a bit deeper, let's examine the details of the DES algorithm. DES uses a 56-bit cipher key applied to a 64-bit block. There is actually a 64-bit key, but 1 bit of every byte is used for error correction, leaving just 56 bits for actual key operations.

DES is a Feistel cipher with 16 rounds and a 48-bit round key for each round. They are called Feistel ciphers (also Feistel functions) after Horst Feistel, the inventor of the concept, and the primary inventor of DES. All Feistel ciphers work in the same way: They divide the block into two halves, apply a round function to one of those halves, and then swap the halves. This is done each round. The primary difference between different Feistel ciphers is what exactly occurs within the round function.

The first issue to address is the key schedule. A *key schedule*, which all block ciphers use, is a simple algorithm that will take the initial key the two parties derived and generate from that a slightly different key each round. DES does this by taking the original 56-bit key and slightly permuting it each round so that each round is applying a slightly different key but one that is based on the original cipher key. To generate the round keys, the 56-bit key is split into two 28-bit halves, and those halves are circularly shifted after each round by 1 or 2 bits to provide a different subkey each round. During the round key generation portion of the algorithm (recall that this is referred to as the *key schedule*) each round,

the two halves of the original cipher key (the 56 bits of key that the two endpoints of encryption must exchange) are shifted a specific amount. The end result is that for each of the 16 rounds of DES, the key is actually a little different from the key used in the previous round. All modern symmetric ciphers do something like this to improve the security of the cipher.

Once the round key has been generated for the current round, the next step is to address the half of the original block that is going to be input into the round function. Recall that the two halves are each 32 bits. The round key is 48 bits. This means that the round key does not match the size of the half block it is going to be applied to. You cannot really XOR a 48-bit round key with a 32-bit half block unless you simply ignore 16 bits of the round key. If you did so, you would basically be making the round key shorter and thus less secure, so this is not a good option. The 32-bit half needs to be expanded to 48 bits before it is XORed with the round key. This is accomplished by replicating some bits so that the 32-bit half becomes 48 bits.

This expansion process is actually quite simple. The 32 bits that are to be expanded are broken into 4-bit sections. The bits on each end are duplicated. If you divide 32 by 4, the answer is 8. So there are 8 of these 4-bit groupings. If you duplicate the end bits of each grouping, that will add 16 bits to the original 32, thus providing a total of 48 bits.

It is also important to keep in mind that it was the bits on each end that were duplicated. This will be a key item later in the round function. Perhaps this example will help you understand what is occurring at this point. Let us assume 32 bits, as shown here:

11110011010111111111000101011001

Now divide this into 8 sections of 4 bits each, as shown here:

1111 0011 0101 1111 1111 0001 0101 1001

Now each of these has its end bits duplicated, as you see here:

1111 becomes 111111

0011 becomes 000111

0101 becomes 001011

1111 becomes 111111

1111 becomes 111111

0001 becomes 000011

0101 becomes 001011

1001 becomes 110011

The resulting 48-bit string is now XORed with the 48-bit round key. Now you are done with the round key. Its only purpose was to XOR with the 32-bit half. It is now discarded, and on the next round another 48-bit round key will be derived from the two 28-bit halves of the 56-bit cipher key, using the key schedule we previously described.

Now we have the 48-bit output of the XOR operation. But this still does not seem to work. Don't we need 32 bits rather than 48? That 48 bits is now split into 8 sections of 6 bits each. Each of those 6-bit sections is going to be input into an s-box (substitution box), and only 4 bits will be output.

The 6-bit section is used as the input to an *s-box*, which is a table that takes input and produces an output based on that input. In other words, it is a substitution box that substitutes new values for the input. There are eight different s-boxes for DES, but below you can see one of them:

		Middle 4 Bits of Input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer Bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

An s-box is really just a hard-coded lookup table. The 2 bits on either end are shown in the left column, and the 4 bits in the middle are shown in the top row. They are matched, and the resulting value is the output of the s-box. For example, with the previous demonstration numbers we were using, our first block would be 111111. So you find 1xxxx1 on the left and x1111x on the top. The resulting value is 3 in decimal, or 0011 in binary.

Recall that during the expansion phase we simply duplicated the outermost bits, so when we come to the s-box phase and drop the outermost bits, no data is lost.

Since each s-box outputs 4 bits and there are 8 s-boxes, the result is 32 bits. That 32 bits is now exclusively ORed with the other half of the original block. Recall that we did nothing with that half originally. Now the two halves are swapped.

If you are new to cryptography, all of this might seem a bit confusing, even with the explanations provided. Many readers will need to reread this section a few times for it to become totally clear.

3DES

Triple DES (*3DES*) was created as a replacement for DES. At the time, the cryptography community was searching for a viable alternative. While that was still being worked on, 3DES was created as a stop-gap measure. It essentially applies DES three times with three different keys, thus the name 3DES.

There were variations of 3DES that used only two keys. The text was first encrypted with key A. The cipher text from that operation was then encrypted with key B. Then the cipher text from that operation was encrypted, this time reusing key A. The reason for this reuse is that creating good cryptographic keys is computationally intensive.

AES

Advanced Encryption Standard (*AES*) was the algorithm eventually chosen to replace DES. It is a block cipher that works on 128-bit blocks. It can have one of three key sizes: 128, 192, or 256 bits. The U.S. government selected AES to be the replacement for DES, and it is now the most widely used symmetric key algorithm.

AES, also known as *Rijndael block cipher*, was officially designated as a replacement for DES in 2001 after a 5-year process involving 15 competing algorithms. AES is designated as FIPS 197. Other algorithms that did not win that competition include such well-known algorithms as Twofish. The importance of AES cannot be overstated. It is widely used around the world and is perhaps the most widely used symmetric cipher. Of all the algorithms in this chapter, AES is the one you should give the most attention to.

As mentioned earlier, AES can have three different key sizes: 128, 192, and 256 bits. The three different implementations of AES are referred to as AES 128, AES 192, and AES 256. The block size can also be 128, 192, or 256 bit. It should be noted that the original Rijndael cipher allowed for variable block and key sizes in 32-bit increments. However, the U.S. government uses these three key sizes with a 128-bit block as the standard for AES.

This algorithm was developed by two Belgian cryptographers, John Daemen and Vincent Rijmen. John Daeman is a Belgian cryptographer who has worked extensively on the cryptanalysis of block ciphers, stream ciphers, and cryptographic hash functions. For those new to security, the brief description given so far is sufficient. However, we will explore the AES algorithm in more detail. Just as with the details of DES, this may be a bit confusing to some readers at first glance and may require a few rereads.

Rijndael uses a substitution-permutation matrix rather than a Feistel network. The Rijndael cipher works by first putting the 128-bit block of plain text into a 4-byte-by-4-byte matrix, termed the *state*, that changes as the algorithm proceeds through its steps. The first step is to convert the plain text block into binary and then put it into a matrix, as shown in Figure 8.3.

11011001	01110010	10110000	11101010
01011111	00011001	11011001	10011001
10011100	11011101	00011001	11111101
11011001	10001001	11011001	10001001

FIGURE 8.3 The Rijndael matrix.

Once you have the original plain text in binary, placed in the 4-byte-by-4-byte matrix, the algorithm consists of a few relatively simple processes that are used during various rounds. The processes are described here:

- **AddRoundKey:** In this process, each byte of the state is exclusively ORed with the round key. Just as with DES, there is a key schedule algorithm that slightly changes the key each round.

- **SubBytes:** This involves substitutions of the input bytes (which are the output from the AddRoundKey phase). This is where the contents of the matrix are put through the s-boxes. Each of the s-boxes is 8 bits.
- **ShiftRows:** This is a transposition step where each row of the state is shifted cyclically a certain number of steps. In this process, the first row is left unchanged. Every byte in the second row is shifted 1 byte to the left (and the far left wraps around). Every byte of the third row is shifted 2 to the left, and every byte of the fourth row is shifted 3 to the left (again with the far left wrapping around). This is shown in Figure 8.4. Notice that in this figure that the bytes are simply labeled by their row and then a letter, such as 1a, 1b, 1c, 1d.

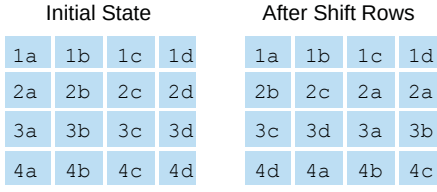

FIGURE 8.4 ShiftRows.

- **MixColumns:** This is a mixing operation that operates on the columns of the state, combining the 4 bytes in each column. In the MixColumns process, each column of the state is multiplied with a fixed polynomial. Each column in the state (remember the matrix we are working with) is treated as a polynomial within the Galois field (2^8). The result is multiplied with a fixed polynomial $c(x) = 3x^3 + x^2 + x + 2$ modulo $x^4 + 1$. The MixColumns process can also be viewed as a multiplication by the shown particular MDS matrix in the finite field GF(2^8).

These processes are executed multiple times in the Rijndael cipher. For 128-bit keys, there are 10 rounds. For 192-bit keys, there are 12 rounds. For 256-bit keys, there are 14 rounds.

These last few steps may be leaving you a bit confused if you don't have a background in number theory. You may be asking questions like "What is a Galois field?" and "What is a fixed polynomial?" A general overview of the math needed to understand this is provided in the next section.

AES Math

A *group* is an algebraic system consisting of a set, an identity element, one operation, and its inverse operation. Basically, groups are ways to limit math operations, such as addition, to specific sets of numbers. There are several specialized types of groups, briefly described here:

- An *abelian group*, or *commutative group*, has an additional axiom $a + b = b + a$ if the operation is addition or $ab = ba$ if the operation is multiplication.
- A *cyclic group* has elements that are all powers of one of its elements.

- A *ring* is an algebraic system consisting of a set, an identity element, two operations, and the inverse operation of the first operation.
- A *field* is an algebraic system consisting of a set, an identity element for each operation, two operations, and their respective inverse operations.

This brings us to the Galois group, or Galois field. GF(p) for any prime, p, this Galois field has p elements that are the residue classes of integers modulo p. That prime number p is the defining element for the field. Now, this is obviously a brief description and does not attempt to get into details. A thorough study of group theory would be needed to get into more detail.

Blowfish

Blowfish is a symmetric block cipher. It uses a variable-length key ranging from 32 to 448 bits. Blowfish was designed in 1993 by Bruce Schneier. It has been analyzed extensively by the cryptography community and has gained wide acceptance. It is also a noncommercial (free of charge) product, thus making it attractive to budget-conscious organizations.

RC4

All the other symmetric algorithms we have discussed have been block ciphers. RC4 is a stream cipher developed by Ron Rivest. RC is an acronym for Ron's Cipher, or sometimes Rivest's Cipher. There are other RC versions, such as RC5 and RC6.

Serpent

Serpent has a block size of 128 bits and can have a key size of 128, 192, or 256 bits, much like AES. The algorithm is also a substitution-permutation network (like AES). It uses 32 rounds working with a block of four 32-bit words. Each round applies one of eight 4-bit to 4-bit s-boxes 32 times in parallel. Serpent was designed so that all operations can be executed in parallel. This is one reason it was not selected as a replacement for DES. At the time it was created, many computers had difficulty with parallel processing. However, modern computers have no problem with parallel processing, so Serpent is once again an attractive choice.

Skipjack

Originally classified, the Skipjack algorithm was developed by the National Security Agency (NSA) for the clipper chip. The clipper chip was a chip with built-in encryption; however, the decryption key would be kept in a key escrow in case law enforcement needed to decrypt data without the computer owner's cooperation. This feature made the process highly controversial. Skipjack uses an 80-bit key to encrypt or decrypt 64-bit data blocks. It is an unbalanced Feistel network with 32 rounds. Unbalanced Feistel simply means a Feistel cipher wherein the two halves of plain text for each block are not the same size. For example, a 64-bit block might be divided into a 48-bit half and a 16-bit half rather than two 32-bit halves.

Modification of Symmetric Methods

Just as important as understanding symmetric ciphers is understanding how they are implemented. There are some common modes that can affect how a symmetric cipher functions.

Electronic Codebook

The most basic encryption mode is the electronic codebook (ECB) mode. With ECB, a message is divided into blocks, and each block is encrypted separately. The problem is that if you submit the same plain text more than once, you always get the same cipher text. This gives attackers a place to begin analyzing the cipher to attempt to derive the key. Put another way, ECB is simply using the cipher exactly as it is described, without attempting to improve its security.

Cipher Block Chaining

When using cipher block chaining (CBC) mode, each block of plain text is XORed with the previous cipher text block before being encrypted. This means there is significantly more randomness in the final cipher text, making it much more secure than electronic codebook mode. It is the most common mode.

There really is no good reason to use ECB over CBC if both ends of communication can support CBC. CBC is a strong deterrent to known plain text attacks, a cryptanalysis method we will examine later in this chapter.

The only issue with CBC is the first block. There is no preceding block of cipher text to XOR the first plain text block with. It is common to add an initialization vector (IV) to the first block so that it has something to be XORed with. The initialization vector is basically a pseudo-random number, much like the cipher key. Usually an IV is only used once, and it is thus called a *nonce* (for "number used only once"). The CBC mode is actually fairly old. It was introduced by IBM in 1976.

Public Key (Asymmetric) Encryption

Public key encryption is essentially the opposite of single-key encryption. With any public key encryption algorithm, one key (called the public key)is used to encrypt a message, and another (called the private key) is used to decrypt the message. You can freely distribute your public key so that anyone can encrypt a message to send to you, but only you have the private key and only you can decrypt the message. The actual mathematics behind the creation and application of the keys will vary between different asymmetric algorithms. We will look at the math for RSA later in this section. It should be pointed out, however, that many public key algorithms are dependent, to some extent, on large prime numbers, factoring, and number theory.

It has become standard in cryptography to use the fictitious Alice and Bob to illustrate asymmetric cryptography. If Alice wants to send Bob a message, she will use Bob's public key to encrypt that message. It does not matter if every other person on the planet also has Bob's public key. That key cannot decrypt the message. Only Bob's private key can do that. This is shown in Figure 8.5.

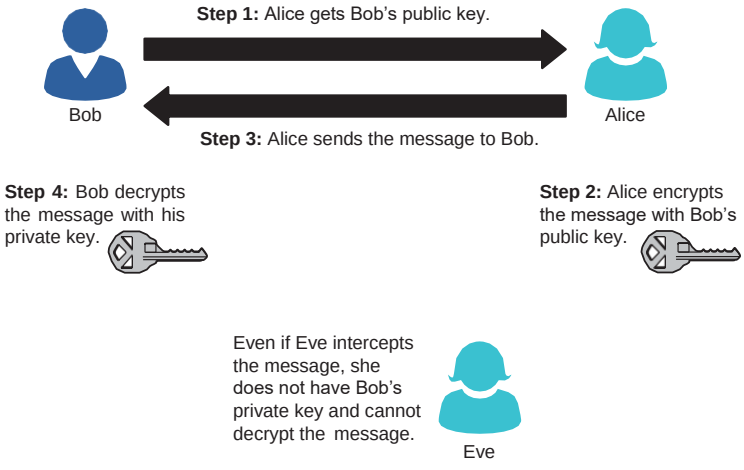

FIGURE 8.5 Public key cryptography.

Public key encryption is important because there are no issues to deal with concerning distribution of the keys. With symmetric key encryption, you must get a copy of the key to every person to whom you wish to send your encrypted messages. If that key were lost or copied, someone else might be able to decrypt all of your messages. With public key encryption, you can freely distribute your public key to the entire world, yet only you can decrypt messages encrypted with that public key.

RSA

It isn't possible to discuss cryptography without at least some discussion of RSA, which is a very widely used encryption algorithm. This public key method was developed in 1977 by three mathematicians: Ron Rivest, Adi Shamir, and Len Adleman. The name RSA is derived from the first letter of each mathematician's last name Let us take a look at the math involved in RSA. (It should be pointed out that knowing the math behind this, or any other algorithm, is not critical for most security professionals. But some readers will have an interest in going deeper into cryptography, and this will be a good place to start.)

There are a few basic math concepts you need to know about in order to understand RSA. Some (or even all) of this material may be a review:

- **Prime numbers:** A prime number is divisible by itself and by 1. So 2, 3, 5, 7, 11, 13, 17, and 23 are all prime numbers. (Note that 1 itself is considered a special case and is not prime.)
- **Co-prime:** This actually does not mean prime; it means two numbers have no common factors. So, for example, the factors of 8 (excluding the special case of 1) are 2 and 4, and the factor of 9 are 3. The numbers 8 and 9 have no common factors, so they are co-prime.

- **Euler's Totient:** Pronounced "oilers" totient, or called just *the totient*, this is the number of integers smaller than *n* that are co-prime with *n*. So let us consider the number 10. Since 2 is a factor of 10, it is not co-prime with 10. But 3 is co-prime with 10. The number 4 is not co-prime since both 4 and 10 have 2 as a factor. The number 5 is not since it is a factor of 10. Neither is 6 since both 6 and 10 have 2 as a cofactor. The number 7 is prime, so it is co-prime with 10. The number 8 is not because both 8 and 10 have 2 as a factor. The number 9 is co-prime with 10. So the numbers 3, 7, and 9 are co-prime with 10. We add in 1 as a special case, the Euler's totient of 10 is 4. Now it just so happens that Leonhard Euler also proved that if the number *n* is a prime number, then its totient is always $n - 1$. So the totient of 7 is 6. The totient of 13 is 12.
- **Multiplying and co-prime:** Now we can easily compute the totient of any number. And we know automatically that the totient of any prime number *n* is just $n - 1$. But what if we multiply two primes? For example, we can multiply 5 and 7 to get 35. Well, we can go through all the numbers up to 35 and tally up the numbers that are co-prime with 35. But the larger the numbers get, the more tedious this process becomes. For example, if you have a 20-digit number, manually calculating the totient is almost impossible. Fortunately, Leonhard Euler also proved that if you have a number that is the product of two primes (let's call them *p* and *q*), such as 5 and 7, then the totient of the product of those two numbers (in this case 35) is equal to $p - 1 \times q - 1$ (in this case 4×6, or 24).
- **Modulus:** This is the last concept you need for RSA. There are a few approaches to explaining this concept. We will actually use two of them. First, from a programmer's perspective, the modulus operation is to divide two numbers but only give the remainder. Programmers often use the symbol % to denote modulo operations. So 10 % 3 is 1. The remainder of 10 divided by 3 is 1. Now, this is not really a mathematical explanation of modulo operations.

Basically, modulo operations take addition and subtraction and limit them by some value. You have actually done this all your life without realizing it. Consider a clock. When you say *2 p.m.*, what you really mean is 14 mod 12 (or 14 divided by 12; just give me the remainder). Or if it is 2 p.m. now (14 actually) and you tell me you will call me in 36 hours, what I do is 14 + 36 mod 12, or 50 mod 12, which is 2 a.m. (a bit early for a phone call, but it illustrates our point).

Now if you understand these basic operations, then you are ready to learn RSA. If needed, reread the preceding section (perhaps even more than once) before proceeding.

To create the key, you start by generating two large random primes, *p* and *q*, of approximately equal size. You need to pick two numbers so that when multiplied together, the product will be the size you want (2048 bits, 4096 bits, and so on).

Now multiply *p* and *q* to get *n*. Let $n = pq$.

The next step is to multiply Euler's totient for each of these primes. Basically, Euler's totient is the total number of co-prime numbers. Two numbers are considered co-prime if they have no common factors. For example, if the original number is 7, then 5 and 7 would be co-prime. Remember that it just so

happens that for prime numbers, this is always the number minus 1. For example, 7 has 6 numbers that are co-prime to it. (If you think about this a bit, you will see that 1, 2, 3, 4, 5, and 6 are all co-prime with 7.)

Let $m = (p - 1)(q - 1)$.

Now we are going to select another number. We will call this number *e*. We want to pick *e* so that it is co-prime to *m*. Choose a small number *e*, co-prime to *m*.

We are almost done generating a key. Now we just find a number *d* that, when multiplied by *e* and modulo *m*, would yield 1. (Remember: *Modulo* means dividing two numbers and returning the remainder. For example, 8 modulo 3 would be 2.)

Find *d*, such that $de \% m = 1$.

Now you will publish *e* and *n* as the public key. Keep *d* as the secret key. To encrypt, you simply take your message raised to the *e* power and modulo *n*:

$$= m^e \% n$$

To decrypt, you take the cipher text and raise it to the *d* power modulo *n*:

$$P = C^d \% n$$

The letter *e* is for *encrypt* and *d* is for *decrypt*. If all this seems a bit complex to you, you must realize that many people work in network security without being familiar with the actual algorithm for RSA (or any other cryptography for that matter). However, if you wish to go deeper into cryptography, then this is a very good start. It involves some fundamental number theory, particularly regarding prime numbers. There are other asymmetric algorithms that work in a different manner. For example, elliptic curve cryptography is one such example.

Let's look at an example that might help you understand. Of course, RSA would be done with very large integers. To make the math easy to follow, we will use small integers in this example.

Choose two distinct prime numbers, such as $p = 61$ and $q = 53$. Compute $n = pq$:

$$n = 61 \times 53 = 3233$$

Compute the totient of the product as $\varphi(n) = (p - 1)(q - 1)$:

$$\varphi(3233) = (61 - 1)(53 - 1) = 3120$$

Choose any number $1 < e < 3120$ that is co-prime to 3120. Choosing a prime number for *e* leaves us only to check that *e* is not a divisor of 3120. Let $e = 17$. Compute *d*, the modular multiplicative inverse of yielding $d = 2753$.

The public key is (n = 3233, e = 17). For a padded plain text message m, the encryption function is:

$$m^{17} \pmod{3233}$$

The private key is:

(n = 3233, d = 2753)

For an encrypted cipher text, c, the decryption function is:

$$c^{2753} \pmod{3233}$$

For those readers new to RSA or new to cryptography in general, it might be helpful to see one more example, with even smaller numbers:

Select primes: p = 17 & q = 11.

Compute $n = pq$ =17 × 11 = 187.

Compute $\varphi(n) = (p - 1)(q - 1)$ = 16 × 10 = 160.

Select *e:* gcd(e,160) = 1; choose e = 7.

Determine *d:* de = 1 mod 160 and d < 160. Value is d = 23 since 23 × 7 = 161 = 10 × 160 + 1.

Publish public key: (7 and 187).

Keep secret private key: 23.

Diffie-Hellman

Diffie-Hellman was the first publicly described asymmetric algorithm. This cryptographic protocol allows two parties to establish a shared key over an usecured channel. In other words, Diffie-Hellman is often used to allow parties to exchange a symmetric key through some unsecured medium, such as the Internet. It was developed by Whitfield Diffie and Martin Hellman in 1976.

One problem with working in cryptology is that much of the work is classified. You could labor away and create something wonderful…that you cannot tell anyone about. Then to make matters worse, years later someone else might develop something similar, release it, and get all the credit. This is exactly the situation with Diffie-Hellman. It turns out that a similar method had been developed a few years earlier by Malcolm J. Williamson of the British Intelligence Service, but it was classified.

There have been several modifications and improvements to Diffie-Hellman. Among the most well known are Elgamal and MQV. Elgamal is named after its inventor, Taher Elgamal. MQV was first proposed in 1995 and is named after its inventors Alfred Menezes, Minghua Qu, and Scott Vanstone. MQV is incorporated in the public key standard IEEE P1363 and NIST's SP800-56A standard.

Elliptic Curve Cryptography

The elliptic curve algorithm was first described in 1985 by Victor Miller and Neal Koblitz. Elliptic curve cryptography (ECC) is based on the fact that finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point is difficult to the point of being impractical. The mathematics behind this algorithm are a bit much for an introductory book on security. However, if you are interested, you should read the great tutorial at http://arstechnica.com/security/2013/10/ a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/.

There are a number of variations, such as ECC-DH (ECC Diffie-Hellman) and ECC-DSA (ECC Digital Signature Algorithm). The real strength of ECC crypto systems is that you can get just as much security with a smaller key as with other systems, like RSA. For example, a 384-bit ECC key is as strong as 2048-bit RSA.

PGP

PGP, a public key system, stands for *Pretty Good Privacy*. It is a widely used system that is considered very secure by most experts. There are several software implementations available as freeware for most desktop operating systems. There are PGP plug-ins for MSN Messenger and many other popular communications software packages. A simple Yahoo! or Google search for *PGP* will help you find many of these software products.

FYI: "Old" Encryption

PGP is quite old, and you might wonder whether it is therefore outdated. Cryptography is unlike other technological endeavors in this regard: Older is better. It is usually unwise to use the "latest thing" in encryption for the simple reason that it is unproven. An older encryption method, provided it has not yet been broken, is usually a strong choice because it has been subjected to years of examination by experts and to cracking attempts by both experts and less honorably motivated individuals. This is sometimes hard for computer professionals to understand since the newest technology is often preferred in the computer business.

PGP was invented by Phil Zimmermann. Before creating PGP, Zimmermann had been a software engineer for 20 years and had experience with existing forms of cryptography. A great deal of controversy surrounded the birth of PGP because it was created without an easy means for government intrusion, and its encryption was considered too strong for export. This led to Zimmermann being the target of a 3-year government investigation. However, those legal matters have since been resolved, and PGP is one of the most widely used encryption methods available.

The important things to know about PGP are that it is:

- A public key encryption
- Considered quite secure
- Available free of charge

These facts make it well worth your time to investigate PGP as a possible solution for your organization's encryption needs.

Legitimate Versus Fraudulent Encryption Methods

The encryption methods discussed earlier in this chapter are just a few of the most widely used modern encryption methods. Dozens of other methods are released to the public for free or are patented and sold for profit every year. However, it is important to realize that this particular area of the computer industry is replete with frauds and charlatans. One need only scan a search engine for *encryption* to find a plethora of advertisements for the latest and greatest "unbreakable" encryption. If you are not knowledgeable about encryption, how do you separate legitimate encryption methods from frauds?

There are many fraudulent cryptographic claims out there. You do not have to be a cryptography expert to be able to avoid many of those fraudulent claims. Here are some warning signs:

- **Unbreakable:** Anyone with experience in cryptography knows that there is no such thing as an unbreakable code. There are codes that have not yet been broken. There are codes that are very hard to break. But when someone claims that a method is "completely unbreakable," you should be suspicious.
- **Certified:** Guess what? There is no recognized certification process for encryption methods. Therefore, any "certification" a company claims is totally worthless.
- **Inexperienced people:** A company is marketing a new encryption method. What experience do the people working with it have? Does the cryptographer have a background in math, encryption, or algorithms? If not, has he submitted his method to experts in peer-reviewed journals? Or, is he at least willing to disclose how his method works so that it can be fairly judged? Recall that PGP's inventor had decades of software engineering and encryption experience.

Auguste Kerckhoffs first articulated what has come to be called Kerckhoffs's principle in the 1800s. He stated that the security of a cipher depends only on the secrecy of the key, not on the secrecy of the algorithm. Claude Shannon rephrased, this stating that, "One ought to design systems under the assumption that the enemy will ultimately gain full familiarity with them." This idea, referred to as Shannon's maxim, states essentially the same idea as Kerckhoffs's principle.

I would add to Kerckhoffs's principle/Shannon's maxim something I will humbly call Easttom's corollary: "You should be very wary of any cryptographic algorithm that has not been published and thoroughly reviewed. Only after extensive peer review should you consider the use of any cryptographic algorithm." I first proposed this corollary in my book *Modern Cryptography: Applied Mathematics for Encryption and Information Security*.

Digital Signatures

A digital signature is not used to ensure the confidentiality of a message but rather to guarantee who sent the message. This is referred to as *nonrepudiation*. Essentially, nonrepudiation means proving who the sender is. Digital signatures are actually rather simple, but they are clever. They simply reverse the asymmetric encryption process. Recall that in asymmetric encryption, the public key (which anyone can have access to) is used to encrypt a message to the recipient, and the private key (which is kept secure and private) can decrypt it. With a digital signature, the sender encrypts something with his private key. If the recipient is able to decrypt that with the sender's public key, then it must have been sent by the person purported to have sent the message. This process is shown in Figure 8.6.

FIGURE 8.6 Digital signatures.

Hashing

A *hashing* is a type of cryptographic algorithm that has some specific characteristics. First and foremost, it is one way. That means you cannot unhash something. Second, you get a fixed-length output no matter what input is given. Third, there are no collisions. A collision occurs when two different inputs to the same hashing algorithm produce the same output (called a *hash* or *digest*). Ideally we would like to have no collisions. But the reality is that with fixed-length output, collisions are possible. So the goal is to make collisions so unlikely as to be something we need not think about.

Hashes are exactly how Windows stores passwords. For example, if your password is *password*, then Windows will first hash it and produce something like this:

0BD181063899C9239016320B50D3E896693A96DF

Windows will then store that in the SAM (Security Accounts Manager) file in the Windows System directory. When you log on, Windows cannot unhash your password (because, remember, it is one way). So, what Windows does is take whatever password you type in, hash it, and then compare the result with what is in the SAM file. If they match (exactly), then you can log in.

Storing Windows passwords is just one application of hashing. There are others. For example, in computer forensics it is common to hash a drive before you begin forensic examination. Then later you can hash it again to see if anything was changed (accidently or intentionally). If the second hash matches the first, then nothing has been changed.

There are various hashing algorithms. The two most common of them are MD5 and SHA. (It was SHA-1, but since then later versions, like SHA-256, have become more common.)

MD5

MD5 is a 128-bit hash that is specified in RFC 1321. It was designed by Ron Rivest in 1991 to replace an earlier hash function, MD4. MD5 produces a 128-bit hash or digest. It has been found to be not as collision resistant as SHA.

SHA

Secure Hash Algorithm (*SHA*) is perhaps the most widely used hash algorithm today. There are now several versions of SHA. All versions of SHA are considered to be secure and collision free:

- **SHA-1:** This is a 160-bit hash function that resembles the earlier MD5 algorithm. It was designed by the NSA to be part of the Digital Signature Algorithm (DSA).
- **SHA-2:** This is actually two similar hash functions, with different block sizes, known as SHA-256 and SHA-512. They differ in the word size; SHA-256 uses 32-byte (256 bits) words, whereas SHA-512 uses 64-byte (512 bits) words. There are also truncated versions of each standardized, known as SHA-224 and SHA-384. These were also designed by the NSA.
- **SHA-3:** This latest version of SHA was adopted in October 2012.

RIPEMD

RACE Integrity Primitives Evaluation Message Digest (*RIPEMD*) is a 160-bit hash algorithm developed by Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. There exist 128-, 256-, and 320-bit versions of this algorithm, called RIPEMD-128, RIPEMD-256, and RIPEMD-320, respectively. All these replace the original RIPEMD, which was found to have collision issues.

MAC and HMAC

Hashes are used for several security-related functions. One of these functions is to store passwords, and we have discussed that already (and we will see more later in this chapter).

A hash of a message can be sent to see if accidental alteration occurred in transit. If a message is altered in transit, the recipient can compare the hash received against the hash the computer sent and detect the error in transmission. But what about intentional alteration of messages? What happens if someone alters the message intentionally, deletes the original hash, and recomputes a new one? Unfortunately, a simple hashing algorithm cannot account for this scenario.

Using a *message authentication code* (*MAC*) is one way to detect intentional alterations in a message. A MAC is also often called a *keyed cryptographic hash function*. That name should tell you how this works. One way to do it is the hashing message authentication code (HMAC). Let us assume you are using MD5 to verify message integrity. To detect an intercepting party intentionally altering a message, both the sender and the recipient must have previously exchanged a key of the appropriate size (in this case, 128 bits). The sender will hash the message and then XOR that hash with this key. The recipient will hash what she receives and XOR that computed hash with the key. Then the two hashes are exchanged. If an intercepting party were to simply recompute the hash, he would not have the key to XOR that with (and may not even be aware that it should be XORed); thus, the hash the interceptor creates won't match the hash the recipient computes, and the interference will be detected.

There are other variations of the concept. Some use a symmetric cipher in CBC (cipher block chaining mode) and then use only the final block as the MAC. These variations are called CBC-MAC.

Rainbow Tables

Since Windows and many other systems store passwords as hashes, many people have had an interest in how to break hashes. As we've mentioned, since a hash is not reversible, there is no way to unhash something. In 1980, Martin Hellman described a cryptanalytic technique that reduces the time of cryptanalysis by using precalculated data stored in memory. This technique was improved by Rivest before 1982. Basically, these types of password crackers are working with precalculated hashes of all passwords available within a certain character space, be that a–z or a–zA–z or a–zA–Z0–9, and more. This is called a *rainbow table*. If you search a rainbow table for a given hash, whatever plain text you find must be the text that was input into the hashing algorithm to produce that specific hash.

Clearly, such a rainbow table would get very large very fast. Assume that the passwords must be limited to keyboard characters. That leaves 52 letters (26 uppercase and 26 lowercase), 10 digits, and roughly 10 symbols, or about 72 characters. As you can imagine, even a 6-character password has a very large number of possible combinations. This means there is a limit to how large a rainbow table can be, and this is why longer passwords are more secure than shorter passwords.

Since the development of rainbow tables, there have been methods designed to thwart such attacks. The most common is salting, in which random bits are added to further secure encryption or hashing. This is most often encountered with hashing to prevent rainbow table attacks.

Essentially, the salt is intermixed with the message that is to be hashed. Consider an example. Say that you have a password that is:

pass001

In binary that is:

01110000 01100001 01110011 01110011 00110000 00110000 00110001

A salt algorithm would insert bits periodically. Let's assume for our example that we insert bits every fourth bit, giving us:

0111100001 0110100011 0111100111 0111100111 0011100001 0011100001 0011100011

If you convert that to text, you would get:

xZ7��#

All this is transparent to the end user, who doesn't even know that salting is happening or what it is. However, an attacker using a rainbow table to get passwords would get the wrong password.

Steganography

Steganography is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message. It is a form of security through obscurity. Often the message is hidden in some other file, such as a digital picture or an audio file, to defy detection.

The advantage of steganography over cryptography alone is that messages do not attract attention to themselves. If someone is aware that a message is even there, she won't try to decipher it. In many cases, messages are encrypted and hidden via steganography.

The most common implementation of steganography utilizes the least significant bits in a file in order to store data. By altering the least significant bit, you can hide additional data without altering the original file in any noticeable way.

There are some basic steganography terms you should know:

- *Payload* is the data to be covertly communicated. In other words, it is the message you wish to hide.
- The *carrier* is the signal, stream, or data file into which the payload is hidden.
- The *channel* is the medium used. This may be still photos, video, or sound files.

The most common way steganography is accomplished today is via least significant bits. Every file has a certain number of bits per unit of the file. For example, an image file in Windows is 24 bits per pixel. If you change the least significant of those bits, then the change is not noticeable with the naked eye. And you can hide information in the least significant bits of an image file. With least significant bit (lsb) replacement, certain bits in the carrier file are replaced.

Historical Steganography

In modern times, steganography involves digital manipulation of files to hide messages. However, the concept of hiding messages is not new. There have been many methods used throughout history.

- The ancient Chinese wrapped notes in wax and swallowed them for transport. This was a crude but effective method of hiding messages.
- In ancient Greece, a messenger's head would be shaved, a message was written on his head, and then his hair was allowed to grow back. Obviously, this method required some time.
- In 1518 Johannes Trithemius wrote a book on cryptography and described a technique in which a message was hidden by having each letter taken as a word from a specific column.
- During World War II the French Resistance sent messages written on the backs of couriers using invisible ink.
- Microdots are images/undeveloped film the size of a typewriter period that are embedded in innocuous documents. These were said to be used by spies during the Cold War.
- Also during the Cold War, the U.S. Central Intelligence Agency used various devices to hide messages. For example, they developed a tobacco pipe that had a small space to hide microfilm but could still be smoked.

In more recent times, but before the advent of computers, other methods were used to hide messages.

Steganography Methods and Tools

There are a number of tools available for implementing steganography. Many are free or at least have free trial versions. A few of these tools are listed here:

- **QuickStego:** Easy to use but very limited
- **Invisible Secrets:** Much more robust, with both a free version and a commercial version available
- **MP3Stego:** Specifically for hiding payload in MP3 files
- **Stealth Files 4:** Works with sound files, video files, and image files
- **Snow:** Hides data in whitespace
- **StegVideo:** Hides data in a video sequence
- **Invisible Secrets:** A very versatile steganography tool that has several options

Cryptanalysis

Cryptanalysis is a daunting task. It essentially involves searching for some means to break through encryption of some sort. And, unlike what you see in the movies, it is a very time-consuming task that frequently leads to only partial success. Cryptanalysis involves using any method to decrypt the message that is more efficient than simple brute-force attempts. (Remember that brute forcing means simply trying every possible key.)

A cryptanalysis success is not necessarily breaking the target cipher. In fact, finding any information about the target cipher or key is considered a success. There are several types of cryptographic success:

- **Total break:** The attacker deduces the secret key.
- **Global deduction:** The attacker discovers a functionally equivalent algorithm for encryption and decryption but without learning the key.
- **Instance (local) deduction:** The attacker discovers additional plain texts (or cipher texts) not previously known.
- **Information deduction:** The attacker gains some Shannon information about plain texts (or cipher texts) not previously known.
- **Distinguishing algorithm:** The attacker can distinguish the cipher from a random permutation.

Entire books have been written on cryptanalysis. The purpose of this section is just to give you some basic concepts from the field so that you have a basic understanding. There are certainly other methods not discussed in this section.

Frequency Analysis

Frequency analysis is a basic tool for breaking most classical ciphers, though it is not useful against modern symmetric or asymmetric cryptography. It is based on the fact that some letters and letter combinations are more common than others. In all languages, certain letters of the alphabet appear more frequently than others. By examining those frequencies, you can derive some information about the key that was used. In English, the words *the* and *and* are the two most common three-letter words. The most common single-letter words are *I* and *a*. If you see two of the same letters together in a word, it is most likely *ee* or *oo*.

Modern Cryptanalysis Methods

Cracking modern cryptographic methods is quite daunting. The level of success depends on a combination of resources, including computational power, time, and data. If you had an infinite amount of any of these, you could crack any modern cipher. But you won't have an infinite amount.

The following sections describe the basic approaches used to attack block ciphers. There are other methods that are beyond the scope of this book, such as differential cryptanalysis and linear cryptanalysis. For the purposes of understanding basic computer security, it is not necessary that you master these techniques.

Known Plain Text Attack

The known plain text attack method is based on having a sample of known plain texts and their resulting cipher texts and then using this information to try to ascertain something about the key used. It is easier to obtain known plain text samples than you might think. Consider email. Many people, myself included, use a standard signature block. If you have ever received an email from me, you know what my signature block is. Then if you intercept encrypted emails I send, you can compare the known signature block to the end of the encrypted email. You would then have a known plain text and the matching cipher text to work with. Success with this method requires many thousands of known plain text samples.

Chosen Plain Text Attack

The chosen plain text attack is closely related to the known plain text attack, but with the chosen plain text attack, the attacker has found a method to get the target to encrypt messages the attacker chooses. This can allow the attacker to attempt to derive the key used and thus decrypt other messages encrypted with that key. The method can be difficult but is not impossible. It requires many thousands of chosen plain text samples to be successful.

Cipher Text Only

With a cipher text only attack, the attacker only has access to a collection of cipher texts. This is much more likely than known plain text, but it is also the most difficult type of attack. A cipher text only attack is completely successful if the corresponding plain texts can be deduced or, even better, if the key can be deduced. Even the ability to obtain any information at all about the underlying plain text is considered a success.

Related-Key Attack

A related-key attack is like a chosen plain text attack except that the attacker can obtain cipher texts encrypted under two different keys. This is actually a very useful attack if you can obtain the plain text and matching cipher text.

Cryptography Used on the Internet

What sort of encryption is used on bank websites and for e-commerce? In general, symmetric algorithms are faster and require a shorter key length to be as secure as asymmetric algorithms. However, there is the problem of how to securely exchange keys. So most e-commerce solutions use an asymmetric algorithm to exchange symmetric keys and then use the symmetric keys to encrypt the actual data.

When visiting websites that have an HTTPS at the beginning rather than HTTP, the *S* denotes *Secure*. It means traffic between your browser and the web server is encrypted—usually with either SSL (Secure Sockets Layer) or TLS (Transport Layer Security). SSL, the older of the two technologies, was developed by Netscape. Both SSL and TLS are asymmetric systems.

Quantum Computing Cryptography

One of the most engaging cryptography-related topics in recent times is quantum computing. Here we explain a bit about quantum computing and then look at its impact on cryptography.

The essential issue with quantum computing is the ability to represent more than two states. Current computing technology, using classical bits, can represent only binary values. Qubits, or quantum bits, store data via the polarization of a single photon. The two basic states are horizontal polarization and vertical polarization; however, quantum mechanics allows for a superposition of the two states at the same time—which is simply not possible in a classical bit. The two states of a qubit are represented with quantum notation as $|0\rangle$ and $|1\rangle$, representing horizontal polarization and vertical polarization. A qubit involves the superposition of these two basis states. This superposition is represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Essentially a classical bit can represent a 1 or a 0. A qubit can represent a 1, a 0, or any quantum superposition of those two qubit states, which allows for much more powerful computing.

Contrary to what you might have heard, current quantum computing is not at a stage to be useful in practical applications. Cutting-edge quantum systems of today have only 20 to 50 qubits and can only maintain data for a very short time. This makes for great research but not practical computing applications. However, many experts believe that within 10 years we will have a working, practical quantum computer. What does that mean for cryptography? Well, in 1995 Peter Shor published Shor's algorithm, which proved that a quantum computer would be able to factor large numbers in a much shorter time than classical computers. A quantum computer can factor an integer *N* in polynomial time; actual time is log *N*. This is substantially faster than the most efficient known classical factoring algorithm (the general number field sieve) which works in subexponential time. RSA is based on the difficulty of factoring large numbers. A quantum computer is expected to be similarly efficient at solving discrete logarithm problems. Diffie-Hellman, MQV, Elgamal, and ECC are all based on the difficulty of solving the discrete logarithm problem.

What all this means is that when quantum computers become a practical reality, current asymmetric (public key) algorithms will become obsolete. At that point, current security for e-commerce, VPNs, and many other applications will no longer be secure. The U.S. National Institute of Standards and Technology (NIST) is already working on finding a quantum proof cryptography standard.[1] That study is due to be finished in about 2022.

1. https://www.nist.gov/news-events/news/2019/01/nist-reveals-26-algorithms-advancing-post-quantum-crypto-semifinals

Summary

A basic element of computer security is encryption. Sending sensitive data that is not encrypted is simply foolish. This chapter provided the basic information on how cryptography works. The most important thing to remember is that, ultimately, it is not your computer or your network that can be compromised but rather your data. Encrypting data when transmitting it is an integral part of any security plan.

In the exercises at the end of this chapter, you will practice using different cipher methods and learn more about a number of encryption methods.

Test Your Skills

MULTIPLE CHOICE QUESTIONS

1. It is important to understand the concepts and application of cryptography. Which of the following most accurately defines encryption?
   A. Changing a message so it can only be easily read by the intended recipient
   B. Using complex mathematics to conceal a message
   C. Changing a message using complex mathematics
   D. Applying keys to a message to conceal it
2. Which of the following is the oldest encryption method discussed in this text?
   A. PGP
   B. Multi-alphabet encryption
   C. Caesar cipher
   D. Cryptic cipher
3. Many classic ciphers are easy to understand but not secure. What is the main problem with simple substitution?
   A. It does not use complex mathematics.
   B. It is easily broken with modern computers.
   C. It is too simple.
   D. It maintains letter and word frequency.

4. Classic ciphers were improved with the addition of multiple shifts (multiple substitution alphabets). Which of the following is an encryption method that uses two or more different shifts?

    A. Caesar cipher
    B. Multi-alphabet encryption
    C. DES
    D. PGP

5. Which binary mathematical operation can be used for a simple (but unsecured) encryption method and is in fact a part of modern symmetric ciphers?

    A. Bit shift
    B. OR
    C. XOR
    D. Bit swap

6. Why is binary mathematical encryption not secure?

    A. It does not change letter or word frequency.
    B. It leaves the message intact.
    C. It is too simple.
    D. The mathematics of it is flawed.

7. Which of the following is most true regarding binary operations and encryption?

    A. They are completely useless.
    B. They can form a part of viable encryption methods.
    C. They are only useful as a teaching method.
    D. They can provide secure encryption.

8. What is PGP?

    A. Pretty Good Privacy, a public key encryption method
    B. Pretty Good Protection, a public key encryption method
    C. Pretty Good Privacy, a symmetric key encryption method
    D. Pretty Good Protection, a symmetric key encryption method

9. Which of the following methods is available as an add-in for most email clients?
   A. DES
   B. RSA
   C. Caesar cipher
   D. PGP
10. Which of the following is a symmetric key system that uses 64-bit blocks?
   A. RSA
   B. DES
   C. PGP
   D. Blowfish
11. What is the advantage of a symmetric key system using 64-bit blocks?
   A. It is fast.
   B. It is unbreakable.
   C. It uses asymmetric keys.
   D. It is complex.
12. What size key does a DES system use?
   A. 64 bit
   B. 128 bit
   C. 56 bit
   D. 256 bit
13. What type of encryption uses different keys to encrypt and decrypt the message?
   A. Private key
   B. Public key
   C. Symmetric
   D. Secure
14. Which of the following methods uses a variable-length symmetric key?
   A. Blowfish
   B. Caesar
   C. DES
   D. RSA

15. What should you be most careful of when looking for an encryption method to use?
    A. Complexity of the algorithm
    B. Veracity of the vendor's claims
    C. Speed of the algorithm
    D. How long the algorithm has been around
16. Which of the following is most likely to be true of an encryption method that is advertised as unbreakable?
    A. It is probably suitable for military use.
    B. It may be too expensive for your organization.
    C. It is likely to be exaggerated.
    D. It is probably one you want to use.
17. Which of the following is most true regarding certified encryption methods?
    A. These are the only methods you should use.
    B. It depends on the level of certification.
    C. It depends on the source of the certification.
    D. There is no such thing as certified encryption.
18. Which of the following is most true regarding new encryption methods?
    A. Never use them until they have been proven.
    B. You can use them, but you must be cautious.
    C. Use them only if they are certified.
    D. Use them only if they are rated unbreakable.

EXERCISES

EXERCISE 8.1: Using the Caesar Cipher

This exercise is well suited for group or classroom exercises.

1. Write a sentence in normal text.
2. Use a Caesar cipher of your own design to encrypt it.
3. Pass it to another person in your group or class.
4. Time how long it takes that person to break the encryption.
5. (Optional) Compute the mean time for the class to break Caesar ciphers.

EXERCISE 8.2: Using Multi-Alphabet Ciphers

This exercise also works well for group settings and is best used in conjunction with Exercise 8.1.

1. Write a sentence in normal text.
2. Use a multi-alphabet cipher of your own design to encrypt it.
3. Pass it to another person in your group or class.
4. Time how long it takes that person to break the encryption.
5. (Optional) Compute the mean time for the class to break these and compare that to the mean time required to break the Caesar ciphers.

EXERCISE 8.3: Using PGP

1. Download a PGP attachment for your favorite email client. A web search for PGP and your email client (that is, PGP and Outlook or PGP and Eudora) should locate both modules and instructions.
2. Install and configure the PGP module.
3. Working with a classmate, send encrypted messages back and forth.

EXERCISE 8.4: Finding Good Encryption Solutions

1. Scan the Web for various commercial encryption algorithms.
2. Find one that you feel may be "snake oil."
3. Write a brief paper explaining your opinion.

PROJECTS

PROJECT 8.1: RSA Encryption

Using the Web or other resources, write a brief paper about RSA, its history, its methodology, and where it is used. Students with a sufficient math background may choose to delve more deeply into the RSA algorithm's mathematical basis.

PROJECT 8.2: Programming Caesar Cipher

This project is for students with some programming background.

Write a simple program in any language you prefer (or that your instructor specifies) that can perform a Caesar cipher. In this chapter, you not only saw how this cipher works but were given some ideas on how to use ASCII codes to make this work in any standard programming language.

PROJECT 8.3: Other Encryption Methods

Write a brief essay describing any encryption method not already mentioned in this chapter. In the paper, describe the history and origin of that algorithm. You should also provide some comparisons with other well-known algorithms.

Case Study

Jane Doe is responsible for selecting an encryption method that is suitable for her company, which sells insurance. The data the company sends is sensitive but is not military or classified in nature. Jane is looking at a variety of methods. She ultimately selects a commercial implementation of RSA. Is this the best choice? Why or why not?