

17 September 2014  
12:35

## Designing a Cryptographic Solution

---

The mention of cryptography may conjure up images of intrigue and cloak-and-dagger spy movies, but in the real world, cryptography is at the heart of many security implementations. Cryptographic solutions provide confidentiality and integrity of data in circumstances where data might be exposed to threats from untrusted individuals. To create a successful security policy, you must understand the basic functionality of cryptography and how you can use encryption and hashing to provide confidentiality and integrity for your data. Along with this, you must understand the importance of key management to keep your solution secure.

### “Do I Know This Already?” Quiz

The “Do I Know This Already?” quiz helps you determine your level of knowledge of this chapter’s topics before you begin. Table 12-1 details the major topics discussed in this chapter and their corresponding quiz questions.

**Table 12-1** “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section	Questions
Introducing Cryptographic Services	1 to 5
Exploring Symmetric Encryption	6 to 10
Understanding Security Algorithms	11 to 15

1. What form of attack are all algorithms susceptible to?
  - a. Meet-in-the-middle
  - b. Spoofing
  - c. Stream cipher
  - d. Brute-force

2. Which type of cipher achieves security by rearranging the letters in a string of text?
  - a. Vigenère cipher
  - b. Stream cipher
  - c. Transposition cipher
  - d. Block cipher
3. In terms of constructing a good encryption algorithm, what does it mean to create an avalanche effect?
  - a. Changing only a few bits of a plain-text message causes the ciphertext to be completely different.
  - b. Altering the key length causes the ciphertext to be completely different.
  - c. Changing only a few bits of a ciphertext message causes the plain text to be completely different.
  - d. Altering the key length causes the plain text to be completely different.
4. Which of the following are techniques used by symmetric encryption cryptography? (Choose all that apply.)
  - a. Block ciphers
  - b. Message Authentication Codes (MAC)
  - c. One-time pad
  - d. Stream ciphers
  - e. Vigenère ciphers
5. Which of the following is not a common stream cipher?
  - a. RC4
  - b. RSA
  - c. SEAL
  - d. DES
6. Which of the following characteristics accurately describe symmetric encryption algorithms? (Choose all that apply.)
  - a. They are faster than asymmetric algorithms.
  - b. They have longer key lengths than asymmetric encryption algorithms.
  - c. They are stronger than asymmetric algorithms.
  - d. They are less complex mathematically than asymmetric algorithms.
  - e. They are slower than asymmetric algorithms.
  - f. They are weaker than asymmetric algorithms.

7. DES typically operates in block mode, where it encrypts data in what size blocks?
  - a. 56-bit blocks
  - b. 40-bit blocks
  - c. 128-bit blocks
  - d. 64-bit blocks
8. Stream ciphers operate on which of the following?
  - a. Fixed-length groups of bits called blocks
  - b. Individual digits, one at a time, with the transformations varying during the encryption
  - c. Individual blocks, one at a time, with the transformations varying during the encryption
  - d. Fixed-length groups of digits called blocks
9. Which statement accurately describes ECB mode?
  - a. In ECB mode, each 64-bit plain-text block is exclusive ORed (XORed) bitwise with the previous ciphertext block.
  - b. ECB mode uses the same 64-bit key to serially encrypt each 56-bit plain-text block.
  - c. ECB mode uses the same 56-bit key to serially encrypt each 64-bit plain-text block.
  - d. In ECB mode, each 56-bit plain-text block is exclusive ORed (XORed) bitwise with the previous ciphertext block.
10. What method does 3DES use to encrypt plain text?
  - a. 3DES-EDE
  - b. EDE-3DES
  - c. 3DES-AES
  - d. AES-3DES
11. Which of the following is not considered a trustworthy symmetric encryption algorithm?
  - a. 3DES
  - b. IDEA
  - c. EDE
  - d. AES



12. In a brute-force attack, generally an attacker has to search through what percentage of the keyspace until he or she finds the key that decrypts the data?
  - a. Roughly 10 percent
  - b. Roughly 75 percent
  - c. Roughly 66 percent
  - d. Roughly 50 percent
13. How many weak keys are a part of the overall DES keyspace?
  - a. Five
  - b. One
  - c. Four
  - d. None
14. Which of the following is not a component of the key management life cycle?
  - a. Key verification
  - b. Key transposition
  - c. Key generation
  - d. Key exchange
  - e. Key storage
15. Hashing is used to provide which of the following?
  - a. Data consistency
  - b. Data binding
  - c. Data checksums
  - d. Data integrity

---

## Foundation Topics

---

### Introducing Cryptographic Services

To understand cryptographic services, first you must understand the science of cryptology, which in essence is the making and breaking of secret codes. Cryptology can be broken into two distinct areas: cryptography and cryptanalysis. Cryptography is the development and use of codes. Cryptanalysis is all about the breaking of these codes. This section explores these two disciplines to give you a better understanding of cryptographic services as a whole.

### Understanding Cryptology

Because cryptography is made up of two halves—the creation of codes and the attempted breaking of those codes—a natural give-and-take relationship is at play. Therefore, it is only natural that at times one side will be ahead of the other.

History offers an excellent example of this during the Hundred Years War between France and England. At that time, the cryptanalysts were ahead of the cryptographers. France believed that the Vigenère cipher was unbreakable. The British, however, cracked the code and broke it.

In another historical example, many historians now believe that the outcome of World War II largely turned on the fact that the winning side on both fronts was much more successful than the other at cracking the encryption of its enemy.

Given these examples, you might wonder who presently has the edge in this game of give and take. For the time being, conventional wisdom within the cryptology community holds that cryptographers currently have the edge. Of course, this can, and likely will, change some day.

So exactly how do we make such judgments about who is ahead or what code is unbreakable? In fact, in cryptography, it truly is impossible to prove that any given algorithm is “secure.” The best that can be accomplished is to show that the algorithm is not vulnerable to any known cryptanalytic attacks. This limits our certainty to an extent, because there may be methods that have been developed but as of yet are unknown, that could crack the algorithm. The one exception to this rule is a brute-force attack.

All algorithms are vulnerable to brute force. It is simply in the nature of the attack. In other words, if every possible key is tried, one of them will surely work. The issue is time.

Depending on the complexity of the algorithm, a brute-force attack could take an inordinate amount of time to ultimately succeed. But no algorithm is truly unbreakable.

### **Cryptography Through the Ages**

Cryptography has a long and storied past, dating back to the courts of kings, who would use early encryption to secure messages sent to other courts. Even these early times involved a degree of intrigue, because some of the courts involved would attempt to steal any message sent to an opposing kingdom.

From the courts of kings to the tools of military commanders, encryption was quickly adopted as a means of securing communications. Because messengers sometimes were killed as they transported critical military messages, encryption was seen as an indispensable means of securing these communications even if they fell into enemy hands.

Even dating back to the days of Julius Caesar, encryption was used to secure communications. Caesar's simple substitution cipher was used on the battlefield to quickly encrypt messages to his commanders. Thomas Jefferson even invented an encryption system that many historians believe he used while serving as Secretary of State from 1790 to 1793.

One of the great advances in encryption came with a machine invented by Arthur Scherbius in 1918. This machine served as a template for the systems used by each of the major participants in World War II. Scherbius called his machine the Enigma and sold it to Germany. When speaking about the security of his machine, he estimated that if 1000 cryptanalysts tested four keys per minute, all day, every day, it would take 1.8 billion years to try them all.

Throughout World War II, both the Germans and the Allies created machines modeled on the Scherbius machine. Arguably, these were the most sophisticated encryption devices ever developed. To defend against this level of encryption, the British created what most call the world's first computer, the Colossus. The Colossus was then used to break the encryption that was used by Germany's Enigma.

### **The Substitution Cipher**

Put simply, a cipher is an algorithm for performing encryption and decryption. Typically, ciphers represent a series of well-defined steps that you can follow as a procedure. With a substitution cipher, one letter is substituted for another to encrypt a message. Substitution ciphers vary in complexity, but in their simplest form, the letter frequency of the original message is retained when character substitution is done.



As mentioned, Julius Caesar made use of a simple substitution cipher on ancient battlefields. During these times, each day would have a different key, and that key would be used to adjust the alphabet accordingly. Let's look at an example.

Let's say that the key for today is 10. In this case the letter to be substituted for A is the character in the alphabet that is ten spaces forward. In other words, to represent an A, we would substitute a K. If we follow this through the alphabet, a B would be an L, a C would be an M, and so on. To keep the nature of the substitution secret, each day the key might move a random number of places, and the process would begin again.

One of the shortcomings of this simple cipher is its vulnerability to frequency analysis. Let's say that a message has 15 occurrences of the letter B, and B is to be replaced by L. This would mean that although we are substituting the character, there would still be 15 occurrences of the letter L. So if a message were long enough, it would be vulnerable to frequency analysis. This is because the message would retain the frequency patterns found in the language even though the characters might be different.

Analysts trying to crack this cipher might look at the natural occurrence pattern for each letter in the English language. They could then use this to compare the frequency of a certain letter in the encrypted text. For instance, if the letter S appears in 20 percent of all English words, and the letter X appears in 20 percent of all the words that have been encrypted, analysts might conclude that for this cipher, X equals S. To defend against this core weakness of the substitution cipher, polyalphabetic ciphers were invented.

#### **The Vigenère Cipher**

Polyalphabetic ciphers were invented to make up for the shortcomings of the substitution cipher. The Vigenère cipher is an excellent example of this kind of cipher. It encrypts text through the use of a series of different Caesar ciphers based on the letters of a particular keyword. Although this is a simple form of polyalphabetic substitution, it still proves invulnerable to frequency analysis.

This form of encryption dates back to a book written in 1553 by Giovan Batista Belaso, although the name of this cipher came from Blaise de Vigenère, a French cryptographer. He was mistakenly credited with its invention, and to this day it carries his name.

Let's take a look at how we might use this cipher to encrypt a message. We begin with a key of SECRET. This key is then used to encode the message X MARKS THE SPOT. We encode the X by looking at the row starting with S for the letter in the X column. In this case, the X is replaced with P. Next we look for the row that begins with E for the letter M. This results in Q as our second character. To encode the full phrase, we would simply map the characters by row and column and continue our substitution.

### Transposition Ciphers

If you have ever seen the beginning of the movie *Sneakers*, as the letters on-screen scramble to then become the correct words, you have a slight feel for a transposition cipher. In these ciphers, no letters are replaced; they are just rearranged. A simple form of this might take a phrase like THE QUICK BROWN FOX and simply transpose the letters so that it becomes XOFNWORBKCIUQEHT.

The Rail Fence Cipher is another kind of transposition cipher in which the words are spelled out as if they were a rail fence. The following example uses a key of three to illustrate how this could be done:

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

To read this message, we need to follow the diagonal pattern along the rail fence. Using this form of encoding, the message THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG would be encoded as TUBNJEEYHQIKRWFUXUPDVRHLZDGECOOMOTAO. Much like the earlier example, no letters have been changed; rather, they have merely been transposed.

### Working with the One-Time Pad

The one-time pad has been around for more than 90 years. It was invented and patented in 1917 by Gilbert Vernam of AT&T. The idea behind the one-time pad was to have a stream cipher that would apply the exclusive OR (XOR) operation to plain text with a key. Vernam's idea was enhanced with a contribution from Joseph Maubourgne, a captain in the U.S. Army Signal Corps, who suggested the use of random data as a key.

The one-time pad represents such a significant contribution to cryptography that the NSA has called this patent "perhaps the most important in the history of cryptography." However, using this significant idea in a real-world application has a number of difficulties.

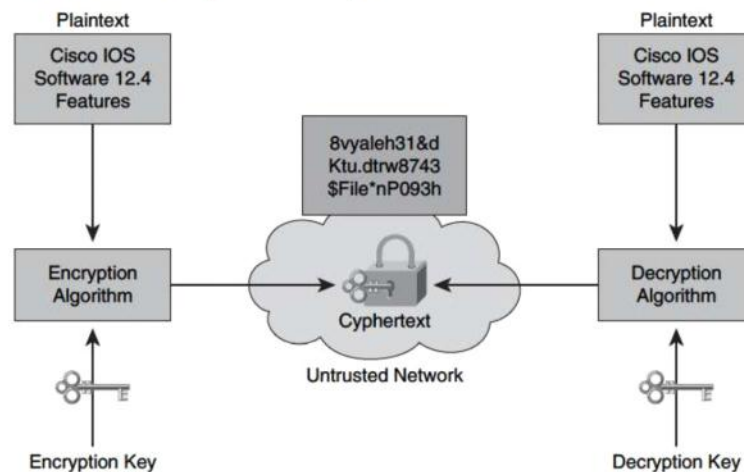
One of the more significant challenges is creating random data. On the surface, this sounds simple enough. However, because computers have a mathematical foundation, they cannot create random data. Another significant issue is that should a key be used more than once, it can easily be broken. Adding to these issues, key distribution can also be quite challenging.

One example in which the Vernam cipher has been successfully used is in RC4, which is widely used across the Internet. However, this is not a true one-time pad, because the key used is not random.

### The Encryption Process

Uses of encryption are all around us, from secured online purchases to transferring data through a VPN connection. When encryption is used, some form of plain, readable text is converted to ciphertext. Ciphertext represents this text in an unreadable form, whereas decryption is the process of reversing this process. The goal of encryption is to guarantee the confidentiality of data so that only those who have authorization may read the original message. Figure 12-1 shows plain text being transformed into ciphertext.

**Figure 12-1** Plain Text Transformed into Ciphertext



With the various older encryption algorithms we have examined, the key to their success was the secrecy of the algorithm. Today, reverse engineering is often quite simple, making this secrecy less important. Therefore, public-domain algorithms are often used. With these algorithms, successful decryption requires knowledge of the appropriate cryptographic keys. In other words, there has been a shift from the importance of the algorithm's secrecy to ensuring the secrecy of the keys.

Encryption is used to provide confidentiality in terms of the Open Systems Interconnection (OSI) layers in these ways:

- At the application layer, data encryption is used for secure e-mail, secure database sessions (Oracle SQL\*net), and secure messaging (Lotus Notes sessions).
- At the session layer, data is encrypted using a protocol such as Secure Socket Layer (SSL) or Transport Layer Security (TLS).

Key  
Topic



- At the network layer, data is encrypted using protocols such as those that make up the IPsec protocol suite.

### Cryptanalysis

When we seek to break encoded data, this undertaking is called cryptanalysis. An attacker who is attempting to break an algorithm or encrypted ciphertext may use one of a variety of attacks:



- Chosen plain-text attack
- Chosen ciphertext attack
- Birthday attack
- Meet-in-the-middle attack
- Brute-force attack
- Ciphertext-only attack
- Known plain-text (the usual brute-force) attack

Table 12-2 describes these attack types to help you understand their usage as a form of cryptanalysis.



**Table 12-2** *Defining Attack Types*

Type of Attack	Description
Chosen plain-text attack	In this attack the attacker chooses what data the encryption device encrypts and then observes the ciphertext output. This is a more powerful attack than a known plain-text attack, because the attacker gets to choose the plain-text blocks to encrypt. This allows the attacker to choose plain text that could potentially yield more information about the key. However, the practicality of this attack may be called into question. This is because it is often difficult, if not impossible, to capture both the ciphertext and plain text. This would generally require that the trusted network be compromised as well, yielding access to confidential information.
Chosen ciphertext attack	<p>In this attack, the attacker may choose different ciphertexts to be decrypted. The attacker also has access to the decrypted plain text. This combination makes it possible for an attacker to search through the keyspace and determine which key decrypts the chosen ciphertext.</p> <p>This attack is somewhat like the chosen plain-text attack and, like that attack, it may not be too practical. Capturing both the ciphertext and plain text without first breaking into the trusted network would prove nearly impossible.</p>



**Table 12-2** *Defining Attack Types*Key  
Topic

Type of Attack	Description
Birthday attack	<p>The birthday attack derives its name from the statistical probability involved in two individuals in a group having the same birthday. Statisticians say that in a group of 23 individuals, the likelihood that two people will have the same birthday is greater than 50 percent.</p> <p>This attack is a form of brute-force attack focused on hash functions. If a given function, when supplied with a random input, returns one of <math>k</math> equally likely values, repeating the function with various inputs, the same output would be expected after <math>1.2k^{1/2}</math> times.</p>
Meet-in-the-middle attack	This is a known plain-text attack in which the attacker knows a portion of the plain text and the corresponding ciphertext. The attacker encrypts the plain text with every possible key, and the results are stored. The attacker then decrypts the ciphertext using every key until one of the results matches one of the stored values.
Brute-force attack	All encryption algorithms are vulnerable to a brute-force attack. In this attack, an attacker tries every possible key with the decryption algorithm. Generally, a brute-force attack will succeed about 50 percent of the way through the keyspace. To defend against this form of attack, modern cryptographers have to create a sufficiently large keyspace so that attacking it in this way requires too much time and money to be practical.
Ciphertext-only attack	<p>In this form of attack, the attacker has the ciphertext of several messages. Each of these messages has been encrypted using the same encryption algorithm. However, the attacker has no knowledge of the underlying plain text. To be successful, the attacker must recover the ciphertext of as many messages as possible.</p> <p>Alternatively, the attacker could deduce the key or keys used to encrypt the messages and use this to decrypt other messages encrypted with the same keys. This could be achieved using statistical analysis. Today, however, these attacks are no longer practical, because modern algorithms produce pseudorandom output that is resistant to statistical analysis.</p>
Known plain-text attack	<p>In this attack, like the ciphertext-only attack, the attacker has access to the ciphertext of several messages, and he also knows something about the plain text underlying that ciphertext. Knowing the underlying protocol, file type, and some characteristic strings that may appear in the plain text, an attacker then employs a brute-force attack to try keys until decryption with the correct key succeeds. Compared to other attacks, this may be the most practical attack. Attackers can usually assume the type and some features of the underlying plain text after capturing the ciphertext. Although it is more practical, the enormous keyspace employed by modern algorithms make it unlikely that this attack will succeed.</p>

**Understanding the Features of Encryption Algorithms**

Good encryption algorithms have several benefits:



- They are resistant to cryptographic attacks.
- They support variable and long key lengths and scalability.
- They create an avalanche effect.
- They have no export or import restrictions.

When an attacker sets out to penetrate data protected by a cryptographic algorithm, the best way to do so is to try to decrypt the data using all the possible keys. Of course, the time required to undertake such an attack is determined by the number of possible keys. In practical terms, this process could take quite a long time. In fact, when appropriately long keys are used, this form of attack generally is infeasible.

A couple of other desirable attributes of a good cryptographic algorithm are variable key lengths and scalability. It stands to reason that the longer the key used for encryption, the longer it will take for an attacker to break it. Having the scalability provided by flexible key lengths lets you select the strength and speed of encryption that you need. Let's compare a couple of possible key lengths.

If we were to use a 16-bit key (nowhere near the strongest possible), there would be 65,536 possible keys. This may sound like a large number of keys, but consider that a 56-bit key would yield  $7.2 * 10^{16}$  possible keys. As you can see, variable key lengths can provide an ever-increasing number of keys, creating ever-stronger levels of encryption.

Another desirable attribute is called the avalanche effect. It says that changing only a few bits of a plain-text message causes its ciphertext to be completely different. An encryption algorithm that provides the avalanche effect makes it possible for messages that are quite similar to be sent over an untrusted medium, because the encrypted (ciphertext) messages remain completely different.

Export and import restrictions must also be carefully considered when you use encryption internationally. Certain countries prohibit the export of encryption algorithms or allow only the export of algorithms with smaller (more easily broken) keys. Other countries have strict regulations and restrictions governing the import of cryptographic algorithms. Before importing or exporting a cryptographic algorithm internationally, it is best to check with the governments involved to better understand their laws and regulations.

The U.S. has specific restrictions for the export of cryptographic algorithms, but in January 2000 these restrictions were substantially relaxed. At present, any cryptographic product

may be exported under a license exception unless the end users are governments outside the U.S. or are among those nations that have an embargo in place. To learn more about current practices for the import and export of cryptographic algorithms in the U.S., visit <http://www.commerce.gov>.

## Symmetric and Asymmetric Encryption Algorithms

This section discusses both symmetric and asymmetric algorithms, noting their differences and uses. Most striking among these two widely used types of encryption algorithms is their differences in key usage. Whereas symmetric encryption algorithms use a single key for both encryption and decryption, asymmetric encryption algorithms employ two separate keys—one for encryption and the other for decryption. Other differences will also be discussed with regard to the algorithms' speed and complexity.

### Encryption Algorithms and Keys

Ciphers are two-part mathematical functions that encrypt and decrypt data. Exposure of the algorithm itself could compromise the security of the encryption system if it is based on the algorithm's secrecy. If this should happen, each party working with the algorithm must change it. However, this is a dated view of cryptography. In modern cryptography, all algorithms are public, and complex cryptographic keys are used to ensure the secrecy of the data.

Cryptographic keys are created from sequences of bits that, together with the data that will be encrypted, are input into an encryption algorithm. Table 12-3 describes the two classes of encryption algorithms and details their use of keys.

**Table 12-3** *Classes of Encryption Algorithms*

Class of Algorithm	Description
Symmetric encryption algorithms	This class of algorithm employs the same key to both encrypt and decrypt data.
Asymmetric encryption algorithms	This class of algorithm employs two separate keys. One key is used to encrypt data, and the other key is used to decrypt data.

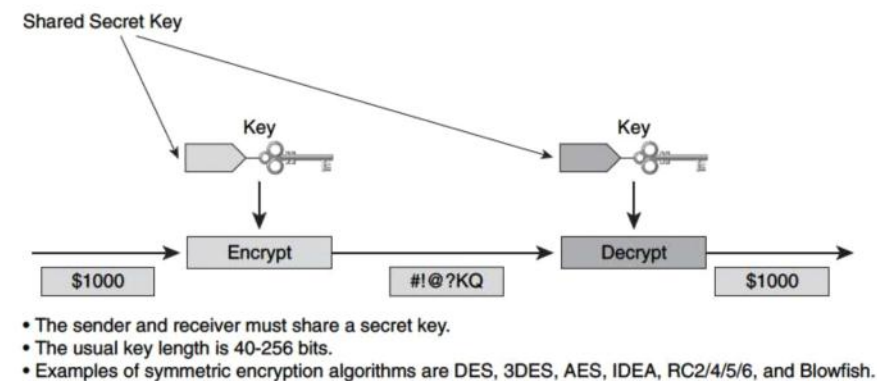
Key  
Topic

### Symmetric Encryption Algorithms

As shown in Table 12-3, symmetric encryption algorithms use the same key for both encryption and decryption. This means that both the sender and receiver must share the same secret key to transfer data securely. Figure 12-2 shows how symmetric encryption encrypts and decrypts data.



**Figure 12-2** *Symmetric Encryption and Decryption Process*



For a symmetric algorithm to be secure, the key itself must remain a secret. Should this key become available, anyone holding it could encrypt and decrypt messages. Because of this need for security, symmetric encryption is frequently called secret-key encryption. Symmetric encryption represents the more traditional form of cryptography. It uses key lengths ranging from 40 to 256 bits.

A number of well-known symmetric encryption algorithms exist. Table 12-4 details some of these, along with their key sizes.



**Table 12-4** *Popular Symmetric Algorithms*

Symmetric Algorithm	Key Size
DES	56-bit keys
Triple Data Encryption Standard (3DES)	112-bit and 168-bit keys
AES	128-bit, 192-bit, and 256-bit keys
International Data Encryption Algorithm (IDEA)	128-bit keys
RC2	40-bit and 64-bit keys
RC4	1-bit to 256-bit keys
RC5	0-bit to 2040-bit keys
RC6	128-bit, 192-bit, and 256-bit keys
Blowfish	32-bit to 448-bit keys

Symmetric encryption cryptography uses a number of different techniques. The most common are

- Block ciphers
- Stream ciphers
- Message Authentication Codes (MAC)



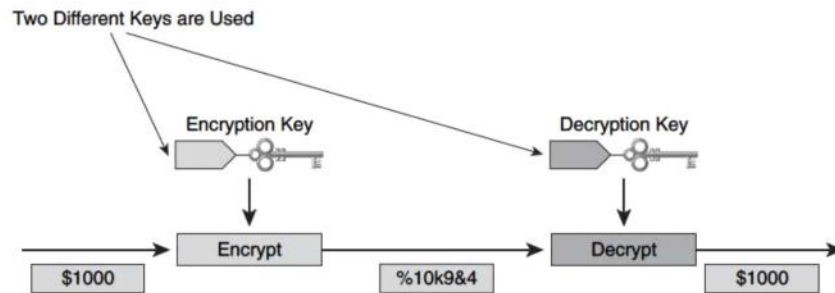
Symmetric algorithms generally are quite fast and therefore are a frequent choice to provide wire-speed encryption in data networks. Because symmetric algorithms are based on less-complex mathematical operations, they can be readily accelerated through the use of hardware. Due to their speed, symmetric algorithms may be used for bulk encryption when data privacy is required. One common example of their practical usage in this regard is to protect a VPN.

Despite the benefit of their speed, symmetric encryption algorithms do present a challenge with regard to key management. In particular, all parties involved in the communication must exchange the secret key over a secure channel before the encryption process can begin. This means that the security of any cryptographic system hinges on the ability of the key exchange method to protect the keys. Given this need, symmetric algorithms often are used to provide encryption services while additional key management algorithms are used to secure the key exchange.

#### **Asymmetric Encryption Algorithms**

Unlike symmetric algorithms, which use the same key for both encryption and decryption, asymmetric algorithms, often called public-key algorithms, use two different keys. One key is used for encryption, and the other is used for decryption. A central facet of this design is that the decryption key cannot feasibly be calculated from the encryption key, and vice versa. Figure 12-3 shows the encryption and decryption process using an asymmetric encryption algorithm.

With asymmetric algorithms, key lengths generally range from 512 to 4096 bits. However, no direct comparison can be made between the key length of asymmetric and symmetric algorithms, because the underlying design of these algorithm classes is quite different. In terms of resistance to brute-force attacks, experts generally agree that an RSA encryption key of 2048 bits generally is equivalent in strength to an RC4 key of only 128 bits.

**Figure 12-3** *Encrypting and Decrypting with Asymmetric Algorithms*

- Asymmetric encryption algorithms are best known as key algorithms.
- The usual key length is 512-4096 bits.
- Examples of asymmetric encryption algorithms are RSA, ElGamal, elliptic curves, and Diffie-Hellman.

One downside of symmetric algorithms is that they can be up to 1000 times slower than symmetric algorithms. This is because their design is based on complex mathematical calculations. Often these designs employ such things as factoring extremely large numbers or computing discrete logarithms of extremely large numbers.

Because of the issues with the speed of these algorithms, they generally are used in low-volume cryptographic mechanisms. For instance, they might be employed in digital signatures or for key exchange. One noted benefit is that key management of these algorithms generally is less complex than for symmetric algorithms. This stems from the fact that typically one of the two keys, either the encryption or decryption key, may be made public.

### The Difference Between Block and Stream Ciphers

Both block and stream ciphers have their place in encryption algorithms as a mechanism for generating ciphertext from plain text. This section explores their basic differences and their uses in modern cryptography.

#### Block Ciphers

Block ciphers derive their name from the fact that they transform a fixed-length “block” of plain text into a “block” of ciphertext. These two blocks are of the same length. When the reverse transformation is applied to the ciphertext block, by using the same secret key, it is decrypted. Block ciphers use a fixed length or block size. Generally this is 128 bits, but they can range in size. For instance, DES has a block size of 64 bits.

The concept of block size determines how much data may be encrypted at a given time. This varies according to key length, because the key length refers to the size of the encryption

key. To use the example from earlier, DES encrypts blocks in 64-bit chunks but does so using a 56-bit key length.

Because ciphertext must always be a multiple of the block size, the output data from a block cipher is larger than the input data. Block algorithms work with data one chunk at a time, such as 8 bytes, and then use padding to add artificial data (blanks) should there be less input data than one full block. Here are some of the more common block ciphers:

- DES and 3DES, running in Electronic Code Book (ECB) or Cipher Block Chaining (CBC) mode
- Skipjack
- Blowfish
- RSA
- AES
- IDEA
- Secure and Fast Encryption Routine (SAFER)

Key  
Topic

#### Stream Ciphers

Stream ciphers use smaller units of plain text than what are used with block ciphers; typically they work with bits. Transformation of these smaller plain-text units also varies, depending on when during the encryption process they are encountered. One of the great benefits of stream ciphers compared to block ciphers is that they are much faster and generally do not increase the message size. This is because they can encrypt an arbitrary number of bits.

Here are some common stream ciphers:

- RC4
- DES and 3DES, running in output feedback (OFB) or cipher feedback (CFB) mode
- Software Encryption Algorithm (SEAL)

Key  
Topic

## Exploring Symmetric Encryption

Encryption algorithms use encryption keys to provide confidentiality of encrypted data. With symmetric encryption algorithms, the same key is used to encrypt and decrypt data. This section explores the principles that underlie symmetric encryption. It also examines



some of the major symmetric encryption algorithms and discusses the means by which they operate, their strengths, and their weaknesses.

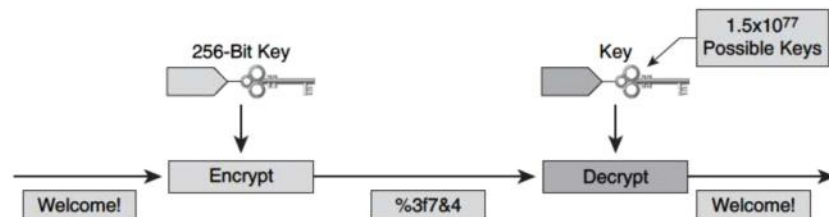
### Functionality of Symmetric Encryption Algorithms

Because of the simplicity of their mathematics and the speed at which they operate, symmetric algorithms are the most commonly used form of cryptography. Symmetric encryption algorithms are also stronger. Therefore, they can use shorter key lengths compared to asymmetric algorithms. This helps increase their speed of execution in software.

#### Key Lengths

Key lengths for current symmetric algorithms range from 40 to 256 bits, giving symmetric algorithms keyspaces that range from  $2^{40}$  (1,099,511,627,776) possible keys to  $2^{256}$  ( $1.5 \times 10^{77}$ ) possible keys. As discussed previously, a large key space is central to determining how vulnerable an algorithm will be to a brute-force attack. Figure 12-4 shows a symmetric algorithm with  $2^{256}$  possible keys.

**Figure 12-4** Key Lengths for Symmetric Encryption



At the low end, a key length of 40 bits may be easily broken using a brute-force attack. On the other hand, if your key length is 256 bits, it is not likely that a brute-force attack will succeed. The keyspace generated with a 256-bit key is simply too large to easily fall victim to a brute-force attack.

Table 12-5 illustrates ongoing expectations for key lengths, assuming that the algorithms are mathematically and cryptographically sound. A further assumption in such calculations is that computing power will continue to keep pace with its present rate of growth and that capacity to perform brute-force attacks will also increase at the same rate. Note that if a method other than brute-force is discovered to crack a given algorithm, the key lengths in the table become obsolete.

Table 12-5 Key Lengths and Their Continued Protection

Key  
Topic

	Symmetric Key	Asymmetric Key	Digital Signature	Hash
Protection up to three years	80	1248	160	160
Protection up to ten years	96	1776	192	192
Protection up to 20 years	112	2432	224	224
Protection up to 30 years	128	3248	256	256
Protection against quantum computers	256	15,424	512	512

Features and Functions of DES

One of the most well-known and most widely used symmetric encryption algorithms is Data Encryption Standard (DES). DES typically operates in block mode, where it encrypts data in 64-bit blocks. Like other symmetric algorithms, DES uses the same algorithm and key for both encryption and decryption. DES has stood the test of time. Cryptography researchers have scrutinized it for nearly 35 years and so far have found no significant flaws. Adding to its appeal, because DES is based on relatively simple mathematical functions, it may be easily implemented and accelerated in hardware.

Working with the DES Key

DES employs a fixed key length of 64 bits, but only 56 of these bits are used for encryption; the other 8 bits are used for parity. The least-significant bit of each key byte indicates odd parity.

This means that each DES key is always 56 bits long. If DES is used with a weaker encryption, such as a 40-bit key, this means that the encryption key is 40 secret bits and 16 known bits, so the key length remains at 56 bits. In this case, however, DES would have a key strength of only 40 bits.

Modes of Operation for DES

DES uses two different types of ciphers to encrypt or decrypt more than 64 bits of data—the block cipher and the stream cipher.

- **Block ciphers** use fixed-length groups of bits known as blocks, with an unvarying transformation.
- **Stream ciphers** operate on individual digits one at a time, with the transformation varying during the encryption.

Key  
Topic



For block cipher mode, DES uses two standardized modes:

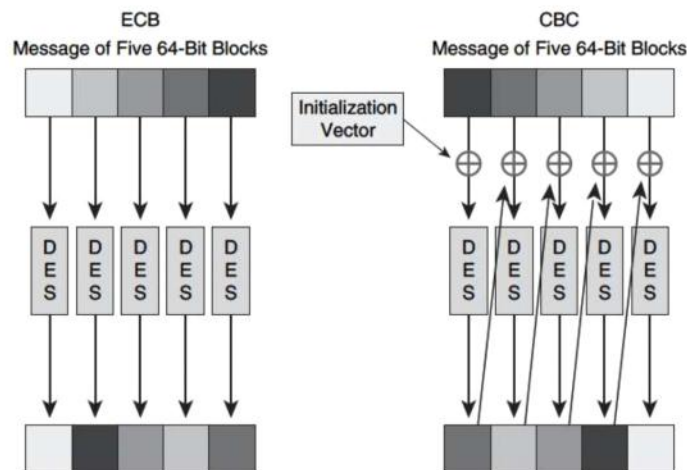
- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)

ECB mode uses the same 56-bit key to serially encrypt each 64-bit plain-text block. Should two identical plain-text blocks be encrypted using the same key, their ciphertext blocks are the same. This means that an attacker could identify similar or identical traffic as it flows across a communications channel. The attacker could use this information to help build a catalogue of messages that have a certain meaning, and then replay them later, without knowing their real meaning. For instance, suppose an attacker captures a login sequence for a user who has administrative privilege and whose traffic is protected by DES-ECB, and then replays it. This sort of risk must be mitigated, and that is why CBC was invented.

With CBC mode, each 64-bit plain-text block is exclusive ORed (XORed) bitwise with the previous ciphertext block. It is then encrypted using the DES key. This means that the encryption of each block depends on previous blocks, and encryption of the same 64-bit plain-text block can result in different ciphertext blocks. Thanks to this, CBC mode can help guard against certain attacks. Of course, it cannot help guard against sophisticated cryptanalysis or if an attacker launches an extended brute-force attack.

Figure 12-5 shows the differences between ECB mode and CBC mode.

**Figure 12-5** DES ECB Mode Versus CBC Mode



Cisco IP Security (IPsec) implementation currently uses DES and Triple Data Encryption Standard (3DES) in CBC mode.

#### Working with DES Stream Cipher Modes

When working with DES in stream cipher mode, the cipher uses previous ciphertext along with the secret key to generate a pseudorandom stream of bits. This may only be generated by the secret key.

To encrypt data, it is XORed with the pseudorandom stream on a bit-by-bit basis. Alternatively, this may be done byte by byte to obtain the ciphertext. To decrypt the data, the process is the same. The receiver uses the secret key to generate the same random stream and then XORs the ciphertext with the pseudorandom stream to gain access to the plain text.

If it is necessary to encrypt or decrypt more than 64 bits of data, two common stream cipher modes may be used:

- **Cipher feedback (CFB)** is similar to CBC. It may be used to encrypt any number of bits, even single bits or single characters.
- **Output feedback (OFB)** generates keystream blocks that are then XORed with the plain-text blocks to generate the ciphertext.

Key  
Topic

#### Usage Guidelines for Working with DES

You should consider a number of things when seeking to protect the security of DES-encrypted data, as described in Table 12-6.

**Table 12-6** *Considerations for Protecting the Security of DES-Encrypted Data*

Consideration	Description
Change keys	Keys should be changed frequently to help prevent brute-force attacks.
Use a secure channel	A secure channel from the sender to the receiver should be used to communicate the DES key.
Use CBC mode	Using DES in CBC mode means that the encryption of each 64-bit block depends on the previous block, making this more secure.
Avoid weak keys	Be sure to test a key before using it to check it for weakness. DES has four weak keys and 12 semiweak keys. Testing will not significantly impact encryption time and can prevent the use of a weak key.

Key  
Topic



### Understanding How 3DES Works

As mentioned, DES, with its original 56-bit key, is too short to withstand even medium-budget attackers. One means of increasing the security of DES without changing the well-analyzed algorithm itself is to use the same algorithm but with different keys multiple times in a row. In essence, that is what 3DES does.

By applying DES three times in a row to a plain-text block, we have what is known as 3DES. This application of DES three times with different keys makes brute-force attacks on 3DES infeasible. This stems from the fact that the basic algorithm has stood the test of time, weathering 35 years in the field and proving quite trustworthy.

### Encrypting with 3DES

To encrypt plain text, 3DES uses a method called 3DES-encrypt-decrypt-encrypt (3DES-EDE). Figure 12-6 shows the 3DES-EDE encryption process, described in the following steps:

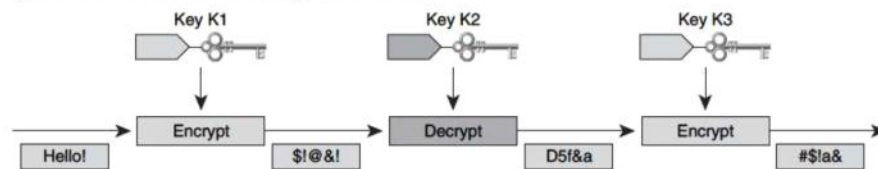


**Step 1** The message to be secured is encrypted using the first 56-bit key (K1).

**Step 2** Data is decrypted using the second 56-bit key (K2).

**Step 3** Data to be secured is again encrypted using a third 56-bit key (K3).

**Figure 12-6** 3DES-EDE Encryption Process



By applying the keys as it does, the 3DES-EDE process provides encryption with an effective key length of 168 bits. Should keys K1 and K3 be equal, a less-secure encryption of 112 bits is achieved.

To decrypt a message that has been encrypted with this process, the following steps, which are the opposite of the 3DES-EDE method, are used:

**Step 1** Use key K3 to decrypt the ciphertext.

**Step 2** Use key K2 to encrypt the data.

**Step 3** Use key K1 to decrypt the data.

Simply encrypting data three times with three different keys does not significantly increase security. To achieve security, the 3DES-EDE method must be employed. In fact, if we were

to simply encrypt data three times in a row using three different 56-bit keys, we would generate an effective 58-bit key strength, rather than the full 168-bit key strength we achieve by using 3DES-EDE.

## AES

Although DES has withstood the test of time, it has been recognized for some time that DES would eventually reach the end of its usefulness. The Advanced Encryption Standard (AES) initiative was announced in 1997. The public was invited to propose candidate encryption schemes to be evaluated as the encryption standard to replace DES.

### The Rijndael Cipher

The Rijndael cipher was selected as the AES algorithm in October 2000 by the U.S. National Institute of Standards and Technology (NIST). In 2002 the U.S. Secretary of Commerce approved the adoption of AES as an official U.S. government standard. Joan Daemen and Vincent Rijmen developed the Rijndael cipher, which employs a variable block length and key length. The algorithm provides nine different combinations of key length and block length. Keys with a length of 128, 192, or 256 bits may be used to encrypt blocks with a length of 128, 192, or 256 bits.

The Rijndael cipher is an iterated block cipher. In this cipher the initial input block and cipher key undergo multiple transformation cycles before producing output. This algorithm can operate over variable-length blocks using variable-length keys. Currently, the AES implementation of Rijndael contains only some of the capabilities of the Rijndael algorithm. One of the key features of this algorithm is that it is written so that the block length or the key length (or both) may be extended easily in multiples of 32 bits. This system was designed for efficient implementation in either hardware or software on a range of processors.

### Comparing AES and 3DES

The key length of AES is much stronger than that of DES, and AES runs much faster than 3DES on comparable hardware. With these features, AES was chosen to replace DES and 3DES. AES is also better suited for high-throughput, low-latency environments. This is especially true when pure software encryption is used.

In terms of longevity, AES is a relatively young algorithm. As mentioned previously, a more mature algorithm is always more trusted. That being the case, 3DES represents a more conservative yet more trusted choice in terms of strength, because it has been analyzed for nearly 35 years.

#### Availability of AES in the Cisco Product Line

Cisco offers AES implementation in a number of virtual private network (VPN) devices as an encryption transform, applied to IPsec-protected traffic:



- Cisco PIX Firewall Software version 6.3 and later
- Cisco ASA Software version 7.0 and later
- Cisco VPN 3000 Software version 3.6 and later
- Cisco IOS Release 12.2(13)T and later

#### SEAL

For those seeking an alternative algorithm to software-based DES, 3DES, and AES, SEAL encryption uses a 160-bit encryption key. SEAL also offers the benefit of having less impact on the CPU compared to other software-based algorithms. Cisco IOS IPsec implementations feature SEAL encryption and provide support for the SEAL algorithm. The Cisco IOS software Release 12.3(7)T also added support for SEAL.

#### SEAL Restrictions

SEAL is bound by several restrictions:



- IPsec must be supported by your Cisco router and the other peer.
- The k9 subsystem must be supported by your Cisco router and the other peer.
- Only Cisco equipment supports this feature.

A further restriction is that your router and the other peer must not have hardware IPsec encryption.

#### The Rivest Ciphers

Many networking applications employ the Rivest cipher (RC) family of algorithms. This is because of their favorable speed and variable key-length capabilities.

Ronald Rivest played a significant role in designing all or at least part of all the RC algorithms. Table 12-7 describes some of the most widely used RC algorithms.



Table 12-7 Most Widely Used RC Algorithms



RC Algorithm	Description
RC2	Designed as a drop-in replacement for DES, RC2 is a variable key-sized cipher.
RC4	Often used in file encryption products, as well as for secure communication, such as in Secure Socket Layer (SSL), RC4 is a variable key-size stream cipher.
RC5	This fast block cipher has a variable block size and variable key length. With its 64-bit block size, it may be used as a drop-in replacement for DES.
RC6	Based on RC5, this block cipher had as its main design goal meeting the requirement of AES.

Of the various RC algorithms listed in Table 12-7, the most popular is RC4. RC4 represents a variable key-size stream cipher that employs byte-oriented operations and is based on the use of a random permutation. Via analysis, it has been determined that the period of the cipher is quite large, likely greater than  $10^{100}$ . To give you a better sense of this, each output byte requires from eight to 16 machine operations and can be expected to run very quickly in software.

RC4 is considered a secure algorithm and as such is often used for file encryption. It is also used frequently to encrypt website traffic within the context of the SSL protocol.

## Understanding Security Algorithms

It is almost hard to imagine modern computing and networking without also thinking about the mechanisms that provide for the underlying security of the data that resides on these systems or travels across the wire. Security algorithms are central to securing the data created within an organization, as well as securing it in transit. This section examines the characteristics of the encryption process and what makes for a strong, trustworthy encryption algorithm. This section also explores the use of cryptographic hashes and how key management plays an important role in securing the encryption process.

### Selecting an Encryption Algorithm

Proper selection of an encryption algorithm is one of the key steps in building a cryptography-based solution. With this selection process, two main criteria should be considered. Table 12-8 details these selection criteria for your consideration.

**Table 12-8** *Criteria for Selecting an Encryption Algorithm*

Selection Criteria	Description
Trust in the algorithm by the cryptographic community	Because many new algorithms are often broken very quickly, algorithms that have stood the test of time by resisting attacks for a number of years are the most preferred. Although there is often a great deal of talk about the benefits of a new algorithm, there are truly few or no revolutions in cryptography.
Protection against brute-force attacks	A trusted algorithm has no shortcut to break it. An attacker has to search through the keyspace to guess the correct key. An algorithm also must allow key lengths that satisfy an organization's confidentiality requirements. This is not always the case. For example, DES does not provide enough protection for most organizations because of its short key length.

The following symmetric encryption algorithms are considered trustworthy:



- DES
- 3DES
- IDEA
- RC4
- AES

Each of these algorithms has its place. For instance, because it uses very short key lengths, DES is a good protocol to protect data for a very short period of time. If you need to protect data with an algorithm that is very trusted and has much greater security strength, 3DES would be a much better choice.

AES, although not proven to the degree that 3DES has been, is still a good choice, because it is more efficient. This makes it ideal in high-throughput, low-latency environments. This is particularly the case when 3DES cannot handle the throughput or latency requirements. Over time, AES will likely gain even greater trust as more attacks are attempted against it.

Symmetric encryption algorithms such as RSA and Diffie-Hellman (DH) are considered trustworthy for confidentiality. But many others, like ECC, generally are considered immature in cryptographic terms.

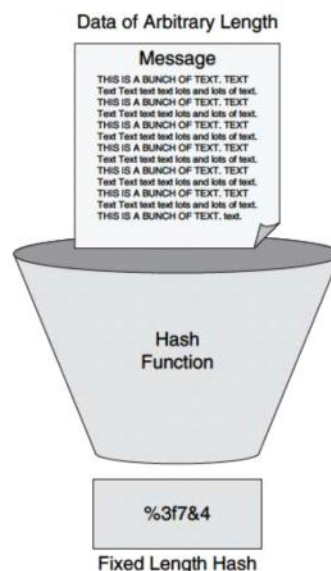
## Understanding Cryptographic Hashes

Hashing is used to provide data integrity. Hashes are based on one-way mathematical functions. These can be easy to compute but extremely challenging to reverse. The process of hashing and the difficulty of reversing the hash is akin to scrambling an egg and then trying to put it back together again.

## Working with Hashing

The way that hashing works in practice is that data of arbitrary length is input into the hash function and then is processed through the function, resulting in a fixed-length hash. The resultant fixed-length hash is called a digest or fingerprint. Figure 12-7 shows the hashing process.

**Figure 12-7** Hashing Process



If you are familiar with the calculation of cyclic redundancy check (CRC) checksums, hashes are quite similar to this but are cryptographically much stronger. If you're not too familiar with CRC, if you have the CRC value, it is relatively easy to generate data with the same CRC. Because of the strength of hash functions, it is computationally infeasible for an attacker to possess two separate sets of data that would come up with the same fingerprint.

### Designing Key Management

One of the most challenging aspects of designing a cryptosystem is planning for key management. In fact, cryptosystem failures have occurred because of shortcomings in key management. Each of the current cryptographic algorithms requires the services of key management procedures, making this an extremely important area to consider. For an attacker, the general target when seeking to attack a cryptographic system is the key management system, rather than the algorithm itself.

### Components of Key Management

When considering key management, you must consider several components that address the life cycle of key management from generation to destruction:



- Key generation
- Key verification
- Key storage
- Key exchange
- Key revocation and destruction

Modern cryptographic systems generate keys automatically, rather than leaving it to the end user. To help ensure that all keys are likely to be equally generated, so that the attacker cannot predict which keys are likely to be used, quality random-number generators are necessary. It is not uncommon for a cryptographic algorithm to have some weak keys that should not be used. Proper key verification procedures should be used to regenerate these keys when they occur.

Key storage is another factor that must be considered. With today's modern multiuser operating systems that work with cryptography, a key can be stored in memory. When memory is swapped to the disk, it presents a possible problem, because a Trojan horse program, if installed on the PC, could then gain access to that user's private keys.

A secure key exchange mechanism is also necessary. It should allow secure agreement on the keying material with the other party, likely over an untrusted medium.

The final element of good key management to consider is key revocation and destruction. The process of key revocation notifies all the parties involved that a given key has been compromised and should not be used. Key destruction goes beyond this by erasing old keys so that a malicious attacker cannot recover them.

### Understanding Keyspaces

The keyspace of an algorithm represents a defined set of all possible key values. For each key of  $n$  bits, a keyspace is produced that has  $2^n$  possible key values. This means that adding 1 bit to the key would effectively double the size of the keyspace.



Let's look at DES by way of an example. DES employs a 56-bit key and with this produces a keyspace of more than 72,000,000,000,000,000 ( $2^{56}$ ) potential keys. If we were to add 1 bit to the key length, the keyspace would double. That means that an attacker would need twice as much time to search the keyspace.

No algorithm is without some weak keys in its keyspace, as discussed previously. These weak keys may enable an attacker to break the encryption via a shortcut. So what exactly constitutes a weak key?

A key is said to be weak when it shows regularities in encryption or poor encryption. A good example is DES. DES has four keys for which encryption is exactly the same as decryption. Should one of these weak keys be encrypted twice, the original plain text would be recovered.

The chance that such keys would be chosen is almost unimaginable. However, each implementation should still verify all keys and take steps to prevent weak keys from being used. This is particularly the case with manual key generation, so you should take special care to avoid defining weak keys.

#### Issues Related to Key Length

As we have discussed, the only way to break a proven cryptographic system is with a brute-force attack. These attacks search the entire keyspace, trying all possible keys, until the key that decrypts the data is found. To defend against this form of attack, the keyspace must be sufficiently large enough that such a search would require an enormous amount of time, rendering this form of attack impractical.

Even for successful brute-force attacks, generally an attacker has to search half the keyspace to find the correct key. Of course, the time required for such a search depends on the computer resources available to the attacker. With key lengths of significant size, such an attack could take many millions, if not billions, of years to yield success.

The protection strength of modern trusted algorithms depends exclusively on the length of the key. You should select a key length so that it protects data confidentiality or integrity for an adequate period of time. The more sensitive the data, and the longer the period required for secrecy, the longer the key that must be used.

When considering the level of protection required, you must also take into account the characteristics of those likely to attack your data. For instance, you must estimate the attacker's resources and how long you must protect the data.

For example, suppose a would-be attacker has \$1 million of funding that can be used toward the attack, and the data must be protected for a period of no less than one year. What form

of encryption should we choose? Classic DES would not be a good choice, because it can be broken by a \$1 million machine in only a couple of minutes. If instead we employed 168-bit 3DES or even 128-bit RC4, it would take that same attacker, funded with that same \$1 million, a million years or more to crack into your data. Considering our attacker, his funding and our need for security are both important in selecting the proper key length.

Another issue impacting the choice of key length is performance. It is important to strive to balance speed and protection strength for the selected algorithm. Certain algorithms, such as RSA, take much longer to run with large key sizes. We should strive for adequate protection, without hindering communication over untrusted networks.

Finally, you also need to be aware that because of rapid advances in technology and cryptanalytic methods, what may be an adequate key size today may quickly no longer be appropriate. The National Institute of Standards and Technology (NIST) offers recommendations on adequate key lengths for various applications. You may review these on the NIST website at <http://www.keylength.com/en/4/>.

### SSL VPNs

Security on the Internet for such applications as web browsing, e-mail, Internet faxing, instant messaging, and other forms of data transfer is provided by cryptographic protocols. Among the most popular choices to support these applications are Transport Layer Security (TLS) and its predecessor, Secure Socket Layer (SSL). Although there are subtle differences between SSL and TLS, the actual protocol remains quite similar.

Both SSL and TLS support a variety of cryptographic algorithms, or ciphers, to help provide such functions as authenticating the server and client to each other, transmitting certificates, and establishing session keys. For bulk encryption, symmetric algorithms are used. Asymmetric algorithms are used for authentication and key exchange. Hashing is used as part of the authentication process.

With an SSL-based VPN, you can easily provide remote-access connectivity from almost any Internet-enabled location. All that is needed is a standard web browser and its native SSL encryption. There is no need for special-purpose client software on the remote system. This flexibility allows SSL VPNs to provide “anywhere” connectivity from corporate desktops, as well as from noncompany-managed desktops. Employees may use the SSL-based VPN to connect from home on their own PCs, contractor or business partner desktops can also be easily connected, and users can even connect via Internet kiosks. Through dynamic download, clients are supplied with all the software needed for application access across the SSL VPN connection. This feature dramatically minimizes the maintenance of desktop software.

If you need to support the remote resource needs of a diverse user base, SSL VPNs and IPsec VPNs provide complementary technologies that can be deployed together to meet these needs. Each of these VPN solutions offers access to virtually any network application or resource from a remote location. However, SSL VPNs do offer some additional features, allowing for easy connectivity from desktops outside your company's management, as well as little or no desktop software maintenance, and user-customized web portals upon login.

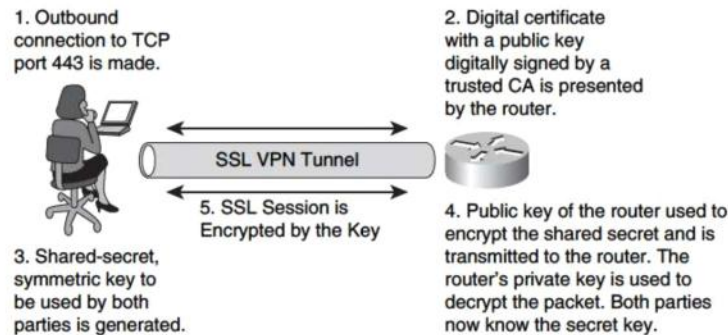
### Establishing an SSL Tunnel

Let's examine the steps involved in establishing an SSL tunnel (see Figure 12-8):

- Step 1** The user makes an outbound connection to TCP port 443.
- Step 2** The router presents a digital certificate that contains a public key that is digitally signed by a trusted certificate authority (CA).
- Step 3** The user's computer generates a shared-secret symmetric key that will be used by both parties.
- Step 4** The router's public key is used to encrypt the shared secret and is transmitted to the router. The router's software uses the private key to decrypt the packet. When this is complete, both parties in the session know the secret key.
- Step 5** The key encrypts the SSL session.

Key  
Topic

**Figure 12-8** Establishing an SSL Tunnel



With the SSL tunnel established, two parties may securely transmit data. By using the mechanisms discussed here, you can effectively extend the reach of your corporate network, allowing users to easily and securely gain access to corporate resources from wherever they may be.



## Exam Preparation Tasks

### Review All the Key Topics



Review the most important topics from this chapter, denoted with the Key Topic icon. Table 12-9 lists these key topics and the page where each is found.

**Table 12-9** *Key Topics for Chapter 12*

Key Topic Element	Description	Page Number
List	How encryption works in terms of the OSI model	437
List	Various cryptographic attacks	438
Table 12-2	Defining attack types	438-439
List	Features of quality encryption algorithms	440
Table 12-3	Classes of encryption algorithms	441
Table 12-4	Popular symmetric algorithms	442
List	Symmetric encryption techniques	443
List	Common block ciphers	445
List	Common stream ciphers	445
Table 12-5	Key lengths and their continued protection	447
List	DES ciphers	447
List	Standardizing block cipher modes with DES	448
List	Common stream cipher modes	449
Table 12-6	Considerations for protecting the security of DES-encrypted data	449
Steps	Detailed steps used in the 3DES-EDE encryption process	450
List	The Cisco AES implementation for VPN devices	452
List	SEAL restrictions	452
Table 12-7	Most widely used RC algorithms	453
Table 12-8	Criteria for selecting an encryption algorithm	454
List	Trustworthy symmetric encryption algorithms	454
List	Components of key management	456
Steps	Detailed steps for establishing an SSL tunnel	459



## Complete the Tables and Lists from Memory

Print a copy of Appendix D, “Memory Tables,” (found on the CD) or at least the section for this chapter, and complete the tables and lists from memory. Appendix E, “Memory Tables Answer Key,” also on the CD, includes completed tables and lists so that you can check your work.

## Definition of Key Terms

Define the following key terms from this chapter, and check your answers in the glossary:

Advanced Encryption Standard (AES), block cipher, cryptography, ciphertext, Data Encryption Standard (DES), Triple Data Encryption Standard (3DES), hashing, keyspace, Rivest Cipher (RC) algorithms, Software Encryption Algorithm (SEAL), stream cipher



---

**This chapter covers the following topics:**

**Examining hash algorithms:** This section explores the construction and use of hash algorithms. It examines a variety of hash algorithms and discusses their relative strengths and weaknesses in practical application.

**Using digital signatures:** This section examines the components that make up a digital signature, as well as the application of these as a means of proving the authenticity of a message.

## Implementing Digital Signatures

---

As you examine the security at play in your network and seek to increase your defenses, it is important to have a general understanding of cryptography and digital signatures. In cryptography, a cryptographic hash function is a transformation that takes an input and returns a string, which is called the hash value. Digital signatures are rather like written signatures in that they are used to provide authentication of the associated input, typically called a message. These messages can be anything from an e-mail to a username-and-password combination or even a message sent with an even more sophisticated cryptographic tool. This chapter examines digital signatures so that you can better understand how these may be implemented to secure your network.

### “Do I Know This Already?” Quiz

The “Do I Know This Already?” quiz helps you determine your level of knowledge of this chapter’s topics before you begin. Table 13-1 details the major topics discussed in this chapter and their corresponding quiz questions.

**Table 13-1** “Do I Know This Already?” Section-to-Question Mapping

Foundation Topic Section	Questions
Examining Hash Algorithms	1 to 5
Using Digital Signatures	6 to 11

1. Cryptographic hashes can be used to provide which of the following? (Choose all that apply.)
  - a. Message integrity
  - b. Functional analysis
  - c. Security checks
  - d. Message lists
  - e. Digital signatures

2. Which of the following is an example of a function intended for cryptographic hashing?
  - a. MD65
  - b. XR12
  - c. SHA-135
  - d. MD5
3. An HMAC provides which of the following benefits? (Choose all that apply.)
  - a. It may be used to verify data integrity.
  - b. It may be used to calculate a checksum.
  - c. It may be used to verify a message's authenticity.
  - d. It may be used to examine a message header.
4. What may be added to a password stored in MD5 to make it more secure?
  - a. Cryptotext
  - b. Ciphertext
  - c. Rainbow table
  - d. Salt
5. Which of the following employ SHA-1? (Choose all that apply.)
  - a. SMTP
  - b. SSL
  - c. TLS
  - d. IGMP
  - e. IPsec
6. A digital signature provides which of the following?
  - a. Auditing
  - b. Authentication
  - c. Authorization
  - d. Analysis



7. Digital signatures employ a pair of keys made up of which of the following? (Choose two.)
  - a. A personal key
  - b. A public key
  - c. A private key
  - d. A universal key
8. A digital signature scheme is made up of which of the following? (Choose all that apply.)
  - a. Authentication algorithm
  - b. Key generation algorithm
  - c. Encryption algorithm
  - d. Signing algorithm
  - e. Signature verification algorithm
9. Which of the following algorithms was the first to be found suitable for both digital signing and encryption?
  - a. MD5
  - b. HMAC
  - c. SHA-1
  - d. RSA
10. Which of the following attacks focus on RSA? (Choose all that apply.)
  - a. Man-in-the-middle attack
  - b. BPA attack
  - c. Adaptive chosen ciphertext attack
  - d. DDoS attack
11. The Digital Signature Standard outlines the use of which of the following algorithms in the creation of digital signatures?
  - a. LSA
  - b. DSA
  - c. PGP
  - d. MD5

## Foundation Topics

### Examining Hash Algorithms

For centuries everyone from kings to generals to college students has wanted to ensure the authenticity of their communications. In this section we will examine the role of hash algorithms in helping provide this assurance through the process of *hashing*. Along the way we will examine hash functions and learn about HMAC, as well as explore MD5 and SHA-1. Let's begin with a brief overview of what a hash function does.

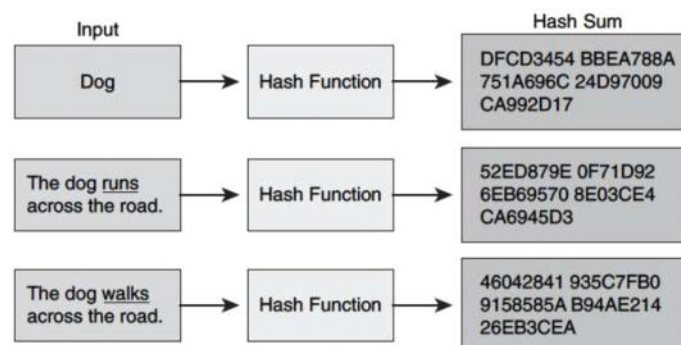
A hash function is a means of turning data into a relatively small number that then may act as a digital fingerprint of the data. The algorithm that is used substitutes or transposes the data to create this unique fingerprint. These fingerprints may be called hash sums, hash values, hash codes, or just hashes. Cryptographic hashes serve a variety of purposes in information security applications. They are used to do message integrity checks and provide digital signatures in various information security applications, such as authentication and message integrity.

### Exploring Hash Algorithms and HMACs

Figure 13-1 is an example of how a simple sentence can be transformed using a hash function to yield a cryptographic result. You can see that changing a single word alters the hash output.



**Figure 13-1** Hashing Example



Changing a single word in the text alters the output of the hash function.

Although you might not need to hash a simple sentence like this, many other applications exist in terms of network security. Hashes can be employed to help secure data as it traverses your network, as well as to secure login authentication credentials as a user is validated before accessing network resources.

### Anatomy of a Hash Function

A variety of hash functions exist, but they all share the common characteristic that they are built for speed and are designed to yield very few hash collisions in their expected input domains. A hash “collision” (sometimes called a “hash clash”) happens when two distinct inputs entered into a hash function produce identical outputs. Each hash function has the potential for collisions, but if you are working with a well-designed hash function, collisions should occur less frequently. In terms of hash functions, collisions inhibit the distinguishing of data, making records more costly to find in hash tables and data processing.

Another characteristic of hash functions is that they must be deterministic. In other words, if a hash function generates two hashes that are different, we can conclude that the two inputs were different in some way.

Hash values that are computed may be the same for different input values. This may seem odd, but it is because of the general requirement that the hash value must be able to be stored in fewer bits than the data that is being hashed. A primary design goal of hash functions is to minimize the likelihood of a hash collision.

One desirable property of a hash function is the mixing property. What this means is that a small change in the input (1 bit) should cause a large change in the output (about half of the bits). This significant change in the outcome is called the avalanche effect.

Most hash functions have what is called an infinite domain. This might be something like byte strings of arbitrary length. They also have a finite range—for instance, bit sequences of a fixed length. In some applications, hash functions may be designed with a one-to-one mapping between an identically sized domain and range. Hash functions such as these, which are one-to-one, are also called permutations. For these hash functions, reversibility is achieved by using a series of reversible “mixing” operations on the function input.

### Application of Hash Functions

Hash functions may be used for a variety of applications; therefore, they are often tailored to a given need. Cryptographic hash functions begin with the assumption that an adversary can deliberately try to find inputs with the same hash value. The creation of a well-designed cryptographic hash involves a one-way operation in which no practical way exists to calculate a particular data input that will result in a desired hash value. This one-way nature





makes the hash very difficult to forge. Message Digest 5 (MD5) is an example of a function intended for cryptographic hashing that is commonly used as a stock hash function.

When a function is used for error detection and correction, it focuses on distinguishing those cases in which data has been disturbed by a random process. One way that a hash function might be employed is as a checksum. Hash functions have a relatively small hash value that can be used to verify that a data file of any size has not been altered, making them valuable in network security implementations.

In terms of real-world application, perhaps an example is in order. One of the most common applications of hash functions is helping prove the authenticity of a message. When a hash function is employed in this manner, it produces a “fingerprint” of the message. This fingerprint is the hash code or message digest, which is the output of the hash function. This is used to create a digital signature for the message to ensure its authenticity. We will explore the concept of digital signatures as well as various hash functions in later sections.

#### **Cryptographic Hash Functions**

Put simply, a cryptographic hash function takes an input and returns a fixed-length string, which is called the hash value or hash sum. These hash functions, as mentioned in the preceding section, may be used for a variety of purposes, including cryptography. A hash value, as complex as it may become, is, on the surface, simply a concise representation of a longer message or document from which it was derived. The output of the hash function, often called the message digest, is a sort of “digital fingerprint” of the larger document. Message integrity checks and digital signatures, used for various security applications such as authentication and message integrity, can be provided by cryptographic hash functions.

The way this works is that the hash function accepts a string of variable length, sometimes called a message, as an input and then produces a fixed-length string, called a message digest or digital fingerprint, as an output. A hash value, often called a “digest” or “checksum,” is a type of signature that represents the contents of a stream of data.

Put another way, a hash is akin to the seal on certain over-the-counter medications. A plastic film covers the top of the bottle. If it has been disturbed, it is evident that the medication has been tampered with. The hash serves this same function, only as a digital mechanism.

Two of the most widely used hash functions are MD5 and SHA-1 (Secure Hash Algorithm 1). However, in 2005 security flaws were identified in both of these common algorithms. Although these hash functions are indeed flawed, they are still widely used today. Why is this?

As cryptographic analysts seek to break hash functions, sometimes they are successful. However, this is often after many years and with the aid of computing technology outside



of what would be considered the norm. So even though these two hash functions have security flaws, it's very unlikely that an attacker, under normal and reasonable circumstances, could exploit them. Therefore, these two functions are still widely used for a variety of purposes today. We will explore both MD5 and SHA-1 and their application to networking technologies later in this chapter.

When we consider the security of a cryptographic hash function, it should behave as much as possible like a random function while still being efficiently computable and deterministic in nature.

If either of the following statements is computationally feasible, a cryptographic hash function is considered insecure:



- A previously unseen message that matches a given digest
- Two different messages having the same message digest, called a collision

If an attacker can discover either of these things, he may be able to exploit the vulnerability in the hash function. For instance, he might be able to substitute an unauthorized message for one that has been authorized.

With a truly secure cryptographic hash function, it should not be possible to find two messages whose message digests are substantially similar, let alone exactly the same. Likewise, with a secure cryptographic hash, an attacker should not be able to learn anything of value about a message from only its digest. However, if an attacker can compromise security and obtain the digest, this could prove valuable should that same message occur again.

#### Application of Cryptographic Hashes

Let's examine a cryptographic hash to better understand how it works.

Suppose Anthony presents Tom with a rather difficult math problem that he claims to have solved. Tom wants to try to solve the problem himself, but he also wants to be sure that Anthony is telling the truth about having solved it. Anthony writes down his solution and then appends a random nonce, computes its hash, and tells Tom this hash value. The nonce that Anthony uses in this case is a random or pseudorandom number that is used only once for the purposes of this communication. All Tom has is the hash value; Anthony keeps the solution and the nonce secret. Tom gets to work on the math problem. When Tom finally solves the problem, Anthony can prove that he solved the problem as well by telling Tom the nonce. This example employs what cryptographers call a simple commitment scheme. In the world of computer networking, "Tom" and "Anthony" might very well be two computer programs attempting to prove a message's authenticity.



Secure hashes often provide a mechanism to verify message integrity. Let's say that a file is to be sent across the Internet, and you are concerned that it might be intercepted and altered in transit. Using a secure cryptographic hash would allow you to determine whether any changes have been made to the file. For example, you could do this by comparing message digests calculated before, and after, transmission of the file over the public network.

One means to reliably identify a file is a message digest. For instance, the Git source code management system uses the `sha1sum` program to examine various types of content such as file content, directory trees, ancestry information, and the like to uniquely identify a file.

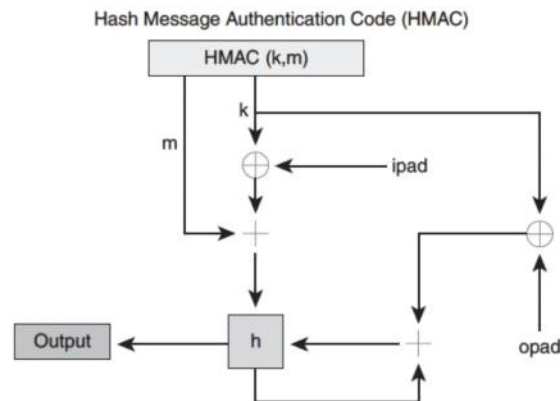
Another frequent use of cryptographic hashes is password verification. When we think in terms of passwords, we know that protecting them is of the utmost importance. That is why passwords generally are not stored in clear text but rather in a digest form. When a digest is used, a user is authenticated in the following manner: The user provides her username and password. The password she presents is hashed and then compared to the hash that has been stored. If a match is found, the user is granted access. This type of cryptographic hash generally is called one-way encryption.

SHA-1, MD5, and as RIPEMD-160 are some of the most widely used message digest algorithms. This is true even though in August 2004, researchers found weaknesses in a number of hash functions, including MD5, SHA-0, and RIPEMD. These findings also have called into question the long-term security of later algorithms derived from these hash functions—in particular, SHA-1, which is a strengthened version of SHA-0. As recently as August 2005, an attack against SHA-1 found collisions in  $2^{63}$  operations. An attack in February of that same year found collisions in about  $2^{69}$  hashing operations, rather than the  $2^{80}$  expected for a 160-bit hash function. Findings such as these call into question the security of these common cryptographic hashes. However, in terms of practical applications, these collisions do not warrant stopping the use of these extremely popular hash functions.

#### **HMAC Explained**

Keyed Hash-based Message Authentication Code (HMAC) in cryptographic terms is a type of message authentication code (MAC) calculated by using a cryptographic hash function along with a secret key. It may be used to simultaneously verify the data's integrity and the message's authenticity. An iterative cryptographic hash function such as MD5 or SHA-1 may be used to calculate the HMAC. When these are used, the resulting MAC algorithm is called HMAC-MD5 or HMAC-SHA-1, for instance. The cryptographic strength of the underlying hash function, along with the size and quality of the key and the size of the hash output length in bits, define the cryptographic strength of the HMAC. Figure 13-2 illustrates HMAC.

Figure 13-2 HMAC



$h$  = Cryptographic hash function.  
 $m$  = Message to be authenticated.  
 $k$  = Secret key padded with extra 0's (ipad/opad) to the block size of the hash function.  
 ipad = Inner padding.  
 opad = Outer padding.

Iterative hash functions, such as MD5 and SHA-1, break a message into blocks of a fixed size and then iterate over them with a compression function. For instance, MD5 and SHA-1 operate on 512-bit blocks. As mentioned, the size of the HMAC output is the same as that of the underlying hash function (128 or 160 bits in the case of MD5 and SHA-1), but you can truncate this if you want to. When the hash image is truncated, the security of the MAC is reduced.

In 1996 Mihir Bellare, Ran Canetti, and Hugo Krawczyk wrote about the construction and analysis of HMACs. These authors also wrote RFC 2104 in 1997 and FIPS PUB 198, which generalizes and standardizes the use of HMACs. Both the IPsec and TLS protocols use HMAC-SHA-1 and HMAC-MD5.

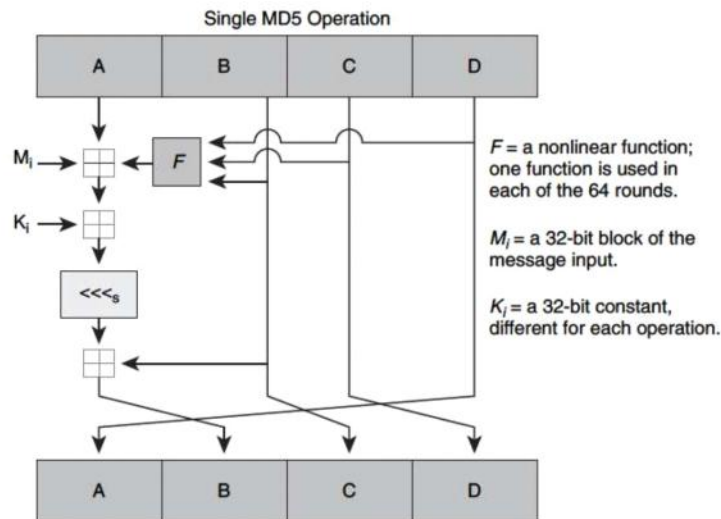
### MD5 Features and Functionality

Defined in RFC 1321, MD5 (Message Digest algorithm 5), with its 128-bit hash value, has been employed in a wide variety of security applications. It is also commonly used to check the integrity of files. An MD5 hash typically is expressed as a 32-character hexadecimal number.

Figure 13-3 shows a single MD5 operation. In practice, MD5 consists of 64 of these operations. These are grouped in four rounds of 16 operations. In this figure,  $F$  is a nonlinear function; one function is used in each round.  $M_i$  denotes a 32-bit block of the message input, and  $K_i$  denotes a 32-bit constant, which is different for each operation.

Key  
Topic



**Figure 13-3** MD5 Algorithm

Ronald Rivest designed MD5 in 1991 as a replacement for the earlier MD4 hash function. Five years later, in 1996, a flaw was found in the design of MD5. Although this flaw was not a fatal weakness, the cryptography community began recommending the use of other algorithms, such as SHA-1. Ironically, the widely used SHA-1 algorithm has since been shown to be vulnerable as well, as noted earlier. In 2004, researchers discovered more serious flaws in the algorithm, calling into question the use of the algorithm for certain security purposes.

#### Origins of MD5

Ronald Rivest of MIT created Message Digest as a series of message digest algorithms. MD5 was designed to be a secure replacement for its predecessor, MD4, when work demonstrated that MD4 was likely unsecure. Security of the MD5 algorithm was initially brought into question in 1993 when researchers found a pseudo-collision of the MD5 compression function. In other words, two different initialization vectors produced an identical digest.

In 1996, a true collision of the MD5 compression function was announced. Although this was not an attack on the full MD5 hash function, it was close enough for cryptographers to recommend switching to a replacement. Among the recommendations were WHIRLPOOL, SHA-1, and RIPEMD-160.

The hash, at only 128 bits, was small enough for cryptographers to fear that it would be vulnerable to a birthday attack. A birthday attack is a type of cryptographic attack that takes advantage of the mathematics behind the birthday paradox. The birthday paradox addresses



the probability that, in a set of randomly chosen people, two of them will have the same birthday. If the random group consists of 23 or more people, the probability that two of them will have the same birthday is greater than 50%. If the number of people in the pool grows to 57, the probability of a shared birthday is more than 99%. The probability approaches 100% as the number of individuals grows.

To test this theory, MD5CRK, a distributed project, was started in March 2004. The aim of the project was to demonstrate that MD5 is practically insecure by finding a collision using a birthday attack.

In less than six months, MD5CRK ended in August 2004 when collisions for the full MD5 were announced. The reported attack took only one hour on an IBM p690 cluster.

Further weaknesses were discovered in March 2005. A team of researchers constructed two x.509 certificates with different public keys and the same MD5 hash, a demonstrably practical collision. Within days, additional researchers announced an improved algorithm that could construct MD5 collisions in a few hours on a single notebook computer. Further hardship followed when, on March 18, 2006, an algorithm was published that could find a collision within one minute on a notebook computer using a method called tunneling.

#### **Vulnerabilities of MD5**

MD5 makes only a single pass over data. Because of this, if two prefixes with the same hash can be constructed, it is possible to add a common suffix to both to make the collision reasonably more possible.

Currently there exist collision-finding techniques that allow the preceding hash state to be specified arbitrarily. Therefore, a collision can be found for any desired prefix. This means that for any given string of characters X (for instance, a password), two colliding files can be determined that both begin with X.



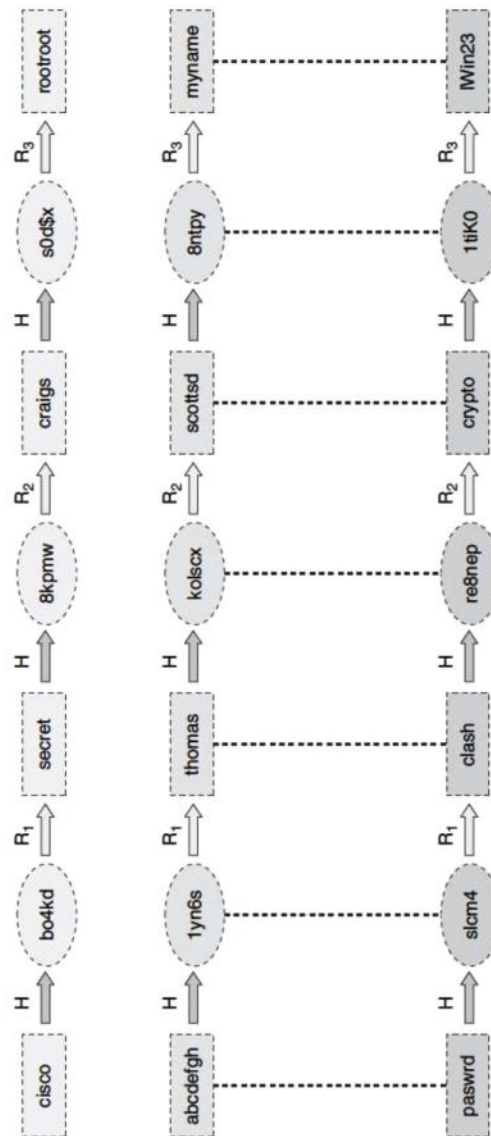
To generate these two colliding files, all that is needed is a template file, with a 128-byte block of data aligned on a 64-byte boundary, that can be changed freely by the collision-finding algorithm.

Figure 13-4 shows a rainbow table. Attackers can use rainbow tables to try to reverse hashes into strings.

Attackers can use MD5 rainbow tables, which are easily accessible online, to reverse many MD5 hashes into strings that collide with the original input. The general purpose of such attacks is password cracking. One means of defense is to combine passwords with a salt (a series of random bits added to the password) before the MD5 digest is generated. This combination makes rainbow tables much less useful.



**Figure 13-4** *Rainbow Table*



### Usage of MD5

One of the most common uses of MD5 digests is to provide some degree of assurance that a transferred file has arrived intact. An example of this is when a file server provides a precomputed MD5 checksum for a file. To ensure that the file has not been tampered with, the user can compare the checksum of the downloaded file to the one provided by the file server. Various UNIX-based operating systems include MD5 checksum utilities. For those working with Windows operating systems, this capability generally is provided by third-party applications.

Recent vulnerabilities that have been discovered with MD5 now make it easy to generate MD5 collisions. That being the case, it is possible for the person who created the file to create a second file with the same checksum, negating the protection against some forms of malicious tampering. At other times, the checksum might not be trustworthy, such as if it was obtained over the same channel as the downloaded file. In a case such as this, MD5 can only provide error checking, allowing it to recognize a corrupt or incomplete download. This is often the case when you download large files.

Another common usage of MD5 is to store passwords. To provide a defense against the vulnerabilities we have discussed, a salt can be added to the password before it is hashed. In fact, some implementations of this technique apply the hashing function multiple times to provide greater security.

### SHA-1 Features and Functionality

SHA-1 (Secure Hash Algorithm 1) is one of five cryptographic hash functions referred to as SHA hash functions. They were designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard. Much like the MD5 hash algorithm, SHA-1 computes a fixed-length digital representation (a *message digest*) from an input data sequence (the *message*) of any length.

A hash such as this is defined as “secure” when it is computationally infeasible to

- Find a message that corresponds to a given message digest
- Find two different messages that produce the same message digest

When working with a secure hash such as this, any change to a message should, with a very high degree of probability, result in a different message digest.

Figure 13-5 shows the SHA-1 hash and how it is used. We will discuss the application of SHA-1 in greater detail in an upcoming section.



**Figure 13-5** *SHA-1 Hash*

```
SHA1("The quick brown fox jumps over the lazy dog")
= 2fd4e1c6 7a2d28fc ed849ee1 bb76e739 1b93eb12
```

With SHA-1, a small change in the message will result in a completely different hash due to the avalanche effect. For example, changing *dog* to *log*:

```
SHA1("The quick brown fox jumps over the lazy log")
= de9f2c7f d25e1b3a fad3e85a 0bd17d9b 100db4b3
```

**Overview of SHA-1**

The SHA-1 hash produces a message digest that is 160 bits long, as opposed to 128 bits for MD5. A number of widely used security applications and protocols employ SHA-1, including TLS, SSL, PGP, SSH, S/MIME, and IPsec. SHA-1 has been positioned as the successor to MD5, which was one of the most widely used hash functions until SHA-1 was introduced.

Like its predecessor, MD5, researchers have attempted to validate the security of SHA-1. Although it has been somewhat compromised, no attacks have yet been reported on the SHA-2 variants, which are algorithmically similar to SHA-1.

To expand on these SHA-2 variants, NIST has published four additional hash functions in the SHA family. Each has a longer digest. Collectively they are known as SHA-2. Each of these individual variants is named after its digest length (in bits): SHA-224, SHA-256, SHA-384, and SHA-512. The final three were first published in 2001 in the draft Federal Information Processing Standard publication (FIPS PUB) 180-2. This publication (FIPS PUB 180-2) also includes SHA-1 and was released as an official standard in 2002. A change notice was published for FIPS PUB 180-2 in February 2004. This specified an additional variant, SHA-224. These variants have not yet received the degree of scrutiny from the cryptographic community that SHA-1 has, so their cryptographic security is not yet as well-established.

Currently efforts are under way within the cryptography community to develop improved alternative hashing algorithms. In one such effort, NIST is seeking to develop one or more additional hash algorithms through a public competition, similar to the development process used for the Advanced Encryption Standard (AES). It is the hope of NIST that a new standard will be developed and announced in 2012.



### Vulnerabilities of SHA-1

Some researchers working with SHA-1 have called into question its use in new cryptosystems. NIST has announced that it plans to phase out the use of SHA-1 by 2010 in favor of the SHA-2 variants. To better understand the vulnerabilities of SHA-1 and why NIST is searching for a replacement, you need to better understand the research that has been done into its weaknesses.

SHA-1 has fallen victim to various attacks that call into question its cryptographic strength. One such attack, published by researchers in early 2005, found collisions when working with a reduced version of SHA-1. In February of that same year, a team of researchers announced an attack that could find collisions in the full version of SHA-1, requiring fewer than  $2^{69}$  operations. To put this in perspective, a brute-force search attack would require  $2^{80}$  operations.

When asked how these attacks were possible, researchers pointed to the exploitation of two weaknesses:



- Weak file processing step
- Certain math operations in the first 20 rounds have unexpected security issues

In the world of research and academic cryptography, an attack that has less computational complexity than a brute-force search is considered a break. However, academic research is sometimes far removed from practical application. The vulnerabilities identified in SHA-1 do not necessarily mean that the attack can be practically exploited. That being said, if an attacker could harness a massive distributed Internet search, theorists believe that finding a collision for SHA-1 would be possible.

In a practical sense, the primary concern about attacks that have been launched against SHA-1 is that they might pave the way to more efficient attacks—ones that might be practically carried out by would-be attackers. It is because of this potential that cryptographers believe that a migration to stronger hashes would be prudent. That is one reason why NIST is working toward a new solution.

A collision attack, as described earlier, does not present the same kinds of risks that a preimage attack does. A preimage attack on a cryptographic hash represents an attempt to find a message with a specific hash value. This type of attack differs from a collision attack in that a fixed hash or message is the focus of the attack.

Current application of cryptographic hashes for such purposes as password storage and document signing is only minimally affected by a collision attack. For instance, where it is used to sign a document, an attacker could not simply fake a signature from an existing document. In a case such as this, the attacker would also have to fool the private key holder

into signing a preselected document. Where it has been applied for password usage, practical security threats are also less likely. An attacker would not be able to reverse password encryption to obtain a user's password to use elsewhere using a collision attack. Constructing a password that works for a given account requires a preimage attack, along with access to the hash of the original password (typically held in a *shadow* file).

#### Usage of SHA-1

SHA-1 and other SHA hash algorithms (SHA-224, SHA-256, SHA-384, and SHA-512) are secure hash algorithms required by law for use in certain U.S. government applications. This includes the use of SHA-1 within other cryptographic algorithms and protocols to protect sensitive yet unclassified information. Adoption and usage of SHA-1 by private and commercial organizations has been encouraged by FIPS PUB 180-1. Central to the publication of SHA was the Digital Signature Standard, in which it is incorporated. We will examine this in the section "Exploring the Digital Signature Standard."

## Using Digital Signatures

Much like written signatures, digital signatures may be used to authenticate an associated input. In the written sense, we might find a signature providing authentication on anything from a letter to a legal contract. In the digital sense, a digital signature's input is called a "message." These messages may be anything. They might be an e-mail or a legal contract, or it might even be a message sent in a more complicated cryptographic protocol.

These digital signatures are used to create public key infrastructure (PKI) schemes wherein a user's public key (whether for public key encryption, digital signatures, or another purpose) is linked to a user by a digital identity certificate issued by a certificate authority (CA). These PKI schemes seek to create an unbreakable bond between the user's information (such as name, address, and phone number) and the public key. This relationship allows the user to use these public keys as a form of electronic identification.

Let's look at an example. Figure 13-6 shows the process of using a digital signature to support a PKI scheme in which the user's public key is linked to the user through the digital certificate issued by the CA.



Here is how the process works:

1. The user's identity is bound to his public key through the CA. This is carried out through the binding and issuance process.
2. The actual binding of the certificate is done by the registration authority (RA).
3. After the binding, the user may use this certificate to represent himself and in the creation of messages.
4. The user may freely distribute his public key to those with whom he wants to communicate.
5. To provide authenticity of a message, the user may sign his message using his private key. Recipients may then validate the message's authenticity by applying the sender's public key.
6. To secure communications with the holder of the public/private key pair, the user can encrypt a message with the recipient's public key. This message then may be decrypted only through the use of the recipient's private key.

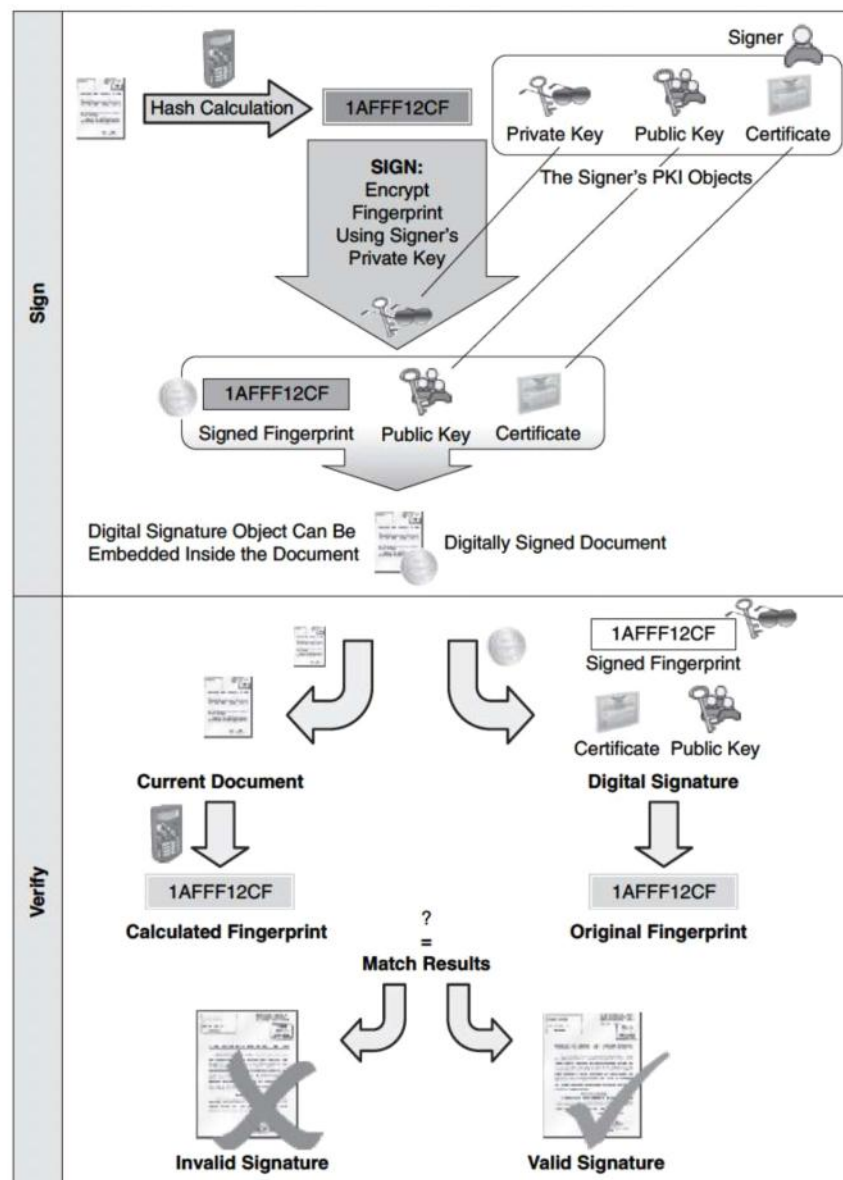
A digital signature (or digital signature scheme) is a form of asymmetric cryptography that is used to simulate the security characteristics of a written signature in digital form. Digital signature schemes typically use two algorithms that employ a pair of public and private keys. One algorithm is used for signing, which involves the user's secret or private key. The other is used to verify these signatures. This typically involves the use of the user's public key. The end result of this signature process is called the digital signature. It has widespread use in electronic security.

### **Understanding Digital Signatures**

To understand digital signatures, we need to begin by examining digital signature schemes and their commonalities. All digital signature schemes have a number of prior requirements. Figure 13-7 shows a digital signature.



Figure 13-7 Digital Signature

Key  
Topic

The first requirement is quality algorithms. As we have discussed, some of the available public key algorithms have been called into question with regard to security. Others are known to be insecure based on predictable attacks having been launched against them.

The second requirement is quality implementations. What this means is that even if you have a quality algorithm, if it is implemented incorrectly, it won't help you.

The third requirement is that the private key must remain secret. If this private key is compromised, an attacker can create an exact digital signature of anything he wants.

The fourth requirement is that the distribution of public keys has to be done in a manner that ensures that a public key belonging to a given user actually does belong to that user. Often this is done using a PKI. The public key user association is attested to by the operator of the PKI, the CA. In the case of "open" PKIs—ones in which anyone can request such an attestation—embodied in an identity certificate, the potential for mistaken attestation is not trivial. Unfortunately, commercial PKI providers have suffered a number of publicly known issues. Mistakes such as these could lead to falsely signed, and thus improperly attributed, documents. Maintaining a "closed" PKI system is more costly for organizations but less easily subverted, providing a stronger level of security for those who can take such steps.

Finally, beyond the PKI infrastructure and the steps that administrators must take to provide security, the fifth and final area of concern is users. The users of these PKI systems themselves (and their software) must take care to carry out the signature protocol properly to not compromise the signature.

All the conditions just listed must be met for a digital signature to reliably provide evidence of who sent the message, and therefore of his assent to its contents. Even legal measures cannot alter this reality.

Based on local laws, many countries accord a digital signature the same status as a traditional pen-and-paper signature with regard to its capacity to bind parties in a legal agreement such as a contract. Because of the legally binding nature of digital signatures in some parts of the world, it is generally best to use separate key pairs for encrypting and signing. Use of these key pairs allows an individual to engage in an encrypted conversation about matters that might be legally binding, such as the negotiation of a contract for employment. When the parties involved in the discussion reach an agreement, they can use their signing keys to "sign" the electronic document. At this point they are legally bound by the terms of a specific document. After this signing has taken place, the electronic document can be sent across an encrypted link to complete the transaction.

**Digital Signature Scheme**

Three algorithms generally make up a digital signature scheme:

- The key generation algorithm, which is used to randomly produce the key pair (public/private keys) used by the signer
- The signing algorithm, which, upon input of a message and a signing key, produces a signature
- The signature verifying algorithm, which, upon input of a message, a verifying key, and a signature, is used to either accept or reject the signature

**Authentication and Integrity**

One of the more practical uses of a digital signature in today's networks is for authentication and integrity checking. An example of this is the verification of authenticity in a message sent across a network.

Many times messages sent across the network include information about the entity sending the message. However, the authenticity of that information might be called into question. Digital signatures give us a mechanism to authenticate the source of such messages. With digital signatures, it is assumed that the ownership of a digital signature secret key is known to only a specific user. That being the case, a valid signature reflects that the message was sent by that specific user. The need for high levels of confidence in these matters is underscored by their use in financial matters, such as conducting credit card transactions or accessing bank account information.

For example, if a user accesses her bank account online and requests a transfer of funds between accounts, the bank must be certain of the authenticity of this request. If the bank has any question about this, transferring the funds would be a mistake.

In scenarios such as bank transactions, in addition to being assured of the user's authenticity, both the user (the sender) and her bank (the recipient of the message) may require confidence that the message has not been altered during transmission. Although the use of proper encryption can hide the contents of a message in transit, it may be possible to alter an encrypted message without understanding it. Certain nonmalleable encryption algorithms prevent this, but others do not. However, if a message is digitally signed, any change to the message while it is in transit invalidates the signature and alerts the parties that the message has been corrupted.

**Examining RSA Signatures**

The first algorithm found to be suitable for signing as well as encryption was RSA. RSA is an algorithm for public key cryptography. It represents what many believe to be one of the



first great advances in public key cryptography. Today RSA is widely used in a number of electronic commerce protocols. Thanks to its long cryptographic keys and the use of up-to-date implementations, it is believed to be secure.

#### Exploring the History of RSA

The algorithm that would become RSA was first described in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman of MIT. A quick look at the first letters of their surnames tells you where the term RSA came from. Today RSA encryption is widely known and widely used for a variety of security needs.

Interestingly, a British mathematician named Clifford Cocks, who worked for the UK intelligence agency GCHQ, described an equivalent system in a top-secret internal document in 1973. However, because of the cost of computing resources to implement it, it was never deployed. This discovery was not revealed until 1997 because of its top-secret classification. Rivest, Shamir, and Adleman devised RSA independently of this initial work. In 1983 MIT was granted U.S. patent 4405829 for a “cryptographic communications system and method” that used the RSA algorithm. The patent expired on September 21, 2000.

#### Understanding How RSA Works



RSA uses a public key/private key combination. The public key in this pair can be known by anyone and can be distributed widely without issue to encrypt messages. After a message has been encrypted with a specific public key, it may be decrypted only through the use of the matching private key, which is privately held and is never distributed publicly.

If you are ready for some math, let's look at how the keys for the RSA algorithm are generated.

We begin by selecting two distinct large random prime numbers  $p$  and  $q$ . Next we compute  $n=pq$ , where  $n$  is used as a modulus for both the public and private keys. Then we compute the totient using the formula  $\phi(n)=(p-1)(q-1)$ .

Next we choose an integer  $e$  such that  $1 < e < \phi(n)$ , and  $e$  and  $\phi(n)$  share no factors other than 1 (coprime). The public key exponent that is released is  $e$ . You compute  $d$  to satisfy the congruence relation as  $de \equiv 1 \pmod{\phi(n)}$ . The private key exponent is  $d$ .

The public key consists of the modulus  $n$  and the public (or encryption) exponent  $e$ . The private key consists of the modulus  $n$  and the private (or decryption) exponent  $d$  that must always remain secret.



### Encrypting and Decrypting Messages with RSA

Based on our earlier discussion, you know that RSA employs a combination of a public and private key to both encrypt and decrypt messages. This means that a user who wants to send and encrypt a message with RSA would transmit her public key ( $n$  and  $e$ ) to another user while keeping her private key to herself. When the recipient of the public key wants to communicate with the sender, he uses the public key he received from the sender. The way this works is that the user (through an application) first turns her message ( $m$ ) into a number where  $m < n$  by using an agreed-upon reversible protocol called a padding scheme. Next, the ciphertext  $c$  is computed such that  $c = m^e \bmod n$ . Finally, the user transmits this ciphertext to the holder of the private key—the only one who can decrypt this message.

Now that you have a sense of how encryption works, let's take a look at the decryption process. From the preceding example, the message recipient (holder of the private key) now must recover the message ( $m$ ) from the ciphertext by using his private key exponent  $d$ . He does this by using the computation  $m = c^d \bmod n$ . Given that the user has the message ( $m$ ), she may now recover the original message.

### Signing Messages with RSA

In addition to encrypting and decrypting messages, RSA can be used to sign messages. To continue our example of two individuals exchanging a message, let's suppose that one of the individuals uses the other's public key to encrypt and send a message. Although the message is encrypted, the sender may not be who we think she is.

In other words, because the public key has been widely distributed, you have no way of verifying that the message is in fact from the individual who claims to have sent it. That's because anyone could use this key to encrypt and send back a message to the recipient, who holds the private key. To be sure of the sender's identity, RSA may also be used to digitally sign the message as well as encrypt it. This ensures both the privacy of the message contents and the validity of its origin.

Let's take a look at the digital signing process with RSA. We can begin with a user who wants to send a signed message to a recipient:



1. The user who wants to send a signed message first must produce a hash value for the message itself and then raise it to the power of  $d \bmod n$ . (This same process is followed when a message is decrypted.)
2. The hash value is attached to the message as a digital signature.
3. When the message is received, the recipient must raise the signature to a power of  $e \bmod n$ , just like when the message contents are encrypted.
4. The resulting hash value is compared with the message's actual hash value to make sure they match.

5. If the two hash values are identical, the recipient can be assured that the author of the message does in fact have a secret key, so the message has not been tampered with.

#### Vulnerabilities of RSA

RSA's strength and security are based on two mathematical problems: the problem of factoring large numbers and the RSA problem. Cryptographers believe that complete decryption of an RSA ciphertext is not feasible based on the assumption that both of these problems are extremely difficult. In other words, currently there exists no efficient algorithm for solving them. Partial decryption is another matter. To provide security against this, it may be necessary to add a secure padding scheme. Even with these factors, a number of specific attacks still focus on RSA. The most common of these are listed in Table 13-2.



**Table 13-2** *RSA Attack Vulnerabilities*

Attack	Description
Timing attack	<p>In 1995 an attack against RSA was described wherein if the attacker knew a user's hardware in enough detail, and he could measure the decryption times for several known ciphertexts, he could deduce the decryption key quickly. This same attack could then also be applied against the RSA signature scheme as well.</p> <p>One way to defend against this form of attack is to make sure that a consistent amount of time is required for the decryption operation of each ciphertext. Although this would work, it may not be worth the performance degradation that would result. Most RSA implementations use an alternative approach known as blinding.</p> <p>In this approach, the multiplicative property of RSA is used. The result of applying RSA blinding is that the decryption time is no longer correlated to the value of the input ciphertext, so the timing attack fails.</p>
Adaptive chosen ciphertext attack	<p>The first practical adaptive chosen ciphertext attack against an RSA-encrypted message was described in 1995. This attack used the targeted flaws in the PKCS #1 scheme, which was used in concert with RSA. This attack focused on RSA implementations of the Secure Socket Layer protocol and was used to recover session keys. Because of the success of this attack, it is now recommended that RSA be used with other, more secure padding schemes, such as Optimal Asymmetric Encryption Padding. Additionally, RSA Laboratories has released updated versions of PKCS #1 that are not vulnerable to this form of attack.</p>
Branch prediction analysis (BPA) attack	<p>A number of processors use a branch predictor to determine whether a conditional branch in a program's instruction flow is likely to be taken. Generally speaking, these types of processors also implement simultaneous multithreading (SMT). A branch prediction analysis attack uses a spy process to statistically discover the private key when it is processed by these processors.</p>

**Exploring the Digital Signature Standard**

NIST created DSS, specified in FIPS 186 [1], and adopted it in 1993. Since its adoption, the standard has been revised in 1996 and in 2000, with a degree of expansion. DSS employs the Digital Signature Algorithm (DSA), which was proposed for use in the standard by NIST.

**Using the DSA Algorithm**

The DSS outlines the use of the DSA by a signer to generate a digital signature to be applied to data and by a recipient of the data to verify the authenticity of the signature. To create the digital signature, you need both a public key and a private key. The private key is used to generate the signature, and the public key is used to verify the signature. For both signature generation and verification, the data, which is called a message, is reduced through the use of SHA.

If an individual does not know the private key of the message signer, he cannot generate the signer's correct signature. This prevents the digital forgery of these signatures. The signature then can be verified by the message recipient or anyone holding the signer's public key. Of course, for this system to be useful, we need a mechanism for associating public and private key pairs to the corresponding users. In other words, we must bind a user's identity and public key. The binding of this association may be certified by a mutually trusted third party.

We see the application of this in secure e-commerce. In these instances, a certifying authority signs the credentials containing a user's (or company's) public key and identity to form a certificate. The DSS does not go into detail about systems for certifying credentials and distributing certificates, because they are beyond the scope of the standard.

## Exam Preparation Tasks

### Review All the Key Topics



Review the most important topics from this chapter, denoted with the Key Topic icon. Table 13-3 lists these key topics and the page where each is found.

**Table 13-3** *Key Topics for Chapter 13*

Key Topic Element	Description	Page Number
Figure 13-1	Hashing example	466
Explanation	Application of hash functions	467
List	Vulnerability of a hash function	469
Example	Application of a cryptographic hash	469
Figure 13-2	HMAC	471
Figure 13-3	An MD5 algorithm	472
Explanation	Collisions	473
Figure 13-4	An MD5 rainbow table	474
List	Components of a secure hash function	475
Figure 13-5	An SHA-1 hash	476
List	SHA-1 weaknesses	477
Figure 13-6	PKI	479
Figure 13-7	A digital signature	481
List	A digital signature scheme	483
Explanation	How RSA works	484
Explanation	The digital signing process with RSA	485
Table 13-2	RSA attack vulnerabilities	486



## Complete the Tables and Lists from Memory

Print a copy of Appendix D, “Memory Tables,” (found on the CD) or at least the section for this chapter, and complete the tables and lists from memory. Appendix E, “Memory Tables Answer Key,” also on the CD, includes completed tables and lists so that you can check your work.

## Definition of Key Terms

Define the following key terms from this chapter, and check your answers in the glossary:

authentication, authorization, certificate authority (CA), checksum, ciphertext, collision, cryptography, cryptographic hash, digital signature, Digital Signature Algorithm (DSA), HMAC, message, Message Digest 5 (MD5), National Institute of Standards and Technology (NIST), private key, public key, Public Key Infrastructure (PKI), rainbow table, RSA, salt, SHA-1