Shannon's Characteristics of "Good" Ciphers

In 1949, Claude Shannon [SHA49] proposed several characteristics that identify a good cipher.

1. The amount of secrecy needed should determine the amount of labor appropriate for the encryption and decryption.

Principle 1 is a reiteration of the principle of timeliness from Chapter 1 and of the earlier observation that even a simple cipher may be strong enough to deter the casual interceptor or to hold off any interceptor for a short time.

2. The set of keys and the enciphering algorithm should be free from complexity.

This principle implies that we should restrict neither the choice of keys nor the types of plaintext on which the algorithm can work. For instance, an algorithm that works only on plaintext having an equal number of As and Es is useless. Similarly, it would be difficult to select keys such that the sum of the values of the letters of the key is a prime number. Restrictions such as these make the use of the encipherment prohibitively complex. If the process is too complex, it will not be used. Furthermore, the key must be transmitted, stored, and remembered, so it must be short.

3. The implementation of the process should be as simple as possible.

Principle 3 was formulated with hand implementation in mind: A complicated algorithm is prone to error or likely to be forgotten. With the development and popularity of digital computers, algorithms far too complex for hand implementation became feasible. Still, the issue of complexity is important. People will avoid an encryption algorithm whose implementation process severely hinders message transmission, thereby undermining security. And a complex algorithm is more likely to be programmed incorrectly.

4. Errors in ciphering should not propagate and cause corruption of further information in the message.

Principle 4 acknowledges that humans make errors in their use of enciphering algorithms. One error early in the process should not throw off the entire remaining ciphertext. For example, dropping one letter in a columnar transposition throws off the entire remaining encipherment. Unless the receiver can guess where the letter was dropped, the remainder of the message will be unintelligible. By contrast, reading the wrong row or column for a polyalphabetic substitution affects only one character—remaining characters are unaffected.

5. The size of the enciphered text should be no larger than the text of the original message.

The idea behind principle 5 is that a ciphertext that expands dramatically in size cannot possibly carry more information than the plaintext, yet it gives the cryptanalyst more data from which to infer a pattern. Furthermore, a longer ciphertext implies more space for storage and more time to communicate.

SECURITY IN COMPUTING · PFLEEGER & PFLEEGER

These principles were developed before the ready availability of digital computers, even though Shannon was aware of computers and the computational power they represented. Thus, some of the concerns he expressed about hand implementation are not really limitations on computer-based implementation. For example, a cipher's implementation on a computer need not be simple, as long as the time complexity of the implementation is tolerable.

Properties of "Trustworthy" Encryption Systems

Commercial users have several requirements that must be satisfied when they select an encryption algorithm. Thus, when we say that encryption is "commercial grade," we mean that it meets these constraints:

- *It is based on sound mathematics.* Good cryptographic algorithms are not just invented; they are derived from solid principles.
- *It has been analyzed by competent experts and found to be sound.* Even the best cryptographic experts can think of only so many possible attacks, and the developers may become too convinced of the strength of their own algorithm. Thus, a review by critical outside experts is essential.
- *It has stood the "test of time."* As a new algorithm gains popularity, people continue to review both its mathematical foundations and the way it builds upon those foundations. Although a long period of successful use and analysis is not a guarantee of a good algorithm, the flaws in many algorithms are discovered relatively soon after their release.

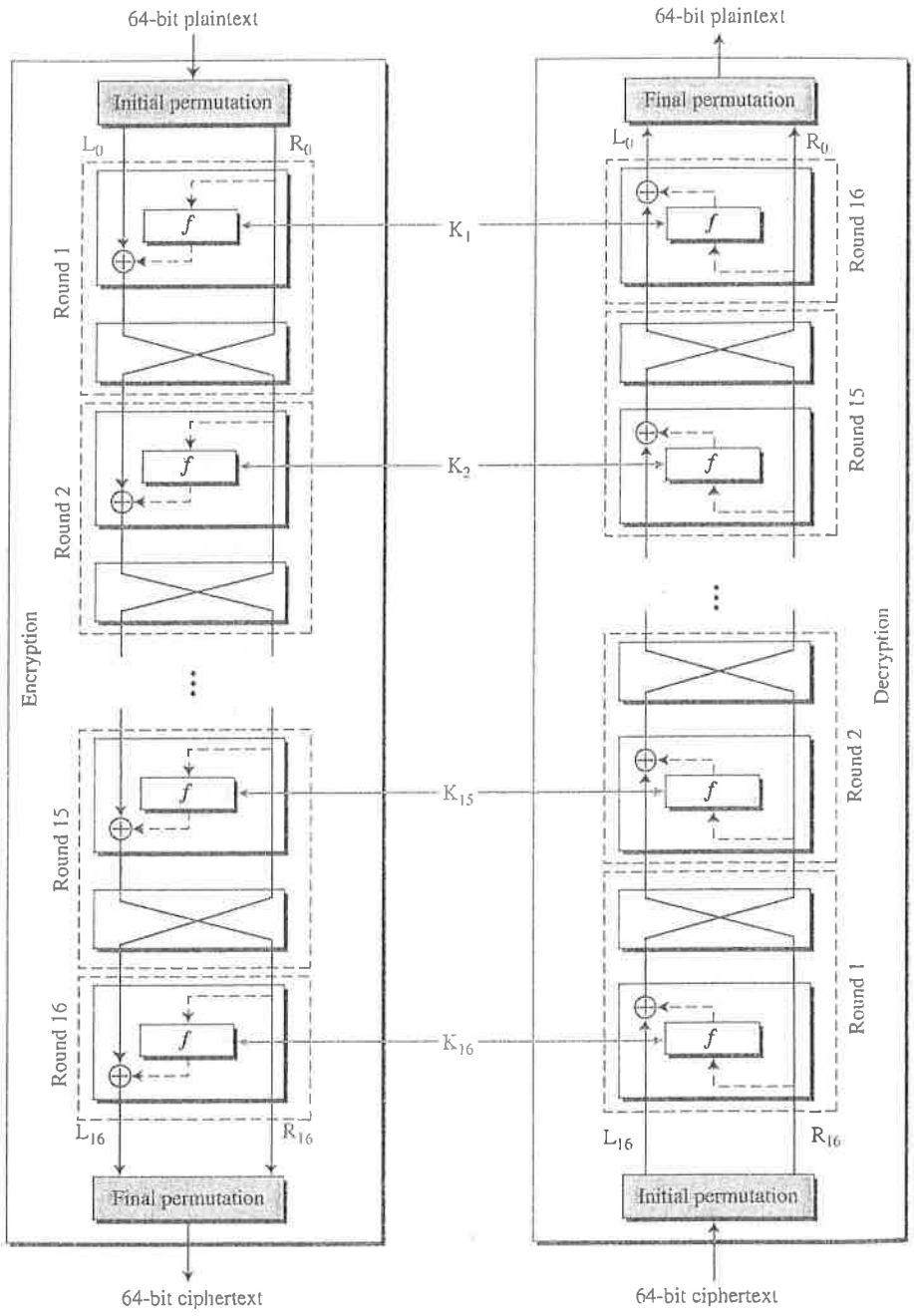
Three algorithms are popular in the commercial world: DES (data encryption standard), RSA (Rivest–Shamir–Adelman, named after the inventors), and AES (advanced encryption standard). The DES and RSA algorithms (as well as others) meet our criteria for commercial-grade encryption; AES, which is quite new, meets the first two and is starting to achieve widespread adoption.

Symmetric and Asymmetric Encryption Systems

Recall that the two basic kinds of encryptions are symmetric (also called "secret key") and asymmetric (also called "public key"). Symmetric algorithms use one key, which works for both encryption and decryption. Usually, the decryption algorithm is closely related to the encryption one. (For example, the Caesar cipher with a shift of 3 uses the encryption algorithm "substitute the character three letters later in the alphabet" with the decryption "substitute the character three letters earlier in the alphabet.")

The symmetric systems provide a two-way channel to their users: A and B share a secret key, and they can both encrypt information to send to the other as well as decrypt information from the other. As long as the key remains secret, the system also provides **authentication,** proof that a message received was not fabricated by someone other than the declared sender. Authenticity is ensured because only the legitimate sender can produce a message that will decrypt properly with the shared key.

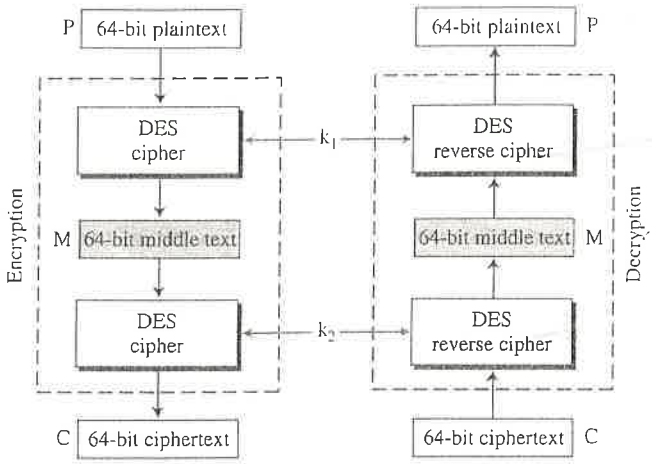
Figure 6.9 *DES cipher and reverse cipher for the first approach*



Meet-in-the-Middle Attack

At first glance, it looks like double DES increases the number of tests for key search from 2^{56} (in single DES) to 2^{112} (in double DES). However, using a known-plaintext attack called **meet-in-the-middle attack** proves that double DES improves this vulnerability slightly (to 2^{57} tests), but not tremendously (to 2^{112}). Figure 6.14 shows the diagram for the double DES. Alice uses two keys, k_1 and k_2, to decrypt plaintext P into ciphertext C; Bob uses ciphertext C and two keys, k_2 and k_1, to recover P.

Figure 6.14 *Meet-in-the-middle attack for double DES*



The point is that the middle text, the text created by the first encryption or first decryption, M, should be the same for encryption and decryption to work. In other words, we have two relationships:

$$M = E_{k_1}(P) \qquad \text{and} \qquad M = D_{k_2}(C)$$

Assume that Eve has intercepted a previous pair P and C (known-plaintext attack). Based on the first relationship mentioned above, Eve encrypts P using all possible values (2^{56}) of k_1 and records all values obtained for M. Based on the second relationship mentioned above, Eve decrypts C using all possible values (2^{56}) of k_2 and records all values obtained for M. Eve creates two tables sorted by M values. She then compares the values for M until she finds those pairs of k_1 and k_2 for which the value of M is the same in both tables as shown in Figure 6.15. Note that there must be at least one pair because she is doing exhaustive search on the combination of two keys.

1. If there is only one match, Eve has found the two keys (k_1 and k_2). If there is more than one candidate, Eve moves to the next step.
2. She takes another intercepted plaintext-ciphertext pair and uses each of the candidate key pairs to see if she can get the ciphertext from the plaintext. If she finds more than one candidate key pair, she repeats step 2 until she finally finds a unique pair.

It has been proved that after applying the second step to a few intercepted plaintext-ciphertext pairs, the keys are found. This means that instead of using 2^{112} key-search tests, Eve uses 2^{56} key-search tests two times (with some more tests if more than a single candidate is found in the first step). In other words, moving from single DES to double DES, we have increased the strength from 2^{56} to 2^{57} (not to 2^{112} as it is believed superficially).

Cryptography and Network Security

Behrouz A. Forouzan

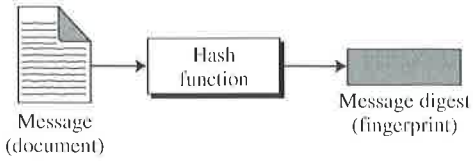
Document and Fingerprint

One way to preserve the integrity of a document is through the use of a *fingerprint*. If Alice needs to be sure that the contents of her document will not be changed, she can put her fingerprint at the bottom of the document. Eve cannot modify the contents of this document or create a false document because she cannot forge Alice's fingerprint. To ensure that the document has not been changed, Alice's fingerprint on the document can be compared to Alice's fingerprint on file. If they are not the same, the document is not from Alice.

Message and Message Digest

The electronic equivalent of the document and fingerprint pair is the *message* and *digest* pair. To preserve the integrity of a message, the message is passed through an algorithm called a **cryptographic hash function.** The function creates a compressed image of the message that can be used like a fingerprint. Figure 11.1 shows the message, cryptographic hash function, and **message digest.**

Figure 11.1 *Message and digest*



Difference

The two pairs (document/fingerprint) and (message/message digest) are similar, with some differences. The document and fingerprint are physically linked together. The message and message digest can be unlinked (or sent) separately, and, most importantly, the message digest needs to be safe from change.

The message digest needs to be safe from change.

Checking Integrity

To check the integrity of a message, or document, we run the cryptographic hash function again and compare the new message digest with the previous one. If both are the same, we are sure that the original message has not been changed. Figure 11.2 shows the idea.

Cryptographic Hash Function Criteria

A cryptographic hash function must satisfy three criteria: **preimage resistance, second preimage resistance,** and **collision resistance,** as shown in Figure 11.3.

Figure 11.2 *Checking integrity*

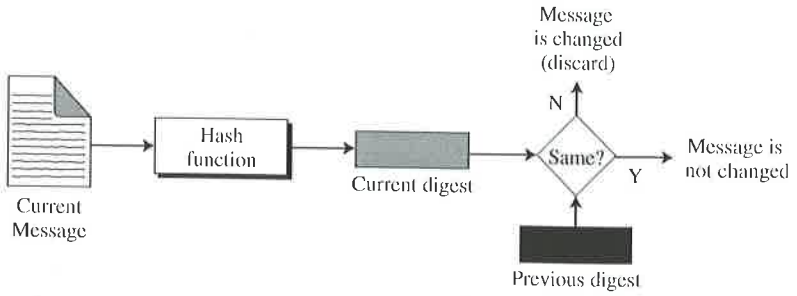
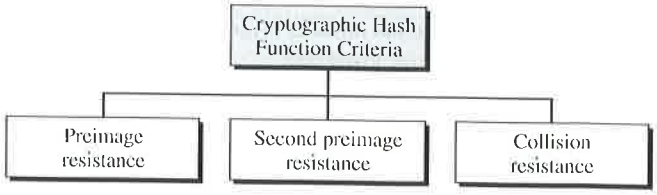


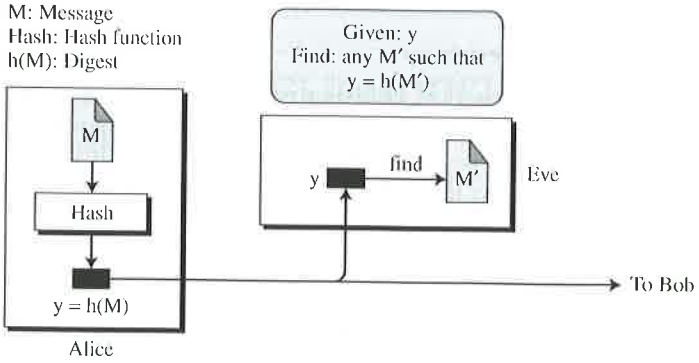
Figure 11.3 *Criteria of a cryptographic hash function*



Preimage Resistance

A cryptographic hash function must be preimage resistant. Given a hash function h and y = h(M), it must be extremely difficult for Eve to find any message, M′, such that y = h(M′). Figure 11.4 shows the idea.

Figure 11.4 *Preimage*



If the hash function is not preimage resistant, Eve can intercept the digest h(M) and create a message M′. Eve can then send M′ to Bob pretending it is M.

Preimage Attack

Given: y = h(M) **Find: M′ such that y = h(M′)**

Example 11.1

Can we use a conventional lossless compression method such as StuffIt as a cryptographic hash function?

Solution

We cannot. A lossless compression method creates a compressed message that is reversible. You can uncompress the compressed message to get the original one.

Example 11.2

Can we use a checksum function as a cryptographic hash function?

Solution

We cannot. A checksum function is not preimage resistant, Eve may find several messages whose checksum matches the given one.

Second Preimage Resistance

The second criterion, **second preimage resistance,** ensures that a message cannot easily be forged. If Alice creates a message and a digest and sends both to Bob, this criterion ensures that Eve cannot easily create another message that hashes to the exact same digest. In other words, given a specific message and its digest, it is impossible (or at least very difficult) to create another message with the same digest. Figure 11.5 shows the idea.

Eve intercepts (has access to) a message M and its digest h(M). She creates another message $M' \neq M$, but $h(M) = h(M')$. Eve sends the M′ and h(M′) to Bob. Eve has forged the message.

Second Preimage Attack

Given: M and h(M) **Find: $M' \neq M$ such that $h(M) = h(M')$**

Collision Resistance

The third criterion, **collision resistance,** ensures that Eve cannot find two messages that hash to the same digest. Here the adversary can create two messages (out of scratch) and hashed to the same digest. We will see later how Eve can benefit from this weakness in the hash function. For the moment, suppose two different wills can be created that hash to the same digest. When the time comes for the execution of the will, the second (forged) will is presented to the heirs. Because the digest matches both wills, the substitution is undetected. Figure 11.6 shows the idea. We will see later that this type of attack is much easier to launch than the two previous kinds. In other words, we need particularly be sure that a hash function is collision resistant.

Figure 11.5 *Second preimage*

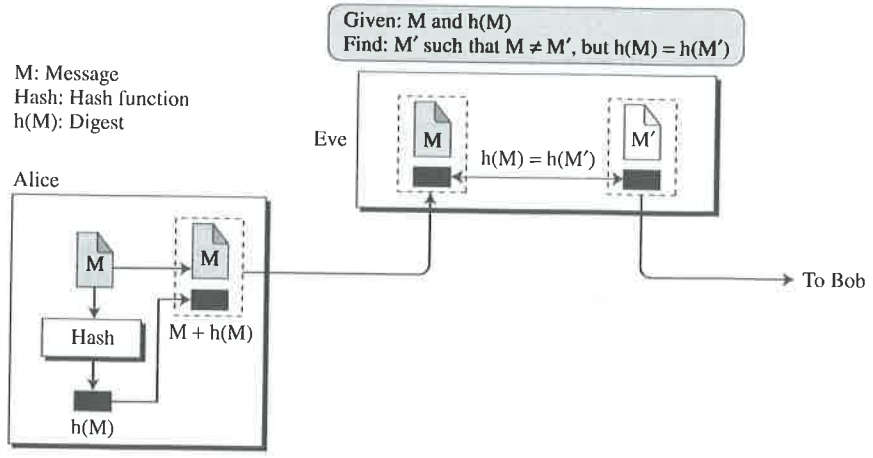
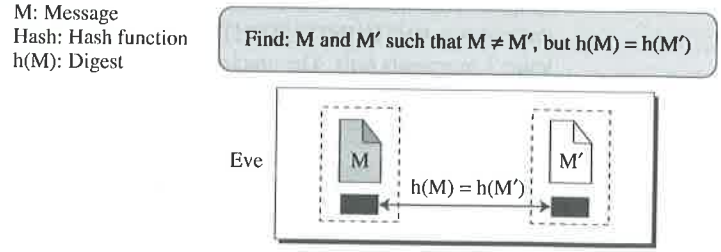


Figure 11.6 *Collision*



Collision Attack

Given: none **Find: M′ ≠ M such that h(M) = h(M′)**

11.2 RANDOM ORACLE MODEL

The **Random Oracle Model,** which was introduced in 1993 by Bellare and Rogaway, is an ideal mathematical model for a hash function. A function based on this model behaves as follows:

1. When a new message of any length is given, the oracle creates and gives a fixed-length message digest that is a random string of 0s and 1s. The oracle records the message and the message digest.
2. When a message is given for which a digest exists, the oracle simply gives the digest in the record.

ediate values). The "||"

$H_i = H_{i-1} \oplus G_i$

ay problem (in general

day problem (in general

ay problem (in general

day problem (in general

CHAPTER 12

Cryptographic Hash Functions

Objectives

This chapter has several objectives:

- To introduce general ideas behind cryptographic hash functions
- To discuss the Merkle-Damgard scheme as the basis for iterated hash functions
- To distinguish between two categories of hash functions: those with a compression function made from scratch and those with a block cipher as the compression function
- To discuss the structure of SHA-512 as an example of a cryptographic hash function with a compression function made from scratch
- To discuss the structure of Whirlpool as an example of a cryptographic hash function with a block cipher as the compression function

12.1 INTRODUCTION

As discussed in Chapter 11, a cryptographic hash function takes a message of arbitrary length and creates a message digest of fixed length. The ultimate goal of this chapter is to discuss the details of the two most promising cryptographic hash algorithms—*SHA-512* and *Whirlpool*. However, we first need to discuss some general ideas that may be applied to any cryptographic hash function.

Iterated Hash Function

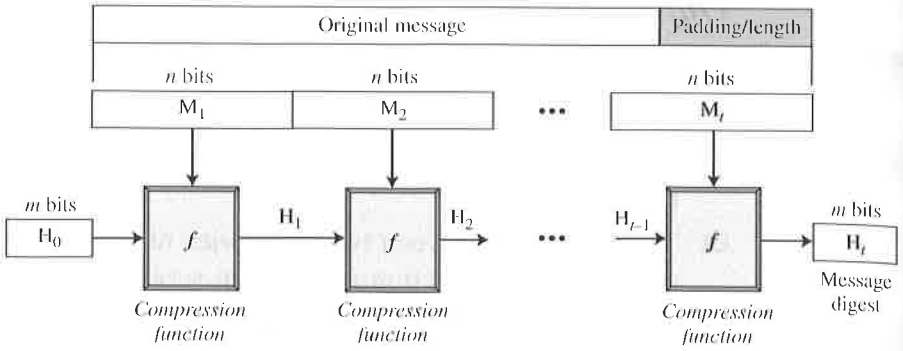
All cryptographic hash functions need to create a fixed-size digest out of a variable-size message. Creating such a function is best accomplished using iteration. Instead of using a hash function with variable-size input, a function with fixed-size input is created and is used a necessary number of times. The fixed-size input function is referred to as a

compression function. It compresses an n-bit string to create an m-bit string where n is normally greater than m. The scheme is referred to as an **iterated cryptographic hash function.**

Merkle-Damgard Scheme

The **Merkle-Damgard scheme** is an iterated hash function that is collision resistant if the compression function is collision resistant. This can be proved, but the proof is left as an exercise. The scheme is shown in Figure 12.1.

Figure 12.1 *Merkle-Damgard scheme*



The scheme uses the following steps:

1. The message length and padding are appended to the message to create an augmented message that can be evenly divided into blocks of n bits, where n is the size of the block to be processed by the compression function.
2. The message is then considered as t blocks, each of n bits. We call each block M_1, M_2,..., M_t. We call the digest created at t iterations H_1, H_2,..., H_t.
3. Before starting the iteration, the digest H_0 is set to a fixed value, normally called IV (initial value or initial vector).
4. The compression function at each iteration operates on H_{i-1} and M_i to create a new H_i. In other words, we have $H_i = f(H_{i-1}, M_i)$, where f is the compression function.
5. H_t is the cryptographic hash function of the original message, that is, h(M).

If the compression function in the Merkle-Damgard scheme is collision resistant, the hash function is also collision resistant.

Two Groups of Compression Functions

The Merkle-Damgard scheme is the basis for many cryptographic hash functions today. The only thing we need to do is design a compression function that is collision resistant

and insert it in the Merkle-Damgard scheme. There is a tendency to use two different approaches in designing a hash function. In the first approach, the compression function is made from scratch: it is particularly designed for this purpose. In the second approach, a symmetric-key block cipher serves as a compression function.

Hash Functions Made from Scratch

A set of cryptographic hash functions uses compression functions that are made from scratch. These compression functions are specifically designed for the purposes they serve.

Message Digest (MD) Several hash algorithms were designed by Ron Rivest. These are referred to as **MD2, MD4,** and **MD5,** where MD stands for Message Digest. The last version, MD5, is a strengthened version of MD4 that divides the message into blocks of 512 bits and creates a 128-bit digest. It turned out that a message digest of size 128 bits is too small to resist collision attack.

Secure Hash Algorithm (SHA) The **Secure Hash Algorithm (SHA)** is a standard that was developed by the National Institute of Standards and Technology (NIST) and published as a Federal Information Processing standard (FIP 180). It is sometimes referred to as **Secure Hash Standard (SHS).** The standard is mostly based on MD5. The standard was revised in 1995 under FIP 180-1, which includes **SHA-1**. It was revised later under FIP 180-2, which defines four new versions: **SHA-224, SHA-256, SHA-384,** and **SHA-512.** Table 12.1 lists some of the characteristics of these versions.

Table 12.1 *Characteristics of Secure Hash Algorithms (SHAs)*

Characteristics	*SHA-1*	*SHA-224*	*SHA-256*	*SHA-384*	*SHA-512*
Maximum Message size	$2^{64} - 1$	$2^{64} - 1$	$2^{64} - 1$	$2^{128} - 1$	$2^{128} - 1$
Block size	512	512	512	1024	1024
Message digest size	160	224	256	384	512
Number of rounds	80	64	64	80	80
Word size	32	32	32	64	64

All of these versions have the same structure. SHA-512 is discussed in detail later in this chapter.

Other Algorithms **RACE Integrity Primitives Evaluation Message Digest (RIPMED)** has several versions. **RIPEMD-160** is a hash algorithm with a 160-bit message digest. RIPEMD-160 uses the same structure as MD5 but uses two parallel lines of execution. **HAVAL** is a variable-length hashing algorithm with a message digest of size 128, 160, 192, 224, and 256.The block size is 1024 bits.

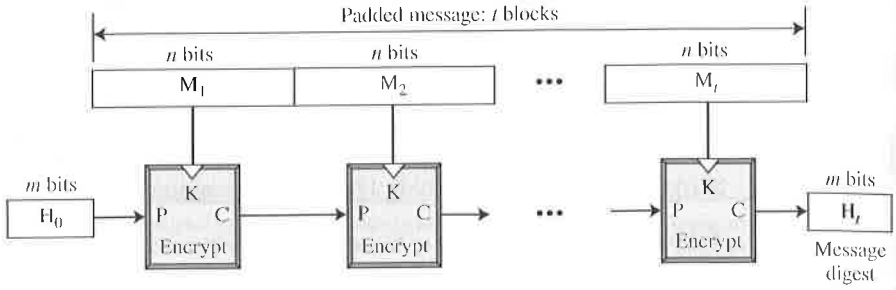
Hash Functions Based on Block Ciphers

An iterated cryptographic hash function can use a symmetric-key block cipher as a compression function. The whole idea is that there are several secure symmetric-key block ciphers, such as triple DES or AES, that can be used to make a one-way function instead of creating a new compression function. The block cipher in this case only

performs encryption. Several schemes have been proposed. We later describe one of the most promising, *Whirlpool*.

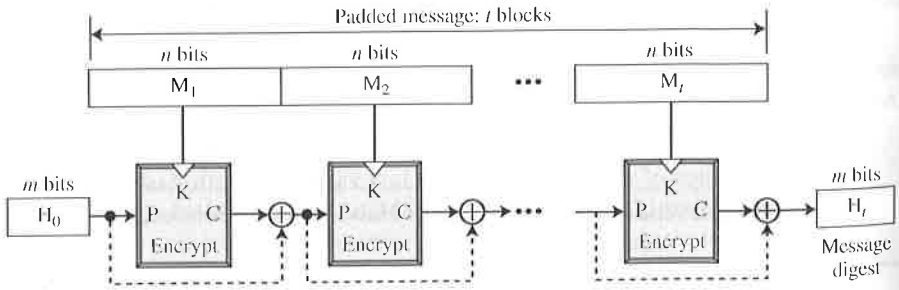
Rabin Scheme The iterated hash function proposed by Rabin is very simple. The **Rabin scheme** is based on the Merkle-Damgard scheme. The compression function is replaced by any encrypting cipher. The message block is used as the key; the previously created digest is used as the plaintext. The ciphertext is the new message digest. Note that the size of the digest is the size of data block cipher in the underlying cryptosystem. For example, if DES is used as the block cipher, the size of the digest is only 64 bits. Although the scheme is very simple, it is subject to a meet-in-the-middle attack discussed in Chapter 6, because the adversary can use the decryption algorithm of the cryptosystem. Figure 12.2 shows the Rabin scheme.

Figure 12.2 *Rabin scheme*



Davies-Meyer Scheme The **Davies-Meyer scheme** is basically the same as the Rabin scheme except that it uses forward feed to protect against meet-in-the-middle attack. Figure 12.3 shows the Davies-Meyer scheme.

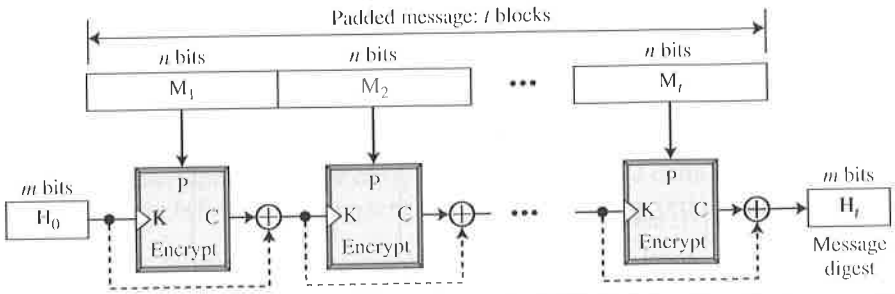
Figure 12.3 *Davies-Meyer scheme*



Matyas-Meyer-Oseas Scheme The **Matyas-Meyer-Oseas scheme** is a dual version of the Davies-Meyer scheme: the message block is used as the key to the cryptosystem. The scheme can be used if the data block and the cipher key are the same size. For

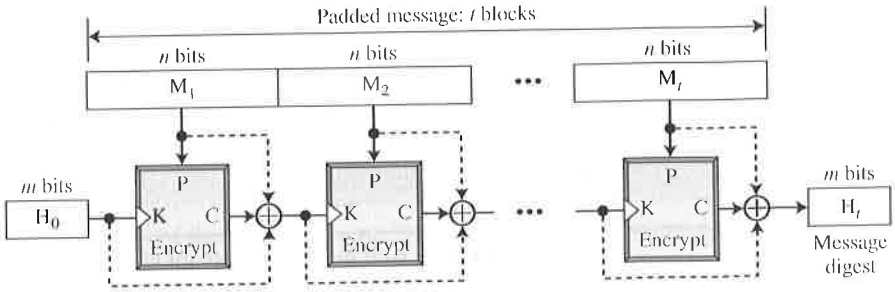
example, AES is a good candidate for this purpose. Figure 12.4 shows the Matyas-Meyer-Oseas scheme.

Figure 12.4 *Matyas-Meyer-Oseas scheme*



Miyaguchi-Preneel Scheme The **Miyaguchi-Preneel scheme** is an extended version of Matyas-Meyer-Oseas. To make the algorithm stronger against attack, the plaintext, the cipher key, and the ciphertext are all exclusive-ored together to create the new digest. This is the scheme used by the Whirlpool hash function. Figure 12.5 shows the Miyaguchi-Preneel scheme.

Figure 12.5 *Miyaguchi-Preneel scheme*



12.2 SHA-512

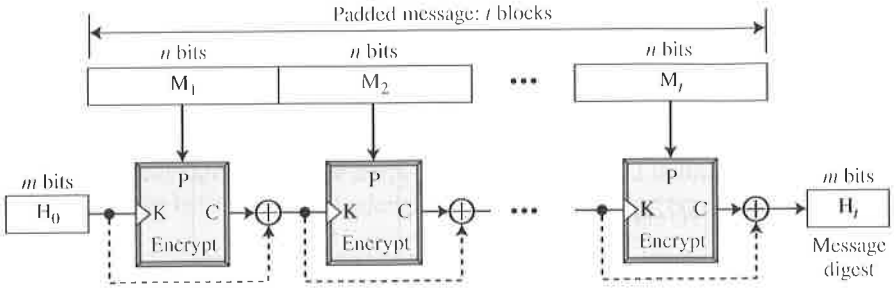
SHA-512 is the version of SHA with a 512-bit message digest. This version, like the others in the SHA family of algorithms, is based on the Merkle-Damgard scheme. We have chosen this particular version for discussion because it is the latest version, it has a more complex structure than the others, and its message digest is the longest. Once the structure of this version is understood, it should not be difficult to understand the structures of the other versions. For characteristics of SHA-512 see Table 12.1.

Introduction

SHA-512 creates a digest of 512 bits from a multiple-block message. Each block is 1024 bits in length, as shown in Figure 12.6.

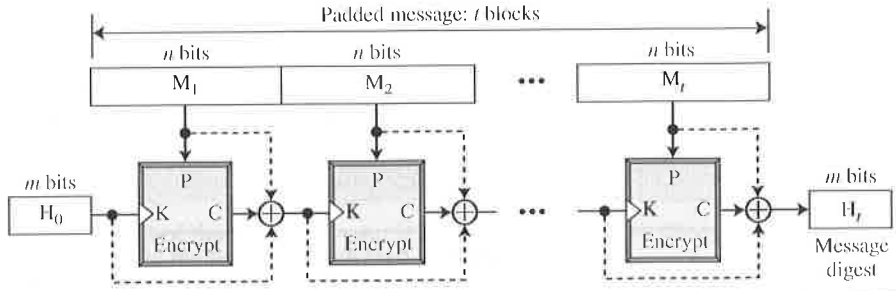
example, AES is a good candidate for this purpose. Figure 12.4 shows the Matyas-Meyer-Oseas scheme.

Figure 12.4 *Matyas-Meyer-Oseas scheme*



Miyaguchi-Preneel Scheme The **Miyaguchi-Preneel scheme** is an extended version of Matyas-Meyer-Oseas. To make the algorithm stronger against attack, the plaintext, the cipher key, and the ciphertext are all exclusive-ored together to create the new digest. This is the scheme used by the Whirlpool hash function. Figure 12.5 shows the Miyaguchi-Preneel scheme.

Figure 12.5 *Miyaguchi-Preneel scheme*



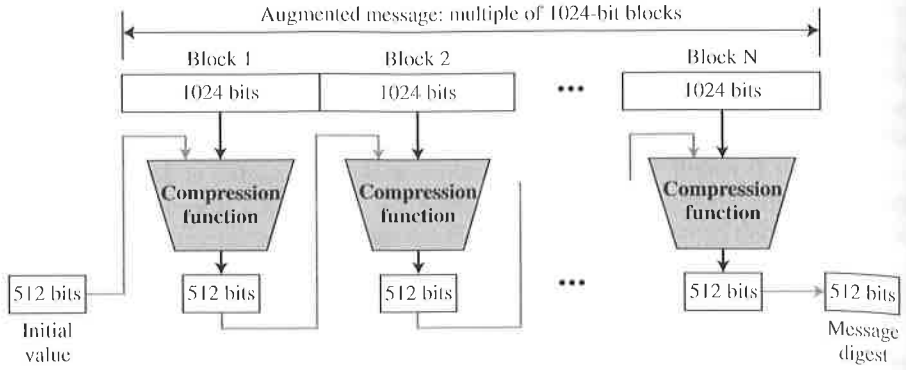
12.2 SHA-512

SHA-512 is the version of SHA with a 512-bit message digest. This version, like the others in the SHA family of algorithms, is based on the Merkle-Damgard scheme. We have chosen this particular version for discussion because it is the latest version, it has a more complex structure than the others, and its message digest is the longest. Once the structure of this version is understood, it should not be difficult to understand the structures of the other versions. For characteristics of SHA-512 see Table 12.1.

Introduction

SHA-512 creates a digest of 512 bits from a multiple-block message. Each block is 1024 bits in length, as shown in Figure 12.6.

Figure 12.6 *Message digest creation SHA-512*



The digest is initialized to a predetermined value of 512 bits. The algorithm mixes this initial value with the first block of the message to create the first intermediate message digest of 512 bits. This digest is then mixed with the second block to create the second intermediate digest. Finally, the $(N - 1)$th digest is mixed with the Nth block to create the Nth digest. When the last block is processed, the resulting digest is the message digest for the entire message.

Message Preparation

SHA-512 insists that the length of the original message be less than 2^{128} bits. This means that if the length of a message is equal to or greater than 2^{128}, it will not be processed by SHA-512. This is not usually a problem because 2^{128} bits is probably larger than the total storage capacity of any system.

SHA-512 creates a 512-bit message digest out of a message less than 2^{128}.

Example 12.1

This example shows that the message length limitation of SHA-512 is not a serious problem. Suppose we need to send a message that is 2^{128} bits in length. How long does it take for a communications network with a data rate of 2^{64} bits per second to send this message?

Solution

A communications network that can send 2^{64} bits per second is not yet available. Even if it were, it would take many years to send this message. This tells us that we do not need to worry about the SHA-512 message length restriction.

Example 12.2

This example also concerns the message length in SHA-512. How many pages are occupied by a message of 2^{128} bits?

This principle requires that processes should be confined to as small a protection domain as possible.

EXAMPLE: A mail server accepts mail from the Internet and copies the messages into a spool directory; a local server will complete delivery. The mail server needs the rights to access the appropriate network port, to create files in the spool directory, and to alter those files (so it can copy the message into the file, rewrite the delivery address if needed, and add the appropriate "Received" lines). It should surrender the right to access the file as soon as it has finished writing the file into the spool directory, because it does not need to access that file again. The server should not be able to access any user's files, or any files other than its own configuration files.

14.2.1.1 Principle of Least Authority

Closely related to the principle of least privilege is the principle of least authority [1349]. The two are often treated as meaning the same. However, some authors make a distinction between "permission" and "authority." They treat permission as determining what actions a process can take on objects directly, and authority as determining that effects a process may have on an object, either directly (as with permission) or indirectly through its interactions with other processes or subsystems.

Miller and Shapiro [1349] give a good example from the Take-Grant Protection Model. In that model, the rights would represent actions that subjects could take over objects, and so represent permissions. But the *de facto* rules of that model, which govern information transfer, show how information can flow from a subject to an object that is not directly connected to the subject. Hence the subject does not have permission to write information into the object, but it does have permission to pass the information to a second subject, and that subject can write the information into the object.[1]

Definition 14–2. The *principle of least authority* states that a subject should be given only the authority that it needs in order to complete its task.

If one reads the principle of least privilege as speaking to *permissions*, then this principle is somewhat different. But if it speaks to *authority*, the two are the same.

14.2.2 Principle of Fail-Safe Defaults

This principle restricts how privileges are initialized when a subject or object is created.

[1] This is the *find* rule described by Bishop and Snyder [213, 233].

Here, the sharing of the Internet with the attackers' sites caused the attack to succeed. The appropriate countermeasure would be to restrict the attackers' access to the segment of the Internet connected to the website. Techniques for doing this include proxy servers such as the Purdue SYN intermediary [1695] or traffic throttling (see Section 7.4, "Availability and Network Flooding"). The former targets suspect connections; the latter reduces the load on the relevant segment of the network indiscriminately.

Minimizing the number of shared mechanisms also reduces the scope of an attack that compromises such a mechanism. If all versions of an operating system use the same program, then compromising that single program enables attackers to compromise any system of that type. But if the systems each use a slightly different version of the program, then compromise becomes more difficult.

EXAMPLE: Attack tools assume an underlying structure or configuration of a system or program. In order to invalidate this assumption, researchers have studied how to inject artificial diversity effectively into programs and systems. Then the attack tools will not work properly.

Object code obfuscation tools scramble the flow of execution and the placement of data in memory. For example, many attacks target the return address for function calls, which is stored on a stack and thus in a predictable location. Adding a layer of indirection requires changing the function call and return sequence. Then an attempt to overwrite the return address will change the index into the table instead. By appropriately constraining that value and obscuring how the actual return addresses are stored, the attacker will be unlikely to guess the actual location of the return address, defeating this class of attacks [381]. Other techniques randomize the order of variables and functions in memory or introduce random gaps between formerly contiguous areas of storage, and locations of memory regions. This renders ineffective attack tools that rely on the memory layout of the program [193].

14.2.8 Principle of Least Astonishment

This principle recognizes the human element in computer security.

> **Definition 14–9.** The *principle of least astonishment* states that security mechanisms should be designed so that users understand the reason that the mechanism works the way it does and that using the mechanism is simple.

This principle requires security mechanisms to use a model that the target audience (users and system administrators, typically) can easily understand. If the audience's mental model is too different than that used by the designers and implementers, then their confusion may undermine the security mechanisms.

Thus, configuring and executing a program should be as easy and as intuitive as possible, and any output should be clear, direct, and useful. If security-related software is too complicated to configure, system administrators may unintentionally set up the software in a nonsecure manner. Similarly, security-related user programs must be easy to use and must output understandable messages. If a user is changing a password, and the proposed password is rejected, the password changing program should state why it was rejected rather than giving a cryptic error message. If a configuration file has an incorrect parameter, the error message should describe the proper parameter.

EXAMPLE: The *ssh* program [131, 2058] allows a user to set up a public key mechanism for enciphering communications between systems. The installation and configuration mechanisms for the UNIX version allow one to arrange that the public key be stored locally without any password protection. In this case, one need not supply a password to connect to the remote system, but will still obtain the enciphered connection. This mechanism satisfies the principle of least astonishment.

On the other hand, security requires that the messages impart no unnecessary information.

EXAMPLE: When a user supplies the wrong password during login, the system should reject the attempt with a message stating that the login failed. If it were to say that the password was incorrect, the user would know that the account name was legitimate. If the "user" were really an unauthorized attacker, she would then know the name of an account for which she could try to guess a password.

Balancing the needs of security and the mental models of users requires that the designers and implementers take into account the environment in which the security mechanisms are used.

EXAMPLE: A mainframe system allows users to place passwords on files. Accessing the files requires that the program supply the password. Although this mechanism violates the principle as stated, it is considered sufficiently minimal to be acceptable. On an interactive system, where the pattern of file accesses is more frequent and more transient, this requirement would be too great a burden to be acceptable.

14.2.8.1 Psychological Acceptability

The principle of least astonishment is similar to one of Saltzer's and Schroeder's original principles, the *principle of psychological acceptability*. That principle stated that that security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present. The difference between that principle and the principle of least astonishment is that the former expressed an ideal, whereas the latter recognizes that security mechanisms may

Shannon's Characteristics of "Good" Ciphers

In 1949, Claude Shannon [SHA49] proposed several characteristics that identify a good cipher.

1. The amount of secrecy needed should determine the amount of labor appropriate for the encryption and decryption.

Principle 1 is a reiteration of the principle of timeliness from Chapter 1 and of the earlier observation that even a simple cipher may be strong enough to deter the casual interceptor or to hold off any interceptor for a short time.

2. The set of keys and the enciphering algorithm should be free from complexity.

This principle implies that we should restrict neither the choice of keys nor the types of plaintext on which the algorithm can work. For instance, an algorithm that works only on plaintext having an equal number of As and Es is useless. Similarly, it would be difficult to select keys such that the sum of the values of the letters of the key is a prime number. Restrictions such as these make the use of the encipherment prohibitively complex. If the process is too complex, it will not be used. Furthermore, the key must be transmitted, stored, and remembered, so it must be short.

3. The implementation of the process should be as simple as possible.

Principle 3 was formulated with hand implementation in mind: A complicated algorithm is prone to error or likely to be forgotten. With the development and popularity of digital computers, algorithms far too complex for hand implementation became feasible. Still, the issue of complexity is important. People will avoid an encryption algorithm whose implementation process severely hinders message transmission, thereby undermining security. And a complex algorithm is more likely to be programmed incorrectly.

4. Errors in ciphering should not propagate and cause corruption of further information in the message.

Principle 4 acknowledges that humans make errors in their use of enciphering algorithms. One error early in the process should not throw off the entire remaining ciphertext. For example, dropping one letter in a columnar transposition throws off the entire remaining encipherment. Unless the receiver can guess where the letter was dropped, the remainder of the message will be unintelligible. By contrast, reading the wrong row or column for a polyalphabetic substitution affects only one character—remaining characters are unaffected.

5. The size of the enciphered text should be no larger than the text of the original message.

The idea behind principle 5 is that a ciphertext that expands dramatically in size cannot possibly carry more information than the plaintext, yet it gives the cryptanalyst more data from which to infer a pattern. Furthermore, a longer ciphertext implies more space for storage and more time to communicate.

SECURITY IN COMPUTING · PFLEEGER & PFLEEGER

These principles were developed before the ready availability of digital computers, even though Shannon was aware of computers and the computational power they represented. Thus, some of the concerns he expressed about hand implementation are not really limitations on computer-based implementation. For example, a cipher's implementation on a computer need not be simple, as long as the time complexity of the implementation is tolerable.

Properties of "Trustworthy" Encryption Systems

Commercial users have several requirements that must be satisfied when they select an encryption algorithm. Thus, when we say that encryption is "commercial grade," we mean that it meets these constraints:

- *It is based on sound mathematics.* Good cryptographic algorithms are not just invented; they are derived from solid principles.
- *It has been analyzed by competent experts and found to be sound.* Even the best cryptographic experts can think of only so many possible attacks, and the developers may become too convinced of the strength of their own algorithm. Thus, a review by critical outside experts is essential.
- *It has stood the "test of time."* As a new algorithm gains popularity, people continue to review both its mathematical foundations and the way it builds upon those foundations. Although a long period of successful use and analysis is not a guarantee of a good algorithm, the flaws in many algorithms are discovered relatively soon after their release.

Three algorithms are popular in the commercial world: DES (data encryption standard), RSA (Rivest–Shamir–Adelman, named after the inventors), and AES (advanced encryption standard). The DES and RSA algorithms (as well as others) meet our criteria for commercial-grade encryption; AES, which is quite new, meets the first two and is starting to achieve widespread adoption.

Symmetric and Asymmetric Encryption Systems

Recall that the two basic kinds of encryptions are symmetric (also called "secret key") and asymmetric (also called "public key"). Symmetric algorithms use one key, which works for both encryption and decryption. Usually, the decryption algorithm is closely related to the encryption one. (For example, the Caesar cipher with a shift of 3 uses the encryption algorithm "substitute the character three letters later in the alphabet" with the decryption "substitute the character three letters earlier in the alphabet.")

The symmetric systems provide a two-way channel to their users: A and B share a secret key, and they can both encrypt information to send to the other as well as decrypt information from the other. As long as the key remains secret, the system also provides **authentication,** proof that a message received was not fabricated by someone other than the declared sender. Authenticity is ensured because only the legitimate sender can produce a message that will decrypt properly with the shared key.

PRENTICE HALL

Security in Computing

Fifth Edition

Charles P. Pfleeger
Shari Lawrence Pfleeger
Jonathan Margulies

TABLE 3-2 Types of Malicious Code

Code Type	Characteristics
Virus	Code that causes malicious behavior and propagates copies of itself to other programs
Trojan horse	Code that contains unexpected, undocumented, additional functionality
Worm	Code that propagates copies of itself through a network; impact is usually degraded performance
Rabbit	Code that replicates itself without limit to exhaust resources
Logic bomb	Code that triggers action when a predetermined condition occurs
Time bomb	Code that triggers action when a predetermined time occurs
Dropper	Transfer agent code only to drop other malicious code, such as virus or Trojan horse
Hostile mobile code agent	Code communicated semi-autonomously by programs transmitted through the web
Script attack, JavaScript, Active code attack	Malicious code communicated in JavaScript, ActiveX, or another scripting language, downloaded as part of displaying a web page
RAT (remote access Trojan)	Trojan horse that, once planted, gives access from remote location
Spyware	Program that intercepts and covertly communicates data on the user or the user's activity
Bot	Semi-autonomous agent, under control of a (usually remote) controller or "herder"; not necessarily malicious
Zombie	Code or entire computer under control of a (usually remote) program
Browser hijacker	Code that changes browser settings, disallows access to certain sites, or redirects browser to others
Rootkit	Code installed in "root" or most privileged section of operating system; hard to detect
Trapdoor or backdoor	Code feature that allows unauthorized access to a machine or program; bypasses normal access control and authentication
Tool or toolkit	Program containing a set of tests for vulnerabilities; not dangerous itself, but each successful test identifies a vulnerable host that can be attacked
Scareware	Not code; false warning of malicious code attack

2014, for example, there were eight adware attacks (ads offering useless or malicious programs for sale), and nine Trojan horses or Trojan horse transmitters in the top 20, and two exploit script attacks, which we also describe in this chapter. But the top attack type, comprising 81.73 percent of attacks, was malicious URLs, described in the next chapter. A different measure counts the number of pieces of malicious code Kaspersky products found on protected computers (that is, malware not blocked by Kaspersky's email and Internet activity screens). Among the top 20 types of malware were five

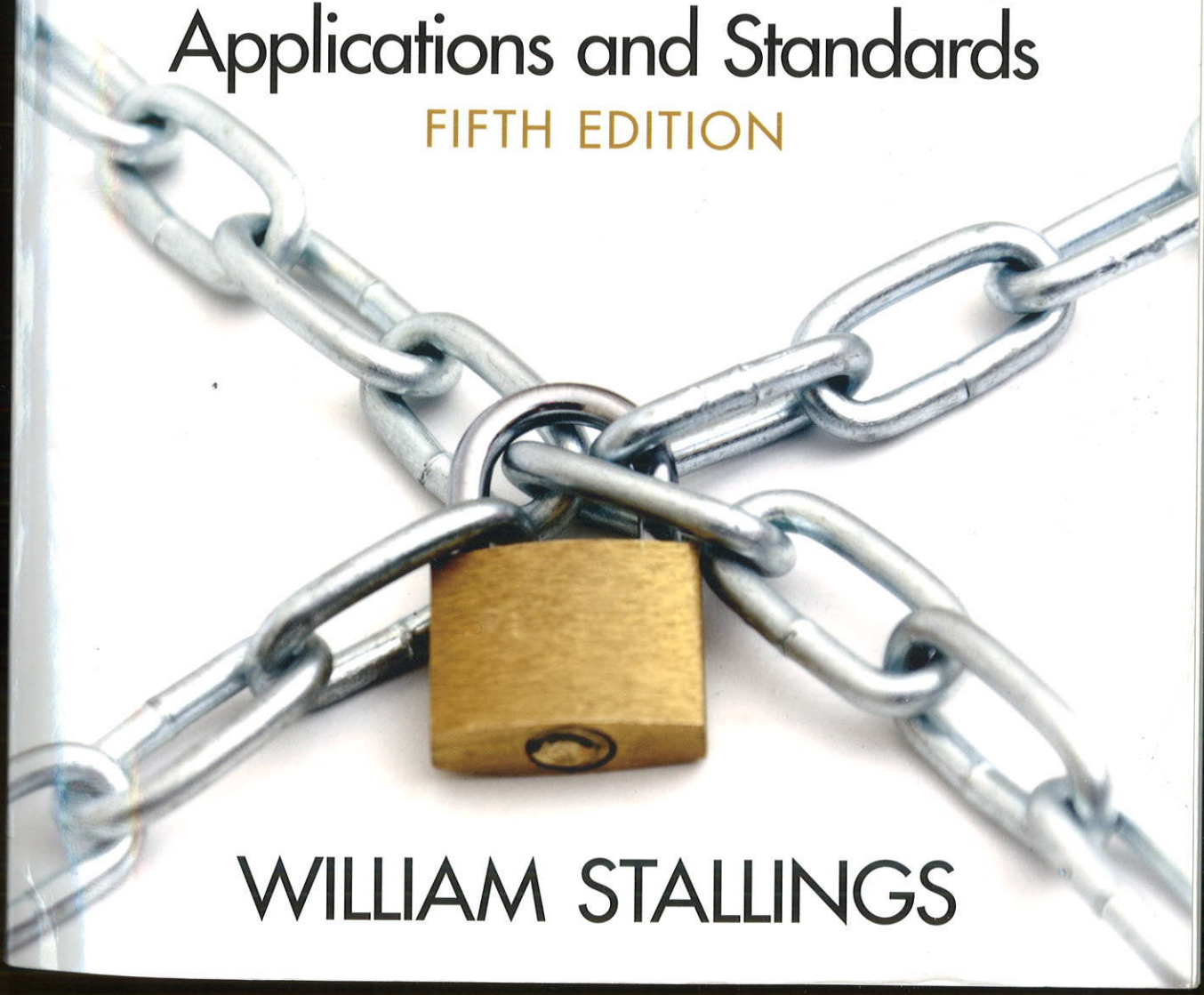
TABLE 3-3 Notable Malicious Code Infections

Year	Name	Characteristics
1982	Elk Cloner	First virus; targets Apple II computers
1985	Brain	First virus to attack IBM PC
1988	Morris worm	Allegedly accidental infection disabled large portion of the ARPANET, precursor to today's Internet
1989	Ghostballs	First multipartite (has more than one executable piece) virus
1990	Chameleon	First polymorphic (changes form to avoid detection) virus
1995	Concept	First virus spread via Microsoft Word document macro
1998	Back Orifice	Tool allows remote execution and monitoring of infected computer
1999	Melissa	Virus spreads through email address book
2000	IloveYou	Worm propagates by email containing malicious script. Retrieves victim's address book to expand infection. Estimated 50 million computers affected.
2000	Timofonica	First virus targeting mobile phones (through SMS text messaging)
2001	Code Red	Virus propagates from 1st to 20th of month, attacks whitehouse.gov web site from 20th to 28th, rests until end of month, and restarts at beginning of next month; resides only in memory, making it undetected by file-searching antivirus products
2001	Code Red II	Like Code Red, but also installing code to permit remote access to compromised machines
2001	Nimda	Exploits known vulnerabilities; reported to have spread through 2 million machines in a 24-hour period
2003	Slammer worm	Attacks SQL database servers; has unintended denial-of-service impact due to massive amount of traffic it generates
2003	SoBig worm	Propagates by sending itself to all email addresses it finds; can fake From: field; can retrieve stored passwords
2004	MyDoom worm	Mass-mailing worm with remote-access capability
2004	Bagle or Beagle worm	Gathers email addresses to be used for subsequent spam mailings; SoBig, MyDoom, and Bagle seemed to enter a war to determine who could capture the most email addresses
2008	Rustock.C	Spam bot and rootkit virus
2008	Conficker	Virus believed to have infected as many as 10 million machines; has gone through five major code versions
2010	Stuxnet	Worm attacks SCADA automated processing systems; zero-day attack
2011	Duqu	Believed to be variant on Stuxnet
2013	CryptoLocker	Ransomware Trojan that encrypts victim's data storage and demands a ransom for the decryption key

NETWORK SECURITY ESSENTIALS

Applications and Standards

FIFTH EDITION



WILLIAM STALLINGS

to initially boot the computer. If it is successful, the boot process fails, and the system is unusable until the BIOS chip is either reprogrammed or replaced.

The Stuxnet worm targets some specific industrial control system software as its key payload [CHEN11]. If control systems using certain Siemens industrial control software with a specific configuration of devices are infected, then the worm replaces the original control code with code that deliberately drives the controlled equipment outside its normal operating range, resulting in the failure of the attached equipment. The centrifuges used in the Iranian uranium enrichment program were strongly suspected as the target, with reports of much higher than normal failure rates observed in them over the period when this worm was active. As noted in our earlier discussion, this has raised concerns over the use of sophisticated targeted malware for industrial sabotage.

Logic Bomb

A key component of data-corrupting malware is the logic bomb. The logic bomb is code embedded in the malware that is set to "explode" when certain conditions are met. Examples of conditions that can be used as triggers for a logic bomb are the presence or absence of certain files or devices on the system, a particular day of the week or date, a particular version or configuration of some software, or a particular user running the application. Once triggered, a bomb may alter or delete data or entire files, cause a machine halt, or do some other damage. All of the examples we describe in this section include such code.

10.6 PAYLOAD—ATTACK AGENT—ZOMBIE, BOTS

The next category of payload we discuss is where the malware subverts the computational and network resources of the infected system for use by the attacker. Such a system is known as a bot (robot), zombie, or drone, and secretly takes over another Internet-attached computer and then uses that computer to launch or manage attacks that are difficult to trace to the bot's creator. The bot is typically planted on hundreds or thousands of computers belonging to unsuspecting third parties. The collection of bots often is capable of acting in a coordinated manner; such a collection is referred to as a **botnet**. This type of payload attacks the integrity and availability of the infected system.

Uses of Bots

[HONE05] lists the following uses of bots:

- **Distributed denial-of-service (DDoS) attacks:** A DDoS attack is an attack on a computer system or network that causes a loss of service to users. We examine DDoS attacks in Section 10.10.
- **Spamming:** With the help of a botnet and thousands of bots, an attacker is able to send massive amounts of bulk e-mail (spam).
- **Sniffing traffic:** Bots can also use a packet sniffer to watch for interesting clear-text data passing by a compromised machine. The sniffers are mostly used to retrieve sensitive information like usernames and passwords.

- **Keylogging:** If the compromised machine uses encrypted communication channels (e.g., HTTPS or POP3S), then just sniffing the network packets on the victim's computer is useless because the appropriate key to decrypt the packets is missing. But by using a keylogger, which captures keystrokes on the infected machine, an attacker can retrieve sensitive information.
- **Spreading new malware:** Botnets are used to spread new bots. This is very easy since all bots implement mechanisms to download and execute a file via HTTP or FTP. A botnet with 10,000 hosts that acts as the start base for a worm or mail virus allows very fast spreading and thus causes more harm.
- **Installing advertisement add-ons and browser helper objects (BHOs):** Botnets can also be used to gain financial advantages. This works by setting up a fake Web site with some advertisements: The operator of this Web site negotiates a deal with some hosting companies that pay for clicks on ads. With the help of a botnet, these clicks can be "automated" so that instantly a few thousand bots click on the pop-ups. This process can be further enhanced if the bot hijacks the start-page of a compromised machine so that the "clicks" are executed each time the victim uses the browser.
- **Attacking IRC chat networks:** Botnets are also used for attacks against Internet Relay Chat (IRC) networks. Popular among attackers is the so-called clone attack: In this kind of attack, the controller orders each bot to connect a large number of clones to the victim IRC network. The victim is flooded by service requests from thousands of bots or thousands of channel-joins by these cloned bots. In this way, the victim IRC network is brought down, similar to a DDoS attack.
- **Manipulating online polls/games:** Online polls/games are getting more and more attention and it is rather easy to manipulate them with botnets. Since every bot has a distinct IP address, every vote will have the same credibility as a vote cast by a real person. Online games can be manipulated in a similar way.

Remote Control Facility

The remote control facility is what distinguishes a bot from a worm. A worm propagates itself and activates itself, whereas a bot is controlled from some central facility, at least initially.

A typical means of implementing the remote control facility is on an IRC server. All bots join a specific channel on this server and treat incoming messages as commands. More recent botnets tend to avoid IRC mechanisms and use covert communication channels via protocols such as HTTP. Distributed control mechanisms, using peer-to-peer protocols, are also used, to avoid a single point of failure.

Once a communications path is established between a control module and the bots, the control module can activate the bots. In its simplest form, the control module simply issues command to the bot that causes the bot to execute routines that are already implemented in the bot. For greater flexibility, the control module can issue update commands that instruct the bots to download a file from some Internet location and execute it. The bot in this latter case becomes a more general-purpose tool that can be used for multiple attacks.

10.7 PAYLOAD—INFORMATION THEFT—KEYLOGGERS, PHISHING, SPYWARE

We now consider payloads where the malware gathers data stored on the infected system for use by the attacker. A common target is the user's login and password credentials to banking, gaming, and related sites, which the attacker then uses to impersonate the user to access these sites for gain. Less commonly, the payload may target documents or system configuration details for the purpose of reconnaissance or espionage. These attacks target the confidentiality of this information.

Credential Theft, Keyloggers, and Spyware

Typically, users send their login and password credentials to banking, gaming, and related sites over encrypted communication channels (e.g., HTTPS or POP3S), which protect them from capture by monitoring network packets. To bypass this, an attacker can install a **keylogger**, which captures keystrokes on the infected machine to allow an attacker to monitor this sensitive information. Since this would result in the attacker receiving a copy of all text entered on the compromised machine, keyloggers typically implement some form of filtering mechanism that only returns information close to desired keywords (e.g., "login" or "password" or "paypal.com").

In response to the use of keyloggers, some banking and other sites switched to using a graphical applet to enter critical information, such as passwords. Since these do not use text entered via the keyboard, traditional keyloggers do not capture this information. In response, attackers developed more general **spyware** payloads, which subvert the compromised machine to allow monitoring of a wide range of activity on the system. This may include monitoring the history and content of browsing activity, redirecting certain Web page requests to fake sites controlled by the attacker, dynamically modifying data exchanged between the browser and certain Web sites of interest. All of which can result in significant compromise of the user's personal information.

Phishing and Identity Theft

Another approach used to capture a user's login and password credentials is to include a URL in a spam e-mail that links to a fake Web site controlled by the attacker, but which mimics the login page of some banking, gaming, or similar site. This is normally included in some message suggesting that urgent action is required by the user to authenticate his or her account, to prevent it being locked. If the user is careless, and doesn't realize that he or she is being conned, then following the link and supplying the requested details will certainly result in the attackers exploiting the user's account using the captured credentials.

More generally, such a spam e-mail may direct a user to a fake Web site controlled by the attacker or to complete some enclosed form and return to an e-mail accessible to the attacker, which is used to gather a range of private, personal, information on the user. Given sufficient details, the attacker can then "assume" the user's identity for the purpose of obtaining credit or sensitive access to other

resources. This is known as a **phishing** attack, which exploits social engineering to leverage user's trust by masquerading as communications from a trusted source [GOLD10].

Such general spam e-mails are typically widely distributed to very large numbers of users, often via a botnet. While the content will not match appropriate trusted sources for a significant fraction of the recipients, the attackers rely on it reaching sufficient users of the named trusted source, a gullible portion of whom will respond, for it to be profitable.

A more dangerous variant of this is the **spear-phishing** attack. This again is an e-mail claiming to be from a trusted source. However, the recipients are carefully researched by the attacker, and each e-mail is carefully crafted to suit its recipient specifically, often quoting a range of information to convince him or her of its authenticity. This greatly increases the likelihood of the recipient responding as desired by the attacker.

Reconnaissance and Espionage

Credential theft and identity theft are special cases of a more general reconnaissance payload, which aims to obtain certain types of desired information and return this to the attacker. These special cases are certainly the most common; however other targets are known. Operation Aurora in 2009 used a Trojan to gain access to and potentially modify source code repositories at a range of high-tech, security, and defense contractor companies [SYMA11]. The Stuxnet worm discovered in 2010 included capture of hardware and software configuration details in order to determine whether it had compromised the specific desired target systems. Early versions of this worm returned this same information, which was then used to develop the attacks deployed in later versions [CHEN11].

10.8 PAYLOAD—STEALTHING—BACKDOORS, ROOTKITS

The final category of payload we discuss concerns techniques used by malware to hide its presence on the infected system and to provide covert access to that system. This type of payload also attacks the integrity of the infected system.

Backdoor

A **backdoor**, also known as a **trapdoor**, is a secret entry point into a program that allows someone who is aware of the backdoor to gain access without going through the usual security access procedures. The backdoor is code that recognizes some special sequence of input or is triggered by being run from a certain user ID or by an unlikely sequence of events.

A backdoor is usually implemented as a network service listening on some nonstandard port that the attacker can connect to and issue commands through to be run on the compromised system.

It is difficult to implement operating system controls for backdoors in applications. Security measures must focus on the program development and software update activities, and on programs that wish to offer a network service.

10.9 COUNTERMEASURES

Malware Countermeasure Approaches

[SOUP12a] lists four main elements of prevention: policy, awareness, vulnerability mitigation, and threat mitigation. Having a suitable policy to address malware prevention provides a basis for implementing appropriate preventative countermeasures.

One of the first countermeasures that should be employed is to ensure all systems are as current as possible, with all patches applied, in order to reduce the number of vulnerabilities that might be exploited on the system. The next is to set appropriate access controls on the applications and data stored on the system, to reduce the number of files that any user can access, and hence potentially infect or corrupt, as a result of them executing some malware code. These measures directly target the key propagation mechanisms used by worms, viruses, and some trojans.

The third common propagation mechanism, which targets users in a social engineering attack, can be countered using appropriate user awareness and training. This aims to equip users to be more aware of these attacks, and less likely to take actions that result in their compromise. [SOUP12a] provides examples of suitable awareness issues.

If prevention fails, then technical mechanisms can be used to support the following threat mitigation options:

- **Detection:** Once the infection has occurred, determine that it has occurred and locate the malware.
- **Identification:** Once detection has been achieved, identify the specific malware that has infected the system.
- **Removal:** Once the specific malware has been identified, remove all traces of malware virus from all infected systems so that it cannot spread further.

If detection succeeds but either identification or removal is not possible, then the alternative is to discard any infected or malicious files and reload a clean backup version. In the case of some particularly nasty infections, this may require a complete wipe of all storage, and rebuild of the infected system from known clean media.

To begin, let us consider some requirements for effective malware countermeasures:

- **Generality:** The approach taken should be able to handle a wide variety of attacks.
- **Timeliness:** The approach should respond quickly so as to limit the number of infected programs or systems and the consequent activity.
- **Resiliency:** The approach should be resistant to evasion techniques employed by attackers to hide the presence of their malware.
- **Minimal denial-of-service costs:** The approach should result in minimal reduction in capacity or service due to the actions of the countermeasure software, and should not significantly disrupt normal operation.

- **Transparency:** The countermeasure software and devices should not require modification to existing (legacy) OSs, application software, and hardware.
- **Global and local coverage:** The approach should be able to deal with attack sources both from outside and inside the enterprise network.

Achieving all these requirements often requires the use of multiple approaches.

Detection of the presence of malware can occur in a number of locations. It may occur on the infected system, where some host-based "antivirus" program is running, monitoring data imported into the system, and the execution and behavior of programs running on the system. Or, it may take place as part of the perimeter security mechanisms used in an organization's firewall and intrusion detection systems (IDSs). Lastly, detection may use distributed mechanisms that gather data from both host-based and perimeter sensors, potentially over a large number of networks and organizations, in order to obtain the largest scale view of the movement of malware.

Host-Based Scanners

The first location where antivirus software is used is on each end system. This gives the software the maximum access to information not only on the behavior of the malware as it interacts with the targeted system but also on the smallest overall view of malware activity. The use of antivirus software on personal computers is now widespread, in part caused by the explosive growth in malware volume and activity. Advances in virus and other malware technology, and in antivirus technology and other countermeasures, go hand in hand. Early malware used relatively simple and easily detected code, and hence could be identified and purged with relatively simple antivirus software packages. As the malware arms race has evolved, both the malware code and, necessarily, antivirus software have grown more complex and sophisticated.

[STEP93] identifies four generations of antivirus software:

- **First generation:** Simple scanners
- **Second generation:** Heuristic scanners
- **Third generation:** Activity traps
- **Fourth generation:** Full-featured protection

A **first-generation** scanner requires a malware signature to identify the malware. The signature may contain "wildcards" but matches essentially the same structure and bit pattern in all copies of the malware. Such signature-specific scanners are limited to the detection of known malware. Another type of first-generation scanner maintains a record of the length of programs and looks for changes in length as a result of virus infection.

A **second-generation** scanner does not rely on a specific signature. Rather, the scanner uses heuristic rules to search for probable malware instances. One class of such scanners looks for fragments of code that are often associated with malware. For example, a scanner may look for the beginning of an encryption loop used in a polymorphic virus and discover the encryption key. Once the key is discovered, the scanner can decrypt the malware to identify it, and then remove the infection and return the program to service.

Another second-generation approach is integrity checking. A checksum can be appended to each program. If malware alters or replaces some program without changing the checksum, then an integrity check will catch this change. To counter malware that is sophisticated enough to change the checksum when it alters a program, an encrypted hash function can be used. The encryption key is stored separately from the program so that the malware cannot generate a new hash code and encrypt that. By using a hash function rather than a simpler checksum, the malware is prevented from adjusting the program to produce the same hash code as before. If a protected list of programs in trusted locations is kept, this approach can also detect attempts to replace or install rogue code or programs in these locations.

Third-generation programs are memory-resident programs that identify malware by its actions rather than its structure in an infected program. Such programs the advantage that it is not necessary to develop signatures and heuristics for a wide array of malware. Rather, it is necessary only to identify the small set of actions that indicate that malicious activity is being attempted and then to intervene.

Fourth-generation products are packages consisting of a variety of antivirus techniques used in conjunction. These include scanning and activity trap components. In addition, such a package includes access control capability, which limits the ability of malware to penetrate a system and then limits the ability of a malware to update files in order to propagate.

The arms race continues. With fourth-generation packages, a more comprehensive defense strategy is employed, broadening the scope of defense to more general-purpose computer security measures. These include more sophisticated antivirus approaches. We now highlight two of the most important.

Host-Based Behavior-Blocking Software Unlike heuristics or fingerprint-based scanners, **behavior-blocking software** integrates with the operating system of a host computer and monitors program behavior in real time for malicious actions [CONR02, NACH02]. The behavior blocking software then blocks potentially malicious actions before they have a chance to affect the system. Monitored behaviors can include the following:

- Attempts to open, view, delete, and/or modify files
- Attempts to format disk drives and other unrecoverable disk operations
- Modifications to the logic of executable files or macros
- Modification of critical system settings, such as start-up settings
- Scripting of e-mail and instant messaging clients to send executable content
- Initiation of network communications

Because a behavior blocker can block suspicious software in real time, it has an advantage over such established antivirus detection techniques as fingerprinting or heuristics. There are literally trillions of different ways to obfuscate and rearrange the instructions of a virus or worm, many of which will evade detection by a fingerprint scanner or heuristic. But eventually, malicious code must make a well-defined request to the operating system. Given that the behavior blocker can intercept all such requests, it can identify and block malicious actions regardless of how obfuscated the program logic appears to be.