11.3 The high-speed transport protocol XTP (Xpress Transfer Protocol) uses a 32-bit checksum function defined as the concatenation of two 16-bit functions: XOR and RXOR, defined in Section 11.4 as "two simple hash functions" and illustrated in Figure 11.7.

a.

Will this checksum detect all errors caused by an odd number of error bits? Explain.

b.

Will this checksum detect all errors caused by an even number of error bits? If not, characterize the error patterns that will cause the checksum to fail.

c.

Comment on the effectiveness of this function for use as a hash function for authentication.

11.4 **a.**

Consider the Davies and Price hash code scheme described in Section 11.4 and assume that DES is used as the encryption algorithm:

$$H_i = H_{i1} \oplus \mathrm{E}(M_i, H_{i1})$$

and recall the complementarity property of DES (Problem 3.14): If $Y = \mathrm{E}(K, X)$, then $Y' = \mathrm{E}(K', X')$. Use this property to show how a message consisting of blocks $M_1, M_2, \ldots, M_N$ can be altered without altering its hash code.

b.

Show that a similar attack will succeed against the scheme proposed in [MEYE88]:

$$H_i = M_i \oplus \mathrm{E}(H_i1, M_i)$$

11.5 **a.**

Consider the following hash function. Messages are in the form of a sequence of decimal numbers, $M = (a_1, a_2, \ldots, a_t)$. The hash value h is calculated as

$$\left(\sum_{i=1}^{t} a_i\right) \bmod n$$

, for some predefined value n. Does this hash function satisfy any of the requirements for a hash function listed in Section 11.4? Explain your answer.

---

b.

Repeat part (a) for the hash function $h = \left(\sum_{i=1}^{t} (a_i)^2\right) \bmod n$.

c.

Calculate the hash function of part (b) for $M = (189, 632, 900, 722, 349)$ and $n = 989$.

11.6 It is possible to use a hash function to construct a block cipher with a structure similar to DES. Because a hash function is one way and a block cipher must be reversible (to decrypt), how is it possible?

11.7 Now consider the opposite problem: using an encryption algorithm to construct a one-way hash function. Consider using RSA with a known key. Then process a message consisting of a sequence of blocks as follows: Encrypt the first block, XOR the result with the second block and encrypt again, etc. Show that this scheme is not secure by solving the following problem. Given a two-block message B1, B2, and its hash

$$\text{RSAH(B1, B2)} = \text{RSA(RSA (B1)} \oplus \text{B2)}$$

Given an arbitrary block C1, choose C2 so that RSAH(C1, C2) = RSAH(B1, B2). Thus, the hash function does not satisfy weak collision resistance.

11.8 Suppose H(m) is a collision resistant hash function that maps a message of arbitrary bit length into an n-bit hash value. Is it true that, for all messages x, x' with $x \neq x'$, we have $H(x) \neq H(x')$? Explain your answer.

[Page 346 (continued)]

Appendix 11A Mathematical Basis of the Birthday Attack

In this appendix, we derive the mathematical justification for the birthday attack. We begin with a related problem and then look at the problem from which the name "birthday attack" is derived.

Related Problem

A general problem relating to hash functions is the following. Given a hash function H, with n possible outputs and a specific value H(x), if H is applied to k random inputs, what must be the value of k so that the probability that at least one input y satisfies H(y) = H(x) is 0.5?

For a single value of y, the probability that H(y) = H(x) is just $1/n$. Conversely, the probability that H(y) $\neq$ H(x) is $[1 - (1/n)]$. If we generate k random values of y, then the probability that none of them match is just the product of the probabilities that each individual value does not match, or $[1 - (1/n)]^k$. Thus, the probability that there is at least one match is $1 - [1 - (1/n)]^k$.

The binomial theorem can be stated as follows:

$$(1 - a)^k = 1 - ka + \frac{k(k-1)}{2!}a^2 - \frac{k(k-1)(k-2)}{3!}a^3 \ldots$$

For very small values of a, this can be approximated as $(1 - ka)$. Thus, the probability of at least one match is approximated as $1 - [1 - (1/n)]^k \approx 1 - [1 - (k/n)] = k/n$. For a probability of 0.5, we have $k = n/2$.

In particular, for an m-bit hash code, the number of possible codes is 2^m and the value of k that produces a probability of one-half is

Equation 11-1

$$k = 2^{(m-1)}$$

[Page 347]

The Birthday Paradox

The birthday paradox is often presented in elementary probability courses to demonstrate that probability results are sometimes counterintuitive. The problem can be stated as follows: What is the minimum value of k such that the probability is greater than 0.5 that at least two people in a group of k people have the same birthday? Ignore February 29 and assume that each birthday is equally likely. To answer, let us define

P(n, k) = Pr[at least one duplicate in k items, with each item able to take on one of n equally likely values between 1 and n]

Thus, we are looking for the smallest value of k such that P(365, k) $\geq$ 0.5. It is easier first to derive the probability that there are no duplicates, which we designate as Q(365, k). If $k \geq$ 365, then it is impossible for all values to be different. So we assume $k \leq$ 365. Now consider the number of different ways, N, that we can have k values with no duplicates. We may choose any of the 365 values for the first item, any of the remaining 364 numbers for the second item, and so on. Hence, the number of different ways is

Equation 11-2

$$N = 365 \times 364 \times \ldots (365 - k + 1) = \frac{365!}{(365 - k)!}$$

If we remove the restriction that there are no duplicates, then each item can be any of 365 values, and the total number of possibilities is 365^k. So the probability of no duplicates is simply the fraction of sets of values that have no duplicates out of all possible sets of values:

$$Q(365, k) = \frac{365!/(365 - k)!}{(365)^k} = \frac{365!}{(365 - k)!(365)^k}$$

and

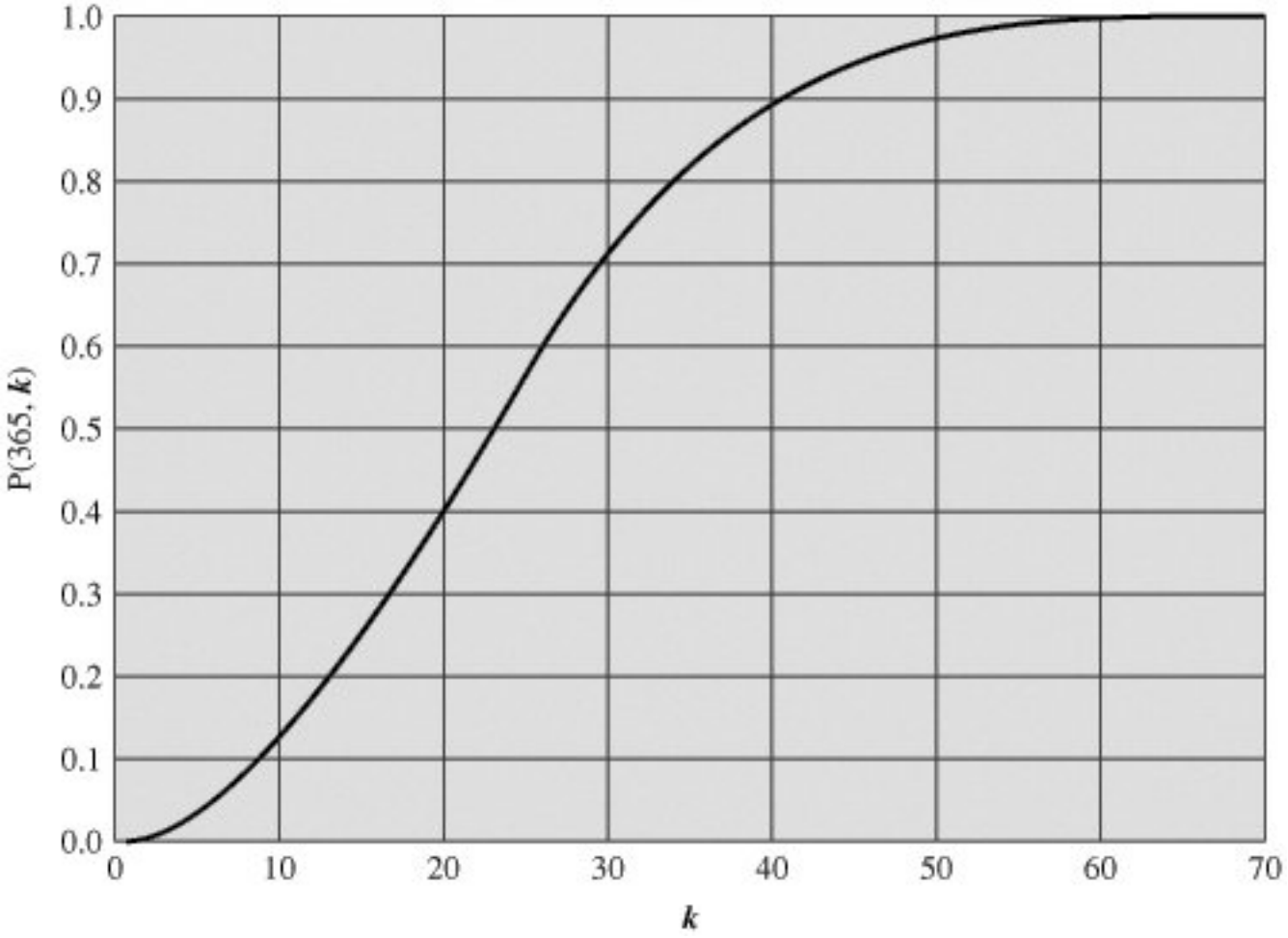
Equation 11-3

$$P(365, k) = 1 - Q(365, k) = 1 - \frac{365!}{(365 - k)!(365)^k}$$

This function is plotted in Figure 11.10. The probabilities may seem surprisingly large to anyone who has not considered the problem before. Many people would guess that to have a probability greater than 0.5 that there is at least one duplicate, the number of people in the group would have to be about 100. In fact, the number is 23, with P(365, 23) = 0.5073. For k = 100, the probability of at least one duplicate is 0.9999997.

Figure 11.10. The Birthday Paradox

(This item is displayed on page 348 in the print version)



Perhaps the reason that the result seems so surprising is that if you consider a particular person in a group, the probability that some other person in the group has the same birthday is small. But the probability that we are concerned with is the probability that *any* pair of people in the group has the same birthday. In a group of 23, there are (23(23 1))/2 = 253 different pairs of people. Hence the high probabilities.

Useful Inequality

Before developing a generalization of the birthday problem, we derive an inequality that will be needed:

Equation 11-4

$$(1 - x) \leq e^{-x} \qquad \text{for all } x \geq 0$$

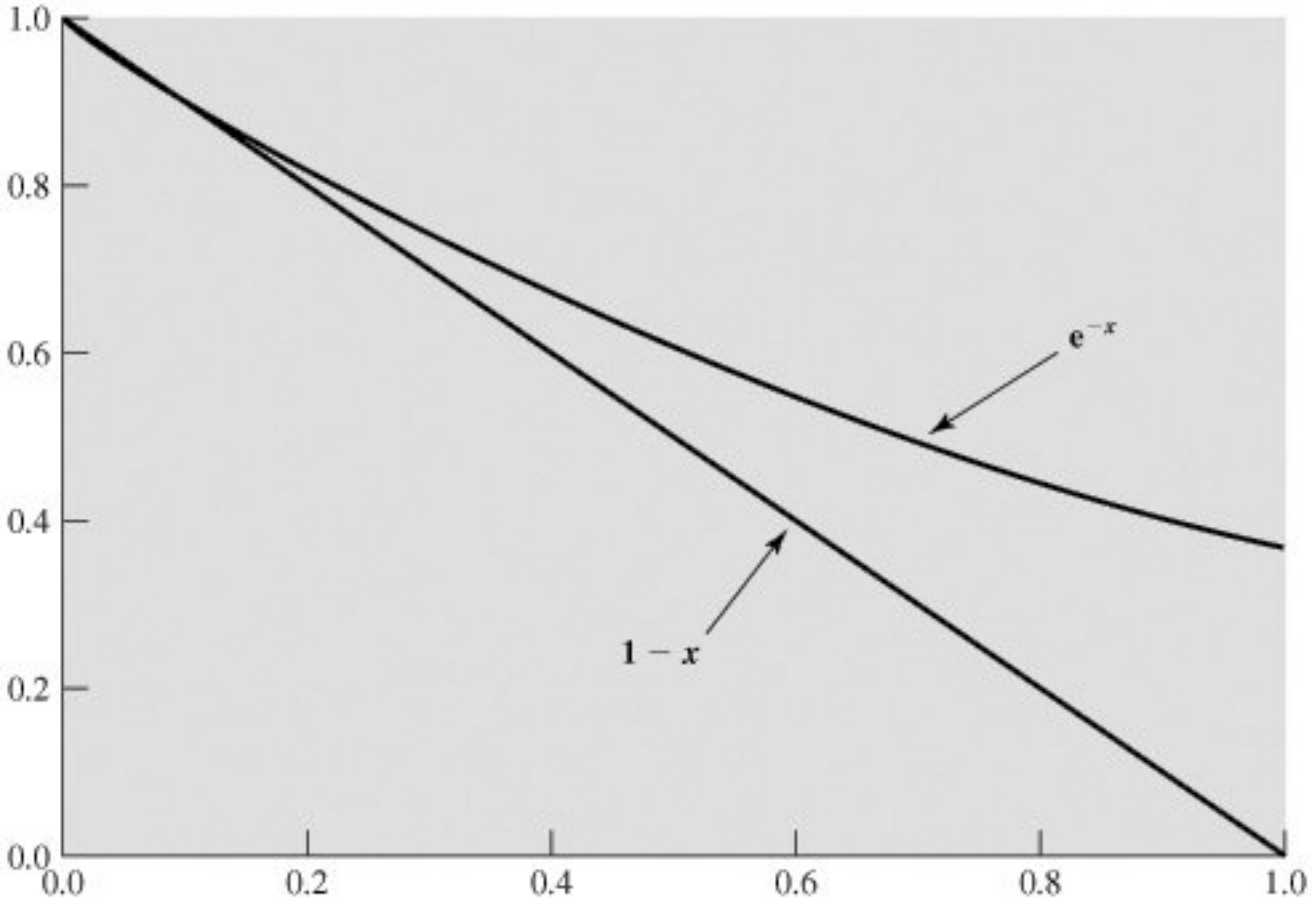
Figure 11.11 illustrates the inequality. To see that the inequality holds, note that the lower line is the tangent to e^x at $x = 0$. at The slope of that line is just the derivative of e^x at $x = 0$;

$$f(x) = e^{-x}$$

$$f'(x) = \frac{d}{dx}e^{-x} = -e^{-x}$$

$$f'(0) = -1$$

Figure 11.11. A Useful Inequality



The tangent is a straight line of the form $ax + b$, with $a = 1$, and the tangent at $x = o$ must equal e^o Thus, the tangent is the function $(1\ x)$, confirming the inequality of Equation (11.4). Further, note that for small x, we have $(1\ x) \approx e^x$.

The General Case of Duplications

The birthday problem can be generalized to the following problem: Given a random variable that is an integer with uniform distribution between 1 and n and a selection of k instances ($k \leq n$) of the random variable, what is the probability, $P(n, k)$, that there is at least one duplicate? The birthday problem is just the special case with $n = 365$. By the same reasoning as before, we have the following generalization of Equation (11.3):

Equation 11-5

$$P(n,k) = 1 - \frac{n!}{(n-k)!n^k}$$

We can rewrite as

$$\begin{aligned} P(n,k) &= 1 - \frac{n \times (n-1) \times \ldots \times (n-k+1)}{n^k} \\ &= 1 - \left[\frac{n-1}{n} \times \frac{n-2}{n} \times \ldots \times \frac{n-k+1}{n}\right] \\ &= 1 - \left[\left(1 - \frac{1}{n}\right) \times \left(1 - \frac{2}{n}\right) \times \ldots \times \left(1 - \frac{k-1}{n}\right)\right] \end{aligned}$$

Using the inequality of Equation (11.4):

$$\begin{aligned} P(n,k) &> 1 - \left[(e^{-1/n}) \times (e^{-2/n}) \times \ldots \times (e^{-(k-1)/n})\right] \\ &> 1 - e^{-[(1/n)+(2/n)+\ldots+((k-1)/n)]} \\ &> 1 - e^{-(k\times(k-1))/2n} \end{aligned}$$

Now let us pose the question: What value of k is required such that $P(n, k) \geq 0.5$? To satisfy the requirement, we have

$$\begin{aligned} 1/2 &= 1 - e^{-(k\times(k-1))/2n} \\ 2 &= e^{(k\times(k-1))/2n} \\ \ln 2 &= \frac{k \times (k-1)}{2n} \end{aligned}$$

For large k, we can replace k x (k 1) by k^2, and we get

Equation 11-6

$$k = \sqrt{2(\ln 2)n} = 1.18\sqrt{n} \approx \sqrt{n}$$

As a reality check, for n = 365, we get $k = 1.18 \times \sqrt{365} = 22.54$ which is very close to the correct answer of 23.

We can now state the basis of the birthday attack in the following terms. Suppose we have a function H, with 2^m possible outputs (i.e., an m-bit output). If H is applied to k random inputs, what must be the value of k so that there is the probability of at least one duplicate [i.e., H(x) = H(y) for some inputs x, y)]? Using the approximation in Equation (11.6):

Equation 11-7

$$k = \sqrt{2^m} = 2^{m/2}$$

Overlap between Two Sets

There is a problem related to the general case of duplications that is also of relevance for our discussions. The problem is this: Given an integer random variable with uniform distribution between 1 and n and two sets of k instances ($k \leq n$) of the random variable, what is the probability, R(n, k), that the two sets are not disjoint; that is, what is the probability that there is at least one value found in both sets?

Let us call the two sets X and Y, with elements $\{x_1, x_2, \ldots, x_k\}$ and $\{y_1, y_2, \ldots, y_k\}$, respectively. Given the value of x_1, the probability that $y_1 = x_1$ is just $1/n$, and therefore probability that does not match x_1 is $[1 - (1/n)]$. If we generate the k random values in Y, the probability that none of these values is equal to is $[1 - (1/n)]^k$. Thus, the probability that there is at least one match to x_1 is $1 - [1 - (1/n)]^k$.

To proceed, let us make the assumption that all the elements of X are distinct. If n is large and if k is also large (e.g., on the order of $\sqrt{n}$), then this is a good approximation. In fact, there may be a few duplications, but most of the values will be distinct. With that assumption, we can make the following derivation:

$$\Pr[\text{no match in } Y \text{ to } x_1] = \left(1 - \frac{1}{n}\right)^k$$

$$\Pr[\text{no match in } Y \text{ to } X] = \left(\left(1 - \frac{1}{n}\right)^k\right)^k = \left(1 - \frac{1}{n}\right)^{k^2}$$

$$R(n, k) = \Pr[\text{at least one match in } Y \text{ to } X] = 1 - \left(1 - \frac{1}{n}\right)^{k^2}$$

Using the inequality of Equation (11.4):

$R(n, k) > 1 (e^{1/n})^{k2}$

$R(n, k) > 1 (e^{k2/n})$

Let us pose the question: What value of k is required such that $R(n, k) > 0.5$? To satisfy the requirement, we have

$$1/2 = 1 - (e^{-k^2/n})$$
$$2 = e^{k^2/n}$$
$$\ln(2) = \frac{k^2}{n}$$

Equation 11-8

$$k = \sqrt{(\ln(2))n} = 0.83\sqrt{n} \approx \sqrt{n}$$

We can state this in terms related to birthday attacks as follows. Suppose we have a function H, with 2^m possible outputs (i.e., an m-bit output). Apply H to k random inputs to produce the set X and again to k additional random inputs to produce the set Y. What must be the value of k so that there is the probability of at least 0.5 that there is a match between the two sets (i.e., $H(x) = H(y)$ for some inputs $x \in X$, $y \in Y$)? Using the approximation in Equation (11.8):

$$k = \sqrt{2^m} = 2^{m/2}$$

PREV NEXT

Chapter 12. Hash and MAC Algorithms

Each of the messages, like each one he had ever read of Stern's commands, began with a number and ended with a number or row of numbers. No efforts on the part of Mungo or

any of his experts had been able to break Stern's code, nor was there any clue as to what the preliminary number and those ultimate numbers signified.

Talking to Strange Men, Ruth Rendell

The Douglas Squirrel has a distinctive eating habit. It usually eats pine cones from the bottom end up. Partially eaten cones can indicate the presence of these squirrels if they have been attacked from the bottom first. If, instead, the cone has been eaten from the top end down, it is more likely to have been a crossbill finch that has been doing the dining.

Squirrels: A Wildlife Handbook, Kim Long

Key Points

- Virtually all secure hash algorithms have the general structure shown in Figure 11.9.
- The compression function used in secure hash algorithms falls into one of two categories: a function specifically designed for the hash function or a symmetric block cipher. SHA and Whirlpool are examples of these two approaches, respectively.
- Message authentication codes also fall into two categories; those based on the use of a secure hash algorithm and those based on the use of a symmetric block cipher. HMAC and CMAC are examples of these two approaches, respectively.

In this chapter, we look at important examples of both secure hash algorithms and message authentication codes (MACs). Most important modern hash functions follow the basic structure of Figure 11.9. This has proved to be a fundamentally sound structure, and newer designs simply refine the structure and add to the hash code length. Within this basic structure, two approaches have been followed in the design of the compression function, which is the basic building block of the hash function. Traditionally, most hash functions that have achieved widespread use rely on a compression function specifically designed for the hash function. Typically, the compression function makes use of modular arithmetic and logical binary operations. Another approach is to use a symmetric block cipher as the compression function. In this chapter, we examine perhaps the most important example of each approach: the Secure Hash Algorithm (SHA) and Whirlpool.

MACs also conveniently fall into two categories based on their fundamental building block. One popular approach is to use a hash algorithm such as SHA as the core of the MAC algorithm. Another approach is to use a symmetric block cipher in a cipher block chaining mode. Again, we look at perhaps the most important example of each approach: HMAC and CMAC.

12.1. Secure Hash Algorithm

The Secure Hash Algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993; a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1. The actual standards document is entitled Secure Hash Standard. SHA is based on the hash function MD4 and its design closely models MD4. SHA-1 is also specified in RFC 3174, which essentially duplicates the material in FIPS 180-1, but adds a C code implementation.

SHA-1 produces a hash value of 160 bits. In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512 (Table 12.1). These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1. In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010. Shortly thereafter, a research team described an attack in which two separate messages could be found that deliver the same SHA-1 hash using 2^{69} operations, far fewer than the 2^{80} operations previously thought needed to find a collision with an SHA-1 hash [WANG05]. This result should hasten the transition to the other versions of SHA.

Table 12.1. Comparison of SHA Parameters

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	64	80	80
Security	80	128	192	256

Notes: 1. All sizes are measured in bits.

2. Security refers to the fact that a birthday attack on a message digest of size *n* produces a collision with a workfactor of approximately $2^{n/2}$

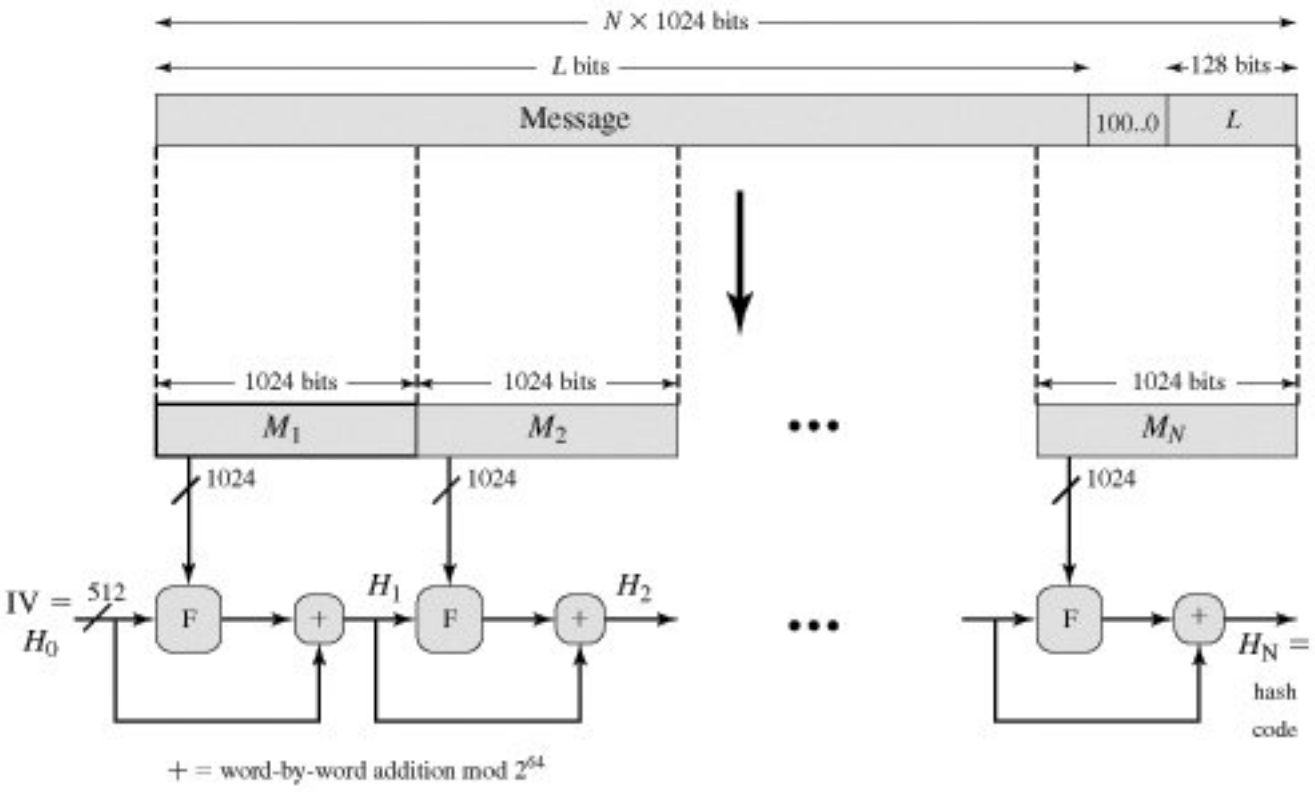
In this section, we provide a description of SHA-512. The other versions are quite similar.

SHA-512 Logic

The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks. Figure 12.1 depicts the overall processing of a message to produce a digest.

Figure 12.1. Message Digest Generation Using SHA-512

[View full size image]



This follows the general structure depicted in Figure 11.9. The processing consists of the following steps:

- **Step 1: Append padding bits.** The message is padded so that its length is congruent to 896 modulo 1024 [length ≡ 896 (mod 1024)]. Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 1024. The padding consists of a single 1-bit followed by the necessary number of 0-bits.
- **Step 2: Append length.** A block of 128 bits is appended to the message. This block is treated as an unsigned 128-bit integer (most significant byte first) and contains the length of the original message (before the padding).

  The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length. In Figure 12.1, the expanded message is represented as the sequence of 1024-bit blocks M_1, M_2,..., M_N, so that the total length of the expanded message is N x 1024 bits.
- **Step 3: Initialize hash buffer.** A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h). These registers are initialized to the following 64-bit integers (hexadecimal values):

  a = 6A09E667F3BCC908

b = BB67AE8584CAA73B

c = 3C6EF372FE94F82B

c = A54FF53A5F1D36F1

e = 510E527FADE682D1

f = 9B05688C2B3E6C1F

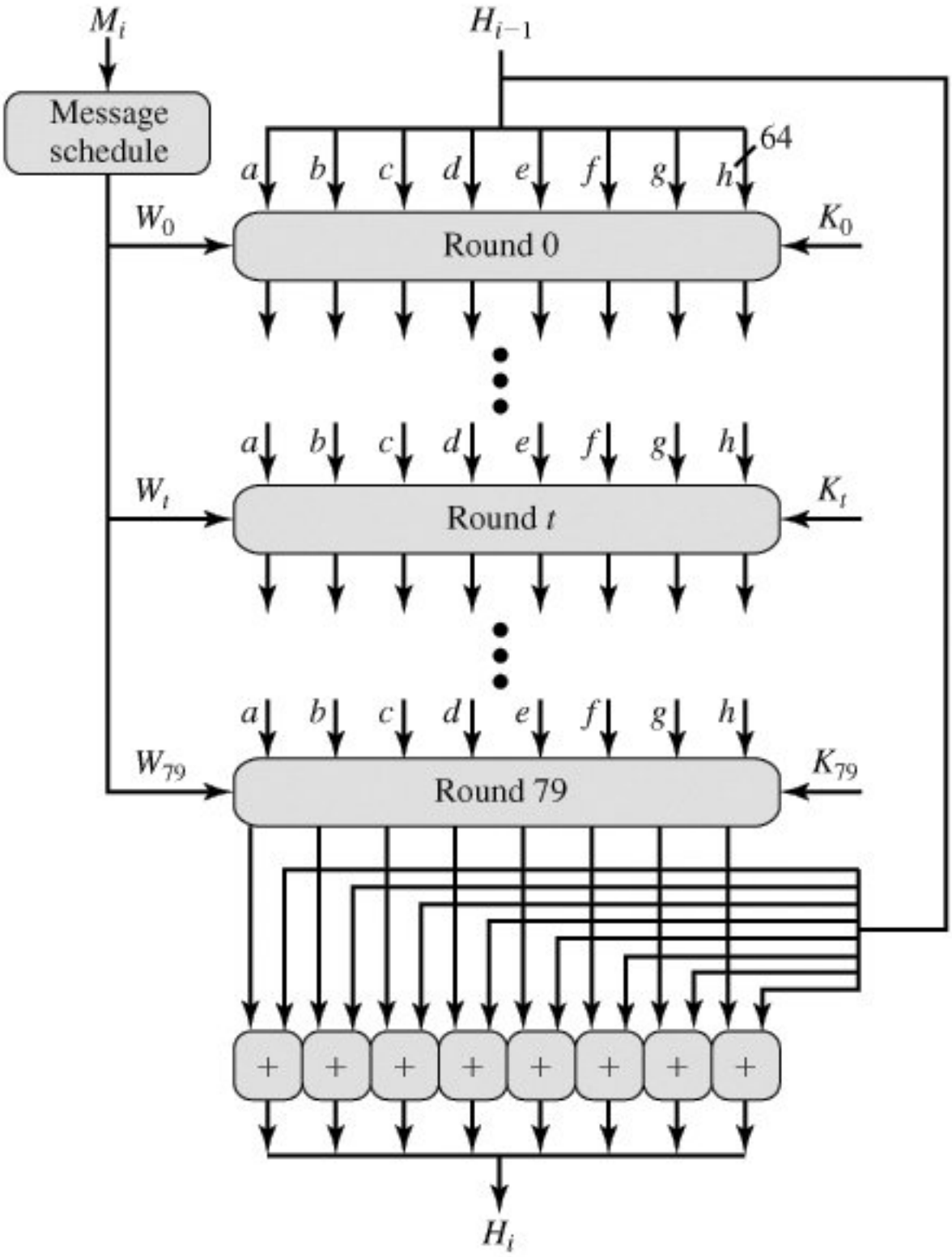
g = 1F83D9ABFB41BD6B

h = 5BE0CDI9137E2179

These values are stored in big-endianformat, which is the most significant byte of a word in the low-address (leftmost) byte position. These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

- **Step 4: Process message in 1024-bit (128-word) blocks.** The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure 12.1. The logic is illustrated in Figure 12.2.

Figure 12.2. SHA-512 Processing of a Single 1024-Bit Block



Each round takes as input the 512-bit buffer value abcdefgh, and updates the contents of the buffer. At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1}.

Each round t makes use of a 64-bit value W_t derived from the current 1024-bit block being processed (M_i) These values are derived using a message schedule described subsequently. Each round also makes use of an additive constant K_t where $0 \leq t \leq 79$ indicates one of the 80 rounds. These words represent the first sixty-four bits of the fractional parts of the cube roots of the first eighty prime numbers. The constants provide a "randomized" set of 64-bit patterns, which should eliminate any regularities in the input data.

The output of the eightieth round is added to the input to the first round (H_{i-1})to produce H_i. The addition is done independently for each of the eight words in the buffer with each of the corresponding words in H_{i-1} using addition modulo 2^{64}.

- **Step 5: Output.** After all *N* 1024-bit blocks have been processed, the output from the *N*th stage is the 512-bit message digest.

We can summarize the behavior of SHA-512 as follows:

$$H_0 = \text{IV}$$

$$H_i = \text{SUM}_{64}(H_{i-1}, \text{abcdefgh}_i)$$

$$MD = H_N$$

where

IV	=	initial value of the abcdefgh buffer, defined in step 3
abcdefgh_i	=	the output of the last round of processing of the *i*th message block
N	=	the number of blocks in the message (including padding and length fields)
SUM_{64}	=	Addition modulo 2^{64} performed separately on each word of the pair of inputs
MD	=	final message digest value

SHA-512 Round Function

Let us look in more detail at the logic in each of the 80 steps of the processing of one 512-bit block (Figure 12.3). Each round is defined by the following set of equations:

$$T_1 = h + \mathrm{Ch}(e, f, g) + \left(\sum_1^{512} e\right) + W_t + K_t$$

$$T_2 = \left(\sum_0^{512} a\right) + \mathrm{Maj}(a, b, c)$$

$$a = T_1 + T_2$$

$$b = a$$

$$c = b$$

$$d = c$$

$$e = d + T_1$$

$$f = e$$

$$g = f$$

$$h = g$$

where

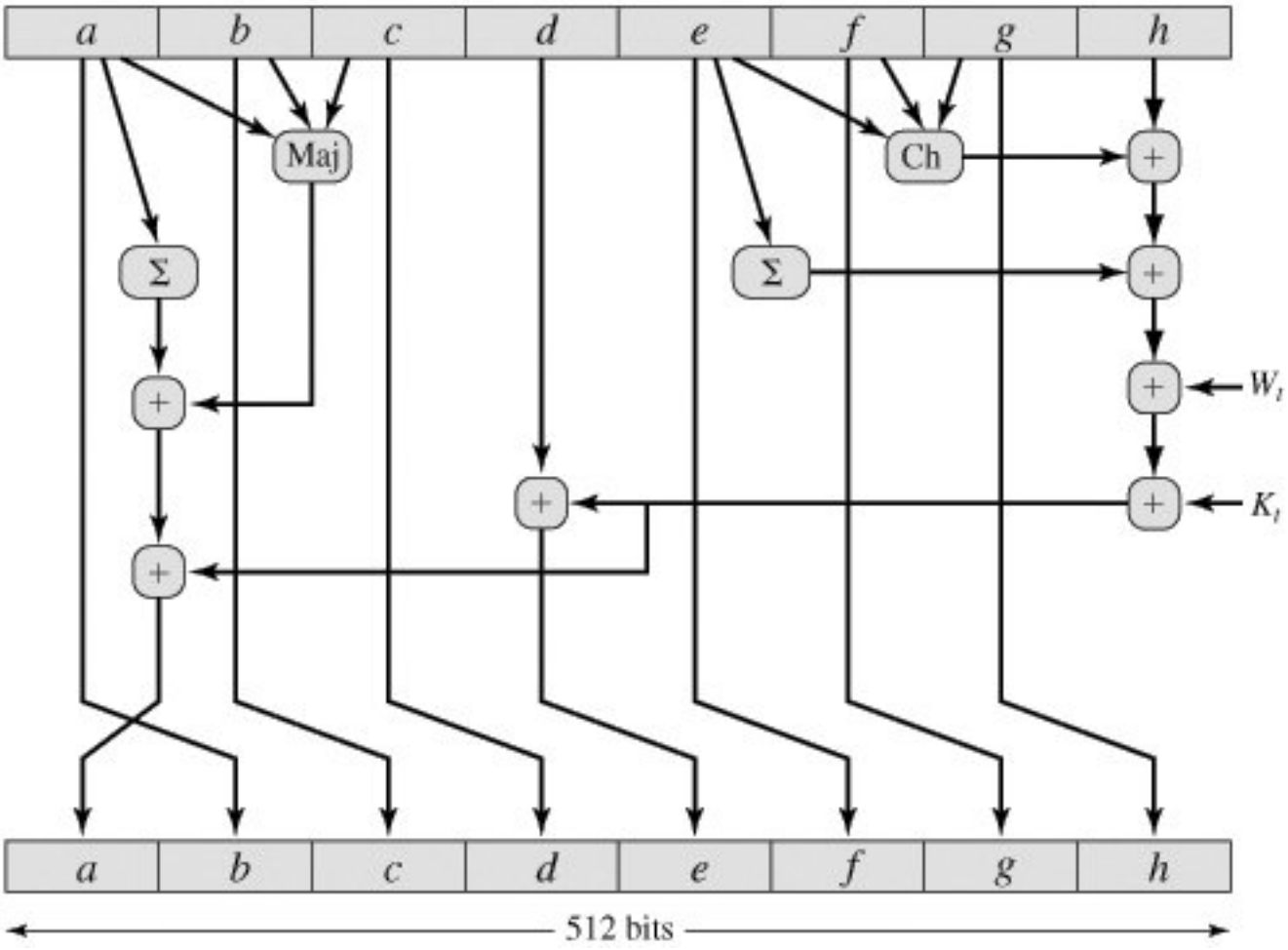
t	=step number; $0 \leq t \leq 79$
Ch(e, f, g)	= (e AND f) $\oplus$ (NOT e AND g) *the conditional function*: *If* e then f else g
Maj(a, b, c)	= (a AND b) $\oplus$ (a AND c) $\oplus$ (b AND c) *the function is true only of the majority (two or three) of the arguments are true.*

$$\left(\sum_0^{512} a\right) = \mathrm{ROTR}^{28}(a) \oplus \mathrm{ROTR}^{34}(a) \oplus \mathrm{ROTR}^{39}(a)$$

$$\left(\sum_1^{512} e\right) = \mathrm{ROTR}^{14}(e) \oplus \mathrm{ROTR}^{18}(e) \oplus \mathrm{ROTR}^{41}(e)$$

$\mathrm{ROTR}^n(x)$	= circular right shift (rotation) of the 64-bit argument x by n bits
W_t	= a 64-bit word derived from the current 512-bit input block
K_t	= a 64-bit additive constant
+	= addition modulo 2^{64}

Figure 12.3. Elementary SHA-512 Operation (single round)



It remains to indicate how the 64-bit word values W_t are derived from the 1024-bit message. Figure 12.4 illustrates the mapping. The first 16 values of W_t are taken directly from the 16 words of the current block. The remaining values are defined as follows:

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

where

$$\sigma_0^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$$
$$\sigma_1^{512}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)$$

$\text{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

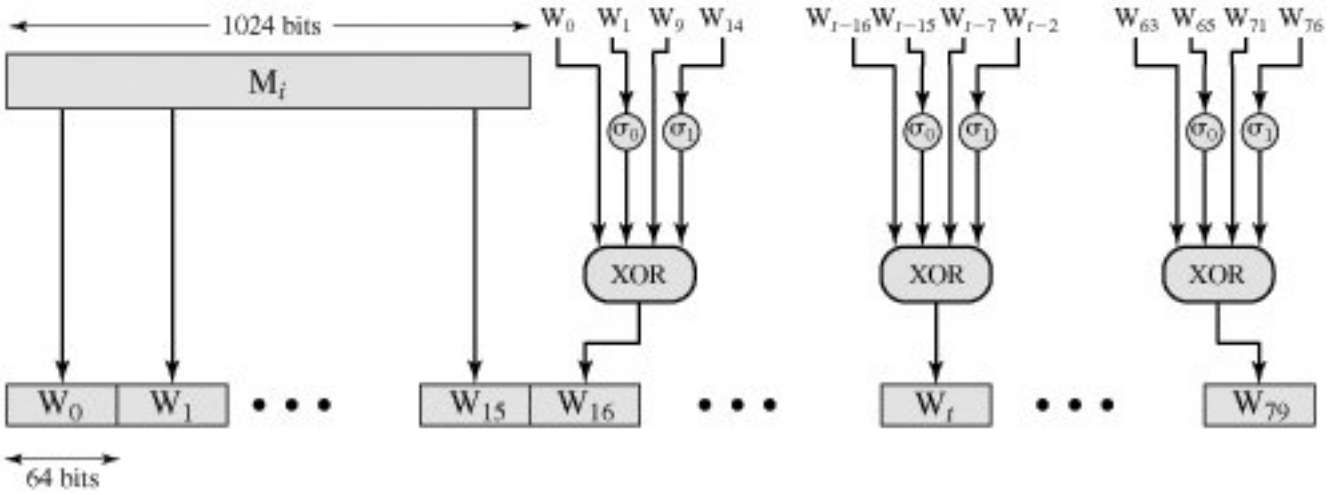
$\text{SHR}^n(x)$ = left shift of the 64-bit argument x by n bits with padding *by zeros on the right*

Figure 12.4. Creation of 80-word Input Sequence for SHA-512 Processing of

Single Block

(This item is displayed on page 358 in the print version)

[View full size image]



Thus, in the first 16 steps of processing, the value of W_t is equal to the corresponding word in the message block. For the remaining 64 steps, the value of W_t consists of the circular left shift by one bit of the XOR of four of the preceding values of W_t, with two of those values subjected to shift and rotate operations. This introduces a great deal of redundancy and interdependence into the message blocks that are compressed, which complicates the task of finding a different message block that maps to the same compression function output.

PREV NEXT

12.2. Whirlpool[1]

[1] Most of the material in this section originally appeared in [STAL O6].

In this section, we examine the hash function Whirlpool [BARR03], one of whose designers is also co-inventor of Rijndael, adopted as the Advanced Encryption Standard (AES). Whirlpool is one of only two hash functions endorsed by NESSIE (New European Schemes for Signatures, Integrity, and Encryption). [2] The NESSIE project is a European Unionsponsored effort to put forward a portfolio of strong cryptographic primitives of various types.

[2] The other endorsed scheme consists of three variants of SHA: SHA-256, SHA-384, and SHA-512.

Whirlpool is based on the use of a block cipher for the compression function. As was mentioned in Chapter 11, there has traditionally been little interest in the use of block-cipher-based hash functions because of the demonstrated security vulnerabilities of the structure. The following are potential drawbacks:

1. Block ciphers do not possess the properties of randomizing functions. For example, they are invertible. This lack of randomness may lead to weaknesses that can be exploited.

2. Block ciphers typically exhibit other regularities or weaknesses. For example, [MIYA90] demonstrates how to compromise many hash schemes based on properties of the underlying block cipher.

3. Typically, block-cipher-based hash functions are significantly slower than hash functions based on a compression function specifically designed for the hash function.

4. A principal measure of the strength of a hash function is the length of the hash code in bits. For block-cipher-based hash codes, proposed designs have a hash code length equal to either the cipher block length or twice the cipher block length. Traditionally, cipher block length has been limited to 64 bits (e.g., DES, triple DES), resulting in a hash code of questionable strength.

However, since the adoption of AES, there has been renewed interested in developing a secure hash function based on a strong block cipher and exhibiting good performance. Whirlpool is a block-cipher-

based hash function intended to provide security and performance that is comparable, if not better, than that found in non-block-cipher based hash functions, such as SHA. Whirlpool has the following features:

1. The hash code length is 512 bits, equaling the longest hash code available with SHA.
2. The overall structure of the hash function is one that has been shown to be resistant to the usual attacks on block-cipher-based hash codes.
3. The underlying block cipher is based on AES and is designed to provide for implementation in both software and hardware that is both compact and exhibits good performance.

The design of Whirlpool sets the following security goals: Assume we take as hash result the value of any n-bit substring of the full Whirlpool output.

- The expected workload of generating a collision is of the order of $2^{n/2}$ executions of Whirlpool.
- Given an n-bit value, the expected workload of finding a message that hashes to that value is of the order of 2^n executions of Whirlpool.
- Given a message and its n-bit hash result, the expected workload of finding a second message that hashes to the same value is of the order of 2^n executions of Whirlpool.
- It is infeasible to detect systematic correlations between any linear combination of input bits and any linear combination of bits of the hash result, or to predict what bits of the hash result will change value when certain input bits are flipped (this means resistance against linear and differential attacks).

The designers assert their confidence that these goals have been met with a considerable safety margin. However, the goals are not susceptible to a formal proof.

We begin with a discussion of the structure of the overall hash function, and then examine the block cipher used as the basic building block.

Whirlpool Hash Structure

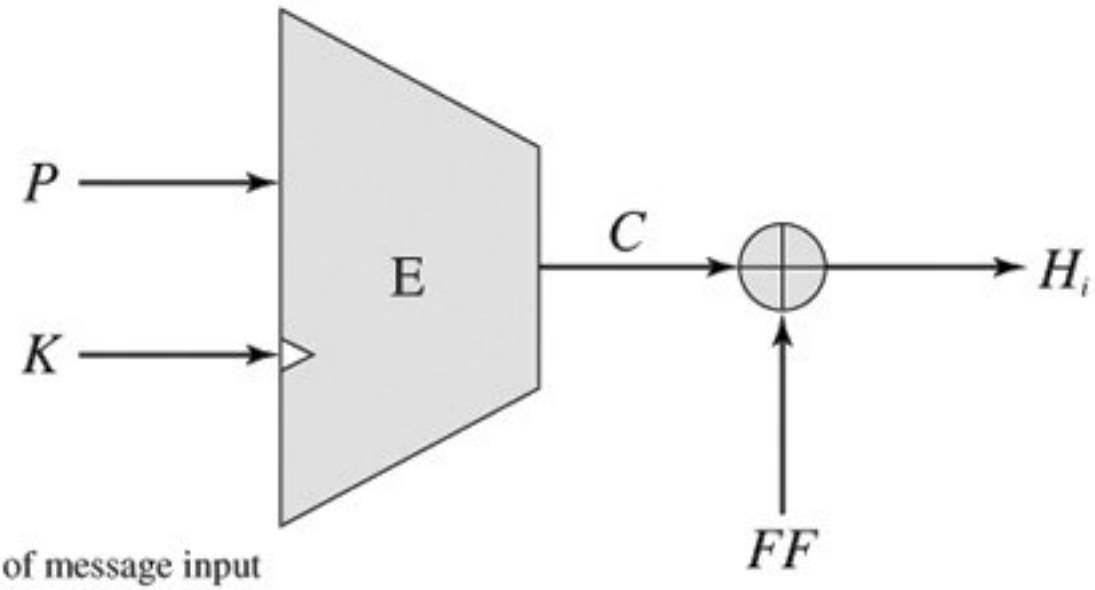
Background

The general iterated hash structure proposed by Merkle (Figure 11.9) is used in virtually all secure hash functions. However, as was pointed out, there are difficulties in designing a truly secure iterated hash function when the compression function is a block cipher. Preneel [PREN93a, PREN93b] performed a systematic analysis of block-cipher-based hash functions, using the model depicted in Figure 12.5. In this model, the hash code length equals the cipher block length. Additional security problems are introduced and the analysis is more difficult if the hash code length exceeds the cipher block length. Preneel devised 64 possible permutations of the basic model, based on which input served as the encryption key and which served as plaintext and on what input, if any, was combined with the ciphertext to produce the intermediate hash code. Based on his analysis, he concluded that only schemes in which the plaintext was fed forward and combined with the ciphertext were secure. Such an arrangement makes the compression function difficult to invert. [BLAC02] confirmed these results, but pointed out the security problem of using an established block cipher such as AES: The 128-bit hash

code value resulting from the use of AES or another scheme with the same block size may be inadequate for security.

Figure 12.5. Model of Single Iteration of Hash Function (hash code equals block length)

Note: **Triangular hatch indicates encryption key input.**



m_i = ith block of message input
H_i = ith intermediate hash value
P = plaintext; K = encryption key; C = ciphertext
FF = feed forward value
P, K, and FF can be chosen from the set $(0, m_i, H_{i-1}, m_i \oplus H_{i-1})$

Whirlpool Logic

Given a message consisting of a sequence of blocks m_1, m_2,..., m_t the Whirlpool hash function is expressed as follows:

H_0 = initial value

$H_i = E(H_{i-1}, m_i) \oplus H_{i-1} \oplus m_i$ = intermediate value

H_t = hash code value

In terms of the model of Figure 12.5, the encryption key input for each iteration is the intermediate hash value from the previous iteration; the plaintext is the current message block; and the feedforward value is the bitwise XOR of the current message block and the intermediate hash value from the previous iteration.

The algorithm takes as input a message with a maximum length of less than 2^{256} bits and produces as output a 512-bit message digest. The input is processed in 512-bit blocks. Figure 12.6 depicts the overall processing of a message to produce a digest. This follows the general structure depicted in Figure 11.9. The processing consists of the following steps:

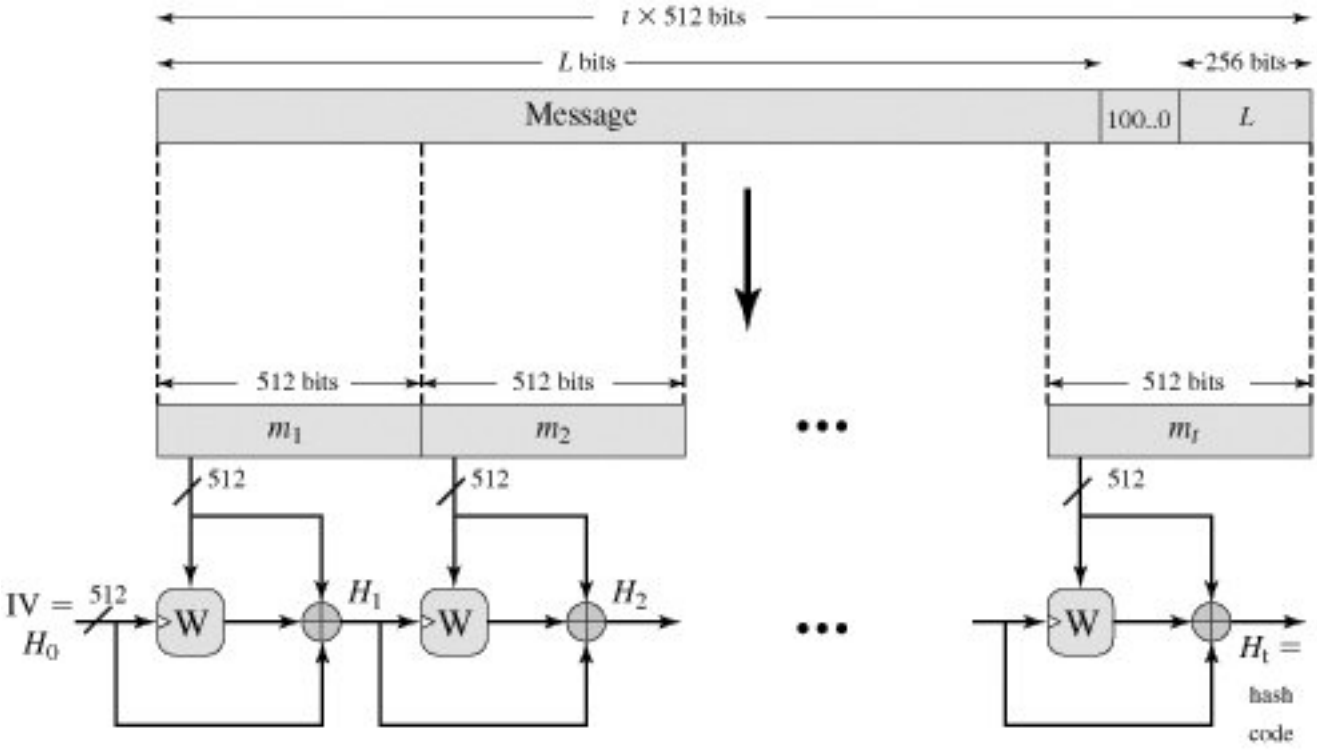
- **Step 1: Append padding bits.** The message is padded so that its length in bits is an odd multiple of 256. Padding is always added, even if the message is already of the desired length. For example, if the message is 256 x 3 = 768 bits long, it is padded by 512 bits to a length of 256 x 5 = 1280 bits. Thus, the number of padding bits is in the range of 1 to 512.

Figure 12.6. Message Digest Generation Using Whirlpool

(This item is displayed on page 361 in the print version)

Note: **Triangular hatch marks key input.**

[View full size image]



The padding consists of a single 1-bit followed by the necessary number of 0-bits.

- **Step 2: Append length.** A block of 256 bits is appended to the message. This block is treated as an unsigned 256-bit integer (most significant byte first) and contains the length in bits of the original message (before the padding).

The outcome of the first two steps yields a message that is an integer multiple of 512 bits in length. In Figure 12.6, the expanded message is represented as the sequence of 512-bit blocks $m_1, m_2, \ldots, m_t$ so that the total length of the expanded message is t x 512 bits. These blocks are

viewed externally as arrays of bytes by sequentially grouping the bits in 8-bit chunks. However, internally, the hash state H_i is viewed as an 8 x 8 matrix of bytes. The transformation between

the two is explained subsequently.

- **Step 3: Initialize hash matrix.** An 8 x 8 matrix of bytes is used to hold intermediate and final results of the hash function. The matrix is initialized as consisting of all 0-bits.
- **Step 4: Process message in 512-bit (64-byte) blocks.** The heart of the algorithm is the block cipher W.

Block Cipher W

Unlike virtually all other proposals for a block-cipher-based hash function, Whirlpool uses a block cipher that is specifically designed for use in the hash function and that is unlikely ever to be used as a standalone encryption function. The reason for this is that the designers wanted to make use of a block cipher with the security and efficiency of AES but with a hash length that provided a potential security equal to SHA-512. The result is the block cipher W, which has a similar structure and uses the same elementary functions as AES, but which uses a block size and a key size of 512 bits. Table 12.2 compares AES and W.

Although W is similar to AES, it is not simply an extension. Recall that the Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192, or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. AES operates on a state of 4 x 4 bytes. Rijndael with block length 192 bits operates on a state of 4 x 6 bytes. Rijndael with block length 256 bits operates on a state of 4 x 8 bytes. W operates on a state of 8 x 8 bytes. The more the state representation differs from a square, the slower the diffusion goes and the more rounds the cipher needs. For a block length of 512 bits, the Whirlpool developers could have defined a Rijndael operating on a state of 4 x 16 bytes, but that cipher would have needed many rounds and it would have been very slow.

As Table 12.2 indicates, W uses a row-oriented matrix whereas AES uses a column-oriented matrix. There is no technical reason to prefer one orientation over another, because one can easily construct an equivalent description of the same cipher, exchanging rows with columns.

Table 12.2. Comparison of Whirlpool Block Cipher W and AES

	W	AES
Block size (bits)	512	128
Key size (bits)	512	128, 192, or 256
Matrix orientation	Input is mapped row-wise	Input is mapped column-wise
Number of rounds	10	10, 12, or 14
Key expansion	W round function	Dedicated expansion algorithm
GF(2^8) polynomial	$x^8 + x^4 + x^3 + x^2 + 1$ (011D)	$x^8 + x^4 + x^3 + x + 1$ (011B)
Origin of S-box	Recursive structure	Multiplicative inverse in GF(2^8) plus affine transformation

Origin of round constants	Successive entries of the S-box	Elements 2^i of GF(2^8)
Diffusion layer	Right multiplication by 8 x 8 circulant MDS matrix (1, 1, 4, 1, 8, 5, 2, 9) - mix rows	Left multiplication by 4 x 4 circulant MDS matrix (2, 3, 1, 1) - mix columns
Permutation	Shift columns	Shift rows

Overall Structure

Figure 12.7 shows the overall structure of W. The encryption algorithm takes a 512-bit block of plaintext and a 512-bit key as input and produces a 512-bit block of ciphertext as output. The encryption algorithm involves the use of four different functions, or transformations: add key (AK), substitute bytes (SB), shift columns (SC), and mix rows (MR), whose operations are explained subsequently. W consists of a single application of AK followed by 10 rounds that involve all four functions. We can concisely express the operation of a round *r* as a round function RF that is a composition of functions:

Equation 12-1

$$\mathrm{RF}(\boldsymbol{K}_r) = \mathrm{AK}[\boldsymbol{K}_r] \circ \mathrm{MR} \circ \mathrm{SC} \circ \mathrm{SB}$$

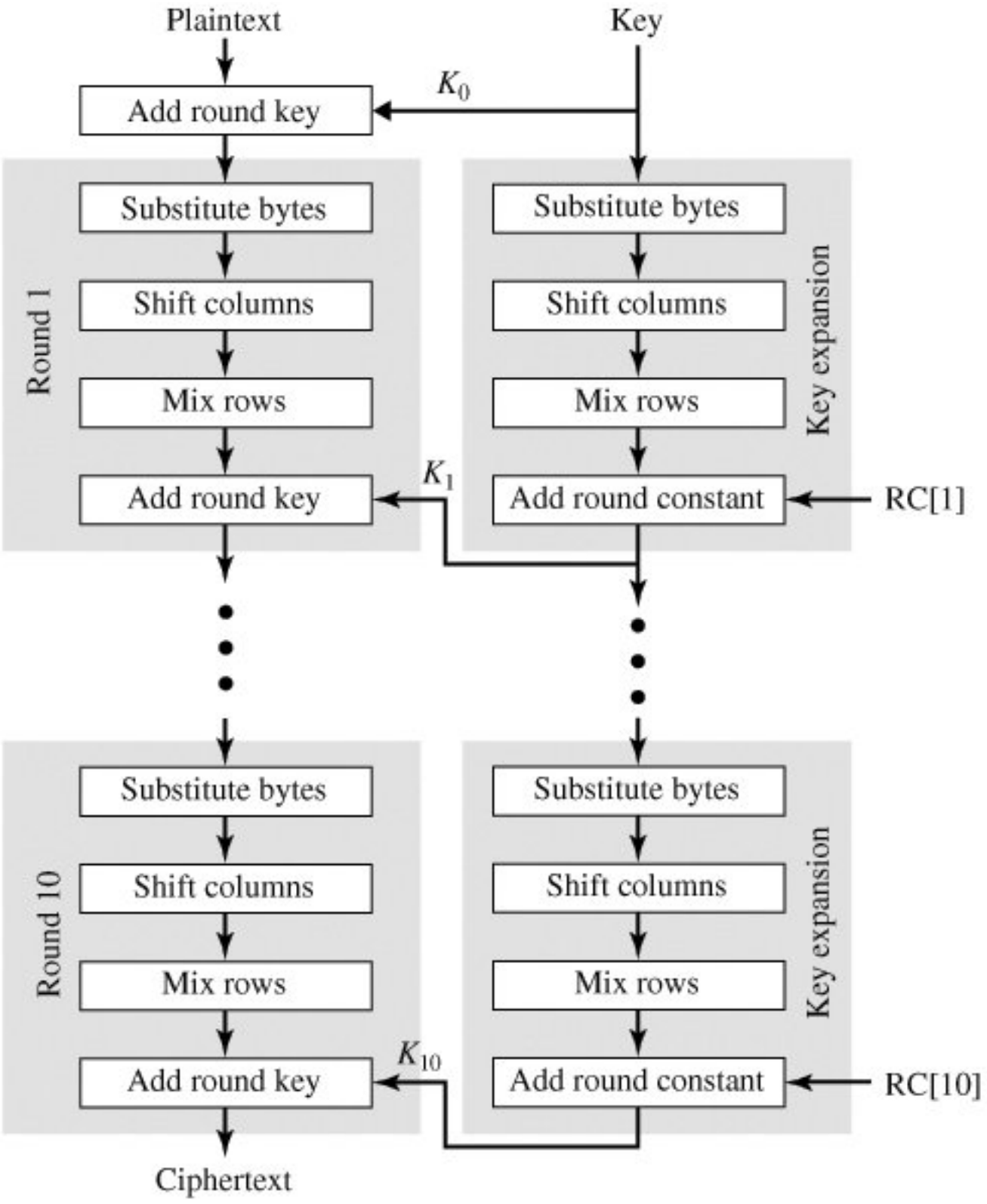
---

where $\boldsymbol{K}_r$ is the round key matrix for round *r*. The overall algorithm, with key input **K**, can be defined as follows:

$$\mathrm{W}(K) = \left(\bigcirc_{r=1}^{10} \mathrm{RF}(\boldsymbol{K}_r)\right) \circ \mathrm{AK}(\boldsymbol{K}_0)$$

where the large circle indicates iteration of the composition function with index *r* running from 1 through 10.

Figure 12.7. Whirlpool Cipher W



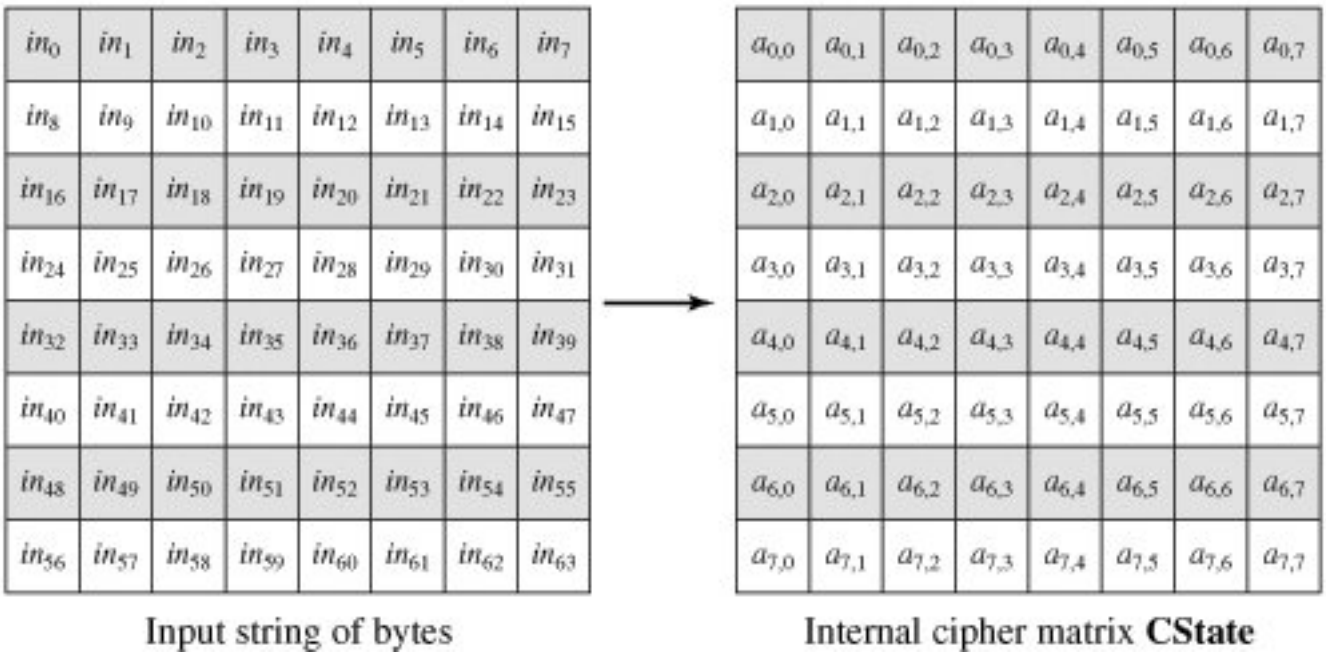
The plaintext input to W is a single 512-bit block. This block is treated as an 8 x 8 square matrix of bytes, labeled **CState**. Figure 12.8 illustrates that the ordering of bytes within a matrix is by row. So, for example, the first eight bytes of a 512-bit plaintext input to the encryption cipher occupy the first row of the internal matrix **CState**, the second eight bytes occupy the second row, and so on. The representation of the linear byte stream as a square matrix can be concisely expressed as a mapping function m For a linear byte array X with elements $x_k (0 \leq k \leq 63)$, the corresponding matrix **A** with elements $a_{i,j} (0 \leq i,j \leq 7)$we have the following correspondence:

$$\mathbf{A} = m(X) \Leftrightarrow a_{i,j} = x_{8_i} + j$$

Figure 12.8. Whirlpool Matrix Structure

(This item is displayed on page 364 in the print version)

[View full size image]



Similarly, the 512-bit key is depicted as a square matrix **KState** of bytes. This key is used as input to the initial AK function. The key is also expanded into a set of 10 round keys, as explained subsequently.

We now look at the individual functions that are part of W.

The Nonlinear Layer SB

The substitute byte function (SB) is a simple table lookup that provides a nonlinear mapping. W defines a 16 x 16 matrix of byte values, called an S-box (Table 12.3), that contains a permutation of all possible 256 8-bit values. Each individual byte of **CState** is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value[3] {95}references row 9, column 5 of the S-box, which contains the value {BA}. Accordingly, the value {95}is mapped into the value {BA}. The SB function can be expressed by the following correspondence, for an input matrix **A** and an output matrix **B**:

[3] As we did for AES, a hexadecimal number is indicated by enclosing it in curly brackets when this is needed for clarity.

$$\mathbf{B} = \text{SB}(\mathbf{A}) \Leftrightarrow b_{i,j} = \text{S}[a_{i,j}],\ 0 \leq i,j \leq 7$$

where S[x] refers to the mapping of input byte x into output byte S[x] by the S-box.

Table 12.3. Whirlpool S-Box

(This item is displayed on page 365 in the print version)

[View full size image]

(a) S-box

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	18	23	C6	E8	87	B8	01	4F	36	A6	D2	F5	79	6F	91	52
1	60	BC	9B	8E	A3	0C	7B	35	1D	E0	D7	C2	2E	4B	FE	57
2	15	77	37	E5	9F	F0	4A	CA	58	C9	29	0A	B1	A0	6B	85
3	BD	5D	10	F4	CB	3E	05	67	E4	27	41	8B	A7	7D	95	C8
4	FB	EE	7C	66	DD	17	47	9E	CA	2D	BF	07	AD	5A	83	33
5	63	02	AA	71	C8	19	49	C9	F2	E3	5B	88	9A	26	32	B0
6	E9	0F	D5	80	BE	CD	34	48	FF	7A	90	5F	20	68	1A	AE
7	B4	54	93	22	64	F1	73	12	40	08	C3	EC	DB	A1	8D	3D
8	97	00	CF	2B	76	82	D6	1B	B5	AF	6A	50	45	F3	30	EF
9	3F	55	A2	EA	65	BA	2F	C0	DE	1C	FD	4D	92	75	06	8A
A	B2	E6	0E	1F	62	D4	A8	96	F9	C5	25	59	84	72	39	4C
B	5E	78	38	8C	C1	A5	E2	61	B3	21	9C	1E	43	C7	FC	04
C	51	99	6D	0D	FA	DF	7E	24	3B	AB	CE	11	8F	4E	B7	EB
D	3C	81	94	F7	B9	13	2C	D3	E7	6E	C4	03	56	44	7F	A9
E	2A	BB	C1	53	DC	0B	9D	6C	31	74	F6	46	AC	89	14	E1
F	16	3A	69	09	70	B6	C0	ED	CC	42	98	A4	28	5C	F8	86

(b) E mini-box

u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E(u)	1	B	9	C	D	6	F	3	E	8	7	4	A	2	5	0

(c) E^{-1} mini-box

u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$E^{-1}(u)$	F	0	D	7	B	E	5	A	9	2	C	1	3	4	8	6

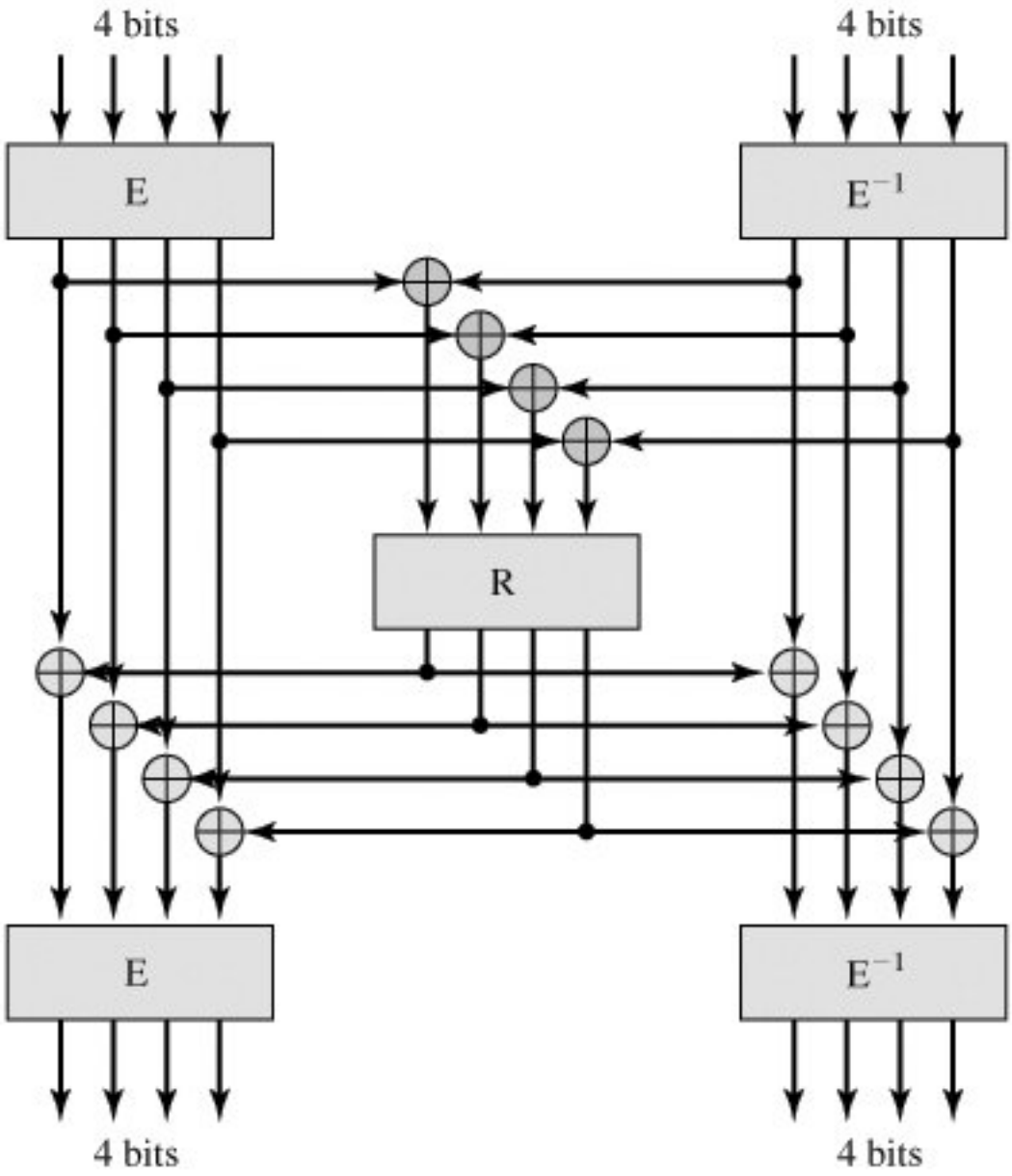
(d) R mini-box

u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
R(u)	7	C	B	D	E	4	9	F	6	3	8	A	2	5	1	0

The S-box can be generated by the structure of Figure 12.9. It consists of two nonlinear layers, each containing two 4 x 4 S-boxes separated by a 4 x 4 randomly generated box. Each of the boxes maps a 4-bit input into a 4-bit output. The E box is defined as $E(u) = \{B\}^u$ if $u \neq \{F\}$ and $E(\{F\}) = 0$, where arithmetic is performed over the finite field GF(2^4)with the irreducible polynomial $f(x) = x^4 + x + 1$.

Figure 12.9. Implementation of Whirlpool S-Box

(This item is displayed on page 366 in the print version)



The SB function is designed to introduce nonlinearity into the algorithm. This means that the SB function should exhibit no correlations between linear combinations of a input bits and linear combinations of a output bits. In addition, differences between sets of input bits should not propagate into similar differences among the corresponding output bits; put another way, small input changes should cause large output changes. These two properties help to make W resistant against linear and differential cryptanalysis.

The Permutation Layer SC

The permutation layer (shift columns) causes a circular downward shift of each column of **CState** except the first column. For the second column, a 1-byte circular downward shift is performed; for the third column, a 2-byte circular downward shift is performed; and so on. The SC function can be expressed by the following correspondence, for an input matrix **A** and an output matrix **B**:

$$\mathbf{B} = SC(\mathbf{A}) \Leftrightarrow b_{i,j} = a_{(i-j) \bmod 8, j} \quad 0 \leq i, j \leq 7$$

The shift column transformation is more substantial than it may first appear. This is because **CState** is treated as an array of eight 8-byte rows. Thus, on encryption, the first 8 bytes of the plaintext are copied to the first row of **CState**, and so on. A column shift moves an individual byte from one row to another, which is a linear distance of a multiple of 8 bytes. Also note that the transformation ensures that the 8 bytes of one row are spread out to eight different rows.

The Diffusion Layer MR

Recall from Chapter 3 that for a function that exhibits diffusion, the statistical structure of the input is dissipated into long-range statistics of the output. This is achieved by having each input bit affect the value of many output bits; generally, this results in each output bit being affected by many input bits. The diffusion layer (mix rows) achieves diffusion within each row individually. Each byte of a row is mapped into a new value that is a function of all eight bytes in that row. The transformation can be defined by the matrix multiplication: **B** = **AC** where **A** is the input matrix, **B** is the output matrix, and **C** is the transformation matrix:

$$C = \begin{bmatrix} 01 & 01 & 04 & 01 & 08 & 05 & 02 & 09 \\ 09 & 01 & 01 & 04 & 01 & 08 & 05 & 02 \\ 02 & 09 & 01 & 01 & 04 & 01 & 08 & 05 \\ 05 & 02 & 09 & 01 & 01 & 04 & 01 & 08 \\ 08 & 05 & 02 & 09 & 01 & 01 & 04 & 01 \\ 01 & 08 & 05 & 02 & 09 & 01 & 01 & 04 \\ 04 & 01 & 08 & 05 & 02 & 09 & 01 & 01 \\ 01 & 04 & 01 & 08 & 05 & 02 & 09 & 01 \end{bmatrix}$$

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications[4] are performed in GF(2^8)with the irreducible polynomial $f(x) = x^8 + x^4 + x^3 + x^2 + 1$. As an example of the matrix multiplication involved, the first element of the output matrix is

[4] As we did for AES, we use the symbol · to indicate multiplication over the finite field GF(2^8)and $\oplus$ to indicate bitwise XOR, which corresponds to addition in GF(2^8)

$$b_{0,0} = a_{0,0} \oplus (9 \cdot a_{0,1}) \oplus (2 \cdot a_{0,2}) \oplus (5 \cdot a_{03}) \oplus (8 \cdot a_{0,4}) \oplus a_{0,5} \oplus (4 \cdot a_{0,6}) \oplus a_{0,7}$$

Note that each row of **C** is constructed by means of a circular right shift of the preceding row. **C** is designed to be a **maximum distance separable** (MDS) matrix. In the field of error-correcting codes, an MDS code takes as input a fixed-length bit string and produces an expanded output string such that there is the maximum Hamming distance between pairs of output strings. With an MDS code, even

multiple bit errors result in a code that is closer to the correct value than to some other value. In the context of block ciphers, a transformation matrix constructed using an MDS code provides a high degree of diffusion [JUNO04]. The use of MDS codes to provide high diffusion was first proposed in [RIJM96].

The matrix **C** is an MDS matrix that has as many 1-elements as possible (3 per row). Overall, the coefficients in **C** provide for efficient hardware implementation.

The Add Key Layer AK

In the add key layer, the 512 bits of **CState** are bitwise XORed with the 512 bits of the round key. The AK function can be expressed by the following correspondence, for an input matrix **A**, an output matrix **B**, and a round key K_i:

$$\mathbf{B} = \mathbf{AK}[K_i](\mathbf{A}) \Leftrightarrow b_{i,j} = a_{i,j} \oplus K_{i,j} \quad 0 \leq i,j \leq 7$$

Key Expansion for the Block Cipher W

As shown in Figure 12.7, key expansion is achieved by using the block cipher itself, with a round constant serving as the round key for the expansion. The round constant for round $r (1 \leq r \leq 10)$ is a matrix **RC**[r] in which only the first row is nonzero, and is defined as follows:

$$rc[r]_{0,j} = S[8(r-1)+j], \quad 0 \leq j \leq 7, 1 \leq r \leq 10$$

$$rc[r]_{i,j} = 0, \quad 1 \leq i \leq 7, 0 \leq j \leq 7, 1 \leq r \leq 10$$

Each element of the first row is a mapping using the S-box. Thus, the first row of **RC**[1] is

S[0]	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]
18	23	C6	E8	87	B8	01	4F

Using the round constants, the key schedule expands the 512-bit cipher key ***K*** onto a sequence of round keys $\mathbf{K}_0, \mathbf{K}_1, \ldots, \mathbf{K}_{10}$:

$$K_0 = K$$

$$K_r = RF[\mathbf{RC}[r]](K_{r-1})$$

where RF is the round function defined in Equation (12.1). Note that for the AK phase of each round, only the first row of **KState** is altered.

Performance of Whirlpool

As yet, there has been little implementation experience with Whirlpool. Recall that the NIST evaluation of Rijndael determined that it exhibited good performance in both hardware and software and that it is well suited to restricted-space (low memory) requirements. These criteria were important in the selection of Rijndael for AES. Because Whirlpool uses the same functional building blocks as AES and has the same structure, we can expect similar performance and space characteristics.

One study that has been done is reported in [KITS04]. The authors compared Whirlpool with a number of other secure hash functions, including all of the versions of SHA. The authors developed multiple hardware implementations of each hash function and concluded that, compared to SHA-512, Whirlpool requires more hardware resources but performs much better in terms of throughput.

PREV NEXT

12.3. HMAC

In Chapter 11, we looked at an example of a message authentication code (MAC) based on the use of a symmetric block cipher, namely the Data Authentication Algorithm defined in FIPS PUB 113. This has traditionally been the most common approach to constructing a MAC. In recent years, there has been increased interest in developing a MAC derived from a cryptographic hash function. The motivations for this interest are

1. Cryptographic hash functions such as MD5 and SHA-1 generally execute faster in software than symmetric block ciphers such as DES.

2. Library code for cryptographic hash functions is widely available.

With the development of AES and the more widespread availability of code for encryption algorithms, these considerations are less significant, but hash-based MACs continue to be widely used.

A hash function such as SHA was not designed for use as a MAC and cannot be used directly for that purpose because it does not rely on a secret key. There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm. The approach that has received the most support is HMAC [BELL96a, BELL96b]. HMAC has been issued as RFC 2104, has been chosen as the mandatory-to-implement MAC for IP security, and is used in other Internet protocols, such as SSL. HMAC has also been issued as a NIST standard (FIPS 198).

HMAC Design Objectives

RFC 2104 lists the following design objectives for HMAC:

- To use, without modifications, available hash functions. In particular, hash functions that perform well in software, and for which code is freely and widely available.
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

The first two objectives are important to the acceptability of HMAC. HMAC treats the hash function as a "black box." This has two benefits. First, an existing implementation of a hash function can be used as a module in implementing HMAC. In this way, the bulk of the HMAC code is prepackaged and ready to use without modification. Second, if it is ever desired to replace a given hash function in an HMAC implementation, all that is required is to remove the existing hash function module and drop in the new module. This could be done if a faster hash function were desired. More important, if the security of the embedded hash function were compromised, the security of HMAC could be retained simply by replacing the embedded hash function with a more secure one (e.g., replacing SHA with Whirlpool).

The last design objective in the preceding list is, in fact, the main advantage of HMAC over other proposed hash-based schemes. HMAC can be proven secure provided that the embedded hash function has some reasonable cryptographic strengths. We return to this point later in this section, but first we examine the structure of HMAC.

HMAC Algorithm

Figure 12.10 illustrates the overall operation of HMAC. Define the following terms:

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)

IV = initial value input to hash function

M = message input to HMAC(including the padding specified in the embedded hash function)

Y_i = *i*th block of *M*, $0 \leq i \leq (L - 1)$

L = number of blocks in *M*

b = number of bits in a block

n = length of hash code produced by embedded hash function

K = secret key recommended length is $\geq n$; if key length is greater than *b*; the key is input to the hash function to produce an *n*-bit key

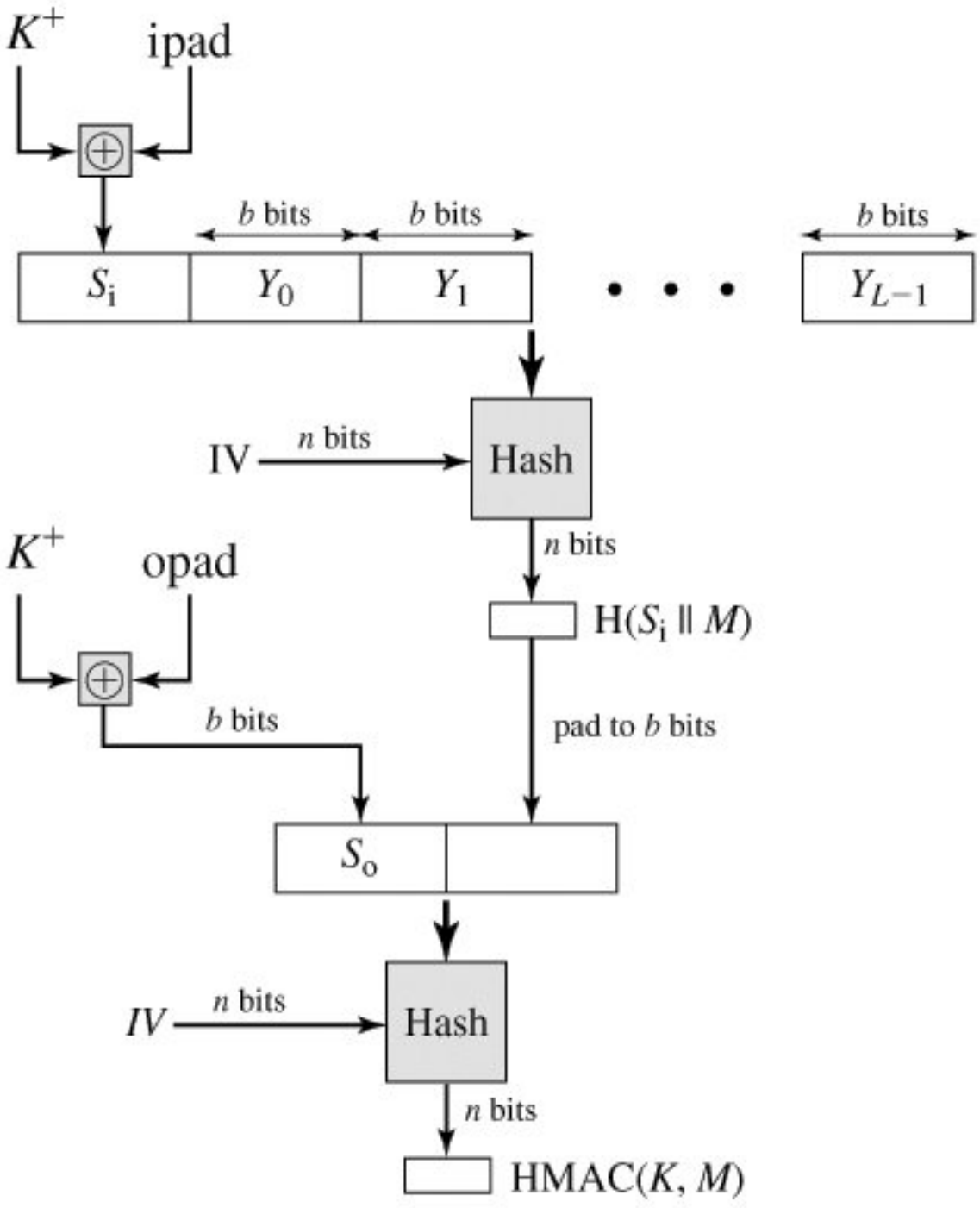
K^+ = *K* padded with zeros on the left so that the result is *b* bits in length

ipad = 00110110 (36 in hexadecimal) repeated *b*/8 times

opad = 01011100 (5C in hexadecimal) repeated *b*/8 times

Figure 12.10. HMAC Structure

(This item is displayed on page 370 in the print version)



Then HMAC can be expressed as follows:

$$\text{HMAC}(K,M) = \text{H}[(K^+ \oplus \text{opad})||\text{H}[(K^+ \oplus \text{ipad})||M]]$$

In words,

1. Append zeros to the left end of K to create a b-bit string K^+(e.g., if K is of length 160 bits and b = 512 then K will be appended with 44 zero bytes 0 x 00).
2.

XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.

3. Append M to S_i.

4. Apply H to the stream generated in step 3.

5. XOR K^+ with opad to produce the b-bit block S_o

6. Append the hash result from step 4 to S_o

7. Apply H to the stream generated in step 6 and output the result.

Note that the XOR with ipad results in flipping one-half of the bits of K. Similarly, the XOR with opad results in flipping one-half of the bits of K, but a different set of bits. In effect, by passing S_i and S_o through the compression function of the hash algorithm, we have pseudorandomly generated two keys from K.

HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the hash compression function (for S_i,S_o and the block produced from the inner hash).

A more efficient implementation is possible, as shown in Figure 12.11. Two quantities are precomputed:

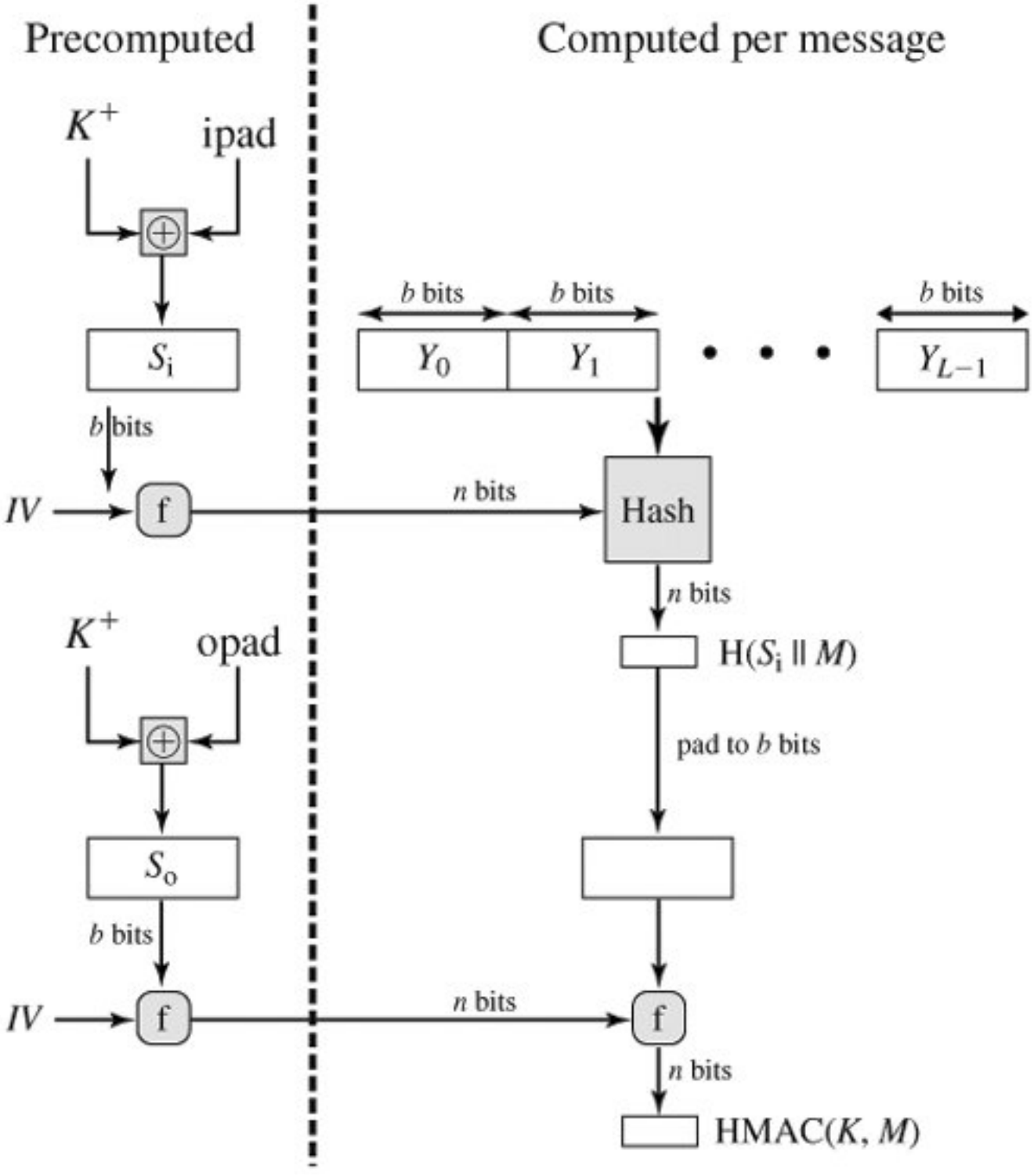
f(IV, ($K^+ \oplus$ ipad))

f(IV, ($K^+ \oplus$ opad))

where f(cv, block) is the compression function for the hash function, which takes as arguments a chaining variable of n bits and a block of b bits and produces a chaining variable of n bits. These quantities only need to be computed initially and every time the key changes. In effect, the precomputed quantities substitute for the initial value (IV) in the hash function. With this implementation, only one additional instance of the compression function is added to the processing normally produced by the hash function. This more efficient implementation is especially worthwhile if most of the messages for which a MAC is computed are short.

Figure 12.11. Efficient Implementation of HMAC

(This item is displayed on page 371 in the print version)



Security of HMAC

The security of any MAC function based on an embedded hash function depends in some way on the cryptographic strength of the underlying hash function. The appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC.

The security of a MAC function is generally expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message-MAC pairs created with the same key. In essence, it is proved in [BELL96a] that for a given level of effort (time, message-MAC pairs) on messages generated by a legitimate user and seen by the attacker, the probability of successful attack on HMAC is equivalent to one of the following attacks on the embedded hash function:

1. The attacker is able to compute an output of the compression function even with an IV that is random, secret, and unknown to the attacker.

2. The attacker finds collisions in the hash function even when the IV is random and secret.

In the first attack, we can view the compression function as equivalent to the hash function applied to a message consisting of a single b-bit block. For this attack, the IV of the hash function is replaced by a secret, random value of n bits. An attack on this hash function requires either a brute-force attack on the key, which is a level of effort on the order of 2^n or a birthday attack, which is a special case of the second attack, discussed next.

In the second attack, the attacker is looking for two messages M and M' that produce the same hash: $H(M) = H(M')$. This is the birthday attack discussed in Chapter 11. We have shown that this requires a level of effort of $2^{n/2}$ for a hash length of n. On this basis, the security of MD5 is called into question, because a level of effort of 2^{64} looks feasible with today's technology. Does this mean that a 128-bit hash function such as MD5 is unsuitable for HMAC? The answer is no, because of the following argument. To attack MD5, the attacker can choose any set of messages and work on these off line on a dedicated computing facility to find a collision. Because the attacker knows the hash algorithm and the default IV, the attacker can generate the hash code for each of the messages that the attacker generates. However, when attacking HMAC, the attacker cannot generate message/code pairs off line because the attacker does not know K. Therefore, the attacker must observe a sequence of messages generated by HMAC under the same key and perform the attack on these known messages. For a hash code length of 128 bits, this requires 2^{64} observed blocks (2^{72}bits) generated using the same key. On a 1-Gbps link, one would need to observe a continuous stream of messages with no change in key for about 150,000 years in order to succeed. Thus, if speed is a concern, it is fully acceptable to use MD5 rather than SHA-1 as the embedded hash function for HMAC.

12.4. CMAC

The Data Authentication Algorithm defined in FIPS PUB 113, also known as the CBC-MAC (cipher block chaining message authentication code), is described in Chapter 11. This cipher-based MAC has been widely adopted in government and industry. [BELL00] demonstrated that this MAC is secure under a reasonable set of security criteria, with the following restriction. Only messages of one fixed length of mn bits are processed, where n is the cipher block size and m is a fixed positive integer. As a simple example, notice that given the CBC MAC of a one-block message X, say $T = \text{MAC}(K, X)$, the adversary immediately knows the CBC MAC for the two-block message $X||(X \oplus T)$since this is once again T.

Black and Rogaway [BLAC00] demonstrated that this limitation could be overcome using three keys: one key of length k to be used at each step of the cipher block chaining and two keys of length n, where k is the key length and n is the cipher block length. This proposed construction was refined by Iwata and Kurosawa so that the two n-bit keys could be derived from the encryption key, rather than being provided separately [IWAT03]. This refinement has been adopted by NIST cipher-based message authentication code (CMAC) mode of operation, for use with AES and triple DES. It is specified in NIST Special Publication 800-38B.

First, let us consider the operation of CMAC when the message is an integer multiple n of the cipher block length b. For AES, $b = 128$ and for triple DES, $b = 64$. The message is divided into n blocks, M_1, $M_2,\ldots,M_n$. The algorithm makes use of a k-bit encryption key K and an n-bit constant K_1. For AES, the key size k is 128, 192, or 256 bits; for triple DES, the key size is 112 or 168 bits. CMAC is calculated as follows (Figure 12.12a):

$$C_1 = \text{E}(K, M_1)$$

$$C_2 = \text{E}(K, [M_2 \oplus C_1])$$

$$C_3 = \text{E}(K, [M_3 \oplus C_2])$$

.

.

.

$$C_n = \text{E}(K, [M_n \oplus C_{n1} \oplus K_1])$$

$$T = \text{MSB}_{Tlen}(C_n)$$

where

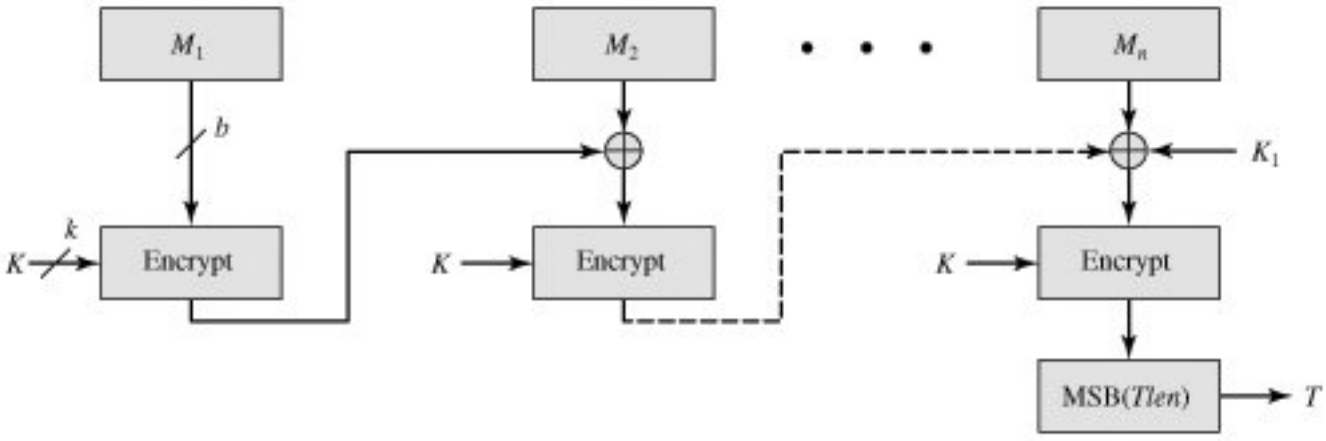
T = message authentication code, also referred to as the tag

$Tlen$ = bit length of T

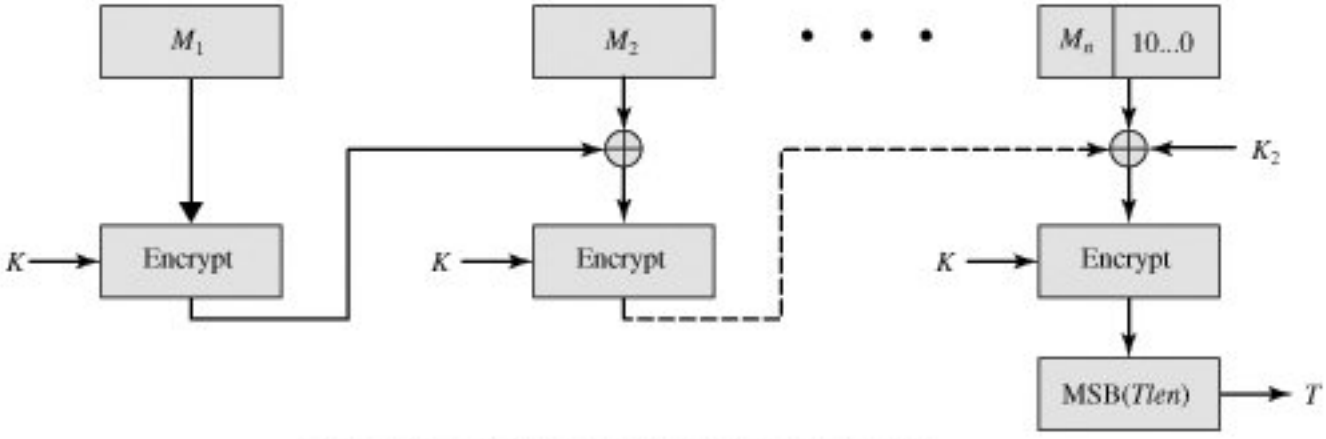
$MSB_s(X)$ = the s leftmost bits of the bit string X

Figure 12.12. Cipher-Based Message Authentication Code (CMAC)

[View full size image]



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

If the message is not an integer multiple of the cipher block length, then the final block is padded to the right (least significant bits) with a 1 and as many 0s as necessary so that the final block is also of length b. The CMAC operation then proceeds as before, except that a different n-bit key K_2 is used instead of K_1.

The two n-bit keys are derived from the k-bit encryption key as follows:

$L = E(K, 0^n)$

$K_1 = L \cdot x$

$K_2 = L \cdot x^2 = (L \cdot x) \cdot x$

where multiplication ($\cdot$) is done in the finite field (2^n) and x and x^2 are first and second order polynomials that are elements of GF(2^n) Thus the binary representation of x consists of n - 2 zeros followed by 10; the binary representation of x^2 consists of n - 3 zeros followed by 100. The finite field is defined with respect to an irreducible polynomial that is lexicographically first among all such polynomials with the minimum possible number of nonzero terms. For the two approved block sizes, the polynomials are and $x^{64} + x^4 + x^3 + x + 1$ and $x^{128} + x^7 + x^2 + x + 1$.

To generate K_1 and K_2 the block cipher is applied to the block that consists entirely of 0 bits. The first subkey is derived from the resulting cipher text by a left shift of one bit, and, conditionally, by XORing a constant that depends on the block size. The second subkey is derived in the same manner from the first subkey. This property of finite fields of the form GF(2^n)was explained in the discussion of MixColumns in Chapter 5.

12.5. Recommended Reading and Web Sites

[GILB03] examines the security of SHA-256 through SHA-512. Overviews of HMAC can be found in [BELL96a] and [BELL96b].

BELL96a Bellare, M.; Canetti, R.; and Krawczyk, H. "Keying Hash Functions for Message Authentication." *Proceedings, CRYPTO '96*, August 1996; published by Springer-Verlag. An expanded version is available at **http://www-cse.ucsd.edu/users/mihir**.

BELL96b Bellare, M.; Canetti, R.; and Krawczyk, H. "The HMAC Construction." *CryptoBytes*, Spring 1996.

GILB03 Gilbert, H. and Handschuh, H. "Security Analysis of SHA-256 and Sisters." "*Proceedings, CRYPTO '03*, 2003; published by Springer-Verlag.

Recommended Web Sites

- **NIST Secure Hashing Page:** SHA FIPS and related documents
- **Whirlpool:** Range of information on Whirlpool
- **Block cipher modes of operation:** NIST page with full information on CMAC

12.6. Key Terms, Review Questions, and Problems

Key Terms

big endian

CMAC

compression function

HMAC

little endian

MD4

MD5

SHA-1

SHA-256

SHA-384

SHA-512

Whirlpool

Review Questions

12.1 What is the difference between little-endian and big-endian format?

12.2 What basic arithmetical and logical functions are used in SHA?

12.3 What basic arithmetical and logical functions are used in Whirlpool?

12.4 Why has there been an interest in developing a message authentication code derived from a cryptographic hash function as opposed to one derived from a symmetric cipher?

12.5 What changes in HMAC are required in order to replace one underlying hash function with another?

Problems

12.1 In Figure 12.4, it is assumed that an array of 80 64-bit words is available to store the values of W_t so that they can be precomputed at the beginning of the processing of a block. Now assume that space is at a premium. As an alternative, consider the use of a 16-word circular buffer that is initially loaded with W_0 through W_{15} Design an algorithm that, for each step *t*, computes the required input value W_t

12.2 For SHA-512, show the equations for the values of W_{16},W_{17}, W_{18}, W_{19}

12.3 Suppose a_1 a_2 a_3 a_4 are the 4 bytes in a 32-bit word. Each a_i can be viewed as an integer in the range 0 to 255, represented in binary. In a big-endian architecture, this word represents the integer

$$a_1 2^{24} + a_2 2^{16} + a_3 2^8 + a_4$$

In a little-endian architecture, this word represents the integer

$$a_4 2^{24} + a_3 2^{16} + a_2 2^8 + a_1$$

a.

Some hash functions, such as MD5, assume a little-endian architecture. It is important that the message digest be independent of the underlying architecture. Therefore, to perform the modulo 2 addition operation of MD5 or RIPEMD-160 on a big-endian architecture, an adjustment must be made. Suppose X = x_1 x_2 x_3 x_4 and Y = y_1 y_2 y_3 y_4. Show how the MD5 addition operation (X + Y) would be carried out on a big-endian machine.

b.

SHA assumes a big-endian architecture. Show how the operation (X + Y) for SHA would be carried out on a little-endian machine.

12.4 This problem introduces a hash function similar in spirit to SHA that operates on letters instead of binary data. It is called the *toy tetragraph hash* (tth).[5] Given a message consisting of a sequence of letters, tth produces a hash value consisting of four letters. First, tth divides the message into blocks of 16 letters, ignoring spaces, punctuation, and capitalization. If the message length is not divisible by 16, it is padded out with nulls. A four-number running total is maintained that starts out with the value (0, 0, 0, 0); this is input to the compression function for processing the first block. The compression function consists of two rounds. **Round 1:** Get the next block of text and arrange it as a row-wise 4 x 4 block of text and convert it to numbers (A = 0, B = 1, etc.). For example, for the block ABCDEFGHIJKLMNOP, we have:

[5] I thank William K. Mason, of the magazine staff of *The Cryptogram*, for providing this example.

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Then, add each column mod 26 and add the result to the running total, mod 26. In this example, the running total is (24, 2, 6, 10). **Round 2:** Using the matrix from round 1, rotate the first row left by 1, second row left by 2, third row left by 3, and reverse the order of the fourth row. In our example:

B	C	D	A
G	H	E	F
L	I	J	K
P	O	N	M

1	2	3	0
6	7	4	5
11	8	9	10

15	14	13	12

Now, add each column mod 26 and add the result to the running total. The new running total is (5, 7, 9, 11). This running total is now the input into the first round of the compression function for the next block of text. After the final block is processed, convert the final running total to letters. For example, if the message is ABCDEFGHIJKLMNOP, then the hash is FHJL.

a.

Draw figures comparable to Figures 12.1 and 12.2 to depict the overall tth logic and the compression function logic.

b.

Calculate the hash function for the 48-letter message "I leave twenty million dollars to my friendly cousin Bill."

c.

To demonstrate the weakness of tth, find a 48-letter block that produces the same hash as that just derived. *Hint:* Use lots of A's.

12.5 Develop a table similar to Table 4.8 for GF(2^8) with $m(x) = x^8 + x^4 + x^3 + x^2 + 1$.

12.6 Show the E and E^{-1} mini-boxes in Table 12.3 in the traditional S-box square matrix format, such as that of Table 5.4.

12.7 Verify that Figure 12.9 is a valid implementation of the S-box shown in Table 12.3a. Do this by showing the calculations involved for three input values: 00, 55, 1E.

12.8 Provide a Boolean expression that defines the S-box functionality of Figure 12.9.

12.9 Whirlpool makes use of the construction $H_i = E(H_{i-1}, M_i) \oplus H_{i-1} \oplus M_{i-1}$ Another construction that was shown by Preneel to be secure is $H_i = E(H_{i-1}, M_i) \oplus M_i$ Now notice that the key schedule for Whirlpool resembles encryption of the cipher key under a pseudo-key defined by the round constants, so that the core of the hashing process could be formally viewed as two interacting encryption $E(H_{i-1}, M_i)$lines.

Consider the encryption We could write the final round key for this block as $K_{10} = E(RC, H_{i-1})$. Now show that the two hash constructions are essentially equivalent because of the way that the key schedule is defined.

12.10 At the beginning of Section 12.4, it was noted that given the CBC MAC of a one-block message X, say $T = MAC(K, X)$,the adversary immediately knows the CBC MAC for the two-block message $X \| (X \oplus T)$since this is once again T. Justify this statement.

12.11 In this problem, we demonstrate that for CMAC, a variant that XORs the second key after applying the final encryption doesn't work. Let us consider this for the case of the message being an integer multiple of the block size. Then the variant can be expressed as VMAC(K,M) = CBC(K,M) $\oplus$ K_1. Now suppose an adversary is able to ask for the MACs of three messages: the message $\mathbf{0} = 0^n$, where n is the cipher block size; the message $\mathbf{1} = 1^n$ and the message $\mathbf{1}||\mathbf{0}$. As a result of these three queries the adversary gets T_0 = CBC(K,$\mathbf{0}$) $\oplus$ K_1; T_1 = CBC(K,$\mathbf{1}$) $\oplus$ K_1, and T_2 = CBC(K,[CBC(K,1)]) $\oplus$ K_1. Show that the adversary can compute the correct MAC for the (unqueried) message $\mathbf{0}||(T_0 \oplus T_1)$.

12.12 In the discussion of subkey generation in CMAC, it states that the block cipher is applied to the block that consists entirely of 0 bits. The first subkey is derived from the resulting string by a left shift of one bit, and, conditionally, by XORing a constant that depends on the block size. The second subkey is derived in the same manner from the first subkey.

a.

What constants are needed for block sizes of 64 and 128 bits?

b.

Explain how the left shift and XOR accomplishes the desired result.

PREV NEXT

Chapter 13. Digital Signatures and Authentication Protocols

To guard against the baneful influence exerted by strangers is therefore an elementary dictate of savage prudence. Hence before strangers are allowed to enter a district, or at least before they are permitted to mingle freely with the inhabitants, certain ceremonies are often performed by the natives of the country for the purpose of disarming the strangers of their magical powers, or of disinfecting, so to speak, the tainted atmosphere by which they are supposed to be surrounded.

The Golden Bough, Sir James George Frazer

Key Points

- A digital signature is an authentication mechanism that enables the creator of a message to attach a code that acts as a signature. The signature is formed by taking the hash of the message and encrypting the message with the creator's private key. The signature guarantees the source and integrity of the message.
- Mutual authentication protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys.
- In one-way authentication, the recipient wants some assurance that a message is from the alleged sender.
- The digital signature standard (DSS) is an NIST standard that uses the secure hash algorithm (SHA).

The most important development from the work on public-key cryptography is the digital signature. The digital signature provides a set of security capabilities that would be difficult to implement in any other way. We begin this chapter with an overview of digital signatures. Then we look at authentication protocols, many of which depend on the use of the digital signature. Finally, we introduce the Digital Signature Standard (DSS).

PREV NEXT

13.1. Digital Signatures

Requirements

Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. Several forms of dispute between the two are possible.

For example, suppose that John sends an authenticated message to Mary, using one of the schemes of Figure 11.4. Consider the following disputes that could arise:

1. Mary may forge a different message and claim that it came from John. Mary would simply have to create a message and append an authentication code using the key that John and Mary share.

2. John can deny sending the message. Because it is possible for Mary to forge a message, there is no way to prove that John did in fact send the message.

Both scenarios are of legitimate concern. Here is an example of the first scenario: An electronic funds transfer takes place, and the receiver increases the amount of funds transferred and claims that the larger amount had arrived from the sender. An example of the second scenario is that an electronic mail message contains instructions to a stockbroker for a transaction that subsequently turns out badly. The sender pretends that the message was never sent.

In situations where there is not complete trust between sender and receiver, something more than authentication is needed. The most attractive solution to this problem is the digital signature. The digital signature is analogous to the handwritten signature. It must have the following properties:

- It must verify the author and the date and time of the signature.
- It must to authenticate the contents at the time of the signature.
- It must be verifiable by third parties, to resolve disputes.

Thus, the digital signature function includes the authentication function.

On the basis of these properties, we can formulate the following requirements for a digital signature:

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information unique to the sender, to prevent both forgery and denial.
- It must be relatively easy to produce the digital signature.

- It must be relatively easy to recognize and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to retain a copy of the digital signature in storage.

A secure hash function, embedded in a scheme such as that of Figure 11.5c or d, satisfies these requirements.

A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated.

Direct Digital Signature

The direct digital signature involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source. A digital signature may be formed by encrypting the entire message with the sender's private key (Figure 11.1c) or by encrypting a hash code of the message with the sender's private key (Figure 11.5c).

Confidentiality can be provided by further encrypting the entire message plus signature with either the receiver's public key (public-key encryption) or a shared secret key (symmetric encryption); for example, see Figures 11.1d and 11.5d. Note that it is important to perform the signature function first and then an outer confidentiality function. In case of dispute, some third party must view the message and its signature. If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message. However, if the signature is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.

All direct schemes described so far share a common weakness. The validity of the scheme depends on the security of the sender's private key. If a sender later wishes to deny sending a particular message, the sender can claim that the private key was lost or stolen and that someone else forged his or her signature. Administrative controls relating to the security of private keys can be employed to thwart or at least weaken this ploy, but the threat is still there, at least to some degree. One example is to require every signed message to include a timestamp (date and time) and to require prompt reporting of compromised keys to a central authority.

Another threat is that some private key might actually be stolen from X at time T. The opponent can then send a message signed with X's signature and stamped with a time before or equal to T.

Arbitrated Digital Signature

The problems associated with direct digital signatures can be addressed by using an arbiter.

As with direct signature schemes, there is a variety of arbitrated signature schemes. In general terms, they all operate as follows. Every signed message from a sender X to a receiver Y goes first to an arbiter A, who subjects the message and its signature to a number of tests to check its origin and content. The message is then dated and sent to Y with an indication that it has been verified to the satisfaction of the arbiter. The presence of A solves the problem faced by direct signature schemes: that X might disown the message.

The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly. The use of a trusted system, described in Chapter 20, might satisfy this requirement.

Table 13.1, based on scenarios described in [AKL83] and [MITC92], gives several examples of arbitrated digital signatures.[1] In the first, symmetric encryption is used. It is assumed that the sender X and the arbiter A share a secret key K_{xa} and that A and Y share secret key K_{ay}. X constructs a message M and computes its hash value H(M). Then X transmits the message plus a signature to A. The signature consists of an identifier ID_X of X plus the hash value, all encrypted using K_{xa}. A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y, encrypted with K_{ay}. The message includes ID_X, the original message from X, the signature, and a timestamp. Y can decrypt this to recover the message and the signature. The timestamp informs Y that this message is timely and not a replay. Y can store M and the signature. In case of dispute, Y, who claims to have received M from X, sends the following message to A:

[1] The following format is used. A communication step in which P sends a message M to Q is represented as P $\rightarrow$ Q: M.

$$E(K_{ay}, [ID_X||M||E(K_{xa}, [ID_X||H(M)])])$$

Table 13.1. Arbitrated Digital Signature Techniques

(1) X $\rightarrow$ A: $M\|\|E(K_{xa}, [ID_X\|\|H(M)])$	
(2) A $\rightarrow$ Y: $E(K_{ay}, [ID_X\|\|M\|\|E(K_{xa}, [ID_X\|\|H(M)])\|\|T])$	
(a) Conventional Encryption, Arbiter Sees Message	
(1) X $\rightarrow$ A: $ID_X\|\|E(K_{xy}, M)\|\|E(K_{xa}, [ID_X\|\|H(E(K_{xy}, M))])$	
(2) A $\rightarrow$ Y: $E(K_{ay},[ID_X\|\|E(K_{xy}, M)])\|\|E(K_{xa}, [ID_X\|\|H(E(K_{xy}, M))\|\|T])$	
(b) Conventional Encryption, Arbiter Does Not See Message	
(1) X $\rightarrow$ A: $ID_X\|\|E(PR_x, [ID_X\|\|E(PU_y, E(PR_x, M))])$	
(2) A $\rightarrow$ Y: $E(PR_a, [ID_X\|\|E(PU_y, E(PR_x, M))\|\|T])$	
(c) Public-Key Encryption, Arbiter Does Not See Message	
Notation:	
X	= sender
Y	= recipient

A	= Arbiter
M	= message
T	= timestamp

The arbiter uses K_{ay} to recover ID_X, M, and the signature, and then uses K_{xa} to decrypt the signature and verify the hash code. In this scheme, Y cannot directly check X's signature; the signature is there solely to settle disputes. Y considers the message from X authentic because it comes through A. In this scenario, both sides must have a high degree of trust in A:

- X must trust A not to reveal K_{xa} and not to generate false signatures of the form $E(K_{xa}, [ID_X||H(M)])$.
- Y must trust A to send $E(K_{ay}, [ID_X||M||E(K_{xa}, [ID_X||H(M)])||T])$ only if the hash value is correct and the signature was generated by X.
- Both sides must trust A to resolve disputes fairly.

If the arbiter does live up to this trust, then X is assured that no one can forge his signature and Y is assured that X cannot disavow his signature.

The preceding scenario also implies that A is able to read messages from X to Y and, indeed, that any eavesdropper is able to do so. Table 13.1b shows a scenario that provides the arbitration as before but also assures confidentiality. In this case it is assumed that X and Y share the secret key K_{xy}. Now, X transmits an identifier, a copy of the message encrypted with K_{xy}, and a signature to A. The signature consists of the identifier plus the hash value of the encrypted message, all encrypted using K_{xa}. As before, A decrypts the signature and checks the hash value to validate the message. In this case, A is working only with the encrypted version of the message and is prevented from reading it. A then transmits everything that it received from X, plus a timestamp, all encrypted with K_{ay}, to Y.

Although unable to read the message, the arbiter is still in a position to prevent fraud on the part of either X or Y. A remaining problem, one shared with the first scenario, is that the arbiter could form an alliance with the sender to deny a signed message, or with the receiver to forge the sender's signature.

All the problems just discussed can be resolved by going to a public-key scheme, one version of which is shown in Table 13.1c. In this case, X double encrypts a message M first with X's private key, PR_x and then with Y's public key, PU_y. This is a signed, secret version of the message. This signed message, together with X's identifier, is encrypted again with PR_x and, together with ID_X, is sent to A. The inner, double-encrypted message is secure from the arbiter (and everyone else except Y). However, A can decrypt the outer encryption to assure that the message must have come from X (because only X has PR_x). A checks to make sure that X's private/public key pair is still valid and, if so, verifies the message. Then A transmits a message to Y, encrypted with PR_a. The message includes ID_X, the double-encrypted message, and a timestamp.

This scheme has a number of advantages over the preceding two schemes. First, no information is shared among the parties before communication, preventing alliances to defraud. Second, no incorrectly dated message can be sent, even if PR_x is compromised, assuming that PR_a is not compromised. Finally,

the content of the message from X to Y is secret from A and anyone else. However, this final scheme involves encryption of the message twice with a public-key algorithm. We discuss more practical approaches subsequently.

PREV NEXT

13.2. Authentication Protocols

The basic tools described in Chapter 11 are used in a variety of applications, including the digital signature discussed in Section 13.1. Other uses are numerous and growing. In this section, we focus on two general areas (mutual authentication and one-way authentication) and examine some of the implications of authentication techniques in both.

Mutual Authentication

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. This topic was examined in Section 7.3 (symmetric techniques) and Section 10.1 (public-key techniques). There, the focus was key distribution. We return to this topic here to consider the wider implications of authentication.

Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness. To prevent masquerade and to prevent compromise of session keys, essential identification and session key information must be communicated in encrypted form. This requires the prior existence of secret or public keys that can be used for this purpose. The second issue, timeliness, is important because of the threat of message replays. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

[GONG93] lists the following examples of replay attacks:

- **Simple replay:** The opponent simply copies a message and replays it later.
- **Repetition that can be logged:** An opponent can replay a timestamped message within the valid time window.
- **Repetition that cannot be detected:** This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only the replay message arrives.
- **Backward replay without modification:** This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order. The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. Instead, one of the following two general approaches is used:

- **Timestamps:** Party A accepts a message as fresh only if the message contains a timestamp that, in A's judgment, is close enough to A's knowledge of current time. This approach requires that clocks among the various participants be synchronized.
- **Challenge/response:** Party A, expecting a fresh message from B, first sends B a nonce (challenge) and requires that the subsequent message (response) received from B contain the

correct nonce value.

It can be argued (e.g., [LAM92a]) that the timestamp approach should not be used for connection-oriented applications because of the inherent difficulties with this technique. First, some sort of protocol is needed to maintain synchronization among the various processor clocks. This protocol must be both fault tolerant, to cope with network errors, and secure, to cope with hostile attacks. Second, the opportunity for a successful attack will arise if there is a temporary loss of synchronization resulting from a fault in the clock mechanism of one of the parties. Finally, because of the variable and unpredictable nature of network delays, distributed clocks cannot be expected to maintain precise synchronization. Therefore, any timestamp-based procedure must allow for a window of time sufficiently large to accommodate network delays yet sufficiently small to minimize the opportunity for attack.

On the other hand, the challenge-response approach is unsuitable for a connectionless type of application because it requires the overhead of a handshake before any connectionless transmission, effectively negating the chief characteristic of a connectionless transaction. For such applications, reliance on some sort of secure time server and a consistent attempt by each party to keep its clocks in synchronization may be the best approach (e.g., [LAM92b]).

Symmetric Encryption Approaches

As was discussed in Section 7.3, a two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment. In general, this strategy involves the use of a trusted key distribution center (KDC). Each party in the network shares a secret key, known as a master key, with the KDC. The KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys, and for distributing those keys using the master keys to protect the distribution. This approach is quite common. As an example, we look at the Kerberos system in Chapter 14. The discussion in this subsection is relevant to an understanding of the Kerberos mechanisms.

Figure 7.9 illustrates a proposal initially put forth by Needham and Schroeder [NEED78] for secret key distribution using a KDC that, as was mentioned in Chapter 7, includes authentication features. The protocol can be summarized as follows:

1. A $\longrightarrow$ KDC: $ID_A||ID_B||N_1$
2. KDC $\longrightarrow$ A: $E(K_a, [K_s||ID_B||N_1||E(K_b, [K_s||ID_A])])$
3. A $\longrightarrow$ B: $E(K_b, [K_s||ID_A])$
4. A $\longrightarrow$ A: $E(K_s, N_2)$
5. A $\longrightarrow$ B: $E(K_s, f(N_2))$

Secret keys K_a and K_b are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key K_s to A and B. A securely acquires a new session key in step 2. The message in step 3 can be decrypted, and hence understood, only by B. Step 4 reflects B's knowledge of K_s, and step 5 assures B of A's knowledge of K_s and assures B that this is a fresh

message because of the use of the nonce N_2. Recall from our discussion in Chapter 7 that the purpose of steps 4 and 5 is to prevent a certain type of replay attack. In particular, if an opponent is able to capture the message in step 3 and replay it, this might in some fashion disrupt operations at B.

Despite the handshake of steps 4 and 5, the protocol is still vulnerable to a form of replay attack. Suppose that an opponent, X, has been able to compromise an old session key. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3. Nevertheless, it is a potential security risk. X can impersonate A and trick B into using the old key by simply replaying step 3. Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay. If X can intercept the handshake message, step 4, then it can impersonate A's response, step 5. From this point on, X can send bogus messages to B that appear to B to come from A using an authenticated session key.

Denning [DENN81, DENN82] proposes to overcome this weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3. Her proposal assumes that the master keys, K_a and K_b are secure, and it consists of the following steps:

---

1. A $\longrightarrow$ KDC: $ID_A||ID_B$
2. KDC $\longrightarrow$ A: $E(K_a, [K_s||ID_B||T||E(K_b, [K_s||ID_A||T])])$
3. A $\longrightarrow$ B: $E(K_b, [K_s||ID_A||T])$
4. B $\longrightarrow$ A: $E(K_s, N_1)$
5. A $\longrightarrow$ B: $E(K_s, f(N_1))$

T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange. A and B can verify timeliness by checking that

$$|\text{Clock } T| < \Delta t_1 + \Delta t_2$$

where Δt_1 is the estimated normal discrepancy between the KDC's clock and the local clock (at A or B) and Δt_2 is the expected network delay time. Each node can set its clock against some standard reference source. Because the timestamp T is encrypted using the secure master keys, an opponent, even with knowledge of an old session key, cannot succeed because a replay of step 3 will be detected by B as untimely.

A final point: Steps 4 and 5 were not included in the original presentation [DENN81] but were added later [DENN82]. These steps confirm the receipt of the session key at B.

The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network. [GONG92] points out a risk involved. The risk is based on the fact that the distributed clocks can become unsynchronized as a result of sabotage on or faults in the clocks or the synchronization mechanism.[2] The problem occurs when a sender's clock is

ahead of the intended recipient's clock. In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site. This replay could cause unexpected results. Gong refers to such attacks as **suppress-replay attacks**.

[2] Such things can and do happen. In recent years, flawed chips were used in a number of computers and other electronic systems to track the time and date. The chips had a tendency to skip forward one day [NEUM90].

One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC's clock. The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonces. This latter alternative is not vulnerable to a suppress-replay attack because the nonces the recipient will choose in the future are unpredictable to the sender. The Needham/Schroeder protocol relies on nonces only but, as we have seen, has other vulnerabilities.

In [KEHN92], an attempt is made to respond to the concerns about suppress-replay attacks and at the same time fix the problems in the Needham/Schroeder protocol. Subsequently, an inconsistency in this latter protocol was noted and an improved strategy was presented in [NEUM93a].[3] The protocol is as follows:

[3] It really is hard to get these things right.

1. A $\rightarrow$ B: $ID_A||N_a$
2. B $\rightarrow$ KDC: $ID_B||N_b||E(K_b, [ID_A||N_a||T_b])$
3. KDC $\rightarrow$ A: $E(K_a, [ID_B||N_a||K_s||T_b])||E(K_b,[ID_A||K_s||T_b])||N_b$
4. A $\rightarrow$ B: $E(K_b, [ID_A||K_s||T_b])||E(K_s, N_b)$

Let us follow this exchange step by step.

1. A initiates the authentication exchange by generating a nonce, N_a, and sending that plus its identifier to B in plaintext. This nonce will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.

2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce, N_b This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness. B's message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDC to issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.

3. The KDC passes on to A B's nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a "ticket" that can be used by A for subsequent authentications, as will be seen. The KDC also sends to A a block encrypted with the secret key shared by A and the KDC. This block verifies that B has received A's initial message (ID_B) and that this is a timely message and not a replay (N_a) and it provides A with a session key (K_s) and the time limit on its use (T_b).

4. A transmits the ticket to B, together with the B's nonce, the latter encrypted with the session key. The ticket provides B with the secret key that is used to decrypt $E(K_s, N_b)$ to recover the nonce.

   The fact that B's nonce is encrypted with the session key authenticates that the message came from A and is not a replay.

This protocol provides an effective, secure means for A and B to establish a session with a secure session key. Furthermore, the protocol leaves A in possession of a key that can be used for subsequent authentication to B, avoiding the need to contact the authentication server repeatedly. Suppose that A and B establish a session using the aforementioned protocol and then conclude that session. Subsequently, but within the time limit established by the protocol, A desires a new session with B. The following protocol ensues:

1. A $\longrightarrow$ B: $E(K_b, [ID_A||K_s||T_b])||N'_a$
2. B $\longrightarrow$ A: $N'_b||E(K_s, N'_a)$
3. A $\longrightarrow$ B: $E(K_s, N'_b)$

When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces N'_a and N'_b assure each party that there is no replay attack.

In all the foregoing, the time specified in T_b is a time relative to B's clock. Thus, this timestamp does not require synchronized clocks because B checks only self-generated timestamps.

Public-Key Encryption Approaches

In Chapter 10, we presented one approach to the use of public-key encryption for the purpose of session key distribution (Figure 10.6). This protocol assumes that each of the two parties is in possession of the current public key of the other. It may not be practical to require this assumption.

A protocol using timestamps is provided in [DENN81]:

1. A $\longrightarrow$ AS: $ID_A||ID_B$
2. AS $\longrightarrow$ A: $E(PR_{as}, [ID_A||PU_a||T])||E(PR_{as}, [ID_B||PU_b||T])$

3. A → B: $E(PR_{as}, [ID_A||PU_a||T])||E(PR_{as}, [ID_B||PU_b||T])$
$||E(PU_b, E(PR_a, [K_s||T]))$

In this case, the central system is referred to as an authentication server (AS), because it is not actually responsible for secret key distribution. Rather, the AS provides public-key certificates. The session key is chosen and encrypted by A; hence, there is no risk of exposure by the AS. The timestamps protect against replays of compromised keys.

This protocol is compact but, as before, requires synchronization of clocks. Another approach, proposed by Woo and Lam [WOO92a], makes use of nonces. The protocol consists of the following steps:

1. A → KDC: $ID_A||ID_B$
2. KDC → A: $E(PR_{auth}, [ID_B||PU_b])$
3. A → B: $E(PU_b, [N_a||ID_A])$
4. B → KDC: $ID_A||ID_B||E(PU_{auth}, N_a)$
5. KDC → B: $E(PR_{auth}, [ID_A||PU_a])||E(PU_b, E(PR_{auth}, [N_a||K_s||ID_B]))$
6. B → A: $E(PU_a, E(PR_{auth}, [(N_a||K_s||ID_B)||N_b]))$
7. A → B: $E(K_s, N_b)$

In step 1, A informs the KDC of its intention to establish a secure connection with B. The KDC returns to A a copy of B's public-key certificate (step 2). Using B's public key, A informs B of its desire to communicate and sends a nonce N_a (step 3). In step 4, B asks the KDC for A's public-key certificate and requests a session key; B includes A's nonce so that the KDC can stamp the session key with that nonce. The nonce is protected using the KDC's public key. In step 5, the KDC returns to B a copy of A's public-key certificate, plus the information $\{N_a, K_s, ID_B\}$. This information basically says that K_s is a secret key generated by the KDC on behalf of B and tied to N_a; the binding of K_s and N_a will assure A that K_s is fresh. This triple is encrypted, using the KDC's private key, to allow B to verify that the triple is in fact from the KDC. It is also encrypted using B's public key, so that no other entity may use the triple in an attempt to establish a fraudulent connection with A. In step 6, the triple $\{N_a, K_s, ID_B\}$, still encrypted with the KDC's private key, is relayed to A, together with a nonce N_b generated by B. All the foregoing are encrypted using A's public key. A retrieves the session key K_s and uses it to encrypt N_b and return it to B. This last message assures B of A's knowledge of the session key.

This seems to be a secure protocol that takes into account the various attacks. However, the authors themselves spotted a flaw and submitted a revised version of the algorithm in [WOO92b]:

---

1. A → KDC: $ID_A||ID_B$

2. KDC ⟶ A: $E(PR_{auth}, [ID_B||PU_b])$

3. A ⟶ B: $E(PU_b, [N_a||ID_A])$

4. B ⟶ KDC: $ID_A||ID_B||E(PU_{auth}, N_a)$

5. KDC ⟶ B: $E(PR_{auth}, [ID_A||PU_a])||E(PU_b, E(PR_{auth}, [N_a||K_s||ID_A||ID_B])$

6. B ⟶ A: $E(PU_a, E(PR_{auth}, [(N_a||K_s||ID_A||ID_B)||N_b]))$

7. A ⟶ B: $E(K_s, N_b)$

The identifier of A, ID_A, is added to the set of items encrypted with the KDC's private key in steps 5 and 6. This binds the session key K_s to the identities of the two parties that will be engaged in the session. This inclusion of ID_A accounts for the fact that the nonce value N_a is considered unique only among all nonces generated by A, not among all nonces generated by all parties. Thus, it is the pair $\{ID_A, N_a\}$ that uniquely identifies the connection request of A.

In both this example and the protocols described earlier, protocols that appeared secure were revised after additional analysis. These examples highlight the difficulty of getting things right in the area of authentication.

One-Way Authentication

One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be online at the same time. Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.

The "envelope" or header of the e-mail message must be in the clear, so that the message can be handled by the store-and-forward e-mail protocol, such as the Simple Mail Transfer Protocol (SMTP) or X.400. However, it is often desirable that the mail-handling protocol not require access to the plaintext form of the message, because that would require trusting the mail-handling mechanism. Accordingly, the e-mail message should be encrypted such that the mail-handling system is not in possession of the decryption key.

A second requirement is that of authentication. Typically, the recipient wants some assurance that the message is from the alleged sender.

Symmetric Encryption Approach

Using symmetric encryption, the decentralized key distribution scenario illustrated in Figure 7.11 is impractical. This scheme requires the sender to issue a request to the intended recipient, await a response that includes a session key, and only then send the message.

With some refinement, the KDC strategy illustrated in Figure 7.9 is a candidate for encrypted electronic mail. Because we wish to avoid requiring that the recipient (B) be on line at the same time as the sender (A), steps 4 and 5 must be eliminated. For a message with content M, the sequence is as follows:

1. A $\rightarrow$ KDC: $ID_A||ID_B||N_1$
2. KDC $\rightarrow$ A: $E(K_a, [K_s||ID_B||N_1||E(K_b, [K_s||ID_A])])$
3. A $\rightarrow$ B: $E(K_b, [K_s||ID_A])||E(K_s, M)$

This approach guarantees that only the intended recipient of a message will be able to read it. It also provides a level of authentication that the sender is A. As specified, the protocol does not protect against replays. Some measure of defense could be provided by including a timestamp with the message. However, because of the potential delays in the e-mail process, such timestamps may have limited usefulness.

Public-Key Encryption Approaches

We have already presented public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for confidentiality (Figure 11.1b), authentication (Figure 11.1c), or both (Figure 11.1d). These approaches require that either the sender know the recipient's public key (confidentiality) or the recipient know the sender's public key (authentication) or both (confidentiality plus authentication). In addition, the public-key algorithm must be applied once or twice to what may be a long message.

If confidentiality is the primary concern, then the following may be more efficient:

A $\rightarrow$ B: $E(PU_b, K_s)||E(K_s, M)$

In this case, the message is encrypted with a one-time secret key. A also encrypts this one-time key with B's public key. Only B will be able to use the corresponding private key to recover the one-time key and then use that key to decrypt the message. This scheme is more efficient than simply encrypting the entire message with B's public key.

If authentication is the primary concern, then a digital signature may suffice, as was illustrated in Figure 11.5c:

A $\rightarrow$ B: $M||E(PR_a, H(M))$

This method guarantees that A cannot later deny having sent the message. However, this technique is open to another kind of fraud. Bob composes a message to his boss Alice that contains an idea that will save the company money. He appends his digital signature and sends it into the e-mail system. Eventually, the message will get delivered to Alice's mailbox. But suppose that Max has heard of Bob's idea and gains access to the mail queue before delivery. He finds Bob's message, strips off his signature, appends his, and requeues the message to be delivered to Alice. Max gets credit for Bob's idea.

To counter such a scheme, both the message and signature can be encrypted with the recipient's public key:

A → B: $E(PU_b, [M||E(PR_a, H(M))])$

The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate, described in Chapter 10. Now we have

A → B: $M||E(PR_a, H(M))||E(PR_{as}, [T||ID_A||PU_a])$

In addition to the message, A sends B the signature, encrypted with A's private key, and A's certificate, encrypted with the private key of the authentication server. The recipient of the message first uses the certificate to obtain the sender's public key and verify that it is authentic and then uses the public key to verify the message itself. If confidentiality is required, then the entire message can be encrypted with B's public key. Alternatively, the entire message can be encrypted with a one-time secret key; the secret key is also transmitted, encrypted with B's public key. This approach is explored in Chapter 15.

PREV NEXT

13.3. Digital Signature Standard

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Hash Algorithm (SHA) described in Chapter 12 and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme. There was a further minor revision in 1996. In 2000, an expanded version of the standard was issued as FIPS 186-2. This latest version also incorporates digital signature algorithms based on RSA and on elliptic curve cryptography. In this section, we discuss the original DSS algorithm.

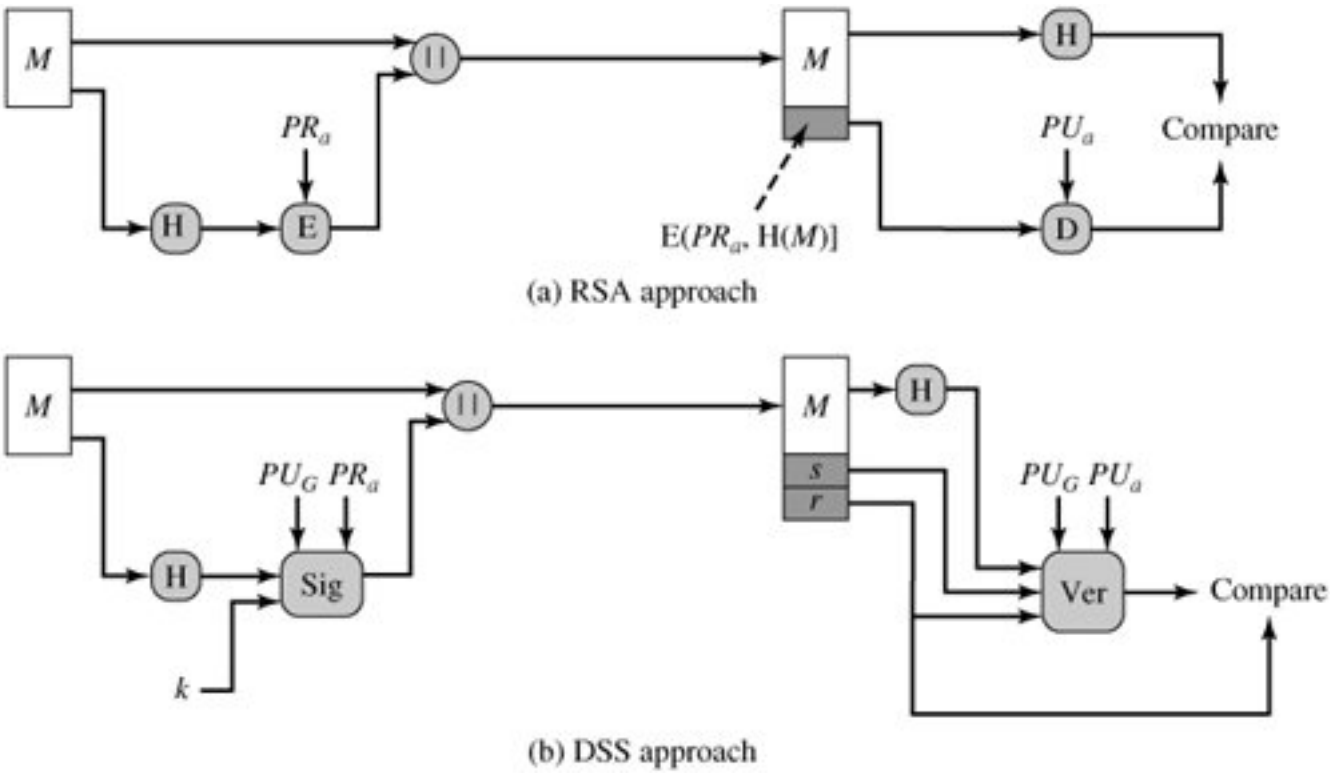
The DSS Approach

The DSS uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique.

Figure 13.1 contrasts the DSS approach for generating digital signatures to that used with RSA. In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.

Figure 13.1. Two Approaches to Digital Signatures

[View full size image]



The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number k generated for this particular signature. The signature function also depends on the sender's private key (PR_a)and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PU_G).[4] The result is a signature consisting of two components, labeled s and r.

[4] It is also possible to allow these additional parameters to vary with each user so that they are a part of a user's public key. In practice, it is more likely that a global public key will be used that is separate from each user's public key.

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender's public key (PU_a), which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component r if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

We turn now to the details of the algorithm.

The Digital Signature Algorithm

The DSA is based on the difficulty of computing discrete logarithms (see Chapter 8) and is based on schemes originally presented by ElGamal [ELGA85] and Schnorr [SCHN91].

Figure 13.2 summarizes the algorithm. There are three parameters that are public and can be common to a group of users. A 160-bit prime number q is chosen. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides (p 1). Finally, g is chosen to be of the form $h^{(p1)/q}$ mod p where h is an integer between 1 and (p 1) with the restriction that g must be greater than 1.[5]

[5] In number-theoretic terms, g is of order q mod p; see Chapter 8.

Figure 13.2. The Digital Signature Algorithm (DSA)

(This item is displayed on page 391 in the print version)

Global Public-Key Components	
p	prime number where $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and L a multiple of 64; i.e., bit length of between 512 and 1024 bits in increments of 64 bits

q	prime divisor of (p 1), where $2^{159} < q < 2^{160}$; i.e., bit length of 160 bits
g	= $h^{(p\ 1)/q}$ mod p, where h is any integer with $1 < h < (p\ 1)$ such that $h^{(p\ 1)/q}$ mod $p > 1$
User's Private Key	
x	random or pseudorandom integer with $0 < x < q$
User's Public Key	
y	= g^x mod p
User's Per-Message Secret Number	
k	= random or pseudorandom integer with $0 < k < q$
Signing	
r	= (g^k mod p) mod q
s	= [k^{-1} (H(M) + xr)] mod q
Signature = (r, s)	
Verifying	
w	= (s')$^{-1}$ mod q
u1	= [**H(M**')w] mod q
u2	=(r')w mod q
v	= [($g^{u\ 1}$ $y^{u\ 2}$) mod p] mod q
TEST: v = r'	
M	= message to be signed
H(M)	= hash of M using SHA-1
M', r', s'	= received versions of M, r, s

With these numbers in hand, each user selects a private key and generates a public key. The private key x must be a number from 1 to (q 1) and should be chosen randomly or pseudorandomly. The public key is calculated from the private key as $y = g^x$ mod p. The calculation of y given x is relatively straightforward. However, given the public key y, it is believed to be computationally infeasible to determine x, which is the discrete logarithm of y to the base g, mod p (see Chapter 8).

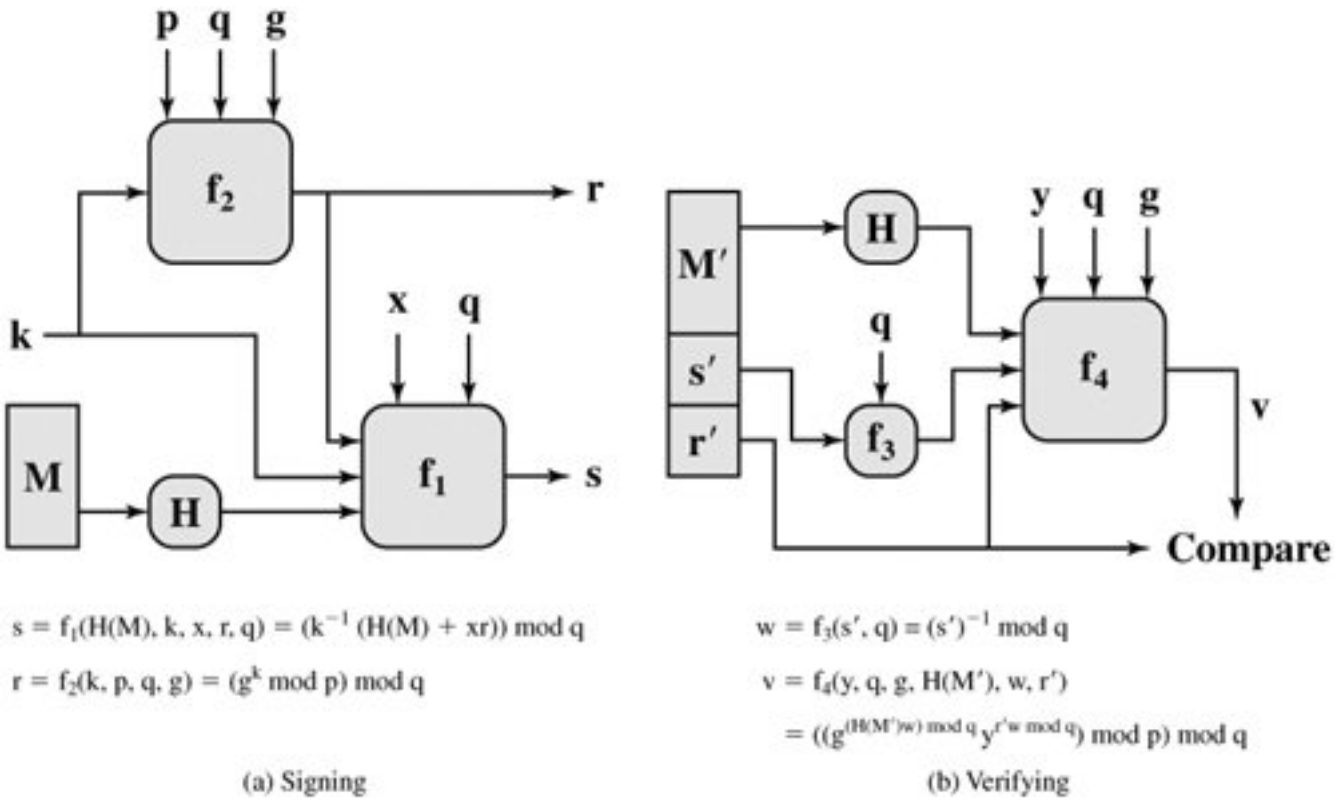
To create a signature, a user calculates two quantities, r and s, that are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message, H(M), and an additional integer k that should be generated randomly or pseudorandomly and be unique for each signing.

At the receiving end, verification is performed using the formulas shown in Figure 13.2. The receiver generates a quantity v that is a function of the public key components, the sender's public key, and the hash code of the incoming message. If this quantity matches the r component of the signature, then the signature is validated.

Figure 13.3 depicts the functions of signing and verifying.

Figure 13.3. DSS Signing and Verifying

[View full size image]



The structure of the algorithm, as revealed in Figure 13.3, is quite interesting. Note that the test at the end is on the value *r*, which does not depend on the message at all. Instead, *r* is a function of *k* and the three global public-key components. The multiplicative inverse of *k* (mod *q*) is passed to a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover *r* using the incoming message and signature, the public key of the user, and the global public key. It is certainly not obvious from Figure 13.2 or Figure 13.3 that such a scheme would work. A proof is provided at this book's Web site.

Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover *k* from *r* or to recover *x* from *s*.

Another point worth noting is that the only computationally demanding task in signature generation is the exponential calculation g^k mod *p*. Because this value does not depend on the message to be signed, it can be computed ahead of time. Indeed, a user could precalculate a number of values of *r* to be used to sign documents as needed. The only other somewhat demanding task is the determination of a multiplicative inverse, K^{-1}. Again, a number of these values can be precalculated.

13.4. Recommended Reading and Web Sites

[AKL83] is the classic paper on digital signatures and is still highly relevant. A more recent, and excellent, survey is [MITC92].

AKL83 Akl, S. "Digital Signatures: A Tutorial Survey." *Computer*, February 1983.

MITC92 Mitchell, C.; Piper, F. ; and Wild, P. "Digital Signatures." In [SIMM92a].

Recommended Web Sites

- **Digital Signatures:** NIST page with information on NIST-approved digital signature options

13.5. Key Terms, Review Questions, and Problems

Key Terms

arbiter

arbitrated digital signature

direct digital signature

digital signature

digital signature algorithm (DSA)

digital signature standard (DSS)

mutual authentication

nonce

one-way authentication

replay attack

suppress-replay attack

timestamp

Review Questions

13.1 List two disputes that can arise in the context of message authentication.

13.2 What are the properties a digital signature should have?

13.3 What requirements should a digital signature scheme satisfy?

13.4 What is the difference between direct and arbitrated digital signature?

13.5 In what order should the signature function and the confidentiality function be applied to a message, and why?

13.6 What are some threats associated with a direct digital signature scheme?

13.7 Give examples of replay attacks.

13.8 List three general approaches to dealing with replay attacks.

13.9 What is a suppress-replay attack?

Problems

13.1 Modify the digital signature techniques of Table 13.1a and b to enable the receiver to verify the signature.

13.2 Modify the digital signature technique of Table 13.1c to avoid triple encryption of the entire message.

13.3 In discussing Table 13.1c, it was stated that alliances to defraud were impossible. In fact, there is one possibility. Describe it and explain why it would have so little credibility that we can safely ignore it.

13.4 In Section 13.2, we outlined the public-key scheme proposed in [WOO92a] for the distribution of secret keys. The revised version includes ID_A in steps 5 and 6. What attack, specifically, is countered by this revision?

13.5 The protocol referred to in Problem 13.1 can be reduced from seven steps to five, having the following sequence:

(1) A $\longrightarrow$ B:

(2) B $\longrightarrow$ KDC:

(3) KDC $\longrightarrow$ B:

(4) B $\longrightarrow$ A:

(5) A $\longrightarrow$ B:

Show the message transmitted at each step. *Hint:* The final message in this protocol is the same as the final message in the original protocol.

13.6 With reference to the suppress-replay attack described in Section 13.2:

a.

Give an example of an attack when a party's clock is ahead of that of the KDC.

b.

Give an example of an attack when a party's clock is ahead of that of another party.

13.7 There are three typical ways to use nonces as challenges. Suppose N_a is a nonce generated by A, A and B share key K, and f() is a function such as increment. The three usages are

Usage 1	Usage 2	Usage 3
(1) A $\rightarrow$ B: N_a	(1) A $\rightarrow$ B: E(K, N_a)	(1) A $\rightarrow$ B: E(K, N_a)
(2) B $\rightarrow$ A: E(K, N_a)	(2) B $\rightarrow$ A: N_a	(2) B $\rightarrow$ A: E(K, f(N_a))

Describe situations for which each usage is appropriate.

13.8 Dr. Watson patiently waited until Sherlock Holmes finished. "Some interesting problem to solve, Holmes?" he asked when Holmes finally logged out.

"Oh, not exactly. I merely checked my e-mail and then made a couple of network experiments instead of my usual chemical ones. I have only one client now and I have already solved his problem. If I remember correctly, you once mentioned cryptology among your other hobbies, so it may interest you."

"Well, I am only an amateur cryptologist, Holmes. But of course I am interested in the problem. What is it about?"

"My client is Mr. Hosgrave, director of a small but progressive bank. The bank is fully computerized and of course uses network communications extensively. The bank already uses RSA to protect its data and to digitally sign documents that are communicated. Now the bank wants to introduce some changes in its procedures; in particular, it needs to digitally sign some documents by *two* signatories so that

1.

The first signatory prepares the document, forms its signature, and passes the

document to the second signatory.

2.

The second signatory as a first step must verify that the document was really signed by the first signatory. She then incorporates her signature into the document's signature so that the recipient, as well as any member of the public, may verify that the document was indeed signed by both signatories. In addition only the second signatory has to be able to verify the document's signature after the step (1); that is the recipient (or any member of the public) should be able to verify only the complete document with signatures of both signatories, but not the document in its intermediate form where only one signatory has signed it. Moreover, the bank would like to make use of its existing modules that support RSA-style digital signatures."

"Hm, I understand how RSA can be used to digitally sign documents by *one* signatory, Holmes. I guess you have solved the problem of Mr. Hosgrave by appropriate generalization of RSA digital signatures."

"Exactly, Watson," nodded Sherlock Holmes. "Originally, the RSA digital signature was formed by encrypting the document by the signatory's private decryption key 'd', and the signature could be verified by anyone through its decryption using publicly known encryption key 'e'. One can verify that the signature S was formed by the person who knows d, which is supposed to be the only signatory. Now the problem of Mr. Hosgrave can be solved in the same way by slight generalization of the process, that is..."

Finish the explanation.

13.9 DSA specifies that if the signature generation process results in a value of $s = 0$, a new value of k should be generated and the signature should be recalculated. Why?

13.10 What happens if a k value used in creating a DSA signature is compromised?

13.11 The DSS document includes a recommended algorithm for testing a number for primality, as follows:

1.

[Choose *w*] Let w be a random odd integer. Then $(w\ 1)$ is even and can be expressed in the form $2^a\ m$ with m odd. That is, 2^a is the largest power of 2 that divides $(w\ 1)$.

2.

[Generate *b*] Let b be a random integer in the range $1 < b < w$.

3.

[Exponentiate] Set $j = 0$ and $z = b^m \bmod w$.

4.

[Done?] If $j = 0$ and $z = 1$, or if $z = w$ 1, then w passes the test and may be prime; go to step 8.

5.

[Terminate?] If $j > 0$ and $z = 1$, then w is not prime; terminate algorithm for this w.

6.

[Increase *j*] Set $j = j + 1$. If $j < a$, set $z = z^2 \bmod w$ and go to step 4.

7.

[Terminate] w is not prime; terminate algorithm for this w.

8.

[Test again?] If enough random values of b have been tested, then accept w as prime and terminate algorithm; otherwise, go to step 2.

a.

Explain how the algorithm works.

b.

Show that it is equivalent to the Miller-Rabin test described in Chapter 8.

13.12 With DSS, because the value of k is generated for each signature, even if the same message is signed twice on different occasions, the signatures will differ. This is not true of RSA signatures. What is the practical implication of this difference?

13.13 Consider the problem of creating domain parameters for DSA. Suppose we have already found primes p and q such that $q|(p\ 1)$. Now we need to find $g \in Z_p$ with g of order q mod p. Consider the following two algorithms:

Algorithm 1	**Algorithm 2**
repeat	**repeat**
select $g \in Z_p$ $h \leftarrow g^q \bmod p$	**select** $h \in Z_p$ $g \leftarrow h^{(p\ 1)/p} \bmod p$
until ($h = 1$ and $g \neq 1$)	**until** ($g \neq 1$)
return g	**return** g

a.

Prove that the value returned by Algorithm 1 has order q.

b.

Prove that the value returned by Algorithm 1 has order q.

c.

Suppose $P = 40193$ and $q = 157$. How many loop iterations do you expect Algorithm 1 to make before it finds a generator?

d.

If p is 1024 bits and q is 160 bits, would you recommend using Algorithm 1 to find g? Explain.

e.

Suppose $P = 40193$ and $q = 157$. What is the probability that Algorithm 2 computes a generator in its very first loop iteration? (if it is helpful, you may use the fact that $\sum_{d|n} \varphi(d) = n$ when answering this question).

13.14 It is tempting to try to develop a variation on Diffie-Hellman that could be used as a digital signature. Here is one that is simpler than DSA and that does not require a secret random number in addition to the private key.

Public elements:

q prime number

α $\alpha < q$ and α is a primitive root of q

Private key:

X $X < q$

Public key:

$Y = \alpha^X \bmod q$

To sign a message M, compute $h = H(M)$, the hash code of the message. We require that gcd(h, q 1) = 1. If not, append the hash to the message and calculate a new hash. Continue this process until a hash code is produced that is relatively prime to (q 1). Then calculate Z to satisfy $Z \times h \equiv X \pmod{q\ 1}$. The signature of the message is α^Z. To verify the signature, a user verifies that $Y = (\alpha^Z)^h = \alpha^X \bmod q$.

a.

Show that this scheme works. That is, show that the verification process produces an equality if the signature is valid.

b.

Show that the scheme is unacceptable by describing a simple technique for forging a user's signature on an arbitrary message.

13.15 An early proposal for a digital signature scheme using symmetric encryption is based on the following: To sign an n-bit message, the sender randomly generates in advance 2n 56-bit cryptographic keys:

$k1, K1, k2, K2, \ldots, kn, Kn$

which are kept secret. The sender prepares in advance two sets of corresponding nonsecret 64-bit validation parameters, which are made public:

$u1, V1, u2, V2, \ldots, un, Vn$ and $v1, V1, v2, V2, \ldots, vn, Vn$

where

$vi = E(ki, ui)$, $Vi = E(ki, Ui)$

The message M is signed as follows. For the i th bit of the message, either ki or Ki is attached to the message, depending on whether the message bit is 0 or 1. For example, if the first three bits of the message are 011, then the first three keys of the signature are $k1$, $K2$, $K3$.

a.

How does the receiver validate the message?

b.

Is the technique secure?

c.

How many times can the same set of secret keys be safely used for different messages?

d.

What, if any, practical problems does this scheme present?

PREV NEXT

Part Three: Network Security Applications

> *In practice, the effectiveness of a countermeasure often depends on how it is used; the best safe in the world is worthless if no one remembers to close the door.*
>
> *Computers at Risk: Safe Computing in the Information Age*, National Research Council, 1991
>
> *Increased use of computer and communications networks, computer literacy, and dependence on information technology heighten U.S. industry's risk of losing proprietary information to economic espionage. In part to reduce the risk, industry is more frequently using hardware and software with encryption capabilities.*
>
> *Communications Privacy: Federal Policy and Actions*. General Accounting Office Report GAO/OSI-94-2, November 1993

In the first two parts, we examined various ciphers and their use for confidentiality, authentication, key exchange, and related functions. Part Three surveys important network security tools and applications that make use of these functions. These applications can be used across a single network, a corporate intranet, or the Internet.

Road Map for Part Three

Chapter 14: Authentication Applications

Chapter 14 is a survey of two of the most important authentication specifications in current use. Kerberos is an authentication protocol based on conventional encryption that has received widespread support and is used in a variety of systems X.509 specifies an authentication algorithm and defines a certificate facility. The latter enables users to obtain certificates of public keys so that a community of users can have confidence in the validity of the public keys. This facility is employed as a building block in a number of applications.

Chapter 15: Electronic Mail Security

The most heavily used distributed application is electronic mail, and there is increasing interest in providing authentication and confidentiality services as part of an electronic mail facility. Chapter 15 looks at the two approaches likely to dominate electronic mail security in the near future. Pretty Good Privacy (PGP) is a widely used scheme that does not depend on any

organization or authority. Thus, it is as well suited to individual, personal use as it is to incorporation in network configurations operated by organizations. S/MIME (Secure/Multipurpose Internet Mail Extension) was developed specifically to be an Internet Standard.

Chapter 16: IP Security

The Internet Protocol (IP) is the central element in the Internet and private intranets. Security at the IP level, accordingly, is important to the design of any internetwork-based security scheme. Chapter 16 looks at the IP security scheme that has been developed to operate both with the current IP and the emerging next-generation IP, known as IPv6.

Chapter 17: Web Security

The explosive growth in the use of the World Wide Web for electronic commerce and to disseminate information has generated the need for strong Web-based security. Chapter 17 provides a survey of this important new security area and looks at two key standards: Secure Sockets Layer (SSL) and Secure Electronic Transaction (SET).

Chapter 14. Authentication Applications

We cannot enter into alliance with neighboring princes until we are acquainted with their designs.

The Art of War, Sun Tzu

Key Points

- Kerberos is an authentication service designed for use in a distributed environment.
- Kerberos makes use of a trusted third-part authentication service that enables clients and servers to establish authenticated communication.
- X.509 defines the format for public-key certificates. This format is widely used in a variety of applications.
- A public key infrastructure (PKI) is defined as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.
- Typically, PKI implementations make use of X.509 certificates.

This chapter examines some of the authentication functions that have been developed to support application-level authentication and digital signatures.

We begin by looking at one of the earliest and also one of the most widely used services: Kerberos. Next, we examine the X.509 directory authentication service. This standard is important as part of the directory service that it supports, but is also a basic building block used in other standards, such as S/MIME, discussed in Chapter 15. Finally, this chapter examines the concept of a public-key infrastructure (PKI).

14.1. Kerberos

Kerberos[1] is an authentication service developed as part of Project Athena at MIT. The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services. In particular, the following three threats exist:

[1] "In Greek mythology, a many headed dog, commonly three, perhaps with a serpent's tail, the guardian of the entrance of Hades." From *Dictionary of Subjects and Symbols in Art*, by James Hall, Harper & Row, 1979. Just as the Greek Kerberos has three heads, the modern Kerberos was intended to have three components to guard a network's gate: authentication, accounting, and audit. The last two heads were never implemented.

- A user may gain access to a particular workstation and pretend to be another user operating from that workstation.

- A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
- A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building in elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes described in this book, Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption.

Two versions of Kerberos are in common use. Version 4 [MILL88, STEI88] implementations still exist. Version 5 [KOHL94] corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard (RFC 1510).[2]

[2] Versions 1 through 3 were internal development versions. Version 4 is the "original" Kerberos.

We begin this section with a brief discussion of the motivation for the Kerberos approach. Then, because of the complexity of Kerberos, it is best to start with a description of the authentication protocol used in version 4. This enables us to see the essence of the Kerberos strategy without considering some of the details required to handle subtle security threats. Finally, we examine version 5.

Motivation

If a set of users is provided with dedicated personal computers that have no network connections, then a user's resources and files can be protected by physically securing each personal computer. When these users instead are served by a centralized time-sharing system, the time-sharing operating system must provide the security. The operating system can enforce access control policies based on user identity and use the logon procedure to identify users.

Today, neither of these scenarios is typical. More common is a distributed architecture consisting of dedicated user workstations (clients) and distributed or centralized servers. In this environment, three approaches to security can be envisioned:

1. Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).

2. Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.

3. Require the user to prove his or her identity for each service invoked. Also require that servers prove their identity to clients.

In a small, closed environment, in which all systems are owned and operated by a single organization, the first or perhaps the second strategy may suffice.[3] But in a more open environment, in which network connections to other machines are supported, the third approach is needed to protect user information and resources housed at the server. Kerberos supports this third approach. Kerberos assumes a distributed client/server architecture and employs one or more Kerberos servers to provide an authentication service.

[3] However, even a closed environment faces the threat of attack by a disgruntled employee.

The first published report on Kerberos [STEI88] listed the following requirements:

- **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- **Reliable:** For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.
- **Transparent:** Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.
- **Scalable:** The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture.

To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78], which was discussed in Chapter 7. It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.[4]

[4] Remember that the security of the Kerberos server should not automatically be assumed but must be guarded carefully (e. g., in a locked room). It is well to remember the fate of the Greek Kerberos, whom Hercules was ordered by Eurystheus to capture as his Twelfth Labor: "Hercules found the great dog on its chain and seized it by the throat. At once the three heads tried to attack, and Kerberos lashed about with his powerful tail. Hercules hung on grimly, and Kerberos relaxed into unconsciousness. Eurystheus may have been surprised to see Hercules alivewhen he saw the three slavering heads and the huge dog they belonged to he was frightened out of his wits, and leapt back into the safety of his great bronze jar." From *The Hamlyn Concise Dictionary of Greek and Roman Mythology*, by Michael Stapleton, Hamlyn, 1982.

Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. Viewing the protocol as a whole, it is difficult to see the need for the many elements contained therein. Therefore, we adopt a strategy used by Bill Bryant of Project Athena [BRYA88] and build up to the full protocol by looking first at several hypothetical dialogues. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue.

After examining the protocol, we look at some other aspects of version 4.

A Simple Authentication Dialogue

In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers must be able to confirm the identities of clients who request service. Each server can be required to undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. Consider the following hypothetical dialogue:[5]

[5] The portion to the left of the colon indicates the sender and receiver; the portion to the right indicates the contents of the message, the symbol || indicates concatenation.

(1) C $\rightarrow$ AS: $ID_C||P_C||ID_V$

(2) AS $\rightarrow$ C: $Ticket$

(3) C $\rightarrow$ V: $ID_C||Ticket$

$Ticket = E(K_V, [ID_C||AD_C||ID_V])$

where

C = client

AS = authentication server

V =server

ID_C = identifier of user on C

ID_V = identifier of V

P_C = password of user on C

AD_C = network address of C

K_v = secret encryption key shared by AS and V

In this scenario, the user logs on to a workstation and requests access to server V. The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password. The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V. If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic. To do so, the AS creates a ticket that contains the user's ID and network address and the server's ID. This ticket is encrypted using the secret key shared by the AS and this server. This ticket is then sent back to C. Because the ticket is encrypted, it cannot be altered by C or by an opponent.

With this ticket, C can now apply to V for service. C sends a message to V containing C's ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message. If these two match, the server considers the user authenticated and grants the requested service.

Each of the ingredients of message (3) is significant. The ticket is encrypted to prevent alteration or forgery. The server's ID (ID_V) is included in the ticket so that the server can verify that it has decrypted the ticket properly. ID_C is included in the ticket to indicate that this ticket has been issued on behalf of C. Finally, AD_C serves to counter the following threat. An opponent could capture the ticket transmitted in message (2), then use the name ID_C and transmit a message of form (3) from another workstation. The server would receive a valid ticket that matches the user ID and grant access to the user on that other workstation. To prevent this attack, the AS includes in the ticket the network address from which the original request came. Now the ticket is valid only if it is transmitted from the same workstation that initially requested the ticket.

A More Secure Authentication Dialogue

Although the foregoing scenario solves some of the problems of authentication in an open network environment, problems remain. Two in particular stand out. First, we would like to minimize the number of times that a user has to enter a password. Suppose each ticket can be used only once. If user C logs on to a workstation in the morning and wishes to check his or her mail at a mail server, C must supply a password to get a ticket for the mail server. If C wishes to check the mail several times during the day, each attempt requires reentering the password. We can improve matters by saying that tickets are reusable. For a single logon session, the workstation can store the mail server ticket after it is received and use it on behalf of the user for multiple accesses to the mail server.

However, under this scheme it remains the case that a user would need a new ticket for every different service. If a user wished to access a print server, a mail server, a file server, and so on, the first instance of each access would require a new ticket and hence require the user to enter the password.

The second problem is that the earlier scenario involved a plaintext transmission of the password [message (1)]. An eavesdropper could capture the password and use any service accessible to the victim.

To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the ticket-granting server (TGS). The new but still hypothetical scenario is as follows:

Once per user logon session:

(1) C $\longrightarrow$ AS: $ID_C||ID_{tgs}$

(2) AS $\longrightarrow$ C: $E(K_c, Ticket_{tgs})$

Once per type of service:

(3) C $\longrightarrow$ TGS: $ID_C||ID_V||Ticket_{tgs}$

(4) TGS $\longrightarrow$ C: $Ticket_v$

Once per service session:

(5) C $\longrightarrow$ V: $ID_C||Ticket_v$

$Ticket_{tgs} = E(K_{tgs}, [ID_C||AD_C||ID_{tgs}||TS_1||Lifetime_1])$

$Ticket_v = E(K_v, [ID_C||AD_C||ID_v||TS_2||Lifetime_2])$

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) from the AS. The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested. Let us look at the details of this scheme:

---

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID and password to the AS, together with the TGS ID, indicating a request to use the TGS service.

2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password. When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.

Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext. The ticket itself consists of the ID and network address of the user, and the ID of the TGS. This corresponds to the first scenario. The idea is that the client can use this ticket to request

multiple service-granting tickets. So the ticket-granting ticket is to be reusable. However, we do not wish an opponent to be able to capture the ticket and use it. Consider the following scenario: An opponent captures the login ticket and waits until the user has logged off his or her workstation. Then the opponent either gains access to that workstation or configures his workstation with the same network address as that of the victim. The opponent would be able to reuse the ticket to spoof the TGS. To counter this, the ticket includes a timestamp, indicating the date and time at which the ticket was issued, and a lifetime, indicating the length of time for which the ticket is valid (e.g., eight hours). Thus, the client now has a reusable ticket and need not bother the user for a password for each new service request. Finally, note that the ticket-granting ticket is encrypted with a secret key known only to the AS and the TGS. This prevents alteration of the ticket. The ticket is reencrypted with a key based on the user's password. This assures that the ticket can be recovered only by the correct user, providing the authentication.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4:

1. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.

2. The TGS decrypts the incoming ticket and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key (K_V) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5:

5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service-granting ticket. The server authenticates by using the contents of the ticket.

This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

The Version 4 Authentication Dialogue

Although the foregoing scenario enhances security compared to the first attempt, two additional problems remain. The heart of the first problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay. An opponent could eavesdrop on the network and capture a copy of the ticket-granting ticket and then wait for the legitimate user to log out. Then the opponent could forge the legitimate user's network address and

send the message of step (3) to the TGS. This would give the opponent unlimited access to the resources and files available to the legitimate user.

Similarly, if an opponent captures a service-granting ticket and uses it before it expires, the opponent has access to the corresponding service.

Thus, we arrive at an additional requirement. A network service (the TGS or an application service) must be able to prove that the person using a ticket is the same person to whom that ticket was issued.

The second problem is that there may be a requirement for servers to authenticate themselves to users. Without such authentication, an opponent could sabotage the configuration so that messages to a server were directed to another location. The false server would then be in a position to act as a real server and capture any information from the user and deny the true service to the user.

We examine these problems in turn and refer to Table 14.1, which shows the actual Kerberos protocol.

Table 14.1. Summary of Kerberos Version 4 Message Exchanges

(This item is displayed on page 408 in the print version)

(1) C → AS	$ID_c \| \| ID_{tgs} \| \| TS_1$
(2) AS → C	$E(K_c, [K_{c,tgs} \| \| ID_{tgs} \| \| TS_2 \| \| Lifetime_2 \| \| Ticket_{tgs}])$
	$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| \| ID_c \| \| AD_c \| \| ID_{tgs} \| \| TS_2 \| \| Lifetime_2])$
(a) Authentication Service Exchange to obtain ticket-granting ticket	
(3) C → TGS	$ID_v \| \| Ticket_{tgs} \| \| Authenticator_c$
(4) TGS → C	$E(K_{c,tgs}, [K_{c,v} \| \| ID_v \| \| TS_4 \| \| Ticket_v])$
	$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| \| ID_C \| \| AD_C \| \| ID_{tgs} \| \| TS_2 \| \| Lifetime_2])$ $Ticket_v = E(K_v, [K_{c,v} \| \| ID_C \| \| AD_C \| \| ID_v \| \| TS_4 \| \| Lifetime_4])$ $Authenticator_c = E(K_{c,tgs}, [ID_C \| \| AD_C \| \| TS_3])$
(b) Ticket-Granting Service Exchange to obtain service-granting ticket	
(5) C → V	$Ticket_v \| \| Authenticator_c$

(6) V → C	$E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)
	$Ticket_v = E(K_v, [K_{c,v}\|\|ID_c\|\|AD_c\|\|ID_v\|\|TS_4\|\|Lifetime_4])$ $Authenticator_c = E(K_{c,v},[ID_c\|\|AD_C\|\|TS_5])$
(c) Client/Server Authentication Exchange to obtain service	

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. The threat is that an opponent will steal the ticket and use it before it expires. To get around this problem, let us have the AS provide both the client and the TGS with a secret piece of information in a secure manner. Then the client can prove its identity to the TGS by revealing the secret information, again in a secure manner. An efficient way of accomplishing this is to use an encryption key as the secure information; this is referred to as a session key in Kerberos.

Table 14.1a shows the technique for distributing the session key. As before, the client sends a message to the AS requesting access to the TGS. The AS responds with a message, encrypted with a key derived from the user's password (K_c) that contains the ticket. The encrypted message also contains a copy of the session key, $K_{c,tgs}$, where the subscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with K_c, only the user's client can read it. The same session key is included in the ticket, which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS.

Note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

Armed with the ticket and the session key, C is ready to approach the TGS. As before, C sends the TGS a message that includes the ticket plus the ID of the requested service (message (3) in Table 14.1b). In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key $K_{c,tgs}$. In effect, the ticket says, "Anyone who uses $K_{c,tgs}$ must be C." The TGS uses the session key to decrypt the authenticator. The TGS can then check the name and address from the authenticator with that of the ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner. In effect, the authenticator says, "At time TS_3, I hereby use $K_{c,tgs}$." Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered.

The reply from the TGS, in message (4), follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V, the ID of V, and the timestamp of the ticket. The ticket itself includes the same session key.

C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator.

If mutual authentication is required, the server can reply as shown in message (6) of Table 14.1. The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assure C that this is not a replay of an old reply.

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.

Table 14.2 summarizes the justification for each of the elements in the Kerberos protocol, and Figure 14.1 provides a simplified overview of the action.

Table 14.2. Rationale for the Elements of the Kerberos Version 4 Protocol

(This item is displayed on pages 410 - 411 in the print version)

Message (1)	Client requests ticket-granting ticket
ID_C	Tells AS identity of user from this client
ID_{tgs}	Tells AS that user requests access to TGS
TS_1	Allows AS to verify that client's clock is synchronized with that of AS
Message (2)	AS returns ticket-granting ticket
K_c	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2)
$K_{c,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key
ID_{tgs}	Confirms that this ticket is for the TGS

TS_2	Informs client of time this ticket was issued
$Lifetime_2$	Informs client of the lifetime of this ticket
$Ticket_{tgs}$	Ticket to be used by client to access TGS
	(a) Authentication Service Exchange
Message (3)	Client requests service-granting ticket
ID_V	Tells TGS that user requests access to server V
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS
$Authenticator_c$	Generated by client to validate ticket
Message (4)	TGS returns service-granting ticket
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4)
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key
ID_v	Confirms that this ticket is for server V
TS_4	Informs client of time this ticket was issued
$Ticket_v$	Ticket to be used by client to access server V
$Ticket_{tgs}$	Reusable so that user does not have to reenter password
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering
$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket

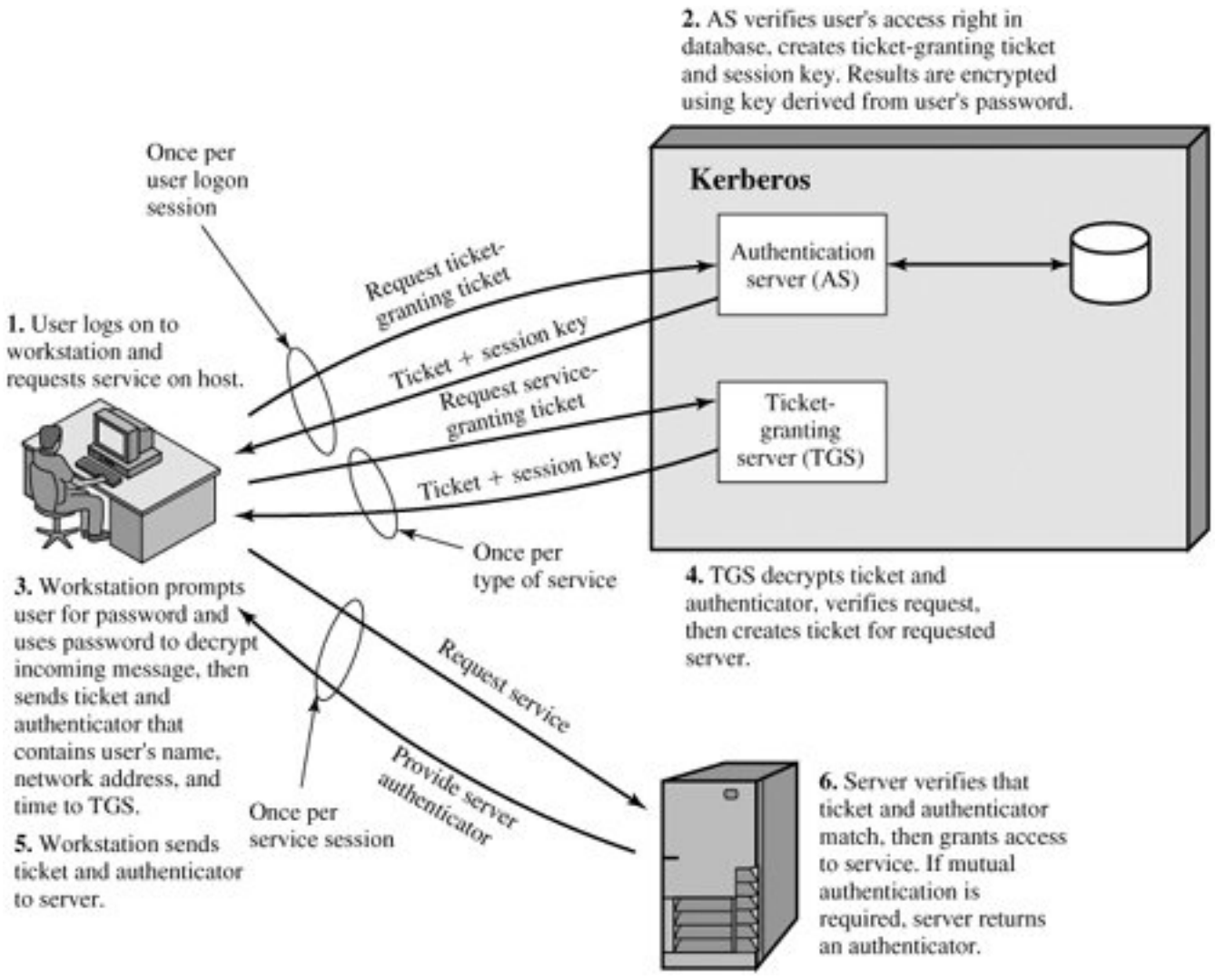
ID_C	Indicates the rightful owner of this ticket
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket
ID_{tgs}	Assures server that it has decrypted ticket properly
TS_2	Informs TGS of time this ticket was issued
$Lifetime_2$	Prevents replay after ticket has expired
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tamperig
ID_c	Must match ID in ticket to authenticate ticket
AD_c	Must match address in ticket to authenticate ticket
TS_3	Informs TGS of time this authenticator was generated
	(b) Ticket-Granting Service Exchange
Message (5)	Client requests service
$Ticket_v$	Assures server that this user has been authenticated by AS
$Authenticator_c$	Generated by client to validate ticket
Message (6)	Optional authentication of server to client
$K_{c,v}$	Assures C that this message is from V
$TS_5 + 1$	Assures C that this is not a replay of an old reply

$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server
K_v	Ticket is encrypted with key known only to TGS and server, to prevent tampering
$K_{c,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket
ID_C	Indicates the rightful owner of this ticket
AD_c	Prevents use of ticket from workstation other than one that initially requested the ticket
ID_v	Assures server that it has decrypted ticket properly
TS_4	Informs server of time this ticket was issued
$Lifetime_4$	Prevents replay after ticket has expired
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay
$K_{c,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering
ID_C	Must match ID in ticket to authenticate ticket
AD_c	Must match address in ticket to authenticate ticket
TS_5	Informs server of time this authenticator was generated
	(c) Client/Server Authentication Exchange

Figure 14.1. Overview of Kerberos

(This item is displayed on page 412 in the print version)

[View full size image]



Kerberos Realms and Multiple Kerberi

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.

2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a **Kerberos realm**. The concept of *realm* can be explained as follows. A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos database resides on the Kerberos master computer system, which should be kept in a physically secure room. A read-only copy of the Kerberos database might also reside on other Kerberos computer systems. However, all changes to the database must be made on the master computer system. Changing or accessing the contents of a Kerberos database requires the Kerberos master password. A related concept is that of a **Kerberos principal**, which is a service or user that is known to the Kerberos system. Each Kerberos principal is identified by its principal name. Principal names consist

of three parts: a service or user name, an instance name, and a realm name

Networks of clients and servers under different administrative organizations typically constitute different realms. That is, it generally is not practical, or does not conform to administrative policy, to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added:

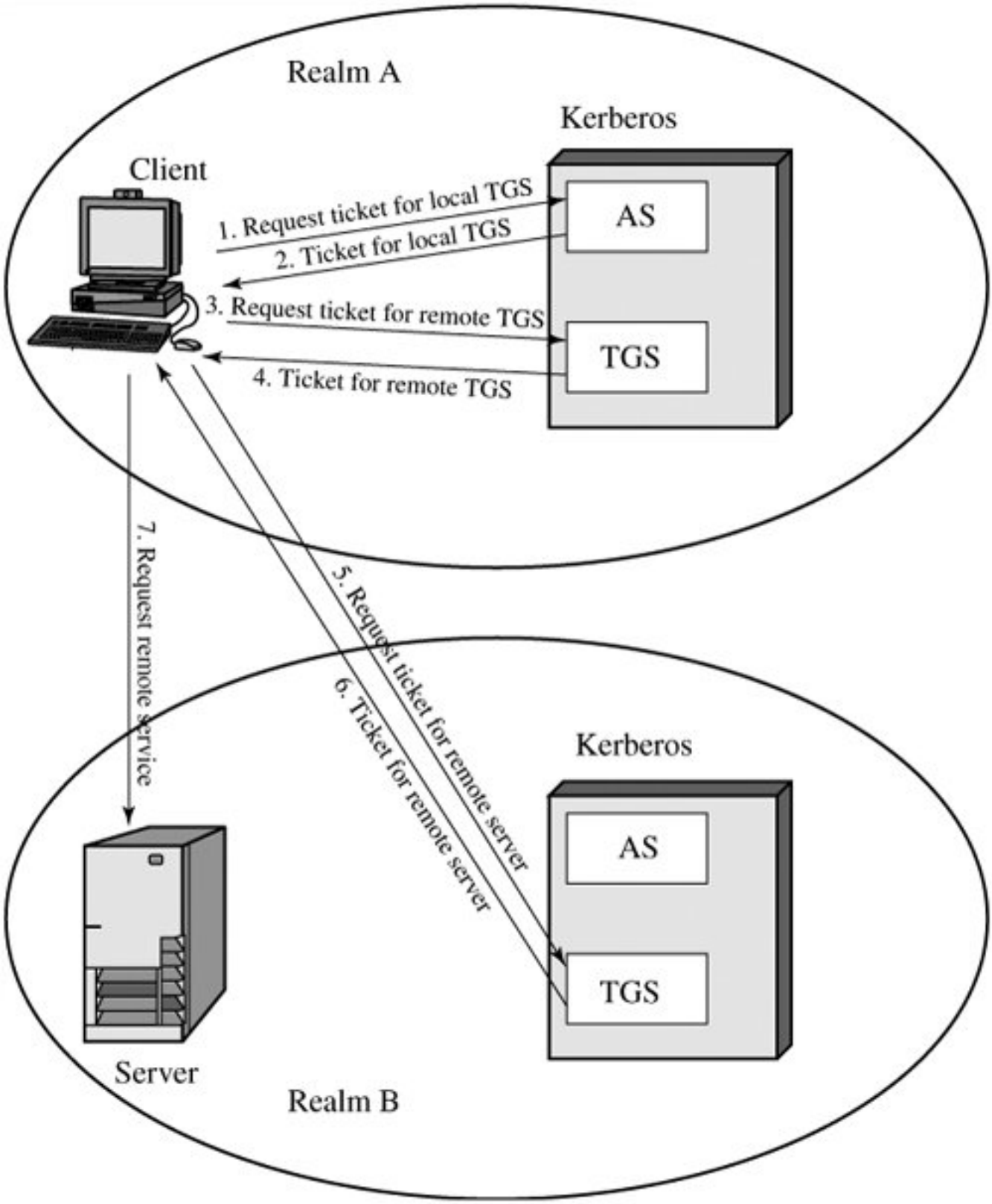
3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

With these ground rules in place, we can describe the mechanism as follows (Figure 14.2): A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm). The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS.

Figure 14.2. Request for Service in Another Realm

(This item is displayed on page 413 in the print version)



The details of the exchanges illustrated in Figure 14.2 are as follows (compare Table 14.1):

(1) C → AS: $ID_c||ID_{tgs}||TS_1$

(2) AS → C: $E(K_c, [K_{c,tgs}||ID_{tgs}||TS_2||Lifetime_2||Ticket_{tgs}])$

(3) C → TGS: $ID_{tgsrem}||Ticket_{tgs}||Authenticator_c$

(4) TGS $\longrightarrow$ C: $E(K_{c,tgs}, [K_{c,tgsrem}||ID_{tgsrem}||TS_4||Ticket_{tgsrem}])$

(5) C $\longrightarrow$ TGS_{rem}: $ID_{vrem}||Ticket_{tgsrem}||Authenticator_c$

(6) TGS_{rem} $\longrightarrow$ C: $E(K_{c,tgsrem}, [K_{c,vrem}||ID_{vrem}||TS_6||Ticket_{vrem}])$

(7) C $\longrightarrow$ V_{rem}: $Ticket_{vrem}||Authenticator_c$

The ticket presented to the remote server (V_{rem}) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are N realms, then there must be $N(N\ 1)/2$ secure key exchanges so that each Kerberos realm can interoperate with all other Kerberos realms.

Kerberos Version 5

Kerberos Version 5 is specified in RFC 1510 and provides a number of improvements over version 4 [KOHL94]. To begin, we provide an overview of the changes from version 4 to version 5 and then look at the version 5 protocol.

Differences between Versions 4 and 5

Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies. Let us briefly summarize the improvements in each area.[6]

[6] The following discussion follows the presentation in [KOHL94].

Kerberos Version 4 was developed for use within the Project Athena environment and, accordingly, did not fully address the need to be of general purpose. This led to the following **environmental shortcomings:**

1. **Encryption system dependence:** Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of concern. In version 5, ciphertext is tagged with an encryption type identifier so that any encryption technique may be used. Encryption keys are tagged with a type and a length, allowing the same key to be used in different algorithms and allowing the specification of different variations on a given algorithm.

2.

Internet protocol dependence: Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.

3.

Message byte ordering: In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This techniques works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.

4.

Ticket lifetime: Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is $2^8 \times 5 = 1280$ minutes, or a little over 21 hours. This may be inadequate for some applications (e.g., a long-running simulation that requires valid Kerberos credentials throughout execution). In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

5.

Authentication forwarding: Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.

6.

Interrealm authentication: In version 4, interoperability among *N* realms requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

Apart from these environmental limitations, there are **technical deficiencies** in the version 4 protocol itself. Most of these deficiencies were documented in [BELL90], and version 5 attempts to address these. The deficiencies are the following:

1.

Double encryption: Note in Table 14.1 [messages (2) and (4)] that tickets provided to clients are encrypted twice, once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.

2.

PCBC encryption: Encryption in version 4 makes use of a nonstandard mode of DES known as propagating cipher block chaining (PCBC).[7] It has been demonstrated that this mode is

vulnerable to an attack involving the interchange of ciphertext blocks [KOHL89]. PCBC was intended to provide an integrity check as part of the encryption operation. Version 5 provides explicit integrity mechanisms, allowing the standard CBC mode to be used for encryption. In particular, a checksum or hash code is attached to the message prior to encryption using CBC.

[7] This is described in Appendix 14A.

3.

Session keys: Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket. In addition, the session key may subsequently be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a subsession key, which is to be used only for that one connection. A new access by the client would result in the use of a new subsession key.

4.

Password attacks: Both versions are vulnerable to a password attack. The message from the AS to the client includes material encrypted with a key based on the client's password.[8] An opponent can capture this message and attempt to decrypt it by trying various passwords. If the result of a test decryption is of the proper form, then the opponent has discovered the client's password and may subsequently use it to gain authentication credentials from Kerberos. This is the same type of password attack described in Chapter 18, with the same kinds of countermeasures being applicable. Version 5 does provide a mechanism known as preauthentication, which should make password attacks more difficult, but it does not prevent them.

[8] Appendix 14A describes the mapping of passwords to encryption keys.

The Version 5 Authentication Dialogue

Table 14.3 summarizes the basic version 5 dialogue. This is best explained by comparison with version 4 (Table 14.1).

Table 14.3. Summary of Kerberos Version 5 Message Exchanges

(This item is displayed on page 416 in the print version)

(1) C → AS	$\text{Options}\|\|ID_c\|\|\text{Realm}_c\|\|ID_{tgs}\|\|Times\|\|Nonce_1$
(2) AS → C	$Realm_c\|\|ID_C\|\|Ticket_{tgs}\|\|\text{E}(K_c, [K_{c,tgs}\|\|Times\|\|Nonce_1\|\|Realm_{tgs}\|\|ID_{tgs}])$
	$Ticket_{tgs} = \text{E}(K_{tgs}, [Flags\|\|K_{c,tgs}\|\|Realm_c\|\|ID_c\|\|AD_c\|\|Times])$

(a) Authentication Service Exchange to obtain ticket-granting ticket	
(3) C → TGS	$Options \| \| ID_v \| \| Times \| \| \| \| Nonce_2 \| \| Ticket_{tgs} \| \| Authenticator_c$
(4) TGS → C	$Realm_c \| \| ID_c \| \| Ticket_v \| \| E(K_{c,tgs}, [K_{c,v} \| \| Times \| \| Nonce_2 \| \| Realm_v \| \| ID_v])$ $Ticket_{tgs} = E(K_{tgs}, [Flags \| \| K_{C,tgs} \| \| Realm_c \| \| ID_C \| \| AD_C \| \| Times])$
	$Ticket_v = E(K_v, [Flags \| \| K_{c,v} \| \| Realm_c \| \| ID_C \| \| AD_c \| \| Times])$ $Authenticator_c = E(K_{c,tgs}, [ID_C \| \| Realm_c \| \| TS_1])$
(b) Ticket-Granting Service Exchange to obtain service-granting ticket	
(5) C → V	$Options \| \| Ticket_v \| \| Authenticator_c$
(6) V → C	$E_{K_{c,v}}[TS_2 \| \| Subkey \| \| Seq\#]$
	$Ticket_v = E(K_v, [Flags \| \| K_{c,v} \| \| Realm_c \| \| ID_C \| \| AD_C \| \| Times])$ $Authenticator_c = E(K_{c,v}, [ID_C \| \| Realm_c \| \| TS_2 \| \| Subkey \| \| Seq\#])$
(c) Client/Server Authentication Exchange to obtain service	

First, consider the **authentication service exchange**. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:

- **Realm:** Indicates realm of user
- **Options:** Used to request that certain flags be set in the returned ticket

---

- **Times:** Used by the client to request the following time settings in the ticket:

  from: the desired start time for the requested ticket

  till: the requested expiration time for the requested ticket

  rtime: requested renew-till time

- **Nonce**: A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used

between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the **ticket-granting service exchange** for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2), returning a ticket plus information needed by the client, the latter encrypted with the session key now shared by the client and the TGS.

Finally, for the **client/server authentication exchange**, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields as follows:

- **Subkey**: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{c,v}$) is used.
- **Sequence number**: An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5 because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys. The subkey field, if present, overrides the subkey field, if present, in message (5). The optional sequence number field specifies the starting sequence number to be used by the client.

Ticket Flags

The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4. Table 14.4 summarizes the flags that may be included in a ticket.

Table 14.4. Kerberos Version 5 Flags

INITIAL	This ticket was issued using the AS protocol and not issued based on a ticket-granting ticket.
PRE-AUTHENT	During initial authentication, the client was authenticated by the KDC before a ticket was issued.
HW-AUTHENT	The protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.

RENEWABLE	Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
MAY-POSTDATE	Tells TGS that a postdated ticket may be issued based on this ticket-granting ticket.
POSTDATED	Indicates that this ticket has been postdated; the end server can check the authtime field to see when the original authentication occurred.
INVALID	This ticket is invalid and must be validated by the KDC before use.
PROXIABLE	Tells TGS that a new service-granting ticket with a different network address may be issued based on the presented ticket.
PROXY	Indicates that this ticket is a proxy.
FORWARDABLE	Tells TGS that a new ticket-granting ticket with a different network address may be issued based on this ticket-granting ticket.
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.

The INITIAL flag indicates that this ticket was issued by the AS, not by the TGS. When a client requests a service-granting ticket from the TGS, it presents a ticket-granting ticket obtained from the AS. In version 4, this was the only way to obtain a service-granting ticket. Version 5 provides the additional capability that the client can get a service-granting ticket directly from the AS. The utility of this is as follows: A server, such as a password-changing server, may wish to know that the client's password was recently tested.

The PRE-AUTHENT flag, if set, indicates that when the AS received the initial request [message (1)], it authenticated the client before issuing a ticket. The exact form of this preauthentication is left unspecified. As an example, the MIT implementation of version 5 has encrypted timestamp preauthentication, enabled by default. When a user wants to get a ticket, it has to send to the AS a preauthentication block containing a random confounder, a version number, and a timestamp, encrypted in the client's password-based key. The AS decrypts the block and will not send a ticket-granting ticket back unless the timestamp in the preauthentication block is within the allowable time skew (time interval to account for clock drift and network delays). Another possibility is the use of a smart card that generates continually changing passwords that are included in the preauthenticated messages. The passwords generated by the card can be based on a user's password but be transformed by the card so that, in effect, arbitrary passwords are used. This prevents an attack based on easily guessed passwords. If a smart card or similar device was used, this is indicated by the HW-AUTHENT flag.

When a ticket has a long lifetime, there is the potential for it to be stolen and used by an opponent for a considerable period. If a short lifetime is used to lessen the threat, then overhead is involved in acquiring new tickets. In the case of a ticket-granting ticket, the client would either have to store the user's secret key, which is clearly risky, or repeatedly ask the user for a password. A compromise scheme is the use of renewable tickets. A ticket with the RENEWABLE flag set includes two expiration times: one for this specific ticket and one that is the latest permissible value for an expiration time. A client can have the ticket renewed by presenting it to the TGS with a requested new expiration time. If the new time is within the limit of the latest permissible value, the TGS can issue a new ticket with a new session time and a later specific expiration time. The advantage of this mechanism is that the TGS may refuse to renew a ticket reported as stolen.

A client may request that the AS provide a ticket-granting ticket with the MAY-POSTDATE flag set. The

client can then use this ticket to request a ticket that is flagged as POSTDATED and INVALID from the TGS. Subsequently, the client may submit the postdated ticket for validation. This scheme can be useful for running a long batch job on a server that requires a ticket periodically. The client can obtain a number of tickets for this session at once, with spread-out time values. All but the first ticket are initially invalid. When the execution reaches a point in time when a new ticket is required, the client can get the appropriate ticket validated. With this approach, the client does not have to repeatedly use its ticket-granting ticket to obtain a service-granting ticket.

In version 5 it is possible for a server to act as a proxy on behalf of a client, in effect adopting the credentials and privileges of the client to request a service from another server. If a client wishes to use this mechanism, it requests a ticket-granting ticket with the PROXIABLE flag set. When this ticket is presented to the TGS, the TGS is permitted to issue a service-granting ticket with a different network address; this latter ticket will have its PROXY flag set. An application receiving such a ticket may accept it or require additional authentication to provide an audit trail.[9]

[9] For a discussion of some of the possible uses of the proxy capability, see [NEUM93b].

The proxy concept is a limited case of the more powerful forwarding procedure. If a ticket is set with the FORWARDABLE flag, a TGS can issue to the requestor a ticket-granting ticket with a different network address and the FORWARDED flag set. This ticket can then be presented to a remote TGS. This capability allows a client to gain access to a server on another realm without requiring that each Kerberos maintain a secret key with Kerberos servers in every other realm. For example, realms could be structured hierarchically. Then a client could walk up the tree to a common node and then back down to reach a target realm. Each step of the walk would involve forwarding a ticket-granting ticket to the next TGS in the path.

14.2. X.509 Authentication Service

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users.

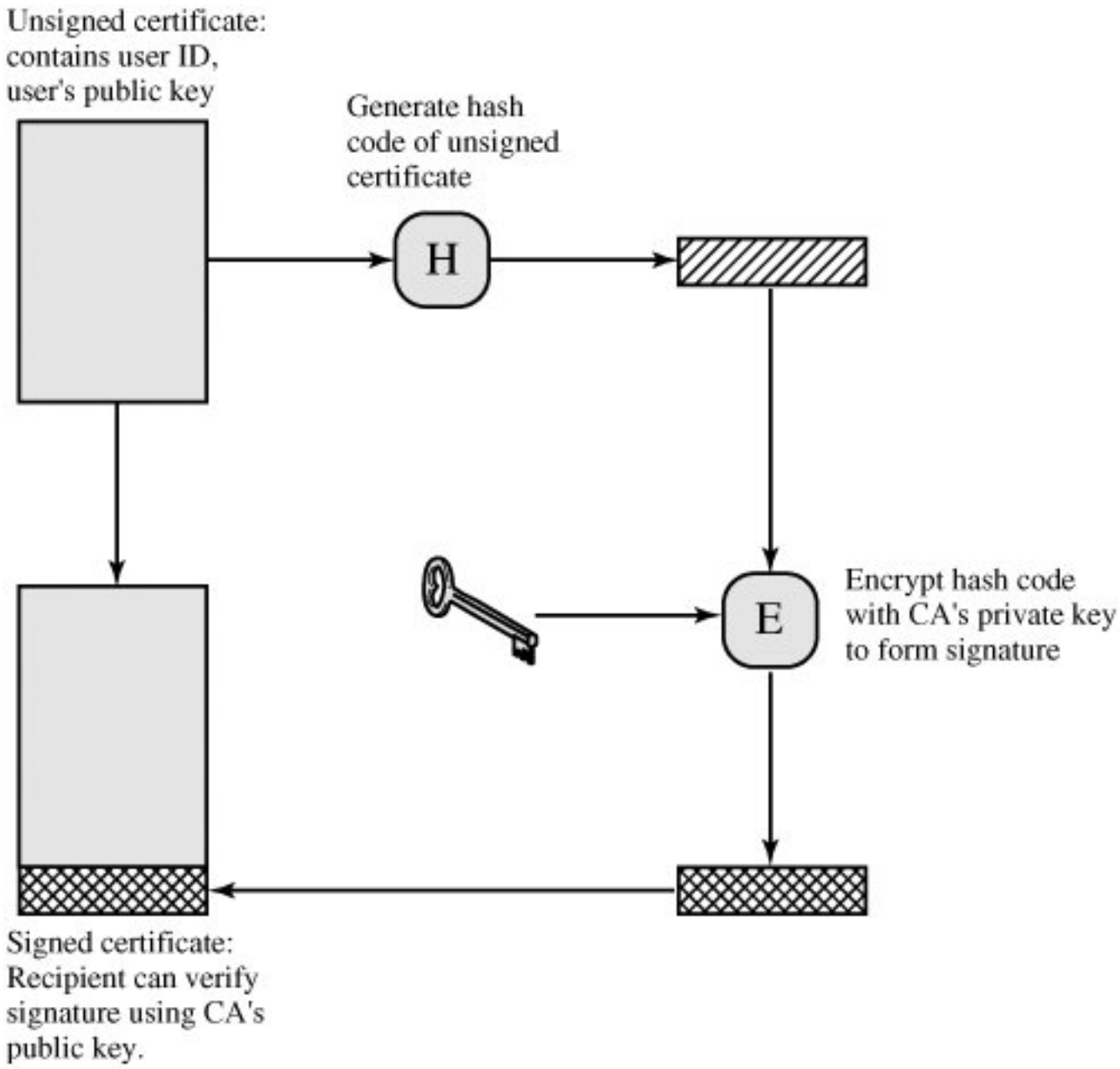
X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates of the type discussed in Chapter 9. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME (Chapter 15), IP Security (Chapter 16), and SSL/TLS and SET (Chapter 17).

X.509 was initially issued in 1988. The standard was subsequently revised to address some of the security concerns documented in [IANS90] and [MITC90]; a revised recommendation was issued in 1993. A third version was issued in 1995 and revised in 2000.

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function. Again, the standard does not dictate a specific hash algorithm. The 1988 recommendation included the description of a recommended hash algorithm; this algorithm has since been shown to be insecure and was dropped from the 1993 recommendation. Figure 14.3 illustrates the generation of a public-key certificate.

Figure 14.3. Public-Key Certificate Use



Certificates

The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

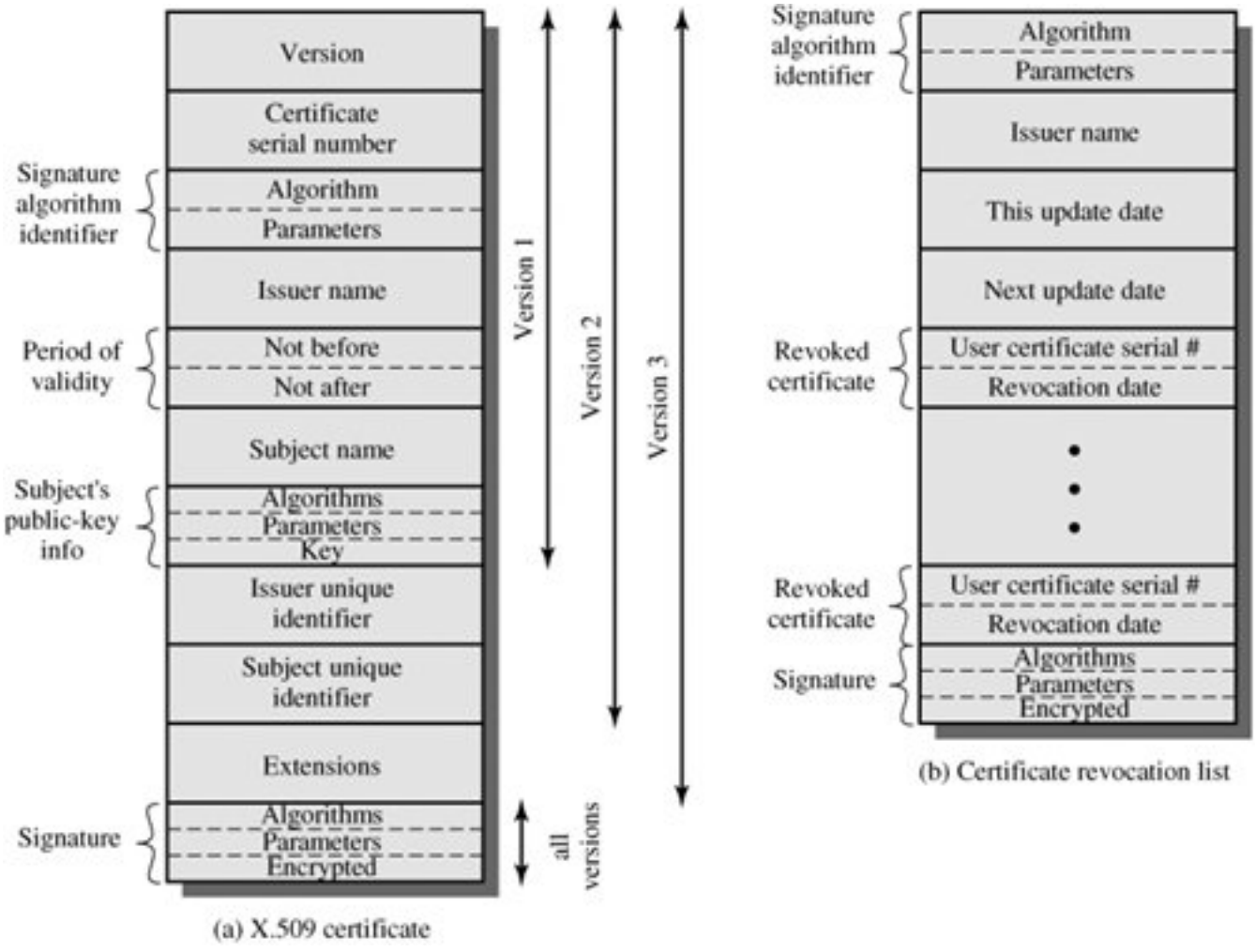
Figure 14.4a shows the general format of a certificate, which includes the following elements:

- **Version:** Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.
- **Serial number:** An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.
- **Signature algorithm identifier:** The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.
- **Issuer name:** X.500 name of the CA that created and signed this certificate.

- **Period of validity:** Consists of two dates: the first and last on which the certificate is valid.
- **Subject name:** The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.
- **Subject's public-key information:** The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
- **Issuer unique identifier:** An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.
- **Subject unique identifier:** An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
- **Extensions:** A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.
- **Signature:** Covers all of the other fields of the certificate; it contains the hash code of the other fields, encrypted with the CA's private key. This field includes the signature algorithm identifier.

Figure 14.4. X.509 Formats

[View full size image]



The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. These fields are rarely used.

The standard uses the following notation to define a certificate:

$CA\langle\langle A\rangle\rangle = CA\{V, SN, AI, CA, T_A, A, Ap\}$

where

$Y\langle\langle X\rangle\rangle$	= the certificate of user X issued by certification authority Y
$Y\{I\}$	= the signing of I by Y. It consists of I with an encrypted hash code appended

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid. This is the typical digital signature approach illustrated in Figure 11.5c.

Obtaining a User's Certificate

User certificates generated by a CA have the following characteristics:

- Any user with access to the public key of the CA can verify the user public key that was certified.
- No party other than the certification authority can modify the certificate without this being detected.

Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users. In addition, a user can transmit his or her certificate directly to other users. In either case, once B is in possession of A's certificate, B has confidence that messages it encrypts with A's public key will be secure from eavesdropping and that messages signed with A's private key are unforgeable.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2. If A does not securely know the public key of X_2, then B's certificate, issued by X_2, is useless to A. A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key:

1.

   A obtains, from the directory, the certificate of X_2 signed by X_1. Because A securely knows X_1's public key, A can obtain X_2's public key from its certificate and verify it by means of X_1's signature on the certificate.

2.

A then goes back to the directory and obtains the certificate of B signed by X_2 Because A now has a trusted copy of X_2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

$X_1 \ll X_2 \gg X_2 \ll B \gg$

In the same fashion, B can obtain A's public key with the reverse chain:

$X_2 \ll X_1 \gg X_1 \ll A \gg$

This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with *N* elements would be expressed as

$X_1 \ll X_2 \gg X_2 \ll X_3 \gg \ldots X_N \ll B \gg$

In this case, each pair of CAs in the chain ($X_{i,}$ X_{i+1}) must have created certificates for each other.

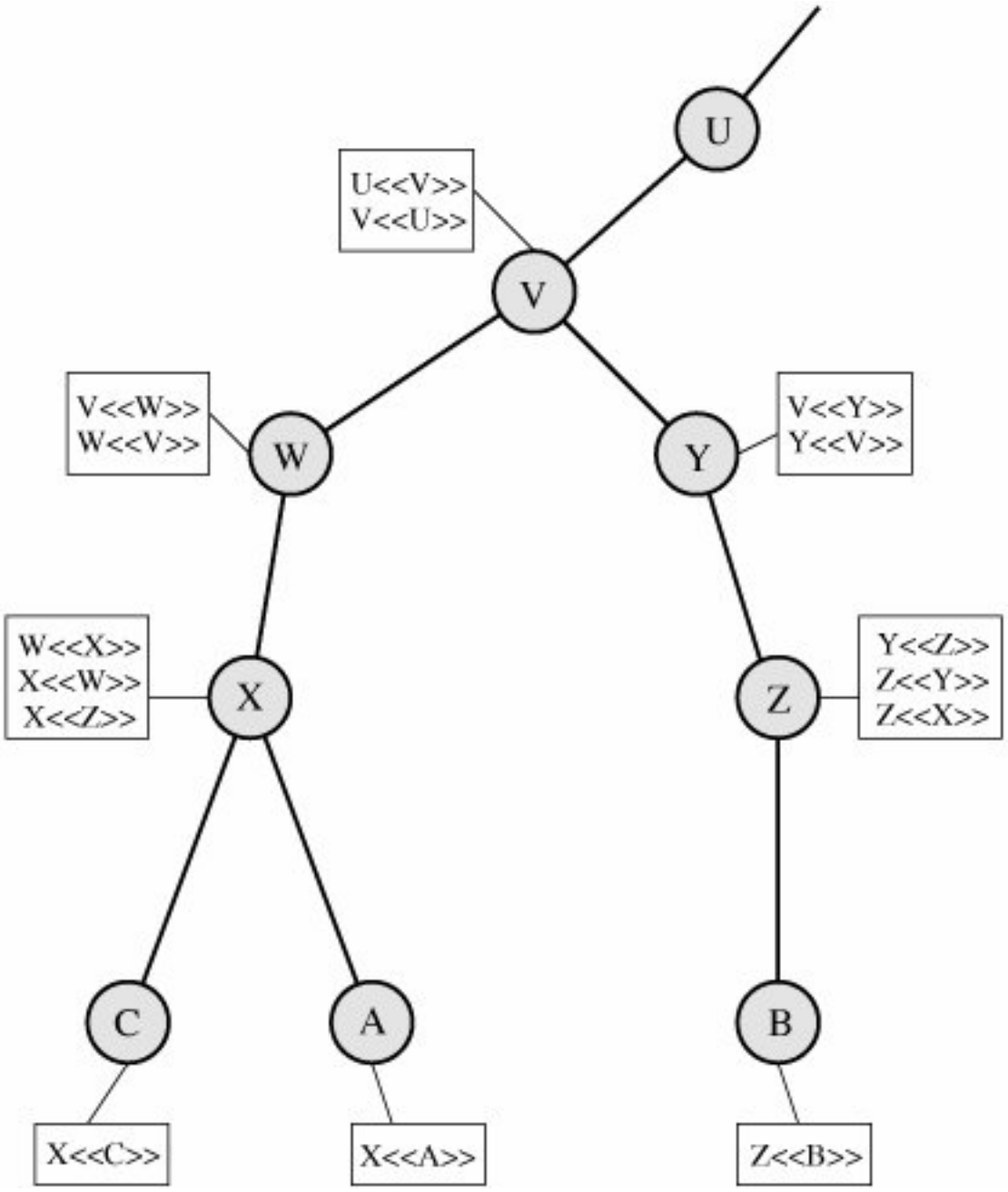
All these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

Figure 14.5, taken from X.509, is an example of such a hierarchy. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

- **Forward certificates:** Certificates of X generated by other CAs
- **Reverse certificates:** Certificates generated by X that are the certificates of other CAs

Figure 14.5. X.509 Hierarchy: A Hypothetical Example

(This item is displayed on page 424 in the print version)



In this example, user A can acquire the following certificates from the directory to establish a certification path to B:

X<<W>> W <<V>> V <<Y>> <<Z>> Z <<B>>

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

Z<<Y>> Y <<V>> V <<W>> W <<X>>X <<A>>

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.

Revocation of Certificates

Recall from Figure 14.4 that each certificate includes a period of validity, much like a credit card. Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

1. The user's private key is assumed to be compromised.
2. The user is no longer certified by this CA.
3. The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory.

Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (Figure 14.4b) the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

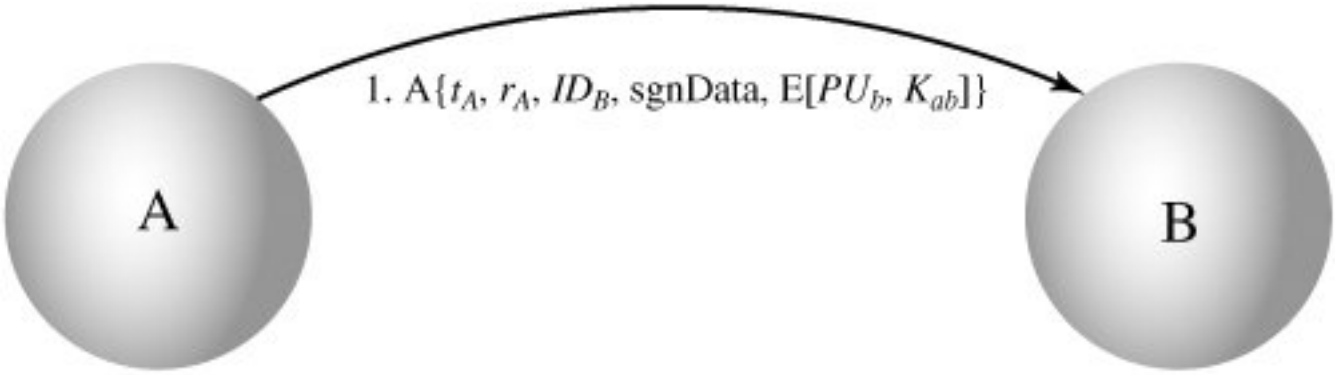
When a user receives a certificate in a message, the user must determine whether the certificate has been revoked. The user could check the directory each time a certificate is received. To avoid the delays (and possible costs) associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certificates.

Authentication Procedures

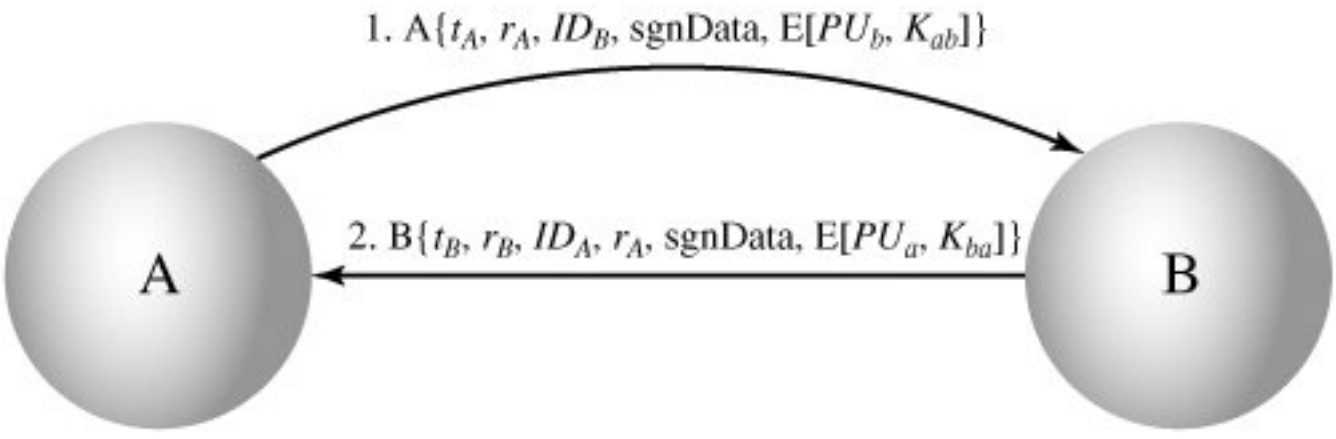
X.509 also includes three alternative authentication procedures that are intended for use across a variety of applications. All these procedures make use of public-key signatures. It is assumed that the two parties know each other's public key, either by obtaining each other's certificates from the directory or because the certificate is included in the initial message from each side.

Figure 14.6 illustrates the three procedures.

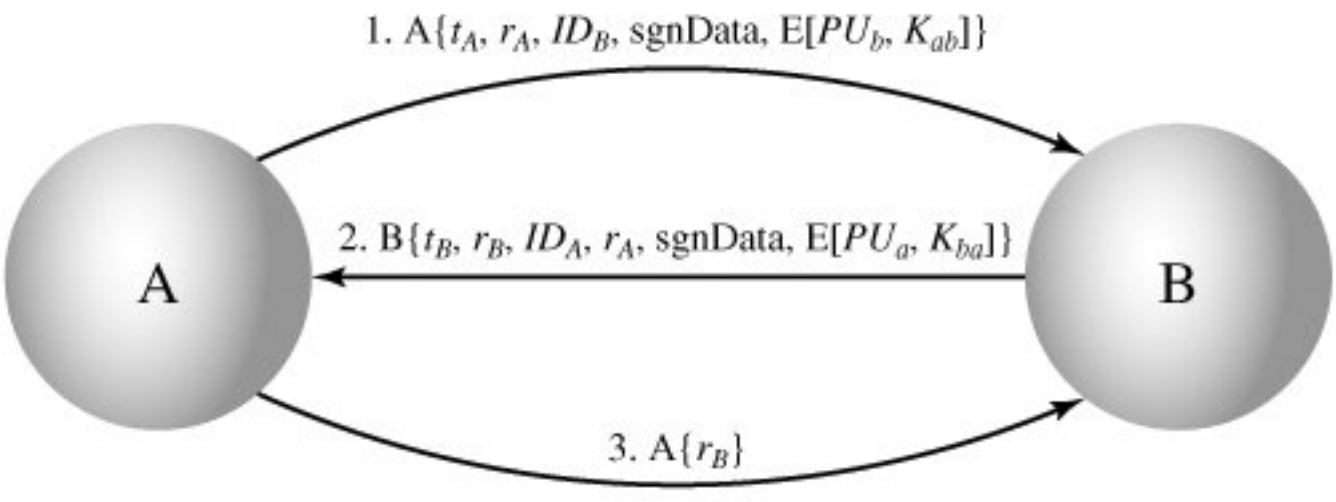
Figure 14.6. X.509 Strong Authentication Procedures



(a) One-way authentication



(b) Two-way authentication



(c) Three-way authentication

One-Way Authentication

One way authentication involves a single transfer of information from one user (A) to another (B), and establishes the following:

1.

   The identity of A and that the message was generated by A

That the message was intended for B

3.

The integrity and originality (it has not been sent multiple times) of the message

Note that only the identity of the initiating entity is verified in this process, not that of the responding entity.

At a minimum, the message includes a timestamp t_A, a nonce r_A and the identity of B and is signed with A's private key. The timestamp consists of an optional generation time and an expiration time. This prevents delayed delivery of messages. The nonce can be used to detect replay attacks. The nonce value must be unique within the expiration time of the message. Thus, B can store the nonce until it expires and reject any new messages with the same nonce.

For pure authentication, the message is used simply to present credentials to B. The message may also include information to be conveyed. This information, sgnData, is included within the scope of the signature, guaranteeing its authenticity and integrity. The message may also be used to convey a session key to B, encrypted with B's public key.

Two-Way Authentication

In addition to the three elements just listed, two-way authentication establishes the following elements:

4.

The identity of B and that the reply message was generated by B

5.

That the message was intended for A

6.

The integrity and originality of the reply

Two-way authentication thus permits both parties in a communication to verify the identity of the other.

The reply message includes the nonce from A, to validate the reply. It also includes a timestamp and nonce generated by B. As before, the message may include signed additional information and a session key encrypted with A's public key.

Three-Way Authentication

In three-way authentication, a final message from A to B is included, which contains a signed copy of the nonce r_B. The intent of this design is that timestamps need not be checked: Because both nonces are echoed back by the other side, each side can check the returned nonce to detect replay attacks. This

approach is needed when synchronized clocks are not available.

X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed. [FORD95] lists the following requirements not satisfied by version 2:

1. The Subject field is inadequate to convey the identity of a key owner to a public-key user. X.509 names may be relatively short and lacking in obvious identification details that may be needed by the user.

2. The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet-related identification.

3. There is a need to indicate security policy information. This enables a security application or function, such as IPSec, to relate an X.509 certificate to a given policy.

4. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.

5. It is important to be able to identify different keys used by the same owner at different times. This feature supports key life cycle management, in particular the ability to update key pairs for users and CAs on a regular basis or under exceptional circumstances.

Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. Thus, version 3 includes a number of optional extensions that may be added to the version 2 format. Each extension consists of an extension identifier, a criticality indicator, and an extension value. The criticality indicator indicates whether an extension can be safely ignored. If the indicator has a value of TRUE and an implementation does not recognize the extension, it must treat the certificate as invalid.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

Key and Policy Information

These extensions convey additional information about the subject and issuer keys, plus indicators of

certificate policy. A certificate policy is a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes the following:

- **Authority key identifier:** Identifies the public key to be used to verify the signature on this certificate or CRL. Enables distinct keys of the same CA to be differentiated. One use of this field is to handle CA key pair updating.
- **Subject key identifier:** Identifies the public key being certified. Useful for subject key pair updating. Also, a subject may have multiple key pairs and, correspondingly, different certificates for different purposes (e.g., digital signature and encryption key agreement).
- **Key usage:** Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used. May indicate one or more of the following: digital signature, nonrepudiation, key encryption, data encryption, key agreement, CA signature verification on certificates, CA signature verification on CRLs.
- **Private-key usage period:** Indicates the period of use of the private key corresponding to the public key. Typically, the private key is used over a different period from the validity of the public key. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.
- **Certificate policies:** Certificates may be used in environments where multiple policies apply. This extension lists policies that the certificate is recognized as supporting, together with optional qualifier information.
- **Policy mappings:** Used only in certificates for CAs issued by other CAs. Policy mappings allow an issuing CA to indicate that one or more of that issuer's policies can be considered equivalent to another policy used in the subject CA's domain.

Certificate Subject and Issuer Attributes

These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject, to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required.

The extension fields in this area include the following:

- **Subject alternative name:** Contains one or more alternative names, using any of a variety of forms. This field is important for supporting certain applications, such as electronic mail, EDI, and IPSec, which may employ their own name forms.
- **Issuer alternative name:** Contains one or more alternative names, using any of a variety of forms.
- **Subject directory attributes:** Conveys any desired X.500 directory attribute values for the subject of this certificate.

Certification Path Constraints

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs. The constraints may restrict the types of certificates that can be issued by the subject CA or that may occur subsequently in a certification chain.

The extension fields in this area include the following:

- **Basic constraints:** Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.
- **Name constraints:** Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.
- **Policy constraints:** Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

PREV NEXT

14.3. Public-Key Infrastructure

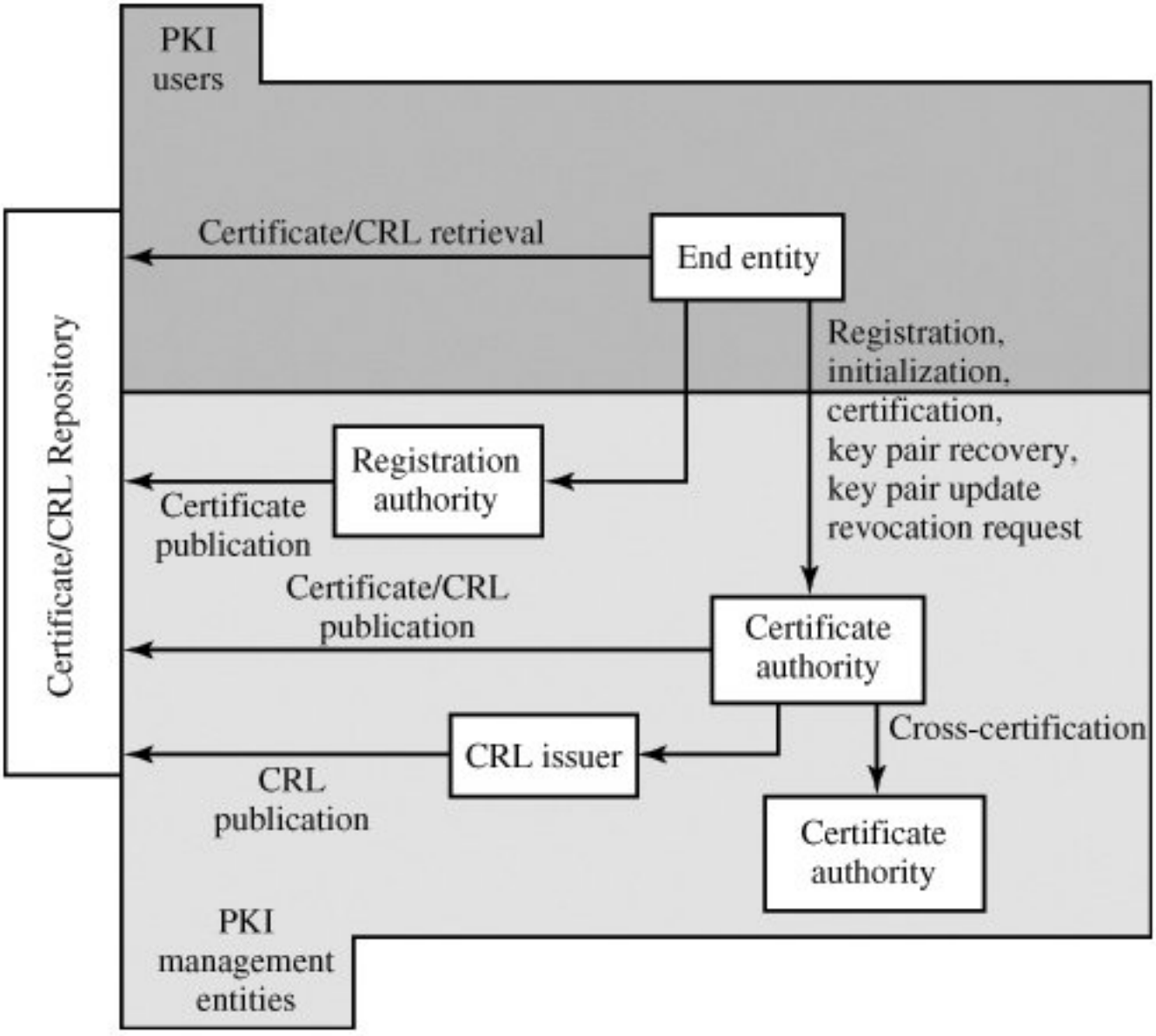
RFC 2822 (*Internet Security Glossary*) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys. The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet. This section describes the PKIX model.

Figure 14.7 shows the interrelationship among the key elements of the PKIX model. These elements are

- **End entity:** A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate. End entities typically consume and/or support PKI-related services.

- **Certification authority (CA):** The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to one or more Registration Authorities.
- **Registration authority (RA):** An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the End Entity registration process, but can assist in a number of other areas as well.
- **CRL issuer:** An optional component that a CA can delegate to publish CRLs.
- **Repository:** A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by End Entities.

Figure 14.7. PKIX Architectural Model



PKIX Management Functions

PKIX identifies a number of management functions that potentially need to be supported by management protocols. These are indicated in Figure 14.7 and include the following:

- **Registration:** This is the process whereby a user first makes itself known to a CA (directly, or through an RA), prior to that CA issuing a certificate or certificates for that user. Registration begins the process of enrolling in a PKI. Registration usually involves some offline or online procedure for mutual authentication. Typically, the end entity is issued one or more shared secret keys used for subsequent authentication.

- **Initialization:** Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure. For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s), to be used in validating certificate paths.
- **Certification:** This is the process in which a CA issues a certificate for a user's public key, and returns that certificate to the user's client system and/or posts that certificate in a repository.
- **Key pair recovery:** Key pairs can be used to support digital signature creation and verification, encryption and decryption, or both. When a key pair is used for encryption/decryption, it is

important to provide a mechanism to recover the necessary decryption keys when normal access to the keying material is no longer possible, otherwise it will not be possible to recover the encrypted data. Loss of access to the decryption key can result from forgotten passwords/PINs, corrupted disk drives, damage to hardware tokens, and so on. Key pair recovery allows end entities to restore their encryption/decryption key pair from an authorized key backup facility (typically, the CA that issued the End Entity's certificate).

- **Key pair update:** All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires and as a result of certificate revocation.
- **Revocation request:** An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private key compromise, change in affiliation, and name change.
- **Cross certification:** Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.

PKIX Management Protocols

The PKIX working group has defines two alternative management protocols between PKIX entities that support the management functions listed in the preceding subsection. RFC 2510 defines the certificate management protocols (CMP). Within CMP, each of the management functions is explicitly identified by specific protocol exchanges. CMP is designed to be a flexible protocol able to accommodate a variety of technical, operational, and business models.

RFC 2797 defines certificate management messages over CMS (CMC), where CMS refers to RFC 2630, cryptographic message syntax. CMC is built on earlier work and is intended to leverage existing implementations. Although all of the PKIX functions are supported, the functions do not all map into specific protocol exchanges.

14.4. Recommended Reading and Web Sites

A painless way to get a grasp of Kerberos concepts is found in [BRYA88]. One of the best treatments of Kerberos is [KOHL94]. [TUNG99] describes Kerberos from a user's point of view.

[PERL99] reviews various trust models that can be used in a PKI. [GUTM02] highlights difficulties in PKI use and recommends approaches for an effective PKI.

BRYA88 Bryant, W. *Designing an Authentication System: A Dialogue in Four Scenes*. Project Athena document, February 1988. Available at **http://web.mit.edu/kerberos/www/dialogue.html**.

GUTM02 Gutmann, P. "PKI: It's Not Dead, Just Resting." *Computer*, August 2002.

KOHL94 Kohl, J.; Neuman, B.; and Ts'o, T. "The Evolution of the Kerberos Authentication Service." in Brazier, F., and Johansen, D. *Distributed Open Systems*. Los Alamitos, CA: IEEE Computer Society Press, 1994. Available at **http://web.mit.edu/kerberos/www/papers.html**.

PERL99 Perlman, R. "An Overview of PKI Trust Models." *IEEE Network*, November/December 1999.

TUNG99 Tung, B. *Kerberos: A Network Authentication System*. Reading, MA: Addison-Wesley, 1999.

Recommended Web Sites

- **MIT Kerberos Site:** Information about Kerberos, including the FAQ, papers and documents, and pointers to commercial product sites
- **USC/ISI Kerberos Page:** Another good source of Kerberos material
- **Kerberos Working Group:** IETF group developing standards based on Kerberos
- **Public-Key Infrastructure Working Group:** IETF group developing standards based on X.509v3
- **Verisign:** A leading commercial vendor of X.509-related products; white papers and other worthwhile material at this site
- **NIST PKI Program:** Good source of information

14.5. Key Terms, Review Questions, and Problems

Key Terms

authentication

authentication server

Kerberos

Kerberos realm

lifetime

nonce

propagating cipher block chaining (PCBC) mode

public-key certificate

realm

sequence number

subkey

ticket

ticket-granting server (TGS)

X.509 certificate

Review Questions

14.1 What problem was Kerberos designed to address?

14.2 What are three threats associated with user authentication over a network or Internet?

14.3 List three approaches to secure user authentication in a distributed environment.

14.4 What four requirements were defined for Kerberos?

14.5 What entities constitute a full-service Kerberos environment?

14.6 In the context of Kerberos, what is a realm?

14.7 What are the principal differences between version 4 and version 5 of Kerberos?

14.8 What is the purpose of the X.509 standard?

14.9 What is a chain of certificates?

14.10 How is an X.509 certificate revoked?

Problems

14.1 Show that a random error in one block of ciphertext is propagated to all subsequent blocks of plaintext in PCBC mode (Figure 14.9).

14.2 Suppose that, in PCBC mode, blocks C_i and C_{i+1} are interchanged during transmission. Show that this affects only the decrypted blocks P_i and P_{i+1} but not subsequent blocks.

14.3 The original three-way authentication procedure for X.509 illustrated in Figure 14.6c contains a security flaw. The essence of the protocol is as follows:

$$A \rightarrow B: \quad A\{t_A, r_A, ID_B\}$$

$$B \rightarrow A: \quad B\{t_B, r_B, ID_A, r_A\}$$

$$A \rightarrow B: \quad A\{r_B\}$$

The text of X.509 states that checking timestamps t_A and t_B is optional for three-way authentication. But consider the following example: Suppose A and B have used the preceding protocol on some previous occasion, and that opponent C has intercepted the preceding three messages. In addition, suppose that timestamps are not used and are all set to 0. Finally, suppose C wishes to impersonate A to B. C initially sends the first captured message to B:

C ⟶ B: $A\{0, r_A, ID_B\}$

B responds, thinking it is talking to A but is actually talking to C:

B ⟶ C: $B\{0, r'_B, ID_A, r_A\}$

C meanwhile causes A to initiate authentication with C by some means. As a result, A sends C the following:

A ⟶ C: $A\{0, r'_A, ID_C\}$

C responds to A using the same nonce provided to C by B.

C ⟶ A: $C\{0,r'_B, ID_A, r'_A\}$

A responds with

A ⟶ C: $A\{r'_B\}$

This is exactly what C needs to convince B that it is talking to A, so C now repeats the incoming message back out to B.

C ⟶ B: $A\{r'_B\}$

So B will believe it is talking to A whereas it is actually talking to C. Suggest a simple solution to this problem that does not involve the use of timestamps.

14.4 The 1988 version of X.509 lists properties that RSA keys must satisfy to be secure, given current knowledge about the difficulty of factoring large numbers. The discussion concludes with a constraint on the public exponent and the modulus n:

> It must be ensured that $e > \log_2(n)$ to prevent attack by taking the$e$ th root mod n to disclose the plaintext.

Although the constraint is correct, the reason given for requiring it is incorrect. What is wrong with the reason given and what is the correct reason?

Appendix 14A Kerberos Encryption Techniques

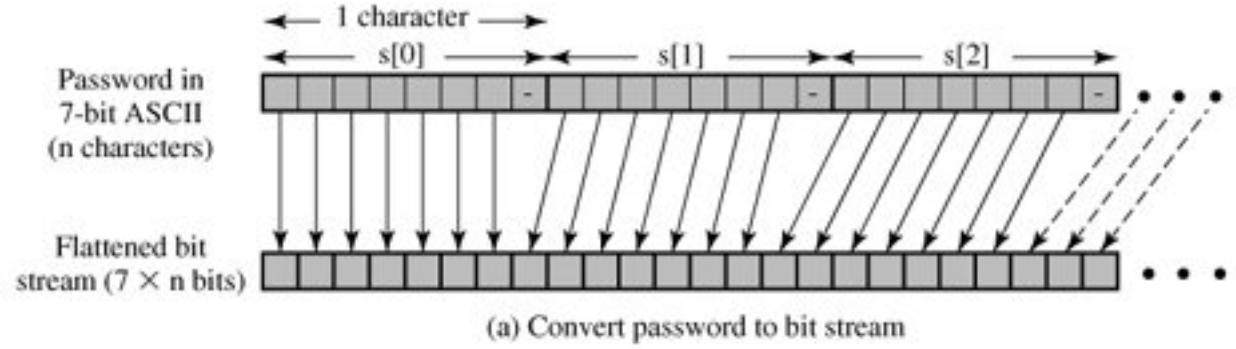
Kerberos includes an encryption library that supports various encryption-related operations. These were included in the Kerberos 5 specification and are common in commercial implementations. In February 2005, IETF issued RFCs 3961 and 3962, which expand the options of cryptographic techniques. In this appendix, we describe the RFC 1510 techniques.

Password-to-Key Transformation

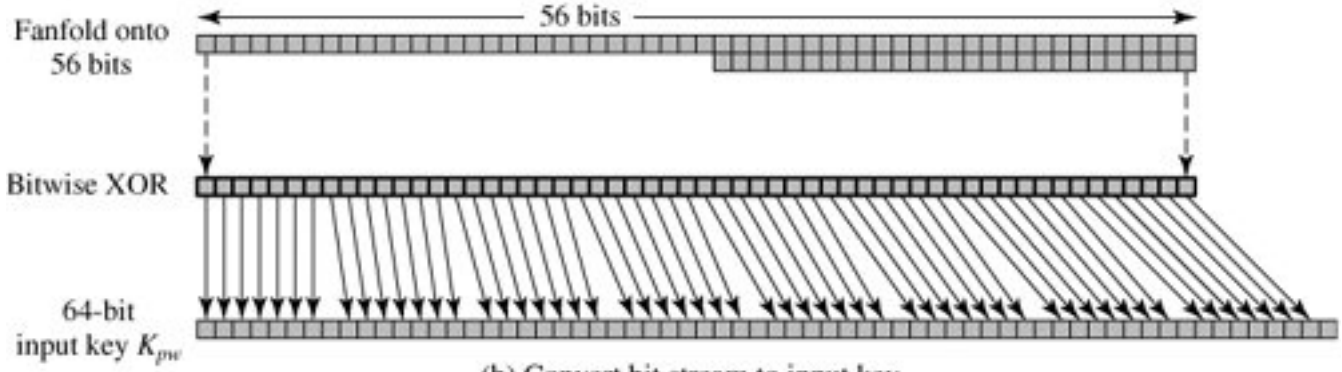
In Kerberos, passwords are limited to the use of the characters that can be represented in a 7-bit ASCII format. This password, of arbitrary length, is converted into an encryption key that is stored in the Kerberos database. Figure 14.8 illustrates the procedure.

Figure 14.8. Generation of Encryption Key from Password

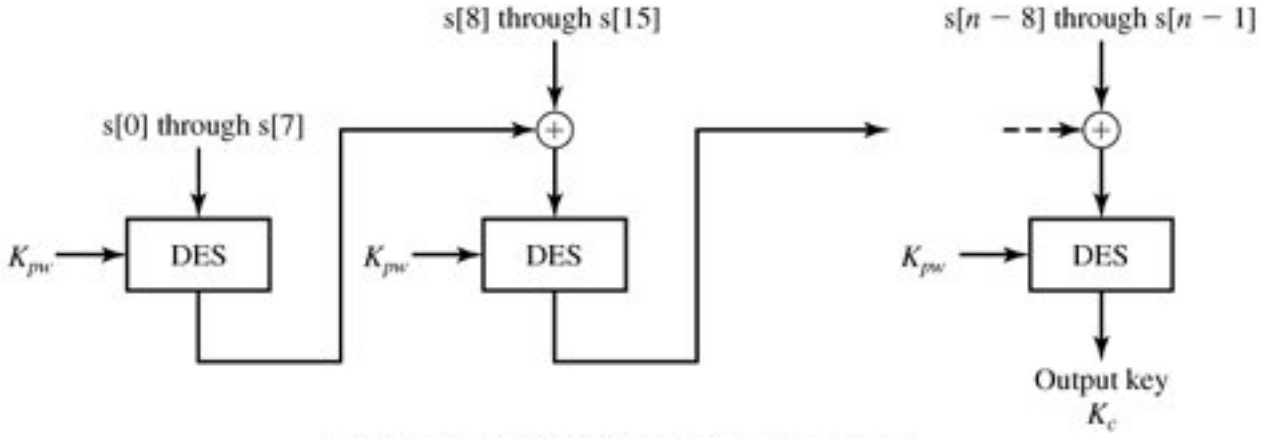
[View full size image]



(a) Convert password to bit stream



(b) Convert bit stream to input key



(c) Generate DES CBC checksum of password

First, the character string, s, is packed into a bit string, b, such that the first character is stored in the first 7 bits, the second character in the second 7 bits, and so on. This can be expressed as

b[0] = bit 0 of s[0]

...

b[6] = bit 6 of s[0]

b[7] = bit 0 of s[1]

...

$$b[7i + m] = \text{bit } m \text{ of } s[i] \quad 0 \leq m \leq 6$$

Next, the bit string is compacted to 56 bits by aligning the bits in "fanfold" fashion and performing a bitwise XOR. For example, if the bit string is of length 59, then

$$b[55] = b[55] \oplus b[56]$$
$$b[54] = b[54] \oplus b[57]$$
$$b[53] = b[53] \oplus b[58]$$

This creates a 56-bit DES key. To conform to the expected 64-bit key format, the string is treated as a sequence of eight 7-bit blocks and is mapped into eight 8-bit blocks to form an input key K_{pw}

Finally, the original password is encrypted using the cipher block chaining (CBC) mode of DES with key K_{pw} The last 64-bit block returned from this process, known as the CBC checksum, is the output key associated with this password.

The entire algorithm can be viewed as a hash function that maps an arbitrary password into a 64-bit hash code.

Propagating Cipher Block Chaining Mode

Recall from Chapter 6 that, in the CBC mode of DES, the input to the DES algorithm at each stage consists of the XOR of the current plaintext block and the preceding ciphertext block, with the same key used for each block (Figure 6.4). The advantage of this mode over the electronic codebook (ECB) mode, in which each plaintext block is independently encrypted, is this: With CBC, the same plaintext block, if repeated, produces different ciphertext blocks.

CBC has the property that if an error occurs in transmission of ciphertext block C_I then this error propagates to the recovered plaintext blocks P_I and P_{I+1}.

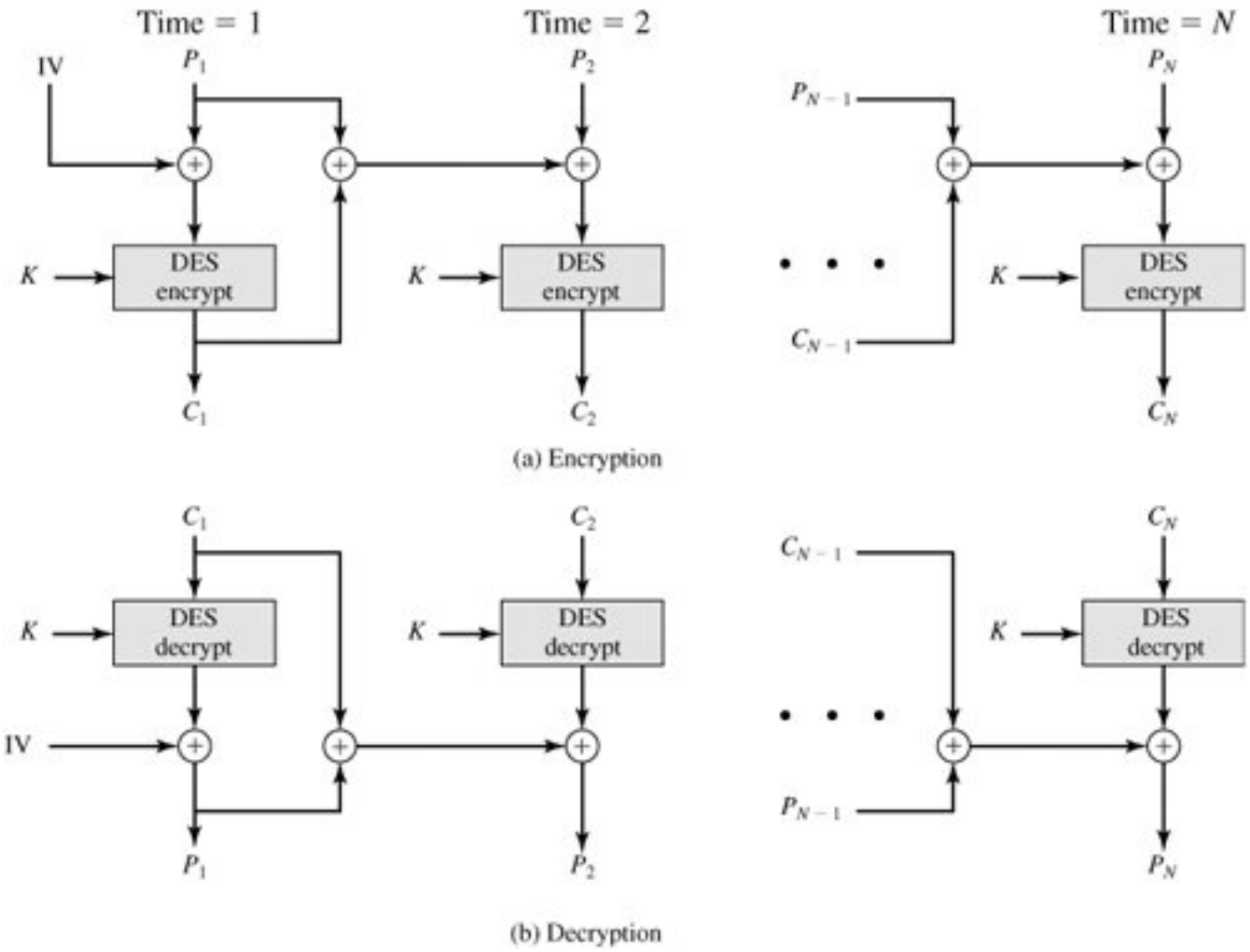
Version 4 of Kerberos uses an extension to CBC, called the propagating CBC (PCBC) mode [MEYE82]. This mode has the property that an error in one ciphertext block is propagated to all subsequent decrypted blocks of the message, rendering each block useless. Thus, data encryption and integrity are combined in one operation. (For an exception, see Problem 14.2).

PCBC is illustrated in Figure 14.9. In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block, the preceding cipher text block, and the preceding plaintext block:

$$C_n = E(K, [C_{n-1} \oplus P_{n-1} \oplus P_n])$$

Figure 14.9. Propagating Cipher Block Chaining (PCBC) Mode

[View full size image]



On decryption, each ciphertext block is passed through the decryption algorithm. Then the output is XORed with the preceding ciphertext block and the preceding plaintext block. We can demonstrate that this scheme works, as follows:

$$D(K, C_n) = D(K, E(K, [C_{n-1} \oplus P_{n-1} \oplus P_n]))$$

$$D(K, C_n) = C_{n-1} \oplus P_n\text{-}1 \oplus$$

$$P_n$$

$$C_n\text{-}1 \oplus P_{n-1} \oplus D(K, C_n) =$$

$$P_n$$

Chapter 15. Electronic Mail Security

Despite the refusal of VADM Poindexter and LtCol North to appear, the Board's access to other sources of information filled much of this gap. The FBI provided documents taken from the files of the National Security Advisor and relevant NSC staff members, including messages from the PROF system between VADM Poindexter and LtCol North. The PROF messages were conversations by computer, written at the time events occurred and presumed by the writers to be protected from disclosure. In this sense, they provide a first-hand, contemporaneous account of events.

The Tower Commission Report to President Reagan on the Iran-Contra Affair, 1987

Bless the man who made it, And pray that he ain't dead. He could've made a million If he'd sold it to the feds, But he was hot for freedom; He gave it out for free. Now every common citizen's got PGP.

From the song "P.G.P."by Leslie Fish

Key Points

- PGP is an open-source freely available software package for e-mail security. It provides authentication through the use of digital signature; confidentiality through the use of symmetric block encryption; compression using the ZIP algorithm; e-mail compatibility using the radix-64 encoding scheme; and segmentation and reassembly to accommodate long e-mails.
- PGP incorporates tools for developing a public-key trust model and public-key certificate management.
- S/MIME is an Internet standard approach to e-mail security that incorporates the same functionality as PGP.

In virtually all distributed environments, electronic mail is the most heavily used network-based application. It is also the only distributed application that is widely used across all architectures and vendor platforms. Users expect to be able to, and do, send mail to others who are connected directly or indirectly to the Internet, regardless of host operating system or communications suite.

With the explosively growing reliance on electronic mail for every conceivable purpose, there grows a demand for authentication and confidentiality services. Two schemes stand out as approaches that enjoy widespread use: Pretty Good Privacy (PGP) and S/MIME. Both are examined in this chapter.

PREV NEXT

15.1. Pretty Good Privacy

PGP is a remarkable phenomenon. Largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. In essence, Zimmermann has done the following:

1. Selected the best available cryptographic algorithms as building blocks
2. Integrated these algorithms into a general-purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands
3. Made the package and its documentation, including the source code, freely available via the Internet, bulletin boards, and commercial networks such as AOL (America On Line)
4. Entered into an agreement with a company (Viacrypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP

PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth:

1. It is available free worldwide in versions that run on a variety of platforms, including Windows, UNIX, Macintosh, and many more. In addition, the commercial version satisfies users who want a product that comes with vendor support.
2. It is based on algorithms that have survived extensive public review and are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES for symmetric encryption; and SHA-1 for hash coding.
3. It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.
4.

It was not developed by, nor is it controlled by, any governmental or standards organization. For those with an instinctive distrust of "the establishment," this makes PGP attractive.

5.

PGP is now on an Internet standards track (RFC 3156). Nevertheless, PGP still has an aura of an antiestablishment endeavor.

We begin with an overall look at the operation of PGP. Next, we examine how cryptographic keys are created and stored. Then, we address the vital issue of public key management.

Notation

Most of the notation used in this chapter has been used before, but a few terms are new. It is perhaps best to summarize those at the beginning. The following symbols are used:

K_s	=session key used in symmetric encryption scheme
PR_a	=private key of user A, used in public-key encryption scheme
PU_a	=public key of user A, used in public-key encryption scheme
EP	= public-key encryption
DP	= public-key decryption
EC	= symmetric encryption
DC	= symmetric decryption
H	= hash function
||	= concatenation
Z	= compression using ZIP algorithm
R64	= conversion to radix 64 ASCII format

The PGP documentation often uses the term *secret key* to refer to a key paired with a public key in a public-key encryption scheme. As was mentioned earlier, this practice risks confusion with a secret key used for symmetric encryption. Hence, we will use the term *private key* instead.

Operational Description

The actual operation of PGP, as opposed to the management of keys, consists of five services: authentication, confidentiality, compression, e-mail compatibility, and segmentation (Table 15.1). We examine each of these in turn.

Table 15.1. Summary of PGP Services

(This item is displayed on page 440 in the print version)

Function	Algorithms	Used Description
Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key and included with the message.
Message encryption	CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key and included with the message.
Compression	ZIP	A message may be compressed, for storage or transmission, using ZIP.
Email compatibility	Radix 64 conversion	To provide transparency for email applications, an encrypted message may be converted to an ASCII string using radix 64 conversion.
Segmentation		To accommodate maximum message size limitations, PGP performs segmentation and reassembly.

Authentication

Figure 15.1a illustrates the digital signature service provided by PGP. This is the digital signature scheme discussed in Chapter 13 and illustrated in Figure 11.5c. The sequence is as follows:

1. The sender creates a message.
2. SHA-1 is used to generate a 160-bit hash code of the message.
3. The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.
4.

The receiver uses RSA with the sender's public key to decrypt and recover the hash code.

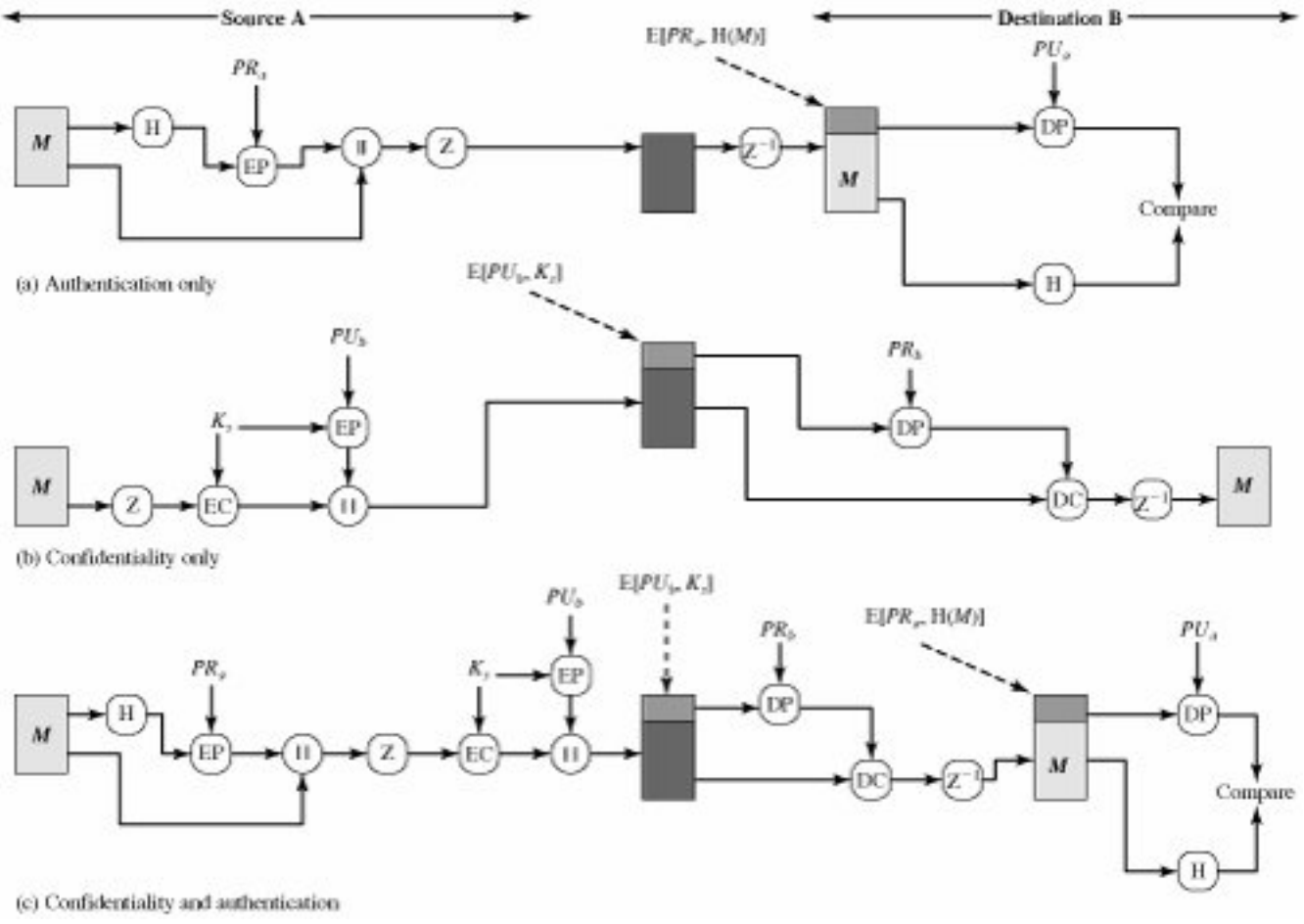
5.

The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.

Figure 15.1. PGP Cryptographic Functions

(This item is displayed on page 441 in the print version)

[View full size image]



The combination of SHA-1 and RSA provides an effective digital signature scheme. Because of the strength of RSA, the recipient is assured that only the possessor of the matching private key can generate the signature. Because of the strength of SHA-1, the recipient is assured that no one else could generate a new message that matches the hash code and, hence, the signature of the original message.

As an alternative, signatures can be generated using DSS/SHA-1.

Although signatures normally are found attached to the message or file that they sign, this is not always the case: Detached signatures are supported. A detached signature may be stored and transmitted

separately from the message it signs. This is useful in several contexts. A user may wish to maintain a separate signature log of all messages sent or received. A detached signature of an executable program can detect subsequent virus infection. Finally, detached signatures can be used when more than one party must sign a document, such as a legal contract. Each person's signature is independent and therefore is applied only to the document. Otherwise, signatures would have to be nested, with the second signer signing both the document and the first signature, and so on.

Confidentiality

Another basic service provided by PGP is confidentiality, which is provided by encrypting messages to be transmitted or to be stored locally as files. In both cases, the symmetric encryption algorithm CAST-128 may be used. Alternatively, IDEA or 3DES may be used. The 64-bit cipher feedback (CFB) mode is used.

As always, one must address the problem of key distribution. In PGP, each symmetric key is used only once. That is, a new key is generated as a random 128-bit number for each message. Thus, although this is referred to in the documentation as a session key, it is in reality a one-time key. Because it is to be used only once, the session key is bound to the message and transmitted with it. To protect the key, it is encrypted with the receiver's public key. Figure 15.1b illustrates the sequence, which can be described as follows:

1. The sender generates a message and a random 128-bit number to be used as a session key for this message only.

---

2. The message is encrypted, using CAST-128 (or IDEA or 3DES) with the session key.
3. The session key is encrypted with RSA, using the recipient's public key, and is prepended to the message.
4. The receiver uses RSA with its private key to decrypt and recover the session key.
5. The session key is used to decrypt the message.

As an alternative to the use of RSA for key encryption, PGP provides an option referred to as *Diffie-Hellman*. As was explained in Chapter 10, Diffie-Hellman is a key exchange algorithm. In fact, PGP uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal (see Problem 10.6).

Several observations may be made. First, to reduce encryption time the combination of symmetric and public-key encryption is used in preference to simply using RSA or ElGamal to encrypt the message directly: CAST-128 and the other symmetric algorithms are substantially faster than RSA or ElGamal.

Second, the use of the public-key algorithm solves the session key distribution problem, because only the recipient is able to recover the session key that is bound to the message. Note that we do not need a session key exchange protocol of the type discussed in Chapter 10, because we are not beginning an ongoing session. Rather, each message is a one-time independent event with its own key. Furthermore, given the store-and-forward nature of electronic mail, the use of handshaking to assure that both sides have the same session key is not practical. Finally, the use of one-time symmetric keys strengthens what is already a strong symmetric encryption approach. Only a small amount of plaintext is encrypted with each key, and there is no relationship among the keys. Thus, to the extent that the public-key algorithm is secure, the entire scheme is secure. To this end, PGP provides the user with a range of key size options from 768 to 3072 bits (the DSS key for signatures is limited to 1024 bits).

Confidentiality and Authentication

As Figure 15.1c illustrates, both services may be used for the same message. First, a signature is generated for the plaintext message and prepended to the message. Then the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES), and the session key is encrypted using RSA (or ElGamal). This sequence is preferable to the opposite: encrypting the message and then generating a signature for the encrypted message. It is generally more convenient to store a signature with a plaintext version of a message. Furthermore, for purposes of third-party verification, if the signature is performed first, a third party need not be concerned with the symmetric key when verifying the signature.

In summary, when both services are used, the sender first signs the message with its own private key, then encrypts the message with a session key, and then encrypts the session key with the recipient's public key.

Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.

The placement of the compression algorithm, indicated by Z for compression and Z^{-1} for decompression in Figure 15.1, is critical:

1. The signature is generated before compression for two reasons:
   a. It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification. If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.
   b. Even if one were willing to generate dynamically a recompressed message for verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic;

various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and, as a result, produce different compressed forms. However, these different compression algorithms are interoperable because any version of the algorithm can correctly decompress the output of any other version. Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm.

2.

Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult.

The compression algorithm used is ZIP, which is described in Appendix 15A.

E-mail Compatibility

When PGP is used, at least part of the block to be transmitted is encrypted. If only the signature service is used, then the message digest is encrypted (with the sender's private key). If the confidentiality service is used, the message plus signature (if present) are encrypted (with a one-time symmetric key). Thus, part or all of the resulting block consists of a stream of arbitrary 8-bit octets. However, many electronic mail systems only permit the use of blocks consisting of ASCII text. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters.

The scheme used for this purpose is radix-64 conversion. Each group of three octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors. See Appendix 15B for a description.

The use of radix 64 expands a message by 33%. Fortunately, the session key and signature portions of the message are relatively compact, and the plaintext message has been compressed. In fact, the compression should be more than enough to compensate for the radix-64 expansion. For example, [HELD96] reports an average compression ratio of about 2.0 using ZIP. If we ignore the relatively small signature and key components, the typical overall effect of compression and expansion of a file of length X would be 1.33 x 0.5 x X = 0.665 x X. Thus, there is still an overall compression of about one-third.

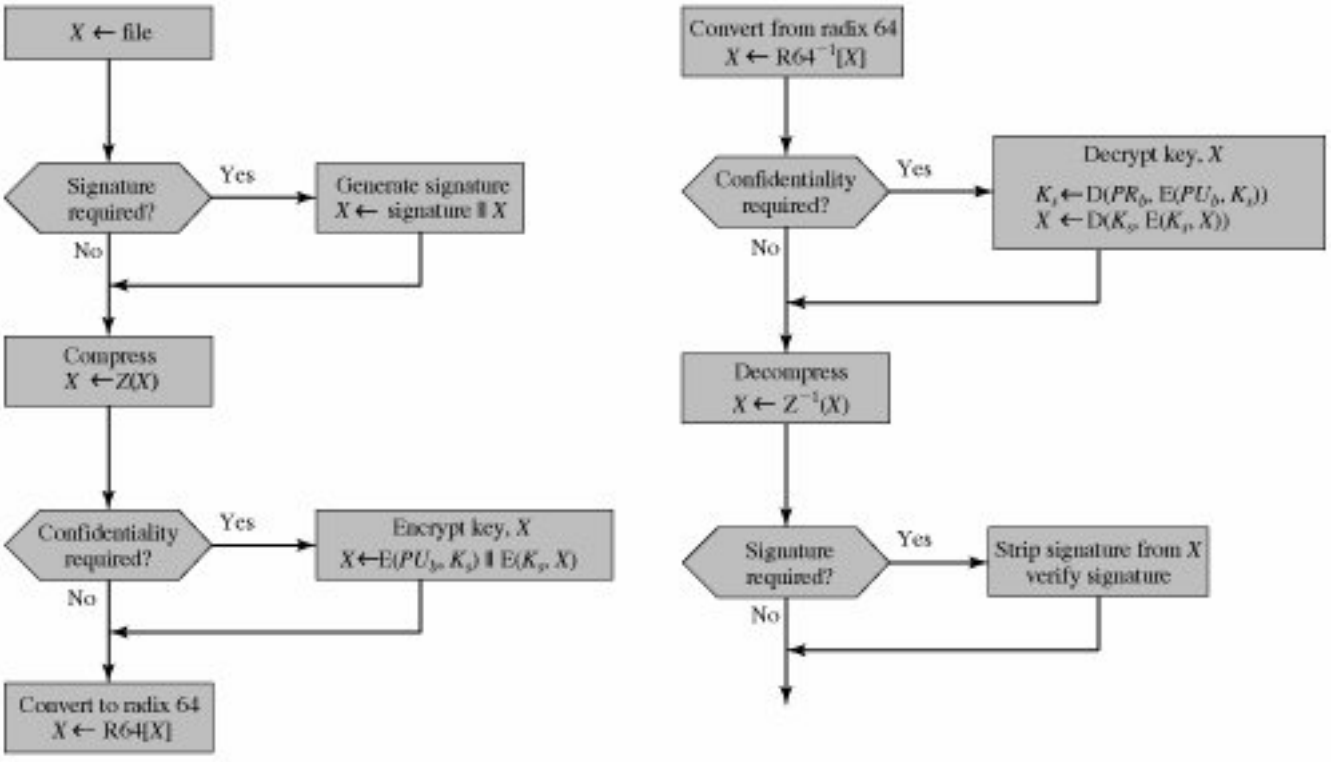
One noteworthy aspect of the radix-64 algorithm is that it blindly converts the input stream to radix-64 format regardless of content, even if the input happens to be ASCII text. Thus, if a message is signed but not encrypted and the conversion is applied to the entire block, the output will be unreadable to the casual observer, which provides a certain level of confidentiality. As an option, PGP can be configured to convert to radix-64 format only the signature portion of signed plaintext messages. This enables the human recipient to read the message without using PGP. PGP would still have to be used to verify the signature.

Figure 15.2 shows the relationship among the four services so far discussed. On transmission, if it is required, a signature is generated using a hash code of the uncompressed plaintext. Then the plaintext, plus signature if present, is compressed. Next, if confidentiality is required, the block (compressed plaintext or compressed signature plus plaintext) is encrypted and prepended with the public-key-encrypted symmetric encryption key. Finally, the entire block is converted to radix-64 format.

Figure 15.2. Transmission and Reception of PGP Messages

(This item is displayed on page 445 in the print version)

[View full size image]



(a) Generic transmission diagram (from A)

(b) Generic reception diagram (to B)

On reception, the incoming block is first converted back from radix-64 format to binary. Then, if the message is encrypted, the recipient recovers the session key and decrypts the message. The resulting block is then decompressed. If the message is signed, the recipient recovers the transmitted hash code and compares it to its own calculation of the hash code.

Segmentation and Reassembly

E-mail facilities often are restricted to a maximum message length. For example, many of the facilities accessible through the Internet impose a maximum length of 50,000 octets. Any message longer than that must be broken up into smaller segments, each of which is mailed separately.

To accommodate this restriction, PGP automatically subdivides a message that is too large into segments that are small enough to send via e-mail. The segmentation is done after all of the other processing, including the radix-64 conversion. Thus, the session key component and signature component appear only once, at the beginning of the first segment. At the receiving end, PGP must strip off all e-mail headers and reassemble the entire original block before performing the steps illustrated in Figure 15.2b.

Cryptographic Keys and Key Rings

PGP makes use of four types of keys: one-time session symmetric keys, public keys, private keys, and passphrase-based symmetric keys (explained subsequently). Three separate requirements can be identified with respect to these keys:

1.

A means of generating unpredictable session keys is needed.

2.

We would like to allow a user to have multiple public-key/private-key pairs. One reason is that the user may wish to change his or her key pair from time to time. When this happens, any messages in the pipeline will be constructed with an obsolete key. Furthermore, recipients will know only the old public key until an update reaches them. In addition to the need to change keys over time, a user may wish to have multiple key pairs at a given time to interact with different groups of correspondents or simply to enhance security by limiting the amount of material encrypted with any one key. The upshot of all this is that there is not a one-to-one correspondence between users and their public keys. Thus, some means is needed for identifying particular keys.

3.

Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

We examine each of these requirements in turn.

Session Key Generation

Each session key is associated with a single message and is used only for the purpose of encrypting and decrypting that message. Recall that message encryption/decryption is done with a symmetric encryption algorithm. CAST-128 and IDEA use 128-bit keys; 3DES uses a 168-bit key. For the following discussion, we assume CAST-128.

Random 128-bit numbers are generated using CAST-128 itself. The input to the random number generator consists of a 128-bit key and two 64-bit blocks that are treated as plaintext to be encrypted. Using cipher feedback mode, the CAST-128 encrypter produces two 64-bit cipher text blocks, which are concatenated to form the 128-bit session key. The algorithm that is used is based on the one specified in ANSI X12.17.

The "plaintext" input to the random number generator, consisting of two 64-bit blocks, is itself derived from a stream of 128-bit randomized numbers. These numbers are based on keystroke input from the user. Both the keystroke timing and the actual keys struck are used to generate the randomized stream. Thus, if the user hits arbitrary keys at his or her normal pace, a reasonably "random" input will be generated. This random input is also combined with previous session key output from CAST-128 to form the key input to the generator. The result, given the effective scrambling of CAST-128, is to produce a sequence of session keys that is effectively unpredictable.

Appendix 15C discusses PGP random number generation techniques in more detail.

Key Identifiers

As we have discussed, an encrypted message is accompanied by an encrypted form of the session key that was used for message encryption. The session key itself is encrypted with the recipient's public key. Hence, only the recipient will be able to recover the session key and therefore recover the message. If

each user employed a single public/private key pair, then the recipient would automatically know which key to use to decrypt the session key: the recipient's unique private key. However, we have stated a requirement that any given user may have multiple public/private key pairs.

How, then, does the recipient know which of its public keys was used to encrypt the session key? One simple solution would be to transmit the public key with the message. The recipient could then verify that this is indeed one of its public keys, and proceed. This scheme would work, but it is unnecessarily wasteful of space. An RSA public key may be hundreds of decimal digits in length. Another solution would be to associate an identifier with each public key that is unique at least within one user. That is, the combination of user ID and key ID would be sufficient to identify a key uniquely. Then only the much shorter key ID would need to be transmitted. This solution, however, raises a management and overhead problem: Key IDs must be assigned and stored so that both sender and recipient could map from key ID to public key. This seems unnecessarily burdensome.

The solution adopted by PGP is to assign a key ID to each public key that is, with very high probability, unique within a user ID.[1] The key ID associated with each public key consists of its least significant 64 bits. That is, the key ID of public PU_a is (PU_a mod 2^{64}). This is a sufficient length that the probability of duplicate key IDs is very small.

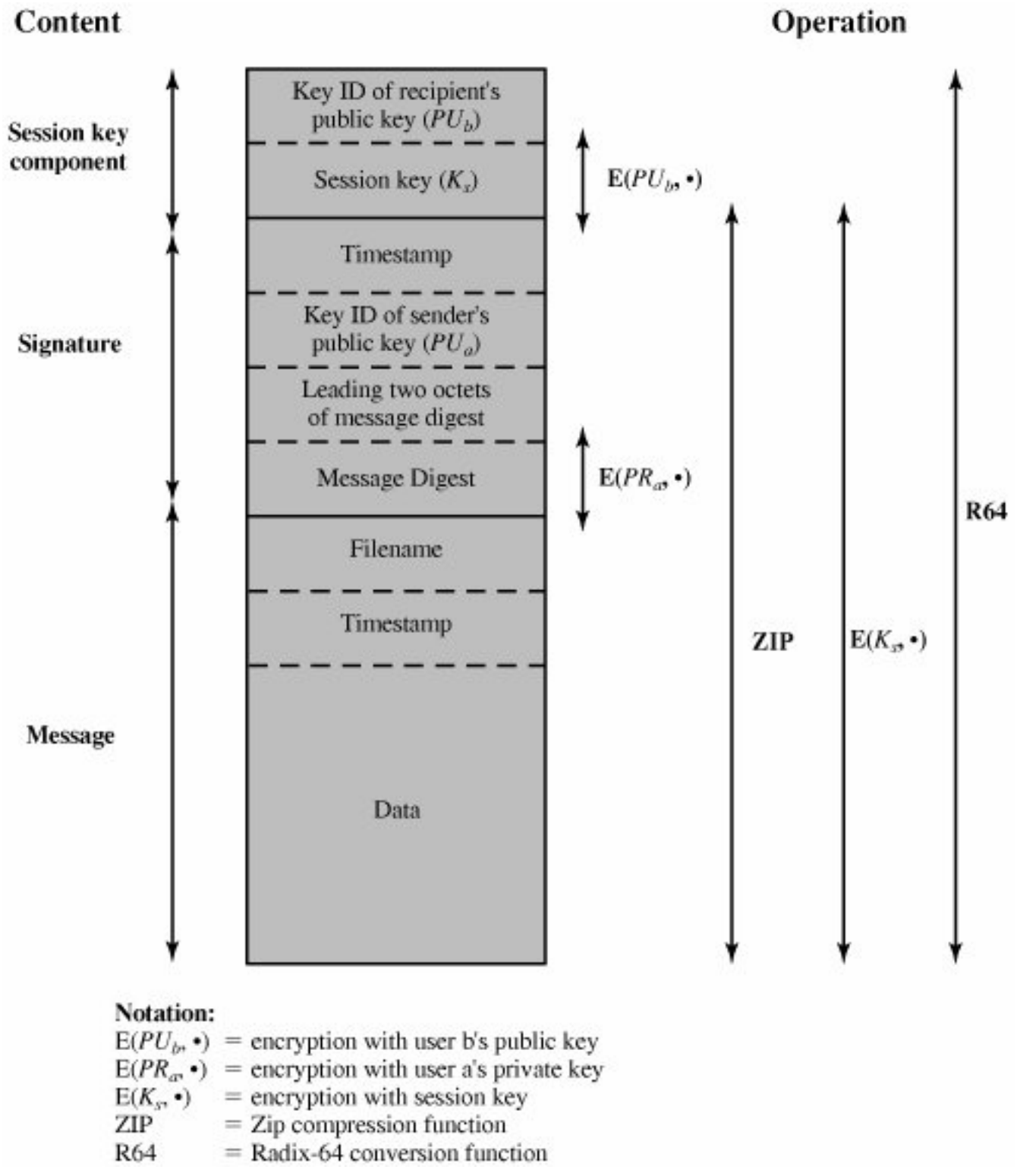
[1] We have seen this introduction of probabilistic concepts before, in Section 8.3, for determining whether a number is prime. It is often the case in designing algorithms that the use of probabilistic techniques results in a less time-consuming or less complex solution, or both.

A key ID is also required for the PGP digital signature. Because a sender may use one of a number of private keys to encrypt the message digest, the recipient must know which public key is intended for use. Accordingly, the digital signature component of a message includes the 64-bit key ID of the required public key. When the message is received, the recipient verifies that the key ID is for a public key that it knows for that sender and then proceeds to verify the signature.

Now that the concept of key ID has been introduced, we can take a more detailed look at the format of a transmitted message, which is shown in Figure 15.3. A message consists of three components: the message component, a signature (optional), and a session key component (optional).

Figure 15.3. General Format of PGP Message (from A to B)

[View full size image]



The **message component** includes the actual data to be stored or transmitted, as well as a filename and a timestamp that specifies the time of creation.

The **signature component** includes the following:

- **Timestamp:** The time at which the signature was made.
- **Message digest:** The 160-bit SHA-1 digest, encrypted with the sender's private signature key. The digest is calculated over the signature timestamp concatenated with the data portion of the

message component. The inclusion of the signature timestamp in the digest assures against replay types of attacks. The exclusion of the filename and timestamp portions of the message component ensures that detached signatures are exactly the same as attached signatures prefixed to the message. Detached signatures are calculated on a separate file that has none of the message component header fields.

- **Leading two octets of message digest:** To enable the recipient to determine if the correct public key was used to decrypt the message digest for authentication, by comparing this plaintext copy of the first two octets with the first two octets of the decrypted digest. These octets also serve as a 16-bit frame check sequence for the message.
- **Key ID of sender's public key:** Identifies the public key that should be used to decrypt the message digest and, hence, identifies the private key that was used to encrypt the message digest.

The message component and optional signature component may be compressed using ZIP and may be encrypted using a session key.

The **session key component** includes the session key and the identifier of the recipient's public key that was used by the sender to encrypt the session key.

The entire block is usually encoded with radix-64 encoding.

Key Rings

We have seen how key IDs are critical to the operation of PGP and that two key IDs are included in any PGP message that provides both confidentiality and authentication. These keys need to be stored and organized in a systematic way for efficient and effective use by all parties. The scheme used in PGP is to provide a pair of data structures at each node, one to store the public/private key pairs owned by that node and one to store the public keys of other users known at this node. These data structures are referred to, respectively, as the private-key ring and the public-key ring.

Figure 15.4 shows the general structure of a **private-key ring**. We can view the ring as a table, in which each row represents one of the public/private key pairs owned by this user. Each row contains the following entries:

- **Timestamp:** The date/time when this key pair was generated.
- **Key ID:** The least significant 64 bits of the public key for this entry.
- **Public key:** The public-key portion of the pair.
- **Private key:** The private-key portion of the pair; this field is encrypted.
- **User ID:** Typically, this will be the user's e-mail address (e.g., **stallings@acm.org**). However, the user may choose to associate a different name with each pair (e.g., Stallings, WStallings, WilliamStallings, etc.) or to reuse the same User ID more than once.

Figure 15.4. General Structure of Private- and Public-Key Rings

[View full size image]

Private-Key Ring

Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
• • •	• • •	• • •	• • •	• • •
T_i	$PU_i \bmod 2^{64}$	PU_i	$E(H(P_i), PR_i)$	User i
• • •	• • •	• • •	• • •	• • •

Public-Key Ring

Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •
T_i	$PU_i \bmod 2^{64}$	PU_i	$trust\_flag_i$	User i	$trust\_flag_i$		
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •

* = field used to index table

The private-key ring can be indexed by either User ID or Key ID; later we will see the need for both means of indexing.

Although it is intended that the private-key ring be stored only on the machine of the user that created and owns the key pairs, and that it be accessible only to that user, it makes sense to make the value of the private key as secure as possible. Accordingly, the private key itself is not stored in the key ring. Rather, this key is encrypted using CAST-128 (or IDEA or 3DES). The procedure is as follows:

1. The user selects a passphrase to be used for encrypting private keys.

2. When the system generates a new public/private key pair using RSA, it asks the user for the passphrase. Using SHA-1, a 160-bit hash code is generated from the passphrase, and the passphrase is discarded.

3. The system encrypts the private key using CAST-128 with the 128 bits of the hash code as the key. The hash code is then discarded, and the encrypted private key is stored in the private-key ring.

Subsequently, when a user accesses the private-key ring to retrieve a private key, he or she must supply the passphrase. PGP will retrieve the encrypted private key, generate the hash code of the passphrase, and decrypt the encrypted private key using CAST-128 with the hash code.

This is a very compact and effective scheme. As in any system based on passwords, the security of this

system depends on the security of the password. To avoid the temptation to write it down, the user should use a passphrase that is not easily guessed but that is easily remembered.

Figure 15.4 also shows the general structure of a **public-key ring**. This data structure is used to store public keys of other users that are known to this user. For the moment, let us ignore some fields shown in the table and describe the following fields:

- **Timestamp:** The date/time when this entry was generated.
- **Key ID:** The least significant 64 bits of the public key for this entry.
- **Public Key:** The public key for this entry.
- **User ID:** Identifies the owner of this key. Multiple user IDs may be associated with a single public key.

The public-key ring can be indexed by either User ID or Key ID; we will see the need for both means of indexing later.

We are now in a position to show how these key rings are used in message transmission and reception. For simplicity, we ignore compression and radix-64 conversion in the following discussion. First consider message transmission (Figure 15.5) and assume that the message is to be both signed and encrypted. The sending PGP entity performs the following steps:

---

1. Signing the message

   a. PGP retrieves the sender's private key from the private-key ring using your_userid as an index. If your_userid was not provided in the command, the first private key on the ring is retrieved.

   b. PGP prompts the user for the passphrase to recover the unencrypted private key.

   c. The signature component of the message is constructed.

2. Encrypting the message

   a.

      PGP generates a session key and encrypts the message.

   b.

      PGP retrieves the recipient's public key from the public-key ring using her_userid as an index.

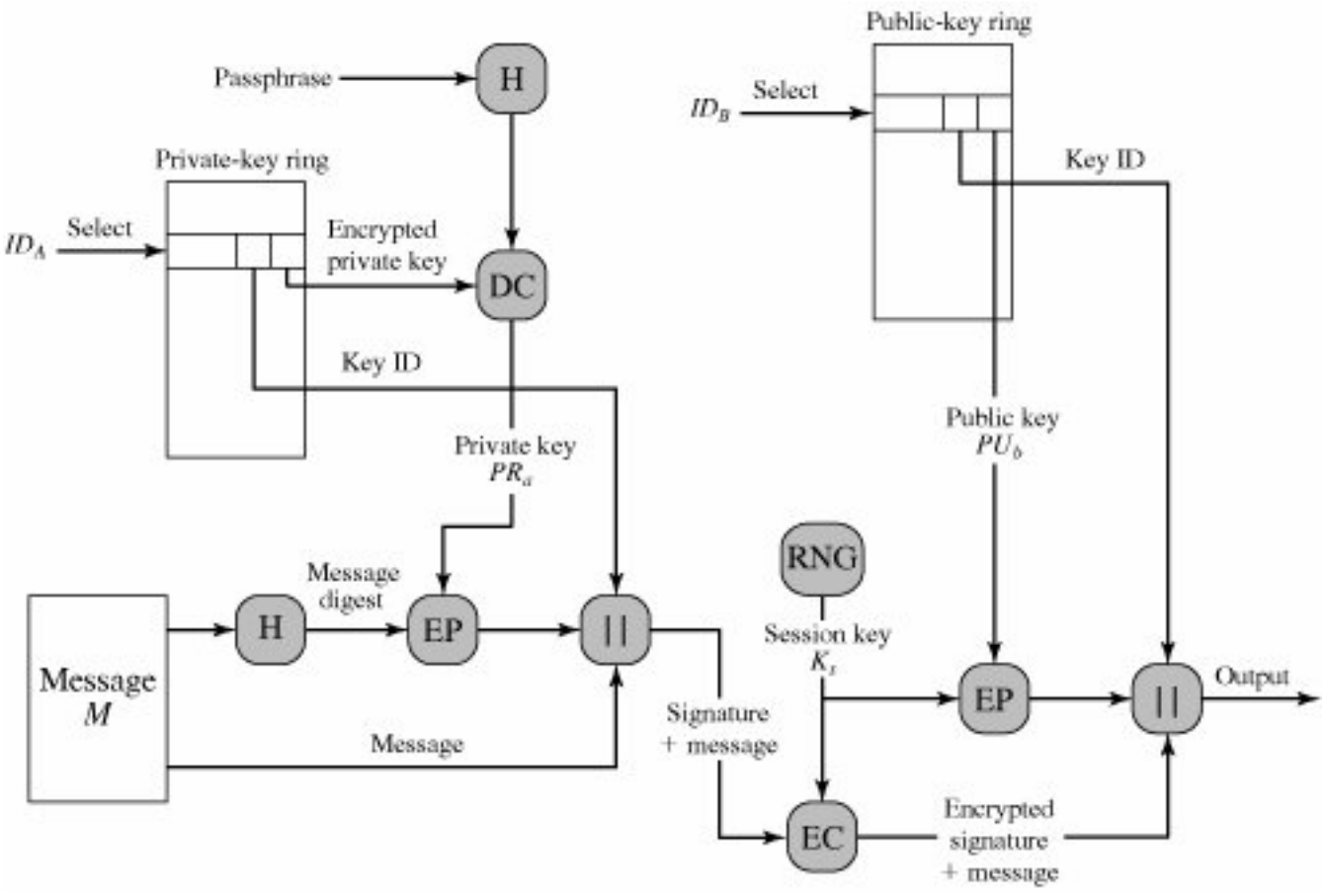
   c.

      The session key component of the message is constructed.

Figure 15.5. PGP Message Generation (from User A to User B; no compression or radix 64 conversion)

(This item is displayed on page 450 in the print version)

[View full size image]



The receiving PGP entity performs the following steps (Figure 15.6):

1. Decrypting the message

   a. PGP retrieves the receiver's private key from the private-key ring, using the Key ID field in the session key component of the message as an index.

---

   b. PGP prompts the user for the passphrase to recover the unencrypted private key.

   c. PGP then recovers the session key and decrypts the message.

2. Authenticating the message

   a. PGP retrieves the sender's public key from the public-key ring, using the Key ID field in the signature key component of the message as an index.

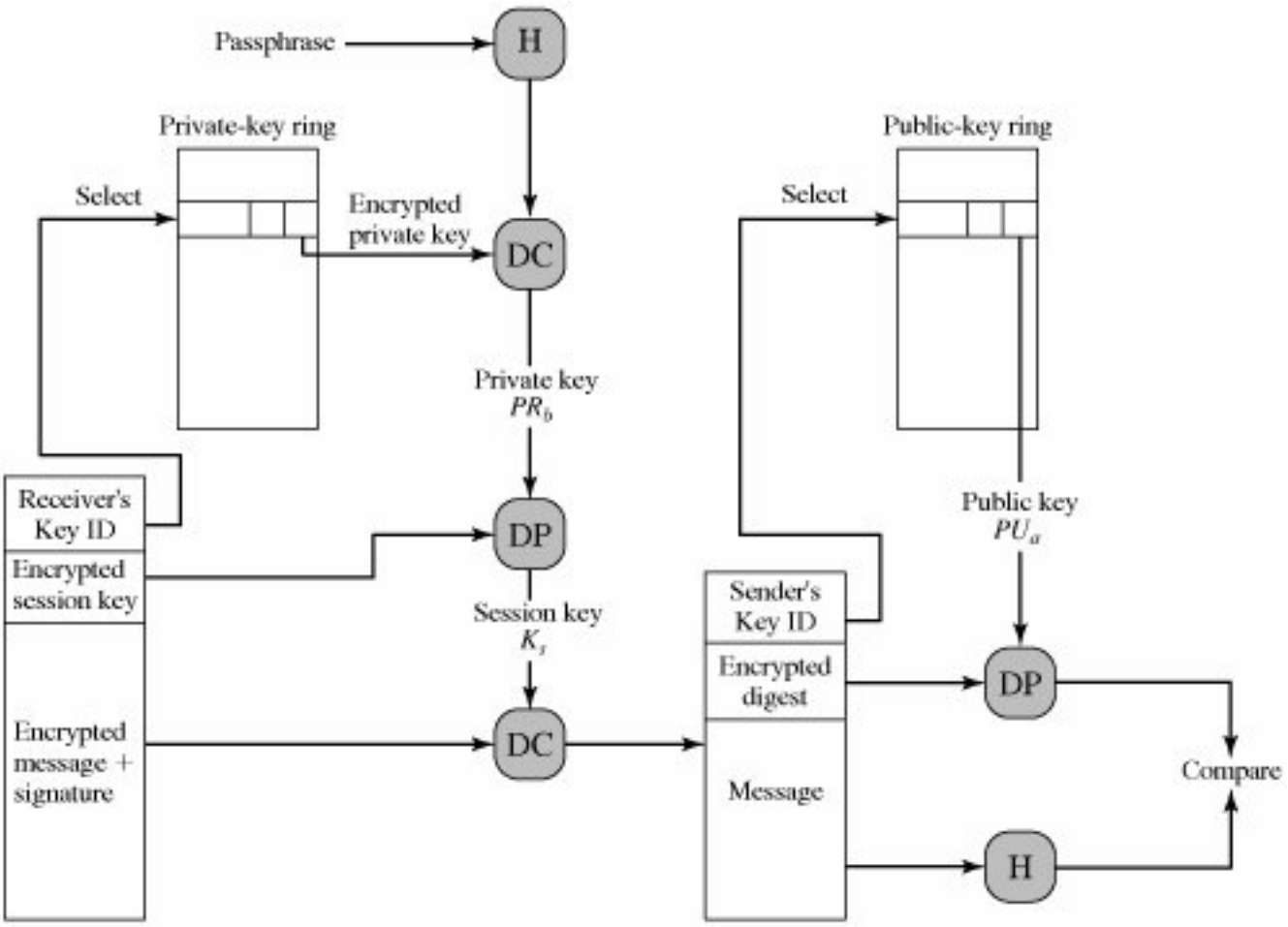
   b. PGP recovers the transmitted message digest.

   c. PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

Figure 15.6. PGP Message Reception (from User A to User B; no compression or radix 64 conversion)

(This item is displayed on page 451 in the print version)

[View full size image]



Public-Key Management

As can be seen from the discussion so far, PGP contains a clever, efficient, interlocking set of functions and formats to provide an effective confidentiality and authentication service. To complete the system, one final area needs to be addressed, that of public-key management. The PGP documentation captures the importance of this area:

> This whole business of protecting public keys from tampering is the single most difficult problem in practical public key applications. It is the "Achilles heel" of public key cryptography, and a lot of software complexity is tied up in solving this one problem.

PGP provides a structure for solving this problem, with several suggested options that may be used. Because PGP is intended for use in a variety of formal and informal environments, no rigid public-key management scheme is set up, such as we will see in our discussion of S/MIME later in this chapter.

Approaches to Public-Key Management

The essence of the problem is this: User A must build up a public-key ring containing the public keys of other users to interoperate with them using PGP. Suppose that A's key ring contains a public key attributed to B but that the key is, in fact, owned by C. This could happen if, for example, A got the key from a bulletin board system (BBS) that was used by B to post the public key but that has been compromised by C. The result is that two threats now exist. First, C can send messages to A and forge B's signature, so that A will accept the message as coming from B. Second, any encrypted message from A to B can be read by C.

A number of approaches are possible for minimizing the risk that a user's public-key ring contains false

public keys. Suppose that A wishes to obtain a reliable public key for B. The following are some approaches that could be used:

1. Physically get the key from B. B could store her public key (PU_b) on a floppy disk and hand it to A. A could then load the key into his system from the floppy disk. This is a very secure method but has obvious practical limitations.

2. Verify a key by telephone. If A can recognize B on the phone, A could call B and ask her to dictate the key, in radix-64 format, over the phone. As a more practical alternative, B could transmit her key in an e-mail message to A. A could have PGP generate a 160-bit SHA-1 digest of the key and display it in hexadecimal format; this is referred to as the "fingerprint" of the key. A could then call B and ask her to dictate the fingerprint over the phone. If the two fingerprints match, the key is verified.

---

3. Obtain B's public key from a mutual trusted individual D. For this purpose, the introducer, D, creates a signed certificate. The certificate includes B's public key, the time of creation of the key, and a validity period for the key. D generates an SHA-1 digest of this certificate, encrypts it with her private key, and attaches the signature to the certificate. Because only D could have created the signature, no one else can create a false public key and pretend that it is signed by D. The signed certificate could be sent directly to A by B or D, or could be posted on a bulletin board.

4. Obtain B's public key from a trusted certifying authority. Again, a public key certificate is created and signed by the authority. A could then access the authority, providing a user name and receiving a signed certificate.

For cases 3 and 4, A would already have to have a copy of the introducer's public key and trust that this key is valid. Ultimately, it is up to A to assign a level of trust to anyone who is to act as an introducer.

The Use of Trust

Although PGP does not include any specification for establishing certifying authorities or for establishing trust, it does provide a convenient means of using trust, associating trust with public keys, and exploiting trust information.

The basic structure is as follows. Each entry in the public-key ring is a public-key certificate, as described in the preceding subsection. Associated with each such entry is a **key legitimacy field** that indicates the extent to which PGP will trust that this is a valid public key for this user; the higher the level of trust, the stronger is the binding of this user ID to this key. This field is computed by PGP. Also associated with the entry are zero or more signatures that the key ring owner has collected that sign this certificate. In turn, each signature has associated with it a **signature trust field** that indicates the degree to which this PGP user trusts the signer to certify public keys. The key legitimacy field is derived from the collection of signature trust fields in the entry. Finally, each entry defines a public key associated with a particular owner, and an **owner trust field** is included that indicates the degree to

which this public key is trusted to sign other public-key certificates; this level of trust is assigned by the user. We can think of the signature trust fields as cached copies of the owner trust field from another entry.

The three fields mentioned in the previous paragraph are each contained in a structure referred to as a trust flag byte. The content of this trust flag for each of these three uses is shown in Table 15.2. Suppose that we are dealing with the public-key ring of user A. We can describe the operation of the trust processing as follows:

1. When A inserts a new public key on the public-key ring, PGP must assign a value to the trust flag that is associated with the owner of this public key. If the owner is A, and therefore this public key also appears in the private-key ring, then a value of *ultimate trust* is automatically assigned to the trust field. Otherwise, PGP asks A for his assessment of the trust to be assigned to the owner of this key, and A must enter the desired level. The user can specify that this owner is unknown, untrusted, marginally trusted, or completely trusted.

---

2. When the new public key is entered, one or more signatures may be attached to it. More signatures may be added later. When a signature is inserted into the entry, PGP searches the public-key ring to see if the author of this signature is among the known public-key owners. If so, the OWNERTRUST value for this owner is assigned to the SIGTRUST field for this signature. If not, an *unknown user* value is assigned.

3. The value of the key legitimacy field is calculated on the basis of the signature trust fields present in this entry. If at least one signature has a signature trust value of *ultimate*, then the key legitimacy value is set to complete. Otherwise, PGP computes a weighted sum of the trust values. A weight of 1/*X* is given to signatures that are always trusted and 1/*Y* to signatures that are usually trusted, where *X* and *Y* are user-configurable parameters. When the total of weights of the introducers of a key/UserID combination reaches 1, the binding is considered to be trustworthy, and the key legitimacy value is set to complete. Thus, in the absence of ultimate trust, at least *X* signatures that are always trusted or *Y* signatures that are usually trusted or some combination is needed.

Table 15.2. Contents of Trust Flag Byte

(This item is displayed on page 454 in the print version)

(a) Trust Assigned to Public-Key Owner (appears after key packet; user defined)	(b) Trust Assigned to Public Key/User ID Pair (appears after User ID packet; computed by PGP)	(c) Trust Assigned to Signature (appears after signature packet; cached copy of OWNERTRUST for this signator)

OWNERTRUST Field	KEYLEGIT Field	SIGTRUST Field
undefined trust	unknown or undefined trust	undefined trust
unknown user	key ownership not trusted	unknown user
usually not trusted to sign other keys	marginal trust in key ownership	usually not trusted to sign other keys
usually trusted to sign other keys	complete trust in key ownership	usually trusted to sign other keys
always trusted to sign other keys	WARNONLY bit	always trusted to sign other keys
this key is present in secret key ring (ultimate trust)	set if user wants only to be warned when key that is not fully validated is used for encryption	this key is present in secret key ring (ultimate trust)
BUCKSTOP bit		CONTIG bit
set if this key appears in secret key ring		set if signature leads up a contiguous trusted certification path back to the ultimately trusted key ring owner

Periodically, PGP processes the public-key ring to achieve consistency. In essence, this is a top-down process. For each OWNERTRUST field, PGP scans the ring for all signatures authored by that owner and updates the SIGTRUST field to equal the OWNERTRUST field. This process starts with keys for which there is ultimate trust. Then all KEYLEGIT fields are computed on the basis of the attached signatures.

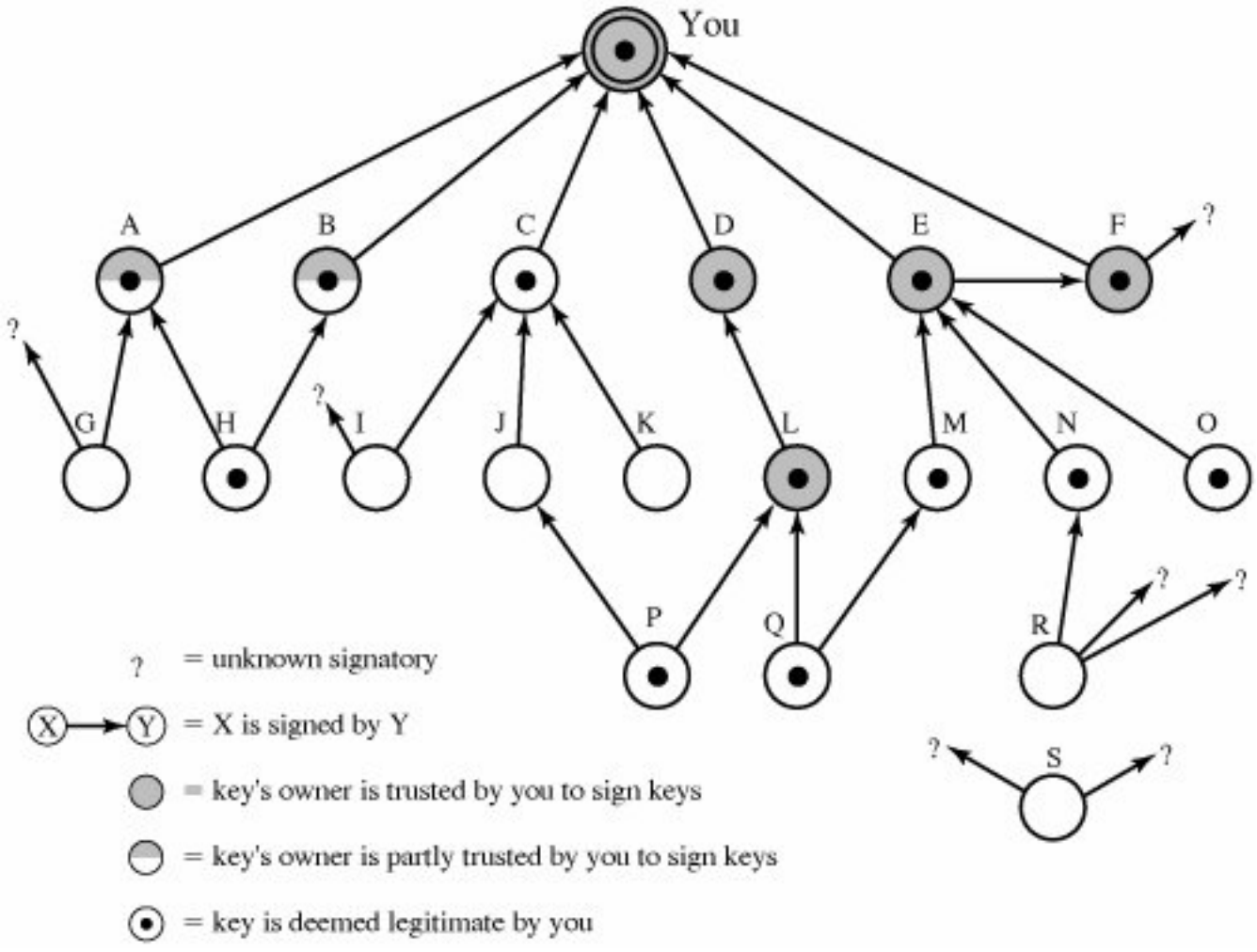
Figure 15.7 provides an example of the way in which signature trust and key legitimacy are related.[2] The figure shows the structure of a public-key ring. The user has acquired a number of public keys, some directly from their owners and some from a third party such as a key server.

[2] Figure provided to the author by Phil Zimmermann.

Figure 15.7. PGP Trust Model Example

(This item is displayed on page 456 in the print version)

[View full size image]



The node labeled "You" refers to the entry in the public-key ring corresponding to this user. This key is legitimate and the OWNERTRUST value is ultimate trust. Each other node in the key ring has an OWNERTRUST value of undefined unless some other value is assigned by the user. In this example, this user has specified that it always trusts the following users to sign other keys: D, E, F, L. This user partially trusts users A and B to sign other keys.

So the shading, or lack thereof, of the nodes in Figure 15.7 indicates the level of trust assigned by this user. The tree structure indicates which keys have been signed by which other users. If a key is signed by a user whose key is also in this key ring, the arrow joins the signed key to the signatory. If the key is signed by a user whose key is not present in this key ring, the arrow joins the signed key to a question mark, indicating that the signatory is unknown to this user.

Several points are illustrated in this Figure 15.7:

1. Note that all keys whose owners are fully or partially trusted by this user have been signed by this user, with the exception of node L. Such a user signature is not always necessary, as the presence of node L indicates, but in practice, most users are likely to sign the keys for most owners that they trust. So, for example, even though E's key is already signed by trusted introducer F, the user chose to sign E's key directly.

2.

We assume that two partially trusted signatures are sufficient to certify a key. Hence, the key for user H is deemed legitimate by PGP because it is signed by A and B, both of whom are partially trusted.

3.

A key may be determined to be legitimate because it is signed by one fully trusted or two partially trusted signatories, but its user may not be trusted to sign other keys. For example, N's key is legitimate because it is signed by E, whom this user trusts, but N is not trusted to sign other keys because this user has not assigned N that trust value. Therefore, although R's key is signed by N, PGP does not consider R's key legitimate. This situation makes perfect sense. If you wish to send a private message to some individual, it is not necessary that you trust that individual in any respect. It is only necessary that you are sure that you have the correct public key for that individual.

4.

Figure 15.7 also shows an example of a detached "orphan" node S, with two unknown signatures. Such a key may have been acquired from a key server. PGP cannot assume that this key is legitimate simply because it came from a reputable server. The user must declare the key legitimate by signing it or by telling PGP that it is willing to trust fully one of the key's signatories.

A final point: Earlier it was mentioned that multiple user IDs may be associated with a single public key on the public-key ring. This could be because a person has changed names or has been introduced via signature under multiple names, indicating different e-mail addresses for the same person, for example. So we can think of a public key as the root of a tree. A public key has a number of user IDs associating with it, with a number of signatures below each user ID. The binding of a particular user ID to a key depends on the signatures associated with that user ID and that key, whereas the level of trust in this key (for use in signing other keys) is a function of all the dependent signatures.

Revoking Public Keys

A user may wish to revoke his or her current public key either because compromise is suspected or simply to avoid the use of the same key for an extended period. Note that a compromise would require that an opponent somehow had obtained a copy of your unencrypted private key or that the opponent had obtained both the private key from your private-key ring and your passphrase.

The convention for revoking a public key is for the owner to issue a key revocation certificate, signed by the owner. This certificate has the same form as a normal signature certificate but includes an indicator that the purpose of this certificate is to revoke the use of this public key. Note that the corresponding private key must be used to sign a certificate that revokes a public key. The owner should then attempt to disseminate this certificate as widely and as quickly as possible to enable potential correspondents to update their public-key rings.

Note that an opponent who has compromised the private key of an owner can also issue such a certificate. However, this would deny the opponent as well as the legitimate owner the use of the public key, and therefore it seems a much less likely threat than the malicious use of a stolen private key.

15.2. S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet e-mail format standard, based on technology from RSA Data Security. Although both PGP and S/MIME are on an IETF standards track, it appears likely that S/MIME will emerge as the industry standard for commercial and organizational use, while PGP will remain the choice for personal e-mail security for many users. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851.

To understand S/MIME, we need first to have a general understanding of the underlying e-mail format that it uses, namely MIME. But to understand the significance of MIME, we need to go back to the traditional e-mail format standard, RFC 822, which is still in common use. Accordingly, this section first provides an introduction to these two earlier standards and then moves on to a discussion of S/MIME.

RFC 822

RFC 822 defines a format for text messages that are sent using electronic mail. It has been the standard for Internet-based text mail message and remains in common use. In the RFC 822 context, messages are viewed as having an envelope and contents. The envelope contains whatever information is needed to accomplish transmission and delivery. The contents compose the object to be delivered to the recipient. The RFC 822 standard applies only to the contents. However, the content standard includes a set of header fields that may be used by the mail system to create the envelope, and the standard is intended to facilitate the acquisition of such information by programs.

The overall structure of a message that conforms to RFC 822 is very simple. A message consists of some number of header lines (*the header*) followed by unrestricted text (*the body*). The header is separated from the body by a blank line. Put differently, a message is ASCII text, and all lines up to the first blank line are assumed to be header lines used by the user agent part of the mail system.

A header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines. The most frequently used keywords are *From*, *To*, *Subject*, and *Date*. Here is an example message:

```
Date: Tue, 16 Jan 1998 10:37:17 (EST)
From: "William Stallings" <ws@shore.net>
Subject: The Syntax in RFC 822
To: Smith@Other-host.com
Cc: Jones@Yet-Another-Host.com

Hello. This section begins the actual
message body, which is delimited from the
message heading by a blank line.
```

Another field that is commonly found in RFC 822 headers is *Message-ID*. This field contains a unique identifier associated with this message.

Multipurpose Internet Mail Extensions

MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol) or some other mail transfer protocol and RFC 822 for electronic mail. [RODR02] lists the following limitations of the SMTP/822 scheme:

1. SMTP cannot transmit executable files or other binary objects. A number of schemes are in use for converting binary files into a text form that can be used by SMTP mail systems, including the popular UNIX UUencode/UUdecode scheme. However, none of these is a standard or even a de facto standard.

2. SMTP cannot transmit text data that includes national language characters because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.

3. SMTP servers may reject mail message over a certain size.

---

4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.

5. SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included in X.400 messages.

6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include:

   - Deletion, addition, or reordering of carriage return and linefeed
   - Truncating or wrapping lines longer than 76 characters
   - Removal of trailing white space (tab and space characters)
   - Padding of lines in a message to the same length
   - Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations. The specification is provided in RFCs 2045 through 2049.

Overview

The MIME specification includes the following elements:

1. Five new message header fields are defined, which may be included in an RFC 822 header. These fields provide information about the body of the message.

2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.

3. Transfer encodings are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

In this subsection, we introduce the five message header fields. The next two subsections deal with content formats and transfer encodings.

The five header fields defined in MIME are as follows:

- **MIME-Version:** Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.
- **Content-Type:** Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to represent the data to the user or otherwise deal with the data in an appropriate manner.
- **Content-Transfer-Encoding:** Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.
- **Content-ID:** Used to identify MIME entities uniquely in multiple contexts.
- **Content-Description:** A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

Any or all of these fields may appear in a normal RFC 822 header. A compliant implementation must support the MIME-Version, Content-Type, and Content-Transfer-Encoding fields; the Content-ID and Content-Description fields are optional and may be ignored by the recipient implementation.

MIME Content Types

The bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment.

Table 15.3 lists the content types specified in RFC 2046. There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data.

Table 15.3. MIME Content Types

(This item is displayed on page 461 in the print version)

Type	Subtype	Description
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
	Enriched	Provides greater format flexibility.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the "best" version to the user.
	Digest	Similar to Mixed, but the default type/subtype of each part is message/ rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript.
	octet-stream	General binary data consisting of 8-bit bytes.

For the **text type** of body, no special software is required to get the full meaning of the text, aside from support of the indicated character set. The primary subtype is *plain text*, which is simply a string of ASCII characters or ISO 8859 characters. The *enriched* subtype allows greater formatting flexibility.

The **multipart type** indicates that the body contains multiple, independent parts. The Content-Type header field includes a parameter, called boundary, that defines the delimiter between body parts. This boundary should not appear in any parts of the message. Each boundary starts on a new line and consists of two hyphens followed by the boundary value. The final boundary, which indicates the end of the last part, also has a suffix of two hyphens. Within each part, there may be an optional ordinary MIME header.

Here is a simple example of a multipart message, containing two parts both consisting of simple text (taken from RFC 2046):

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To:  Ned Freed <ned@innosoft.com>
Subject: Sample message
MIME-Version: 1.0
Content-type: multipart/mixed; boundary="simple
boundary"
This is the preamble. It is to be ignored, though it
 is a handy place for mail composers to include an
explanatory note to non-MIME conformant readers.
simple boundary
This is implicitly typed plain ASCII text. It does NOT
end with a linebreak.
simple boundary
Content-type: text/plain; charset=us-ascii
This is explicitly typed plain ASCII text. It DOES end
with a linebreak.
simple boundary
This is the epilogue.  It is also to be ignored.
```

There are four subtypes of the multipart type, all of which have the same overall syntax. The **multipart/mixed subtype** is used when there are multiple independent body parts that need to be bundled in a particular order. For the **multipart/parallel subtype**, the order of the parts is not significant. If the recipient's system is appropriate, the multiple parts can be presented in parallel. For example, a picture or text part could be accompanied by a voice commentary that is played while the picture or text is displayed.

For the **multipart/alternative subtype**, the various parts are different representations of the same information. The following is an example:

```
From:  Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary=boundary42
--boundary42
Content-Type: text/plain; charset=us-ascii
   ... plain text version of message goes here....
--boundary42
Content-Type: text/enriched
   .... RFC 1896 text/enriched version of same message
goes here ...
boundary42
```

In this subtype, the body parts are ordered in terms of increasing preference. For this example, if the recipient system is capable of displaying the message in the text/enriched format, this is done; otherwise, the plain text format is used.

The **multipart/digest subtype** is used when each of the body parts is interpreted as an RFC 822 message with headers. This subtype enables the construction of a message whose parts are individual messages. For example, the moderator of a group might collect e-mail messages from participants, bundle these messages, and send them out in one encapsulating MIME message.

The **message type** provides a number of important capabilities in MIME. The **message/rfc822 subtype** indicates that the body is an entire message, including header and body. Despite the name of this subtype, the encapsulated message may be not only a simple RFC 822 message, but also any MIME message.

The **message/partial subtype** enables fragmentation of a large message into a number of parts, which must be reassembled at the destination. For this subtype, three parameters are specified in the Content-Type: Message/Partial field: an *id* common to all fragments of the same message, a *sequence number* unique to each fragment, and the *total* number of fragments.

The **message/external-body subtype** indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data. As with the other message types, the message/external-body subtype has an outer header and an encapsulated message with its own header. The only necessary field in the outer header is the Content-Type field, which identifies this as a message/external-body subtype. The inner header is the message header for the encapsulated message. The Content-Type field in the outer header must include an access-type parameter, which indicates the method of access, such as FTP (file transfer protocol).

The **application type** refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application.

MIME Transfer Encodings

The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

The MIME standard defines two methods of encoding data. The Content-Transfer-Encoding field can actually take on six values, as listed in Table 15.4. However, three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data. For SMTP transfer, it is safe to use the 7bit form. The 8bit and binary forms may be usable in other mail transport contexts. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used, for which a name is to be supplied. This could be a vendor-specific or application-specific scheme. The two actual encoding schemes defined are quoted-printable and base64. Two schemes are defined to provide a choice between a transfer technique that is essentially human readable and one that is safe for all types of data in a way that is reasonably compact.

Table 15.4. MIME Transfer Encodings

7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
binary	Not only may non-ASCII characters be present but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by humans.

base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

The **quoted-printable** transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters.

The **base64 transfer encoding**, also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail transport programs. It is also used in PGP and is described in Appendix 15B.

A Multipart Example

Figure 15.8, taken from RFC 2045, is the outline of a complex multipart message. The message has five parts to be displayed serially: two introductory plain text parts, an embedded multipart message, a richtext part, and a closing encapsulated text message in a non-ASCII character set. The embedded multipart message has two parts to be displayed in parallel, a picture and an audio fragment.

Figure 15.8. Example MIME Message Structure

(This item is displayed on page 464 in the print version)

```
MIME-Version: 1.0

From: Nathaniel Borenstein <nsb@bellcore.com>

To: Ned Freed <ned@innosoft.com>

Subject: A multipart example

Content-Type: multipart/mixed;

        boundary=unique-boundary-1

This is the preamble area of a multipart message. Mail readers that understand multipart format
should ignore this preamble. If you are reading this text, you might want to consider changing to a
mail reader that understands how to properly display multipart messages.

--unique-boundary-1

        ...Some text appears here...
```

[Note that the preceding blank line means no header fields were given and this is text, with charset US ASCII. It could have been done with explicit typing as in the next part.]

--unique-boundary-1

Content-type: text/plain; charset=US-ASCII

This could have been part of the previous part, but illustrates explicit versus implicit typing of body parts.

--unique-boundary-1

Content-Type: multipart/parallel; boundary=unique-boundary-2

--unique-boundary-2

Content-Type: audio/basic

Content-Transfer-Encoding: base64

... base64-encoded 8000 Hz single-channel mu-law-format audio data goes here....

--unique-boundary-2

Content-Type: image/jpeg

Content-Transfer-Encoding: base64

... base64-encoded image data goes here....

--unique-boundary-2--

--unique-boundary-1

Content-type: text/enriched

This is <bold><italic>richtext.</italic></bold> <smaller>as defined in RFC 1896</smaller>

Isn't it <bigger><bigger>cool?</bigger></bigger>

--unique-boundary-1

Content-Type: message/rfc822

From: (mailbox in US-ASCII)

To: (address in US-ASCII)

Subject: (subject in US-ASCII)

```
Content-Type: Text/plain; charset=ISO-8859-1

Content-Transfer-Encoding: Quoted-printable

        ... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```

Canonical Form

An important concept in MIME and S/MIME is that of canonical form. Canonical form is a format, appropriate to the content type, that is standardized for use between systems. This is in contrast to native form, which is a format that may be peculiar to a particular system. Table 15.5, from RFC 2049, should help clarify this matter.

Table 15.5. Native and Canonical Form

Native Form	The body to be transmitted is created in the system's native format. The native character set is used and, where appropriate, local end-of-line conventions are used as well. The body may be a UNIX-style text file, or a Sun raster image, or a VMS indexed file, or audio data in a system-dependent format stored only in memory, or anything else that corresponds to the local model for the representation of some form of information. Fundamentally, the data is created in the "native" form that corresponds to the type specified by the media type.
Canonical Form	The entire body, including "out-of-band" information such as record lengths and possibly file attribute information, is converted to a universal canonical form. The specific media type of the body as well as its associated attributes dictate the nature of the canonical form that is used. Conversion to the proper canonical form may involve character set conversion, transformation of audio data, compression, or various other operations specific to the various media types. If character set conversion is involved, however, care must be taken to understand the semanties of the media type, which may have strong implications for any character set conversion (e.g. with regard to syntactically meaningful characters in a text subtype other than "plain").

S/MIME Functionality

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability. We then look in more detail at this capability by examining message formats and message preparation.

Functions

S/MIME provides the following functions:

- **Enveloped data:** This consists of encrypted content of any type and encrypted-content encryption keys for one or more recipients.
- **Signed data:** A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.
- **Clear-signed data:** As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.
- **Signed and enveloped data:** Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

Cryptographic Algorithms

Table 15.6 summarizes the cryptographic algorithms used in S/MIME. S/MIME uses the following terminology, taken from RFC 2119 to specify the requirement level:

- **Must:** The definition is an absolute requirement of the specification. An implementation must include this feature or function to be in conformance with the specification.
- **Should:** There may exist valid reasons in particular circumstances to ignore this feature or function, but it is recommended that an implementation include the feature or function.

Table 15.6. Cryptographic Algorithms Used in S/MIME

Function	Requirement
Create a message digest to be used in forming a digital signature. Encrypt message digest to form digital signature.	MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility. Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of RSA signatures with key sizes 512 bits to 1024 bits.
Encrypt session key for transmission with message.	Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA encryption with key sizes 512 bits to 1024 bits.

Encrypt message for transmission with one-time session key.	Sending and receiving agents MUST support encryption with triple DES Sending agents SHOULD support encryption with AES. Sending agents SHOULD support encryption with RC2/40.
Create a message authentication code	Receiving agents MUST support HMAC with SHA-1. Receiving agents SHOULD support HMAC with SHA-1.

S/MIME incorporates three public-key algorithms. The Digital Signature Standard (DSS) described in Chapter 13 is the preferred algorithm for digital signature. S/MIME lists Diffie-Hellman as the preferred algorithm for encrypting session keys; in fact, S/MIME uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal (see Problem 10.6). As an alternative, RSA, described in Chapter 9, can be used for both signatures and session key encryption. These are the same algorithms used in PGP and provide a high level of security. For the hash function used to create the digital signature, the specification requires the 160-bit SHA-1 but recommends receiver support for the 128-bit MD5 for backward compatibility with older versions of S/MIME. As we discussed in Chapter 12, there is justifiable concern about the security of MD5, so SHA-1 is clearly the preferred alternative.

For message encryption, three-key triple DES (tripleDES) is recommended, but compliant implementations must support 40-bit RC2. The latter is a weak encryption algorithm but allows compliance with U.S. export controls.

The S/MIME specification includes a discussion of the procedure for deciding which content encryption algorithm to use. In essence, a sending agent has two decisions to make. First, the sending agent must determine if the receiving agent is capable of decrypting using a given encryption algorithm. Second, if the receiving agent is only capable of accepting weakly encrypted content, the sending agent must decide if it is acceptable to send using weak encryption. To support this decision process, a sending agent may announce its decrypting capabilities in order of preference any message that it sends out. A receiving agent may store that information for future use.

The following rules, in the following order, should be followed by a sending agent:

1. If the sending agent has a list of preferred decrypting capabilities from an intended recipient, it SHOULD choose the first (highest preference) capability on the list that it is capable of using.

2. If the sending agent has no such list of capabilities from an intended recipient but has received one or more messages from the recipient, then the outgoing message SHOULD use the same

encryption algorithm as was used on the last signed and encrypted message received from that intended recipient.

3.

If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is willing to risk that the recipient may not be able to decrypt the message, then the sending agent SHOULD use tripleDES.

4.

If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is not willing to risk that the recipient may not be able to decrypt the message, then the sending agent MUST use RC2/40.

If a message is to be sent to multiple recipients and a common encryption algorithm cannot be selected for all, then the sending agent will need to send two messages. However, in that case, it is important to note that the security of the message is made vulnerable by the transmission of one copy with lower security.

S/MIME Messages

S/MIME makes use of a number of new MIME content types, which are shown in Table 15.7. All of the new application types use the designation PKCS. This refers to a set of public-key cryptography specifications issued by RSA Laboratories and made available for the S/MIME effort.

Table 15.7. S/MIME Content Types

(This item is displayed on page 468 in the print version)

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	pkcs 7-mime	signedData	A signed S/MIME entity.
	pkcs 7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs 7-mime	degenerate signedData	An entity containing only public- key certificates.
	pkcs 7-mime	CompressedData	A compressed S/MIME entity
	pkcs 7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

We examine each of these in turn after first looking at the general procedures for S/MIME message preparation.

Securing a MIME Entity

S/MIME secures a MIME entity with a signature, encryption, or both. A MIME entity may be an entire message (except for the RFC 822 headers), or if the MIME content type is multipart, then a MIME entity is one or more of the subparts of the message. The MIME entity is prepared according to the normal rules for MIME message preparation. Then the MIME entity plus some security-related data, such as algorithm identifiers and certificates, are processed by S/MIME to produce what is known as a PKCS object. A PKCS object is then treated as message content and wrapped in MIME (provided with appropriate MIME headers). This process should become clear as we look at specific objects and provide examples.

In all cases, the message to be sent is converted to canonical form. In particular, for a given type and subtype, the appropriate canonical form is used for the message content. For a multipart message, the appropriate canonical form is used for each subpart.

The use of transfer encoding requires special attention. For most cases, the result of applying the security algorithm will be to produce an object that is partially or totally represented in arbitrary binary data. This will then be wrapped in an outer MIME message and transfer encoding can be applied at that point, typically base64. However, in the case of a multipart signed message, described in more detail later, the message content in one of the subparts is unchanged by the security process. Unless that content is 7bit, it should be transfer encoded using base64 or quoted-printable, so that there is no danger of altering the content to which the signature was applied.

We now look at each of the S/MIME content types.

EnvelopedData

An application/pkcs7-mime subtype is used for one of four categories of S/MIME processing, each with a unique smime-type parameter. In all cases, the resulting entity, referred to as an *object*, is represented in a form known as Basic Encoding Rules (BER), which is defined in ITU-T Recommendation X.209. The BER format consists of arbitrary octet strings and is therefore binary data. Such an object should be transfer encoded with base64 in the outer MIME message. We first look at envelopedData.

The steps for preparing an envelopedData MIME entity are as follows:

1. Generate a pseudorandom session key for a particular symmetric encryption algorithm (RC2/40 or tripleDES).

2. For each recipient, encrypt the session key with the recipient's public RSA key.

3. For each recipient, prepare a block known as RecipientInfo that contains an identifier of the recipient's public-key certificate,[3] an identifier of the algorithm used to encrypt the session key, and the encrypted session key.

[3] This is an X.509 certificate, discussed later in this section.

---

4. Encrypt the message content with the session key.

The RecipientInfo blocks followed by the encrypted content constitute the envelopedData. This information is then encoded into base64. A sample message (excluding the RFC 822 headers) is the following:

```
Content-Type: application/pkcs7-mime; smime-type=enveloped-
       data; name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
rfvbnj75.6tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```

To recover the encrypted message, the recipient first strips off the base64 encoding. Then the recipient's private key is used to recover the session key. Finally, the message content is decrypted with the session key.

SignedData

The signedData smime-type can actually be used with one or more signers. For clarity, we confine our description to the case of a single digital signature. The steps for preparing a signedData MIME entity are as follows:

1. Select a message digest algorithm (SHA or MD5).
2. Compute the message digest, or hash function, of the content to be signed.
3. Encrypt the message digest with the signer's private key.
4. Prepare a block known as SignerInfo that contains the signer's public-key certificate, an identifier of the message digest algorithm, an identifier of the algorithm used to encrypt the message digest, and the encrypted message digest.

The signedData entity consists of a series of blocks, including a message digest algorithm identifier, the message being signed, and SignerInfo. The signedData entity may also include a set of public-key certificates sufficient to constitute a chain from a recognized root or top-level certification authority to the signer. This information is then encoded into base64. A sample message (excluding the RFC 822 headers) is the following:

```
Content-Type: application/pkcs7-mime; smime-type=signed-data;
       name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj7
77n8HHGT9HG4VQpfyF467GhIGfHfYT6rfvbnj756tbBghyHhHUujhJhjH
HUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H7n8HHGghyHh
6YT64V0GhIGfHfQbnj75
```

To recover the signed message and verify the signature, the recipient first strips off the base64 encoding. Then the signer's public key is used to decrypt the message digest. The recipient independently computes the message digest and compares it to the decrypted message digest to verify the signature.

Clear Signing

Clear signing is achieved using the multipart content type with a signed subtype. As was mentioned, this signing process does not involve transforming the message to be signed, so that the message is sent "in the clear." Thus, recipients with MIME capability but not S/MIME capability are able to read the incoming message.

A multipart/signed message has two parts. The first part can be any MIME type but must be prepared so that it will not be altered during transfer from source to destination. This means that if the first part is not 7bit, then it needs to be encoded using base64 or quoted-printable. Then this part is processed in the same manner as signedData, but in this case an object with signedData format is created that has an empty message content field. This object is a detached signature. It is then transfer encoded using base64 to become the second part of the multipart/signed message. This second part has a MIME content type of application and a subtype of pkcs7-signature. Here is a sample message:

```
Content-Type: multipart/signed;
   protocol="application/pkcs7-signature";
   micalg=sha1; boundary=boundary42
boundary42
Content-Type: text/plain
This is a clear-signed message.
boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s
ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756
boundary42
```

The protocol parameter indicates that this is a two-part clear-signed entity. The micalg parameter indicates the type of message digest used. The receiver can verify the signature by taking the message digest of the first part and comparing this to the message digest recovered from the signature in the second part.

Registration Request

Typically, an application or user will apply to a certification authority for a public-key certificate. The application/pkcs10 S/MIME entity is used to transfer a certification request. The certification request includes certificationRequestInfo block, followed by an identifier of the public-key encryption algorithm, followed by the signature of the certificationRequestInfo block, made using the sender's private key. The certificationRequestInfo block includes a name of the certificate subject (the entity whose public key is to be certified) and a bit-string representation of the user's public key.

Certificates-Only Message

A message containing only certificates or a certificate revocation list (CRL) can be sent in response to a registration request. The message is an application/pkcs7-mime type/subtype with an smime-type parameter of degenerate. The steps involved are the same as those for creating a signedData message, except that there is no message content and the signerInfo field is empty.

S/MIME Certificate Processing

S/MIME uses public-key certificates that conform to version 3 of X.509 (see Chapter 14). The key-management scheme used by S/MIME is in some ways a hybrid between a strict X.509 certification hierarchy and PGP's web of trust. As with the PGP model, S/MIME managers and/or users must configure each client with a list of trusted keys and with certificate revocation lists. That is, the responsibility is local for maintaining the certificates needed to verify incoming signatures and to encrypt outgoing messages. On the other hand, the certificates are signed by certification authorities.

User Agent Role

An S/MIME user has several key-management functions to perform:

- **Key generation:** The user of some related administrative utility (e.g., one associated with LAN management) MUST be capable of generating separate Diffie-Hellman and DSS key pairs and SHOULD be capable of generating RSA key pairs. Each key pair MUST be generated from a good source of nondeterministic random input and be protected in a secure fashion. A user agent SHOULD generate RSA key pairs with a length in the range of 768 to 1024 bits and MUST NOT generate a length of less than 512 bits.
- **Registration:** A user's public key must be registered with a certification authority in order to receive an X.509 public-key certificate.
- **Certificate storage and retrieval:** A user requires access to a local list of certificates in order to verify incoming signatures and to encrypt outgoing messages. Such a list could be maintained by the user or by some local administrative entity on behalf of a number of users.

VeriSign Certificates

There are several companies that provide certification authority (CA) services. For example, Nortel has designed an enterprise CA solution and can provide S/MIME support within an organization. There are a number of Internet-based CAs, including VeriSign, GTE, and the U.S. Postal Service. Of these, the most widely used is the VeriSign CA service, a brief description of which we now provide.

VeriSign provides a CA service that is intended to be compatible with S/MIME and a variety of other applications. VeriSign issues X.509 certificates with the product name VeriSign Digital ID. As of early 1998, over 35,000 commercial Web sites were using VeriSign Server Digital IDs, and over a million consumer Digital IDs had been issued to users of Netscape and Microsoft browsers.

The information contained in a Digital ID depends on the type of Digital ID and its use. At a minimum, each Digital ID contains

- Owner's public key

- Owner's name or alias
- Expiration date of the Digital ID
- Serial number of the Digital ID
- Name of the certification authority that issued the Digital ID
- Digital signature of the certification authority that issued the Digital ID

Digital IDs can also contain other user-supplied information, including

- Address
- E-mail address
- Basic registration information (country, zip code, age, and gender)

VeriSign provides three levels, or classes, of security for public-key certificates, as summarized in Table 15.8. A user requests a certificate online at VeriSign's Web site or other participating Web sites. Class 1 and Class 2 requests are processed on line, and in most cases take only a few seconds to approve. Briefly, the following procedures are used:

- For Class 1 Digital IDs, VeriSign confirms the user's e-mail address by sending a PIN and Digital ID pick-up information to the e-mail address provided in the application.
- For Class 2 Digital IDs, VeriSign verifies the information in the application through an automated comparison with a consumer database in addition to performing all of the checking associated with a Class 1 Digital ID. Finally, confirmation is sent to the specified postal address alerting the user that a Digital ID has been issued in his or her name.
- For Class 3 Digital IDs, VeriSign requires a higher level of identity assurance. An individual must prove his or her identity by providing notarized credentials or applying in person.

Table 15.8. VeriSign Public-Key Certificate Classes

(This item is displayed on page 473 in the print version)

	Summary of Confirmation of Identity	IA Private Key Protection	Certificate Applicant and Subscriber Private Key Protection	Applications implemented or contemplated by Users
Class 1	Automated unambiguous name and e-mail address search	PCA: trustworthy hardware; CA: trust-worthy software or trustworthy hardware	Encryption software (PIN protected) recommended but not required	Web-browsing and certain e-mail usage
Class 2	Same as Class 1, plus automated enrollment information check plus automated address check	PCA and CA: trustworthy hardware	Encryption software (PIN protected) required	Individual and intra and inter-company E-mail, online subscriptions, password replacement, and software validation

Class 3	Same as Class 1, plus personal presence and ID documents plus Class 2 automated ID check for individuals; business records (or filings) for organizations	PCA and CA: trustworthy hardware	Encryption software (PIN protected) required; hardware token recommended but not required	E-banking, corp, database access, personal banking, membership-based online services, content integrity services, e-commerce server, software validation; authentication of LRAAs; and strong encryption for certain servers

IA Issuing Authority

CA Certification Authority

PCA VeriSign public primary certification authority

PIN Personal Identification Number

LRAA Local Registration Authority Administrator

Enhanced Security Services

As of this writing, three enhanced security services have been proposed in an Internet draft. The details of these may change, and additional services may be added. The three services are as follows:

- **Signed receipts:** A signed receipt may be requested in a SignedData object. Returning a signed receipt provides proof of delivery to the originator of a message and allows the originator to demonstrate to a third party that the recipient received the message. In essence, the recipient signs the entire original message plus original (sender's) signature and appends the new signature to form a new S/MIME message.
- **Security labels:** A security label may be included in the authenticated attributes of a SignedData object. A security label is a set of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation. The labels may be used for access control, by indicating which users are permitted access to an object. Other uses include priority (secret, confidential, restricted, and so on) or role based, describing which kind of people can see the information (e.g., patient's health-care team, medical billing agents, etc.).

- **Secure mailing lists:** When a user sends a message to multiple recipients, a certain amount of per-recipient processing is required, including the use of each recipient's public key. The user can be relieved of this work by employing the services of an S/MIME Mail List Agent (MLA). An MLA can take a single incoming message, perform the recipient-specific encryption for each recipient, and forward the message. The originator of a message need only send the message to the MLA, with encryption performed using the MLA's public key.

Recommended Web Sites

- **PGP Home Page:** PGP Web site by PGP Corp., the leading PGP commercial vendor.
- **International PGP Home Page:** Designed to promote worldwide use of PGP. Contains documents and links of interest.
- **MIT Distribution Site for PGP:** Leading distributor of freeware PGP. Contains FAQ, other information, and links to other PGP sites.
- **PGP Charter:** Latest RFCs and Internet drafts for Open Specification PGP.
- **S/MIME Charter:** Latest RFCs and Internet drafts for S/MIME.

PREV NEXT

15.3. Key Terms, Review Questions, and Problems

Key Terms

detached signature

electronic mail

Multipurpose Internet Mail Extensions (MIME)

Pretty Good Privacy (PGP)

radix 64

session key

S/MIME

trust

ZIP

Review Questions

15.1 What are the five principal services provided by PGP?

15.2 What is the utility of a detached signature?

15.3 Why does PGP generate a signature before applying compression?

15.4 What is R64 conversion?

15.5 Why is R64 conversion useful for an e-mail application?

15.6 Why is the segmentation and reassembly function in PGP needed?

15.7 How does PGP use the concept of trust?

15.8 What is RFC 822?

15.9 What is MIME?

15.10 What is S/MIME?

Problems

15.1 PGP makes use of the cipher feedback (CFB) mode of CAST-128, whereas most symmetric encryption applications (other than key encryption) use the cipher block chaining (CBC) mode. We have

$$\text{CBC: } C_i = \text{E}(K, [C_{i-1} \oplus P_i]); \; P_i = C_{i-1} \oplus D(K, C_i)$$

$$\textit{CFB: } C_i = P_i \oplus E(K, C_{i-1}); \; P_i = C_i \oplus E(K, C_{i-1})$$

These two appear to provide equal security. Suggest a reason why PGP uses the CFB mode.

15.2 In the PGP scheme, what is the expected number of session keys generated before a previously created key is produced?

15.3 In PGP, what is the probability that a user with N public keys will have at least one duplicate key ID?

15.4 The first 16 bits of the message digest in a PGP signature are translated in the clear.

a. To what extent does this compromise the security of the hash algorithm?

b. To what extent does it in fact perform its intended function, namely, to help determine if the correct RSA key was used to decrypt the digest?

15.5 In Figure 15.4, each entry in the public-key ring contains an owner trust field that indicates the degree of trust associated with this public-key owner. Why is that not enough? That is, if this owner is trusted and this is supposed to be the owner's public key, why is not that trust enough to permit PGP to use this public key?

15.6 Consider radix-64 conversion as a form of encryption. In this case, there is no key. But suppose that an opponent knew only that some form of substitution algorithm was being used to encrypt English text and did not guess it was R64. How effective would this algorithm be against cryptanalysis?

15.7 Phil Zimmermann chose IDEA, three-key triple DES, and CAST-128 as symmetric encryption algorithms for PGP. Give reasons why each of the following symmetric encryption algorithms described in this book is suitable or unsuitable for PGP: DES, two-key triple DES, and AES.

PREV NEXT

Appendix 15A Data Compression Using Zip

PGP makes use of a compression package called ZIP, written by Jean-lup Gailly, Mark Adler, and Richard Wales. ZIP is a freeware package written in C that runs as a utility on UNIX and some other systems. ZIP is functionally equivalent to PKZIP, a widely available shareware package for Windows systems developed by PKWARE, Inc. The zip algorithm is perhaps the most commonly used cross-platform compression technique; freeware and shareware versions are available for Macintosh and other systems as well as Windows and UNIX systems.

Zip and similar algorithms stem from research by Jacob Ziv and Abraham Lempel. In 1977, they described a technique based on a sliding window buffer that holds the most recently processed text [ZIV77]. This algorithm is generally referred to as LZ77. A version of this algorithm is used in the zip compression scheme (PKZIP, gzip, zipit, etc.).

LZ77 and its variants exploit the fact that words and phrases within a text stream (image patterns in the case of GIF) are likely to be repeated. When a repetition occurs, the repeated sequence can be replaced by a short code. The compression program scans for such repetitions and develops codes on the fly to replace the repeated sequence. Over time, codes are reused to capture new sequences. The algorithm must be defined in such a way that the decompression program is able to deduce the current mapping between codes and sequences of source data.

Before looking at the details of LZ77, let us look at a simple example.[4] Consider the nonsense phrase

[4] Based on an example in [WEIS93].

`the brown fox jumped over the brown foxy jumping frog`

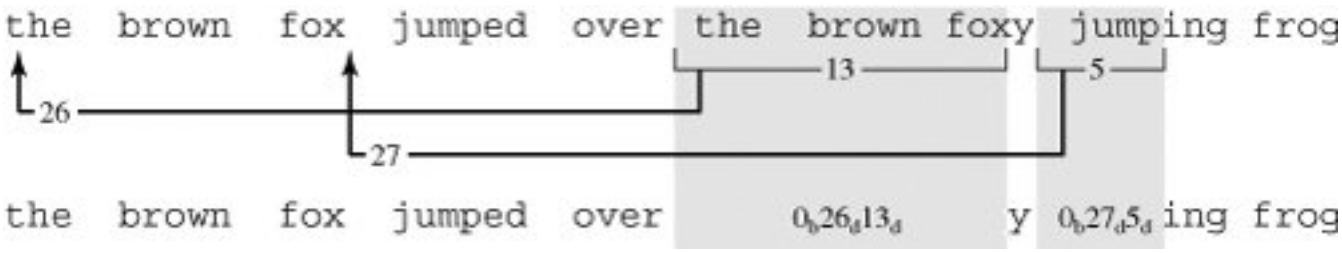
which is 53 octets = 424 bits long. The algorithm processes this text from left to right. Initially, each character is mapped into a 9-bit pattern consisting of a binary 1 followed by the 8-bit ASCII representation of the character. As the processing proceeds, the algorithm looks for repeated sequences. When a repetition is encountered, the algorithm continues scanning until the repetition ends. In other words, each time a repetition occurs, the algorithm includes as many characters as possible. The first such sequence encountered is **`the brown fox`**. This sequence is replaced by a pointer to the prior sequence and the length of the sequence. In this case the prior sequence of **`the brown fox`** occurs 26 character positions before and the length of the sequence is 13 characters. For this example, assume two options for encoding; an 8-bit pointer and a 4-bit length, or a 12-bit pointer and a 6-bit length; a 2-bit header indicates which option is chosen, with 00 indicating the first option and 01 the second option. Thus, the second occurrence of **`the brown fox`** is encoded as $<00_b><26_d><13_d>$, or 00 00011010 1101.

The remaining parts of the compressed message are the letter **`y;`** the sequence $<00_b><27_d><5_d>$, which replaces the sequence consisting of the space character followed by **`jump;`** and the character sequence **`ing frog`**.

Figure 15.9 illustrates the compression mapping. The compressed message consists of 35 9-bit characters and two codes, for a total of 35 x 9 + 2 x 14 = 343 bits. This compares with 424 bits in the uncompressed message for a compression ratio of 1.24.

Figure 15.9. Example of LZ77 Scheme

[View full size image]



Compression Algorithm

The compression algorithm for LZ77 and its variants makes use of two buffers. A **sliding history buffer** contains the last *N* characters of source that have been processed, and a **look-ahead buffer** contains the next *L* characters to be processed (Figure 15.10a). The algorithm attempts to match two or more characters from the beginning of the look-ahead buffer to a string in the sliding history buffer. If no match is found, the first character in the look-ahead buffer is output as a 9-bit character and is also shifted into the sliding window, with the oldest character in the sliding window shifted out. If a match is found, the algorithm continues to scan for the longest match. Then the matched string is output as a triplet (indicator, pointer, length). For a *K*-character string, the *K* oldest characters in the sliding window are shifted out, and the *K* characters of the encoded string are shifted into the window.

Figure 15.10. LZ77 Scheme

[View full size image]

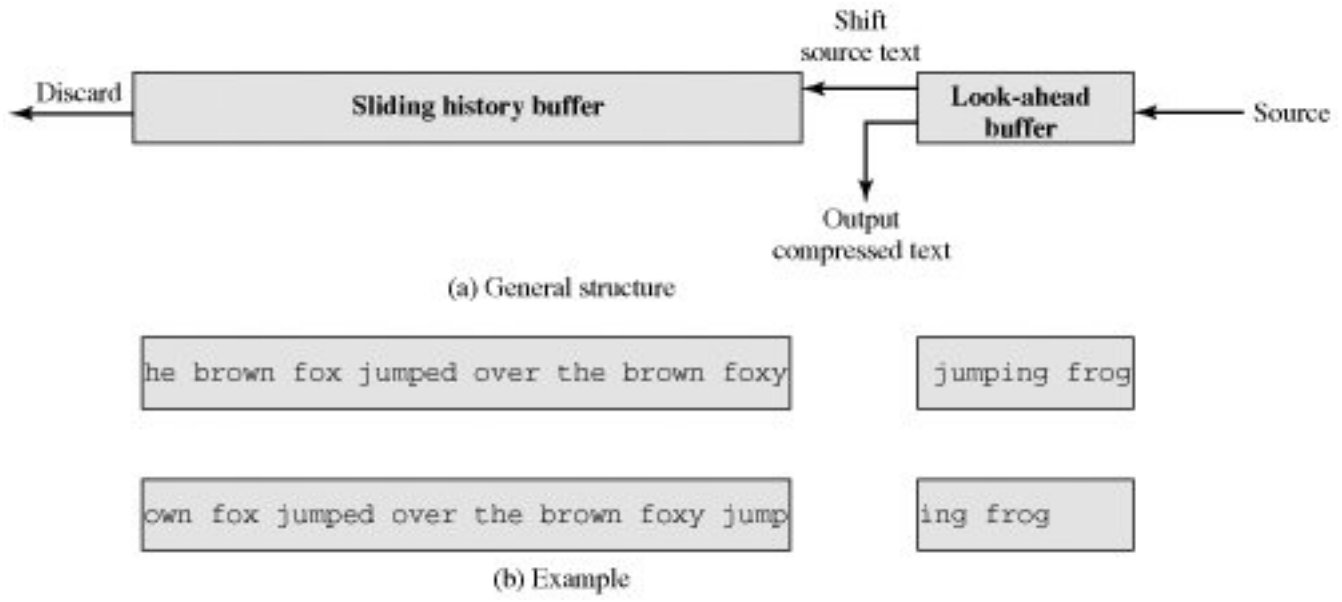


Figure 15.10b shows the operation of this scheme on our example sequence. The illustration assumes a

39-character sliding window and a 13-character look-ahead buffer. In the upper part of the example, the first 40 characters have been processed and the uncompressed version of the most recent 39 of these characters is in the sliding window. The remaining source is in the look-ahead window. The compression algorithm determines the next match, shifts 5 characters from the look-ahead buffer into the sliding window, and outputs the code for this string. The state of the buffer after these operations is shown in the lower part of the example.

While LZ77 is effective and does adapt to the nature of the current input, it has some drawbacks. The algorithm uses a finite window to look for matches in previous text. For a very long block of text, compared to the size of the window, many potential matches are eliminated. The window size can be increased, but this imposes two penalties: (1) The processing time of the algorithm increases because it must perform a string comparison against the look-ahead buffer for every position in the sliding window, and (2) the <pointer> field must be larger to accommodate the longer jumps.

Decompression Algorithm

Decompression of LZ77-compressed text is simple. The decompression algorithm must save the last *N* characters of decompressed output. When an encoded string is encountered, the decompression algorithm uses the and fields to replace the code with the actual text string.

PREV NEXT

Appendix 15B Radix-64 Conversion

Both PGP and S/MIME make use of an encoding technique referred to as radix-64 conversion. This technique maps arbitrary binary input into printable character output. The form of encoding has the following relevant characteristics:

1. The range of the function is a character set that is universally representable at all sites, not a specific binary encoding of that character set. Thus, the characters themselves can be encoded into whatever form is needed by a specific system. For example, the character "E" is represented in an ASCII-based system as hexadecimal 45 and in an EBCDIC-based system as hexadecimal C5.

2. The character set consists of 65 printable characters, one of which is used for padding. With 2^6 = 64 available characters, each character can be used to represent 6 bits of input.

3. No control characters are included in the set. Thus, a message encoded in radix 64 can traverse mail-handling systems that scan the data stream for control characters.

4. The hyphen character ("-")is not used. This character has significance in the RFC 822 format and should therefore be avoided.

Table 15.9 shows the mapping of 6-bit input values to characters. The character set consists of the alphanumeric characters plus "+" and "/". The "=" character is used as the padding character.

Table 15.9. Radix-64 Encoding

6-Bit	Character Encoding
0	A
1	B
2	C
3	D
4	E

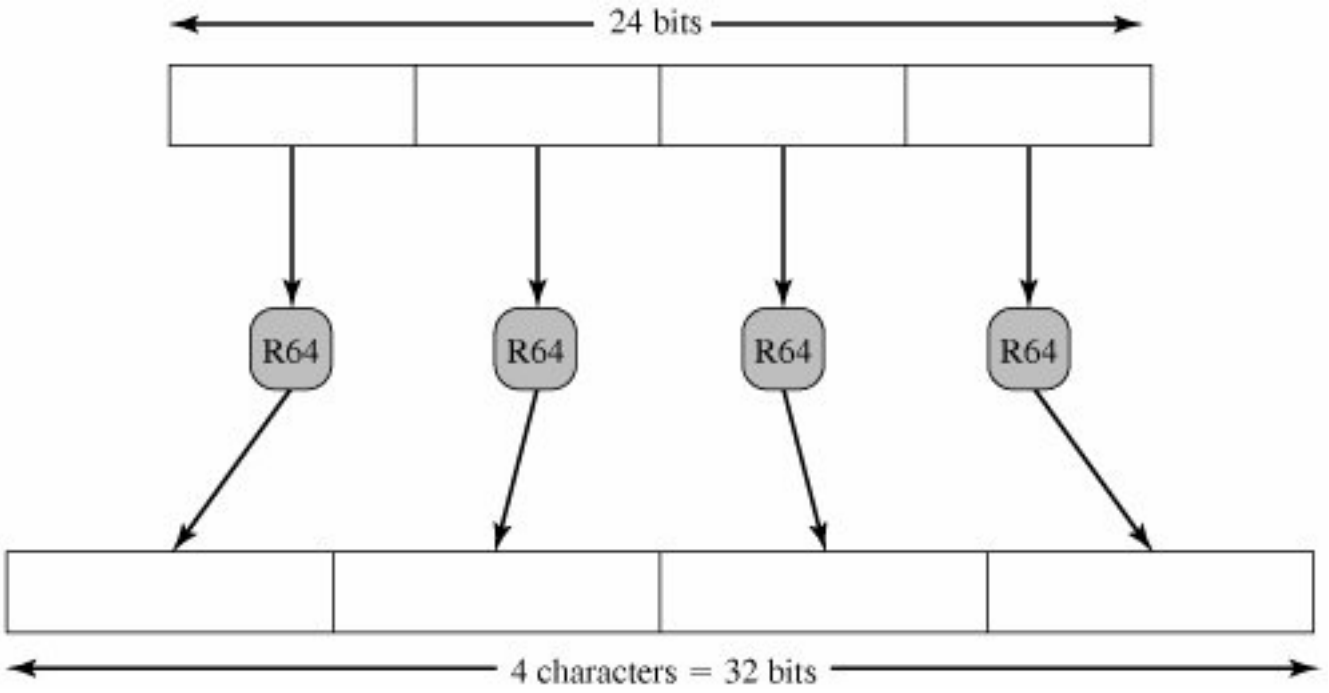
5	F
6	G
7	H
8	I
9	J
10	K
11	L
12	M
13	N
14	O
15	P
16	Q
17	R
18	S
19	T
20	U
21	V
22	W
23	X
24	Y
25	Z
26	a
27	b
28	c
29	d
30	e
31	f
32	g
33	h
34	i
35	j
36	k
37	l
38	m

39	n
40	o
41	p
42	q
43	r
44	s
45	t
46	u
47	v
48	w
49	x
50	y
51	z
52	0
53	1
54	2
55	3
56	4
57	5
58	6
59	7
60	8
61	9
62	+
63	/
(pad)	=

Figure 15.11 illustrates the simple mapping scheme. Binary input is processed in blocks of 3 octets, or 24 bits. Each set of 6 bits in the 24-bit block is mapped into a character. In the figure, the characters are shown encoded as 8-bit quantities. In this typical case, each 24-bit input is expanded to 32 bits of output.

Figure 15.11. Printable Encoding of Binary Data into Radix-64 Format

[View full size image]



For example, consider the 24-bit raw text sequence 00100011 01011100 10010001, which can be expressed in hexadecimal as 235C91. We arrange this input in blocks of 6 bits:

001000 110101 110010 010001

The extracted 6-bit decimal values are 8, 53, 50, 17. Looking these up in Table 15.9 yields the radix-64 encoding as the following characters: I1yR. If these characters are stored in 8-bit ASCII format with parity bit set to zero, we have

01001001 00110001 01111001 01010010

In hexadecimal, this is 49317952. To summarize,

Input Data	
Binary representation	00100011 01011100 10010001
Hexadecimal representation	235C91
Radix-64 Encoding of Input Data	
Character representation	I1yR
ASCII code (8 bit, zero parity)	01001001 00110001 01111001 01010010
Hexadecimal representation	49317952

PREV NEXT

Appendix 15C PGP Random Number Generation

PGP uses a complex and powerful scheme for generating random numbers and pseudorandom numbers, for a variety of purposes. PGP generates random numbers from the content and timing of user keystrokes, and pseudorandom numbers using an algorithm based on the one in ANSI X9.17. PGP uses these numbers for the following purposes:

- True random numbers:

  used to generate RSA key pairs

  provide the initial seed for the pseudorandom number generator

  provide additional input during pseudorandom number generation

- Pseudorandom numbers:

  used to generate session keys

  used to generate initialization vectors (IVs) for use with the session key in CFB mode encryption

True Random Numbers

PGP maintains a 256-byte buffer of random bits. Each time PGP expects a keystroke, it records the time, in 32-bit format, at which it starts waiting. When it receives the keystroke, it records the time the key was pressed and the 8-bit value of the keystroke. The time and keystroke information are used to generate a key, which is, in turn, used to encrypt the current value of the random-bit buffer.

Pseudorandom Numbers

Pseudorandom number generation makes use of a 24-octet seed and produces a 16-octet session key, an 8-octet initialization vector, and a new seed to be used for the next pseudorandom number generation. The algorithm is based on the X9.17 algorithm described in Chapter 7 (see Figure 7.14) but uses CAST-128 instead of triple DES for encryption. The algorithm uses the following data structures:

1.

   Input

   randseed.bin (24 octets): If this file is empty, it is filled with 24 true random octets.

message: The session key and IV that will be used to encrypt a message are themselves a function of that message. This further contributes to the randomness of the key and IV, and if an opponent already knows the plaintext content of the message, there is no apparent need for capturing the one-time session key.

2.

Output

K (24 octets): The first 16 octets, K[0..15], contain a session key, and the last eight octets, K[16..23], contain an IV.

randseed.bin (24 octets): A new seed value is placed in this file.

3.

Internal data structures

dtbuf (8 octets): The first 4 octets, dtbuf[0..3], are initialized with the current date/time value. This buffer is equivalent to the DT variable in the X12.17 algorithm.

rkey (16 octets): CAST-128 encryption key used at all stages of the algorithm.

rseed (8 octets): Equivalent to the X12.17 V*i* variable.

---

rbuf (8 octets): A pseudorandom number generated by the algorithm. This buffer is equivalent to the X12.17 R*i* variable.

K' (24 octets): Temporary buffer for the new value of randseed.bin.

The algorithm consists of nine steps, G1 through G9. The first and last steps are obfuscation steps, intended to reduce the value of a captured randseed.bin file to an opponent. The remaining steps are essentially equivalent to three iterations of the X12.17 algorithm and are illustrated in Figure 15.12 (compare Figure 7.14). To summarize,

G1. [Prewash previous seed]
a. Copy randseed.bin to K[0..23].
b. Take the hash of the message (this has already been generated if the message is being signed; otherwise the first 4K octets of the message are used). Use the result as a key, use a null IV, and encrypt K in CFB mode; store result back in K.
G2. [Set initial seed]
a. Set dtbuf[0..3] to the 32-bit local time. Set dtbuf[4..7] to all zeros. Copy rkey ⟵ K[0..15]. Copy rseed ⟵ K[16..23].
b. Encrypt the 64-bit dtbuf using the 128-bit rkey in ECB mode; store the result back in dtbuf.

G3. [Prepare to generate random octets] Set rcount ⟵ 0 and k ⟵ 23. The loop of steps G4-G7 will be executed 24 times (k = 23...0), once for each random octet produced and placed in K. The variable rcount is the number of unused random octets in rbuf. It will count down from 8 to 0 three times to generate the 24 octets.

G4. [Bytes available?] If rcount = 0 goto G5 else goto G7. Steps G5 and G6 perform one instance of the X12.17 algorithm to generate a new batch of eight random octets.

G5. [Generate new random octets]

a. rseed ⟵ rseed ⊕ dtbuf

b. rbuf ⟵ E_{rkey}[rseed] in ECB mode

G6. [Generate next seed]

a.rseed ⟵ rbuf ⊕ dtbuf

b. rseed ⟵ E_{rkey}[rseed] in ECB mode

c. Set rcount ⟵ 8

G7. [Transfer one byte at a time from rbuf to K]

a. Set rcount ⟵ rcount 1

b. Generate a true random byte b, and set K[k] ⟵ rbuf[rcount] ⊕ b

G8. [Done?] If k = 0 goto G9 else set k ⟵ k 1 and goto G4

G9. [Postwash seed and return result]

a. Generate 24 more bytes by the method of steps G4-G7, except do not XOR in a random byte in G7. Place the result in buffer K'

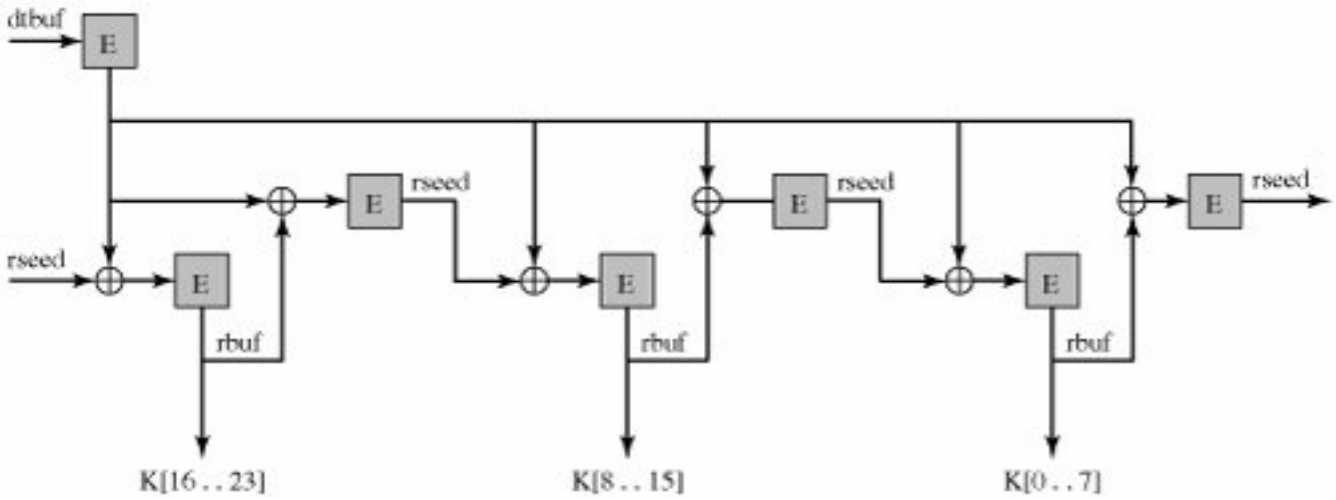
b. Encrypt K' with key K[0..15] and IV K[16..23] in CFB mode; store result in randseed.bin

c. Return K

Figure 15.12. PGP Session Key and IV Generation (steps G2 through G8)

(This item is displayed on page 481 in the print version)

[View full size image]



It should not be possible to determine the session key from the 24 new octets generated in step G9.a. However, to make sure that the stored randseed.bin file provides no information about the most recent session key, the 24 new octets are encrypted and the result is stored as the new seed.

This elaborate algorithm should provide cryptographically strong pseudorandom numbers.

PREV NEXT

Chapter 16. IP Security

If a secret piece of news is divulged by a spy before the time is ripe, he must be put to death, together with the man to whom the secret was told.

The Art of War, Sun Tzu

Key Points

- IP security (IPSec) is a capability that can be added to either current version of the Internet Protocol (IPv4 or IPv6), by means of additional headers.
- IPSec encompasses three functional areas: authentication, confidentiality, and key management.
- Authentication makes use of the HMAC message authentication code. Authentication can be applied to the entire original IP packet (tunnel mode) or to all of the packet except for the IP header (transport mode).
- Confidentiality is provided by an encryption format known as encapsulating security payload. Both tunnel and transport modes can be accommodated.
- IPSec defines a number of techniques for key management.

The Internet community has developed application-specific security mechanisms in a number of

application areas, including electronic mail (S/MIME, PGP), client/server (Kerberos), Web access (Secure Sockets Layer), and others. However, users have some security concerns that cut across protocol layers. For example, an enterprise can run a secure, private TCP/IP network by disallowing links to untrusted sites, encrypting packets that leave the premises, and authenticating packets that enter the premises. By implementing security at the IP level, an organization can ensure secure networking not only for applications that have security mechanisms but also for the many security-ignorant applications.

IP-level security encompasses three functional areas: authentication, confidentiality, and key management. The authentication mechanism assures that a received packet was, in fact, transmitted by the party identified as the source in the packet header. In addition, this mechanism assures that the packet has not been altered in transit. The confidentiality facility enables communicating nodes to encrypt messages to prevent eavesdropping by third parties. The key management facility is concerned with the secure exchange of keys.

We begin this chapter with an overview of IP security (IPSec) and an introduction to the IPSec architecture. We then look at each of the three functional areas in detail. The appendix to this chapter reviews internet protocols.

PREV NEXT

[Page 485]

16.1. IP Security Overview

In response to these issues, the IAB included authentication and encryption as necessary security features in the next-generation IP, which has been issued as IPv6. Fortunately, these security capabilities were designed to be usable both with the current IPv4 and the future IPv6. This means that vendors can begin offering these features now, and many vendors do now have some IPSec capability in their products.

Applications of IPSec

IPSec provides the capability to secure communications across a LAN, across private and public WANs, and across the Internet. Examples of its use include the following:

- **Secure branch office connectivity over the Internet:** A company can build a secure virtual private network over the Internet or over a public WAN. This enables a business to rely heavily on the Internet and reduce its need for private networks, saving costs and network management overhead.
- **Secure remote access over the Internet:** An end user whose system is equipped with IP security protocols can make a local call to an Internet service provider (ISP) and gain secure access to a company network. This reduces the cost of toll charges for traveling employees and telecommuters.
- **Establishing extranet and intranet connectivity with partners:** IPSec can be used to secure communication with other organizations, ensuring authentication and confidentiality and providing a key exchange mechanism.
- **Enhancing electronic commerce security:** Even though some Web and electronic commerce applications have built-in security protocols, the use of IPSec enhances that security.

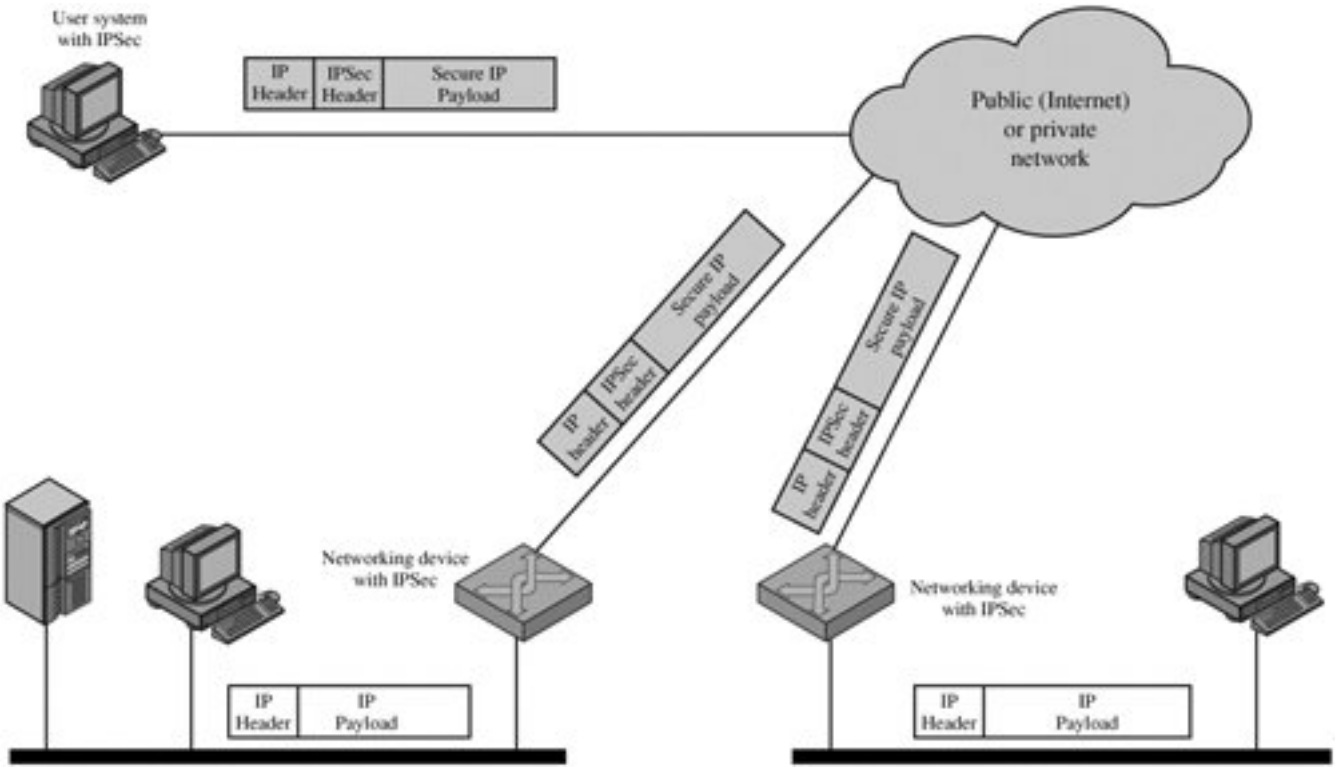
The principal feature of IPSec that enables it to support these varied applications is that it can encrypt and/or authenticate *all* traffic at the IP level. Thus, all distributed applications, including remote logon, client/server, e-mail, file transfer, Web access, and so on, can be secured.

Figure 16.1 is a typical scenario of IPSec usage. An organization maintains LANs at dispersed locations. Nonsecure IP traffic is conducted on each LAN. For traffic offsite, through some sort of private or public WAN, IPSec protocols are used. These protocols operate in networking devices, such as a router or firewall, that connect each LAN to the outside world. The IPSec networking device will typically encrypt and compress all traffic going into the WAN, and decrypt and decompress traffic coming from the WAN; these operations are transparent to workstations and servers on the LAN. Secure transmission is also possible with individual users who dial into the WAN. Such user workstations must implement the IPSec protocols to provide security.

[Page 486]

Figure 16.1. An IP Security Scenario

[View full size image]



Benefits of IPSec

[MARK97] lists the following benefits of IPSec:

- When IPSec is implemented in a firewall or router, it provides strong security that can be applied to all traffic crossing the perimeter. Traffic within a company or workgroup does not incur the overhead of security-related processing.
- IPSec in a firewall is resistant to bypass if all traffic from the outside must use IP, and the firewall is the only means of entrance from the Internet into the organization.
- IPSec is below the transport layer (TCP, UDP) and so is transparent to applications. There is no need to change software on a user or server system when IPSec is implemented in the firewall or router. Even if IPSec is implemented in end systems, upper-layer software, including applications, is not affected.
- IPSec can be transparent to end users. There is no need to train users on security mechanisms, issue keying material on a per-user basis, or revoke keying material when users leave the organization.
- IPSec can provide security for individual users if needed. This is useful for offsite workers and for setting up a secure virtual subnetwork within an organization for sensitive applications.

Routing Applications

In addition to supporting end users and protecting premises systems and networks, IPSec can play a vital role in the routing architecture required for internetworking. [HUIT98] lists the following examples of the use of IPSec. IPSec can assure that

- A router advertisement (a new router advertises its presence) comes from an authorized router
- A neighbor advertisement (a router seeks to establish or maintain a neighbor relationship with a router in another routing domain) comes from an authorized router.

- A redirect message comes from the router to which the initial packet was sent.
- A routing update is not forged.

Without such security measures, an opponent can disrupt communications or divert some traffic. Routing protocols such as OSPF should be run on top of security associations between routers that are defined by IPSec.

PREV NEXT

16.2. IP Security Architecture

The IPSec specification has become quite complex. To get a feel for the overall architecture, we begin with a look at the documents that define IPSec. Then we discuss IPSec services and introduce the concept of security association.

IPSec Documents

The IPSec specification consists of numerous documents. The most important of these, issued in November of 1998, are RFCs 2401, 2402, 2406, and 2408:

- RFC 2401: An overview of a security architecture
- RFC 2402: Description of a packet authentication extension to IPv4 and IPv6
- RFC 2406: Description of a packet encryption extension to IPv4 and IPv6
- RFC 2408: Specification of key management capabilities

Support for these features is mandatory for IPv6 and optional for IPv4. In both cases, the security features are implemented as extension headers that follow the main IP header. The extension header for authentication is known as the Authentication header; that for encryption is known as the Encapsulating Security Payload (ESP) header.

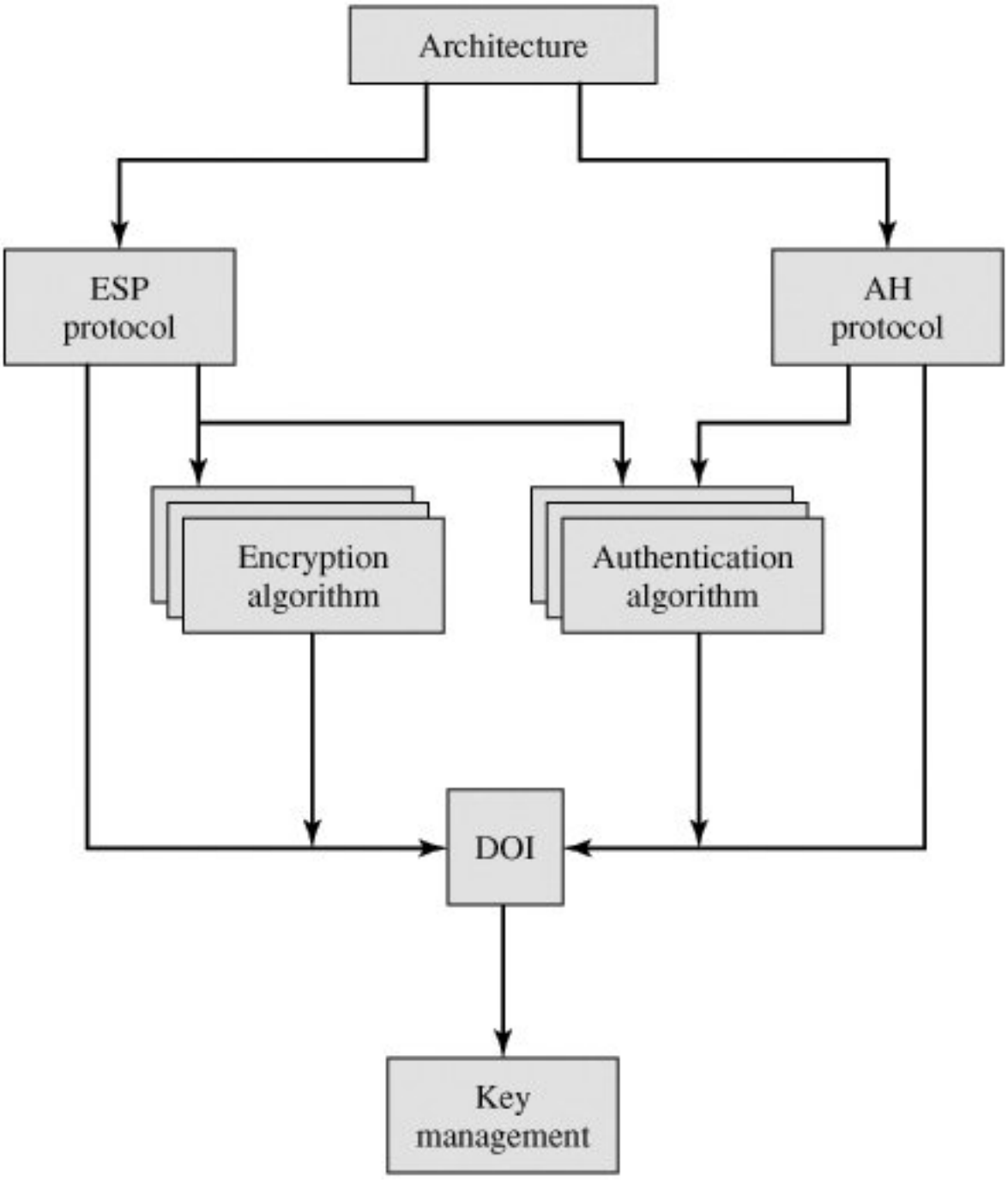
In addition to these four RFCs, a number of additional drafts have been published by the IP Security Protocol Working Group set up by the IETF. The documents are divided into seven groups, as depicted in Figure 16.2 (RFC 2401):

- **Architecture:** Covers the general concepts, security requirements, definitions, and mechanisms defining IPSec technology.

- **Encapsulating Security Payload (ESP):** Covers the packet format and general issues related to the use of the ESP for packet encryption and, optionally, authentication.
- **Authentication Header (AH):** Covers the packet format and general issues related to the use of AH for packet authentication.
- **Encryption Algorithm:** A set of documents that describe how various encryption algorithms are used for ESP.
- **Authentication Algorithm:** A set of documents that describe how various authentication algorithms are used for AH and for the authentication option of ESP.
- **Key Management:** Documents that describe key management schemes.
- **Domain of Interpretation (DOI):** Contains values needed for the other documents to relate to each other. These include identifiers for approved encryption and authentication algorithms, as well as operational parameters such as key lifetime.

Figure 16.2. IPSec Document Overview

(This item is displayed on page 488 in the print version)



IPSec Services

IPSec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. Two protocols are used to provide security: an authentication protocol designated by the header of the protocol, Authentication Header (AH); and a combined encryption/authentication protocol designated by the format of the packet for that protocol, Encapsulating Security Payload (ESP). The services are

- Access control
- Connectionless integrity
- Data origin authentication
- Rejection of replayed packets (a form of partial sequence integrity)
- Confidentiality (encryption)
- Limited traffic flow confidentiality

Table 16.1 shows which services are provided by the AH and ESP protocols. For ESP, there are two cases: with and without the authentication option. Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols.

Table 16.1. IPSec Services

(This item is displayed on page 490 in the print version)

[View full size image]

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✔	✔	✔
Connectionless integrity	✔		✔
Data origin authentication	✔		✔
Rejection of replayed packets	✔	✔	✔
Confidentiality		✔	✔
Limited traffic flow confidentiality		✔	✔

Security Associations

A key concept that appears in both the authentication and confidentiality mechanisms for IP is the security association (SA). An association is a one-way relationship between a sender and a receiver that affords security services to the traffic carried on it. If a peer relationship is needed, for two-way secure exchange, then two security associations are required. Security services are afforded to an SA for the use of AH or ESP, but not both.

A security association is uniquely identified by three parameters:

- **Security Parameters Index (SPI):** A bit string assigned to this SA and having local significance only. The SPI is carried in AH and ESP headers to enable the receiving system to select the SA under which a received packet will be processed.
- **IP Destination Address:** Currently, only unicast addresses are allowed; this is the address of the destination endpoint of the SA, which may be an end user system or a network system such as a firewall or router.
- **Security Protocol Identifier:** This indicates whether the association is an AH or ESP security association.

Hence, in any IP packet,[1] the security association is uniquely identified by the Destination Address in the IPv4 or IPv6 header and the SPI in the enclosed extension header (AH or ESP).

[1] In this chapter, the term *IP packet* refers to either an IPv4 datagram or an IPv6 packet.

SA Parameters

In each IPSec implementation, there is a nominal[2] Security Association Database that defines the parameters associated with each SA. A security association is normally defined by the following parameters:

[2] Nominal in the sense that the functionality provided by a Security Association Database must be present in any IPSec implementation, but the way in which that functionality is provided is up to the implementer.

- **Sequence Number Counter:** A 32-bit value used to generate the Sequence Number field in AH or ESP headers, described in Section 16.3 (required for all implementations).
- **Sequence Counter Overflow:** A flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA (required for all implementations).
- **Anti-Replay Window:** Used to determine whether an inbound AH or ESP packet is a replay, described in Section 16.3 (required for all implementations).
- **AH Information:** Authentication algorithm, keys, key lifetimes, and related parameters being used with AH (required for AH implementations).

- **ESP Information:** Encryption and authentication algorithm, keys, initialization values, key lifetimes, and related parameters being used with ESP (required for ESP implementations).
- **Lifetime of This Security Association:** A time interval or byte count after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur (required for all implementations).
- **IPSec Protocol Mode:** Tunnel, transport, or wildcard (required for all implementations). These modes are discussed later in this section.
- **Path MTU:** Any observed path maximum transmission unit (maximum size of a packet that can be transmitted without fragmentation) and aging variables (required for all implementations).

The key management mechanism that is used to distribute keys is coupled to the authentication and privacy mechanisms only by way of the Security Parameters Index. Hence, authentication and privacy have been specified independent of any specific key management mechanism.

SA Selectors

IPSec provides the user with considerable flexibility in the way in which IPSec services are applied to IP traffic. As we will see later, SAs can be combined in a number of ways to yield the desired user configuration. Furthermore, IPSec provides a high degree of granularity in discriminating between traffic that is afforded IPSec protection and traffic that is allowed to bypass IPSec, in the former case relating IP traffic to specific SAs.

The means by which IP traffic is related to specific SAs (or no SA in the case of traffic allowed to bypass IPSec) is the nominal Security Policy Database (SPD). In its simplest form, an SPD contains entries, each of which defines a subset of IP traffic and points to an SA for that traffic. In more complex environments, there may be multiple entries that potentially relate to a single SA or multiple SAs associated with a single SPD entry. The reader is referred to the relevant IPSec documents for a full discussion.

Each SPD entry is defined by a set of IP and upper-layer protocol field values, called *selectors*. In effect, these selectors are used to filter outgoing traffic in order to map it into a particular SA. Outbound processing obeys the following general sequence for each IP packet:

1.

Compare the values of the appropriate fields in the packet (the selector fields) against the SPD to find a matching SPD entry, which will point to zero or more SAs.

2. Determine the SA if any for this packet and its associated SPI.

3. Do the required IPSec processing (i.e., AH or ESP processing).

The following selectors determine an SPD entry:

- **Destination IP Address:** This may be a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address. The latter two are required to support more than one destination system sharing the same SA (e.g., behind a firewall).

- **Source IP Address:** This may be a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address. The latter two are required to support more than one source system sharing the same SA (e.g., behind a firewall).
- **UserID:** A user identifier from the operating system. This is not a field in the IP or upper-layer headers but is available if IPSec is running on the same operating system as the user.
- **Data Sensitivity Level:** Used for systems providing information flow security (e.g., Secret or Unclassified).
- **Transport Layer Protocol:** Obtained from the IPv4 Protocol or IPv6 Next Header field. This may be an individual protocol number, a list of protocol numbers, or a range of protocol numbers.
- **Source and Destination Ports:** These may be individual TCP or UDP port values, an enumerated list of ports, or a wildcard port.

Transport and Tunnel Modes

Both AH and ESP support two modes of use: transport and tunnel mode. The operation of these two modes is best understood in the context of a description of AH and ESP, which are covered in Sections 16.3 and 16.4, respectively. Here we provide a brief overview.

Transport Mode

Transport mode provides protection primarily for upper-layer protocols. That is, transport mode protection extends to the payload of an IP packet. Examples include a TCP or UDP segment or an ICMP packet, all of which operate directly above IP in a host protocol stack. Typically, transport mode is used for end-to-end communication between two hosts (e.g., a client and a server, or two workstations). When a host runs AH or ESP over IPv4, the payload is the data that normally follow the IP header. For IPv6, the payload is the data that normally follow both the IP header and any IPv6 extensions headers that are present, with the possible exception of the destination options header, which may be included in the protection.

ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header. AH in transport mode authenticates the IP payload and selected portions of the IP header.

Tunnel Mode

Tunnel mode provides protection to the entire IP packet. To achieve this, after the AH or ESP fields are

added to the IP packet, the entire packet plus security fields is treated as the payload of new "outer" IP packet with a new outer IP header. The entire original, or inner, packet travels through a "tunnel" from one point of an IP network to another; no routers along the way are able to examine the inner IP header. Because the original packet is encapsulated, the new, larger packet may have totally different source and destination addresses, adding to the security. Tunnel mode is used when one or both ends of an SA are a security gateway, such as a firewall or router that implements IPSec. With tunnel mode, a number of hosts on networks behind firewalls may engage in secure communications without implementing IPSec. The unprotected packets generated by such hosts are tunneled through external networks by tunnel mode SAs set up by the IPSec software in the firewall or secure router at the boundary of the local network.

Here is an example of how tunnel mode IPSec operates. Host A on a network generates an IP packet with the destination address of host B on another network. This packet is routed from the originating host to a firewall or secure router at the boundary of A's network. The firewall filters all outgoing packets to determine the need for IPSec processing. If this packet from A to B requires IPSec, the firewall performs IPSec processing and encapsulates the packet with an outer IP header. The source IP address of this outer IP packet is this firewall, and the destination address may be a firewall that forms the boundary to B's local network. This packet is now routed to B's firewall, with intermediate routers examining only the outer IP header. At B's firewall, the outer IP header is stripped off, and the inner packet is delivered to B.

ESP in tunnel mode encrypts and optionally authenticates the entire inner IP packet, including the inner IP header. AH in tunnel mode authenticates the entire inner IP packet and selected portions of the outer IP header.

Table 16.2 summarizes transport and tunnel mode functionality.

Table 16.2. Tunnel Mode and Transport Mode Functionality

	Transport Mode SA	Tunnel Mode SA
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts entire inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts entire inner IP packet. Authenticates inner IP packet.

16.3. Authentication Header

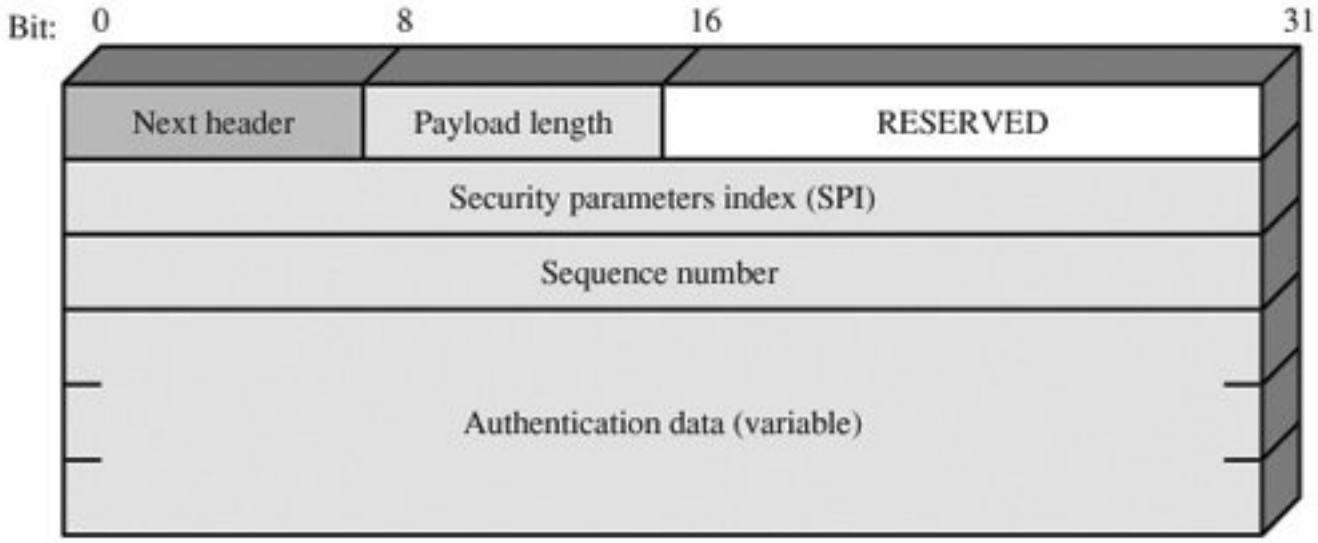
The Authentication Header provides support for data integrity and authentication of IP packets. The data integrity feature ensures that undetected modification to a packet's content in transit is not possible. The authentication feature enables an end system or network device to authenticate the user or application and filter traffic accordingly; it also prevents the address spoofing attacks observed in today's Internet. The AH also guards against the replay attack described later in this section.

Authentication is based on the use of a message authentication code (MAC), as described in Chapter 11; hence the two parties must share a secret key.

The Authentication Header consists of the following fields (Figure 16.3):

- **Next Header (8 bits):** Identifies the type of header immediately following this header.
- **Payload Length (8 bits):** Length of Authentication Header in 32-bit words, minus 2. For example, the default length of the authentication data field is 96 bits, or three 32-bit words. With a three-word fixed header, there are a total of six words in the header, and the Payload Length field has a value of 4.
- **Reserved (16 bits):** For future use.
- **Security Parameters Index (32 bits):** Identifies a security association.
- **Sequence Number (32 bits):** A monotonically increasing counter value, discussed later.
- **Authentication Data (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value (ICV), or MAC, for this packet, discussed later.

Figure 16.3. IPSec Authentication Header



Anti-Replay Service

A replay attack is one in which an attacker obtains a copy of an authenticated packet and later transmits

it to the intended destination. The receipt of duplicate, authenticated IP packets may disrupt service in some way or may have some other undesired consequence. The Sequence Number field is designed to thwart such attacks. First, we discuss sequence number generation by the sender, and then we look at how it is processed by the recipient.

When a new SA is established, the **sender** initializes a sequence number counter to 0. Each time that a packet is sent on this SA, the sender increments the counter and places the value in the Sequence Number field. Thus, the first value to be used is 1. If anti-replay is enabled (the default), the sender must not allow the sequence number to cycle past 2^{32} 1 back to zero. Otherwise, there would be multiple valid packets with the same sequence number. If the limit of 2^{32} 1 is reached, the sender should terminate this SA and negotiate a new SA with a new key.

Because IP is a connectionless, unreliable service, the protocol does not guarantee that packets will be delivered in order and does not guarantee that all packets will be delivered. Therefore, the IPSec authentication document dictates that the **receiver** should implement a window of size *W*, with a default of *W* = 64. The right edge of the window represents the highest sequence number, *N*, so far received for a valid packet. For any packet with a sequence number in the range from *N W* + 1 to *N* that has been correctly received (i.e., properly authenticated), the corresponding slot in the window is marked (Figure 16.4). Inbound processing proceeds as follows when a packet is received:

---

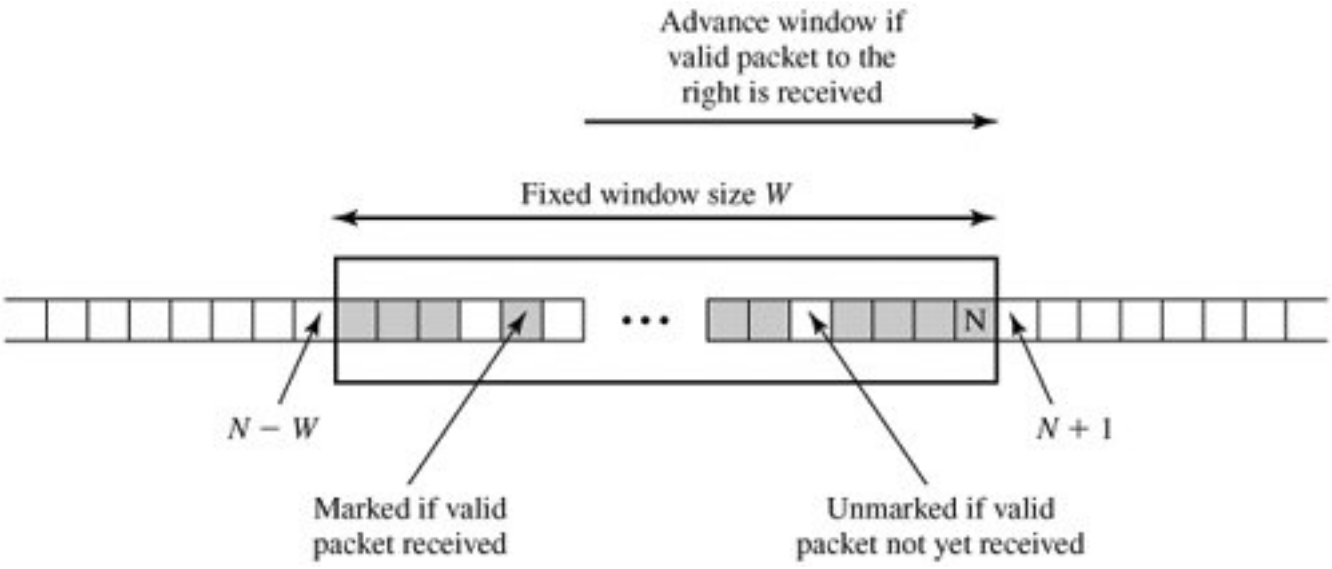
1. If the received packet falls within the window and is new, the MAC is checked. If the packet is authenticated, the corresponding slot in the window is marked.

2. If the received packet is to the right of the window and is new, the MAC is checked. If the packet is authenticated, the window is advanced so that this sequence number is the right edge of the window, and the corresponding slot in the window is marked.

3. If the received packet is to the left of the window, or if authentication fails, the packet is discarded; this is an auditable event.

Figure 16.4. Antireplay Mechanism

[View full size image]



Integrity Check Value

The Authentication Data field holds a value referred to as the Integrity Check Value. The ICV is a message authentication code or a truncated version of a code produced by a MAC algorithm. The current specification dictates that a compliant implementation must support

- HMAC-MD5-96
- HMAC-SHA-1-96

Both of these use the HMAC algorithm, the first with the MD5 hash code and the second with the SHA-1 hash code (all of these algorithms are described in Chapter 12). In both cases, the full HMAC value is calculated but then truncated by using the first 96 bits, which is the default length for the Authentication Data field.

The MAC is calculated over

- IP header fields that either do not change in transit (immutable) or that are predictable in value upon arrival at the endpoint for the AH SA. Fields that may change in transit and whose value on arrival are unpredictable are set to zero for purposes of calculation at both source and destination.
- The AH header other than the Authentication Data field. The Authentication Data field is set to zero for purposes of calculation at both source and destination.
- The entire upper-level protocol data, which is assumed to be immutable in transit (e.g., a TCP segment or an inner IP packet in tunnel mode).

For IPv4, examples of immutable fields are Internet Header Length and Source Address. An example of a mutable but predictable field is the Destination Address (with loose or strict source routing). Examples of mutable fields that are zeroed prior to ICV calculation are the Time to Live and Header Checksum fields. Note that both source and destination address fields are protected, so that address spoofing is prevented.

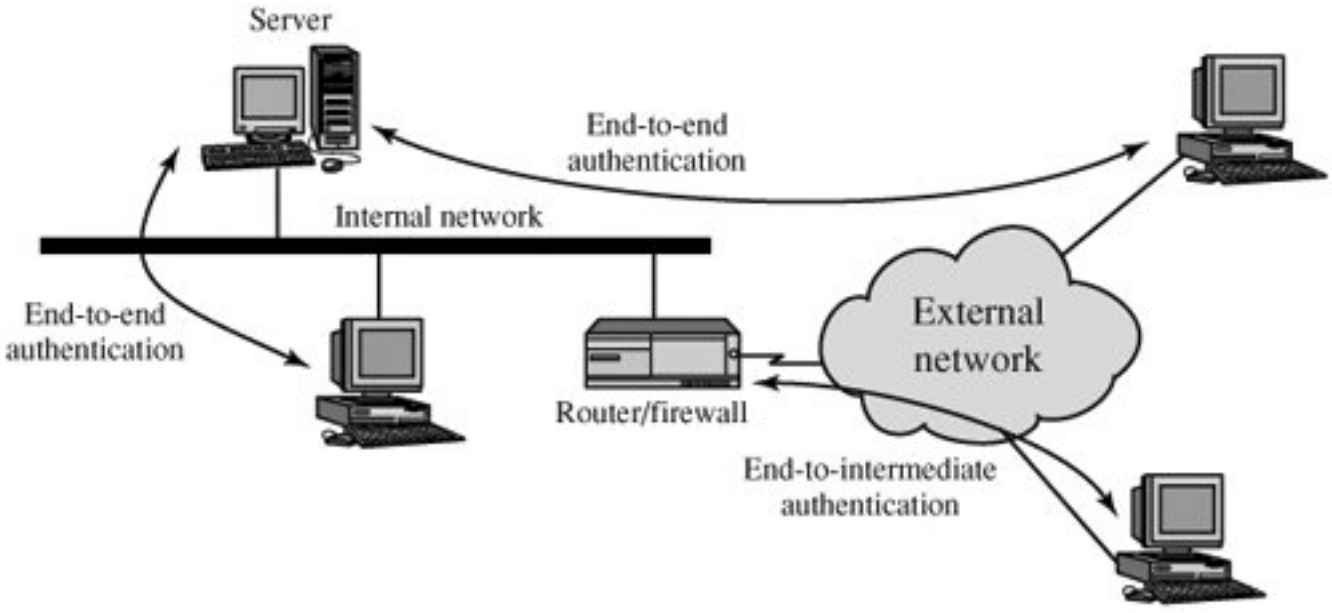
For IPv6, examples in the base header are Version (immutable), Destination Address (mutable but predictable), and Flow Label (mutable and zeroed for calculation).

Transport and Tunnel Modes

Figure 16.5 shows two ways in which the IPSec authentication service can be used. In one case, authentication is provided directly between a server and client workstations; the workstation can be either on the same network as the server or on an external network. As long as the workstation and the server share a protected secret key, the authentication process is secure. This case uses a transport mode SA. In the other case, a remote workstation authenticates itself to the corporate firewall, either for access to the entire internal network or because the requested server does not support the authentication feature. This case uses a tunnel mode SA.

Figure 16.5. End-to-End versus End-to-Intermediate Authentication

[View full size image]

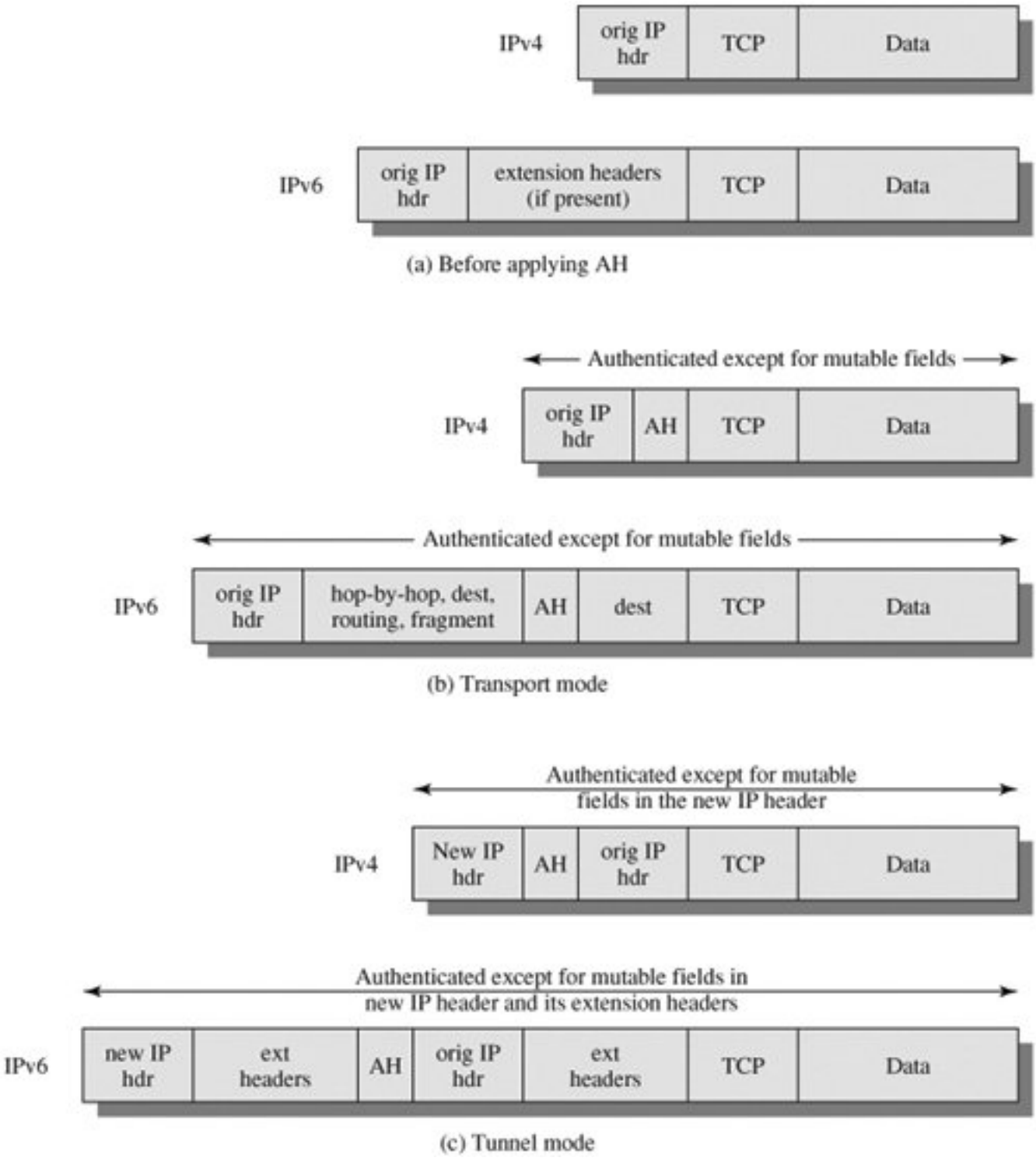


In this subsection, we look at the scope of authentication provided by AH and the authentication header location for the two modes. The considerations are somewhat different for IPv4 and IPv6. Figure 16.6a shows typical IPv4 and IPv6 packets. In this case, the IP payload is a TCP segment; it could also be a data unit for any other protocol that uses IP, such as UDP or ICMP.

For **transport mode AH** using IPv4, the AH is inserted after the original IP header and before the IP payload (e.g., a TCP segment); this is shown in the upper part of Figure 16.6b. Authentication covers the entire packet, excluding mutable fields in the IPv4 header that are set to zero for MAC calculation.

Figure 16.6. Scope of AH Authentication

[View full size image]



In the context of IPv6, AH is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the AH appears after the IPv6 base header and the hop-by-hop, routing, and fragment extension headers. The destination options extension header could appear before or after the AH header, depending on the semantics desired. Again, authentication covers the entire packet, excluding mutable fields that are set to zero for MAC calculation.

For **tunnel mode AH**, the entire original IP packet is authenticated, and the AH is inserted between the original IP header and a new outer IP header (Figure 16.6c). The inner IP header carries the ultimate source and destination addresses, while an outer IP header may contain different IP addresses (e.g., addresses of firewalls or other security gateways).

With tunnel mode, the entire inner IP packet, including the entire inner IP header is protected by AH. The outer IP header (and in the case of IPv6, the outer IP extension headers) is protected except for mutable and unpredictable fields.

PREV NEXT

16.4. Encapsulating Security Payload

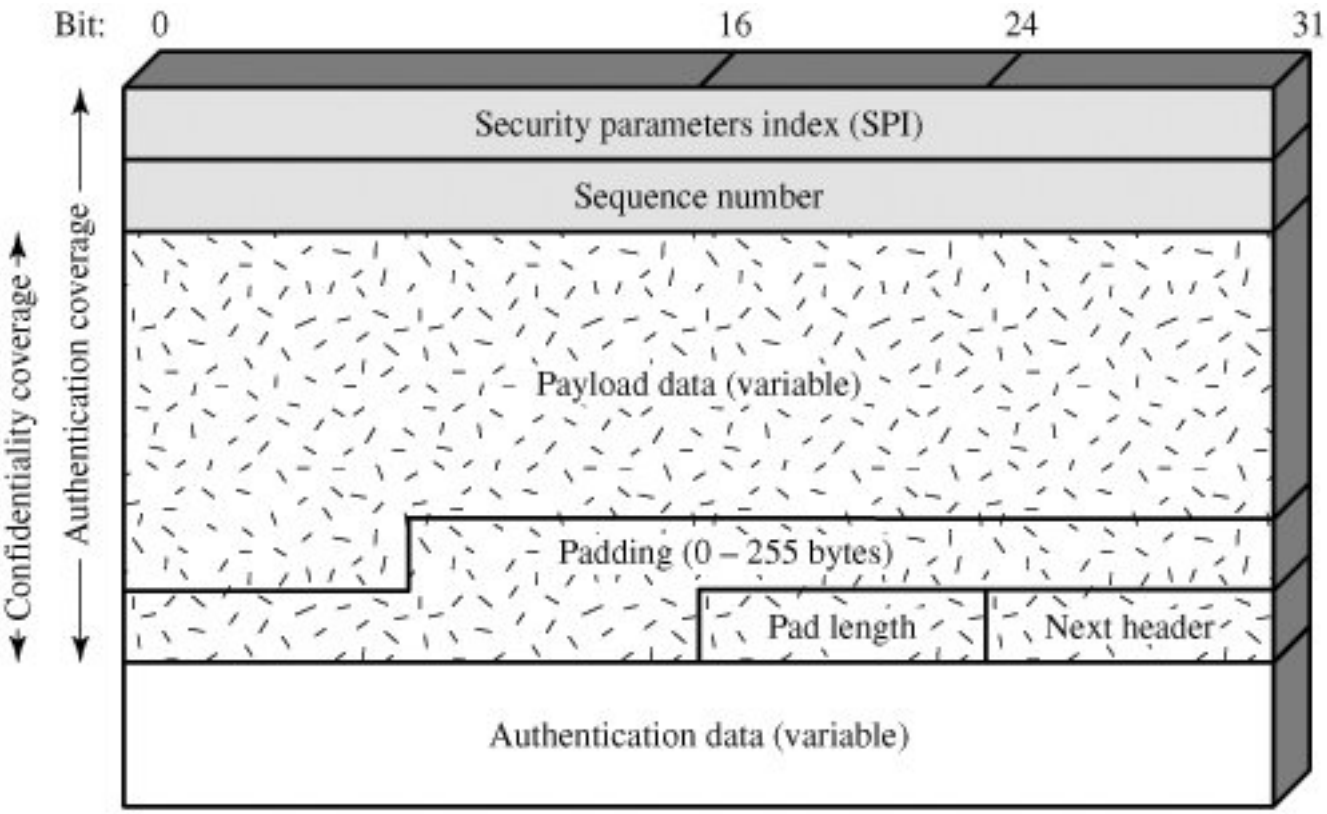
The Encapsulating Security Payload provides confidentiality services, including confidentiality of message contents and limited traffic flow confidentiality. As an optional feature, ESP can also provide an authentication service.

ESP Format

Figure 16.7 shows the format of an ESP packet. It contains the following fields:

- **Security Parameters Index (32 bits):** Identifies a security association.
- **Sequence Number (32 bits):** A monotonically increasing counter value; this provides an anti-replay function, as discussed for AH.
- **Payload Data (variable):** This is a transport-level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption.

Figure 16.7. IPSec ESP format



- **Padding (0255 bytes):** The purpose of this field is discussed later.
- **Pad Length (8 bits):** Indicates the number of pad bytes immediately preceding this field.
- **Next Header (8 bits):** Identifies the type of data contained in the payload data field by identifying the first header in that payload (for example, an extension header in IPv6, or an upper-layer protocol such as TCP).

- **Authentication Data (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value computed over the ESP packet minus the Authentication Data field.

Encryption and Authentication Algorithms

The Payload Data, Padding, Pad Length, and Next Header fields are encrypted by the ESP service. If the algorithm used to encrypt the payload requires cryptographic synchronization data, such as an initialization vector (IV), then these data may be carried explicitly at the beginning of the Payload Data field. If included, an IV is usually not encrypted, although it is often referred to as being part of the ciphertext.

The current specification dictates that a compliant implementation must support DES in cipher block chaining (CBC) mode (described in Chapter 3). A number of other algorithms have been assigned identifiers in the DOI document and could therefore easily be used for encryption; these include

- Three-key triple DES
- RC5
- IDEA
- Three-key triple IDEA
- CAST
- Blowfish

Many of these algorithms are described in Chapter 6.

As with AH, ESP supports the use of a MAC with a default length of 96 bits. Also as with AH, the current specification dictates that a compliant implementation must support HMAC-MD5-96 and HMAC-SHA-1-96.

Padding

The Padding field serves several purposes:

- If an encryption algorithm requires the plaintext to be a multiple of some number of bytes (e.g., the multiple of a single block for a block cipher), the Padding field is used to expand the plaintext (consisting of the Payload Data, Padding, Pad Length, and Next Header fields) to the required length.
- The ESP format requires that the Pad Length and Next Header fields be right aligned within a 32-bit word. Equivalently, the ciphertext must be an integer multiple of 32 bits. The Padding field is used to assure this alignment.
- Additional padding may be added to provide partial traffic flow confidentiality by concealing the actual length of the payload.

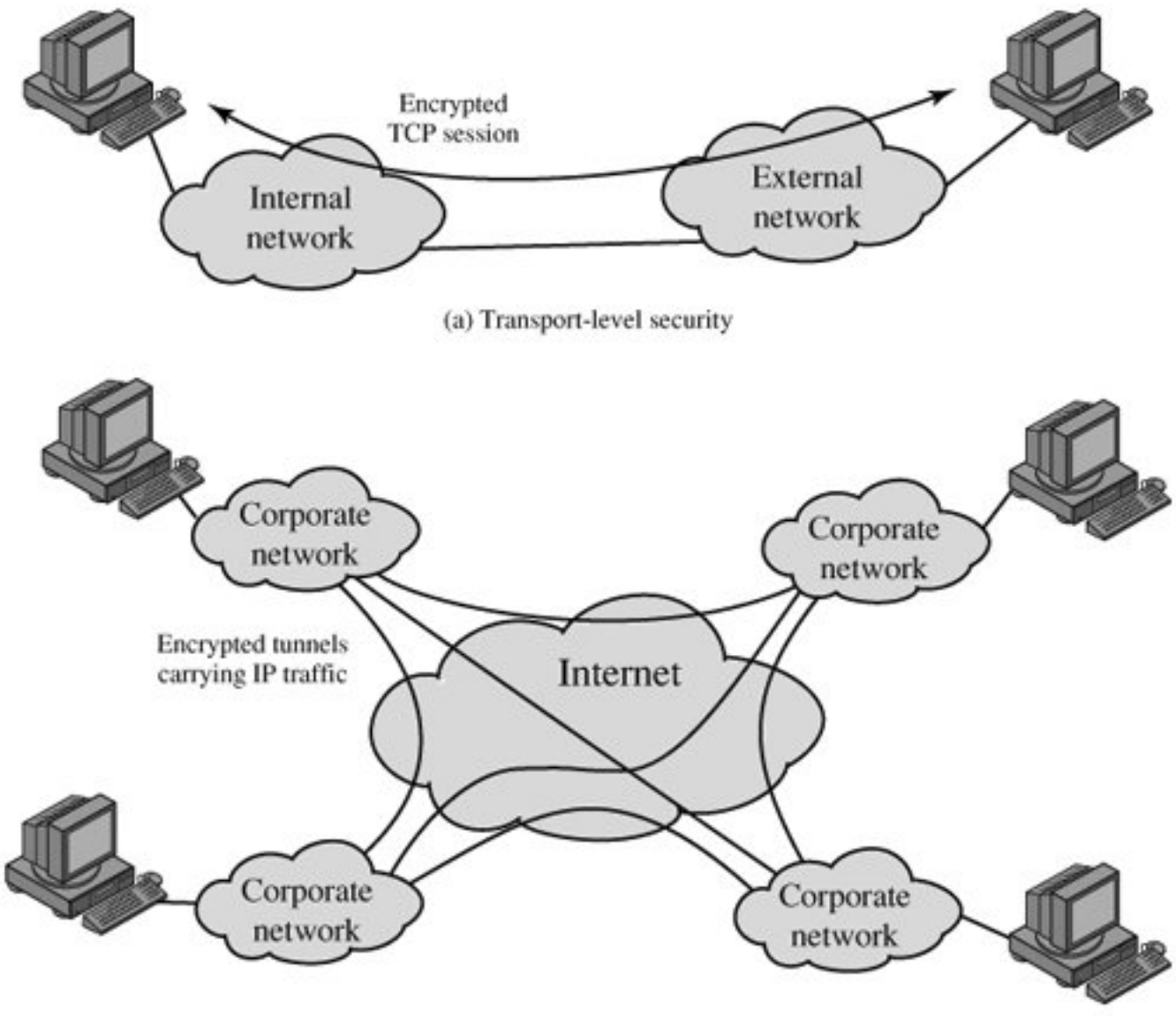
Transport and Tunnel Modes

Figure 16.8 shows two ways in which the IPSec ESP service can be used. In the upper part of the figure, encryption (and optionally authentication) is provided directly between two hosts. Figure 16.8b shows how tunnel mode operation can be used to set up a *virtual private network*. In this example, an organization has four private networks interconnected across the Internet. Hosts on the internal networks use the Internet for transport of data but do not interact with other Internet-based hosts. By

terminating the tunnels at the security gateway to each internal network, the configuration allows the hosts to avoid implementing the security capability. The former technique is support by a transport mode SA, while the latter technique uses a tunnel mode SA.

Figure 16.8. Transport-Mode vs. Tunnel-Mode Encryption

[View full size image]



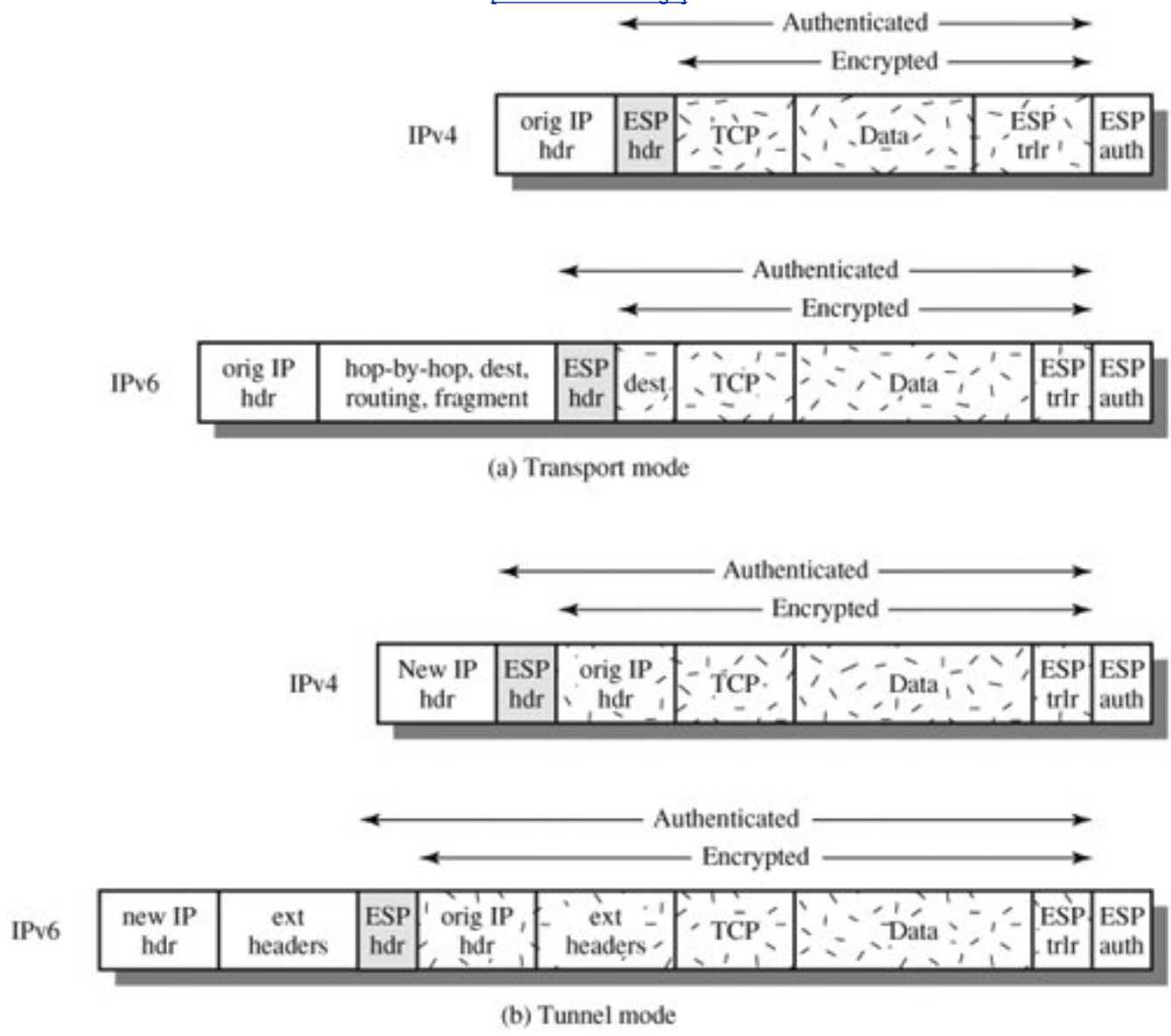
In this section, we look at the scope of ESP for the two modes. The considerations are somewhat different for IPv4 and IPv6. As with our discussion of AH scope, we will use the packet formats of Figure 16.6a as a starting point.

Transport Mode ESP

Transport mode ESP is used to encrypt and optionally authenticate the data carried by IP (e.g., a TCP segment), as shown in Figure 16.9a. For this mode using IPv4, the ESP header is inserted into the IP packet immediately prior to the transport-layer header (e.g., TCP, UDP, ICMP) and an ESP trailer (Padding, Pad Length, and Next Header fields) is placed after the IP packet; if authentication is selected, the ESP Authentication Data field is added after the ESP trailer. The entire transport-level segment plus the ESP trailer are encrypted. Authentication covers all of the ciphertext plus the ESP header.

Figure 16.9. Scope of ESP Encryption and Authentication

[View full size image]



In the context of IPv6, ESP is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the ESP header appears after the IPv6 base header and the hop-by-hop, routing, and fragment extension headers. The destination options extension header could appear before or after the ESP header, depending on the semantics desired. For IPv6, encryption covers the entire transport-level segment plus the ESP trailer plus the destination options extension header if it occurs after the ESP header. Again, authentication covers the ciphertext plus the ESP header.

Transport mode operation may be summarized as follows:

1.

   At the source, the block of data consisting of the ESP trailer plus the entire transport-layer segment is encrypted and the plaintext of this block is replaced with its ciphertext to form the IP packet for transmission. Authentication is added if this option is selected.

2.

The packet is then routed to the destination. Each intermediate router needs to examine and process the IP header plus any plaintext IP extension headers but does not need to examine the ciphertext.

3.

The destination node examines and processes the IP header plus any plaintext IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext transport-layer segment.

Transport mode operation provides confidentiality for any application that uses it, thus avoiding the need to implement confidentiality in every individual application. This mode of operation is also reasonably efficient, adding little to the total length of the IP packet. One drawback to this mode is that it is possible to do traffic analysis on the transmitted packets.

Tunnel Mode ESP

Tunnel mode ESP is used to encrypt an entire IP packet (Figure 16.9b). For this mode, the ESP header is prefixed to the packet and then the packet plus the ESP trailer is encrypted. This method can be used to counter traffic analysis.

Because the IP header contains the destination address and possibly source routing directives and hop-by-hop option information, it is not possible simply to transmit the encrypted IP packet prefixed by the ESP header. Intermediate routers would be unable to process such a packet. Therefore, it is necessary to encapsulate the entire block (ESP header plus ciphertext plus Authentication Data, if present) with a new IP header that will contain sufficient information for routing but not for traffic analysis.

Whereas the transport mode is suitable for protecting connections between hosts that support the ESP feature, the tunnel mode is useful in a configuration that includes a firewall or other sort of security gateway that protects a trusted network from external networks. In this latter case, encryption occurs only between an external host and the security gateway or between two security gateways. This relieves hosts on the internal network of the processing burden of encryption and simplifies the key distribution task by reducing the number of needed keys. Further, it thwarts traffic analysis based on ultimate destination.

Consider a case in which an external host wishes to communicate with a host on an internal network protected by a firewall, and in which ESP is implemented in the external host and the firewalls. The following steps occur for transfer of a transport-layer segment from the external host to the internal host:

1.

The source prepares an inner IP packet with a destination address of the target internal host. This packet is prefixed by an ESP header; then the packet and ESP trailer are encrypted and Authentication Data may be added. The resulting block is encapsulated with a new IP header (base header plus optional extensions such as routing and hop-by-hop options for IPv6) whose destination address is the firewall; this forms the outer IP packet.

2.

The outer packet is routed to the destination firewall. Each intermediate router needs to examine and process the outer IP header plus any outer IP extension headers but does not need to examine the ciphertext.

3.

The destination firewall examines and processes the outer IP header plus any outer IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext inner IP packet. This packet is then transmitted in the internal network.

4.

The inner packet is routed through zero or more routers in the internal network to the destination host.

16.5. Combining Security Associations

An individual SA can implement either the AH or ESP protocol but not both. Sometimes a particular traffic flow will call for the services provided by both AH and ESP. Further, a particular traffic flow may require IPSec services between hosts and, for that same flow, separate services between security gateways, such as firewalls. In all of these cases, multiple SAs must be employed for the same traffic flow to achieve the desired IPSec services. The term *security association bundle* refers to a sequence of SAs through which traffic must be processed to provide a desired set of IPSec services. The SAs in a bundle may terminate at different endpoints or at the same endpoints.

Security associations may be combined into bundles in two ways:

- **Transport adjacency:** Refers to applying more than one security protocol to the same IP packet, without invoking tunneling. This approach to combining AH and ESP allows for only one level of combination; further nesting yields no added benefit since the processing is performed at one IPsec instance: the (ultimate) destination.
- **Iterated tunneling:** Refers to the application of multiple layers of security protocols effected through IP tunneling. This approach allows for multiple levels of nesting, since each tunnel can originate or terminate at a different IPsec site along the path.

The two approaches can be combined, for example, by having a transport SA between hosts travel part of the way through a tunnel SA between security gateways.

One interesting issue that arises when considering SA bundles is the order in which authentication and encryption may be applied between a given pair of endpoints and the ways of doing so. We examine that issue next. Then we look at combinations of SAs that involve at least one tunnel.

Authentication Plus Confidentiality

Encryption and authentication can be combined in order to transmit an IP packet that has both confidentiality and authentication between hosts. We look at several approaches.

ESP with Authentication Option

This approach is illustrated in Figure 16.9. In this approach, the user first applies ESP to the data to be protected and then appends the authentication data field. There are actually two subcases:

- **Transport mode ESP:** Authentication and encryption apply to the IP payload delivered to the host, but the IP header is not protected.

- **Tunnel mode ESP:** Authentication applies to the entire IP packet delivered to the outer IP destination address (e.g., a firewall), and authentication is performed at that destination. The entire inner IP packet is protected by the privacy mechanism, for delivery to the inner IP destination.

For both cases, authentication applies to the ciphertext rather than the plaintext.

Transport Adjacency

Another way to apply authentication after encryption is to use two bundled transport SAs, with the inner being an ESP SA and the outer being an AH SA. In this case ESP is used without its authentication option. Because the inner SA is a transport SA, encryption is applied to the IP payload. The resulting packet consists of an IP header (and possibly IPv6 header extensions) followed by an ESP. AH is then applied in transport mode, so that authentication covers the ESP plus the original IP header (and extensions) except for mutable fields. The advantage of this approach over simply using a single ESP SA with the ESP authentication option is that the authentication covers more fields, including the source and destination IP addresses. The disadvantage is the overhead of two SAs versus one SA.

Transport-Tunnel Bundle

The use of authentication prior to encryption might be preferable for several reasons. First, because the authentication data are protected by encryption, it is impossible for anyone to intercept the message and alter the authentication data without detection. Second, it may be desirable to store the authentication information with the message at the destination for later reference. It is more convenient to do this if the authentication information applies to the unencrypted message; otherwise the message would have to be reencrypted to verify the authentication information.

One approach to applying authentication before encryption between two hosts is to use a bundle consisting of an inner AH transport SA and an outer ESP tunnel SA. In this case, authentication is applied to the IP payload plus the IP header (and extensions) except for mutable fields. The resulting IP packet is then processed in tunnel mode by ESP; the result is that the entire, authenticated inner packet is encrypted and a new outer IP header (and extensions) is added.

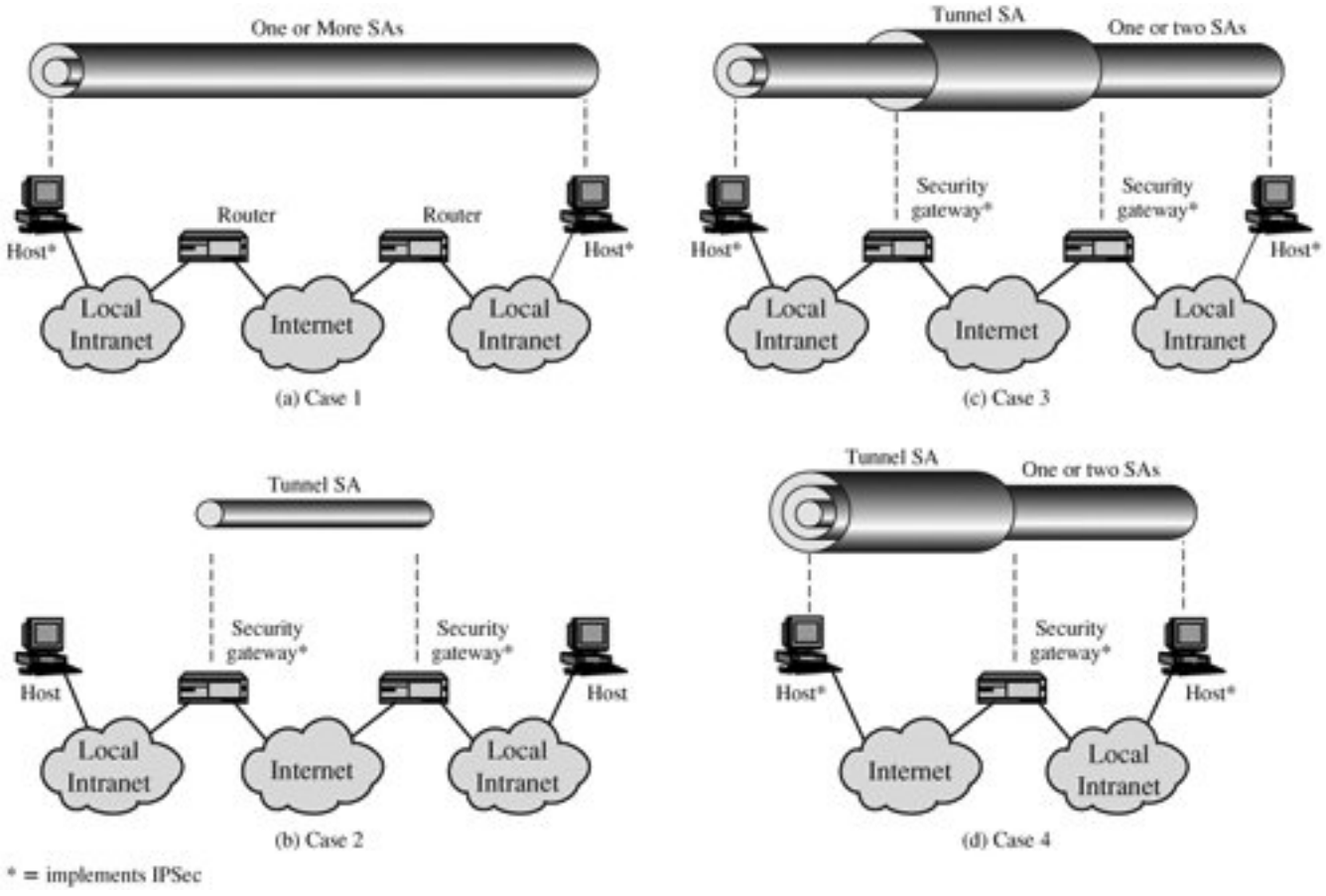
Basic Combinations of Security Associations

The IPSec Architecture document lists four examples of combinations of SAs that must be supported by compliant IPSec hosts (e.g., workstation, server) or security gateways (e.g. firewall, router). These are illustrated in Figure 16.10. The lower part of each case in the figure represents the physical connectivity of the elements; the upper part represents logical connectivity via one or more nested SAs. Each SA can be either AH or ESP. For host-to-host SAs, the mode may be either transport or tunnel; otherwise it must be tunnel mode.

Figure 16.10. Basic Combinations of Security Associations

(This item is displayed on page 505 in the print version)

[View full size image]



In **Case 1**, all security is provided between end systems that implement IPSec. For any two end systems to communicate via an SA, they must share the appropriate secret keys. Among the possible combinations:

a.

AH in transport mode

b.

ESP in transport mode

c.

ESP followed by AH in transport mode (an ESP SA inside an AH SA)

d.

Any one of a, b, or c inside an AH or ESP in tunnel mode

We have already discussed how these various combinations can be used to support authentication, encryption, authentication before encryption, and authentication after encryption.

For **Case 2**, security is provided only between gateways (routers, firewalls, etc.) and no hosts implement IPSec. This case illustrates simple virtual private network support. The security architecture document specifies that only a single tunnel SA is needed for this case. The tunnel could support AH, ESP, or ESP with the authentication option. Nested tunnels are not required because the IPSec services apply to the entire inner packet.

Case 3 builds on Case 2 by adding end-to-end security. The same combinations discussed for cases 1 and 2 are allowed here. The gateway-to-gateway tunnel provides either authentication or confidentiality or both for all traffic between end systems. When the gateway-to-gateway tunnel is ESP, it also provides a limited form of traffic confidentiality. Individual hosts can implement any additional IPSec services required for given applications or given users by means of end-to-end SAs.

Case 4 provides support for a remote host that uses the Internet to reach an organization's firewall and then to gain access to some server or workstation behind the firewall. Only tunnel mode is required between the remote host and the firewall. As in Case 1, one or two SAs may be used between the remote host and the local host.

PREV NEXT

[Page 506 (continued)]

16.6. Key Management

The key management portion of IPSec involves the determination and distribution of secret keys. A typical requirement is four keys for communication between two applications: transmit and receive pairs for both AH and ESP. The IPSec Architecture document mandates support for two types of key management:

- **Manual:** A system administrator manually configures each system with its own keys and with the keys of other communicating systems. This is practical for small, relatively static environments.
- **Automated:** An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration.

The default automated key management protocol for IPSec is referred to as ISAKMP/Oakley and consists of the following elements:

- **Oakley Key Determination Protocol**: Oakley is a key exchange protocol based on the Diffie-Hellman algorithm but providing added security. Oakley is generic in that it does not dictate specific formats.
- **Internet Security Association and Key Management Protocol (ISAKMP):** ISAKMP provides a framework for Internet key management and provides the specific protocol support, including formats, for negotiation of security attributes.

ISAKMP by itself does not dictate a specific key exchange algorithm; rather, ISAKMP consists of a set of message types that enable the use of a variety of key exchange algorithms. Oakley is the specific key exchange algorithm mandated for use with the initial version of ISAKMP.

[Page 507]

We begin with an overview of Oakley and then look at ISAKMP.

Oakley Key Determination Protocol

Oakley is a refinement of the Diffie-Hellman key exchange algorithm. Recall that Diffie-Hellman involves the following interaction between users A and B. There is prior agreement on two global parameters: q, a large prime number; and α a primitive root of q. A selects a random integer X_A as its private key, and transmits to B its public key $Y_A = a^{XA} \bmod q$. Similarly, B selects a random integer X_B as its private key and transmits to A its public key $Y_B = \alpha^{XB} \bmod q$. Each side can now compute the secret session key:

$$K = (Y_B)^{X_A} \bmod q = (Y_A)^{X_B} \bmod q = \alpha^{X_A X_B} \bmod q$$

The Diffie-Hellman algorithm has two attractive features:

- Secret keys are created only when needed. There is no need to store secret keys for a long period of time, exposing them to increased vulnerability.

- The exchange requires no preexisting infrastructure other than an agreement on the global parameters.

However, there are a number of weaknesses to Diffie-Hellman, as pointed out in [HUIT98]:

- It does not provide any information about the identities of the parties.
- It is subject to a man-in-the-middle attack, in which a third party C impersonates B while communicating with A and impersonates A while communicating with B. Both A and B end up negotiating a key with C, which can then listen to and pass on traffic. The man-in-the-middle attack proceeds as follows:
  1. B sends his public key Y_B in a message addressed to A (see Figure 10.8).
  2. The enemy (E) intercepts this message. E saves B's public key and sends a message to A that has B's User ID but E's public key Y_E. This message is sent in such a way that it appears as though it was sent from B's host system. A receives E's message and stores E's public key with B's User ID. Similarly, E sends a message to B with E's public key, purporting to come from A.
  3. B computes a secret key K_1 based on B's private key and Y_E. A computes a secret key K_2 based on A's private key and Y_E. E computes K_1 using E's secret key X_E and Y_B and computer K_2 using Y_E and Y_B.
  4. From now on E is able to relay messages from A to B and from B to A, appropriately changing their encipherment en route in such a way that neither A nor B will know that they share their communication with E.
- It is computationally intensive. As a result, it is vulnerable to a clogging attack, in which an opponent requests a high number of keys. The victim spends considerable computing resources doing useless modular exponentiation rather than real work.

Oakley is designed to retain the advantages of Diffie-Hellman while countering its weaknesses.

Features of Oakley

The Oakley algorithm is characterized by five important features:

1. It employs a mechanism known as cookies to thwart clogging attacks.

2.

It enables the two parties to negotiate a *group*; this, in essence, specifies the global parameters of the Diffie-Hellman key exchange.

3.

It uses nonces to ensure against replay attacks.

4.

It enables the exchange of Diffie-Hellman public key values.

5.

It authenticates the Diffie-Hellman exchange to thwart man-in-the-middle attacks.

We have already discussed Diffie-Hellman. Let us look the remainder of these elements in turn. First, consider the problem of clogging attacks. In this attack, an opponent forges the source address of a legitimate user and sends a public Diffie-Hellman key to the victim. The victim then performs a modular exponentiation to compute the secret key. Repeated messages of this type can *clog* the victim's system with useless work. The **cookie exchange** requires that each side send a pseudorandom number, the cookie, in the initial message, which the other side acknowledges. This acknowledgment must be repeated in the first message of the Diffie-Hellman key exchange. If the source address was forged, the opponent gets no answer. Thus, an opponent can only force a user to generate acknowledgments and not to perform the Diffie-Hellman calculation.

ISAKMP mandates that cookie generation satisfy three basic requirements:

1.

The cookie must depend on the specific parties. This prevents an attacker from obtaining a cookie using a real IP address and UDP port and then using it to swamp the victim with requests from randomly chosen IP addresses or ports.

2.

It must not be possible for anyone other than the issuing entity to generate cookies that will be accepted by that entity. This implies that the issuing entity will use local secret information in the generation and subsequent verification of a cookie. It must not be possible to deduce this secret information from any particular cookie. The point of this requirement is that the issuing entity need not save copies of its cookies, which are then more vulnerable to discovery, but can verify an incoming cookie acknowledgment when it needs to.

3.

The cookie generation and verification methods must be fast to thwart attacks intended to sabotage processor resources.

The recommended method for creating the cookie is to perform a fast hash (e.g., MD5) over the IP Source and Destination addresses, the UDP Source and Destination ports, and a locally generated secret

value.

Oakley supports the use of different **groups** for the Diffie-Hellman key exchange. Each group includes the definition of the two global parameters and the identity of the algorithm. The current specification includes the following groups:

- Modular exponentiation with a 768-bit modulus

  $q = 2^{768} - 2^{704} - 1 + 2^{64} \times ([2^{638} \times \pi] + 149686)$

  $\alpha = 2$
- Modular exponentiation with a 1024-bit modulus

  $q = 2^{1024} - 2^{960} - 1 + 2^{64} \times ([2^{894} \times \pi] + 129093)$

  $\alpha = 2$
- Modular exponentiation with a 1536-bit modulus

  Parameters to be determined
- Elliptic curve group over 2^{155}

  Generator (hexadecimal):X = 7B, Y = 1C8

  Elliptic curve parameters (hexadecimal):A = 0, Y = 7338F
- Elliptic curve group over 2^{185}

  Generator (hexadecimal):X = 18, Y = D

  Elliptic curve parameters (hexadecimal):A = 0, Y = 1EE9

The first three groups are the classic Diffie-Hellman algorithm using modular exponentiation. The last two groups use the elliptic curve analog to Diffie-Hellman, which was described in Chapter 10.

Oakley employs **nonces** to ensure against replay attacks. Each nonce is a locally generated pseudorandom number. Nonces appear in responses and are encrypted during certain portions of the exchange to secure their use.

Three different **authentication** methods can be used with Oakley:

- **Digital signatures:** The exchange is authenticated by signing a mutually obtainable hash; each party encrypts the hash with its private key. The hash is generated over important parameters, such as user IDs and nonces.
- **Public-key encryption**: The exchange is authenticated by encrypting parameters such as IDs and nonces with the sender's private key.
- **Symmetric-key encryption:** A key derived by some out-of-band mechanism can be used to authenticate the exchange by symmetric encryption of exchange parameters.

Oakley Exchange Example

The Oakley specification includes a number of examples of exchanges that are allowable under the protocol. To give a flavor of Oakley, we present one example, called aggressive key exchange in the specification, so called because only three messages are exchanged.

Figure 16.11 shows the aggressive key exchange protocol. In the first step, the initiator (I) transmits a cookie, the group to be used, and I's public Diffie-Hellman key for this exchange. I also indicates the offered public-key encryption, hash, and authentication algorithms to be used in this exchange. Also included in this message are the identifiers of I and the responder (R) and I's nonce for this exchange. Finally, I appends a signature using I's private key that signs the two identifiers, the nonce, the group, the Diffie-Hellman public key, and the offered algorithms.

Figure 16.11. Example of Aggressive Oakley Key Exchange

[View full size image]

$$\mathbf{I \rightarrow R:}\ CKY_I, OK\_KEYX, GRP, g^x, EHAO, NIDP, ID_I, ID_R, N_I, S_{KI}[ID_I \| ID_R \| N_I \| GRP \| g^x \| EHAO]$$

$$\mathbf{R \rightarrow I:}\ CKY_R, CKY_I, OK\_KEYX, GRP, g^y, EHAS, NIDP, ID_R, ID_I, N_R, N_I, S_{KR}[ID_R \| ID_I \| N_R \| N_I \| GRP \| g^y \| g^x \| EHAS]$$

$$\mathbf{I \rightarrow R:}\ CKY_I, CKY_R, OK\_KEYX, GRP, g^x, EHAS, NIDP, ID_I, ID_R, N_I, N_R, S_{KI}[ID_I \| ID_R \| N_I \| N_R \| GRP \| g^x \| g^y \| EHAS]$$

Notation:

I	=	Initiator
R	=	Responder
CKY_I, CKY_R	=	Initiator, responder cookies
OK_KEYX	=	Key exchange message type
GRP	=	Name of Diffie-Hellman group for this exchange
g^x, g^y	=	Public key of initiator, responder; g^{xy} = session key from this exchange
EHAO, EHAS	=	Encryption, hash authentication functions, offered and selected
NIDP	=	Indicates encryption is not used for remainder of this message
ID_I, ID_R	=	Identifier for initiator, responder
N_I, N_R	=	Random nonce supplied by initiator, responder for this exchange
$S_{KI}[X], S_{KR}[X]$	=	Indicates the signature over X using the private key (signing key) of intiator, responder

When R receives the message, R verifies the signature using I's public signing key. R acknowledges the message by echoing back I's cookie, identifier, and nonce, as well as the group. R also includes in the message a cookie, R's Diffie-Hellman public key, the selected algorithms (which must be among the offered algorithms), R's identifier, and R's nonce for this exchange. Finally, R appends a signature using R's private key that signs the two identifiers, the two nonces, the group, the two Diffie-Hellman public keys, and the selected algorithms.

When I receives the second message, I verifies the signature using R's public key. The nonce values in the message assure that this is not a replay of an old message. To complete the exchange, I must send a message back to R to verify that I has received R's public key.

ISAKMP

ISAKMP defines procedures and packet formats to establish, negotiate, modify, and delete security associations. As part of SA establishment, ISAKMP defines payloads for exchanging key generation and authentication data. These payload formats provide a consistent framework independent of the specific key exchange protocol, encryption algorithm, and authentication mechanism.

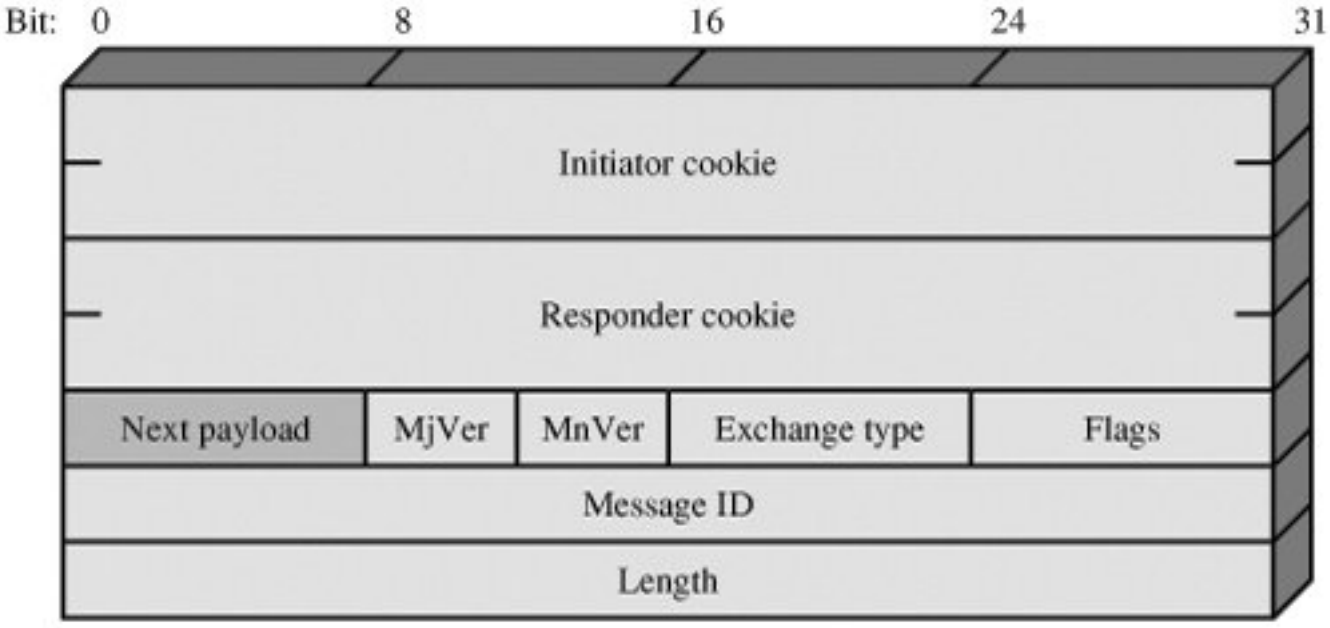
ISAKMP Header Format

An ISAKMP message consists of an ISAKMP header followed by one or more payloads. All of this is carried in a transport protocol. The specification dictates that implementations must support the use of UDP for the transport protocol.

Figure 16.12a shows the header format for an ISAKMP message. It consists of the following fields:

- **Initiator Cookie (64 bits):** Cookie of entity that initiated SA establishment, SA notification, or SA deletion.
- **Responder Cookie (64 bits):** Cookie of responding entity; null in first message from initiator.
- **Next Payload (8 bits):** Indicates the type of the first payload in the message; payloads are discussed in the next subsection.
- **Major Version (4 bits):** Indicates major version of ISAKMP in use.
- **Minor Version (4 bits):** Indicates minor version in use.
- **Exchange Type (8 bits):** Indicates the type of exchange; these are discussed later in this section.
- **Flags (8 bits):** Indicates specific options set for this ISAKMP exchange. Two bits so far defined: The Encryption bit is set if all payloads following the header are encrypted using the encryption algorithm for this SA. The Commit bit is used to ensure that encrypted material is not received prior to completion of SA establishment.
- **Message ID (32 bits):** Unique ID for this message.
- **Length (32 bits):** Length of total message (header plus all payloads) in octets.

Figure 16.12. ISAKMP Formats



(a) ISAKMP header

(b) Generic payload header

ISAKMP Payload Types

All ISAKMP payloads begin with the same generic payload header shown in Figure 16.12b. The Next Payload field has a value of 0 if this is the last payload in the message; otherwise its value is the type of the next payload. The Payload Length field indicates the length in octets of this payload, including the generic payload header.

Table 16.3 summarizes the payload types defined for ISAKMP, and lists the fields, or parameters, that are part of each payload. The **SA payload** is used to begin the establishment of an SA. In this payload, the Domain of Interpretation parameter identifies the DOI under which negotiation is taking place. The IPSec DOI is one example, but ISAKMP can be used in other contexts. The Situation parameter defines the security policy for this negotiation; in essence, the levels of security required for encryption and confidentiality are specified (e.g., sensitivity level, security compartment).

Table 16.3. ISAKMP Payload Types

(This item is displayed on page 513 in the print version)

Type	Parameters	Description

Security Association (SA)	Domain of Interpretation, Situation	Used to negotiate security attributes and indicate the DOI and Situation under which negotiation is taking place.
Proposal (P)	Proposal #, Protocol-ID, SPI Size, # of Transforms, SPI	Used during SA negotiation; indicates protocol to be used and number of transforms.
Transform (T)	Transform #, Transform-ID, SA Attributes	Used during SA negotiation; indicates transform and related SA attributes.
Key Exchange (KE)	Key Exchange Data	Supports a variety of key exchange techniques.
Identification (ID)	ID Type, ID Data	Used to exchange identification information.
Certificate (CERT)	Cert Encoding, Certificate Data	Used to transport certificates and other certificate- related information.
Certificate Request (CR)	# Cert Types, Certificate Types, # Cert Auths, Certificate Authorities	Used to request certificates; indicates the types of certificates requested and the acceptable certificate authorities.
Hash (HASH)	Hash Data	Contains data generated by a hash function.
Signature (SIG)	Signature Data	Contains data generated by a digital signature function.
Nonce (NONCE)	Nonce Data	Contains a nonce.
Notification (N)	DOI, Protocol-ID, SPI Size, Notify Message Type, SPI, Notification Data	Used to transmit notification data, such as an error condition.
Delete (D)	DOI, Protocol-ID, SPI Size, #of SPIs, SPI (one or more)	Indicates an SA that is no longer valid.

The **Proposal payload** contains information used during SA negotiation. The payload indicates the protocol for this SA (ESP or AH) for which services and mechanisms are being negotiated. The payload also includes the sending entity's SPI and the number of transforms. Each transform is contained in a transform payload. The use of multiple transform payloads enables the initiator to offer several possibilities, of which the responder must choose one or reject the offer.

The **Transform payload** defines a security transform to be used to secure the communications channel for the designated protocol. The Transform # parameter serves to identify this particular payload so that the responder may use it to indicate acceptance of this transform. The Transform-ID and Attributes fields identify a specific transform (e.g., 3DES for ESP, HMAC-SHA-1-96 for AH) with its associated attributes (e.g., hash length).

The **Key Exchange payload** can be used for a variety of key exchange techniques, including Oakley, Diffie-Hellman, and the RSA-based key exchange used by PGP. The Key Exchange data field contains the data required to generate a session key and is dependent on the key exchange algorithm used.

The **Identification payload** is used to determine the identity of communicating peers and may be used

for determining authenticity of information. Typically the ID Data field will contain an IPv4 or IPv6 address.

The **Certificate payload** transfers a public-key certificate. The Certificate Encoding field indicates the type of certificate or certificate-related information, which may include the following:

- PKCS #7 wrapped X.509 certificate
- PGP certificate
- DNS signed key
- X.509 certificatesignature
- X.509 certificatekey exchange
- Kerberos tokens
- Certificate Revocation List (CRL)
- Authority Revocation List (ARL)
- SPKI certificate

At any point in an ISAKMP exchange, the sender may include a **Certificate Request** payload to request the certificate of the other communicating entity. The payload may list more than one certificate type that is acceptable and more than one certificate authority that is acceptable.

The **Hash payload** contains data generated by a hash function over some part of the message and/or ISAKMP state. This payload may be used to verify the integrity of the data in a message or to authenticate negotiating entities.

The **Signature payload** contains data generated by a digital signature function over some part of the message and/or ISAKMP state. This payload is used to verify the integrity of the data in a message and may be used for nonrepudiation services.

The **Nonce payload** contains random data used to guarantee liveness during an exchange and protect against replay attacks.

The **Notification payload** contains either error or status information associated with this SA or this SA negotiation. The following ISAKMP error messages have been defined:

Invalid Payload Type	Invalid Protocol ID	Invalid Cert Encoding
DOI Not Supported	Invalid SPI	Invalid Certificate
Situation Not Supported	Invalid Transform ID	Bad Cert Request Syntax
Invalid Cookie	Attributes Not Supported	Invalid Cert Authority
Invalid Major Version	No Proposal Chosen	Invalid Hash Information
Invalid Minor Version	Bad Proposal Syntax	Authentication Failed
Invalid Exchange Type	Payload Malformed	Invalid Signature
Invalid Flags	Invalid Key Information	Address Notification
Invalid Message ID		

The only ISAKMP status message so far defined is Connected. In addition to these ISAKMP notifications, DOI-specific notifications are used. For IPSec, the following additional status messages are defined:

- **Responder-Lifetime:** Communicates the SA lifetime chosen by the responder.
- **Replay-Status:** Used for positive confirmation of the responder's election of whether or not the responder will perform anti-replay detection.
- **Initial-Contact:** Informs the other side that this is the first SA being established with the remote system. The receiver of this notification might then delete any existing SA's it has for the sending system under the assumption that the sending system has rebooted and no longer has access to those SAs.

The **Delete payload** indicates one or more SAs that the sender has deleted from its database and that therefore are no longer valid.

ISAKMP Exchanges

ISAKMP provides a framework for message exchange, with the payload types serving as the building blocks. The specification identifies five default exchange types that should be supported; these are summarized in Table 16.4. In the table, SA refers to an SA payload with associated Protocol and Transform payloads.

Table 16.4. ISAKMP Exchange Types

Exchange	Note
(a) Base Exchange	
(1) **I** $\rightarrow$ **R**: SA; NONCE	Begin ISAKMP-SA negotiation
(2) **R** $\rightarrow$ **E**: SA; NONCE	Basic SA agreed upon
(3) **I** $\rightarrow$ **R**: KE; ID_I AUTH	Key generated; Initiator identity verified by responder
(4) **R** $\rightarrow$ **E**: KE; ID_R AUTH	Responder identity verified by initiator; Key generated; SA established
(b) Identity Protection Exchange	
(1) **I** $\rightarrow$ **R**: SA	Begin ISAKMP-SA negotiation
(2) **R** $\rightarrow$ **E**: SA	Basic SA agreed upon
(3) **I** $\rightarrow$ **R**: KE; NONCE	Key generated
(4) **R** $\rightarrow$ **E**: KE; NONCE	Key generated
(5)* **I** $\rightarrow$ **R**: ID_I; AUTH	Initiator identity verified by responder

(6)*R → E: ID_R; AUTH	Responder identity verified by initiator; SA established
(c) Authentication Only Exchange	
(1)I → R: SA; NONCE	Begin ISAKMP-SA negotiation
(2)R → E: SA; NONCE; ID_R; AUTH	Basic SA agreed upon; Responder identity verified by initiator
(3)I → R: ID_I; AUTH	Initiator identity verified by responder; SA established
(d) Aggressive Exchange	
(1)I → R: SA; KE; NONCE; ID_I;	Begin ISAKMP-SA negotiation and key exchange
(2)R → E: SA; KE; NONCE; ID_R; AUTH	Initiator identity verified by responder; Key generated; Basic SA agreed upon
(3)*I → R: AUTH	Responder identity verified by initiator; SA established
(e) Informational Exchange	
(1)*I → R: N/D	Error or status notification, or deletion
Notation: I = initiator R = responder * = signifies payload encryption after the ISAKMP header AUTH = authentication mechanism used	

The Base Exchange allows key exchange and authentication material to be transmitted together. This minimizes the number of exchanges at the expense of not providing identity protection. The first two messages provide cookies and establish an SA with agreed protocol and transforms; both sides use a nonce to ensure against replay attacks. The last two messages exchange the key material and user IDs, with an authentication mechanism used to authenticate keys, identities, and the nonces from the first two messages.

The **Identity Protection Exchange** expands the Base Exchange to protect the users' identities. The first two messages establish the SA. The next two messages perform key exchange, with nonces for replay protection. Once the session key has been computed, the two parties exchange encrypted messages that contain authentication information, such as digital signatures and optionally certificates validating the public keys.

The **Authentication Only Exchange** is used to perform mutual authentication, without a key

exchange. The first two messages establish the SA. In addition, the responder uses the second message to convey its ID and uses authentication to protect the message. The initiator sends the third message to transmit its authenticated ID.

The **Aggressive Exchange** minimizes the number of exchanges at the expense of not providing identity protection. In the first message, the initiator proposes an SA with associated offered protocol and transform options. The initiator also begins the key exchange and provides its ID. In the second message, the responder indicates its acceptance of the SA with a particular protocol and transform, completes the key exchange, and authenticates the transmitted information. In the third message, the initiator transmits an authentication result that covers the previous information, encrypted using the shared secret session key.

The **Informational Exchange** is used for one-way transmittal of information for SA management.

PREV NEXT

16.7. Recommended Reading and Web Site

IPv6 and IPv4 are covered in more detail in [STAL04]. [CHEN98] provides a good discussion of an IPSec design. [FRAN01] and [DORA99] are more comprehensive treatments of IPSec.

CHEN98 Cheng, P., et al. "A Security Architecture for the Internet Protocol." *IBM Systems Journal*, Number 1, 1998.

DORA03 Doraswamy, N., and Harkins, D. *IPSec*. Upper Saddle River, NJ: Prentice Hall, 2003.

FRAN01 Frankel, S. *Demystifying the IPSec Puzzle*. Boston: Artech House, 2001.

STAL04 Stallings, W. *Computer Networking with Internet Protocols and Technology*. Upper Saddle River, NJ: Prentice Hall, 2004.

Recommended Web Site

- **NIST IPSEC Project:** Contains papers, presentations, and reference implementations

16.8. Key Terms, Review Questions, and Problems

Key Terms

anti-replay service

authentication header (AH)

encapsulating security payload (ESP)

Internet Security Association and Key Management Protocol (ISAKMP)

IP Security (IPSec)

IPv4

IPv6

Oakley key determination protocol

replay attack

security association (SA)

transport mode

tunnel mode

Review Questions

16.1 Give examples of applications of IPSec.

16.2 What services are provided by IPSec?

16.3 What parameters identify an SA and what parameters characterize the nature of a particular SA?

16.4 What is the difference between transport mode and tunnel mode?

16.5 What is a replay attack?

16.6 Why does ESP include a padding field?

16.7 What are the basic approaches to bundling SAs?

16.8 What are the roles of the Oakley key determination protocol and ISAKMP in IPSec?

Problems

16.1 In discussing AH processing, it was mentioned that not all of the fields in an IP header are included in MAC calculation.

a.

For each of the fields in the IPv4 header, indicate whether the field is immutable, mutable but predictable, or mutable (zeroed prior to ICV calculation).

b.

Do the same for the IPv6 header.

c.

Do the same for the IPv6 extension headers.

In each case, justify your decision for each field.

16.2 When tunnel mode is used, a new outer IP header is constructed. For both IPv4 and IPv6, indicate the relationship of each outer IP header field and each extension header in the outer packet to the corresponding field or extension header of the inner IP packet. That is, indicate which outer values are derived from inner values and which are constructed independently of the inner values.

16.3 End-to-end authentication and encryption are desired between two hosts. Draw figures similar to Figures 16.6 and 16.9 that show

a.

Transport adjacency, with encryption applied before authentication

b.

A transport SA bundled inside a tunnel SA, with encryption applied before authentication

c.

A transport SA bundled inside a tunnel SA, with authentication applied before encryption

16.4 The IPSec architecture document states that when two transport mode SA's are bundled to allow both AH and ESP protocols on the same end-to-end flow, only one ordering of security protocols seems appropriate: performing the ESP protocol before performing the AH protocol. Why is this approach recommended rather than authentication before encryption?

16.5 **a.**

Which of the ISAKMP Exchange Types (Table 16.4) corresponds to the aggressive Oakley key exchange (Figure 16.11)?

b.

For the Oakley aggressive key exchange, indicate which parameters in each message go in which ISAKMP payload types.

Appendix 16A Internetworking and Internet Protocols

This appendix provides an overview of Internet protocols. We begin with a summary of the role of an internet protocol in providing internetworking. Then the two main internet protocols, IPv4 and IPv6, are introduced.

The Role of an Internet Protocol

An internet protocol (IP) provides the functionality for interconnecting end systems across multiple networks. For this purpose, IP is implemented in each end system and in routers, which are devices that provide connection between networks. Higher-level data at a source end system are encapsulated in an IP protocol data unit (PDU) for transmission. This PDU is then passed through one or more networks and connecting routers to reach the destination end system.

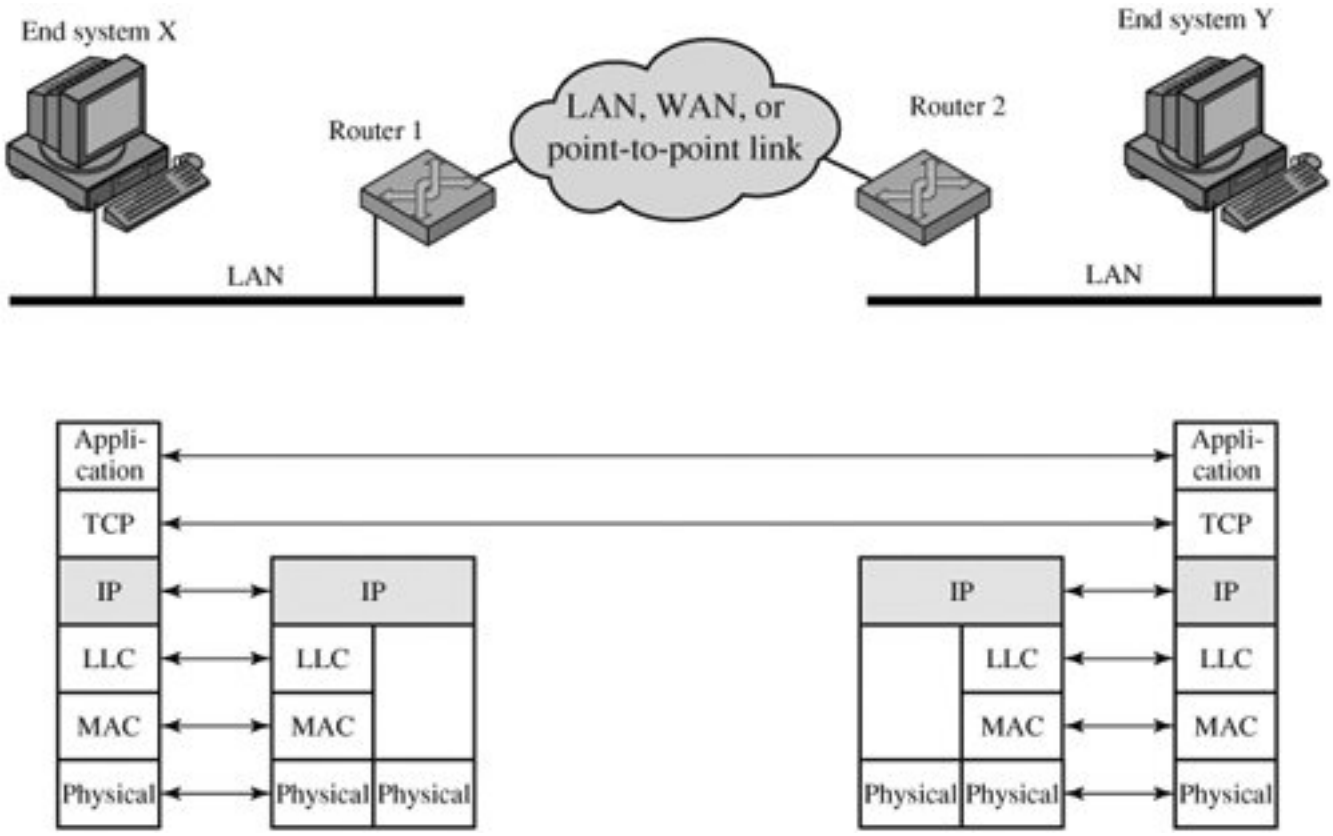
The router must be able to cope with a variety of differences among networks, including

- **Addressing schemes:** The networks may use different schemes for assigning addresses to devices. For example, an IEEE 802 LAN uses either 16-bit or 48-bit binary addresses for each attached device; an X.25 public packet-switching network uses 12-digit decimal addresses (encoded as 4 bits per digit for a 48-bit address). Some form of global network addressing must be provided, as well as a directory service.
- **Maximum packet sizes:** Packets from one network may have to be broken into smaller pieces to be transmitted on another network, a process known as **fragmentation**. For example, Ethernet imposes a maximum packet size of 1500 bytes; a maximum packet size of 1000 bytes is common on X.25 networks. A packet that is transmitted on an Ethernet system and picked up by a router for retransmission on an X.25 network may have to fragment the incoming packet into two smaller ones.
- **Interfaces:** The hardware and software interfaces to various networks differ. The concept of a router must be independent of these differences.
- **Reliability:** Various network services may provide anything from a reliable end-to-end virtual circuit to an unreliable service. The operation of the routers should not depend on an assumption of network reliability.

The operation of the router, as Figure 16.13 indicates, depends on an internet protocol. In this example, the Internet Protocol (IP) of the TCP/IP protocol suite performs that function. IP must be implemented in all end systems on all networks as well as on the routers. In addition, each end system must have compatible protocols above IP to communicate successfully. The intermediate routers need only have up through IP.

Figure 16.13. Configuration for TCP/IP Example

[View full size image]



Consider the transfer of a block of data from end system *X* to end system *Y* in Figure 16.13. The IP layer at *X* receives blocks of data to be sent to *Y* from TCP in *X*. The IP layer attaches a header that specifies the global internet address of *Y*. That address is in two parts: network identifier and end system identifier. Let us refer to this block as the IP packet. Next, IP recognizes that the destination (*Y*) is on another subnetwork. So the first step is to send the packet to a router, in this case router 1. To accomplish this, IP hands its data unit down to LLC with the appropriate addressing information. LLC creates an LLC PDU, which is handed down to the MAC layer. The MAC layer constructs a MAC packet whose header contains the address of router 1.

Next, the packet travels through LAN to router 1. The router removes the packet and LLC headers and trailers and analyzes the IP header to determine the ultimate destination of the data, in this case *Y*. The router must now make a routing decision. There are two possibilities:

1. The destination end system *Y* is connected directly to one of the subnetworks to which the router is attached.
2. To reach the destination, one or more additional routers must be traversed.

In this example, the packet must be routed through router 2 before reaching the destination. So router 1 passes the IP packet to router 2 via the intermediate network. For this purpose, the protocols of that network are used. For example, if the intermediate network is an X.25 network, the IP data unit is wrapped in an X.25 packet with appropriate addressing information to reach router 2. When this packet arrives at router 2, the packet header is stripped off. The router determines that this IP packet is destined for *Y*, which is connected directly to a subnetwork to which the router is attached. The router therefore creates a packet with a destination address of *Y* and sends it out onto the LAN. The data finally arrive at *Y*, where the packet, LLC, and internet headers and trailers can be stripped off.

This service offered by IP is an unreliable one. That is, IP does not guarantee that all data will be delivered or that the data that are delivered will arrive in the proper order. It is the responsibility of the next higher layer, in this case TCP, to recover from any errors that occur. This approach provides for a great deal of flexibility. Because delivery is not guaranteed, there is no particular reliability requirement on any of the subnetworks. Thus, the protocol will work with any combination of subnetwork types. Because the sequence of delivery is not guaranteed, successive packets can follow different paths through the internet. This allows the protocol to react to congestion and failure in the internet by changing routes.

IPv4

For decades, the keystone of the TCP/IP protocol architecture has been the Internet Protocol (IP) version 4. Figure 16.14a shows the IP header format, which is a minimum of 20 octets, or 160 bits. The fields are as follows:

- **Version (4 bits):** Indicates version number, to allow evolution of the protocol; the value is 4.
- **Internet Header Length (IHL) (4 bits):** Length of header in 32-bit words. The minimum value is five, for a minimum header length of 20 octets.
- **DS/ECN (8 bits):** Prior to the introduction of differentiated services, this field was referred to as the **Type of Service** field and specified reliability, precedence, delay, and throughput parameters. This interpretation has now been superseded. The first 6 bits of the TOS field are now referred to as the DS (Differentiated Services) field. The remaining 2 bits are reserved for an ECN (Explicit Congestion Notification) field.
- **Total Length (16 bits):** Total IP packet length, in octets.
- **Identification (16 bits):** A sequence number that, together with the source address, destination address, and user protocol, is intended to identify a packet uniquely. Thus, this number should be unique for the packet's source address, destination address, and user protocol for the time during which the packet will remain in the internet.
- **Flags (3 bits):** Only two of the bits are currently defined. When a packet is fragmented, the More bit indicates whether this is the last fragment in the original packet. The Don't Fragment bit prohibits fragmentation when set. This bit may be useful if it is known that the destination does not have the capability to reassemble fragments. However, if this bit is set, the packet will be discarded if it exceeds the maximum size of an en route subnetwork. Therefore, if the bit is set, it may be advisable to use source routing to avoid subnetworks with small maximum packet size.

- **Fragment Offset (13 bits):** Indicates where in the original packet this fragment belongs, measured in 64-bit units. This implies that fragments other than the last fragment must contain a data field that is a multiple of 64 bits in length.
- **Time to Live (8 bits):** Specifies how long, in seconds, a packet is allowed to remain in the internet. Every router that processes a packet must decrease the TTL by at least one, so the TTL is somewhat similar to a hop count.

- **Protocol (8 bits):** Indicates the next higher level protocol, which is to receive the data field at the destination; thus, this field identifies the type of the next header in the packet after the IP header.
- **Header Checksum (16 bits):** An error-detecting code applied to the header only. Because some header fields may change during transit (e.g., time to live, segmentation-related fields), this is reverified and recomputed at each router. The checksum field is the 16-bit one's

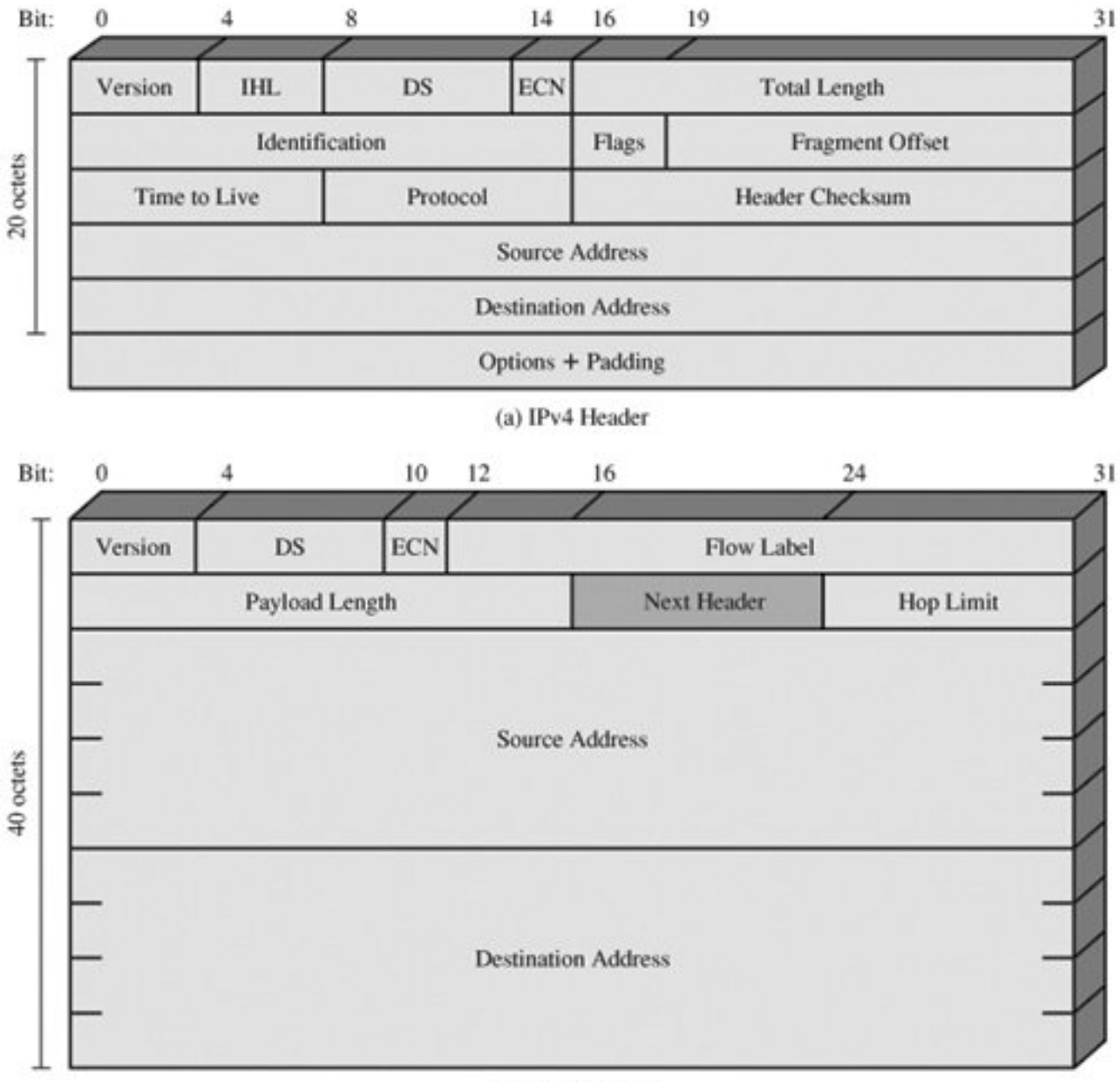
complement addition of all 16-bit words in the header. For purposes of computation, the checksum field is itself initialized to a value of zero.

- **Source Address (32 bits):** Coded to allow a variable allocation of bits to specify the network and the end system attached to the specified network (7 and 24 bits, 14 and 16 bits, or 21 and 8 bits).
- **Destination Address (32 bits):** Same characteristics as source address.
- **Options (variable):** Encodes the options requested by the sending user; these may include security label, source routing, record routing, and timestamping.
- **Padding (variable):** Used to ensure that the packet header is a multiple of 32 bits in length.

Figure 16.14. IP Headers

(This item is displayed on page 521 in the print version)

[View full size image]



DS = Differentiated services field
ECN = Explicit congestion notification field

Note: The 8-bit DS/ECN fields were formerly known as the Type of Service field in the IPv4 header and the Traffic Class field in the IPv6 header.

IPv6

In 1995, the Internet Engineering Task Force (IETF), which develops protocol standards for the Internet, issued a specification for a next-generation IP, known then as IPng. This specification was turned into a standard in 1996 known as IPv6. IPv6 provides a number of functional enhancements over the existing IP (known as IPv4), designed to accommodate the higher speeds of today's networks and the mix of data streams, including graphic and video, that are becoming more prevalent. But the driving force behind the development of the new protocol was the need for more addresses. IPv4 uses a 32-bit address to specify a source or destination. With the explosive growth of the Internet and of private networks attached to the Internet, this address length became insufficient to accommodate all systems needing addresses. As Figure 16.14b shows, IPv6 includes 128-bit source and destination address fields. Ultimately, all installations using TCP/IP are expected to migrate from the current IP to IPv6, but this process will take many years, if not decades.

IPv6 Header

The IPv6 header has a fixed length of 40 octets, consisting of the following fields (Figure 16.14b):

- **Version (4 bits):** Internet Protocol version number; the value is 6.
- **DS/ECN (8 bits):** Prior to the introduction of differentiated services, this field was referred to as the **Traffic Class** field and was reserved for use by originating nodes and/or forwarding routers to identify and distinguish between different classes or priorities of IPv6 packets. The first six bits of the Traffic Class field are now referred to as the DS (Differentiated Services) field. The remaining 2 bits are reserved for an ECN (Explicit Congestion Notification) field.
- **Flow Label (20 bits):** May be used by a host to label those packets for which it is requesting special handling by routers within a network. Flow labeling may assist resource reservation and real-time traffic processing.

- **Payload Length (16 bits):** Length of the remainder of the IPv6 packet following the header, in octets. In other words, this is the total length of all of the extension headers plus the transport-level PDU.
- **Next Header (8 bits):** Identifies the type of header immediately following the IPv6 header; this will either be an IPv6 extension header or a higher-layer header, such as TCP or UDP.
- **Hop Limit (8 bits):** The remaining number of allowable hops for this packet. The hop limit is set to some desired maximum value by the source and decremented by 1 by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero.
- **Source Address (128 bits):** The address of the originator of the packet.
- **Destination Address (128 bits):** The address of the intended recipient of the packet. This may not in fact be the intended ultimate destination if a Routing extension header is present, as explained later.

Although the IPv6 header is longer than the mandatory portion of the IPv4 header (40 octets versus 20 octets), it contains fewer fields (8 versus 12). Thus, routers have less processing to do per header, which should speed up routing.

IPv6 Extension Headers

An IPv6 packet includes the IPv6 header, just discussed, and zero or more extension headers. Outside of IPSec, the following extension headers have been defined:

- **Hop-by-Hop Options Header:** Defines special options that require hop-by-hop processing
- **Routing Header:** Provides extended routing, similar to IPv4 source routing

- **Fragment Header:** Contains fragmentation and reassembly information
- **Authentication Header:** Provides packet integrity and authentication
- **Encapsulating Security Payload Header:** Provides privacy
- **Destination Options Header:** Contains optional information to be examined by the destination node

The IPv6 standard recommends that, when multiple extension headers are used, the IPv6 headers appear in the following order:

1. IPv6 header: Mandatory, must always appear first
2. Hop-by-Hop Options header
3. Destination Options header: For options to be processed by the first destination that appears in the IPv6 Destination Address field plus subsequent destinations listed in the Routing header
4. Routing header
5. Fragment header
6. Authentication header
7. Encapsulating Security Payload header

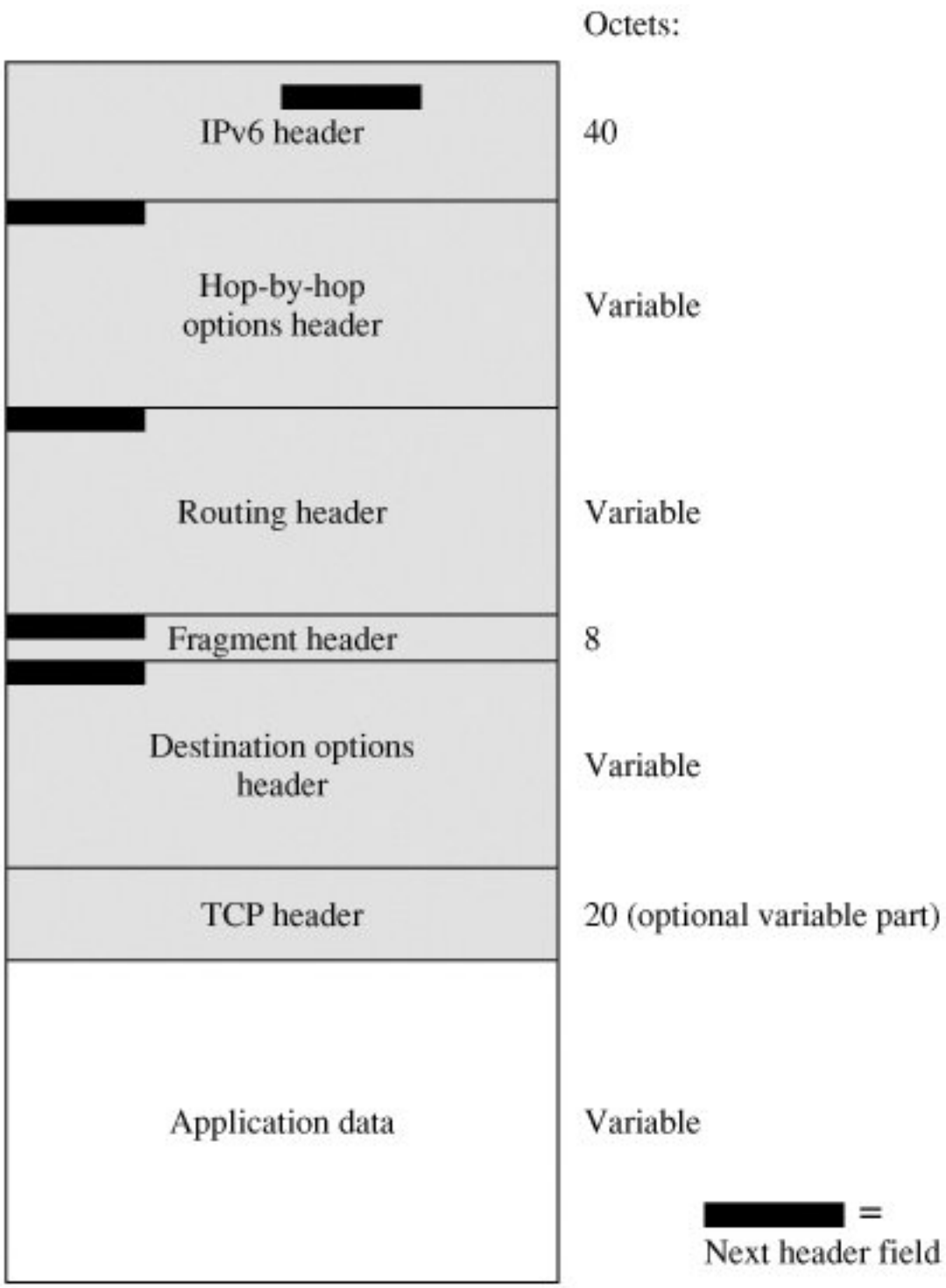
---

8. Destination Options header: For options to be processed only by the final destination of the packet

Figure 16.15 shows an example of an IPv6 packet that includes an instance of each nonsecurity header. Note that the IPv6 header and each extension header include a Next Header field. This field identifies the type of the immediately following header. If the next header is an extension header, then this field contains the type identifier of that header. Otherwise, this field contains the protocol identifier of the upper-layer protocol using IPv6 (typically a transport-level protocol), using the same values as the IPv4 Protocol field. In the figure, the upper-layer protocol is TCP, so the upper-layer data carried by the IPv6

packet consist of a TCP header followed by a block of application data.

Figure 16.15. Ipv6 Packet with Extension Headers (containing a TCP segment)



The **Hop-by-Hop Options header** carries optional information that, if present, must be examined by every router along the path. The header consists of the following fields:

- **Next Header (8 bits):** Identifies the type of header immediately following this header.

- **Header Extension Length (8 bits):** Length of this header in 64-bit units, not including the first 64 bits.
- **Options:** Contains one or more options. Each option consists of three subfields: a tag, indicating

the option type; a length, and a value.

Only one option has so far been defined: the Jumbo Payload option, used to send IPv6 packets with payloads longer than $2^{16} - 1 = 65,535$ octets. The Option Data field of this option is 32 bits long and gives the length of the packet in octets, excluding the IPv6 header. For such packets, the Payload Length field in the IPv6 header must be set to zero, and there must be no Fragment header. With this option, IPv6 supports packet sizes up to more than 4 billion octets. This facilitates the transmission of large video packets and enables IPv6 to make the best use of available capacity over any transmission medium.

The **Routing header** contains a list of one or more intermediate nodes to be visited on the way to a packet's destination. All routing headers start with a 32-bit block consisting of four 8-bit fields, followed by routing data specific to a given routing type. The four 8-bit fields are Next Header, Header Extension Length, and

- **Routing Type:** Identifies a particular Routing header variant. If a router does not recognize the Routing Type value, it must discard the packet.
- **Segments Left:** Number of explicitly listed intermediate nodes still to be visited before reaching the final destination.

In addition to this general header definition, the IPv6 specification defines the Type 0 Routing header. When using the Type 0 Routing header, the source node does not place the ultimate destination address in the IPv6 header. Instead, that address is the last address listed in the Routing header, and the IPv6 header contains the destination address of the first desired router on the path. The Routing header will not be examined until the packet reaches the node identified in the IPv6 header. At that point, the IPv6 and Routing header contents are updated and the packet is forwarded. The update consists of placing the next address to be visited in the IPv6 header and decrementing the Segments Left field in the Routing header.

IPv6 requires an IPv6 node to reverse routes in a packet it receives containing a Routing header, to return a packet to the sender.

The **Fragment header** is used by a source when fragmentation is required. In IPv6, fragmentation may only be performed by source nodes, not by routers along a packet's delivery path. To take full advantage of the internetworking environment, a node must perform a path discovery algorithm that enables it to learn the smallest maximum transmission unit (MTU) supported by any subnetwork on the path. In other words, the path discovery algorithm enables a node to learn the MTU of the "bottleneck" subnetwork on the path. With this knowledge, the source node will fragment, as required, for each given destination address. Otherwise the source must limit all packets to 1280 octets, which is the minimum MTU that must be supported by each subnetwork.

In addition to the Next Header field, the fragment header includes the following fields:

- **Fragment Offset (13 bits):** Indicates where in the original packet the payload of this fragment belongs. It is measured in 64-bit units. This implies that fragments (other than the last fragment) must contain a data field that is a multiple of 64 bits long.

- **Res (2 bits):** Reserved for future use.
- **M Flag (1 bit):** 1 = more fragments; 0 = last fragment.
- **Identification (32 bits):** Intended to identify uniquely the original packet. The identifier must be unique for the packet's source address and destination address for the time during which the packet will remain in the internet. All fragments with the same identifier, source address, and

destination address are reassembled to form the original packet.

The **Destination Options header** carries optional information that, if present, is examined only by the packet's destination node. The format of this header is the same as that of the Hop-by-Hop Options header.

PREV NEXT

Chapter 17. Web Security

Use your mentality

Wake up to reality

From the song, "I've Got You under My Skin"by Cole Porter

Key Points

- Secure socket layer (SSL) provides security services between TCP and applications that use TCP. The Internet standard version is called transport layer service (TLS).
- SSL/TLS provides confidentiality using symmetric encryption and message integrity using a message authentication code.
- SSL/TLS includes protocol mechanisms to enable two TCP users to determine the security mechanisms and services they will use.
- Secure electronic transaction (SET) is an open encryption and security specification designed to protect credit card transactions on the Internet.

Virtually all businesses, most government agencies, and many individuals now have Web sites. The number of individuals and companies with Internet access is expanding rapidly and all of these have graphical Web browsers. As a result, businesses are enthusiastic about setting up facilities on the Web for electronic commerce. But the reality is that the Internet and the Web are extremely vulnerable to compromises of various sorts. As businesses wake up to this reality, the demand for secure Web services grows.

The topic of Web security is a broad one and can easily fill a book (several are recommended at the end of this chapter). In this chapter, we begin with a discussion of the general requirements for Web security and then focus on two standardized schemes that are becoming increasingly important as part of Web commerce: SSL/TLS and SET.

17.1. Web Security Considerations

The World Wide Web is fundamentally a client/server application running over the Internet and TCP/IP intranets. As such, the security tools and approaches discussed so far in this book are relevant to the issue of Web security. But, as pointed out in [GARF97], the Web presents new challenges not generally appreciated in the context of computer and network security:

- The Internet is two way. Unlike traditional publishing environments, even electronic publishing systems involving teletext, voice response, or fax-back, the Web is vulnerable to attacks on the Web servers over the Internet.
- The Web is increasingly serving as a highly visible outlet for corporate and product information and as the platform for business transactions. Reputations can be damaged and money can be lost if the Web servers are subverted.

- Although Web browsers are very easy to use, Web servers are relatively easy to configure and manage, and Web content is increasingly easy to develop, the underlying software is extraordinarily complex. This complex software may hide many potential security flaws. The short history of the Web is filled with examples of new and upgraded systems, properly installed, that are vulnerable to a variety of security attacks.
- A Web server can be exploited as a launching pad into the corporation's or agency's entire computer complex. Once the Web server is subverted, an attacker may be able to gain access to data and systems not part of the Web itself but connected to the server at the local site.
- Casual and untrained (in security matters) users are common clients for Web-based services. Such users are not necessarily aware of the security risks that exist and do not have the tools or knowledge to take effective countermeasures.

Web Security Threats

Table 17.1 provides a summary of the types of security threats faced in using the Web. One way to group these threats is in terms of passive and active attacks. Passive attacks include eavesdropping on network traffic between browser and server and gaining access to information on a Web site that is supposed to be restricted. Active attacks include impersonating another user, altering messages in transit between client and server, and altering information on a Web site.

Table 17.1. A Comparison of Threats on the Web [RUBI97]

(This item is displayed on page 530 in the print version)

	Threats	Consequences	Countermeasures

Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerabilty to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping on the Net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server	• Loss of information • Loss of privacy	Encryption, web proxies
Denial of Service	• Killing of user threads • Flooding machine with bogus requests • Filling up disk or memory • Isolating machine by DNS attacks	• Disruptive • Annoying • Prevent user from getting work done	Difficult to prevent
Authentication	• Impersonation of legitimate users • Data forgery	• Misrepresentation of user • Belief that false information is valid	Cryptographic techniques

Another way to classify Web security threats is in terms of the location of the threat: Web server, Web browser, and network traffic between browser and server. Issues of server and browser security fall into the category of computer system security; Part Four of this book addresses the issue of system security in general but is also applicable to Web system security. Issues of traffic security fall into the category of network security and are addressed in this chapter.

Web Traffic Security Approaches

A number of approaches to providing Web security are possible. The various approaches that have been considered are similar in the services they provide and, to some extent, in the mechanisms that they use, but they differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack.

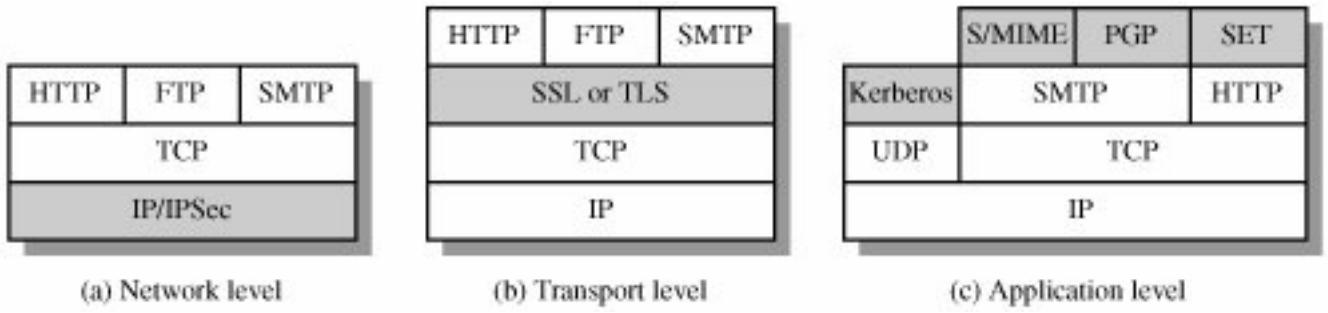
Figure 17.1 illustrates this difference. One way to provide Web security is to use IP Security (Figure

17.1a). The advantage of using IPSec is that it is transparent to end users and applications and provides a general-purpose solution. Further, IPSec includes a filtering capability so that only selected traffic need incur the overhead of IPSec processing.

Figure 17.1. Relative Location of Security Facilities in the TCP/IP Protocol Stack

(This item is displayed on page 531 in the print version)

[View full size image]



Another relatively general-purpose solution is to implement security just above TCP (Figure 17.1b). The foremost example of this approach is the Secure Sockets Layer (SSL) and the follow-on Internet standard known as Transport Layer Security (TLS). At this level, there are two implementation choices. For full generality, SSL (or TLS) could be provided as part of the underlying protocol suite and therefore be transparent to applications. Alternatively, SSL can be embedded in specific packages. For example, Netscape and Microsoft Explorer browsers come equipped with SSL, and most Web servers have implemented the protocol.

Application-specific security services are embedded within the particular application. Figure 17.1c shows examples of this architecture. The advantage of this approach is that the service can be tailored to the specific needs of a given application. In the context of Web security, an important example of this approach is Secure Electronic Transaction (SET).[1]

[1] Figure 17.1c shows SET on top of HTTP; this is a common implementation. In some implementations, SET makes use of TCP directly.

The remainder of this chapter is devoted to a discussion of SSL/TLS and SET.

PREV NEXT

17.2. Secure Socket Layer and Transport Layer Security

Netscape originated SSL. Version 3 of the protocol was designed with public review and input from industry and was published as an Internet draft document. Subsequently, when a consensus was reached to submit the protocol for Internet standardization, the TLS working group was formed within IETF to develop a common standard. This first published version of TLS can be viewed as essentially an SSLv3.1 and is very close to and backward compatible with SSLv3.

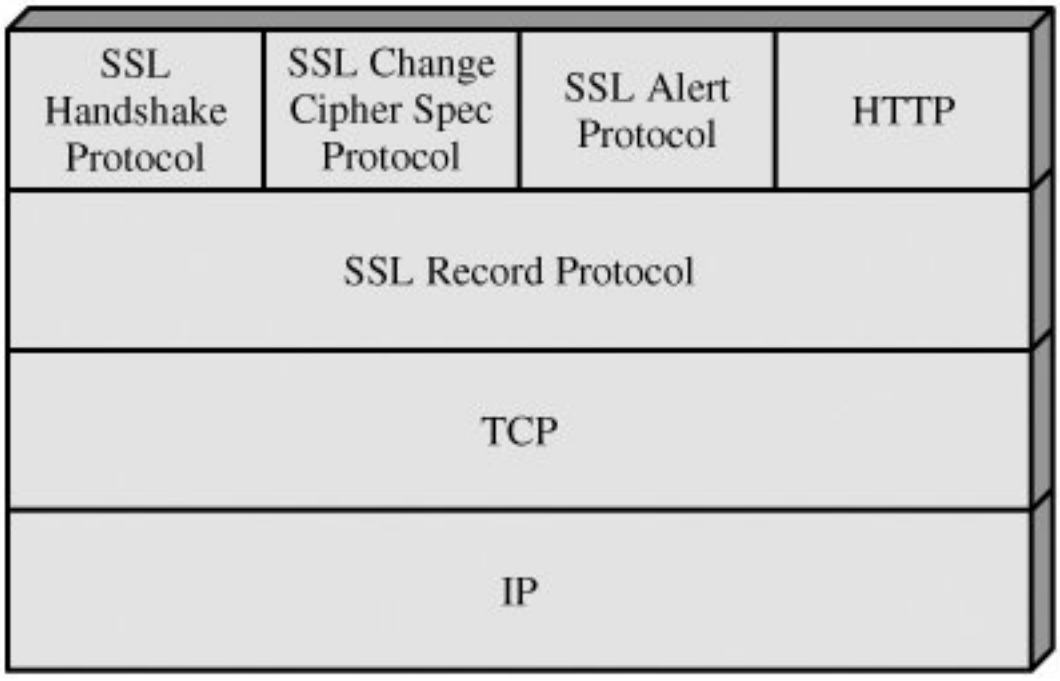
The bulk of this section is devoted to a discussion of SSLv3. At the end of the section, the principal differences between SSLv3 and TLS are described.

SSL Architecture

SSL is designed to make use of TCP to provide a reliable end-to-end secure service. SSL is not a single protocol but rather two layers of protocols, as illustrated in Figure 17.2.

Figure 17.2. SSL Protocol Stack

(This item is displayed on page 532 in the print version)



The SSL Record Protocol provides basic security services to various higher-layer protocols. In particular, the Hypertext Transfer Protocol (HTTP), which provides the transfer service for Web client/server interaction, can operate on top of SSL. Three higher-layer protocols are defined as part of SSL: the Handshake Protocol, The Change Cipher Spec Protocol, and the Alert Protocol. These SSL-specific protocols are used in the management of SSL exchanges and are examined later in this section.

Two important SSL concepts are the SSL session and the SSL connection, which are defined in the specification as follows:

- **Connection:** A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For SSL, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
- **Session:** An SSL session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

Between any pair of parties (applications such as HTTP on client and server), there may be multiple secure connections. In theory, there may also be multiple simultaneous sessions between parties, but this feature is not used in practice.

There are actually a number of states associated with each session. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created. Upon successful conclusion of the Handshake Protocol, the pending states become the current states.

A session state is defined by the following parameters (definitions taken from the SSL specification):

- **Session identifier:** An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- **Peer certificate:** An X509.v3 certificate of the peer. This element of the state may be null.
- **Compression method:** The algorithm used to compress data prior to encryption.
- **Cipher spec:** Specifies the bulk data encryption algorithm (such as null, AES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash_size.

- **Master secret:** 48-byte secret shared between the client and server.
- **Is resumable:** A flag indicating whether the session can be used to initiate new connections.

A connection state is defined by the following parameters:

- **Server and client random:** Byte sequences that are chosen by the server and client for each connection.
- **Server write MAC secret:** The secret key used in MAC operations on data sent by the server.
- **Client write MAC secret:** The secret key used in MAC operations on data sent by the client.
- **Server write key:** The conventional encryption key for data encrypted by the server and decrypted by the client.
- **Client write key:** The conventional encryption key for data encrypted by the client and decrypted by the server.
- **Initialization vectors:** When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter the final ciphertext block from each record is preserved for use as the IV with the following record.
- **Sequence numbers:** Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed $2^{64} 1$.

SSL Record Protocol

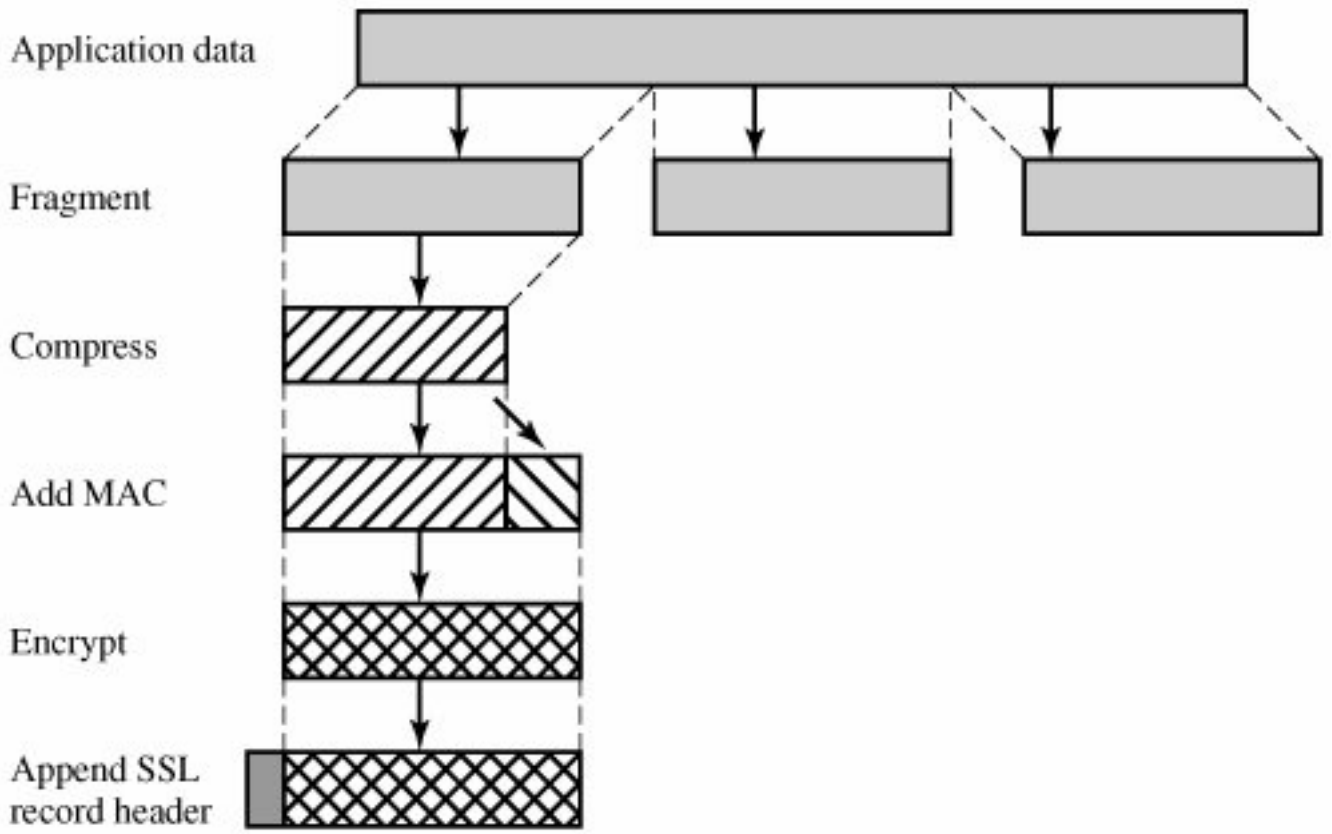
The SSL Record Protocol provides two services for SSL connections:

- **Confidentiality:** The Handshake Protocol defines a shared secret key that is used for conventional encryption of SSL payloads.
- **Message Integrity:** The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).

Figure 17.3 indicates the overall operation of the SSL Record Protocol. The Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment. Received data are decrypted, verified, decompressed, and reassembled and then delivered to higher-level users.

Figure 17.3. SSL Record Protocol Operation

(This item is displayed on page 534 in the print version)



The first step is **fragmentation**. Each upper-layer message is fragmented into blocks of 2^{14} bytes (16384 bytes) or less. Next, **compression** is optionally applied. Compression must be lossless and may not increase the content length by more than 1024 bytes.[2] In SSLv3 (as well as the current version of TLS), no compression algorithm is specified, so the default compression algorithm is null.

[2] Of course, one hopes that compression shrinks rather than expands the data. However, for very short blocks, it is possible, because of formatting conventions, that the compression algorithm will actually provide output that is longer than the input.

The next step in processing is to compute a **message authentication code** over the compressed data. For this purpose, a shared secret key is used. The calculation is defined as

```
hash(MAC_write_secret || pad_2 ||
      hash(MAC_write_secret || pad_1 || seq_num ||
      SSLCompressed.type ||
      SSLCompressed.length || SSLCompressed.fragment))
```

where

||	=	concatenation
MAC_write_secret	=	shared secret key
hash	=	cryptographic hash algorithm; either MD5 or SHA-1
pad_1	=	the byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times (320 bits) for SHA-1
pad_2	=	the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1
seq_num	=	the sequence number for this message
SSLCompressed.type	=	the higher-level protocol used to process this fragment
SSLCompressed.length	=	the length of the compressed fragment
SSLCompressed.fragment	=	the compressed fragment (if compression is not used, the plaintext fragment)

Note that this is very similar to the HMAC algorithm defined in Chapter 12. The difference is that the two pads are concatenated in SSLv3 and are XORed in HMAC. The SSLv3 MAC algorithm is based on the original Internet draft for HMAC, which used concatenation. The final version of HMAC, defined in RFC 2104, uses the XOR.

Next, the compressed message plus the MAC are **encrypted** using symmetric encryption. Encryption may not increase the content length by more than 1024 bytes, so that the total length may not exceed $2^{14} + 2048$. The following encryption algorithms are permitted:

Block Cipher		Stream Cipher	
Algorithm	**Key Size**	**Algorithm**	**Key Size**
AES	128,256	RC4-40	40
IDEA	128	RC4-128	128

RC2-40	40		
DES-40	40		
DES	56		
3DES	168		
Fortezza	80		

Fortezza can be used in a smart card encryption scheme.

For stream encryption, the compressed message plus the MAC are encrypted. Note that the MAC is computed before encryption takes place and that the MAC is then encrypted along with the plaintext or compressed plaintext.

For block encryption, padding may be added after the MAC prior to encryption. The padding is in the form of a number of padding bytes followed by a one-byte indication of the length of the padding. The total amount of padding is the smallest amount such that the total size of the data to be encrypted (plaintext plus MAC plus padding) is a multiple of the cipher's block length. An example is a plaintext (or compressed text if compression is used) of 58 bytes, with a MAC of 20 bytes (using SHA-1), that is encrypted using a block length of 8 bytes (e.g., DES). With the padding.length byte, this yields a total of 79 bytes. To make the total an integer multiple of 8, one byte of padding is added.

The final step of SSL Record Protocol processing is to prepend a header, consisting of the following fields:

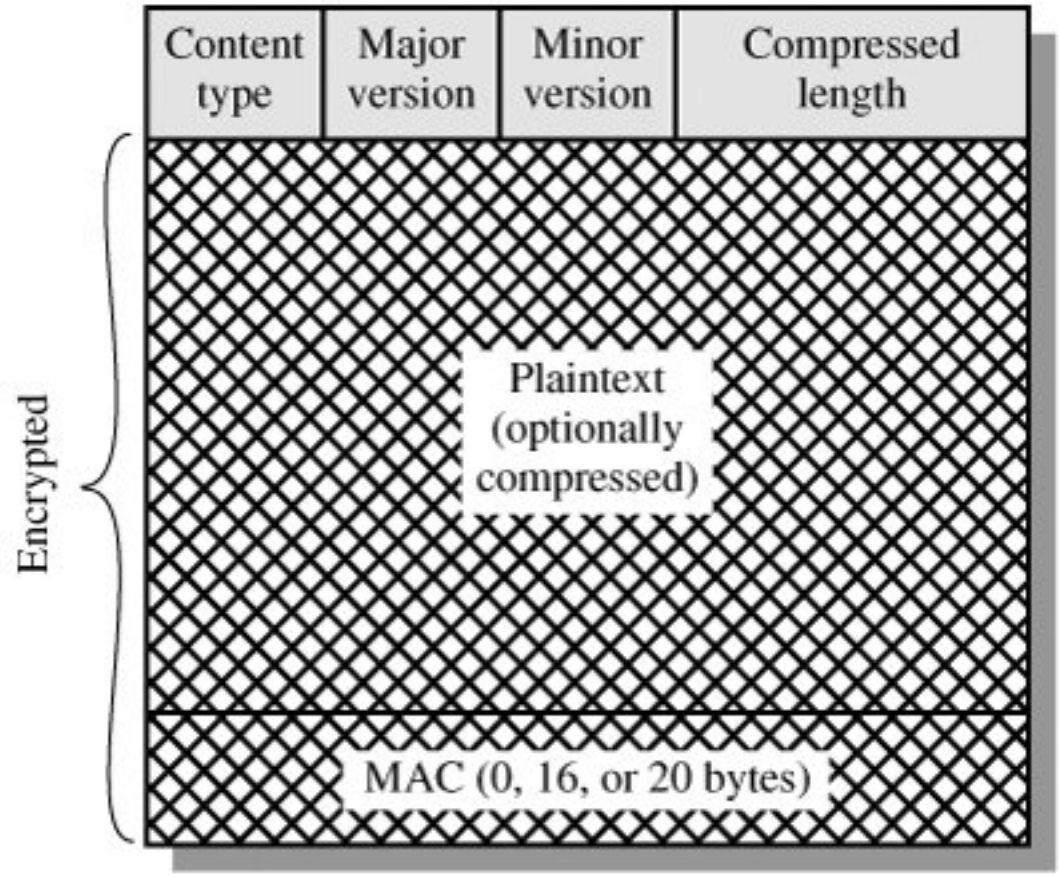
- **Content Type (8 bits):** The higher layer protocol used to process the enclosed fragment.
- **Major Version (8 bits):** Indicates major version of SSL in use. For SSLv3, the value is 3.

- **Minor Version (8 bits):** Indicates minor version in use. For SSLv3, the value is 0.
- **Compressed Length (16 bits):** The length in bytes of the plaintext fragment (or compressed fragment if compression is used). The maximum value is 2^{14} + 2048.

The content types that have been defined are change_cipher_spec, alert, handshake, and application_data. The first three are the SSL-specific protocols, discussed next. Note that no distinction is made among the various applications (e.g., HTTP) that might use SSL; the content of the data created by such applications is opaque to SSL.

Figure 17.4 illustrates the SSL record format.

Figure 17.4. SSL Record Format



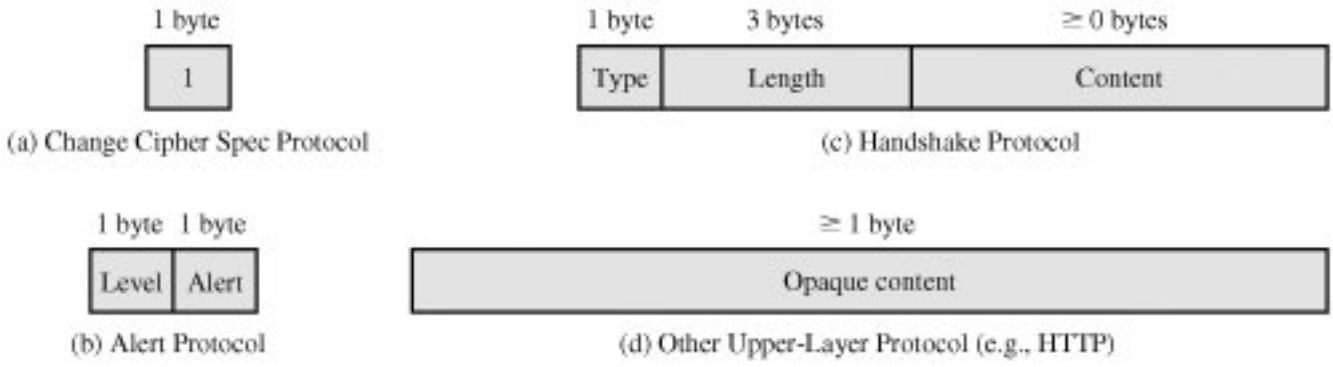
Change Cipher Spec Protocol

The Change Cipher Spec Protocol is one of the three SSL-specific protocols that use the SSL Record Protocol, and it is the simplest. This protocol consists of a single message (Figure 17.5a), which consists of a single byte with the value 1. The sole purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.

Figure 17.5. SSL Record Protocol Payload

(This item is displayed on page 536 in the print version)

[View full size image]



Alert Protocol

The Alert Protocol is used to convey SSL-related alerts to the peer entity. As with other applications that use SSL, alert messages are compressed and encrypted, as specified by the current state.

Each message in this protocol consists of two bytes (Figure 17.5b). The first byte takes the value warning(1) or fatal(2) to convey the severity of the message. If the level is fatal, SSL immediately terminates the connection. Other connections on the same session may continue, but no new connections on this session may be established. The second byte contains a code that indicates the specific alert. First, we list those alerts that are always fatal (definitions from the SSL specification):

- **unexpected_message:** An inappropriate message was received.
- **bad_record_mac:** An incorrect MAC was received.
- **decompression_failure:** The decompression function received improper input (e.g., unable to decompress or decompress to greater than maximum allowable length).
- **handshake_failure:** Sender was unable to negotiate an acceptable set of security parameters given the options available.
- **illegal_parameter:** A field in a handshake message was out of range or inconsistent with other fields.

The remainder of the alerts are the following:

- **close_notify:** Notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send a close_notify alert before closing the write side of a connection.
- **no_certificate:** May be sent in response to a certificate request if no appropriate certificate is available.
- **bad_certificate:** A received certificate was corrupt (e.g., contained a signature that did not verify).
- **unsupported_certificate:** The type of the received certificate is not supported.
- **certificate_revoked:** A certificate has been revoked by its signer.
- **certificate_expired:** A certificate has expired.
- **certificate_unknown:** Some other unspecified issue arose in processing the certificate, rendering it unacceptable.

Handshake Protocol

The most complex part of SSL is the Handshake Protocol. This protocol allows the server and client to authenticate each other and to negotiate an encryption and MAC algorithm and cryptographic keys to be used to protect data sent in an SSL record. The Handshake Protocol is used before any application data is transmitted.

The Handshake Protocol consists of a series of messages exchanged by client and server. All of these have the format shown in Figure 17.5c. Each message has three fields:

- **Type (1 byte):** Indicates one of 10 messages. Table 17.2 lists the defined message types.
- **Length (3 bytes):** The length of the message in bytes.
- **Content ($\geq$0 bytes):** The parameters associated with this message; these are listed in Table 17.2.

Table 17.2. SSL Handshake Protocol Message Types

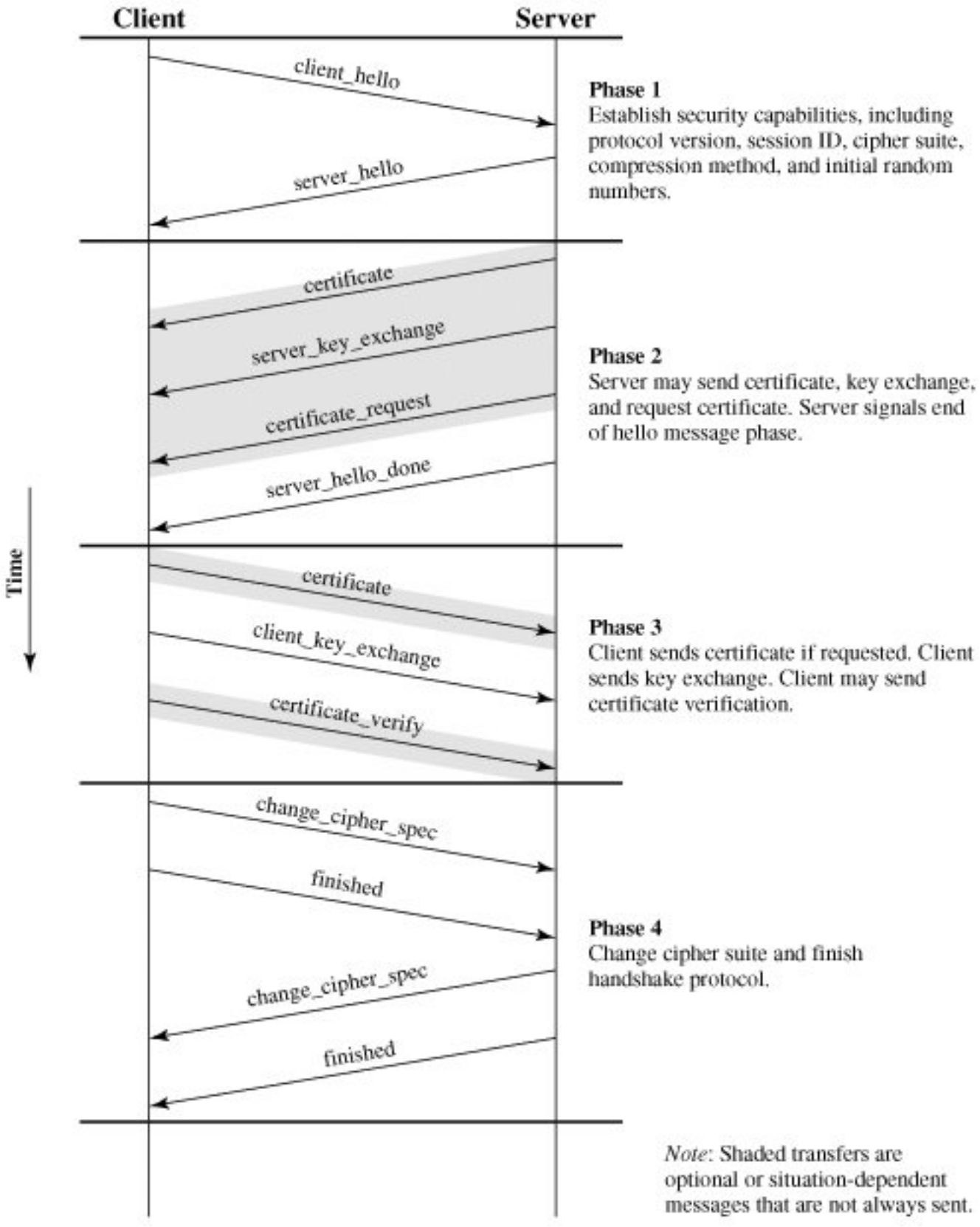
Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Figure 17.6 shows the initial exchange needed to establish a logical connection between client and server. The exchange can be viewed as having four phases.

Figure 17.6. Handshake Protocol Action

(This item is displayed on page 539 in the print version)

[View full size image]



Phase 1. Establish Security Capabilities

This phase is used to initiate a logical connection and to establish the security capabilities that will be associated with it. The exchange is initiated by the client, which sends a **client_hello message** with the following parameters:

- **Version:** The highest SSL version understood by the client.
- **Random:** A client-generated random structure, consisting of a 32-bit timestamp and 28 bytes

generated by a secure random number generator. These values serve as nonces and are used during key exchange to prevent replay attacks.
- **Session ID:** A variable-length session identifier. A nonzero value indicates that the client wishes to update the parameters of an existing connection or create a new connection on this session. A zero value indicates that the client wishes to establish a new connection on a new session.
- **CipherSuite:** This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec; these are discussed subsequently.
- **Compression Method:** This is a list of the compression methods the client supports.

After sending the client_hello message, the client waits for the **server_hello message**, which contains the same parameters as the client_hello message. For the server_hello message, the following conventions apply. The Version field contains the lower of the version suggested by the client and the highest supported by the server. The Random field is generated by the server and is independent of the client's Random field. If the SessionID field of the client was nonzero, the same value is used by the server; otherwise the server's SessionID field contains the value for a new session. The CipherSuite field contains the single cipher suite selected by the server from those proposed by the client. The Compression field contains the compression method selected by the server from those proposed by the client.

The first element of the Cipher Suite parameter is the key exchange method (i.e., the means by which the cryptographic keys for conventional encryption and MAC are exchanged). The following key exchange methods are supported:

- **RSA:** The secret key is encrypted with the receiver's RSA public key. A public-key certificate for the receiver's key must be made available.
- **Fixed Diffie-Hellman:** This is a Diffie-Hellman key exchange in which the server's certificate contains the Diffie-Hellman public parameters signed by the certificate authority (CA). That is, the public-key certificate contains the Diffie-Hellman public-key parameters. The client provides its Diffie-Hellman public key parameters either in a certificate, if client authentication is required, or in a key exchange message. This method results in a fixed secret key between two peers, based on the Diffie-Hellman calculation using the fixed public keys.
- **Ephemeral Diffie-Hellman:** This technique is used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, signed using the sender's private RSA or DSS key. The receiver can use the corresponding public key to verify the signature. Certificates are used to authenticate the public keys. This would appear to be the most secure of the three Diffie-Hellman options because it results in a temporary, authenticated key.
- **Anonymous Diffie-Hellman:** The base Diffie-Hellman algorithm is used, with no authentication. That is, each side sends its public Diffie-Hellman parameters to the other, with no authentication. This approach is vulnerable to man-in-the-middle attacks, in which the attacker conducts anonymous Diffie-Hellman with both parties.
- **Fortezza:** The technique defined for the Fortezza scheme.

Following the definition of a key exchange method is the CipherSpec, which includes the following fields:

- **CipherAlgorithm:** Any of the algorithms mentioned earlier: RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza
- **MACAlgorithm:** MD5 or SHA-1
- **CipherType:** Stream or Block
- **IsExportable:** True or False

- **HashSize:** 0, 16 (for MD5), or 20 (for SHA-1) bytes
- **Key Material:** A sequence of bytes that contain data used in generating the write keys
- **IV Size:** The size of the Initialization Value for Cipher Block Chaining (CBC) encryption

Phase 2. Server Authentication and Key Exchange

The server begins this phase by sending its certificate, if it needs to be authenticated; the message contains one or a chain of X.509 certificates. The **certificate message** is required for any agreed-on key exchange method except anonymous Diffie-Hellman. Note that if fixed Diffie-Hellman is used, this certificate message functions as the server's key exchange message because it contains the server's public Diffie-Hellman parameters.

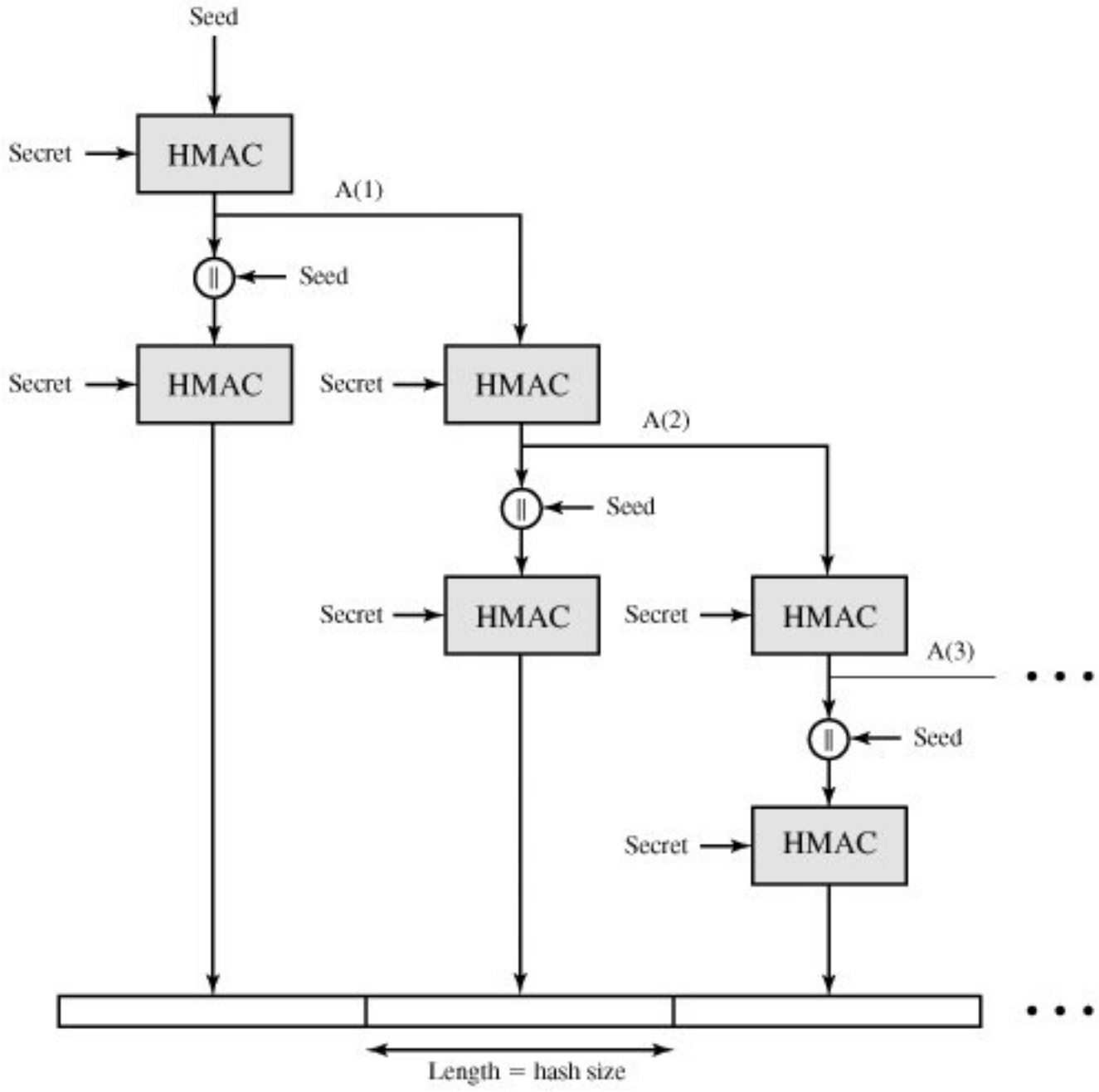
Next, a **server_key_exchange message** may be sent if it is required. It is not required in two instances: (1) The server has sent a certificate with fixed Diffie-Hellman parameters, or (2) RSA key exchange is to be used. The server_key_exchange message is needed for the following:

- **Anonymous Diffie-Hellman:** The message content consists of the two global Diffie-Hellman values (a prime number and a primitive root of that number) plus the server's public Diffie-Hellman key (see Figure 10.7).

Figure 17.7. TLS Function P_hash (secret, seed)

(This item is displayed on page 546 in the print version)

[View full size image]



- **Ephemeral Diffie-Hellman:** The message content includes the three Diffie-Hellman parameters provided for anonymous Diffie-Hellman, plus a signature of those parameters.
- **RSA key exchange, in which the server is using RSA but has a signature-only RSA key:** Accordingly, the client cannot simply send a secret key encrypted with the server's public key. Instead, the server must create a temporary RSA public/private key pair and use the server_key_exchange message to send the public key. The message content includes the two parameters of the temporary RSA public key (exponent and modulus; see Figure 9.5) plus a signature of those parameters.
- **Fortezza**

Some further details about the signatures are warranted. As usual, a signature is created by taking the hash of a message and encrypting it with the sender's private key. In this case the hash is defined as

```
hash(ClientHello.random || ServerHello.random || ServerParams)
```

So the hash covers not only the Diffie-Hellman or RSA parameters, but also the two nonces from the initial hello messages. This ensures against replay attacks and misrepresentation. In the case of a DSS signature, the hash is performed using the SHA-1 algorithm. In the case of an RSA signature, both an

MD5 and an SHA-1 hash are calculated, and the concatenation of the two hashes (36 bytes) is encrypted with the server's private key.

Next, a nonanonymous server (server not using anonymous Diffie-Hellman) can request a certificate from the client. The **certificate_request message** includes two parameters: certificate_type and certificate_authorities. The certificate type indicates the public-key algorithm and its use:

- RSA, signature only
- DSS, signature only
- RSA for fixed Diffie-Hellman; in this case the signature is used only for authentication, by sending a certificate signed with RSA
- DSS for fixed Diffie-Hellman; again, used only for authentication

---

- RSA for ephemeral Diffie-Hellman
- DSS for ephemeral Diffie-Hellman
- Fortezza

The second parameter in the certificate_request message is a list of the distinguished names of acceptable certificate authorities.

The final message in Phase 2, and one that is always required, is the **server_done message**, which is sent by the server to indicate the end of the server hello and associated messages. After sending this message, the server will wait for a client response. This message has no parameters.

Phase 3. Client Authentication and Key Exchange

Upon receipt of the server_done message, the client should verify that the server provided a valid certificate if required and check that the server_hello parameters are acceptable. If all is satisfactory, the client sends one or more messages back to the server.

If the server has requested a certificate, the client begins this phase by sending a **certificate message**. If no suitable certificate is available, the client sends a no_certificate alert instead.

Next is the **client_key_exchange message**, which must be sent in this phase. The content of the message depends on the type of key exchange, as follows:

- **RSA:** The client generates a 48-byte *pre-master secret* and encrypts with the public key from the server's certificate or temporary RSA key from a server_key_exchange message. Its use to compute a *master secret* is explained later.
- **Ephemeral or Anonymous Diffie-Hellman:** The client's public Diffie-Hellman parameters are sent.
- **Fixed Diffie-Hellman:** The client's public Diffie-Hellman parameters were sent in a certificate message, so the content of this message is null.
- **Fortezza:** The client's Fortezza parameters are sent.

Finally, in this phase, the client may send a **certificate_verify message** to provide explicit verification of a client certificate. This message is only sent following any client certificate that has signing capability (i.e., all certificates except those containing fixed Diffie-Hellman parameters). This message signs a hash code based on the preceding messages, defined as follows:

```
CertificateVerify.signature.md5_hash
     MD5(master_secret || pad_2 || MD5(handshake_messages ||
         master_secret || pad_1));
```

```
Certificate.signature.sha_hash
      SHA(master_secret || pad_2 || SHA(handshake_messages ||
          master_secret || pad_1));
```

where pad_1 and pad_2 are the values defined earlier for the MAC, handshake_messages refers to all Handshake Protocol messages sent or received starting at client_hello but not including this message, and master_secret is the calculated secret whose construction is explained later in this section. If the user's private key is DSS, then it is used to encrypt the SHA-1 hash. If the user's private key is RSA, it is used to encrypt the concatenation of the MD5 and SHA-1 hashes. In either case, the purpose is to verify the client's ownership of the private key for the client certificate. Even if someone is misusing the client's certificate, he or she would be unable to send this message.

Phase 4. Finish

This phase completes the setting up of a secure connection. The client sends a **change_cipher_spec message** and copies the pending CipherSpec into the current CipherSpec. Note that this message is not considered part of the Handshake Protocol but is sent using the Change Cipher Spec Protocol. The client then immediately sends the **finished message** under the new algorithms, keys, and secrets. The finished message verifies that the key exchange and authentication processes were successful. The content of the finished message is the concatenation of two hash values:

```
MD5(master_secret || pad2 || MD5(handshake_messages ||
    Sender || master_secret || pad1))
SHA(master_secret || pad2 || SHA(handshake_messages ||
    Sender || master_secret || pad1))
```

where Sender is a code that identifies that the sender is the client and handshake_messages is all of the data from all handshake messages up to but not including this message.

In response to these two messages, the server sends its own change_cipher_spec message, transfers the pending to the current CipherSpec, and sends its finished message. At this point the handshake is complete and the client and server may begin to exchange application layer data.

Cryptographic Computations

Two further items are of interest: the creation of a shared master secret by means of the key exchange, and the generation of cryptographic parameters from the master secret.

Master Secret Creation

The shared master secret is a one-time 48-byte value (384 bits) generated for this session by means of secure key exchange. The creation is in two stages. First, a pre_master_secret is exchanged. Second, the master_secret is calculated by both parties. For pre_master_secret exchange, there are two possibilities:

- **RSA:** A 48-byte pre_master_secret is generated by the client, encrypted with the server's public RSA key, and sent to the server. The server decrypts the ciphertext using its private key to recover the pre_master_secret.

- **Diffie-Hellman:** Both client and server generate a Diffie-Hellman public key. After these are exchanged, each side performs the Diffie-Hellman calculation to create the shared pre_master_secret.

Both sides now compute the master_secret as follows:

```
master_secret = MD5(pre_master_secret || SHA('A' ||
                    pre_master_secret ||ClientHello.random ||
                    ServerHello.random)) ||
                MD5(pre_master_secret || SHA('BB' ||
                    pre_master_secret || ClientHello.random ||
                    ServerHello.random)) ||
                MD5(pre_master_secret || SHA('CCC' ||
                    pre_master_secret || ClientHello.random ||
                    ServerHello.random))
```

where ClientHello.random and ServerHello.random are the two nonce values exchanged in the initial hello messages.

Generation of Cryptographic Parameters

CipherSpecs require a client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV, which are generated from the master secret in that order. These parameters are generated from the master secret by hashing the master secret into a sequence of secure bytes of sufficient length for all needed parameters.

The generation of the key material from the master secret uses the same format for generation of the master secret from the pre-master secret:

```
key_block = MD5(master_secret || SHA('A' || master_secret ||
                ServerHello.random || ClientHello.random)) ||
            MD5(master_secret || SHA('BB' || master_secret ||
                ServerHello.random || ClientHello.random)) ||
            MD5(master_secret || SHA('CCC' || master_
                secret || ServerHello.random ||
                ClientHello.random)) || . . .
```

until enough output has been generated. The result of this algorithmic structure is a pseudorandom function. We can view the master_secret as the pseudorandom seed value to the function. The client and server random numbers can be viewed as salt values to complicate cryptanalysis (see Chapter 18 for a discussion of the use of salt values).

Transport Layer Security

TLS is an IETF standardization initiative whose goal is to produce an Internet standard version of SSL. TLS is defined as a Proposed Internet Standard in RFC 2246. RFC 2246 is very similar to SSLv3. In this section, we highlight the differences.

Version Number

The TLS Record Format is the same as that of the SSL Record Format (Figure 17.4), and the fields in the header have the same meanings. The one difference is in version values. For the current version of TLS, the Major Version is 3 and the Minor Version is 1.

Message Authentication Code

There are two differences between the SSLv3 and TLS MAC schemes: the actual algorithm and the scope of the MAC calculation. TLS makes use of the HMAC algorithm defined in RFC 2104. Recall from Chapter 12 that HMAC is defined as follows:

$$\text{HMAC}_K(M) = \text{H}[(K^+ \oplus \text{opad})||\text{H}[(K^+ \oplus \text{ipad})||M]]$$

where

H	=	embedded hash function (for TLS, either MD5 or SHA-1)
M	=	message input to HMAC
K^+	=	secret key padded with zeros on the left so that the result is equal to the block length of the hash code(for MD5 and SHA-1, block length = 512 bits)
ipad	=	00110110 (36 in hexadecimal) repeated 64 times (512 bits)
opad	=	01011100 (5C in hexadecimal) repeated 64 times (512 bits)

SSLv3 uses the same algorithm, except that the padding bytes are concatenated with the secret key rather than being XORed with the secret key padded to the block length. The level of security should be about the same in both cases.

For TLS, the MAC calculation encompasses the fields indicated in the following expression:

```
HMAC_hash(MAC_write_secret, seq_num || TLSCompressed.type ||
          TLSCompressed.version || TLSCompressed.length ||
          TLSCompressed.fragment)
```

The MAC calculation covers all of the fields covered by the SSLv3 calculation, plus the field TLSCompressed.version, which is the version of the protocol being employed.

Pseudorandom Function

TLS makes use of a pseudorandom function referred to as PRF to expand secrets into blocks of data for purposes of key generation or validation. The objective is to make use of a relatively small shared secret value but to generate longer blocks of data in a way that is secure from the kinds of attacks made on hash functions and MACs. The PRF is based on the following data expansion function (Figure 17.7):

```
P_hash(secret, seed) = HMAC_hash(secret, A(1)  || seed) ||
                       HMAC_hash(secret, A(2)  || seed) ||
                       HMAC_hash(secret, A(3) || seed) || ...
```

where A() is defined as

A(0) = seed

A(*i*) = HMAC_hash (secret, A(*i* - 1))

The data expansion function makes use of the HMAC algorithm, with either MD5 or SHA-1 as the underlying hash function. As can be seen, P_hash can be iterated as many times as necessary to produce the required quantity of data. For example, if P_SHA-1 was used to generate 64 bytes of data, it would have to be iterated four times, producing 80 bytes of data, of which the last 16 would be discarded. In this case, P_MD5 would also have to be iterated four times, producing exactly 64 bytes of data. Note that each iteration involves two executions of HMAC, each of which in turn involves two executions of the underlying hash algorithm.

To make PRF as secure as possible, it uses two hash algorithms in a way that should guarantee its security if either algorithm remains secure. PRF is defined as

```
PRF(secret, label, seed) = P_MD5(S1, label || seed) ⊕
                           P_SHA-1(S2, label || seed)
```

PRF takes as input a secret value, an identifying label, and a seed value and produces an output of arbitrary length. The output is created by splitting the secret value into two halves (S1 and S2) and performing P_hash on each half, using MD5 on one half and SHA-1 on the other half. The two results are exclusive-ORed to produce the output; for this purpose, P_MD5 will generally have to be iterated more times than P_SHA-1 to produce an equal amount of data for input to the exclusive-OR function.

Alert Codes

TLS supports all of the alert codes defined in SSLv3 with the exception of no_certificate. A number of additional codes are defined in TLS; of these, the following are always fatal:

- **decryption_failed:** A ciphertext decrypted in an invalid way; either it was not an even multiple of the block length or its padding values, when checked, were incorrect.
- **record_overflow:** A TLS record was received with a payload (ciphertext) whose length exceeds 2^{14} + 2048 bytes, or the ciphertext decrypted to a length of greater than 2^{14} + 1024 bytes.
- **unknown_ca:** A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or could not be matched with a known, trusted CA.
- **access_denied:** A valid certificate was received, but when access control was applied, the sender decided not to proceed with the negotiation.

- **decode_error:** A message could not be decoded because a field was out of its specified range or the length of the message was incorrect.
- **export_restriction:** A negotiation not in compliance with export restrictions on key length was detected.
- **protocol_version:** The protocol version the client attempted to negotiate is recognized but not supported.
- **insufficient_security:** Returned instead of handshake_failure when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client.
- **internal_error:** An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue.

The remainder of the new alerts include the following:

- **decrypt_error:** A handshake cryptographic operation failed, including being unable to verify a signature, decrypt a key exchange, or validate a finished message.
- **user_canceled:** This handshake is being canceled for some reason unrelated to a protocol failure.
- **no_renegotiation:** Sent by a client in response to a hello request or by the server in response to a client hello after initial handshaking. Either of these messages would normally result in renegotiation, but this alert indicates that the sender is not able to renegotiate. This message is always a warning.

Cipher Suites

There are several small differences between the cipher suites available under SSLv3 and under TLS:

- **Key Exchange:** TLS supports all of the key exchange techniques of SSLv3 with the exception of Fortezza.
- **Symmetric Encryption Algorithms:** TLS includes all of the symmetric encryption algorithms found in SSLv3, with the exception of Fortezza.

Client Certificate Types

TLS defines the following certificate types to be requested in a certificate_request message: rsa_sign, dss_sign, rsa_fixed_dh, and dss_fixed_dh. These are all defined in SSLv3. In addition, SSLv3 includes rsa_ephemeral_dh, dss_ephemeral_dh, and fortezza_kea. Ephemeral Diffie-Hellman involves signing the Diffie-Hellman parameters with either RSA or DSS; for TLS, the rsa_sign and dss_sign types are used for that function; a separate signing type is not needed to sign Diffie-Hellman parameters. TLS does not include the Fortezza scheme.

Certificate_Verify and Finished Messages

In the TLS certificate_verify message, the MD5 and SHA-1 hashes are calculated only over handshake_messages. Recall that for SSLv3, the hash calculation also included the master secret and pads. These extra fields were felt to add no additional security.

As with the finished message in SSLv3, the finished message in TLS is a hash based on the shared master_secret, the previous handshake messages, and a label that identifies client or server. The calculation is somewhat different. For TLS, we have

```
PRF(master_secret, finished_label, MD5(handshake_messages)||
    SHA-1(handshake_messages))
```

where finished_label is the string "client finished" for the client and "server finished" for the server.

Cryptographic Computations

The pre_master_secret for TLS is calculated in the same way as in SSLv3. As in SSLv3, the master_secret in TLS is calculated as a hash function of the pre_master_secret and the two hello random numbers. The form of the TLS calculation is different from that of SSLv3 and is defined as follows:

```
master_secret = PRF(pre_master_secret, "master secret",
                ClientHello.random || ServerHello.random)
```

The algorithm is performed until 48 bytes of pseudorandom output are produced. The calculation of the key block material (MAC secret keys, session encryption keys, and IVs) is defined as follows:

```
key_block = PRF(master_secret, "key expansion",
            SecurityParameters.server_random ||
            SecurityParameters.client_random)
```

---

until enough output has been generated. As with SSLv3, the key_block is a function of the master_secret and the client and server random numbers, but for TLS the actual algorithm is different.

Padding

In SSL, the padding added prior to encryption of user data is the minimum amount required so that the total size of the data to be encrypted is a multiple of the cipher's block length. In TLS, the padding can be any amount that results in a total that is a multiple of the cipher's block length, up to a maximum of 255 bytes. For example, if the plaintext (or compressed text if compression is used) plus MAC plus padding.length byte is 79 bytes long, then the padding length, in bytes, can be 1, 9, 17, and so on, up to 249. A variable padding length may be used to frustrate attacks based on an analysis of the lengths of exchanged messages.

17.3. Secure Electronic Transaction

SET is an open encryption and security specification designed to protect credit card transactions on the Internet. The current version, SETv1, emerged from a call for security standards by MasterCard and Visa in February 1996. A wide range of companies were involved in developing the initial specification, including IBM, Microsoft, Netscape, RSA, Terisa, and Verisign. Beginning in 1996, there have been numerous tests of the concept, and by 1998 the first wave of SET-compliant products was available.

SET is not itself a payment system. Rather it is a set of security protocols and formats that enables users to employ the existing credit card payment infrastructure on an open network, such as the Internet, in a secure fashion. In essence, SET provides three services:

- Provides a secure communications channel among all parties involved in a transaction
- Provides trust by the use of X.509v3 digital certificates
- Ensures privacy because the information is only available to parties in a transaction when and where necessary

SET is a complex specification defined in three books issued in May of 1997:

- **Book 1:** Business Description (80 pages)
- **Book 2:** Programmer's Guide (629 pages)
- **Book 3:** Formal Protocol Definition (262 pages)

This is a total of 971 pages of specification. In contrast, the SSLv3 specification is 63 pages long and the TLS specification is 80 pages long. Accordingly, only a summary of this many-faceted specification is provided in this section.

SET Overview

A good way to begin our discussion of SET is to look at the business requirements for SET, its key features, and the participants in SET transactions.

Requirements

Book 1 of the SET specification lists the following business requirements for secure payment processing with credit cards over the Internet and other networks:

- **Provide confidentiality of payment and ordering information:** It is necessary to assure cardholders that this information is safe and accessible only to the intended recipient. Confidentiality also reduces the risk of fraud by either party to the transaction or by malicious third parties. SET uses encryption to provide confidentiality.
- **Ensure the integrity of all transmitted data:** That is, ensure that no changes in content occur during transmission of SET messages. Digital signatures are used to provide integrity.
- **Provide authentication that a cardholder is a legitimate user of a credit card account:** A mechanism that links a cardholder to a specific account number reduces the incidence of fraud and the overall cost of payment processing. Digital signatures and certificates are used to verify that a cardholder is a legitimate user of a valid account.

- **Provide authentication that a merchant can accept credit card transactions through its relationship with a financial institution:** This is the complement to the preceding requirement. Cardholders need to be able to identify merchants with whom they can conduct secure transactions. Again, digital signatures and certificates are used.
- **Ensure the use of the best security practices and system design techniques to protect all legitimate parties in an electronic commerce transaction:** SET is a well-tested specification based on highly secure cryptographic algorithms and protocols.
- **Create a protocol that neither depends on transport security mechanisms nor prevents their use:** SET can securely operate over a "raw" TCP/IP stack. However, SET does not interfere with the use of other security mechanisms, such as IPSec and SSL/TLS.
- **Facilitate and encourage interoperability among software and network providers:** The SET protocols and formats are independent of hardware platform, operating system, and Web software.

Key Features of SET

To meet the requirements just outlined, SET incorporates the following features:

- **Confidentiality of information:** Cardholder account and payment information is secured as it travels across the network. An interesting and important feature of SET is that it prevents the merchant from learning the cardholder's credit card number; this is only provided to the issuing bank. Conventional encryption by DES is used to provide confidentiality.
- **Integrity of data:** Payment information sent from cardholders to merchants includes order information, personal data, and payment instructions. SET guarantees that these message contents are not altered in transit. RSA digital signatures, using SHA-1 hash codes, provide message integrity. Certain messages are also protected by HMAC using SHA-1.

- **Cardholder account authentication:** SET enables merchants to verify that a cardholder is a legitimate user of a valid card account number. SET uses X.509v3 digital certificates with RSA signatures for this purpose.
- **Merchant authentication:** SET enables cardholders to verify that a merchant has a relationship with a financial institution allowing it to accept payment cards. SET uses X.509v3 digital certificates with RSA signatures for this purpose.

Note that unlike IPSec and SSL/TLS, SET provides only one choice for each cryptographic algorithm. This makes sense, because SET is a single application with a single set of requirements, whereas IPSec and SSL/TLS are intended to support a range of applications.

SET Participants

Figure 17.8 indicates the participants in the SET system, which include the following:

- **Cardholder:** In the electronic environment, consumers and corporate purchasers interact with merchants from personal computers over the Internet. A cardholder is an authorized holder of a payment card (e.g., MasterCard, Visa) that has been issued by an issuer.
- **Merchant:** A merchant is a person or organization that has goods or services to sell to the cardholder. Typically, these goods and services are offered via a Web site or by electronic mail. A merchant that accepts payment cards must have a relationship with an acquirer.

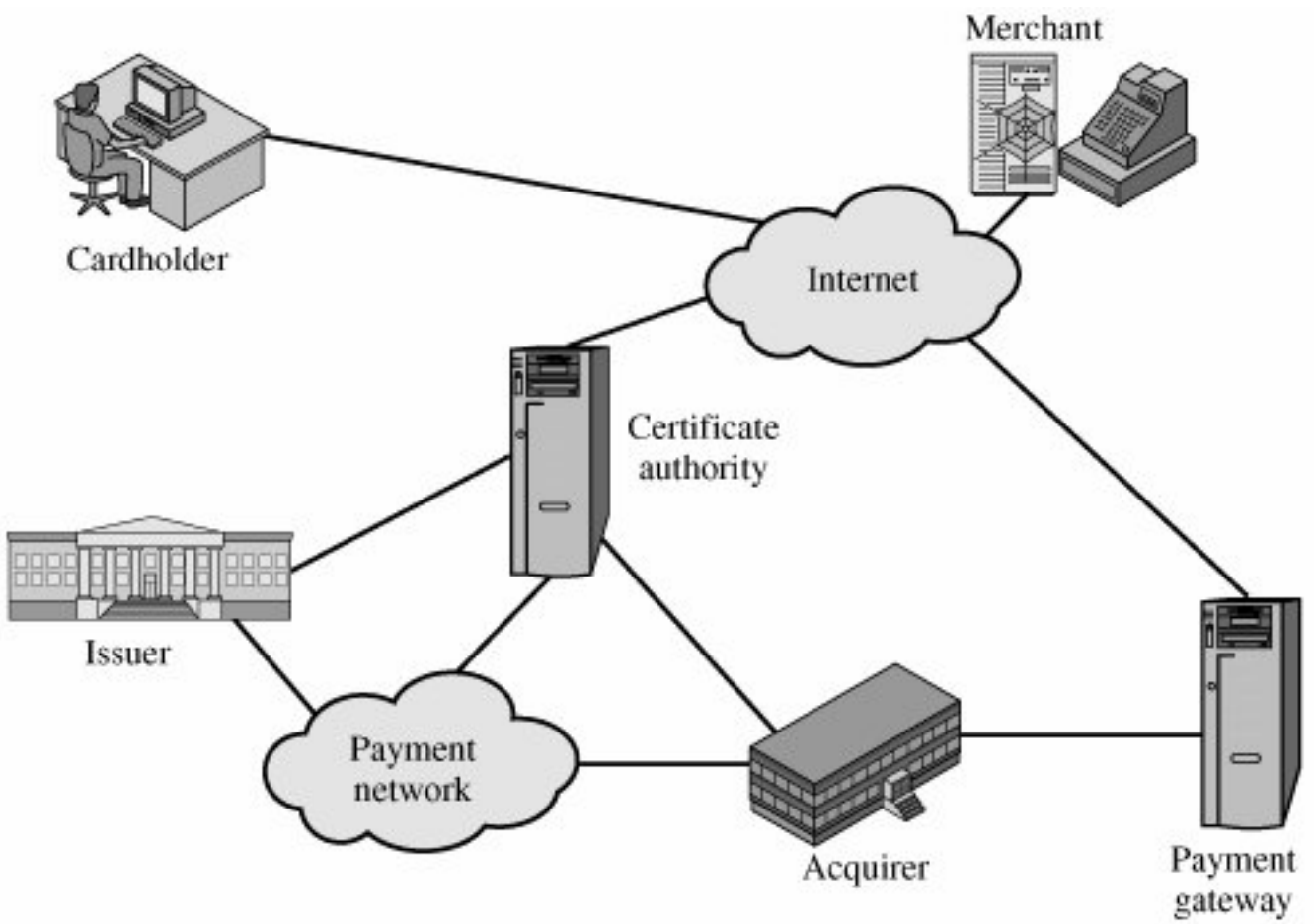
- **Issuer**: This is a financial institution, such as a bank, that provides the cardholder with the payment card. Typically, accounts are applied for and opened by mail or in person. Ultimately, it

is the issuer that is responsible for the payment of the debt of the cardholder.

- **Acquirer**: This is a financial institution that establishes an account with a merchant and processes payment card authorizations and payments. Merchants will usually accept more than one credit card brand but do not want to deal with multiple bankcard associations or with multiple individual issuers. The acquirer provides authorization to the merchant that a given card account is active and that the proposed purchase does not exceed the credit limit. The acquirer also provides electronic transfer of payments to the merchant's account. Subsequently, the acquirer is reimbursed by the issuer over some sort of payment network for electronic funds transfer.
- **Payment gateway**: This is a function operated by the acquirer or a designated third party that processes merchant payment messages. The payment gateway interfaces between SET and the existing bankcard payment networks for authorization and payment functions. The merchant exchanges SET messages with the payment gateway over the Internet, while the payment gateway has some direct or network connection to the acquirer's financial processing system.
- **Certification authority (CA):** This is an entity that is trusted to issue X.509v3 public-key certificates for cardholders, merchants, and payment gateways. The success of SET will depend on the existence of a CA infrastructure available for this purpose. As was discussed in previous chapters, a hierarchy of CAs is used, so that participants need not be directly certified by a root authority.

Figure 17.8. Secure Electronic Commerce Components

(This item is displayed on page 551 in the print version)



We now briefly describe the sequence of events that are required for a transaction. We will then look at some of the cryptographic details.

1.

The customer opens an account. The customer obtains a credit card account, such as MasterCard or Visa, with a bank that supports electronic payment and SET.

2.

The customer receives a certificate. After suitable verification of identity, the customer receives an X.509v3 digital certificate, which is signed by the bank. The certificate verifies the customer's RSA public key and its expiration date. It also establishes a relationship, guaranteed by the bank, between the customer's key pair and his or her credit card.

3.

Merchants have their own certificates. A merchant who accepts a certain brand of card must be in possession of two certificates for two public keys owned by the merchant: one for signing messages, and one for key exchange. The merchant also needs a copy of the payment gateway's public-key certificate.

4.

The customer places an order. This is a process that may involve the customer first browsing through the merchant's Web site to select items and determine the price. The customer then sends a list of the items to be purchased to the merchant, who returns an order form containing the list of items, their price, a total price, and an order number.

5.

The merchant is verified. In addition to the order form, the merchant sends a copy of its certificate, so that the customer can verify that he or she is dealing with a valid store.

6.

The order and payment are sent. The customer sends both order and payment information to the merchant, along with the customer's certificate. The order confirms the purchase of the items in the order form. The payment contains credit card details. The payment information is encrypted in such a way that it cannot be read by the merchant. The customer's certificate enables the merchant to verify the customer.

7.

The merchant requests payment authorization. The merchant sends the payment information to the payment gateway, requesting authorization that the customer's available credit is sufficient for this purchase.

8.

The merchant confirms the order. The merchant sends confirmation of the order to the customer.

9.

The merchant provides the goods or service. The merchant ships the goods or provides the service to the customer.

10.

The merchant requests payment. This request is sent to the payment gateway, which handles all of the payment processing.

Dual Signature

Before looking at the details of the SET protocol, let us discuss an important innovation introduced in SET: the dual signature. The purpose of the dual signature is to link two messages that are intended for two different recipients. In this case, the customer wants to send the order information (OI) to the merchant and the payment information (PI) to the bank. The merchant does not need to know the customer's credit card number, and the bank does not need to know the details of the customer's order. The customer is afforded extra protection in terms of privacy by keeping these two items separate. However, the two items must be linked in a way that can be used to resolve disputes if necessary. The link is needed so that the customer can prove that this payment is intended for this order and not for some other goods or service.

To see the need for the link, suppose that the customers send the merchant two messages: a signed OI and a signed PI, and the merchant passes the PI on to the bank. If the merchant can capture another OI from this customer, the merchant could claim that this OI goes with the PI rather than the original OI. The linkage prevents this.

Figure 17.9 shows the use of a dual signature to meet the requirement of the preceding paragraph. The customer takes the hash (using SHA-1) of the PI and the hash of the OI. These two hashes are then concatenated and the hash of the result is taken. Finally, the customer encrypts the final hash with his or her private signature key, creating the dual signature. The operation can be summarized as

$$DS = E(PR_C, [H(H(PI)||H(OI)])$$

where PR_C is the customer's private signature key. Now suppose that the merchant is in possession of the dual signature (DS), the OI, and the message digest for the PI (PIMD). The merchant also has the public key of the customer, taken from the customer's certificate. Then the merchant can compute the quantities

---

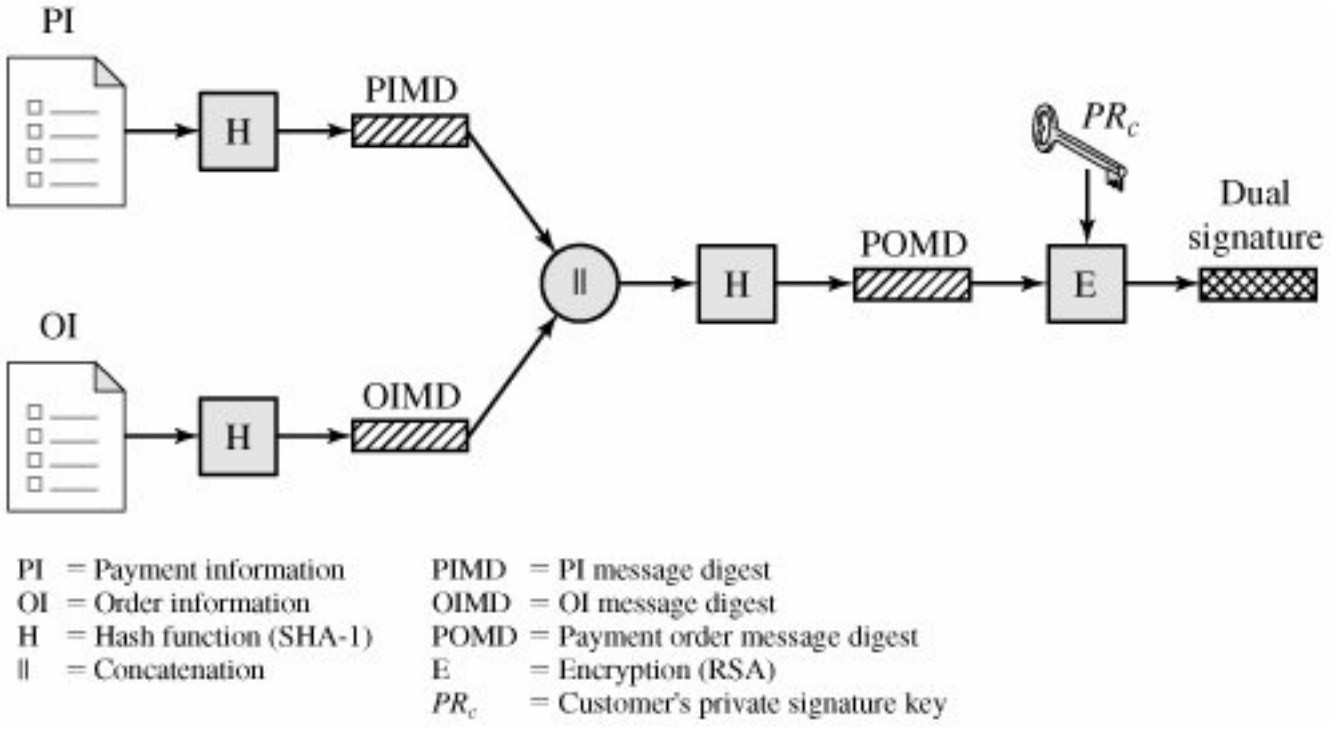
$$H(PIMS||H[OI]); D(PU_C, DS)$$

where PU_C is the customer's public signature key. If these two quantities are equal, then the merchant has verified the signature. Similarly, if the bank is in possession of DS, PI, the message digest for OI (OIMD), and the customer's public key, then the bank can compute

$$H(H[OI]||OIMD); D(PU_C, DS)$$

Figure 17.9. Construction of Dual Signature

[View full size image]



Again, if these two quantities are equal, then the bank has verified the signature. In summary,

1. The merchant has received OI and verified the signature.
2. The bank has received PI and verified the signature.
3. The customer has linked the OI and PI and can prove the linkage.

For example, suppose the merchant wishes to substitute another OI in this transaction, to its advantage. It would then have to find another OI whose hash matches the existing OIMD. With SHA-1, this is deemed not to be feasible. Thus, the merchant cannot link another OI with this PI.

Payment Processing

Table 17.3 lists the transaction types supported by SET. In what follows we look in some detail at the following transactions:

- Purchase request
- Payment authorization
- Payment capture

Table 17.3. SET Transaction Types

Cardholder registration	Cardholders must register with a CA before they can send SET messages to merchants.
Merchant registration	Merchants must register with a CA before they can exchange SET messages with customers and payment gateways.
Purchase request	Message from customer to merchant containing OI for merchant and PI for bank.
Payment authorization	Exchange between merchant and payment gateway to authorize a given amount for a purchase on a given credit card account.
Payment capture	Allows the merchant to request payment from the payment gateway.
Certificate inquiry and status	If the CA is unable to complete the processing of a certificate request quickly, it will send a reply to the cardholder or merchant indicating that the requester should check back later. The cardholder or merchant sends the *Certificate Inquiry* message to determine the status of the certificate request and to receive the certificate if the request has been approved.
Purchase inquiry	Allows the cardholder to check the status of the processing of an order after the purchase response has been received. Note that this message does not include information such as the status of back ordered goods, but does indicate the status of authorization, capture and credit processing.
Authorization reversal	Allows a merchant to correct previous authorization requests. If the order will not be completed, the merchant reverses the entire authorization. If part of the order will not be completed (such as when goods are back ordered), the merchant reverses part of the amount of the authorization.
Capture reversal	Allows a merchant to correct errors in capture requests such as transaction amounts that were entered incorrectly by a clerk.
Credit	Allows a merchant to issue a credit to a cardholder's account such as when goods are returned or were damaged during shipping. Note that the SET *Credit* message is always initiated by the merchant, not the cardholder. All communications between the cardholder and merchant that result in a credit being processed happen outside of SET.
Credit reversal	Allows a merchant to correct a previously request credit.
Payment gateway certificate request	Allows a merchant to query the payment gateway and receive a copy of the gateway's current key-exchange and signature certificates.
Batch administration	Allows a merchant to communicate information to the payment gateway regarding merchant batches.
Error message	Indicates that a responder rejects a message because it fails format or content verification tests.

Purchase Request

Before the Purchase Request exchange begins, the cardholder has completed browsing, selecting, and ordering. The end of this preliminary phase occurs when the merchant sends a completed order form to the customer. All of the preceding occurs without the use of SET.

The purchase request exchange consists of four messages: Initiate Request, Initiate Response, Purchase Request, and Purchase Response.

In order to send SET messages to the merchant, the cardholder must have a copy of the certificates of the merchant and the payment gateway. The customer requests the certificates in the **Initiate Request** message, sent to the merchant. This message includes the brand of the credit card that the customer is using. The message also includes an ID assigned to this request/response pair by the customer and a nonce used to ensure timeliness.

The merchant generates a response and signs it with its private signature key. The response includes the nonce from the customer, another nonce for the customer to return in the next message, and a transaction ID for this purchase transaction. In addition to the signed response, the **Initiate Response** message includes the merchant's signature certificate and the payment gateway's key exchange certificate.

The cardholder verifies the merchant and gateway certificates by means of their respective CA signatures and then creates the OI and PI. The transaction ID assigned by the merchant is placed in both the OI and PI. The OI does not contain explicit order data such as the number and price of items. Rather, it contains an order reference generated in the exchange between merchant and customer during the shopping phase before the first SET message. Next, the cardholder prepares the **Purchase Request** message (Figure 17.10). For this purpose, the cardholder generates a one-time symmetric encryption key, K_S. The message includes the following:

1.

Purchase-related information. This information will be forwarded to the payment gateway by the merchant and consists of

- The PI

- The dual signature, calculated over the PI and OI, signed with the customer's private signature key

- The OI message digest (OIMD)

The OIMD is needed for the payment gateway to verify the dual signature, as explained previously. All of these items are encrypted with K_S. The final item is

- The digital envelope. This is formed by encrypting K_S with the payment gateway's public key-exchange key. It is called a digital envelope because this envelope must be opened (decrypted) before the other items listed previously can be read.

The value of K_S is not made available to the merchant. Therefore, the merchant cannot read any of this payment-related information.

2.

Order-related information. This information is needed by the merchant and consists of

- The OI

- The dual signature, calculated over the PI and OI, signed with the customer's private signature key

- The PI message digest (PIMD)

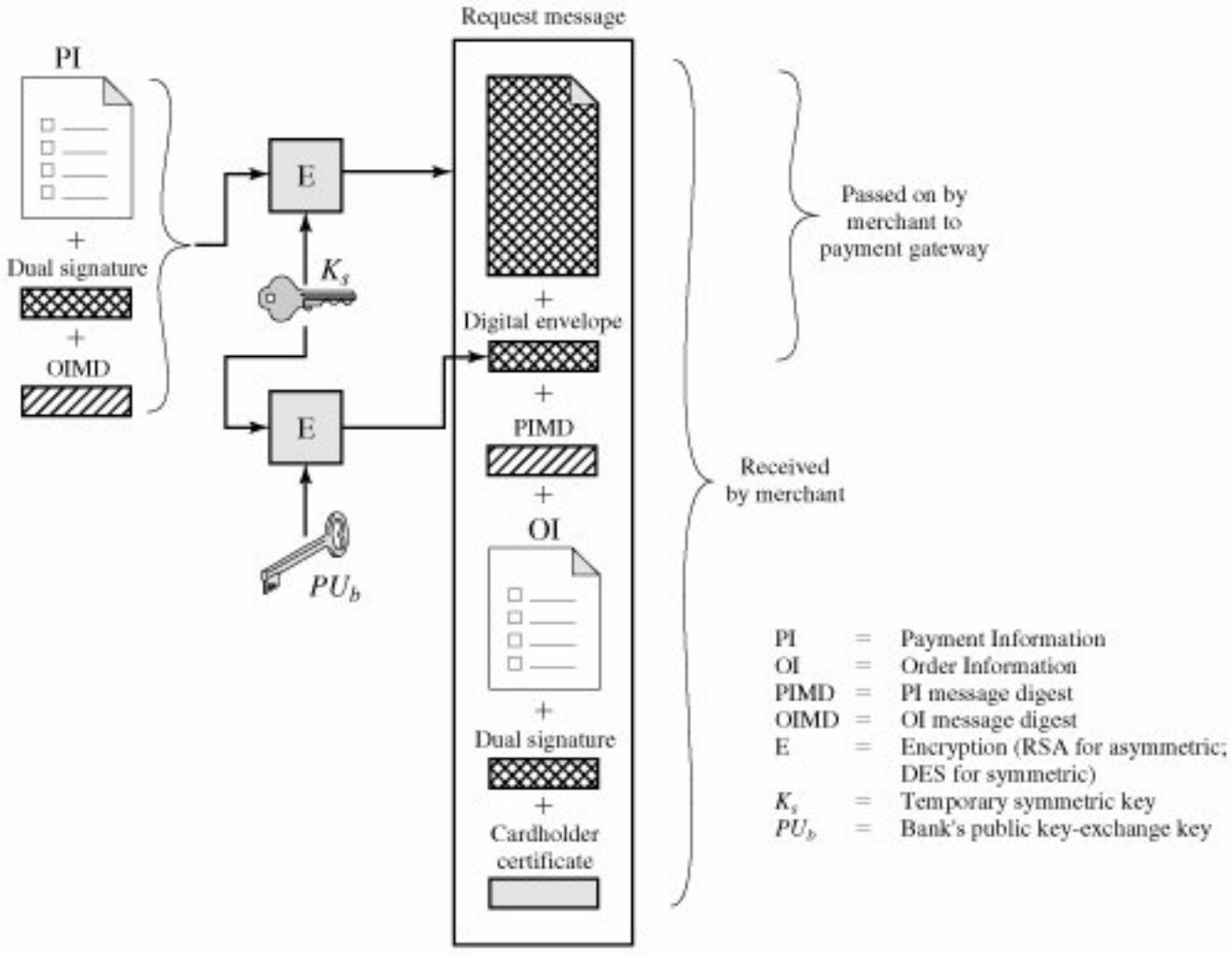
The PIMD is needed for the merchant to verify the dual signature. Note that the OI is sent in the clear.

3.

Cardholder certificate. This contains the cardholder's public signature key. It is needed by the merchant and by the payment gateway.

Figure 17.10. Cardholder Sends Purchase Request

[View full size image]



When the merchant receives the Purchase Request message, it performs the following actions (Figure 17.11):

1. Verifies the cardholder certificates by means of its CA signatures.

2. Verifies the dual signature using the customer's public signature key. This ensures that the order has not been tampered with in transit and that it was signed using the cardholder's private signature key.

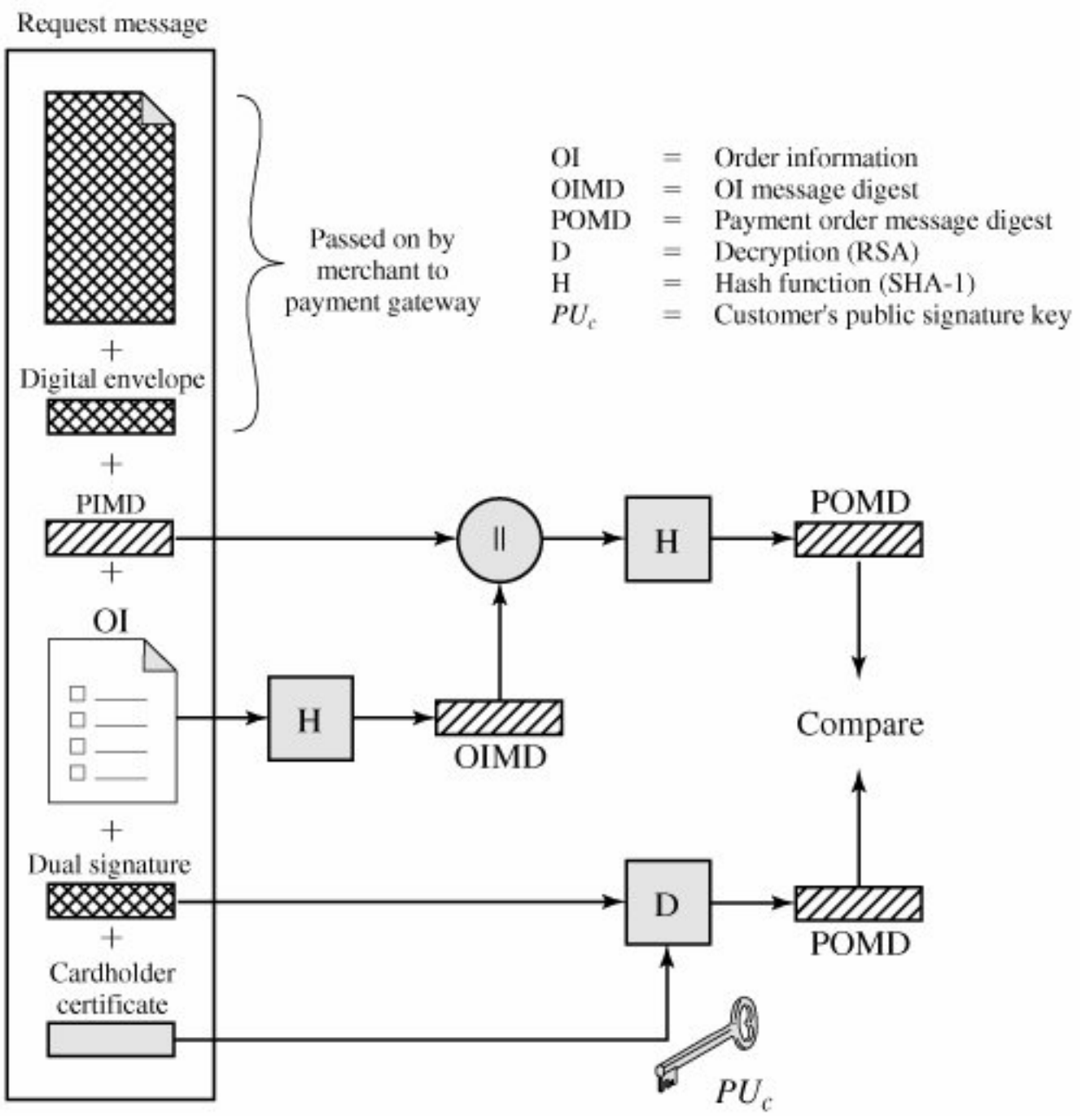
3. Processes the order and forwards the payment information to the payment gateway for authorization (described later).

4. Sends a purchase response to the cardholder.

Figure 17.11. Merchant Verifies Customer Purchase Request

(This item is displayed on page 558 in the print version)

[View full size image]



The **Purchase Response** message includes a response block that acknowledges the order and references the corresponding transaction number. This block is signed by the merchant using its private signature key. The block and its signature are sent to the customer, along with the merchant's signature certificate.

When the cardholder software receives the purchase response message, it verifies the merchant's certificate and then verifies the signature on the response block. Finally, it takes some action based on the response, such as displaying a message to the user or updating a database with the status of the

order.

Payment Authorization

During the processing of an order from a cardholder, the merchant authorizes the transaction with the payment gateway. The payment authorization ensures that the transaction was approved by the issuer. This authorization guarantees that the merchant will receive payment; the merchant can therefore provide the services or goods to the customer. The payment authorization exchange consists of two messages: Authorization Request and Authorization response.

The merchant sends an **Authorization Request** message to the payment gateway consisting of the following:

1. **Purchase-related information.** This information was obtained from the customer and consists of
   - The PI
   - The dual signature, calculated over the PI and OI, signed with the customer's private signature key
   - The OI message digest (OIMD)
   - The digital envelope
2. **Authorization-related information.** This information is generated by the merchant and consists of
   - An authorization block that includes the transaction ID, signed with the merchant's private signature key and encrypted with a one-time symmetric key generated by the merchant
   - A digital envelope. This is formed by encrypting the one-time key with the payment gateway's public key-exchange key.
3. **Certificates.** The merchant includes the cardholder's signature key certificate (used to verify the dual signature), the merchant's signature key certificate (used to verify the merchant's signature), and the merchant's key-exchange certificate (needed in the payment gateway's response).

The payment gateway performs the following tasks:

1. Verifies all certificates

2.

Decrypts the digital envelope of the authorization block to obtain the symmetric key and then decrypts the authorization block

3.

Verifies the merchant's signature on the authorization block

4.

Decrypts the digital envelope of the payment block to obtain the symmetric key and then decrypts the payment block

5.

Verifies the dual signature on the payment block

6.

Verifies that the transaction ID received from the merchant matches that in the PI received (indirectly) from the customer

7.

Requests and receives an authorization from the issuer

Having obtained authorization from the issuer, the payment gateway returns an **Authorization Response** message to the merchant. It includes the following elements:

1.

Authorization-related information. Includes an authorization block, signed with the gateway's private signature key and encrypted with a one-time symmetric key generated by the gateway. Also includes a digital envelope that contains the one-time key encrypted with the merchants public key-exchange key.

2.

Capture token information. This information will be used to effect payment later. This block is of the same form as (1), namely, a signed, encrypted capture token together with a digital envelope. This token is not processed by the merchant. Rather, it must be returned, as is, with a payment request.

3.

Certificate. The gateway's signature key certificate.

With the authorization from the gateway, the merchant can provide the goods or service to the customer.

Payment Capture

To obtain payment, the merchant engages the payment gateway in a payment capture transaction, consisting of a capture request and a capture response message.

For the **Capture Request** message, the merchant generates, signs, and encrypts a capture request block, which includes the payment amount and the transaction ID. The message also includes the encrypted capture token received earlier (in the Authorization Response) for this transaction, as well as the merchant's signature key and key-exchange key certificates.

When the payment gateway receives the capture request message, it decrypts and verifies the capture request block and decrypts and verifies the capture token block. It then checks for consistency between the capture request and capture token. It then creates a clearing request that is sent to the issuer over the private payment network. This request causes funds to be transferred to the merchant's account.

The gateway then notifies the merchant of payment in a **Capture Response** message. The message includes a capture response block that the gateway signs and encrypts. The message also includes the gateway's signature key certificate. The merchant software stores the capture response to be used for reconciliation with payment received from the acquirer.

PREV NEXT

17.4. Recommended Reading and Web Sites

[RESC01] is a good detailed treatment of SSL and TLS.

The best-detailed overview of SET is in Book 1 of the specification, available at the MasterCard SET Web site. Another excellent overview is [MACG97]. [DREW99] is also a good source.

DREW99 Drew, G. *Using SET for Secure Electronic Commerce*. Upper Saddle River, NJ: Prentice Hall, 1999.

MACG97 Macgregor, R.; Ezvan, C.; Liguori, L.; and Han, J. *Secure Electronic Transactions: Credit Card Payment on the Web in Theory and Practice*. IBM RedBook SG24-4978-00, 1997. Available at **www.redbooks.ibm.com**.

RESC01 Rescorla, E. *SSL and TLS: Designing and Building Secure Systems*. Reading, MA: Addison-Wesley, 2001.

Recommended Web Sites

- **Netscape's SSL Page:** Contains the SSL specification.
- **Transport Layer Security Charter:** Latest RFCs and Internet drafts for TLS.
- **OpenSSL Project:** Project to develop open-source SSL and TLS software. Site includes documents and links.

17.5. Key Terms, Review Questions, and Problems

Key Terms

acquirer

cardholder

certification authority (CA)

dual signature

issuer

merchant

payment gateway

Secure Electronic Transaction (SET)

Secure Socket Layer (SSL)

Transport Layer Security (TLS)

Review Questions

17.1 What are the advantages of each of the three approaches shown in Figure 17.1?

17.2 What protocols comprise SSL?

17.3 What is the difference between an SSL connection and an SSL session?

17.4 List and briefly define the parameters that define an SSL session state.

17.5 List and briefly define the parameters that define an SSL session connection.

17.6 What services are provided by the SSL Record Protocol?

17.7 What steps are involved in the SSL Record Protocol transmission?

17.8 List and briefly define the principal categories of SET participants.

17.9 What is a dual signature and what is its purpose?

Problems

17.1 In SSL and TLS, why is there a separate Change Cipher Spec Protocol, rather than including a change_cipher_spec message in the Handshake Protocol?

17.2 Consider the following threats to Web security and describe how each is countered by a particular feature of SSL.

a.

Brute-Force Cryptanalytic Attack: An exhaustive search of the key space for a conventional encryption algorithm.

b.

Known Plaintext Dictionary Attack: Many messages will contain predictable plaintext, such as the HTTP GET command. An attacker constructs a dictionary containing every possible encryption of the known-plaintext message. When an encrypted message is intercepted, the attacker takes the portion containing the encrypted known plaintext and looks up the ciphertext in the dictionary. The ciphertext should match against an entry that was encrypted with the same secret key. If there are several matches, each of these can be tried against the full ciphertext to determine the right one. This attack is especially effective against small key sizes (e.g., 40-bit keys).

---

c.

Replay Attack: Earlier SSL handshake messages are replayed.

d.

Man-in-the-Middle Attack: An attacker interposes during key exchange, acting as the client to the server and as the server to the client.

Password Sniffing: Passwords in HTTP or other application traffic are eavesdropped.

f.

IP Spoofing: Uses forged IP addresses to fool a host into accepting bogus data.

g.

IP Hijacking: An active, authenticated connection between two hosts is disrupted and the attacker takes the place of one of the hosts.

h.

SYN Flooding: An attacker sends TCP SYN messages to request a connection but does not respond to the final message to establish the connection fully. The attacked TCP module typically leaves the "half-open connection" around for a few minutes. Repeated SYN messages can clog the TCP module.

17.3 Based on what you have learned in this chapter, is it possible in SSL for the receiver to reorder SSL record blocks that arrive out of order? If so, explain how it can be done. If not, why not?

PREV NEXT

Part Four: System Security

Security is a concern of organizations with assets that are controlled by computer systems. By accessing or altering data, an attacker can steal tangible assets or lead an organization to take actions it would not otherwise take. By merely examining data, an attacker can gain a competitive advantage, without the owner of the data being any the wiser.

Computers at Risk: Safe Computing in the Information Age, National Research Council, 1991

The developers of secure software cannot adopt the various probabilistic measures of quality that developers of other software often can. For many applications, it is quite reasonable to tolerate a flaw that is rarely exposed and to assume that its having occurred once does not increase the likelihood that it will occur again. It is also reasonable to assume that logically independent failures will be statistically independent and not happen in concert. In contrast, a security vulnerability, once discovered, will be rapidly disseminated among a community of attackers and can be expected to be exploited on a regular basis until it is fixed.

Computers at Risk: Safe Computing in the Information Age, National Research Council, GAO/OSI-94-2, November 1993

Part Four looks at system-level security issues, including the threat of and countermeasures for intruders and viruses and the use of firewalls and trusted systems.

Road Map for Part Four

Chapter 18: Intruders

Chapter 18 examines a variety of information access and service threats presented by hackers that exploit vulnerabilities in network-based computing systems. The chapter begins with a discussion of the types of attacks that can be made by unauthorized users, or intruders, and analyzes various approaches to prevention and detection. This chapter also covers the related issue of password management.

Chapter 19: Malicious Software

Chapter 19 examines software threats to systems, with a special emphasis on viruses and worms. The chapter begins with a survey of various types of

malicious software, with a more detailed look at the nature of viruses and worms. The chapter then looks at countermeasures. Finally, this chapter deals with distributed denial of service attacks.

Chapter 20: Firewalls

A standard approach to the protection of local computer assets from external threats is the use of a firewall. Chapter 20 discusses the principles of firewall design and looks at specific techniques. This chapter also covers the related issue of trusted systems.

PREV NEXT

Chapter 18. Intruders

They agreed that Graham should set the test for Charles Mabledene. It was neither more nor less than that Dragon should get Stern's code. If he had the 'in' at Utting which he claimed to have this should be possible, only loyalty to Moscow Centre would prevent it. If he got the key to the code he would prove his loyalty to London Central beyond a doubt.

Talking to Strange Men, Ruth Rendell

Key Points

- Unauthorized intrusion into a computer system or network is one of the most serious threats to computer security.
- Intrusion detection systems have been developed to provide early warning of an intrusion so that defensive action can be taken to prevent or minimize damage.
- Intrusion detection involves detecting unusual patterns of activity or patterns of activity that are known to correlate with intrusions.
- One important element of intrusion prevention is password management, with the goal of preventing unauthorized users from having access to the passwords of others.

A significant security problem for networked systems is hostile, or at least unwanted, trespass by users or software. User trespass can take the form of unauthorized logon to a machine or, in the case of an authorized user, acquisition of privileges or performance of actions beyond those that have been authorized. Software trespass can take the form of a virus, worm, or Trojan horse.

All these attacks relate to network security because system entry can be achieved by means of a network. However, these attacks are not confined to network-based attacks. A user with access to a local terminal may attempt trespass without using an intermediate network. A virus or Trojan horse may be introduced into a system by means of a diskette. Only the worm is a uniquely network phenomenon. Thus, system trespass is an area in which the concerns of network security and computer security overlap.

Because the focus of this book is network security, we do not attempt a comprehensive analysis of either the attacks or the security countermeasures related to system trespass. Instead, in this Part we present a broad overview of these concerns.

This chapter covers with the subject of intruders. First, we examine the nature of the attack and then

look at strategies intended for prevention and, failing that, detection. Next we examine the related topic of password management.

PREV NEXT

18.1. Intruders

One of the two most publicized threats to security is the intruder (the other is viruses), generally referred to as a hacker or cracker. In an important early study of intrusion, Anderson [ANDE80] identified three classes of intruders:

- **Masquerader:** An individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account
- **Misfeasor:** A legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuses his or her privileges
- **Clandestine user:** An individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection

The masquerader is likely to be an outsider; the misfeasor generally is an insider; and the clandestine user can be either an outsider or an insider.

Intruder attacks range from the benign to the serious. At the benign end of the scale, there are many people who simply wish to explore internets and see what is out there. At the serious end are individuals who are attempting to read privileged data, perform unauthorized modifications to data, or disrupt the system.

The intruder threat has been well publicized, particularly because of the famous "Wily Hacker" incident of 19861987, documented by Cliff Stoll [STOL88, 89]. In 1990 there was a nationwide crackdown on illicit computer hackers, with arrests, criminal charges, one dramatic show trial, several guilty pleas, and confiscation of massive amounts of data and computer equipment [STER92]. Many people believed that the problem had been brought under control.

In fact, the problem has not been brought under control. To cite one example, a group at Bell Labs [BELL92, BELL93] has reported persistent and frequent attacks on its computer complex via the Internet over an extended period and from a variety of sources. At the time of these reports, the Bell group was experiencing the following:

- Attempts to copy the password file (discussed later) at a rate exceeding once every other day
- Suspicious remote procedure call (RPC) requests at a rate exceeding once per week
- Attempts to connect to nonexistent "bait" machines at least every two weeks

Benign intruders might be tolerable, although they do consume resources and may slow performance for legitimate users. However, there is no way in advance to know whether an intruder will be benign or malign. Consequently, even for systems with no particularly sensitive resources, there is a motivation to control this problem.

An example that dramatically illustrates the threat occurred at Texas A&M University [SAFF93]. In August 1992, the computer center there was notified that one of its machines was being used to attack computers at another location via the Internet. By monitoring activity, the computer center personnel learned that there were several outside intruders involved, who were running password-cracking routines on various computers (the site consists of a total of 12,000 interconnected machines). The center disconnected affected machines, plugged known security holes, and resumed normal operation. A few days later, one of the local system managers detected that the intruder attack had resumed. It turned out that the attack was far more sophisticated than had been originally believed. Files were found containing hundreds of captured passwords, including some on major and supposedly secure servers. In

addition, one local machine had been set up as a hacker bulletin board, which the hackers used to contact each other and to discuss techniques and progress.

An analysis of this attack revealed that there were actually two levels of hackers. The high level were sophisticated users with a thorough knowledge of the technology; the low level were the "foot soldiers" who merely used the supplied cracking programs with little understanding of how they worked. This teamwork combined the two most serious weapons in the intruder armory: sophisticated knowledge of how to intrude and a willingness to spend countless hours "turning doorknobs" to probe for weaknesses.

One of the results of the growing awareness of the intruder problem has been the establishment of a number of computer emergency response teams (CERTs). These cooperative ventures collect information about system vulnerabilities and disseminate it to systems managers. Unfortunately, hackers can also gain access to CERT reports. In the Texas A&M incident, later analysis showed that the hackers had developed programs to test the attacked machines for virtually every vulnerability that had been announced by CERT. If even one machine had failed to respond promptly to a CERT advisory, it was wide open to such attacks.

In addition to running password-cracking programs, the intruders attempted to modify login software to enable them to capture passwords of users logging on to systems. This made it possible for them to build up an impressive collection of compromised passwords, which was made available on the bulletin board set up on one of the victim's own machines.

In this section, we look at the techniques used for intrusion. Then we examine ways to detect intrusion. Finally, we look at password-based approaches to prevention.

Intrusion Techniques

The objective of the intruder is to gain access to a system or to increase the range of privileges accessible on a system. Generally, this requires the intruder to acquire information that should have been protected. In some cases, this information is in the form of a user password. With knowledge of some other user's password, an intruder can log in to a system and exercise all the privileges accorded to the legitimate user.

Typically, a system must maintain a file that associates a password with each authorized user. If such a file is stored with no protection, then it is an easy matter to gain access to it and learn passwords. The password file can be protected in one of two ways:

- **One-way function:** The system stores only the value of a function based on the user's password. When the user presents a password, the system transforms that password and compares it with the stored value. In practice, the system usually performs a one-way transformation (not reversible) in which the password is used to generate a key for the one-way function and in which a fixed-length output is produced.

- **Access control:** Access to the password file is limited to one or a very few accounts.

If one or both of these countermeasures are in place, some effort is needed for a potential intruder to learn passwords. On the basis of a survey of the literature and interviews with a number of password

crackers, [ALVA90] reports the following techniques for learning passwords:

1. Try default passwords used with standard accounts that are shipped with the system. Many administrators do not bother to change these defaults.
2. Exhaustively try all short passwords (those of one to three characters).
3. Try words in the system's online dictionary or a list of likely passwords. Examples of the latter are readily available on hacker bulletin boards.
4. Collect information about users, such as their full names, the names of their spouse and children, pictures in their office, and books in their office that are related to hobbies.
5. Try users' phone numbers, Social Security numbers, and room numbers.
6. Try all legitimate license plate numbers for this state.
7. Use a Trojan horse (described in Section 18.2) to bypass restrictions on access.
8. Tap the line between a remote user and the host system.

The first six methods are various ways of guessing a password. If an intruder has to verify the guess by attempting to log in, it is a tedious and easily countered means of attack. For example, a system can simply reject any login after three password attempts, thus requiring the intruder to reconnect to the host to try again. Under these circumstances, it is not practical to try more than a handful of passwords. However, the intruder is unlikely to try such crude methods. For example, if an intruder can gain access with a low level of privileges to an encrypted password file, then the strategy would be to capture that file and then use the encryption mechanism of that particular system at leisure until a valid password that provided greater privileges was discovered.

Guessing attacks are feasible, and indeed highly effective, when a large number of guesses can be attempted automatically and each guess verified, without the guessing process being detectable. Later in this chapter, we have much to say about thwarting guessing attacks.

The seventh method of attack listed earlier, the Trojan horse, can be particularly difficult to counter. An

example of a program that bypasses access controls was cited in [ALVA90]. A low-privilege user produced a game program and invited the system operator to use it in his or her spare time. The program did indeed play a game, but in the background it also contained code to copy the password file, which was unencrypted but access protected, into the user's file. Because the game was running under the operator's high-privilege mode, it was able to gain access to the password file.

The eighth attack listed, line tapping, is a matter of physical security. It can be countered with link encryption techniques, discussed in Section 7.1.

Other intrusion techniques do not require learning a password. Intruders can get access to a system by exploiting attacks such as buffer overflows on a program that runs with certain privileges. Privilege escalation can be done this way as well.

We turn now to a discussion of the two principal countermeasures: detection and prevention. Detection is concerned with learning of an attack, either before or after its success. Prevention is a challenging security goal and an uphill battle at all times. The difficulty stems from the fact that the defender must attempt to thwart all possible attacks, whereas the attacker is free to try to find the weakest link in the defense chain and attack at that point.

18.2. Intrusion Detection

Inevitably, the best intrusion prevention system will fail. A system's second line of defense is intrusion detection, and this has been the focus of much research in recent years. This interest is motivated by a number of considerations, including the following:

1. If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised. Even if the detection is not sufficiently timely to preempt the intruder, the sooner that the intrusion is detected, the less the amount of damage and the more quickly that recovery can be achieved.

2. An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions.

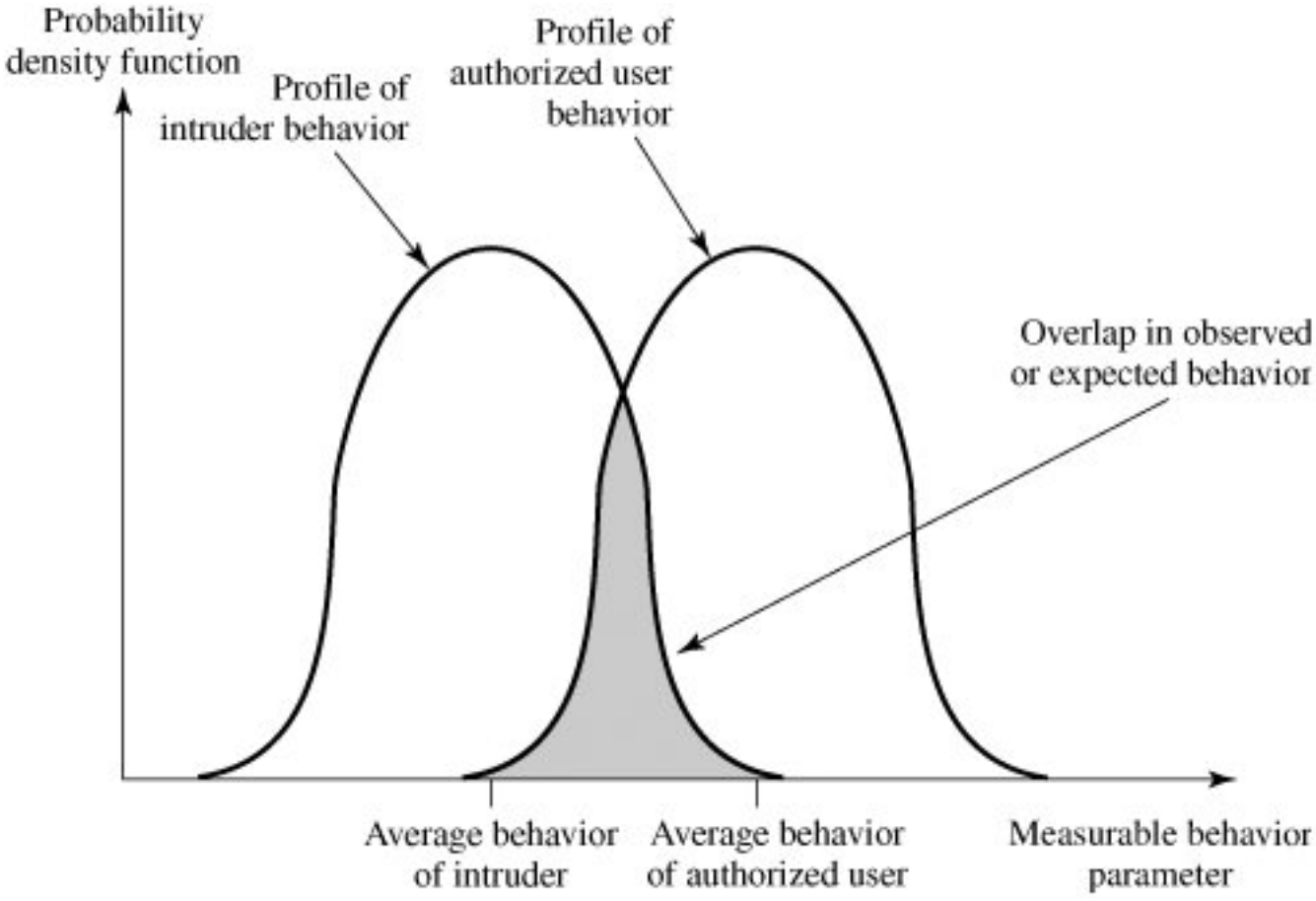
3. Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

Intrusion detection is based on the assumption that the behavior of the intruder differs from that of a legitimate user in ways that can be quantified. Of course, we cannot expect that there will be a crisp, exact distinction between an attack by an intruder and the normal use of resources by an authorized user. Rather, we must expect that there will be some overlap.

Figure 18.1 suggests, in very abstract terms, the nature of the task confronting the designer of an intrusion detection system. Although the typical behavior of an intruder differs from the typical behavior of an authorized user, there is an overlap in these behaviors. Thus, a loose interpretation of intruder behavior, which will catch more intruders, will also lead to a number of "false positives," or authorized users identified as intruders. On the other hand, an attempt to limit false positives by a tight interpretation of intruder behavior will lead to an increase in false negatives, or intruders not identified as intruders. Thus, there is an element of compromise and art in the practice of intrusion detection.

Figure 18.1. Profiles of Behavior of Intruders and Authorized Users

(This item is displayed on page 571 in the print version)



In Anderson's study [ANDE80], it was postulated that one could, with reasonable confidence, distinguish between a masquerader and a legitimate user. Patterns of legitimate user behavior can be established by observing past history, and significant deviation from such patterns can be detected. Anderson suggests that the task of detecting a misfeasor (legitimate user performing in an unauthorized fashion) is more difficult, in that the distinction between abnormal and normal behavior may be small. Anderson concluded that such violations would be undetectable solely through the search for anomalous behavior. However, misfeasor behavior might nevertheless be detectable by intelligent definition of the class of conditions that suggest unauthorized use. Finally, the detection of the clandestine user was felt to be beyond the scope of purely automated techniques. These observations, which were made in 1980, remain true today.

[PORR92] identifies the following approaches to intrusion detection:

1.

Statistical anomaly detection: Involves the collection of data relating to the behavior of legitimate users over a period of time. Then statistical tests are applied to observed behavior to determine with a high level of confidence whether that behavior is not legitimate user behavior.

a.

Threshold detection: This approach involves defining thresholds, independent of user, for the frequency of occurrence of various events.

Profile based: A profile of the activity of each user is developed and used to detect changes in the behavior of individual accounts.

2.

Rule-based detection: Involves an attempt to define a set of rules that can be used to decide that a given behavior is that of an intruder.

a.

Anomaly detection: Rules are developed to detect deviation from previous usage patterns.

b.

Penetration identification: An expert system approach that searches for suspicious behavior.

In a nutshell, statistical approaches attempt to define normal, or expected, behavior, whereas rule-based approaches attempt to define proper behavior.

In terms of the types of attackers listed earlier, statistical anomaly detection is effective against masqueraders, who are unlikely to mimic the behavior patterns of the accounts they appropriate. On the other hand, such techniques may be unable to deal with misfeasors. For such attacks, rule-based approaches may be able to recognize events and sequences that, in context, reveal penetration. In practice, a system may exhibit a combination of both approaches to be effective against a broad range of attacks.

Audit Records

A fundamental tool for intrusion detection is the audit record. Some record of ongoing activity by users must be maintained as input to an intrusion detection system. Basically, two plans are used:

- **Native audit records:** Virtually all multiuser operating systems include accounting software that collects information on user activity. The advantage of using this information is that no additional collection software is needed. The disadvantage is that the native audit records may not contain the needed information or may not contain it in a convenient form.
- **Detection-specific audit records:** A collection facility can be implemented that generates audit records containing only that information required by the intrusion detection system. One advantage of such an approach is that it could be made vendor independent and ported to a variety of systems. The disadvantage is the extra overhead involved in having, in effect, two accounting packages running on a machine.

A good example of detection-specific audit records is one developed by Dorothy Denning [DENN87]. Each audit record contains the following fields:

- **Subject:** Initiators of actions. A subject is typically a terminal user but might also be a process acting on behalf of users or groups of users. All activity arises through commands issued by subjects. Subjects may be grouped into different access classes, and these classes may overlap.
- **Action:** Operation performed by the subject on or with an object; for example, login, read, perform I/O, execute.

- **Object:** Receptors of actions. Examples include files, programs, messages, records, terminals, printers, and user- or program-created structures. When a subject is the recipient of an action, such as electronic mail, then that subject is considered an object. Objects may be grouped by type. Object granularity may vary by object type and by environment. For example, database actions may be audited for the database as a whole or at the record level.
- **Exception-Condition:** Denotes which, if any, exception condition is raised on return.
- **Resource-Usage:** A list of quantitative elements in which each element gives the amount used of some resource (e.g., number of lines printed or displayed, number of records read or written, processor time, I/O units used, session elapsed time).
- **Time-Stamp:** Unique time-and-date stamp identifying when the action took place.

Most user operations are made up of a number of elementary actions. For example, a file copy involves the execution of the user command, which includes doing access validation and setting up the copy, plus the read from one file, plus the write to another file. Consider the command

```
COPY GAME.EXE TO <Library>GAME.EXE
```

issued by Smith to copy an executable file GAME from the current directory to the <Library> directory. The following audit records may be generated:

Smith	execute	<Library>COPY.EXE	0	CPU = 00002	11058721678

Smith	read	<Smith>GAME.EXE	0	RECORDS = 0	11058721679

Smith	execute	<Library>COPY.EXE	write-viol	RECORDS = 0	11058721680

In this case, the copy is aborted because Smith does not have write permission to <Library>.

The decomposition of a user operation into elementary actions has three advantages:

1.

   Because objects are the protectable entities in a system, the use of elementary actions enables an audit of all behavior affecting an object. Thus, the system can detect attempted subversions of access controls (by noting an abnormality in the number of exception conditions returned) and can detect successful subversions by noting an abnormality in the set of objects accessible to the subject.

2.

   Single-object, single-action audit records simplify the model and the implementation.

3.

Because of the simple, uniform structure of the detection-specific audit records, it may be relatively easy to obtain this information or at least part of it by a straightforward mapping from existing native audit records to the detection-specific audit records.

Statistical Anomaly Detection

As was mentioned, statistical anomaly detection techniques fall into two broad categories: threshold detection and profile-based systems. Threshold detection involves counting the number of occurrences of a specific event type over an interval of time. If the count surpasses what is considered a reasonable number that one might expect to occur, then intrusion is assumed.

Threshold analysis, by itself, is a crude and ineffective detector of even moderately sophisticated attacks. Both the threshold and the time interval must be determined. Because of the variability across users, such thresholds are likely to generate either a lot of false positives or a lot of false negatives. However, simple threshold detectors may be useful in conjunction with more sophisticated techniques.

Profile-based anomaly detection focuses on characterizing the past behavior of individual users or related groups of users and then detecting significant deviations. A profile may consist of a set of parameters, so that deviation on just a single parameter may not be sufficient in itself to signal an alert.

The foundation of this approach is an analysis of audit records. The audit records provide input to the intrusion detection function in two ways. First, the designer must decide on a number of quantitative metrics that can be used to measure user behavior. An analysis of audit records over a period of time can be used to determine the activity profile of the average user. Thus, the audit records serve to define typical behavior. Second, current audit records are the input used to detect intrusion. That is, the intrusion detection model analyzes incoming audit records to determine deviation from average behavior.

Examples of metrics that are useful for profile-based intrusion detection are the following:

- **Counter:** A nonnegative integer that may be incremented but not decremented until it is reset by management action. Typically, a count of certain event types is kept over a particular period of time. Examples include the number of logins by a single user during an hour, the number of times a given command is executed during a single user session, and the number of password failures during a minute.
- **Gauge:** A nonnegative integer that may be incremented or decremented. Typically, a gauge is used to measure the current value of some entity. Examples include the number of logical connections assigned to a user application and the number of outgoing messages queued for a user process.
- **Interval timer:** The length of time between two related events. An example is the length of time between successive logins to an account.
- **Resource utilization:** Quantity of resources consumed during a specified period. Examples include the number of pages printed during a user session and total time consumed by a program execution.

Given these general metrics, various tests can be performed to determine whether current activity fits within acceptable limits. [DENN87] lists the following approaches that may be taken:

- Mean and standard deviation
- Multivariate
- Markov process
- Time series

- Operational

The simplest statistical test is to measure the **mean and standard deviation** of a parameter over some historical period. This gives a reflection of the average behavior and its variability. The use of mean and standard deviation is applicable to a wide variety of counters, timers, and resource measures. But these measures, by themselves, are typically too crude for intrusion detection purposes.

A **multivariate** model is based on correlations between two or more variables. Intruder behavior may be characterized with greater confidence by considering such correlations (for example, processor time and resource usage, or login frequency and session elapsed time).

A **Markov process** model is used to establish transition probabilities among various states. As an example, this model might be used to look at transitions between certain commands.

A **time series** model focuses on time intervals, looking for sequences of events that happen too rapidly or too slowly. A variety of statistical tests can be applied to characterize abnormal timing.

Finally, an **operational model** is based on a judgment of what is considered abnormal, rather than an automated analysis of past audit records. Typically, fixed limits are defined and intrusion is suspected for an observation that is outside the limits. This type of approach works best where intruder behavior can be deduced from certain types of activities. For example, a large number of login attempts over a short period suggests an attempted intrusion.

As an example of the use of these various metrics and models, Table 18.1 shows various measures considered or tested for the Stanford Research Institute (SRI) intrusion detection system (IDES) [DENN87, JAVI91, LUNT88].

Table 18.1. Measures That May Be Used for Intrusion Detection

(This item is displayed on page 576 in the print version)

Measure	Model	Type of Intrusion Detected
Login and Session Activity		
Login frequency by day and time	Mean and standard deviation	Intruders may be likely to log in during off-hours.
Frequency of login at different locations	Mean and standard deviation	Intruders may log in from a location that a particular user rarely or never uses.
Time since last login	Operational	Break-in on a "dead" account.
Elapsed time per session	Mean and standard deviation	Significant deviations might indicate masquerader.

Quantity of output to location	Mean and standard deviation	Excessive amounts of data transmitted to remote locations could signify leakage of sensitive data.
Session resource utilization	Mean and standard deviation	Unusual processor or I/O levels could signal an intruder.
Password failures at login	Operational	Attempted break-in by password guessing.
Failures to login from specified terminals	Operational	Attempted break-in.
Command or Program Execution Activity		
Execution frequency	Mean and standard deviation	May detect intruders, who are likely to use different commands, or a successful penetration by a legitimate user, who has gained access to privileged commands.
Program resource utilization	Mean and standard deviation	An abnormal value might suggest injection of a virus or Trojan horse, which performs side-effects that increase I/O or processor utilization.
Execution denials	Operational model	May detect penetration attempt by individual user who seeks higher privileges.
File Access Activity		
Read, write, create, delete frequency	Mean and standard deviation	Abnormalities for read and write access for individual users may signify masquerading or browsing.
Records read, written	Mean and standard deviation	Abnormality could signify an attempt to obtain sensitive data by inference and aggregation.
Failure count for read, write, create, delete	Operational	May detect users who persistently attempt to access unauthorized files.

The main advantage of the use of statistical profiles is that a prior knowledge of security flaws is not required. The detector program learns what is "normal" behavior and then looks for deviations. The approach is not based on system-dependent characteristics and vulnerabilities. Thus, it should be readily portable among a variety of systems.

Rule-Based Intrusion Detection

Rule-based techniques detect intrusion by observing events in the system and applying a set of rules that lead to a decision regarding whether a given pattern of activity is or is not suspicious. In very general terms, we can characterize all approaches as focusing on either anomaly detection or penetration identification, although there is some overlap in these approaches.

Rule-based anomaly detection is similar in terms of its approach and strengths to statistical anomaly detection. With the rule-based approach, historical audit records are analyzed to identify usage patterns

and to generate automatically rules that describe those patterns. Rules may represent past behavior patterns of users, programs, privileges, time slots, terminals, and so on. Current behavior is then observed, and each transaction is matched against the set of rules to determine if it conforms to any historically observed pattern of behavior.

As with statistical anomaly detection, rule-based anomaly detection does not require knowledge of security vulnerabilities within the system. Rather, the scheme is based on observing past behavior and, in effect, assuming that the future will be like the past. In order for this approach to be effective, a rather large database of rules will be needed. For example, a scheme described in [VACC89] contains anywhere from 10^4 to 10^6 rules.

Rule-based penetration identification takes a very different approach to intrusion detection, one based on expert system technology. The key feature of such systems is the use of rules for identifying known penetrations or penetrations that would exploit known weaknesses. Rules can also be defined that identify suspicious behavior, even when the behavior is within the bounds of established patterns of usage. Typically, the rules used in these systems are specific to the machine and operating system. Also, such rules are generated by "experts" rather than by means of an automated analysis of audit records. The normal procedure is to interview system administrators and security analysts to collect a suite of known penetration scenarios and key events that threaten the security of the target system.[1] Thus, the strength of the approach depends on the skill of those involved in setting up the rules.

[1] Such interviews may even extend to reformed or unreformed crackers who will share their expertise for a fee [FREE93].

A simple example of the type of rules that can be used is found in NIDX, an early system that used heuristic rules that can be used to assign degrees of suspicion to activities [BAUE88]. Example heuristics are the following:

1. Users should not read files in other users' personal directories.
2. Users must not write other users' files.
3. Users who log in after hours often access the same files they used earlier.
4. Users do not generally open disk devices directly but rely on higher-level operating system utilities.
5. Users should not be logged in more than once to the same system.

Users do not make copies of system programs.

The penetration identification scheme used in IDES is representative of the strategy followed. Audit records are examined as they are generated, and they are matched against the rule base. If a match is found, then the user's *suspicion rating* is increased. If enough rules are matched, then the rating will pass a threshold that results in the reporting of an anomaly.

The IDES approach is based on an examination of audit records. A weakness of this plan is its lack of flexibility. For a given penetration scenario, there may be a number of alternative audit record sequences that could be produced, each varying from the others slightly or in subtle ways. It may be difficult to pin down all these variations in explicit rules. Another method is to develop a higher-level model independent of specific audit records. An example of this is a state transition model known as USTAT [ILGU93]. USTAT deals in general actions rather than the detailed specific actions recorded by the UNIX auditing mechanism. USTAT is implemented on a SunOS system that provides audit records on 239 events. Of these, only 28 are used by a preprocessor, which maps these onto 10 general actions (Table 18.2). Using just these actions and the parameters that are invoked with each action, a state transition diagram is developed that characterizes suspicious activity. Because a number of different auditable events map into a smaller number of actions, the rule-creation process is simpler. Furthermore, the state transition diagram model is easily modified to accommodate newly learned intrusion behaviors.

Table 18.2. USTAT Actions versus SunOS Event Types

USTAT Action	SunOS Event Type
Read	open_r, open_rc, open_rtc, open_rwc, open_rwtc, open_rt, open_rw, open_rwt
Write	truncate, ftruncate, creat, open_rtc, open_rwc, open_rwtc, open_rt, open_rw, open_rwt, open_w, open_wt, open_wc, open_wct
Create	mkdir, creat, open_rc, open_rtc, open_rwc, open_rwtc, open_wc, open_wtc, mknod
Delete	rmdir, unlink
Execute	exec, execve
Exit	exit
Modify_Owner	chown, fchown
Modify_Perm	chmod, fchmod
Rename	rename
Hardlink	link

The Base-Rate Fallacy

To be of practical use, an intrusion detection system should detect a substantial percentage of intrusions while keeping the false alarm rate at an acceptable level. If only a modest percentage of actual

intrusions are detected, the system provides a false sense of security. On the other hand, if the system frequently triggers an alert when there is no intrusion (a false alarm), then either system managers will begin to ignore the alarms, or much time will be wasted analyzing the false alarms.

Unfortunately, because of the nature of the probabilities involved, it is very difficult to meet the standard of high rate of detections with a low rate of false alarms. In general, if the actual numbers of intrusions is low compared to the number of legitimate uses of a system, then the false alarm rate will be high unless the test is extremely discriminating. A study of existing intrusion detection systems, reported in [AXEL00], indicated that current systems have not overcome the problem of the base-rate fallacy. See Appendix 18A for a brief background on the mathematics of this problem.

Distributed Intrusion Detection

Until recently, work on intrusion detection systems focused on single-system stand-alone facilities. The typical organization, however, needs to defend a distributed collection of hosts supported by a LAN or internetwork. Although it is possible to mount a defense by using stand-alone intrusion detection systems on each host, a more effective defense can be achieved by coordination and cooperation among intrusion detection systems across the network.

Porras points out the following major issues in the design of a distributed intrusion detection system [PORR92]:

- A distributed intrusion detection system may need to deal with different audit record formats. In a heterogeneous environment, different systems will employ different native audit collection systems and, if using intrusion detection, may employ different formats for security-related audit records.

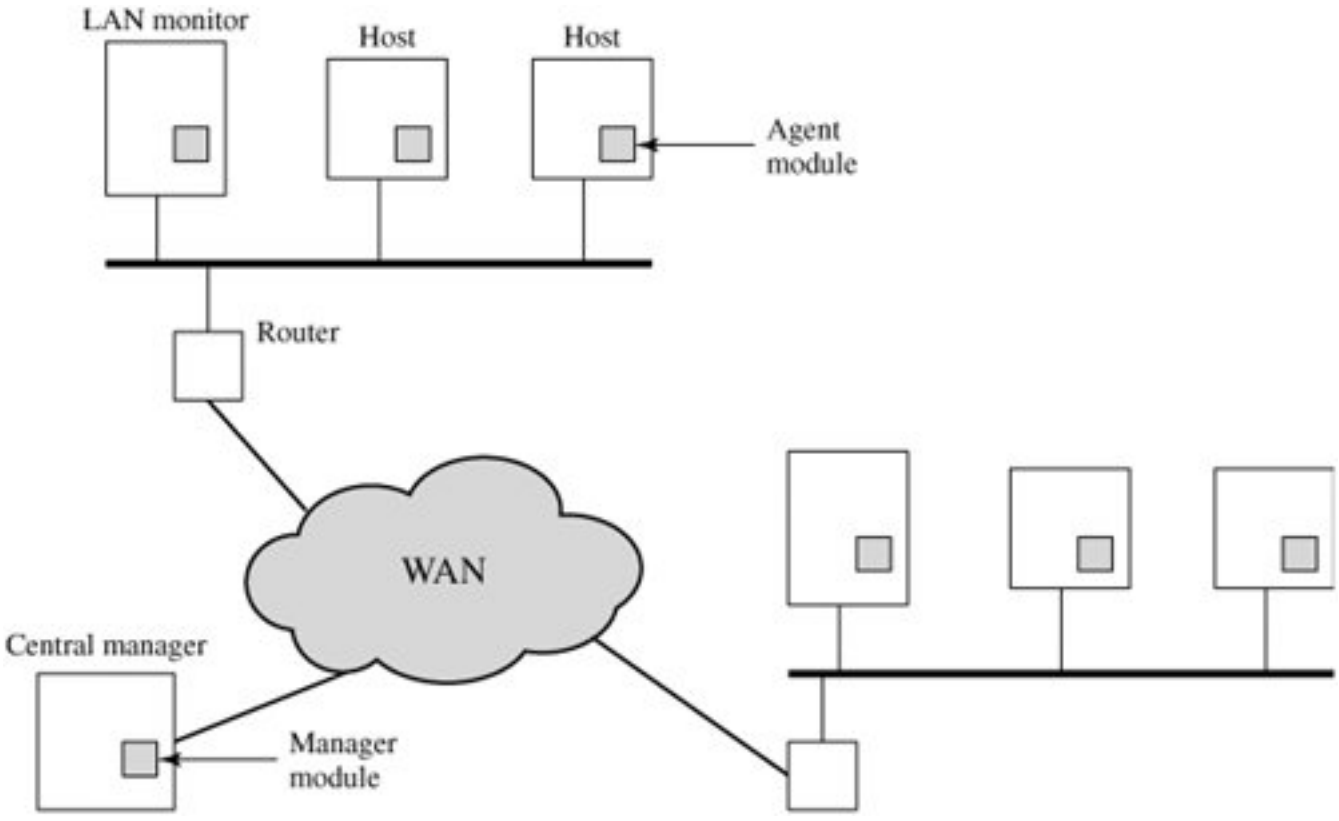
- One or more nodes in the network will serve as collection and analysis points for the data from the systems on the network. Thus, either raw audit data or summary data must be transmitted across the network. Therefore, there is a requirement to assure the integrity and confidentiality of these data. Integrity is required to prevent an intruder from masking his or her activities by altering the transmitted audit information. Confidentiality is required because the transmitted audit information could be valuable.
- Either a centralized or decentralized architecture can be used. With a centralized architecture, there is a single central point of collection and analysis of all audit data. This eases the task of correlating incoming reports but creates a potential bottleneck and single point of failure. With a decentralized architecture, there are more than one analysis centers, but these must coordinate their activities and exchange information.

A good example of a distributed intrusion detection system is one developed at the University of California at Davis [HEBE92, SNAP91]. Figure 18.2 shows the overall architecture, which consists of three main components:

- **Host agent module:** An audit collection module operating as a background process on a monitored system. Its purpose is to collect data on security-related events on the host and transmit these to the central manager.
- **LAN monitor agent module:** Operates in the same fashion as a host agent module except that it analyzes LAN traffic and reports the results to the central manager.
- **Central manager module:** Receives reports from LAN monitor and host agents and processes and correlates these reports to detect intrusion.

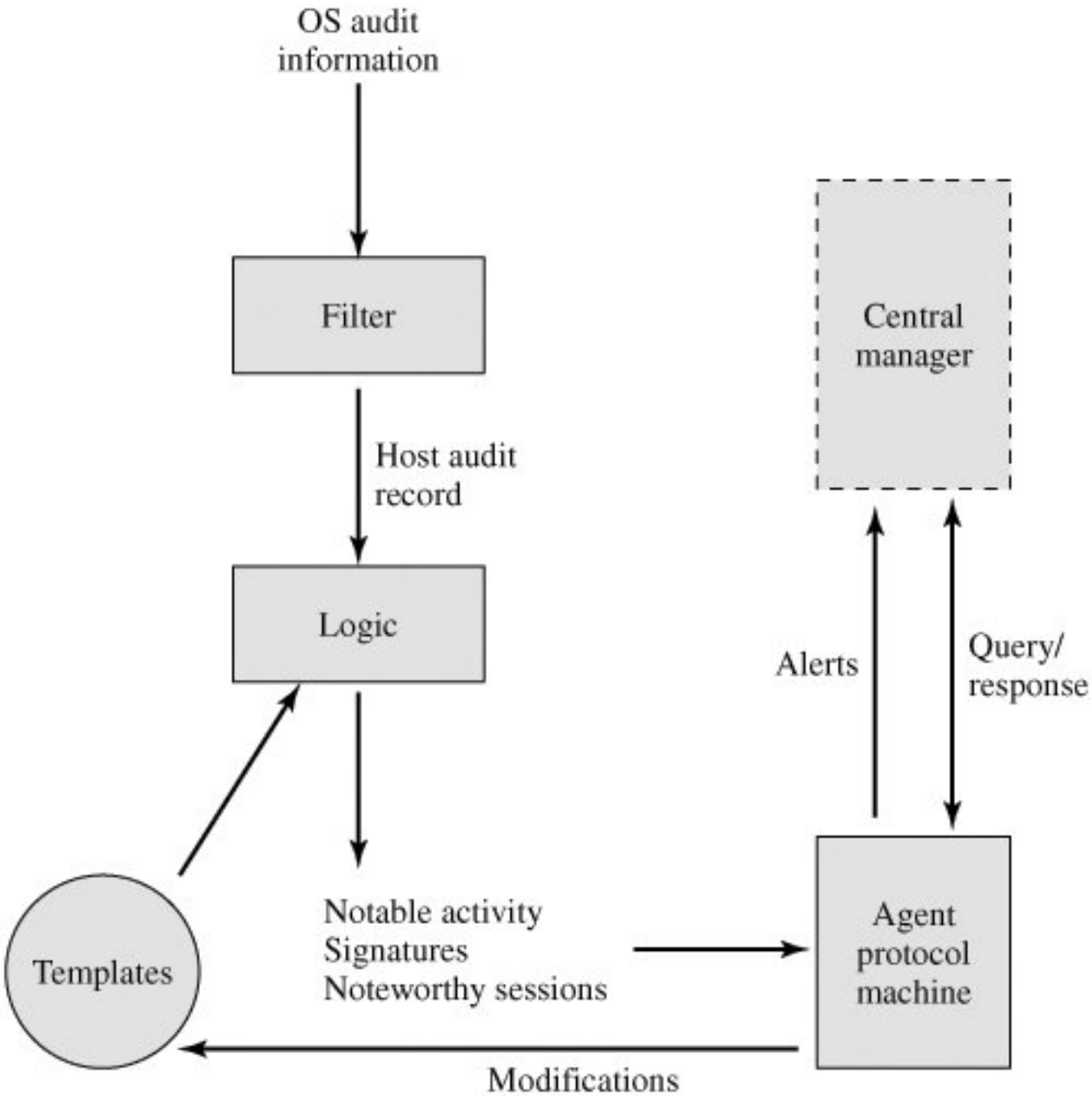
Figure 18.2. Architecture for Distributed Intrusion Detection

[View full size image]



The scheme is designed to be independent of any operating system or system auditing implementation. Figure 18.3 [SNAP91] shows the general approach that is taken. The agent captures each audit record produced by the native audit collection system. A filter is applied that retains only those records that are of security interest. These records are then reformatted into a standardized format referred to as the host audit record (HAR). Next, a template-driven logic module analyzes the records for suspicious activity. At the lowest level, the agent scans for notable events that are of interest independent of any past events. Examples include failed file accesses, accessing system files, and changing a file's access control. At the next higher level, the agent looks for sequences of events, such as known attack patterns (signatures). Finally, the agent looks for anomalous behavior of an individual user based on a historical profile of that user, such as number of programs executed, number of files accessed, and the like.

Figure 18.3. Agent Architecture



When suspicious activity is detected, an alert is sent to the central manager. The central manager includes an expert system that can draw inferences from received data. The manager may also query individual systems for copies of HARs to correlate with those from other agents.

The LAN monitor agent also supplies information to the central manager. The LAN monitor agent audits host-host connections, services used, and volume of traffic. It searches for significant events, such as sudden changes in network load, the use of security-related services, and network activities such as *rlogin*.

The architecture depicted in Figures 18.2 and 18.3 is quite general and flexible. It offers a foundation for a machine-independent approach that can expand from stand-alone intrusion detection to a system that is able to correlate activity from a number of sites and networks to detect suspicious activity that would otherwise remain undetected.

Honeypots

A relatively recent innovation in intrusion detection technology is the honeypot. Honeypots are decoy systems that are designed to lure a potential attacker away from critical systems. Honeypots are designed to

- divert an attacker from accessing critical systems
- collect information about the attacker's activity
- encourage the attacker to stay on the system long enough for administrators to respond

These systems are filled with fabricated information designed to appear valuable but that a legitimate user of the system wouldn't access. Thus, any access to the honeypot is suspect. The system is instrumented with sensitive monitors and event loggers that detect these accesses and collect information about the attacker's activities. Because any attack against the honeypot is made to seem successful, administrators have time to mobilize and log and track the attacker without ever exposing productive systems.

Initial efforts involved a single honeypot computer with IP addresses designed to attract hackers. More recent research has focused on building entire honeypot networks that emulate an enterprise, possibly with actual or simulated traffic and data. Once hackers are within the network, administrators can observe their behavior in detail and figure out defenses.

Intrusion Detection Exchange Format

To facilitate the development of distributed intrusion detection systems that can function across a wide range of platforms and environments, standards are needed to support interoperability. Such standards are the focus of the IETF Intrusion Detection Working Group. The purpose of the working group is to define data formats and exchange procedures for sharing information of interest to intrusion detection and response systems and to management systems that may need to interact with them. The outputs of this working group include the following:

1. A requirements document, which describes the high-level functional requirements for communication between intrusion detection systems and requirements for communication between intrusion detection systems and with management systems, including the rationale for those requirements. Scenarios will be used to illustrate the requirements.

2. A common intrusion language specification, which describes data formats that satisfy the requirements.

3. A framework document, which identifies existing protocols best used for communication between intrusion detection systems, and describes how the devised data formats relate to them.

As of this writing, all of these documents are in an Internet-draft document stage.

18.3. Password Management

Password Protection

The front line of defense against intruders is the password system. Virtually all multiuser systems require that a user provide not only a name or identifier (ID) but also a password. The password serves to authenticate the ID of the individual logging on to the system. In turn, the ID provides security in the following ways:

- The ID determines whether the user is authorized to gain access to a system. In some systems, only those who already have an ID filed on the system are allowed to gain access.
- The ID determines the privileges accorded to the user. A few users may have supervisory or "superuser" status that enables them to read files and perform functions that are especially protected by the operating system. Some systems have guest or anonymous accounts, and users of these accounts have more limited privileges than others.
- The ID is used in what is referred to as discretionary access control. For example, by listing the IDs of the other users, a user may grant permission to them to read files owned by that user.

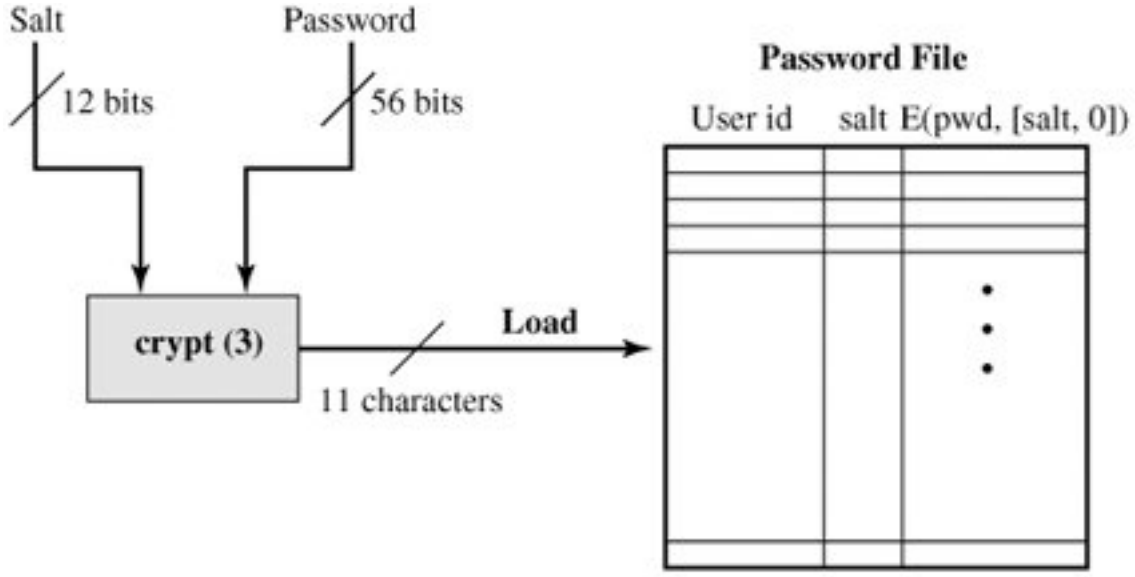
The Vulnerability of Passwords

To understand the nature of the threat to password-based systems, let us consider a scheme that is widely used on UNIX, in which passwords are never stored in the clear. Rather, the following procedure is employed (Figure 18.4a). Each user selects a password of up to eight printable characters in length. This is converted into a 56-bit value (using 7-bit ASCII) that serves as the key input to an encryption routine. The encryption routine, known as crypt(3), is based on DES. The DES algorithm is modified using a 12-bit "salt" value. Typically, this value is related to the time at which the password is assigned to the user. The modified DES algorithm is exercised with a data input consisting of a 64-bit block of zeros. The output of the algorithm then serves as input for a second encryption. This process is repeated for a total of 25 encryptions. The resulting 64-bit output is then translated into an 11-character sequence. The hashed password is then stored, together with a plaintext copy of the salt, in the password file for the corresponding user ID. This method has been shown to be secure against a variety of cryptanalytic attacks [WAGN00].

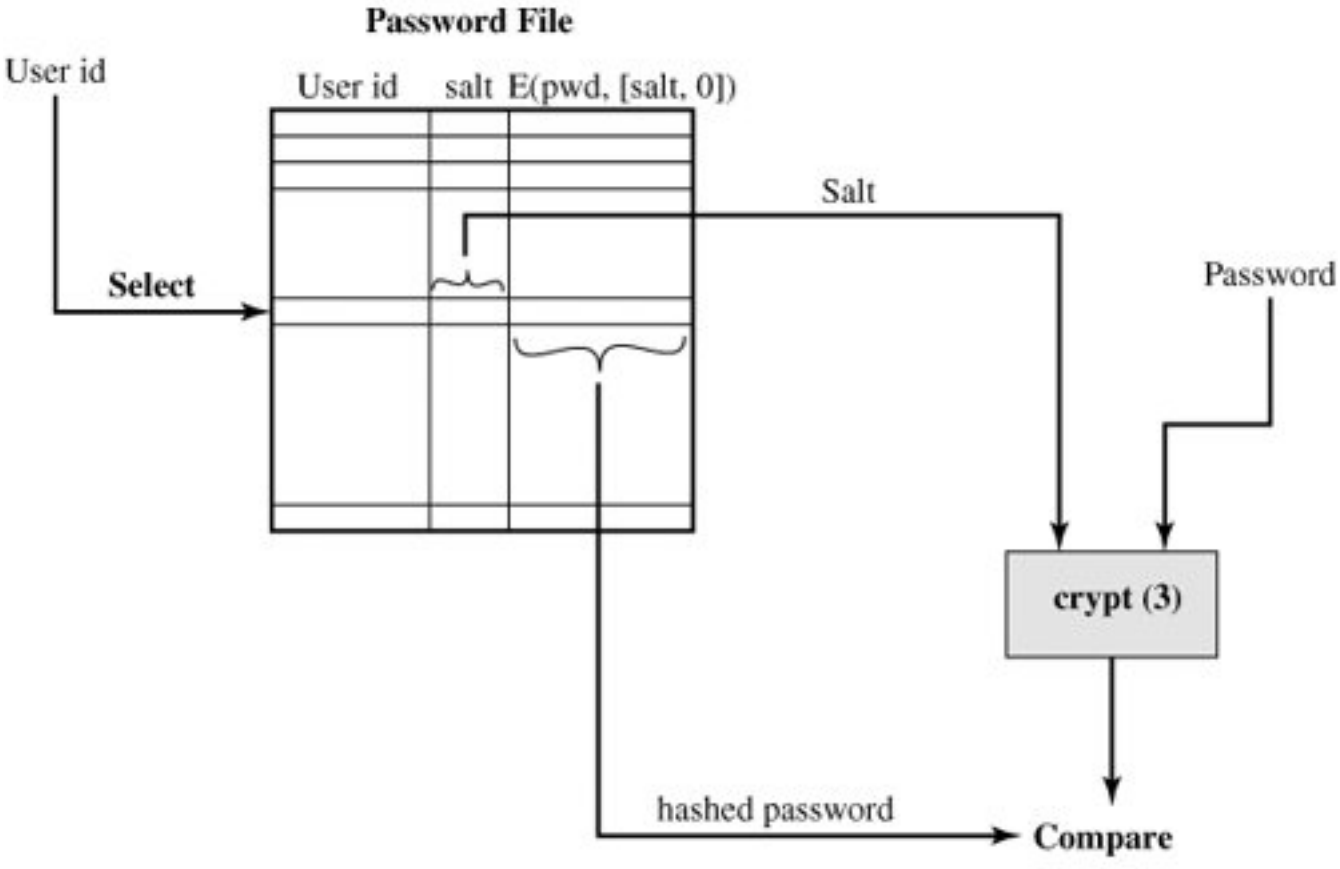
Figure 18.4. UNIX Password Scheme

(This item is displayed on page 583 in the print version)

[View full size image]



(a) Loading a new password



(b) Verifying a password

The salt serves three purposes:

- It prevents duplicate passwords from being visible in the password file. Even if two users choose the same password, those passwords will be assigned at different times. Hence, the "extended" passwords of the two users will differ.
- It effectively increases the length of the password without requiring the user to remember two additional characters. Hence, the number of possible passwords is increased by a factor of 4096, increasing the difficulty of guessing a password.
- It prevents the use of a hardware implementation of DES, which would ease the difficulty of a brute-force guessing attack.

[Page 583]

When a user attempts to log on to a UNIX system, the user provides an ID and a password. The operating system uses the ID to index into the password file and retrieve the plaintext salt and the encrypted password. The salt and user-supplied password are used as input to the encryption routine. If the result matches the stored value, the password is accepted.

The encryption routine is designed to discourage guessing attacks. Software implementations of DES are slow compared to hardware versions, and the use of 25 iterations multiplies the time required by 25. However, since the original design of this algorithm, two changes have occurred. First, newer implementations of the algorithm itself have resulted in speedups. For example, the Internet worm described in Chapter 19 was able to do online password guessing of a few hundred passwords in a reasonably short time by using a more efficient encryption algorithm than the standard one stored on the UNIX systems that it attacked. Second, hardware performance continues to increase, so that any software algorithm executes more quickly.

[Page 584]

Thus, there are two threats to the UNIX password scheme. First, a user can gain access on a machine using a guest account or by some other means and then run a password guessing program, called a password cracker, on that machine. The attacker should be able to check hundreds and perhaps thousands of possible passwords with little resource consumption. In addition, if an opponent is able to obtain a copy of the password file, then a cracker program can be run on another machine at leisure. This enables the opponent to run through many thousands of possible passwords in a reasonable period.

As an example, a password cracker was reported on the Internet in August 1993 [MADS93]. Using a Thinking Machines Corporation parallel computer, a performance of 1560 encryptions per second per vector unit was achieved. With four vector units per processing node (a standard configuration), this works out to 800,000 encryptions per second on a 128-node machine (which is a modest size) and 6.4 million encryptions per second on a 1024-node machine.

Even these stupendous guessing rates do not yet make it feasible for an attacker to use a dumb brute-force technique of trying all possible combinations of characters to discover a password. Instead, password crackers rely on the fact that some people choose easily guessable passwords.

Some users, when permitted to choose their own password, pick one that is absurdly short. The results of one study at Purdue University are shown in Table 18.3. The study observed password change choices on 54 machines, representing approximately 7000 user accounts. Almost 3% of the passwords were three characters or fewer in length. An attacker could begin the attack by exhaustively testing all possible passwords of length 3 or fewer. A simple remedy is for the system to reject any password choice of fewer than, say, six characters or even to require that all passwords be exactly eight characters in length. Most users would not complain about such a restriction.

Table 18.3. Observed Password Lengths [SPAF92a]

Length	Number	Fraction of Total

1	55	.004
2	87	.006
3	212	.02
4	449	.03
5	1260	.09
6	3035	.22
7	2917	.21
8	5772	.42
Total	13787	1.0

Password length is only part of the problem. Many people, when permitted to choose their own password, pick a password that is guessable, such as their own name, their street name, a common dictionary word, and so forth. This makes the job of password cracking straightforward. The cracker simply has to test the password file against lists of likely passwords. Because many people use guessable passwords, such a strategy should succeed on virtually all systems.

One demonstration of the effectiveness of guessing is reported in [KLEI90]. From a variety of sources, the author collected UNIX password files, containing nearly 14,000 encrypted passwords. The result, which the author rightly characterizes as frightening, is shown in Table 18.4. In all, nearly one-fourth of the passwords were guessed. The following strategy was used:

1. Try the user's name, initials, account name, and other relevant personal information. In all, 130 different permutations for each user were tried.

2. Try words from various dictionaries. The author compiled a dictionary of over 60,000 words, including the online dictionary on the system itself, and various other lists as shown.

3. Try various permutations on the words from step 2. This included making the first letter uppercase or a control character, making the entire word uppercase, reversing the word, changing the letter "o" to the digit "zero," and so on. These permutations added another 1 million words to the list.

4. Try various capitalization permutations on the words from step 2 that were not considered in step 3. This added almost 2 million additional words to the list.

Table 18.4. Passwords Cracked from a Sample Set of 13,797 Accounts [KLEI90]

(This item is displayed on page 586 in the print version)

Type of Password	Search Size	Number of Matches	Percentage of Passwords Matched	Cost/Benefit Ratio[a]
User/account name	130	368	2.7%	2.830
Character sequences	866	22	0.2%	0.025
Numbers	427	9	0.1%	0.021
Chinese	392	56	0.4%	0.143
Place names	628	82	0.6%	0.131
Common names	2239	548	4.0%	0.245
Female names	4280	161	1.2%	0.038
Male names	2866	140	1.0%	0.049
Uncommon names	4955	130	0.9%	0.026
Myths & legends	1246	66	0.5%	0.053
Shakespearean	473	11	0.1%	0.023
Sports terms	238	32	0.2%	0.134
Science fiction	691	59	0.4%	0.085
Movies and actors	99	12	0.1%	0.121
Cartoons	92	9	0.1%	0.098
Famous people	290	55	0.4%	0.190
Phrases and patterns	933	253	1.8%	0.271
Surnames	33	9	0.1%	0.273
Biology	58	1	0.0%	0.017
System dictionary	19683	1027	7.4%	0.052
Machine names	9018	132	1.0%	0.015
Mnemonics	14	2	0.0%	0.143
King James bible	7525	83	0.6%	0.011

Miscellaneous words	3212	54	0.4%	0.017
Yiddish words	56	0	0.0%	0.000
Asteroids	2407	19	0.1%	0.007
TOTAL	62727	3340	24.2%	0.053

[a] Computed as the number of matches divided by the search size. The more words that needed to be tested for a match, the lower the cost/benefit ratio.

Thus, the test involved in the neighborhood of 3 million words. Using the fastest Thinking Machines implementation listed earlier, the time to encrypt all these words for all possible salt values is under an hour. Keep in mind that such a thorough search could produce a success rate of about 25%, whereas even a single hit may be enough to gain a wide range of privileges on a system.

Access Control

One way to thwart a password attack is to deny the opponent access to the password file. If the encrypted password portion of the file is accessible only by a privileged user, then the opponent cannot read it without already knowing the password of a privileged user. [SPAF92a] points out several flaws in this strategy:

- Many systems, including most UNIX systems, are susceptible to unanticipated break-ins. Once an attacker has gained access by some means, he or she may wish to obtain a collection of passwords in order to use different accounts for different logon sessions to decrease the risk of detection. Or a user with an account may desire another user's account to access privileged data or to sabotage the system.
- An accident of protection might render the password file readable, thus compromising all the accounts.
- Some of the users have accounts on other machines in other protection domains, and they use the same password. Thus, if the passwords could be read by anyone on one machine, a machine in another location might be compromised.

Thus, a more effective strategy would be to force users to select passwords that are difficult to guess.

Password Selection Strategies

The lesson from the two experiments just described (Tables 18.3 and 18.4) is that, left to their own devices, many users choose a password that is too short or too easy to guess. At the other extreme, if users are assigned passwords consisting of eight randomly selected printable characters, password cracking is effectively impossible. But it would be almost as impossible for most users to remember their passwords. Fortunately, even if we limit the password universe to strings of characters that are reasonably memorable, the size of the universe is still too large to permit practical cracking. Our goal, then, is to eliminate guessable passwords while allowing the user to select a password that is memorable. Four basic techniques are in use:

- User education
- Computer-generated passwords

- Reactive password checking
- Proactive password checking

Users can be told the importance of using hard-to-guess passwords and can be provided with guidelines for selecting strong passwords. This **user education** strategy is unlikely to succeed at most installations, particularly where there is a large user population or a lot of turnover. Many users will simply ignore the guidelines. Others may not be good judges of what is a strong password. For example, many users (mistakenly) believe that reversing a word or capitalizing the last letter makes a password unguessable.

Computer-generated passwords also have problems. If the passwords are quite random in nature, users will not be able to remember them. Even if the password is pronounceable, the user may have difficulty remembering it and so be tempted to write it down. In general, computer-generated password schemes have a history of poor acceptance by users. FIPS PUB 181 defines one of the best-designed automated password generators. The standard includes not only a description of the approach but also a complete listing of the C source code of the algorithm. The algorithm generates words by forming pronounceable syllables and concatenating them to form a word. A random number generator produces a random stream of characters used to construct the syllables and words.

A **reactive password checking** strategy is one in which the system periodically runs its own password cracker to find guessable passwords. The system cancels any passwords that are guessed and notifies the user. This tactic has a number of drawbacks. First, it is resource intensive if the job is done right. Because a determined opponent who is able to steal a password file can devote full CPU time to the task for hours or even days, an effective reactive password checker is at a distinct disadvantage. Furthermore, any existing passwords remain vulnerable until the reactive password checker finds them.

The most promising approach to improved password security is a **proactive password checker**. In this scheme, a user is allowed to select his or her own password. However, at the time of selection, the system checks to see if the password is allowable and, if not, rejects it. Such checkers are based on the philosophy that, with sufficient guidance from the system, users can select memorable passwords from a fairly large password space that are not likely to be guessed in a dictionary attack.

The trick with a proactive password checker is to strike a balance between user acceptability and strength. If the system rejects too many passwords, users will complain that it is too hard to select a password. If the system uses some simple algorithm to define what is acceptable, this provides guidance to password crackers to refine their guessing technique. In the remainder of this subsection, we look at possible approaches to proactive password checking.

The first approach is a simple system for rule enforcement. For example, the following rules could be enforced:

- All passwords must be at least eight characters long.
- In the first eight characters, the passwords must include at least one each of uppercase, lowercase, numeric digits, and punctuation marks.

These rules could be coupled with advice to the user. Although this approach is superior to simply educating users, it may not be sufficient to thwart password crackers. This scheme alerts crackers as to which passwords *not* to try but may still make it possible to do password cracking.

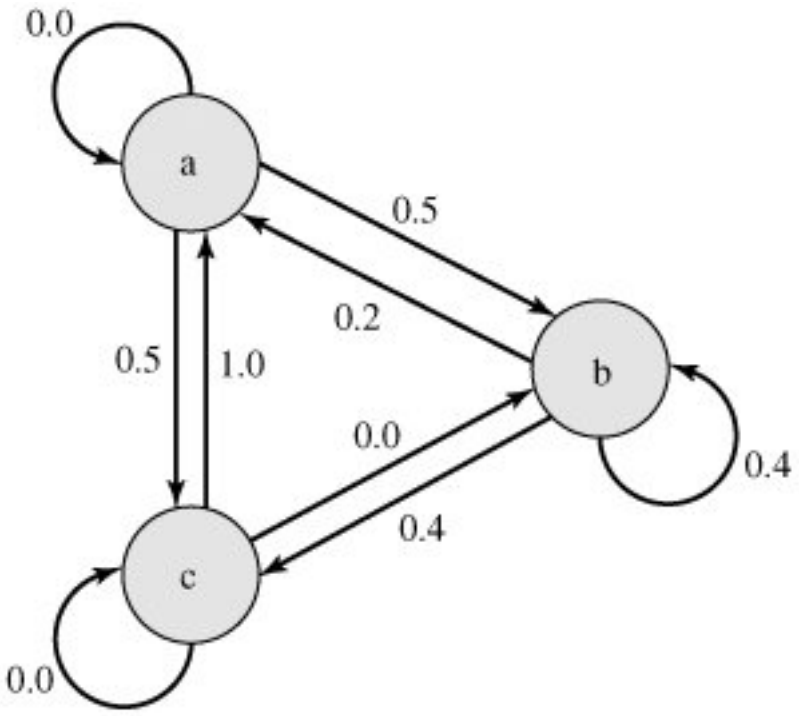
Another possible procedure is simply to compile a large dictionary of possible "bad" passwords. When a

user selects a password, the system checks to make sure that it is not on the disapproved list. There are two problems with this approach:

- **Space:** The dictionary must be very large to be effective. For example, the dictionary used in the Purdue study [SPAF92a] occupies more than 30 megabytes of storage.
- **Time:** The time required to search a large dictionary may itself be large. In addition, to check for likely permutations of dictionary words, either those words most be included in the dictionary, making it truly huge, or each search must also involve considerable processing.

Two techniques for developing an effective and efficient proactive password checker that is based on rejecting words on a list show promise. One of these develops a Markov model for the generation of guessable passwords [DAVI93]. Figure 18.5 shows a simplified version of such a model. This model shows a language consisting of an alphabet of three characters. The state of the system at any time is the identity of the most recent letter. The value on the transition from one state to another represents the probability that one letter follows another. Thus, the probability that the next letter is b, given that the current letter is a, is 0.5.

Figure 18.5. An Example Markov Model



$M = \{3, \{a, b, c\}, T, 1\}$ where

$$T = \begin{bmatrix} 0.0 & 0.5 & 0.5 \\ 0.2 & 0.4 & 0.4 \\ 1.0 & 0.0 & 0.0 \end{bmatrix}$$

e.g., string probably from this language: abbcacaba

e.g., string probably not from this language: aacccbaaa

In general, a Markov model is a quadruple [*m*, *A*, **T**, *k*], where *m* is the number of states in the model, *A* is the state space, **T** is the matrix of transition probabilities, and *k* is the order of the model. For a *k*th-order model, the probability of making a transition to a particular letter depends on the previous *k* letters that have been generated. Figure 18.5 shows a simple first-order model.

The authors report on the development and use of a second-order model. To begin, a dictionary of guessable passwords is constructed. Then the transition matrix is calculated as follows:

1. Determine the frequency matrix **f**, where **f**(*i*, *j*, *k*) is the number of occurrences of the trigram consisting of the *i*th, *j*th, and *k*th character. For example, the password *parsnips* yields the trigrams par, ars, rsn, sni, nip, and ips.

2. For each bigram *ij*, calculate **f**(*i*, *j*, ∞) as the total number of trigrams beginning with *ij*. For example, **f**(a, b, ∞) would be the total number of trigrams of the form aba, abb, abc, and so on.

3. Compute the entries of **T** as follows:

$$\mathbf{T}(i,j,k) = \frac{\mathbf{f}(i,j,k)}{\mathbf{f}(i,j,\infty)}$$

The result is a model that reflects the structure of the words in the dictionary. With this model, the question "Is this a bad password?" is transformed into the question "Was this string (password) generated by this Markov model?" For a given password, the transition probabilities of all its trigrams can be looked up. Some standard statistical tests can then be used to determine if the password is likely or unlikely for that model. Passwords that are likely to be generated by the model are rejected. The authors report good results for a second-order model. Their system catches virtually all the passwords in their dictionary and does not exclude so many potentially good passwords as to be user unfriendly.

A quite different approach has been reported by Spafford [SPAF92a, SPAF92b]. It is based on the use of a Bloom filter [BLOO70]. To begin, we explain the operation of the Bloom filter. A Bloom filter of order *k* consists of a set of *k* independent hash functions $H_1(x)$, $H_2(x)$,..., $H_k(x)$, where each function maps a password into a hash value in the range 0 to *N* - 1 That is,

$$H_i(X_j) = y \quad 1 \leq i \leq k; \quad 1 \leq j \leq D; \quad 0 \leq y \leq N\ 1$$

where

X_j = *j*th word in password dictionary

D = number of words in password dictionary

The following procedure is then applied to the dictionary:

1 A hash table of N bits is defined, with all bits initially set to 0.

2 For each password, its k hash values are calculated, and the corresponding bits in the hash table are set to 1. Thus, if $H_i(X_j) = 67$ for some (i, j), then the sixty-seventh bit of the hash table is set to 1; if the bit already has the value 1, it remains at 1.

When a new password is presented to the checker, its k hash values are calculated. If all the corresponding bits of the hash table are equal to 1, then the password is rejected. All passwords in the dictionary will be rejected. But there will also be some "false positives" (that is, passwords that are not in the dictionary but that produce a match in the hash table). To see this, consider a scheme with two hash functions. Suppose that the passwords *undertaker* and *hulkhogan* are in the dictionary, but *xG% #jj98* is not. Further suppose that

H_1(undertaker) = 25 H_1(hulkhogan) = 83 H_1(xG%#jj98) = 665

H_2(undertaker) = 998 H_2(hulkhogan) = 665 H_2(xG%#jj98) = 998

If the password xG%#jj98 is presented to the system, it will be rejected even though it is not in the dictionary. If there are too many such false positives, it will be difficult for users to select passwords. Therefore, we would like to design the hash scheme to minimize false positives. It can be shown that the probability of a false positive can be approximated by

$$P \approx (1 - e^{kD/N})^k = (1 - e^{k/R})^k$$

or, equivalently,

$$R \approx \frac{-k}{\ln(1 - P^{1/k})}$$

where

k = number of hash functions

N = number of bits in hash table

D = number of words in dictionary

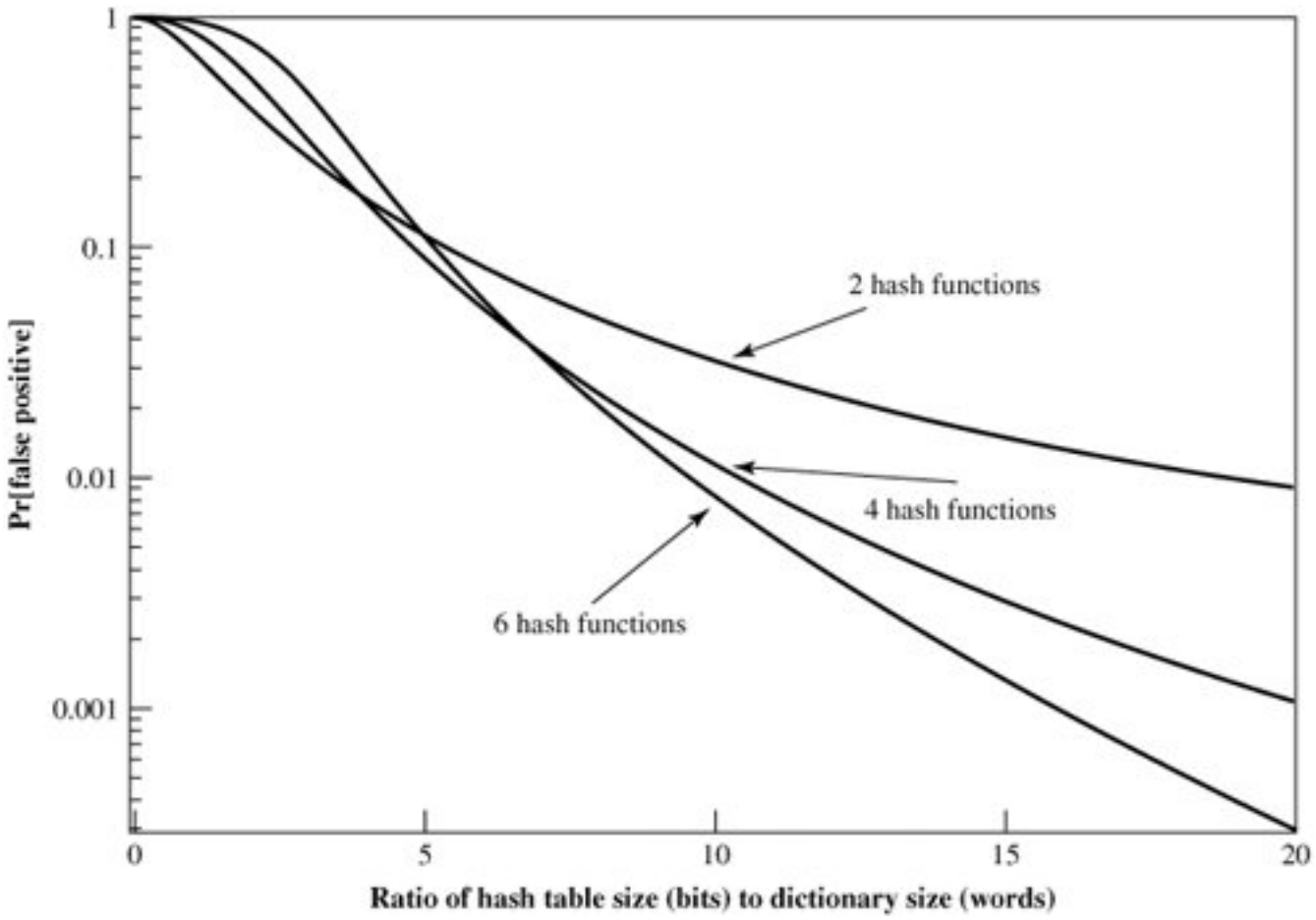
$R = N/D$, ratio of hash table size (bits) to dictionary size (words)

Figure 18.6 plots P as a function of R for various values of k. Suppose we have a dictionary of 1 million words and we wish to have a 0.01 probability of rejecting a password not in the dictionary. If we choose six hash functions, the required ratio is $R = 9.6$. Therefore, we need a hash table of 9.6×10^6 bits or about 1.2 MBytes of storage. In contrast, storage of the entire dictionary would require on the order of 8 MBytes. Thus, we achieve a compression of almost a factor of 7. Furthermore, password checking involves the straightforward calculation of six hash functions and is independent of the size of the dictionary, whereas with the use of the full dictionary, there is substantial searching.[2]

[2] Both the Markov model and the Bloom filter involve the use of probabilistic techniques. In the case of the Markov model, there is a small probability that some passwords in the dictionary will not be caught and a small probability that some passwords not in the dictionary will be rejected. In the case of the Bloom filter, there is a small probability that some passwords not in the dictionary will be rejected. Again we see that taking a probabilistic approach simplifies the solution (e.g., see footnote 1 in Chapter 15).

Figure 18.6. Performance of Bloom Filter

[View full size image]



18.4. Recommended Reading and Web Sites

Two thorough treatments of intrusion detection are [BACE00] and [PROC01]. A more concise but very worthwhile treatment is [BACE01]. Two short but useful survey articles on the subject are [KENT00] and [MCHU00]. [NING04] surveys recent advances in intrusion detection techniques. [HONE01] is the definitive account on honeypots and provides a detailed analysis of the tools and methods of hackers.

BACE00 Bace, R. *Intrusion Detection.* Indianapolis, IN: Macmillan Technical Publishing, 2000.

BACE01 Bace, R., and Mell, P. *Intrusion Detection Systems*. NIST Special Publication SP 800-31, November 2000.

HONE01 The Honeynet Project. *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community.* Reading, MA: Addison-Wesley, 2001.

KENT00 Kent, S. "On the Trail of Intrusions into Information Systems." *IEEE Spectrum*, December 2000.

MCHU00 McHugh, J.; Christie, A.; and Allen, J. "The Role of Intrusion Detection Systems." *IEEE Software*, September/October 2000.

NING04 Ning, P., et al. "Techniques and Tools for Analyzing Intrusion Alerts." *ACM Transactions on Information and System Security*, May 2004.

PROC01 Proctor, P., *The Practical Intrusion Detection Handbook*. Upper Saddle River, NJ: Prentice Hall, 2001.

Recommended Web Sites

- **CERT Coordination Center:** The organization that grew from the computer emergency response team formed by the Defense Advanced Research Projects Agency. Site provides good information on Internet security threats, vulnerabilities, and attack statistics.
- **Honeynet Project:** A research project studying the techniques of predatory hackers and developing honeypot products.
- **Honeypots**: A good collection of research papers and technical articles.

- **Intrusion Detection Working Group:** Includes all of the documents generated by this group.

PREV NEXT

18.5. Key Terms, Review Questions, and Problems

Key Terms

audit record

Bayes' Theorem

base-rate fallacy

honeypot

intruder

intrusion detection

intrusion detection exchange format

password

rule-based intrusion detection

salt

statistical anomaly detection

Review Questions

18.1 List and briefly define three classes of intruders.

18.2 What are two common techniques used to protect a password file?

18.3 What are three benefits that can be provided by an intrusion detection system?

18.4 What is the difference between statistical anomaly detection and rule-based intrusion detection?

18.5 What metrics are useful for profile-based intrusion detection?

18.6 What is the difference between rule-based anomaly detection and rule-based penetration identification?

18.7 What is a honeypot?

18.8 What is a salt in the context of UNIX password management?

18.9 List and briefly define four techniques used to avoid guessable passwords.

Problems

18.1 A taxicab was involved in a fatal hit-and-run accident at night. Two cab companies, the Green and the Blue, operate in the city. You are told that

- 85% of the cabs in the city are Green and 15% are Blue.
- A witness identified the cab as Blue.

The court tested the reliability of the witness under the same circumstances that existed on the night of the accident and concluded that the witness was correct in identifying the color of the cab 80% of the time. What is the probability that the cab involved in the incident was Blue rather than Green?

18.2 Assume that passwords are selected from four-character combinations of 26 alphabetic characters. Assume that an adversary is able to attempt passwords at a rate of one per second.

a.

Assuming no feedback to the adversary until each attempt has been completed, what is the expected time to discover the correct password?

b.

Assuming feedback to the adversary flagging an error as each incorrect character is entered, what is the expected time to discover the correct password?

18.3 Assume that source elements of length k is mapped in some uniform fashion into a target elements of length p. If each digit can take on one of r values, then the number of source elements is r^k and the number of target elements is the smaller number r^p A particular source element x_i is mapped to a particular target element y_j.

a.

What is the probability that the correct source element can be selected by an adversary on one try?

b.

What is the probability that a different source element $x_k (x_i \neq x_k)$ that results in the same target element, y_j, could be produced by an adversary?

c.

What is the probability that the correct target element can be produced by an adversary on one try?

18.4 A phonetic password generator picks two segments randomly for each six-letter password. The form of each segment is CVC (consonant, vowel, consonant), where V = <a, e, i, o, u> and $C = \overline{V}$

a.

What is the total password population?

b.

What is the probability of an adversary guessing a password correctly?

18.5 Assume that passwords are limited to the use of the 95 printable ASCII characters and that all passwords are 10 characters in length. Assume a password cracker with an encryption rate of 6.4 million encryptions per second. How long will it take to test exhaustively all possible passwords on a UNIX system?

18.6 Because of the known risks of the UNIX password system, the SunOS-4.0 documentation recommends that the password file be removed and replaced with a publicly readable file called /etc/publickey. An entry in the file for user A consists of a user's identifier ID_A, the user's public key, PU_a, and the corresponding private key PR_a. This private key is encrypted using DES with a key derived from the user's login password P_a. When A logs in, the system decrypts E[Pa,PR_a] to obtain PR_a.

a. The system then verifies that P_a was correctly supplied. How?

b. How can an opponent attack this system?

18.7 The encryption scheme used for UNIX passwords is one way; it is not possible to reverse it. Therefore, would it be accurate to say that this is, in fact, a hash code rather than an encryption of the password?

18.8 It was stated that the inclusion of the salt in the UNIX password scheme increases the difficulty of guessing by a factor of 4096. But the salt is stored in plaintext in the same entry as the corresponding ciphertext password. Therefore, those two characters are known to the attacker and need not be guessed. Why is it asserted that the salt increases security?

18.9 Assuming that you have successfully answered the preceding problem and understand the significance of the salt, here is another question. Wouldn't it be possible to thwart completely all password crackers by dramatically increasing the salt size to, say, 24 or 48 bits?

18.10 Consider the Bloom filter discussed in Section 18.3. Define k = number of hash functions; N = number of bits in hash table; and D = number of words in dictionary.

---

a. Show that the expected number of bits in the hash table that are equal to zero is expressed as

$$\phi = \left(1 - \frac{k}{N}\right)^D$$

b. Show that the probability that an input word, not in the dictionary, will be

falsely accepted as being in the dictionary is

$$P = (1-\phi)^k$$

c.

Show that the preceding expression can be approximated as

$$P \approx (1 - e^{-KD/N})^k$$

18.11 Design a file access system to allow certain users read and write access to a file, depending on authorization set up by the system. The instructions should be of the format

READ (F, User A): attempt by User A to read file F

WRITE (F, User A): attempt by User A to store a possibly modified copy of F

Each file has a *header record,* which contains authorization privileges; that is, a list of users who can read and write. The file is to be encrypted by a key that is not shared by the users but known only to the system.

PREV NEXT

Appendix 18A The Base-Rate Fallacy

We begin with a review of important results from probability theory, then demonstrate the base-rate fallacy.

Conditional Probability and Independence

We often want to know a probability that is conditional on some event. The effect of the condition is to remove some of the outcomes from the sample space. For example, what is the probability of getting a sum of 8 on the roll of two dice, if we know that the face of at least one die is an even number? We can reason as follows. Because one die is even and the sum is even, the second die must show an even number. Thus, there are three equally likely successful outcomes: (2, 6), (4, 4) and (6, 2), out of a total set of possibilities of [36 - (number of events with both faces odd)] = 36 - 3 x 3 = 27. The resulting probability is 3/27 = 1/9.

Formally, the **conditional probability** of an event *A* assuming the event *B* has occurred, denoted by Pr [*A*|*B*]is defined as the ratio

$$\Pr[A|B] = \frac{\Pr[AB]}{\Pr[B]}$$

where we assume Pr[*B*] is not zero.

In our example, *A* = {sum of 8} and *B* = {at least one die even}. The quantity Pr[*AB*] encompasses all of those outcomes in which the sum is 8 and at least one die is even. As we have seen, there are three such outcomes. Thus, Pr[*AB*] = 3/36 = 1/12. A moment's thought should convince you that Pr[*B*] = 3/4. We can now calculate

---

$$\Pr[A|B] = \frac{1/12}{3/4} = \frac{1}{9}$$

This agrees with our previous reasoning.

Two events *A* and *B* are called **independent** if Pr[*AB*] = Pr[*A*]Pr[*B*]. It can easily be seen that if *A* and *B* are independent, Pr[*A*|*B*] = Pr[*A*] and Pr[*B*|*A*] = Pr[*B*].

Bayes' Theorem

One of the most important results from probability theory is known as Bayes' theorem. First we need to

state the total probability formula. Given a set of mutually exclusive events E_1, E_2,... E_n such that the union of these events covers all possible outcomes, and given an arbitrary event A, then it can be shown that

Equation 18-1

$$\Pr[A] = \sum_{i=1}^{n} \Pr[A|E_i]\Pr[E_i]$$

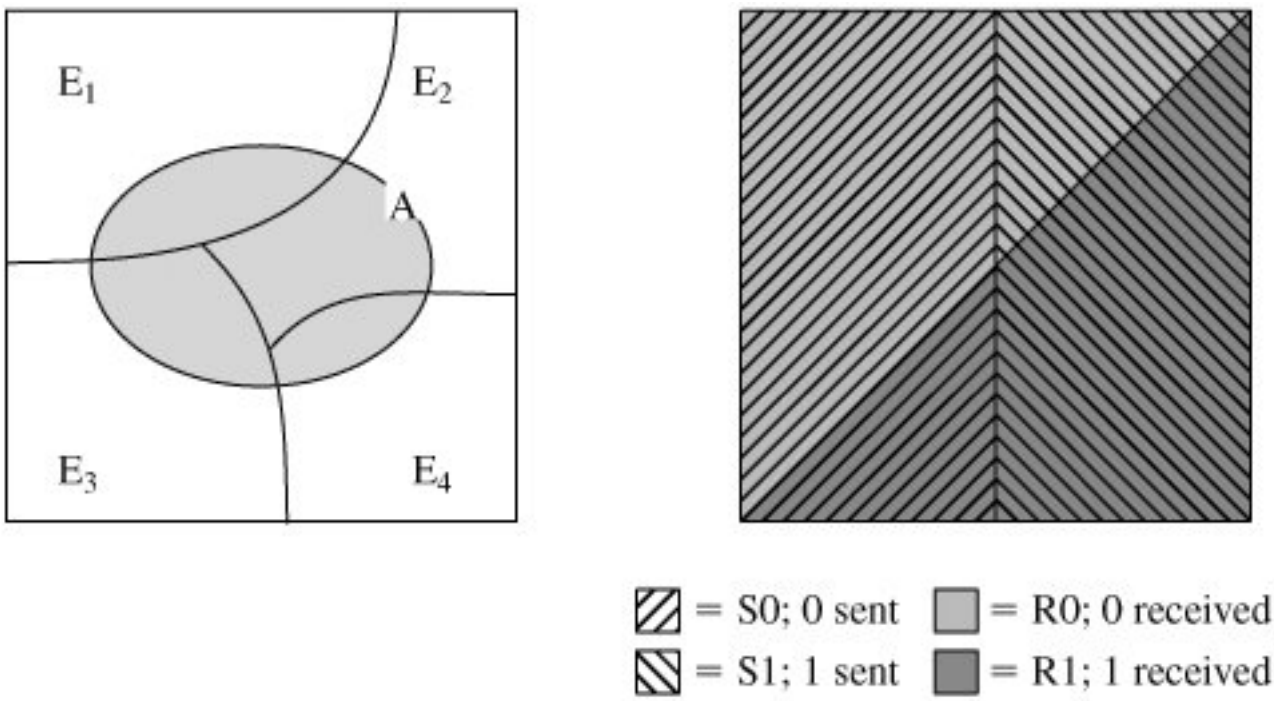
Bayes' theorem may be stated as follows:

Equation 18-2

$$\Pr[E_i|A] = \frac{\Pr[A|E_i]\mathrm{P}[E_i]}{\Pr[A]} = \frac{\Pr[A|E_i]\mathrm{P}[E_i]}{\sum_{j=1}^{n} \Pr[A|E_j]\Pr[E_j]}$$

Figure 18.7a illustrates the concepts of total probability and Bayes' theorem.

Figure 18.7. Illustration of Total Probability and Bayes' Theorem



(a) Diagram to illustrate concepts (b) Example

Bayes' theorem is used to calculate "posterior odds," that is, the probability that something really is the case, given evidence in favor of it. For example, suppose we are transmitting a sequence of zeroes and

ones over a noisy transmission line. Let S0 and S1 be the events a zero is sent at a given time and a one is sent, respectively, and R0 and R1 be the events that a zero is received and a one is received. Suppose we know the probabilities of the source, namely Pr[S1] = p and Pr[S0] = 1 p. Now the line is observed to determine how frequently an error occurs when a one is sent and when a zero is sent, and the following probabilities are calculated: Pr[R0|S1] = p_a and Pr[R1|S0] = p_b. If a zero is received, we can then calculate the conditional probability of an error, namely the conditional probability that a one was sent given that a zero was received, using Bayes' theorem:

$$\Pr[S1|R0] = \frac{\Pr[R0|S1]\Pr[S1]}{\Pr[R0|S1]\Pr[S1] + \Pr[R0|S0]\Pr[S0]} = \frac{p_a p}{p_a p + (1 - p_b)(1 - p)}$$

Figure 18.7b illustrates the preceding equation. In the figure, the sample space is represented by a unit square. Half of the square corresponds to S0 and half to S1, so Pr[S0] = Pr[S1] = 0.5. Similarly, half of the square corresponds to R0 and half to R1, so Pr[R0] = Pr[R1] = 0.5. Within the area representing S0, 1/4 of that area corresponds to R1, so Pr[R1/S0] = 0.25. Other conditional probabilities are similarly evident.

The Base-Rate Fallacy Demonstrated

Consider the following situation. A patient has a test for some disease that comes back positive (indicating he has the disease). You are told that

- The accuracy of the test is 87% (i.e., if a patient has the disease, 87% of the time, the test yields the correct result, and if the patient does not have the disease, 87% of the time, the test yields the correct result).
- The incidence of the disease in the population is 1%.

Given that the test is positive, how probable is it that the patient does not have the disease? That is, what is the probability that this is a false alarm? We need Bayes' theorem to get the correct answer:

$$\Pr[\text{well/positive}] = \frac{\Pr[\text{positive/well}]\Pr[\text{well}]}{\Pr[\text{positive/disease}]\Pr[\text{disease}] + \Pr[\text{positive/well}]\Pr[\text{well}]}$$
$$= \frac{(0.13)(0.99)}{(0.87)(0.01) + (0.13)(0.99)} = 0.937$$

Thus, in the vast majority of cases, when a disease condition is detected, it is a false alarm.

This problem, used in a study [PIAT91], was presented to a number of people. Most subjects gave the answer 13%. The vast majority, including many physicians, gave a number below 50%. Many physicians who guessed wrong lamented, "If you are right, there is no point in making clinical tests!" The reason most people get it wrong is that they do not take into account the basic rate of incidence (the base rate) when intuitively solving the problem. This error is known as the *base-rate fallacy*.

How could this problem be fixed? Suppose we could drive both of the correct result rates to 99.9%. That is, suppose we have Pr[positive/disease] = 0.999 and Pr[negative/well] = 0.999. Plugging these

numbers into the Equation (18.2), we get Pr[well/positive] = 0.09. Thus, if we can accurately detect disease and accurately detect lack of disease at a level of 99.9%, then the rate of false alarms will be 9%. This is much better, but still not ideal. Moreover, again assume 99.9% accuracy, but now suppose that the incidence of the disease in the population is only 1/10000 = 0.0001. We then end up with a rate of false alarms of 91%. In actual situations, [AXEL00] found that the probabilities associated with intrusion detection systems were such that the false alarm rate was unsatisfactory.

PREV NEXT

Chapter 19. Malicious Software

What is the concept of defense: The parrying of a blow. What is its characteristic feature: Awaiting the blow.

On War, Carl Von Clausewitz

Key Points

- Malicious software is software that is intentionally included or inserted in a system for a harmful purpose.
- A virus is a piece of software that can "infect" other programs by modifying them; the modification includes a copy of the virus program, which can then go on to infect other programs.
- A worm is a program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. In addition to propagation, the worm usually performs some unwanted function.
- A denial of service (DoS) attack is an attempt to prevent legitimate users of a service from using that service.
- A distributed denial of service attack is launched from multiple coordinated sources.

This chapter examines malicious software (malware), especially viruses and worms.

[Page 599 (continued)]

19.1. Viruses and Related Threats

Perhaps the most sophisticated types of threats to computer systems are presented by programs that exploit vulnerabilities in computing systems. In this context, we are concerned with application programs as well as utility programs, such as editors and compilers.

We begin this section with an overview of the spectrum of such software threats. The remainder of the section is devoted to viruses and worms.

Malicious Programs

The terminology in this area presents problems because of a lack of universal agreement on all of the terms and because some of the categories overlap. Table 19.1, based principally on [SZOR05], is a useful guide.

Table 19.1. Terminology of Malicious Programs

(This item is displayed on page 600 in the print version)

Name	Description
Virus	Attaches itself to a program and propagates copies of itself to other programs
Worm	Program that propagates copies of itself to other computers
Logic bomb	Triggers action when condition occurs
Trojan horse	Program that contains unexpected additional functionality
Backdoor (trapdoor)	Program modification that allows unauthorized access to functionality
Exploits	Code specific to a single vulnerability or set of vulnerabilities
Downloaders	Program that installs other items on a machine that is under attack. Usually, a downloader is sent in an e-mail.
Auto-rooter	Malicious hacker tools used to break into new machines remotely
Kit (virus generator)	Set of tools for generating new viruses automatically
Spammer programs	Used to send large volumes of unwanted e-mail
Flooders	Used to attack networked computer systems with a large volume of traffic to carry out a denial of service (DoS) attack
Keyloggers	Captures keystrokes on a compromised system
Rootkit	Set of hacker tools used after attacker has broken into a computer system and gained root-level access

Zombie	Program activated on an infected machine that is activated to launch attacks on other machines

Malicious software can be divided into two categories: those that need a host program, and those that are independent. The former are essentially fragments of programs that cannot exist independently of some actual application program, utility, or system program. Viruses, logic bombs, and backdoors are examples. The latter are self-contained programs that can be scheduled and run by the operating system. Worms and zombie programs are examples.

We can also differentiate between those software threats that do not replicate and those that do. The former are programs or fragments of programs that are activated by a trigger. Examples are logic bombs, backdoors, and zombie programs. The latter consist of either a program fragment or an independent program that, when executed, may produce one or more copies of itself to be activated later on the same system or some other system. Viruses and worms are examples.

In the remainder of this subsection, we briefly survey some of the key categories of malicious software, with the exception of viruses and worms, which are covered in more detail later in this section.

Backdoor

A backdoor, also known as a trapdoor, is a secret entry point into a program that allows someone that is aware of the backdoor to gain access without going through the usual security access procedures. Programmers have used backdoors legitimately for many years to debug and test programs. This usually is done when the programmer is developing an application that has an authentication procedure, or a long setup, requiring the user to enter many different values to run the application. To debug the program, the developer may wish to gain special privileges or to avoid all the necessary setup and authentication. The programmer may also want to ensure that there is a method of activating the program should something be wrong with the authentication procedure that is being built into the application. 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.

Backdoors become threats when unscrupulous programmers use them to gain unauthorized access. The backdoor was the basic idea for the vulnerability portrayed in the movie *War Games*. Another example is that during the development of Multics, penetration tests were conducted by an Air Force "tiger team" (simulating adversaries). One tactic employed was to send a bogus operating system update to a site running Multics. The update contained a Trojan horse (described later) that could be activated by a backdoor and that allowed the tiger team to gain access. The threat was so well implemented that the Multics developers could not find it, even after they were informed of its presence [ENGE80].

It is difficult to implement operating system controls for backdoors. Security measures must focus on the program development and software update activities.

Logic Bomb

One of the oldest types of program threat, predating viruses and worms, is the logic bomb. The logic bomb is code embedded in some legitimate program 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, a particular day of the week or date, 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. A striking example of how logic bombs can be employed was the case of Tim Lloyd, who was convicted of setting a logic bomb that cost his employer, Omega Engineering, more than $10 million, derailed its corporate growth strategy, and eventually led to the layoff of 80 workers [GAUD00]. Ultimately, Lloyd was sentenced to 41 months in prison and ordered to pay $2 million in restitution.

Trojan Horses

A Trojan horse is a useful, or apparently useful, program or command procedure containing hidden code that, when invoked, performs some unwanted or harmful function.

Trojan horse programs can be used to accomplish functions indirectly that an unauthorized user could not accomplish directly. For example, to gain access to the files of another user on a shared system, a user could create a Trojan horse program that, when executed, changed the invoking user's file permissions so that the files are readable by any user. The author could then induce users to run the program by placing it in a common directory and naming it such that it appears to be a useful utility. An example is a program that ostensibly produces a listing of the user's files in a desirable format. After another user has run the program, the author can then access the information in the user's files. An example of a Trojan horse program that would be difficult to detect is a compiler that has been modified to insert additional code into certain programs as they are compiled, such as a system login program [THOM84]. The code creates a backdoor in the login program that permits the author to log on to the system using a special password. This Trojan horse can never be discovered by reading the source code of the login program.

Another common motivation for the Trojan horse is data destruction. The program appears to be performing a useful function (e.g., a calculator program), but it may also be quietly deleting the user's files. For example, a CBS executive was victimized by a Trojan horse that destroyed all information contained in his computer's memory [TIME90]. The Trojan horse was implanted in a graphics routine offered on an electronic bulletin board system.

Zombie

A zombie is a program that secretly takes over another Internet-attached computer and then uses that computer to launch attacks that are difficult to trace to the zombie's creator. Zombies are used in denial-of-service attacks, typically against targeted Web sites. The zombie is planted on hundreds of computers belonging to unsuspecting third parties, and then used to overwhelm the target Web site by launching an overwhelming onslaught of Internet traffic. Section 19.3 discusses zombies in the context of denial of service attacks.

The Nature of Viruses

A virus is a piece of software that can "infect" other programs by modifying them; the modification includes a copy of the virus program, which can then go on to infect other programs.

Biological viruses are tiny scraps of genetic codeDNA or RNAthat can take over the machinery of a living cell and trick it into making thousands of flawless replicas of the original virus. Like its biological counterpart, a computer virus carries in its instructional code the recipe for making perfect copies of itself. The typical virus becomes embedded in a program on a computer. Then, whenever the infected computer comes into contact with an uninfected piece of software, a fresh copy of the virus passes into

the new program. Thus, the infection can be spread from computer to computer by unsuspecting users who either swap disks or send programs to one another over a network. In a network environment, the ability to access applications and system services on other computers provides a perfect culture for the spread of a virus.

A virus can do anything that other programs do. The only difference is that it attaches itself to another program and executes secretly when the host program is run. Once a virus is executing, it can perform any function, such as erasing files and programs.

During its lifetime, a typical virus goes through the following four phases:

- **Dormant phase:** The virus is idle. The virus will eventually be activated by some event, such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.
- **Propagation phase:** The virus places an identical copy of itself into other programs or into certain system areas on the disk. Each infected program will now contain a clone of the virus, which will itself enter a propagation phase.
- **Triggering phase:** The virus is activated to perform the function for which it was intended. As with the dormant phase, the triggering phase can be caused by a variety of system events, including a count of the number of times that this copy of the virus has made copies of itself.
- **Execution phase:** The function is performed. The function may be harmless, such as a message on the screen, or damaging, such as the destruction of programs and data files.

Most viruses carry out their work in a manner that is specific to a particular operating system and, in some cases, specific to a particular hardware platform. Thus, they are designed to take advantage of the details and weaknesses of particular systems.

Virus Structure

A virus can be prepended or postpended to an executable program, or it can be embedded in some other fashion. The key to its operation is that the infected program, when invoked, will first execute the virus code and then execute the original code of the program.

A very general depiction of virus structure is shown in Figure 19.1 (based on [COHE94]. In this case, the virus code, V, is prepended to infected programs, and it is assumed that the entry point to the program, when invoked, is the first line of the program.

Figure 19.1. A Simple Virus

```
        program V :=

{goto main;
        1234567;

        subroutine infect-executable :=
                {loop:
                file := get-random-executable-file;
                if (first-line-of-file = 1234567)
                   then goto loop
                   else prepend V to file; }

        subroutine do-damage :=
                {whatever damage is to be done}

        subroutine trigger-pulled :=
                {return true if some condition holds}

main:   main-program :=
                {infect-executable;
                if trigger-pulled then do-damage;
                goto next;}
next:

}
```

An infected program begins with the virus code and works as follows. The first line of code is a jump to the main virus program. The second line is a special marker that is used by the virus to determine whether or not a potential victim program has already been infected with this virus. When the program is invoked, control is immediately transferred to the main virus program. The virus program first seeks out uninfected executable files and infects them. Next, the virus may perform some action, usually detrimental to the system. This action could be performed every time the program is invoked, or it could be a logic bomb that triggers only under certain conditions. Finally, the virus transfers control to the original program. If the infection phase of the program is reasonably rapid, a user is unlikely to notice any difference between the execution of an infected and uninfected program.

A virus such as the one just described is easily detected because an infected version of a program is longer than the corresponding uninfected one. A way to thwart such a simple means of detecting a virus is to compress the executable file so that both the infected and uninfected versions are of identical

length. Figure 19.2 [COHE94] shows in general terms the logic required. The key lines in this virus are numbered, and Figure 19.3 [COHE94] illustrates the operation. We assume that program P_1 is infected with the virus CV. When this program is invoked, control passes to its virus, which performs the following steps:

---

1. For each uninfected file P_2 that is found, the virus first compresses that file to produce P'_2, which is shorter than the original program by the size of the virus.
2. A copy of the virus is prepended to the compressed program.
3. The compressed version of the original infected program, P'_1, is uncompressed.
4. The uncompressed original program is executed.

Figure 19.2. Logic for a Compression Virus

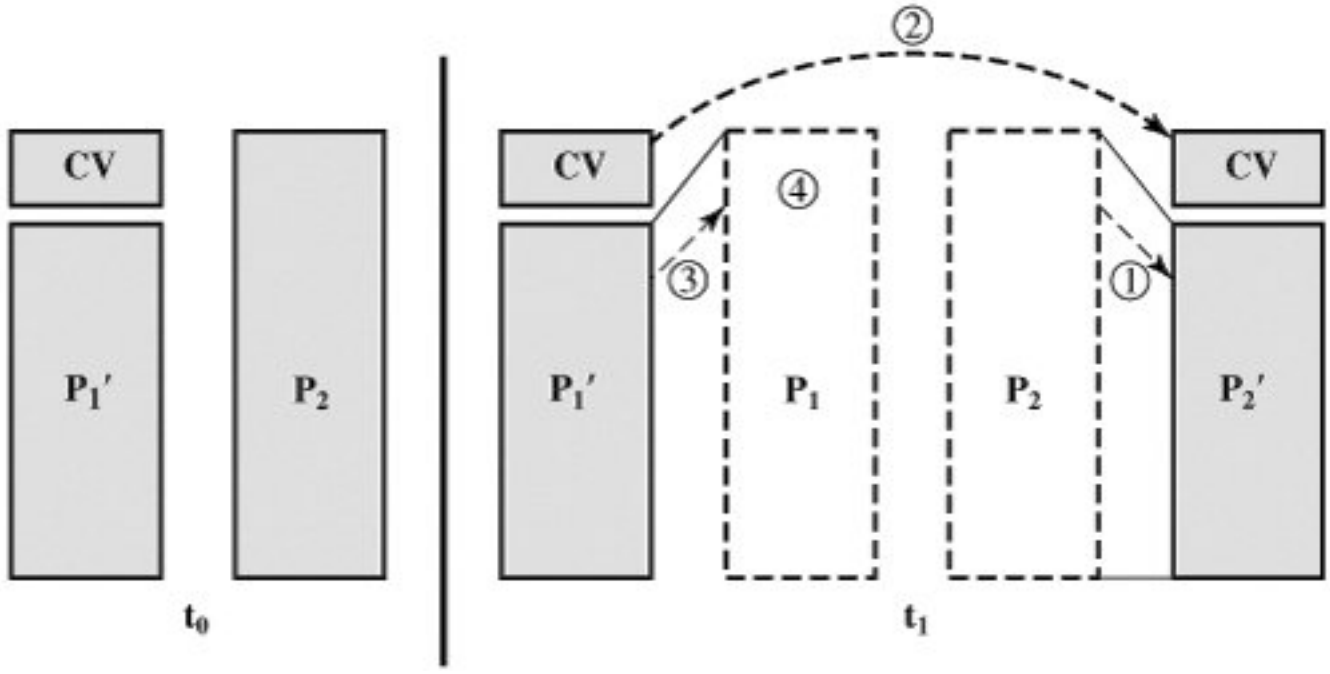
```
     program CV :=

{goto main;
     01234567;

     subroutine infect-executable :=
               {loop:
                    file := get-random-executable-file;
               if (first-line-of-file = 01234567) then goto loop;
          (1)  compress file;
          (2)  prepend CV to file;
          }

main:  main-program :=
               {if ask-permission then infect-executable;
          (3)  uncompress rest-of-file;
          (4)  run uncompressed file;}
          }
```

Figure 19.3. A Compression Virus



In this example, the virus does nothing other than propagate. As in the previous example, the virus may include a logic bomb.

Initial Infection

Once a virus has gained entry to a system by infecting a single program, it is in a position to infect some or all other executable files on that system when the infected program executes. Thus, viral infection can be completely prevented by preventing the virus from gaining entry in the first place. Unfortunately, prevention is extraordinarily difficult because a virus can be part of any program outside a system. Thus, unless one is content to take an absolutely bare piece of iron and write all one's own system and application programs, one is vulnerable.

Types of Viruses

There has been a continuous arms race between virus writers and writers of antivirus software since viruses first appeared. As effective countermeasures have been developed for existing types of viruses, new types have been developed. [STEP93] suggests the following categories as being among the most significant types of viruses:

- **Parasitic virus:** The traditional and still most common form of virus. A parasitic virus attaches itself to executable files and replicates, when the infected program is executed, by finding other executable files to infect.
- **Memory-resident virus:** Lodges in main memory as part of a resident system program. From that point on, the virus infects every program that executes.
- **Boot sector virus:** Infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus.
- **Stealth virus:** A form of virus explicitly designed to hide itself from detection by antivirus software.
- **Polymorphic virus:** A virus that mutates with every infection, making detection by the "signature" of the virus impossible.
- **Metamorphic virus:** As with a polymorphic virus, a metamorphic virus mutates with every

infection. The difference is that a metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses my change their behavior as well as their appearance.

One example of a **stealth virus** was discussed earlier: a virus that uses compression so that the infected program is exactly the same length as an uninfected version. Far more sophisticated techniques are possible. For example, a virus can place intercept logic in disk I/O routines, so that when there is an attempt to read suspected portions of the disk using these routines, the virus will present back the original, uninfected program. Thus, *stealth* is not a term that applies to a virus as such but, rather, is a technique used by a virus to evade detection.

A **polymorphic virus** creates copies during replication that are functionally equivalent but have distinctly different bit patterns. As with a stealth virus, the purpose is to defeat programs that scan for viruses. In this case, the "signature" of the virus will vary with each copy. To achieve this variation, the virus may randomly insert superfluous instructions or interchange the order of independent instructions. A more effective approach is to use encryption. A portion of the virus, generally called a *mutation engine*, creates a random encryption key to encrypt the remainder of the virus. The key is stored with the virus, and the mutation engine itself is altered. When an infected program is invoked, the virus uses the stored random key to decrypt the virus. When the virus replicates, a different random key is selected.

Another weapon in the virus writers' armory is the virus-creation toolkit. Such a toolkit enables a relative novice to create quickly a number of different viruses. Although viruses created with toolkits tend to be less sophisticated than viruses designed from scratch, the sheer number of new viruses that can be generated creates a problem for antivirus schemes.

Macro Viruses

In the mid-1990s, macro viruses became by far the most prevalent type of virus. Macro viruses are particularly threatening for a number of reasons:

1. A macro virus is platform independent. Virtually all of the macro viruses infect Microsoft Word documents. Any hardware platform and operating system that supports Word can be infected.

2. Macro viruses infect documents, not executable portions of code. Most of the information introduced onto a computer system is in the form of a document rather than a program.

3. Macro viruses are easily spread. A very common method is by electronic mail.

Macro viruses take advantage of a feature found in Word and other office applications such as Microsoft Excel, namely the macro. In essence, a macro is an executable program embedded in a word processing document or other type of file. Typically, users employ macros to automate repetitive tasks and thereby save keystrokes. The macro language is usually some form of the Basic programming language. A user might define a sequence of keystrokes in a macro and set it up so that the macro is invoked when a

function key or special short combination of keys is input.

Successive releases of Word provide increased protection against macro viruses. For example, Microsoft offers an optional Macro Virus Protection tool that detects suspicious Word files and alerts the customer to the potential risk of opening a file with macros. Various antivirus product vendors have also developed tools to detect and correct macro viruses. As in other types of viruses, the arms race continues in the field of macro viruses, but they no longer are the predominant virus threat.

E-mail Viruses

A more recent development in malicious software is the e-mail virus. The first rapidly spreading e-mail viruses, such as Melissa, made use of a Microsoft Word macro embedded in an attachment. If the recipient opens the e-mail attachment, the Word macro is activated. Then

1. The e-mail virus sends itself to everyone on the mailing list in the user's e-mail package.
2. The virus does local damage.

At the end of 1999, a more powerful version of the e-mail virus appeared. This newer version can be activated merely by opening an e-mail that contains the virus rather than opening an attachment. The virus uses the Visual Basic scripting language supported by the e-mail package.

Thus we see a new generation of malware that arrives via e-mail and uses e-mail software features to replicate itself across the Internet. The virus propagates itself as soon as activated (either by opening an e-mail attachment of by opening the e-mail) to all of the e-mail addresses known to the infected host. As a result, whereas viruses used to take months or years to propagate, they now do so in hours. This makes it very difficult for antivirus software to respond before much damage is done. Ultimately, a greater degree of security must be built into Internet utility and application software on PCs to counter the growing threat.

Worms

A worm is a program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. In addition to propagation, the worm usually performs some unwanted function. An e-mail virus has some of the characteristics of a worm, because it propagates itself from system to system. However, we can still classify it as a virus because it requires a human to move it forward. A worm actively seeks out more machines to infect and each machine that is infected serves as an automated launching pad for attacks on other machines.

Network worm programs use network connections to spread from system to system. Once active within a system, a network worm can behave as a computer virus or bacteria, or it could implant Trojan horse programs or perform any number of disruptive or destructive actions.

To replicate itself, a network worm uses some sort of network vehicle. Examples include the following:

- **Electronic mail facility:** A worm mails a copy of itself to other systems.
- **Remote execution capability:** A worm executes a copy of itself on another system.
- **Remote login capability:** A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other.

The new copy of the worm program is then run on the remote system where, in addition to any functions that it performs at that system, it continues to spread in the same fashion.

A network worm exhibits the same characteristics as a computer virus: a dormant phase, a propagation phase, a triggering phase, and an execution phase. The propagation phase generally performs the following functions:

1. Search for other systems to infect by examining host tables or similar repositories of remote system addresses.

2. Establish a connection with a remote system.

3. Copy itself to the remote system and cause the copy to be run.

The network worm may also attempt to determine whether a system has previously been infected before copying itself to the system. In a multiprogramming system, it may also disguise its presence by naming itself as a system process or using some other name that may not be noticed by a system operator.

As with viruses, network worms are difficult to counter.

The Morris Worm

Until the current generation of worms, the best known was the worm released onto the Internet by Robert Morris in 1998. The Morris worm was designed to spread on UNIX systems and used a number of different techniques for propagation. When a copy began execution, its first task was to discover other hosts known to this host that would allow entry from this host. The worm performed this task by examining a variety of lists and tables, including system tables that declared which other machines were trusted by this host, users' mail forwarding files, tables by which users gave themselves permission for access to remote accounts, and from a program that reported the status of network connections. For each discovered host, the worm tried a number of methods for gaining access:

1.

   It attempted to log on to a remote host as a legitimate user. In this method, the worm first attempted to crack the local password file, and then used the discovered passwords and corresponding user IDs. The assumption was that many users would use the same password on different systems. To obtain the passwords, the worm ran a password-cracking program that tried

   a.

      Each user's account name and simple permutations of it

A list of 432 built-in passwords that Morris thought to be likely candidates

c.

All the words in the local system directory

2.

It exploited a bug in the finger protocol, which reports the whereabouts of a remote user.

3.

It exploited a trapdoor in the debug option of the remote process that receives and sends mail.

If any of these attacks succeeded, the worm achieved communication with the operating system command interpreter. It then sent this interpreter a short bootstrap program, issued a command to execute that program, and then logged off. The bootstrap program then called back the parent program and downloaded the remainder of the worm. The new worm was then executed.

Recent Worm Attacks

The contemporary era of worm threats began with the release of the Code Red worm in July of 2001. Code Red exploits a security hole in the Microsoft Internet Information Server (IIS) to penetrate and spread. It also disables the system file checker in Windows. The worm probes random IP addresses to spread to other hosts. During a certain period of time, it only spreads. It then initiates a denial-of-service attack against a government Web site by flooding the site with packets from numerous hosts. The worm then suspends activities and reactivates periodically. In the second wave of attack, Code Red infected nearly 360,000 servers in 14 hours. In addition to the havoc it causes at the targeted server, Code Red can consume enormous amounts of Internet capacity, disrupting service.

Code Red II is a variant that targets Microsoft IISs. In addition, this newer worm installs a backdoor allowing a hacker to direct activities of victim computers.

In late 2001, a more versatile worm appeared, known as Nimda. Nimda spreads by multiple mechanisms:

- from client to client via e-mail
- from client to client via open network shares

- from Web server to client via browsing of compromised Web sites
- from client to Web server via active scanning for and exploitation of various Microsoft IIS 4.0 / 5.0 directory traversal vulnerabilities
- from client to Web server via scanning for the back doors left behind by the "Code Red II" worms

The worm modifies Web documents (e.g., .htm, .html, and .asp files) and certain executable files found on the systems it infects and creates numerous copies of itself under various filenames.

In early 2003, the SQL Slammer worm appeared. This worm exploited a buffer overflow vulnerability in Microsoft SQL server. The Slammer was extremely compact and spread rapidly, infecting 90% of vulnerable hosts within 10 minutes. Late 2003 saw the arrival of the Sobig.f worm, which exploited open proxy servers to turn infected machines into spam engines. At its peak, Sobig.f reportedly accounted for one in every 17 messages and produced more than one million copies of itself within the first 24 hours.

Mydoom is a mass-mailing e-mail worm that appeared in 2004. It followed a growing trend of installing a backdoor in infected computers, thereby enabling hackers to gain remote access to data such as passwords and credit card numbers. Mydoom replicated up to 1000 times per minute and reportedly flooded the Internet with 100 million infected messages in 36 hours.

State of Worm Technology

The state of the art in worm technology includes the following:

- **Multiplatform:** Newer worms are not limited to Windows machines but can attack a variety of platforms, especially the popular varieties of UNIX.
- **Multiexploit:** New worms penetrate systems in a variety of ways, using exploits against Web servers, browsers, e-mail, file sharing, and other network-based applications.
- **Ultrafast spreading:** One technique to accelerate the spread of a worm is to conduct a prior Internet scan to accumulate Internet addresses of vulnerable machines.
- **Polymorphic:** To evade detection, skip past filters, and foil real-time analysis, worms adopt the virus polymorphic technique. Each copy of the worm has new code generated on the fly using functionally equivalent instructions and encryption techniques.
- **Metamorphic:** In addition to changing their appearance, metamorphic worms have a repertoire of behavior patterns that are unleashed at different stages of propagation.
- **Transport vehicles:** Because worms can rapidly compromise a large number of systems, they are ideal for spreading other distributed attack tools, such as distributed denial of service zombies.
- **Zero-day exploit:** To achieve maximum surprise and distribution, a worm should exploit an unknown vulnerability that is only discovered by the general network community when the worm is launched.

PREV NEXT

19.2. Virus Countermeasures

Antivirus Approaches

The ideal solution to the threat of viruses is prevention: Do not allow a virus to get into the system in the first place. This goal is, in general, impossible to achieve, although prevention can reduce the number of successful viral attacks. The next best approach is to be able to do the following:

- **Detection:** Once the infection has occurred, determine that it has occurred and locate the virus.
- **Identification:** Once detection has been achieved, identify the specific virus that has infected a program.
- **Removal:** Once the specific virus has been identified, remove all traces of the virus from the infected program and restore it to its original state. Remove the virus from all infected systems so that the disease cannot spread further.

If detection succeeds but either identification or removal is not possible, then the alternative is to discard the infected program and reload a clean backup version.

Advances in virus and antivirus technology go hand in hand. Early viruses were relatively simple code fragments and could be identified and purged with relatively simple antivirus software packages. As the virus arms race has evolved, both viruses 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 virus signature to identify a virus. The virus may contain "wildcards" but has essentially the same structure and bit pattern in all copies. Such signature-specific scanners are limited to the detection of known viruses. Another type of first-generation scanner maintains a record of the length of programs and looks for changes in length.

A **second-generation** scanner does not rely on a specific signature. Rather, the scanner uses heuristic rules to search for probable virus infection. One class of such scanners looks for fragments of code that are often associated with viruses. 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 virus to identify it, 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 a virus infects the program without changing the checksum, then an integrity check will catch the change. To counter a virus that is sophisticated enough to change the checksum when it infects a program, an encrypted hash function can be used. The encryption key is stored separately from the program so that the virus cannot generate a new hash code and encrypt that. By using a hash function rather than a simpler checksum, the virus is prevented from adjusting the program to produce the same hash code as before.

Third-generation programs are memory-resident programs that identify a virus by its actions rather than its structure in an infected program. Such programs have the advantage that it is not necessary to develop signatures and heuristics for a wide array of viruses. Rather, it is necessary only to identify the small set of actions that indicate an infection 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 viruses to penetrate a system and then limits the ability of a virus to update files in order to pass on the infection.

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.

Advanced Antivirus Techniques

More sophisticated antivirus approaches and products continue to appear. In this subsection, we highlight two of the most important.

Generic Decryption

Generic decryption (GD) technology enables the antivirus program to easily detect even the most complex polymorphic viruses, while maintaining fast scanning speeds [NACH97]. Recall that when a file containing a polymorphic virus is executed, the virus must decrypt itself to activate. In order to detect such a structure, executable files are run through a GD scanner, which contains the following elements:

- **CPU emulator:** A software-based virtual computer. Instructions in an executable file are interpreted by the emulator rather than executed on the underlying processor. The emulator includes software versions of all registers and other processor hardware, so that the underlying processor is unaffected by programs interpreted on the emulator.
- **Virus signature scanner:** A module that scans the target code looking for known virus signatures.
- **Emulation control module:** Controls the execution of the target code.

At the start of each simulation, the emulator begins interpreting instructions in the target code, one at a time. Thus, if the code includes a decryption routine that decrypts and hence exposes the virus, that code is interpreted. In effect, the virus does the work for the antivirus program by exposing the virus. Periodically, the control module interrupts interpretation to scan the target code for virus signatures.

During interpretation, the target code can cause no damage to the actual personal computer environment, because it is being interpreted in a completely controlled environment.

The most difficult design issue with a GD scanner is to determine how long to run each interpretation. Typically, virus elements are activated soon after a program begins executing, but this need not be the case. The longer the scanner emulates a particular program, the more likely it is to catch any hidden viruses. However, the antivirus program can take up only a limited amount of time and resources before users complain.

Digital Immune System

The digital immune system is a comprehensive approach to virus protection developed by IBM [KEPH97a, KEPH97b]. The motivation for this development has been the rising threat of Internet-based virus propagation. We first say a few words about this threat and then summarize IBM's approach.

Traditionally, the virus threat was characterized by the relatively slow spread of new viruses and new mutations. Antivirus software was typically updated on a monthly basis, and this has been sufficient to control the problem. Also traditionally, the Internet played a comparatively small role in the spread of viruses. But as [CHES97] points out, two major trends in Internet technology have had an increasing impact on the rate of virus propagation in recent years:

- **Integrated mail systems:** Systems such as Lotus Notes and Microsoft Outlook make it very simple to send anything to anyone and to work with objects that are received.
- **Mobile-program systems:** Capabilities such as Java and ActiveX allow programs to move on their own from one system to another.

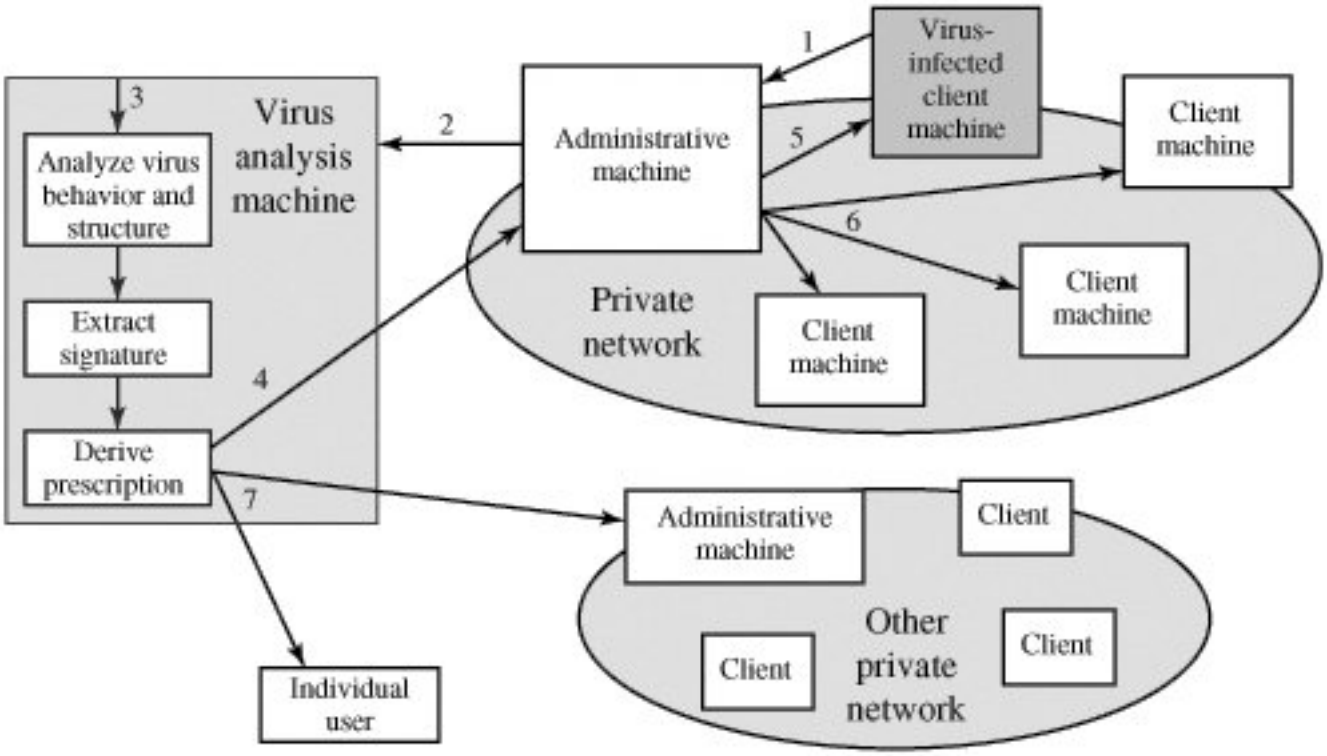
In response to the threat posed by these Internet-based capabilities, IBM has developed a prototype digital immune system. This system expands on the use of program emulation discussed in the preceding subsection and provides a general-purpose emulation and virus-detection system. The objective of this system is to provide rapid response time so that viruses can be stamped out almost as soon as they are introduced. When a new virus enters an organization, the immune system automatically captures it, analyzes it, adds detection and shielding for it, removes it, and passes information about that virus to systems running IBM AntiVirus so that it can be detected before it is allowed to run elsewhere.

Figure 19.4 illustrates the typical steps in digital immune system operation:

1. A monitoring program on each PC uses a variety of heuristics based on system behavior, suspicious changes to programs, or family signature to infer that a virus may be present. The monitoring program forwards a copy of any program thought to be infected to an administrative machine within the organization.
2. The administrative machine encrypts the sample and sends it to a central virus analysis machine.
3. This machine creates an environment in which the infected program can be safely run for analysis. Techniques used for this purpose include emulation, or the creation of a protected environment within which the suspect program can be executed and monitored. The virus analysis machine then produces a prescription for identifying and removing the virus.
4. The resulting prescription is sent back to the administrative machine.
5. The administrative machine forwards the prescription to the infected client.
6. The prescription is also forwarded to other clients in the organization.
7. Subscribers around the world receive regular antivirus updates that protect them from the new virus.

Figure 19.4. Digital Immune System

[View full size image]



The success of the digital immune system depends on the ability of the virus analysis machine to detect new and innovative virus strains. By constantly analyzing and monitoring the viruses found in the wild, it should be possible to continually update the digital immune software to keep up with the threat.

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. 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; and
- Initiation of network communications.

If the behavior blocker detects that a program is initiating would-be malicious behaviors as it runs, it can block these behaviors in real-time and/or terminate the offending software. This gives it a fundamental advantage over such established antivirus detection techniques as fingerprinting or heuristics. While 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, 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.

The ability to watch software as it runs in real time clearly confers a huge benefit to the behavior

blocker; however, it also has drawbacks. Since the malicious code must actually run on the target machine before all its behaviors can be identified, it can cause a great deal of harm to the system before it has been detected and blocked by the behavior blocking system. For instance, a new virus might shuffle a number of seemingly unimportant files around the hard drive before infecting a single file and being blocked. Even though the actual infection was blocked, the user may be unable to locate their files, causing a loss to productivity or possibly worse.

PREV NEXT

19.3. Distributed Denial of Service Attacks

Distributed denial of service (DDoS) attacks present a significant security threat to corporations, and the threat appears to be growing [VIJA02]. In one study, covering a three-week period in 2001, investigators observed more than 12,000 attacks against more than 5000 distinct targets, ranging from well-known ecommerce companies such as Amazon and Hotmail to small foreign ISPs and dial-up connections [MOOR01]. DDoS attacks make computer systems inaccessible by flooding servers, networks, or even end user systems with useless traffic so that legitimate users can no longer gain access to those resources. In a typical DDoS attack, a large number of compromised hosts are amassed to send useless packets. In recent years, the attack methods and tools have become more sophisticated, effective, and more difficult to trace to the real attackers, while defense technologies have been unable to withstand large-scale attacks [CHAN02].

A denial of service (DoS) attack is an attempt to prevent legitimate users of a service from using that service. When this attack comes from a single host or network node, then it is simply referred to as a DoS attack. A more serious threat is posed by a DDoS attack. In a DDoS attack, an attacker is able to recruit a number of hosts throughout the Internet to simultaneously or in a coordinated fashion launch an attack upon the target. This section is concerned with DDoS attacks. First, we look at the nature and types of attacks. Next, we examine means by which an attacker is able to recruit a network of hosts for attack launch. Finally, this section looks at countermeasures.

DDoS Attack Description

A DDoS attack attempts to consume the target's resources so that it cannot provide service. One way to classify DDoS attacks is in terms of the type of resource that is consumed. Broadly speaking, the resource consumed is either an internal host resource on the target system or data transmission capacity in the local network to which the target is attacked.

A simple example of an **internal resource attack** is the SYN flood attack. Figure 19.5a shows the steps involved:

1. The attacker takes control of multiple hosts over the Internet, instructing them to contact the target Web server.

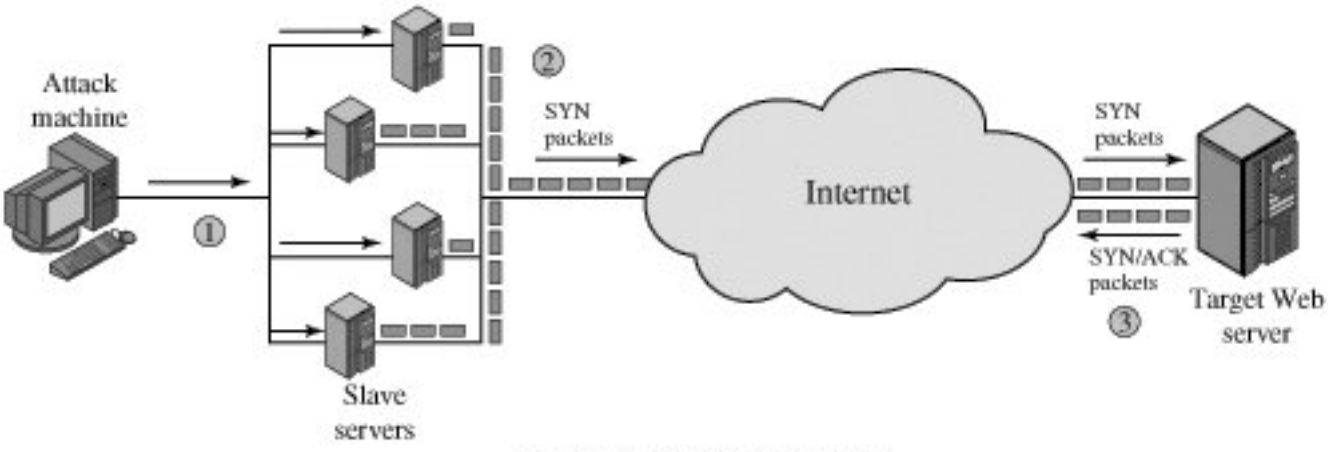
2. The slave hosts begin sending TCP/IP SYN (synchronize/initialization) packets, with erroneous return IP address information, to the target.

---

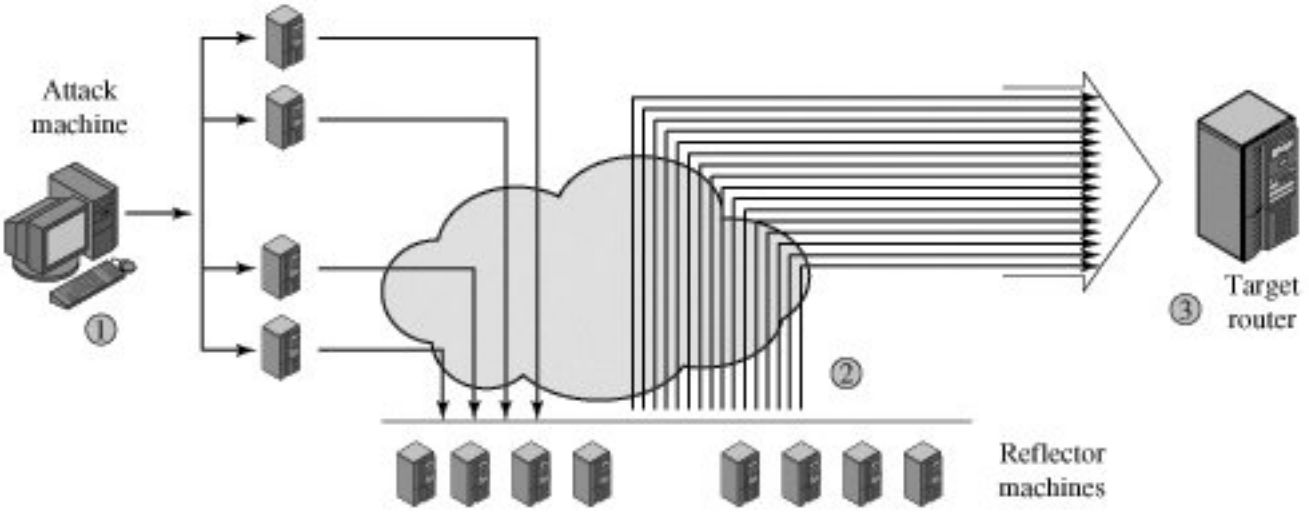
3. Each SYN packet is a request to open a TCP connection. For each such packet, the Web server responds with a SYN/ACK (synchronize/acknowledge) packet, trying to establish a TCP connection with a TCP entity at a spurious IP address. The Web server maintains a data structure for each SYN request waiting for a response back and becomes bogged down as more traffic floods in. The result is that legitimate connections are denied while the victim machine is waiting to complete bogus "half-open" connections.

Figure 19.5. Examples of Simple DDoS Attacks

[View full size image]



(a) Distributed SYN flood attack



(a) Distributed ICMP attack

The TCP state data structure is a popular internal resource target but by no means the only one. [CERT01] gives the following examples:

1.

In many systems, a limited number of data structures are available to hold process information (process identifiers, process table entries, process slots, etc.). An intruder may be able to consume these data structures by writing a simple program or script that does nothing but repeatedly create copies of itself.

2.

An intruder may also attempt to consume disk space in other ways, including

- generating excessive numbers of mail messages
- intentionally generating errors that must be logged
- placing files in anonymous ftp areas or network-shared areas

Figure 19.5b illustrates an example of an **attack that consumes data transmission resources**. The following steps are involved:

1. The attacker takes control of multiple hosts over the Internet, instructing them to send ICMP ECHO packets[1] with the target's spoofed IP address to a group of hosts that act as reflectors, as described subsequently.

   [1] The Internet Control Message Protocol (ICMP) is an IP-level protocol for the exchange of control packets between a router and a host or between hosts. The ECHO packet requires the recipient to respond with an echo reply to check that communication is possible between entities.

2. Nodes at the bounce site receive multiple spoofed requests and respond by sending echo reply packets to the target site.

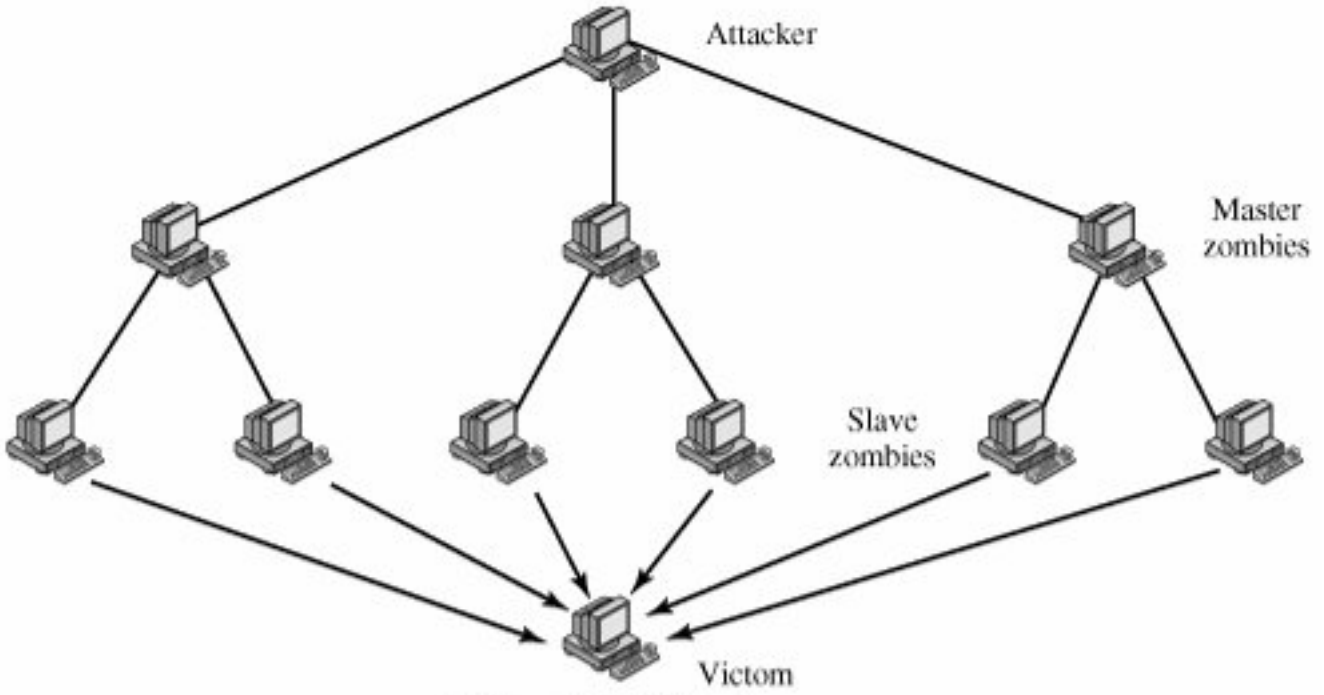
3. The target's router is flooded with packets from the bounce site, leaving no data transmission capacity for legitimate traffic.

Another way to classify DDoS attacks is as either direct or reflector DDoS attacks. In a **direct DDoS** attack (Figure 19.6a), the attacker is able to implant zombie software on a number of sites distributed throughout the Internet. Often, the DDoS attack involves two levels of zombie machines: master zombies and slave zombies. The hosts of both machines have been infected with malicious code. The attacker coordinates and triggers the master zombies, which in turn coordinate and trigger the slave zombies. The use of two levels of zombies makes it more difficult to trace the attack back to its source and provides for a more resilient network of attackers.

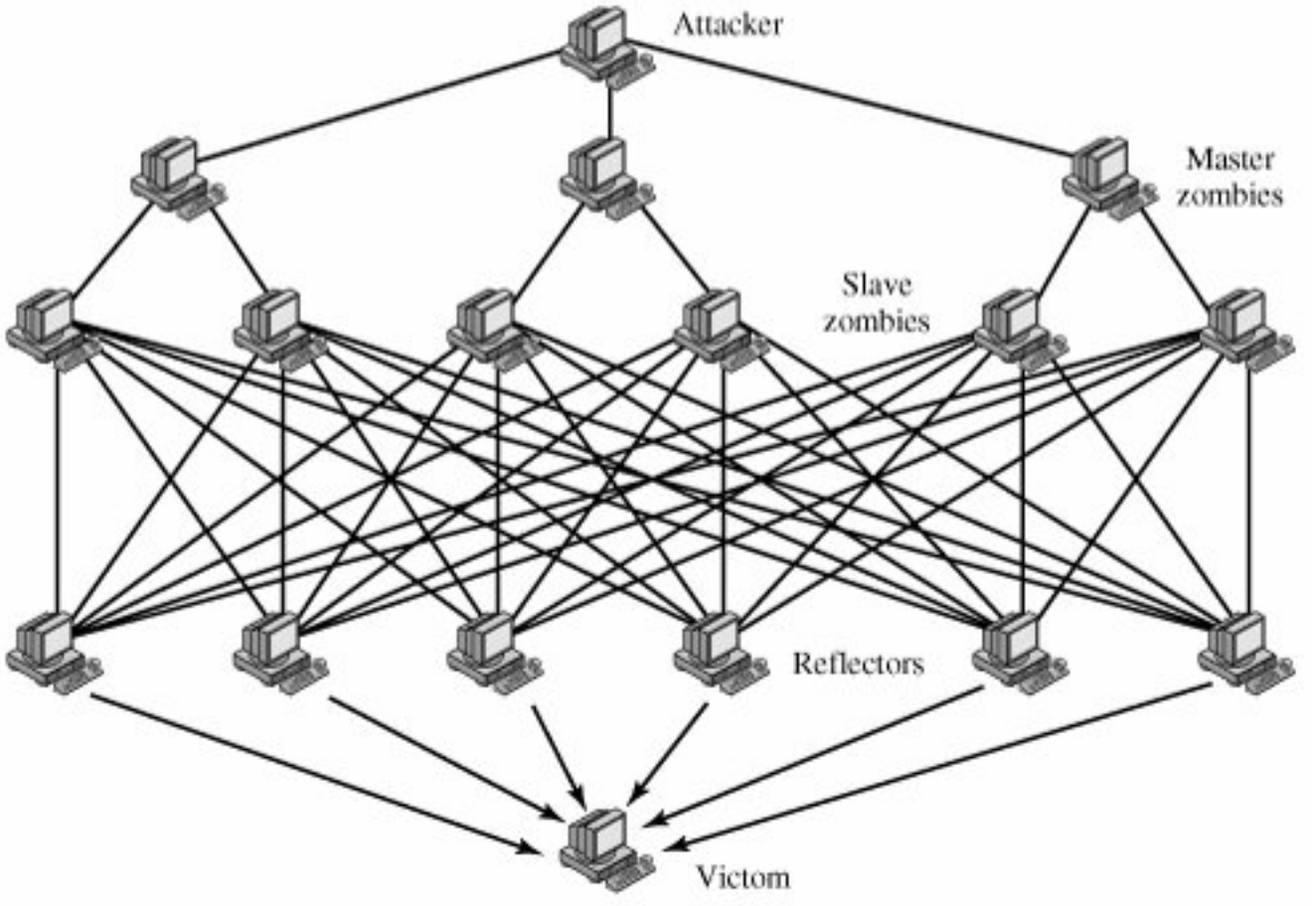
Figure 19.6. Types of Flooding-Based DDoS Attacks

(This item is displayed on page 617 in the print version)

[View full size image]



(a) Direct DDoS Attack



(b) Reflector DDoS Attack

A **reflector DDoS** attack adds another layer of machines (Figure 19.6b). In this type of attack, the slave zombies construct packets requiring a response that contain the target's IP address as the source IP address in the packet's IP header. These packets are sent to uninfected machines known as reflectors. The uninfected machines respond with packets directed at the target machine. A reflector DDoS attack can easily involve more machines and more traffic than a direct DDoS attack and hence be more damaging. Further, tracing back the attack or filtering out the attack packets is more difficult because the attack comes from widely dispersed uninfected machines.

Constructing the Attack Network

The first step in a DDoS attack is for the attacker to infect a number of machines with zombie software that will ultimately be used to carry out the attack. The essential ingredients in this phase of the attack are the following:

1. Software that can carry out the DDoS attack. The software must be able to run on a large number of machines, must be able to conceal its existence, must be able to communicate with the attacker or have some sort of time-triggered mechanism, and must be able to launch the intended attack toward the target.

2. A vulnerability in a large number of systems. The attacker must become aware of a vulnerability that many system administrators and individual users have failed to patch and that enables the attacker to install the zombie software.

3. A strategy for locating vulnerable machines, a process known as scanning.

In the scanning process, the attacker first seeks out a number of vulnerable machines and infects them. Then, typically, the zombie software that is installed in the infected machines repeats the same scanning process, until a large distributed network of infected machines is created. [MIRK04] lists the following types of scanning strategies:

- **Random:** Each compromised host probes random addresses in the IP address space, using a different seed. This technique produces a high volume of Internet traffic, which may cause generalized disruption even before the actual attack is launched.
- **Hit-list:** The attacker first compiles a long list of potential vulnerable machines. This can be a slow process done over a long period to avoid detection that an attack is underway. Once the list is compiled, the attacker begins infecting machines on the list. Each infected machine is provided with a portion of list to scan. This strategy results in a very short scanning period, which may make it difficult to detect that infection is taking place.
- **Topological:** This method uses information contained on an infected victim machine to find more hosts to scan.
- **Local subnet:** If a host can be infected behind a firewall, that host then looks for targets in its own local network. The host uses the subnet address structure to find other hosts that would otherwise be protected by the firewall.

DDoS Countermeasures

In general, there are three lines of defense against DDoS attacks [CHAN02]:

- **Attack prevention and preemption (before the attack):** These mechanisms enable the victim to endure attack attempts without denying service to legitimate clients. Techniques include enforcing policies for resource consumption and providing backup resources available on demand. In addition, prevention mechanisms modify systems and protocols on the Internet to reduce the possibility of DDoS attacks.
- **Attack detection and filtering (during the attack):** These mechanisms attempt to detect the attack as it begins and respond immediately. This minimizes the impact of the attack on the target. Detection involves looking for suspicious patterns of behavior. Response involves filtering out packets likely to be part of the attack.

- **Attack source traceback and identification (during and after the attack):** This is an attempt to identify the source of the attack as a first step in preventing future attacks. However, this method typically does not yield results fast enough, if at all, to mitigate an ongoing attack.

The challenge in coping with DDoS attacks is the sheer number of ways in which they can operate. Thus DDoS countermeasures must evolve with the threat.

19.4. Recommended Reading and Web Sites

For a thorough understanding of viruses, the book to read is [SZOR05]. Another excellent treatment is [HARL01]. Good overview articles on viruses and worms are [CASS01], [FORR97], [KEPH97], and [NACH97]. [MEIN01] provides a good treatment of the Code Red worm.

[PATR04] is a worthwhile survey of DDoS attacks. [MIRK04] is a thorough description of the variety of DDoS attacks and countermeasures. [CHAN02] is a good examination of DDoS defense strategies.

CASS01 Cass, S. "Anatomy of Malice." *IEEE Spectrum*, November 2001.

CHAN02 Chang, R. "Defending Against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial." *IEEE Communications Magazine*, October 2002.

FORR97 Forrest, S.; Hofmeyr, S.; and Somayaji, A. "Computer Immunology." *Communications of the ACM*, October 1997.

HARL01 Harley, D.; Slade, R.; and Gattiker, U. *Viruses Revealed*. New York: Osborne/McGraw-Hill, 2001.

KEPH97 Kephart, J.; Sorkin, G.; Chess, D.; and White, S. "Fighting Computer Viruses." *Scientific American*, November 1997.

MEIN01 Meinel, C. "Code Red for the Web." *Scientific American*, October 2001.

MIRK04 Mirkovic, J., and Relher, P. "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms." *ACM SIGCOMM Computer Communications Review*, April 2004.

NACH97 Nachenberg, C. "Computer Virus-Antivirus Coevolution." *Communications of the ACM*, January 1997.

PATR04 Patrikakis, C.; Masikos, M.; and Zouraraki, O. "Distributed Denial of Service Attacks." *The Internet Protocol Journal*, December 2004.

SZOR05 Szor, P., *The Art of Computer Virus Research and Defense*. Reading, MA: Addison-Wesley, 2005.

Recommended Web Sites

- **AntiVirus Online:** IBM's site on virus information
- **Vmyths:** Dedicated to exposing virus hoaxes and dispelling misconceptions about real viruses
- **DDoS Attacks/Tools:** Extensive list of links and documents

PREV NEXT

19.5. Key Terms, Review Questions, and Problems

Key Terms

auto-rooter

backdoor

digital immune system

direct DDoS attack

distributed denial of service (DDoS)

downloaders

e-mail virus

exploits

flooder

keylogger

kit

logic bomb

macro virus

malicious software (malware)

polymorphic virus

reflector DDoS attack

rootkit

spammer program

stealth virus

trapdoor

trojan horse

virus

worm

zombie

Review Questions

19.1 What is the role of compression in the operation of a virus?

19.2 What is the role of encryption in the operation of a virus?

19.3 What are typical phases of operation of a virus or worm?

19.4 In general terms, how does a worm propagate?

19.5 What is a digital immune system?

19.6 How does behavior-blocking software work?

19.7 What is a DDoS?

Problems

19.1 There is a flaw in the virus program of Figure 19.1. What is it?

19.2 The question arises as to whether it is possible to develop a program that can analyze a piece of software to determine if it is a virus. Consider that we have a program D that is supposed to be able to do that. That is, for any program P, if we run D(P), the result returned is TRUE (P is a virus) or FALSE (P is not a virus). Now consider the following program:

```
Program CV :=
  { . . .
  main-program :=
        {if D(CV) then goto next:
               else infect-executable;
        }
next:
  }
```

In the preceding program, infect-executable is a module that scans memory for executable programs and replicates itself in those programs. Determine if D can correctly decide whether CV is a virus.

PREV NEXT

Chapter 20. Firewalls

The function of a strong position is to make the forces holding it practically unassailable.

On War, Carl Von Clausewitz

On the day that you take up your command, block the frontier passes, destroy the official tallies, and stop the passage of all emissaries.

The Art of War, Sun Tzu

Key Points

- A firewall forms a barrier through which the traffic going in each direction must pass. A firewall security policy dictates which traffic is authorized to pass in each direction.
- A firewall may be designed to operate as a filter at the level of IP packets, or may operate at a higher protocol layer.
- A trusted system is a computer and operating system that can be verified to implement a given security policy. Typically, the focus of a trusted system is access control. A policy is implemented that dictates what objects may be accessed by what subjects.
- The common criteria for information technology security is an international standards initiative to define a common set of security requirements and a systematic means of evaluating products against those requirements.

Firewalls can be an effective means of protecting a local system or network of systems from network-based security threats while at the same time affording access to the outside world via wide area networks and the Internet.

We begin this chapter with an overview of the functionality and design principles of firewalls. Next, we address the issue of the security of the firewall itself and, in particular, the concept of a trusted system, or secure operating system.

PREV NEXT

20.1. Firewall Design Principles

Information systems in corporations, government agencies, and other organizations have undergone a steady evolution:

- Centralized data processing system, with a central mainframe supporting a number of directly connected terminals
- Local area networks (LANs) interconnecting PCs and terminals to each other and the mainframe
- Premises network, consisting of a number of LANs, interconnecting PCs, servers, and perhaps a mainframe or two

---

- Enterprise-wide network, consisting of multiple, geographically distributed premises networks interconnected by a private wide area network (WAN)
- Internet connectivity, in which the various premises networks all hook into the Internet and may or may not also be connected by a private WAN

Internet connectivity is no longer optional for organizations. The information and services available are essential to the organization. Moreover, individual users within the organization want and need Internet access, and if this is not provided via their LAN, they will use dial-up capability from their PC to an Internet service provider (ISP). However, while Internet access provides benefits to the organization, it enables the outside world to reach and interact with local network assets. This creates a threat to the organization. While it is possible to equip each workstation and server on the premises network with strong security features, such as intrusion protection, this is not a practical approach. Consider a network with hundreds or even thousands of systems, running a mix of various versions of UNIX, plus Windows. When a security flaw is discovered, each potentially affected system must be upgraded to fix that flaw. The alternative, increasingly accepted, is the firewall. The firewall is inserted between the premises network and the Internet to establish a controlled link and to erect an outer security wall or perimeter. The aim of this perimeter is to protect the premises network from Internet-based attacks and to provide a single choke point where security and audit can be imposed. The firewall may be a single computer system or a set of two or more systems that cooperate to perform the firewall function.

In this section, we look first at the general characteristics of firewalls. Then we look at the types of firewalls currently in common use. Finally, we examine some of the most common firewall configurations.

Firewall Characteristics

[BELL94b] lists the following design goals for a firewall:

1. All traffic from inside to outside, and vice versa, must pass through the firewall. This is achieved by physically blocking all access to the local network except via the firewall. Various configurations are possible, as explained later in this section.

2. Only authorized traffic, as defined by the local security policy, will be allowed to pass. Various types of firewalls are used, which implement various types of security policies, as explained later

in this section.

3.

The firewall itself is immune to penetration. This implies that use of a trusted system with a secure operating system. This topic is discussed in Section 20.2.

[SMIT97] lists four general techniques that firewalls use to control access and enforce the site's security policy. Originally, firewalls focused primarily on service control, but they have since evolved to provide all four:

- **Service control:** Determines the types of Internet services that can be accessed, inbound or outbound. The firewall may filter traffic on the basis of IP address and TCP port number; may provide proxy software that receives and interprets each service request before passing it on; or may host the server software itself, such as a Web or mail service.

- **Direction control:** Determines the direction in which particular service requests may be initiated and allowed to flow through the firewall.
- **User control:** Controls access to a service according to which user is attempting to access it. This feature is typically applied to users inside the firewall perimeter (local users). It may also be applied to incoming traffic from external users; the latter requires some form of secure authentication technology, such as is provided in IPSec (Chapter 16).
- **Behavior control:** Controls how particular services are used. For example, the firewall may filter e-mail to eliminate spam, or it may enable external access to only a portion of the information on a local Web server.

Before proceeding to the details of firewall types and configurations, it is best to summarize what one can expect from a firewall. The following capabilities are within the scope of a firewall:

1.

A firewall defines a single choke point that keeps unauthorized users out of the protected network, prohibits potentially vulnerable services from entering or leaving the network, and provides protection from various kinds of IP spoofing and routing attacks. The use of a single choke point simplifies security management because security capabilities are consolidated on a single system or set of systems.

2.

A firewall provides a location for monitoring security-related events. Audits and alarms can be implemented on the firewall system.

3.

A firewall is a convenient platform for several Internet functions that are not security related. These include a network address translator, which maps local addresses to Internet addresses, and a network management function that audits or logs Internet usage.

4.

A firewall can serve as the platform for IPSec. Using the tunnel mode capability described in Chapter 16, the firewall can be used to implement virtual private networks.

Firewalls have their limitations, including the following:

1. The firewall cannot protect against attacks that bypass the firewall. Internal systems may have dial-out capability to connect to an ISP. An internal LAN may support a modem pool that provides dial-in capability for traveling employees and telecommuters.

2. The firewall does not protect against internal threats, such as a disgruntled employee or an employee who unwittingly cooperates with an external attacker.

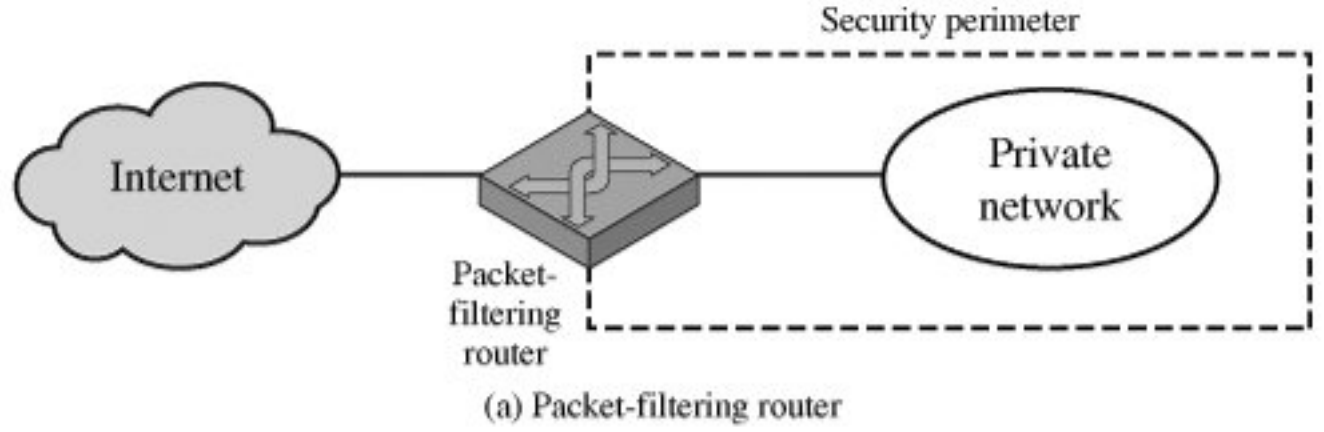
3. The firewall cannot protect against the transfer of virus-infected programs or files. Because of the variety of operating systems and applications supported inside the perimeter, it would be impractical and perhaps impossible for the firewall to scan all incoming files, e-mail, and messages for viruses.

Types of Firewalls

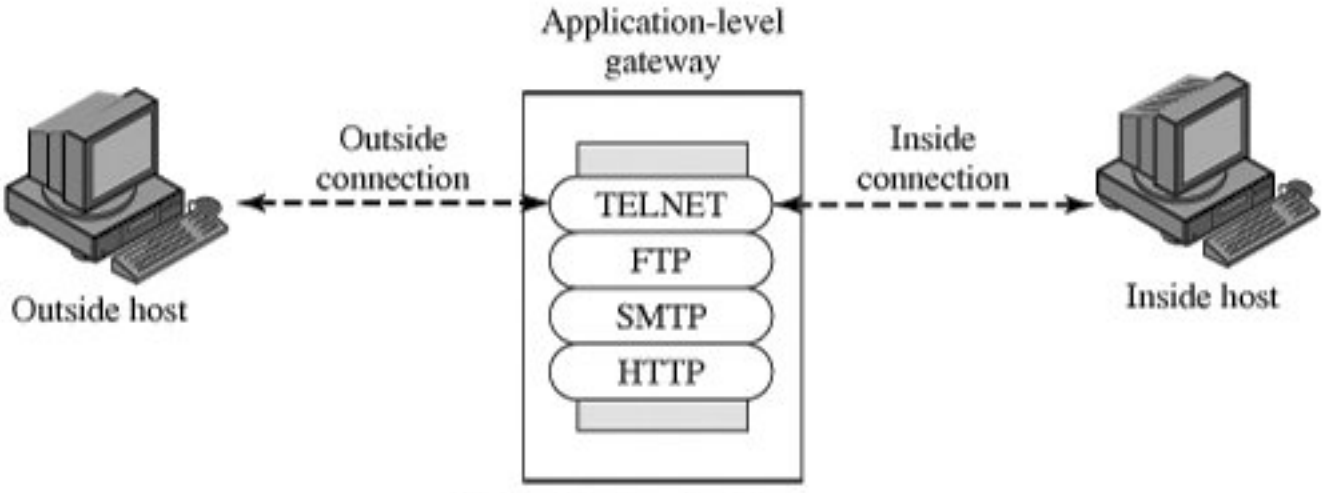
Figure 20.1 illustrates the three common types of firewalls: packet filters, application-level gateways, and circuit-level gateways. We examine each of these in turn.

Figure 20.1. Firewall Types

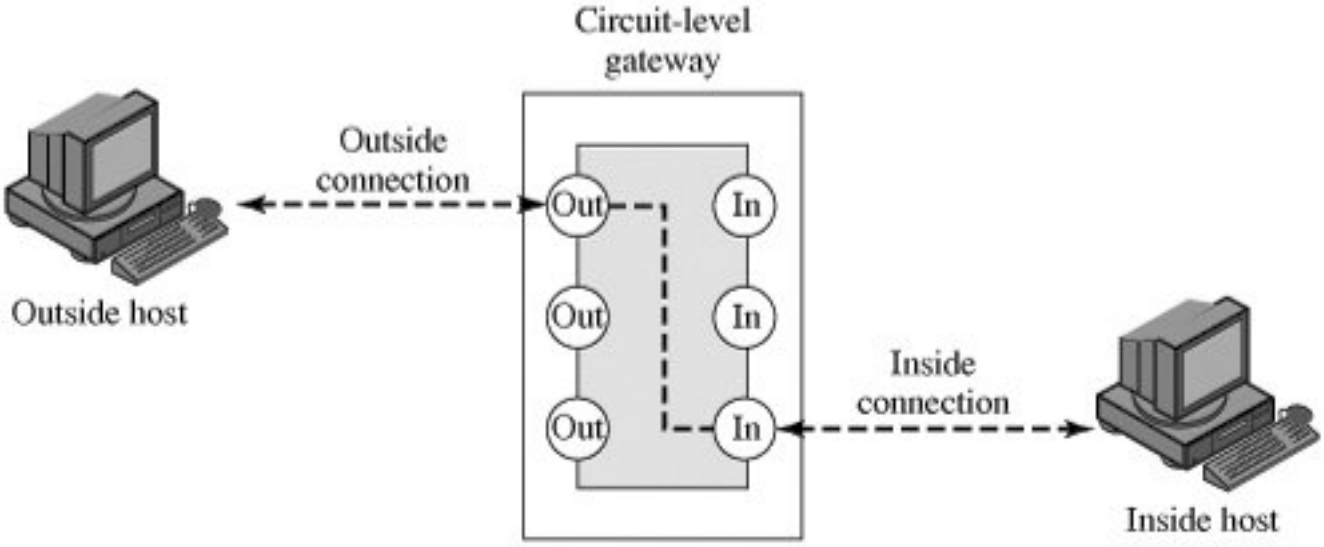
[View full size image]



(a) Packet-filtering router



(b) Application-level gateway



(c) Circuit-level gateway

Packet-Filtering Router

A packet-filtering router applies a set of rules to each incoming and outgoing IP packet and then forwards or discards the packet. The router is typically configured to filter packets going in both directions (from and to the internal network). Filtering rules are based on information contained in a network packet:

- **Source IP address:** The IP address of the system that originated the IP packet (e.g.,

192.178.1.1)

- **Destination IP address:** The IP address of the system the IP packet is trying to reach (e.g., 192.168.1.2)
- **Source and destination transport-level address:** The transport level (e.g., TCP or UDP) port number, which defines applications such as SNMP or TELNET

- **IP protocol field:** Defines the transport protocol
- **Interface:** For a router with three or more ports, which interface of the router the packet came from or which interface of the router the packet is destined for

The packet filter is typically set up as a list of rules based on matches to fields in the IP or TCP header. If there is a match to one of the rules, that rule is invoked to determine whether to forward or discard the packet. If there is no match to any rule, then a default action is taken. Two default policies are possible:

- **Default** = *discard*: That which is not expressly permitted is prohibited.
- **Default** = *forward*: That which is not expressly prohibited is permitted.

The default discard policy is more conservative. Initially, everything is blocked, and services must be added on a case-by-case basis. This policy is more visible to users, who are more likely to see the firewall as a hindrance. The default forward policy increases ease of use for end users but provides reduced security; the security administrator must, in essence, react to each new security threat as it becomes known.

Table 20.1, from [BELL94b], gives some examples of packet-filtering rule sets. In each set, the rules are applied top to bottom. The "*" in a field is a wildcard designator that matches everything. We assume that the default = discard policy is in force.

Table 20.1. Packet-Filtering Examples

(This item is displayed on page 627 in the print version)

A	**action**	**ourhost**	**port**	**theirhost**	**port**	**comment**	
	block	*	*	SPIGOT	*	we don't trust these people	
	allow	OUR-GW	25	*	*	connection to our SMTP port	
B	**action**	**ourhost**	**port**	**theirhost**	**port**	**comment**	
	block	*	*	*	*	default	
C	**action**	**ourhost**	**port**	**theirhost**	**port**	**comment**	
	allow	*	*	*	25	connection to their SMTP port	
D	**action**	**src**	**port**	**dest**	**port**	**flags**	**comment**
	allow	{our hosts}	*	*	25		our packets to their SMTP port
	allow	*	25	*	*	ACK	their replies
	action	**src**	**port**	**dest**	**port**	**flags**	**comment**
	allow	{our hosts}	*	*	*		our outgoing calls

E	allow	*	*	*	*	ACK	replies to our calls
	allow	*	*	*	>1024		traffic to nonservers

A.

Inbound mail is allowed (port 25 is for SMTP incoming), but only to a gateway host. However, packets from a particular external host, SPIGOT, are blocked because that host has a history of sending massive files in e-mail messages.

B.

This is an explicit statement of the default policy. All rule sets include this rule implicitly as the last rule.

C.

This rule set is intended to specify that any inside host can send mail to the outside. A TCP packet with a destination port of 25 is routed to the SMTP server on the destination machine. The problem with this rule is that the use of port 25 for SMTP receipt is only a default; an outside machine could be configured to have some other application linked to port 25. As this rule is written, an attacker could gain access to internal machines by sending packets with a TCP source port number of 25.

D.

This rule set achieves the intended result that was not achieved in C. The rules take advantage of a feature of TCP connections. Once a connection is set up, the ACK flag of a TCP segment is set to acknowledge segments sent from the other side. Thus, this rule set states that it allows IP packets where the source IP address is one of a list of designated internal hosts and the destination TCP port number is 25. It also allows incoming packets with a source port number of 25 that include the ACK flag in the TCP segment. Note that we explicitly designate source and destination systems to define these rules explicitly.

E.

This rule set is one approach to handling FTP connections. With FTP, two TCP connections are used: a control connection to set up the file transfer and a data connection for the actual file transfer. The data connection uses a different port number that is dynamically assigned for the transfer. Most servers, and hence most attack targets, live on low-numbered ports; most outgoing calls tend to use a higher-numbered port, typically above 1023. Thus, this rule set allows

---

Packets that originate internally

Reply packets to a connection initiated by an internal machine

Packets destined for a high-numbered port on an internal machine

This scheme requires that the systems be configured so that only the appropriate port numbers are in use.

Rule set E points out the difficulty in dealing with applications at the packet-filtering level. Another way to deal with FTP and similar applications is an application-level gateway, described later in this section.

One advantage of a packet-filtering router is its simplicity. Also, packet filters typically are transparent to users and are very fast. [WACK02] lists the following weaknesses of packet filter firewalls:

- Because packet filter firewalls do not examine upper-layer data, they cannot prevent attacks that employ application-specific vulnerabilities or functions. For example, a packet filter firewall cannot block specific application commands; if a packet filter firewall allows a given application, all functions available within that application will be permitted.
- Because of the limited information available to the firewall, the logging functionality present in packet filter firewalls is limited. Packet filter logs normally contain the same information used to make access control decisions (source address, destination address, and traffic type).
- Most packet filter firewalls do not support advanced user authentication schemes. Once again, this limitation is mostly due to the lack of upper-layer functionality by the firewall.
- They are generally vulnerable to attacks and exploits that take advantage of problems within the TCP/IP specification and protocol stack, such as *network layer address spoofing*. Many packet filter firewalls cannot detect a network packet in which the OSI Layer 3 addressing information has been altered. Spoofing attacks are generally employed by intruders to bypass the security controls implemented in a firewall platform.
- Finally, due to the small number of variables used in access control decisions, packet filter firewalls are susceptible to security breaches caused by improper configurations. In other words, it is easy to accidentally configure a packet filter firewall to allow traffic types, sources, and destinations that should be denied based on an organization's information security policy.

Some of the attacks that can be made on packet-filtering routers and the appropriate countermeasures are the following:

- **IP address spoofing:** The intruder transmits packets from the outside with a source IP address field containing an address of an internal host. The attacker hopes that the use of a spoofed address will allow penetration of systems that employ simple source address security, in which packets from specific trusted internal hosts are accepted. The countermeasure is to discard packets with an inside source address if the packet arrives on an external interface.

- **Source routing attacks:** The source station specifies the route that a packet should take as it crosses the Internet, in the hopes that this will bypass security measures that do not analyze the source routing information. The countermeasure is to discard all packets that use this option.
- **Tiny fragment attacks:** The intruder uses the IP fragmentation option to create extremely small fragments and force the TCP header information into a separate packet fragment. This attack is designed to circumvent filtering rules that depend on TCP header information. Typically, a packet filter will make a filtering decision on the first fragment of a packet. All subsequent fragments of that packet are filtered out solely on the basis that they are part of the packet whose first fragment was rejected. The attacker hopes that the filtering router examines only the first fragment and that the remaining fragments are passed through. A tiny fragment attack can be defeated by enforcing a rule that the first fragment of a packet must contain a predefined minimum amount of the transport header. If the first fragment is rejected, the filter can remember the packet and discard all subsequent fragments.

Stateful Inspection Firewalls

A traditional packet filter makes filtering decisions on an individual packet basis and does not take into

consideration any higher layer context. To understand what is meant by context and why a traditional packet filter is limited with regard to context, a little background is needed. Most standardized applications that run on top of TCP follow a client/server model. For example, for the Simple Mail Transfer Protocol (SMTP), e-mail is transmitted from a client system to a server system. The client system generates new e-mail messages, typically from user input. The server system accepts incoming e-mail messages and places them in the appropriate user mailboxes. SMTP operates by setting up a TCP connection between client and server, in which the TCP server port number, which identifies the SMTP server application, is 25. The TCP port number for the SMTP client is a number between 1024 and 65535 that is generated by the SMTP client.

In general, when an application that uses TCP creates a session with a remote host, it creates a TCP connection in which the TCP port number for the remote (server) application is a number less than 1024 and the TCP port number for the local (client) application is a number between 1024 and 65535. The numbers less than 1024 are the "well-known" port numbers and are assigned permanently to particular applications (e.g., 25 for server SMTP). The numbers between 1024 and 65535 are generated dynamically and have temporary significance only for the lifetime of a TCP connection.

A simple packet-filtering firewall must permit inbound network traffic on all these high-numbered ports for TCP-based traffic to occur. This creates a vulnerability that can be exploited by unauthorized users.

A stateful inspection packet filter tightens up the rules for TCP traffic by creating a directory of outbound TCP connections, as shown in Table 20.2. There is an entry for each currently established connection. The packet filter will now allow incoming traffic to high-numbered ports only for those packets that fit the profile of one of the entries in this directory.

Table 20.2. Example Stateful Firewall Connection State Table [WACK02]

(This item is displayed on page 630 in the print version)

Source Address	Source Port	Destination Address	Destination Port	Connection State
192.168.1.100	1030	210.9.88.29	80	Established
192.168.1.102	1031	216.32.42.123	80	Established
192.168.1.101	1033	173.66.32.122	25	Established
192.168.1.106	1035	177.231.32.12	79	Established
223.43.21.231	1990	192.168.1.6	80	Established
219.22.123.32	2112	192.168.1.6	80	Established
210.99.212.18	3321	192.168.1.6	80	Established
24.102.32.23	1025	192.168.1.6	80	Established
223.212.212	1046	192.168.1.6	80	Established

Application-Level Gateway

An application-level gateway, also called a proxy server, acts as a relay of application-level traffic

(Figure 20.1b). The user contacts the gateway using a TCP/IP application, such as Telnet or FTP, and the gateway asks the user for the name of the remote host to be accessed. When the user responds and provides a valid user ID and authentication information, the gateway contacts the application on the remote host and relays TCP segments containing the application data between the two endpoints. If the gateway does not implement the proxy code for a specific application, the service is not supported and cannot be forwarded across the firewall. Further, the gateway can be configured to support only specific features of an application that the network administrator considers acceptable while denying all other features.

Application-level gateways tend to be more secure than packet filters. Rather than trying to deal with the numerous possible combinations that are to be allowed and forbidden at the TCP and IP level, the application-level gateway need only scrutinize a few allowable applications. In addition, it is easy to log and audit all incoming traffic at the application level.

A prime disadvantage of this type of gateway is the additional processing overhead on each connection. In effect, there are two spliced connections between the end users, with the gateway at the splice point, and the gateway must examine and forward all traffic in both directions.

Circuit-Level Gateway

A third type of firewall is the circuit-level gateway (Figure 20.1c). This can be a stand-alone system or it can be a specialized function performed by an application-level gateway for certain applications. A circuit-level gateway does not permit an end-to-end TCP connection; rather, the gateway sets up two TCP connections, one between itself and a TCP user on an inner host and one between itself and a TCP user on an outside host. Once the two connections are established, the gateway typically relays TCP segments from one connection to the other without examining the contents. The security function consists of determining which connections will be allowed.

A typical use of circuit-level gateways is a situation in which the system administrator trusts the internal users. The gateway can be configured to support application-level or proxy service on inbound connections and circuit-level functions for outbound connections. In this configuration, the gateway can incur the processing overhead of examining incoming application data for forbidden functions but does not incur that overhead on outgoing data.

An example of a circuit-level gateway implementation is the SOCKS package [KOBL92]; version 5 of SOCKS is defined in RFC 1928. The RFC defines SOCKS in the following fashion:

> The protocol described here is designed to provide a framework for client-server applications in both the TCP and UDP domains to conveniently and securely use the services of a network firewall. The protocol is conceptually a "shim-layer" between the application layer and the transport layer, and as such does not provide network-layer gateway services, such as forwarding of ICMP messages.

SOCKS consists of the following components:

- The SOCKS server, which runs on a UNIX-based firewall.
- The SOCKS client library, which runs on internal hosts protected by the firewall.
- SOCKS-ified versions of several standard client programs such as FTP and TELNET. The

implementation of the SOCKS protocol typically involves the recompilation or relinking of TCP-based client applications to use the appropriate encapsulation routines in the SOCKS library.

When a TCP-based client wishes to establish a connection to an object that is reachable only via a firewall (such determination is left up to the implementation), it must open a TCP connection to the appropriate SOCKS port on the SOCKS server system. The SOCKS service is located on TCP port 1080. If the connection request succeeds, the client enters a negotiation for the authentication method to be used, authenticates with the chosen method, and then sends a relay request. The SOCKS server evaluates the request and either establishes the appropriate connection or denies it. UDP exchanges are handled in a similar fashion. In essence, a TCP connection is opened to authenticate a user to send and receive UDP segments, and the UDP segments are forwarded as long as the TCP connection is open.

Bastion Host

A bastion host is a system identified by the firewall administrator as a critical strong point in the network's security. Typically, the bastion host serves as a platform for an application-level or circuit-level gateway. Common characteristics of a bastion host include the following:

- The bastion host hardware platform executes a secure version of its operating system, making it a trusted system.
- Only the services that the network administrator considers essential are installed on the bastion host. These include proxy applications such as Telnet, DNS, FTP, SMTP, and user authentication.
- The bastion host may require additional authentication before a user is allowed access to the proxy services. In addition, each proxy service may require its own authentication before granting user access.
- Each proxy is configured to support only a subset of the standard application's command set.

- Each proxy is configured to allow access only to specific host systems. This means that the limited command/feature set may be applied only to a subset of systems on the protected network.
- Each proxy maintains detailed audit information by logging all traffic, each connection, and the duration of each connection. The audit log is an essential tool for discovering and terminating intruder attacks.
- Each proxy module is a very small software package specifically designed for network security. Because of its relative simplicity, it is easier to check such modules for security flaws. For example, a typical UNIX mail application may contain over 20,000 lines of code, while a mail proxy may contain fewer than 1000.
- Each proxy is independent of other proxies on the bastion host. If there is a problem with the operation of any proxy, or if a future vulnerability is discovered, it can be uninstalled without affecting the operation of the other proxy applications. Also, if the user population requires support for a new service, the network administrator can easily install the required proxy on the bastion host.
- A proxy generally performs no disk access other than to read its initial configuration file. This makes it difficult for an intruder to install Trojan horse sniffers or other dangerous files on the bastion host.
- Each proxy runs as a nonprivileged user in a private and secured directory on the bastion host.

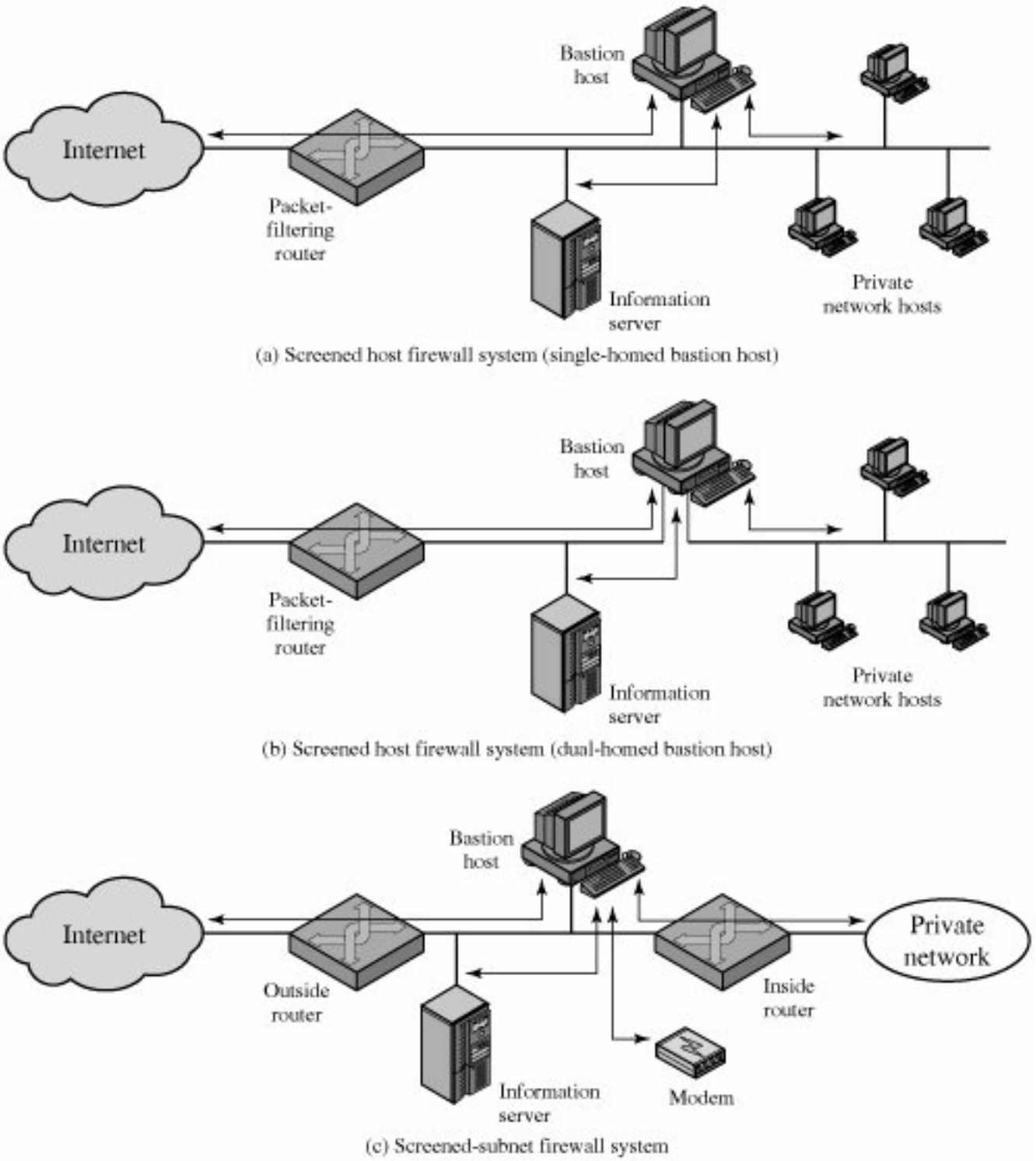
Firewall Configurations

In addition to the use of a simple configuration consisting of a single system, such as a single packet-filtering router or a single gateway (Figure 20.1), more complex configurations are possible and indeed more common. Figure 20.2 illustrates three common firewall configurations. We examine each of these in turn.

Figure 20.2. Firewall Configurations

(This item is displayed on page 633 in the print version)

[View full size image]



In the **screened host firewall, single-homed bastion** configuration (Figure 20.2a), the firewall consists of two systems: a packet-filtering router and a bastion host. Typically, the router is configured so that

1.

For traffic from the Internet, only IP packets destined for the bastion host are allowed in.

2.

For traffic from the internal network, only IP packets from the bastion host are allowed out.

The bastion host performs authentication and proxy functions. This configuration has greater security than simply a packet-filtering router or an application-level gateway alone, for two reasons. First, this configuration implements both packet-level and application-level filtering, allowing for considerable flexibility in defining security policy. Second, an intruder must generally penetrate two separate systems before the security of the internal network is compromised.

This configuration also affords flexibility in providing direct Internet access. For example, the internal network may include a public information server, such as a Web server, for which a high level of security is not required. In that case, the router can be configured to allow direct traffic between the information server and the Internet.

In the single-homed configuration just described, if the packet-filtering router is completely compromised, traffic could flow directly through the router between the Internet and other hosts on the private network. The **screened host firewall, dual-homed bastion** configuration physically prevents such a security breach (Figure 20.2b). The advantages of dual layers of security that were present in the previous configuration are present here as well. Again, an information server or other hosts can be allowed direct communication with the router if this is in accord with the security policy.

The **screened subnet firewall** configuration of Figure 20.2c is the most secure of those we have considered. In this configuration, two packet-filtering routers are used, one between the bastion host and the Internet and one between the bastion host and the internal network. This configuration creates an isolated subnetwork, which may consist of simply the bastion host but may also include one or more information servers and modems for dial-in capability. Typically, both the Internet and the internal network have access to hosts on the screened subnet, but traffic across the screened subnet is blocked. This configuration offers several advantages:

- There are now three levels of defense to thwart intruders.
- The outside router advertises only the existence of the screened subnet to the Internet; therefore, the internal network is invisible to the Internet.
- Similarly, the inside router advertises only the existence of the screened subnet to the internal network; therefore, the systems on the inside network cannot construct direct routes to the Internet.

20.2. Trusted Systems

One way to enhance the ability of a system to defend against intruders and malicious programs is to implement trusted system technology. This section provides a brief overview of this topic. We begin by looking at some basic concepts of data access control.

Data Access Control

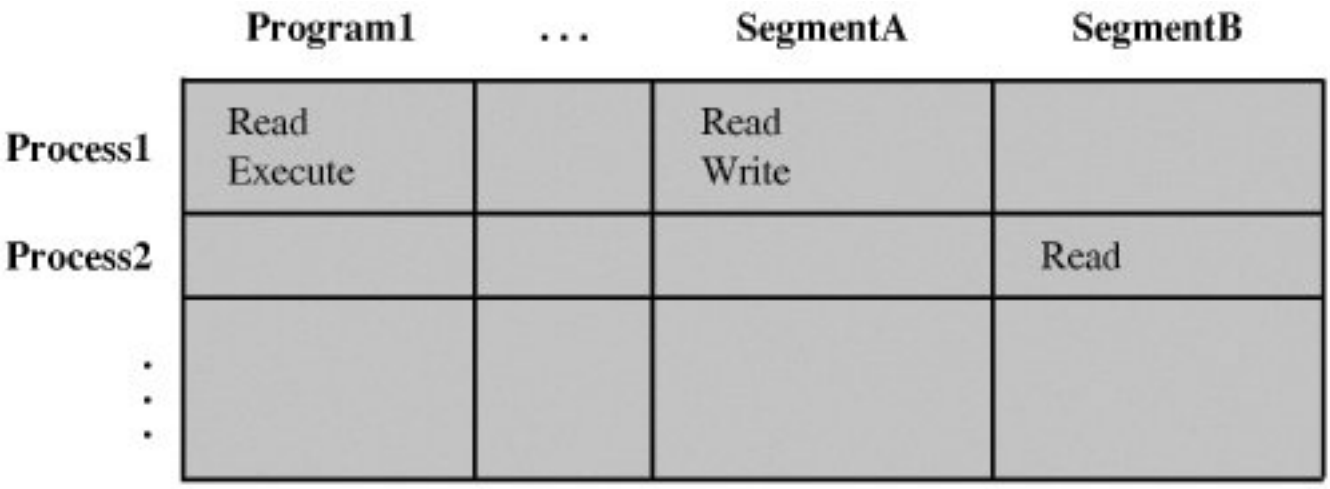
Following successful logon, the user has been granted access to one or a set of hosts and applications. This is generally not sufficient for a system that includes sensitive data in its database. Through the user access control procedure, a user can be identified to the system. Associated with each user, there can be a profile that specifies permissible operations and file accesses. The operating system can then enforce rules based on the user profile. The database management system, however, must control access to specific records or even portions of records. For example, it may be permissible for anyone in administration to obtain a list of company personnel, but only selected individuals may have access to salary information. The issue is more than just one of level of detail. Whereas the operating system may grant a user permission to access a file or use an application, following which there are no further security checks, the database management system must make a decision on each individual access attempt. That decision will depend not only on the user's identity but also on the specific parts of the data being accessed and even on the information already divulged to the user.

A general model of access control as exercised by a file or database management system is that of an **access matrix** (Figure 20.3a). The basic elements of the model are as follows:

- **Subject**: An entity capable of accessing objects. Generally, the concept of subject equates with that of process. Any user or application actually gains access to an object by means of a process that represents that user or application.

- **Object:** Anything to which access is controlled. Examples include files, portions of files, programs, and segments of memory.
- **Access right**: The way in which an object is accessed by a subject. Examples are read, write, and execute.

Figure 20.3. Access Control Structure



(a) Access matrix

Access control list for Program1:
Process1 (Read, Execute)

Access control list for SegmentA:
Process1 (Read, Write)

Access control list for SegmentB:
Process2 (Read)

(b) Access control list

Capability list for Process1:
Program1 (Read, Execute)
SegmentA (Read, Write)

Capability list for Process2:
Segment B (Read)

(c) Capability list

One axis of the matrix consists of identified subjects that may attempt data access. Typically, this list will consist of individual users or user groups, although access could be controlled for terminals, hosts, or applications instead of or in addition to users. The other axis lists the objects that may be accessed. At the greatest level of detail, objects may be individual data fields. More aggregate groupings, such as records, files, or even the entire database, may also be objects in the matrix. Each entry in the matrix indicates the access rights of that subject for that object.

In practice, an access matrix is usually sparse and is implemented by decomposition in one of two ways. The matrix may be decomposed by columns, yielding **access control lists** (Figure 20.3b). Thus, for each object, an access control list lists users and their permitted access rights. The access control list may contain a default, or public, entry. This allows users that are not explicitly listed as having special

rights to have a default set of rights. Elements of the list may include individual users as well as groups of users.

Decomposition by rows yields **capability tickets** (Figure 20.3c). A capability ticket specifies authorized objects and operations for a user. Each user has a number of tickets and may be authorized to loan or give them to others. Because tickets may be dispersed around the system, they present a greater security problem than access control lists. In particular, the ticket must be unforgeable. One way to accomplish this is to have the operating system hold all tickets on behalf of users. These tickets would have to be held in a region of memory inaccessible to users.

The Concept of Trusted Systems

Much of what we have discussed so far has been concerned with protecting a given message or item from passive or active attacks by a given user. A somewhat different but widely applicable requirement is to protect data or resources on the basis of levels of security. This is commonly found in the military, where information is categorized as unclassified (U), confidential (C), secret (S), top secret (TS), or beyond. This concept is equally applicable in other areas, where information can be organized into gross categories and users can be granted clearances to access certain categories of data. For example, the highest level of security might be for strategic corporate planning documents and data, accessible by only corporate officers and their staff; next might come sensitive financial and personnel data, accessible only by administration personnel, corporate officers, and so on.

When multiple categories or levels of data are defined, the requirement is referred to as **multilevel security**. The general statement of the requirement for multilevel security is that a subject at a high level may not convey information to a subject at a lower or noncomparable level unless that flow accurately reflects the will of an authorized user. For implementation purposes, this requirement is in two parts and is simply stated. A multilevel secure system must enforce the following:

- **No read up:** A subject can only read an object of less or equal security level. This is referred to in the literature as the **Simple Security Property**.
- **No write down:** A subject can only write into an object of greater or equal security level. This is referred to in the literature as the ***-Property**[1] (pronounced *star property*).

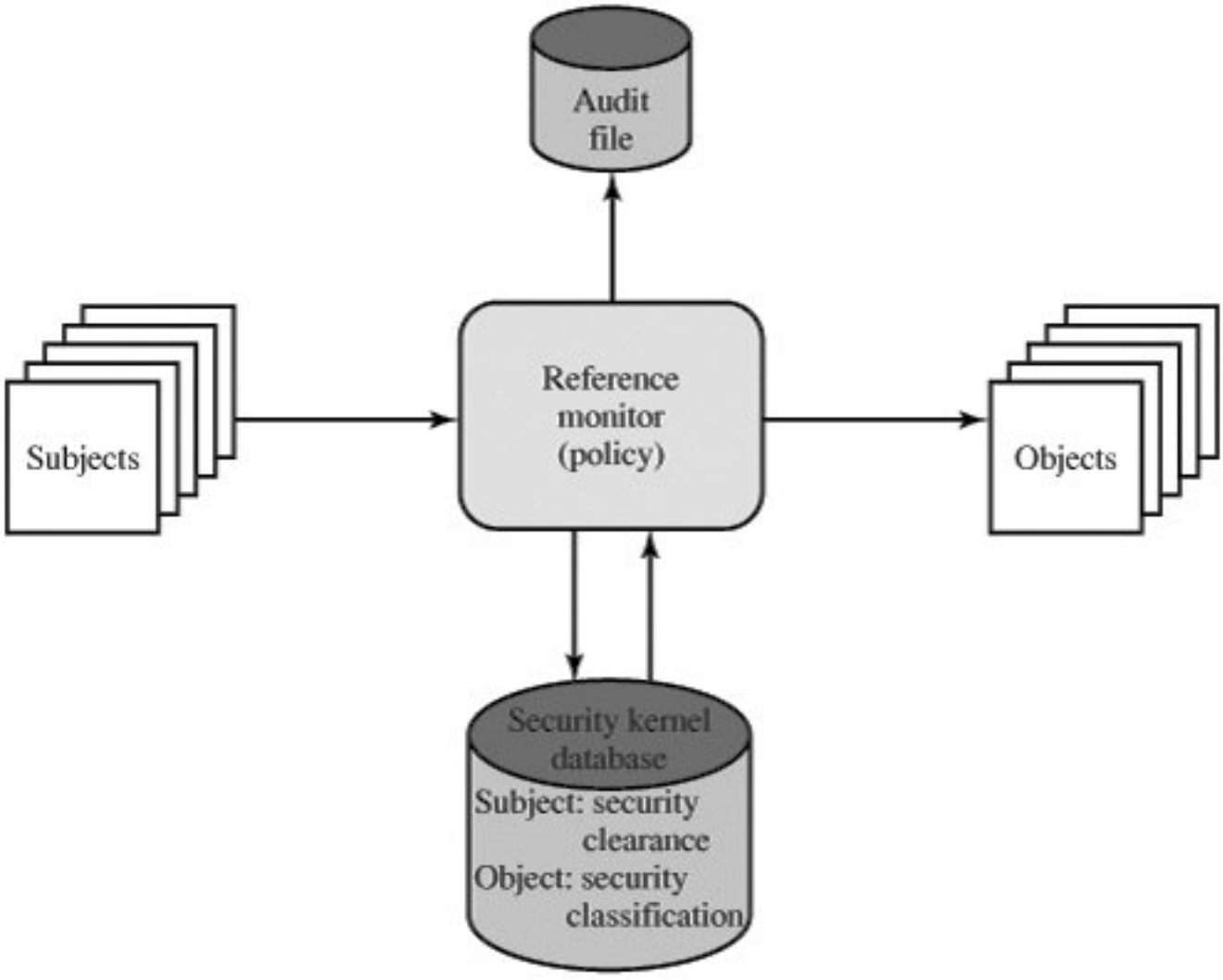
[1] The "*" does not stand for anything. No one could think of an appropriate name for the property during the writing of the first report on the model. The asterisk was a dummy character entered in the draft so that a text editor could rapidly find and replace all instances of its use once the property was named. No name was ever devised, and so the report was published with the "*" intact.

These two rules, if properly enforced, provide multilevel security. For a data processing system, the approach that has been taken, and has been the object of much research and development, is based on the *reference monitor* concept. This approach is depicted in Figure 20.4. The reference monitor is a controlling element in the hardware and operating system of a computer that regulates the access of subjects to objects on the basis of security parameters of the subject and object. The reference monitor has access to a file, known as the *security kernel database*, that lists the access privileges (security clearance) of each subject and the protection attributes (classification level) of each object. The reference monitor enforces the security rules (no read up, no write down) and has the following properties:

- **Complete mediation:** The security rules are enforced on every access, not just, for example, when a file is opened.
- **Isolation:** The reference monitor and database are protected from unauthorized modification.
- **Verifiability:** The reference monitor's correctness must be provable. That is, it must be possible to demonstrate mathematically that the reference monitor enforces the security rules and provides complete mediation and isolation.

Figure 20.4. Reference Monitor Concept

[View full size image]



These are stiff requirements. The requirement for complete mediation means that every access to data within main memory and on disk and tape must be mediated. Pure software implementations impose too high a performance penalty to be practical; the solution must be at least partly in hardware. The requirement for isolation means that it must not be possible for an attacker, no matter how clever, to change the logic of the reference monitor or the contents of the security kernel database. Finally, the requirement for mathematical proof is formidable for something as complex as a general-purpose computer. A system that can provide such verification is referred to as a **trusted system**.

A final element illustrated in Figure 20.4 is an audit file. Important security events, such as detected security violations and authorized changes to the security kernel database, are stored in the audit file.

In an effort to meet its own needs and as a service to the public, the U.S. Department of Defense in 1981 established the Computer Security Center within the National Security Agency (NSA) with the goal of encouraging the widespread availability of trusted computer systems. This goal is realized through the center's Commercial Product Evaluation Program. In essence, the center attempts to evaluate commercially available products as meeting the security requirements just outlined. The center classifies evaluated products according to the range of security features that they provide. These evaluations are needed for Department of Defense procurements but are published and freely available. Hence, they can serve as guidance to commercial customers for the purchase of commercially available, off-the-shelf equipment.

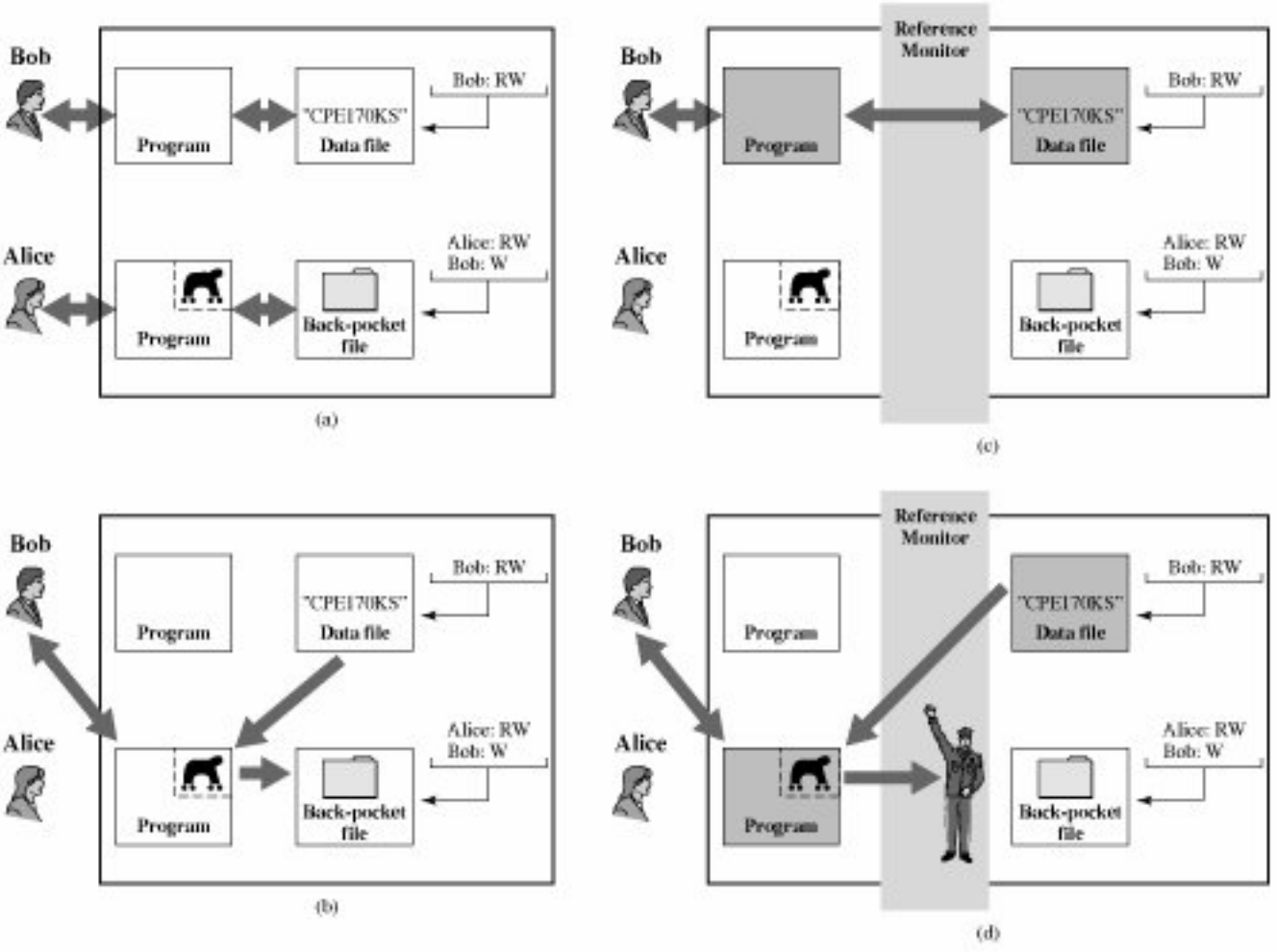
Trojan Horse Defense

One way to secure against Trojan horse attacks is the use of a secure, trusted operating system. Figure 20.5 illustrates an example. In this case, a Trojan horse is used to get around the standard security mechanism used by most file management and operating systems: the access control list. In this example, a user named Bob interacts through a program with a data file containing the critically sensitive character string "CPE170KS." User Bob has created the file with read/write permission provided only to programs executing on his own behalf: that is, only processes that are owned by Bob may access the file.

Figure 20.5. Trojan Horse and Secure Operating System

(This item is displayed on page 639 in the print version)

[View full size image]



The Trojan horse attack begins when a hostile user, named Alice, gains legitimate access to the system and installs both a Trojan horse program and a private file to be used in the attack as a "back pocket." Alice gives read/write permission to herself for this file and gives Bob write-only permission (Figure 20.5a). Alice now induces Bob to invoke the Trojan horse program, perhaps by advertising it as a useful utility. When the program detects that it is being executed by Bob, it reads the sensitive character string from Bob's file and copies it into Alice's back-pocket file (Figure 20.5b). Both the read and write operations satisfy the constraints imposed by access control lists. Alice then has only to access Bob's file at a later time to learn the value of the string.

Now consider the use of a secure operating system in this scenario (Figure 20.5c). Security levels are assigned to subjects at logon on the basis of criteria such as the terminal from which the computer is being accessed and the user involved, as identified by password/ID. In this example, there are two security levels, sensitive and public, ordered so that sensitive is higher than public. Processes owned by Bob and Bob's data file are assigned the security level sensitive. Alice's file and processes are restricted to public. If Bob invokes the Trojan horse program (Figure 20.5d), that program acquires Bob's security level. It is therefore able, under the simple security property, to observe the sensitive character string. When the program attempts to store the string in a public file (the back-pocket file), however, the is violated and the attempt is disallowed by the reference monitor. Thus, the attempt to write into the back-pocket file is denied even though the access control list permits it: The security policy takes precedence over the access control list mechanism.

PREV NEXT

20.3. Common Criteria for Information Technology Security Evaluation

The work done by the National Security Agency and other U.S. government agencies to develop requirements and evaluation criteria for trusted systems is mirrored by similar work in other countries. The Common Criteria (CC) for Information Technology and Security Evaluation is an international initiative by standards bodies in a number of countries to develop international standards for specifying security requirements and defining evaluation criteria.

Requirements

The CC defines a common set of potential security requirements for use in evaluation. The term **target of evaluation** (TOE) refers to that part of the product or system that is subject to evaluation. The requirements fall in two categories:

- **Functional requirements:** Define desired security behavior. CC documents establish a set of security functional components that provide a standard way of expressing the security functional requirements for a TOE.
- **Assurance requirements:** The basis for gaining confidence that the claimed security measures are effective and implemented correctly. CC documents establish a set of assurance components that provide a standard way of expressing the assurance requirements for a TOE.

Both functional requirements and assurance requirements are organized into classes: A **class** is a collection of requirements that share a common focus or intent. Tables 20.3 and 20.4 briefly define the requirements classes for functional and assurance requirements. Each of these classes contains a number of families. The requirements within each **family** share security objectives, but differ in emphasis or rigor. For example, the audit class contains six families dealing with various aspects of auditing (e.g., audit data generation, audit analysis, and audit event storage). Each family, in turn, contains one or more components. A **component** describes a specific set of security requirements and is the smallest selectable set of security requirements for inclusion in the structures defined in the CC.

Table 20.3. CC Security Functional Requirements

(This item is displayed on page 641 in the print version)

Class	Description
Audit	Involves recognizing, recording, storing and analyzing information related to security activities. Audit records are produced by these activities, and can be examined to determine their security relevance.
Cryptographic support	Used when the TOE implements cryptographic functions. These may be used, for example, to support communications, identification and authentication, or data separation.

Communications	Provides two families concerned with non-repudiation by the originator and by the recipient of data.
User data protection	Specifying requirements relating to the protection of user data within the TOE during import, export and storage, in addition to security attributes related to user data.
Identification and authentication	Ensure the unambiguous identification of authorized users and the correct association of security attributes with users and subjects.
Security management	Specifies the management of security attributes, data and functions.
Privacy	Provides a user with protection against discovery and misuse of his or her identity by other users.
Protection of the TOE security functions	Focused on protection of TSF (TOE security functions) data, rather than of user data. The class relates to the integrity and management of the TSF mechanisms and data.
Resource utilization	Supports the availability of required resources, such as processing capability and storage capacity. Includes requirements for fault tolerance, priority of service and resource allocation.
TOE access	Specifies functional requirements, in addition to those specified for identification and authentication, for controlling the establishment of a user's session. The requirements for TOE access govern such things as limiting the number and scope of user sessions, displaying the access history and the modification of access parameters.
Trusted path/channels	Concerned with trusted communications paths between the users and the TSF, and between TSFs.

Table 20.4. CC Security Assurance Requirements

(This item is displayed on page 642 in the print version)

Class	Description
Configuration management	Requires that the integrity of the TOE is adequately preserved. Specifically, configuration management provides confidence that the TOE and documentation used for evaluation are the ones prepared for distribution.
Delivery and operation	Concerned with the measures, procedures and standards for secure delivery, installation and operational use of the TOE, to ensure that the security protection offered by the TOE is not compromised during these events.
Development	Concerned with the refinement of the TSF from the specification defined in the ST to the implementation, and a mapping from the security requirements to the lowest level representation.

Guidance documents	Concerned with the secure operational use of the TOE, by the users and administrators.
Life cycle support	Concerned with the life-cycle of the TOE include lifecycle definition, tools and techniques, security of the development environment and the remediation of flaws found by TOE consumers.
Tests	Concerned with demonstrating that the TOE meets its functional requirements. The families address coverage and depth of developer testing, and requirements for independent testing.
Vulnerability assessment	Defines requirements directed at the identification of exploitable vulnerabilities, which could be introduced by construction, operation, misuse or incorrect configuration of the TOE. The families identified here are concerned with identifying vulnerabilities through covert channel analysis, analysis of the configuration of the TOE, examining the strength of mechanisms of the security functions, and identifying flaws introduced during development of the TOE. The second family covers the security categorization of TOE components. The third and fourth cover the analysis of changes for security impact, and the provision of evidence that procedures are being followed. This class provides building blocks for the establishment of assurance maintenance schemes.
Assurance maintenance	Provides requirements that are intended to be applied after a TOE has been certified against the CC. These requirements are aimed at assuring that the TOE will continue to meet its security target as changes are made to the TOE or its environment.

For example, the cryptographic support class of functional requirements includes two families: cryptographic key management and cryptographic operation. There are four components under the cryptographic key management family, which are used to specify: key generation algorithm and key size; key distribution method; key access method; and key destruction method. For each component, a standard may be referenced to define the requirement. Under the cryptographic operation family, there is a single component, which specifies an algorithm and key size based on a an assigned standard.

Sets of functional and assurance components may be grouped together into re-usable packages, which are known to be useful in meeting identified objectives. An example of such a package would be functional components required for Discretionary Access Controls.

Profiles and Targets

The CC also defines two kinds of documents that can be generated using the CC-defined requirements.

- **Protection profiles (PPs):** Define an implementation-independent set of security requirements and objectives for a category of products or systems that meet similar consumer needs for IT security. A PP is intended to be reusable and to define requirements that are known to be useful and effective in meeting the identified objectives. The PP concept has been developed to support the definition of functional standards, and as an aid to formulating procurement specifications. The PP reflects user security requirements
- **Security targets (STs):** Contain the IT security objectives and requirements of a specific identified TOE and defines the functional and assurance measures offered by that TOE to meet stated requirements. The ST may claim conformance to one or more PPs, and forms the basis for an evaluation. The ST is supplied by a vendor or developer.

[Page 642]

Figure 20.6 illustrates the relationship between requirements on the one hand and profiles and targets on the other. For a PP, a user can select a number of components to define the requirements for the desired product. The user may also refer to predefined packages that assemble a number of requirements commonly grouped together within a product requirements document. Similarly, a vendor or designer can select a number of components and packages to define an ST.

Figure 20.6. Organization and Construction of Common Criteria Requirements

(This item is displayed on page 643 in the print version)

[View full size image]

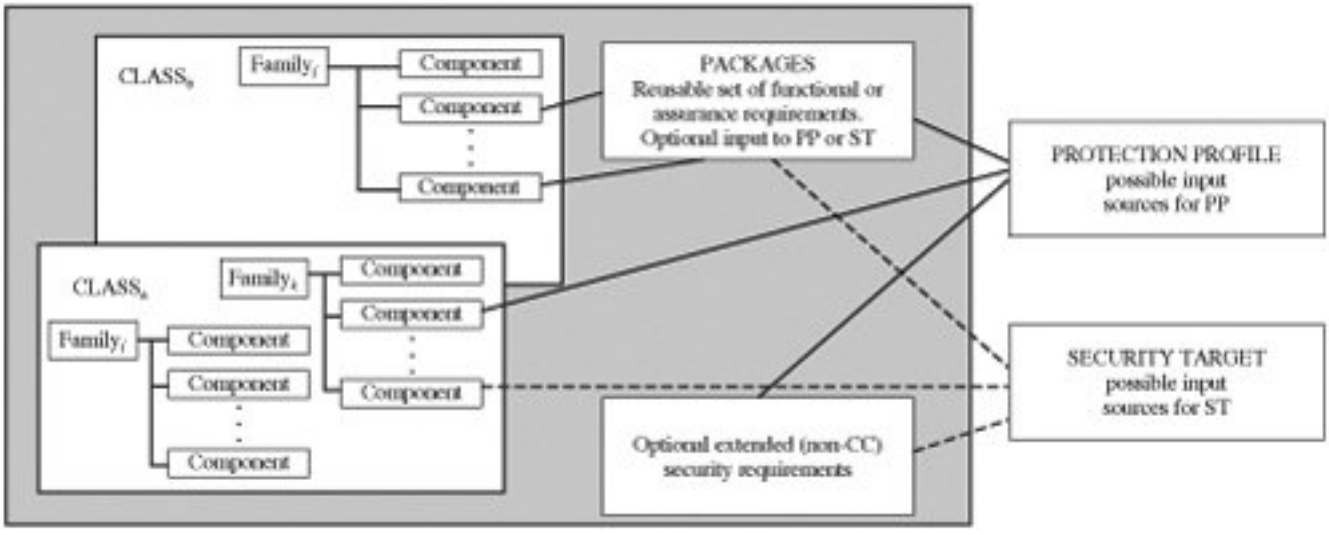
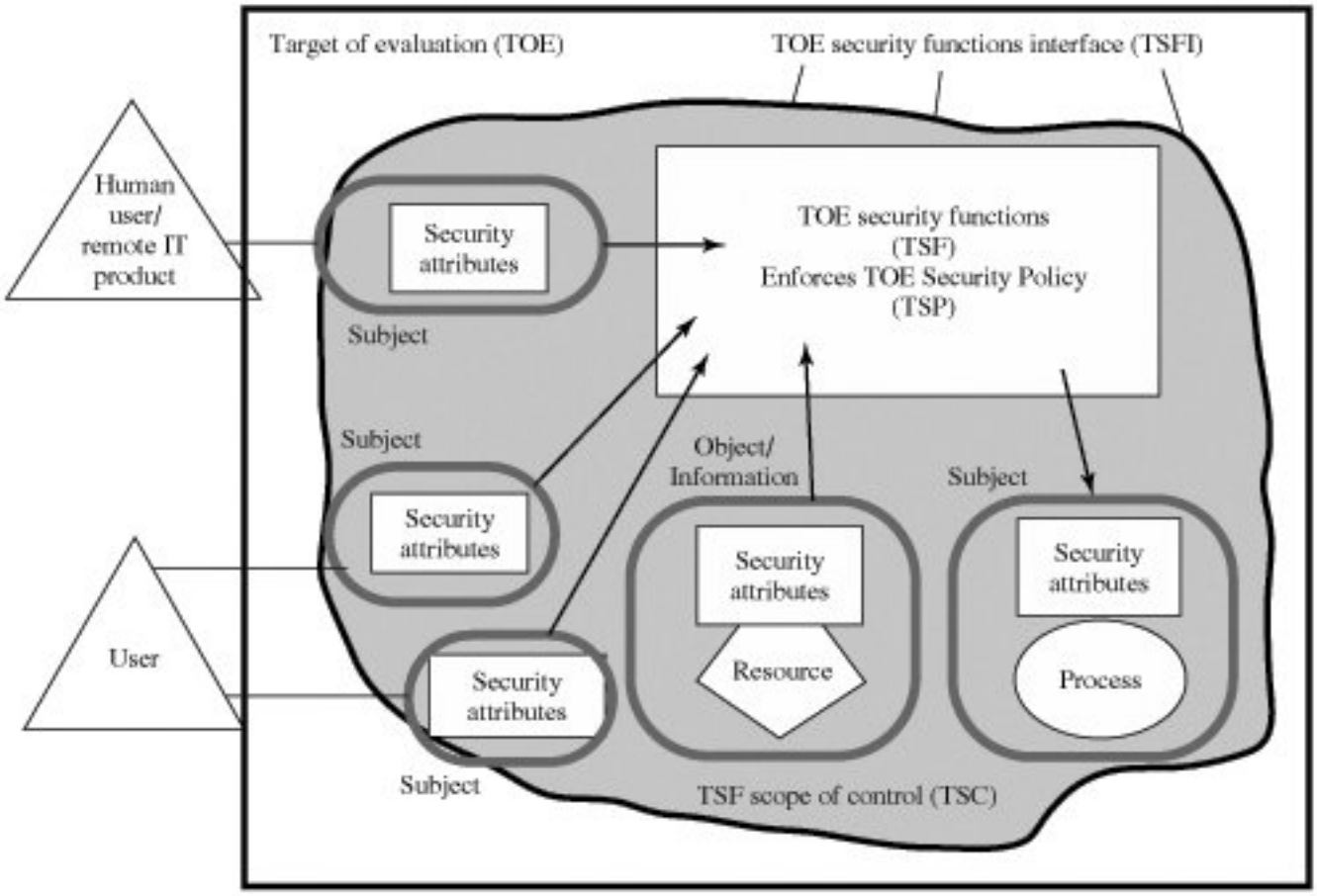


Figure 20.7 shows what is referred to in the CC documents as the security functional requirements paradigm. In essence, this illustration is based on the reference monitor concept but makes use of the terminology and design philosophy of the CC.

[Page 644]

Figure 20.7. Security Functional Requirements Paradigm

[View full size image]


Target of evaluation (TOE)
TOE security functions interface (TSFI)
Human user/ remote IT product
Security attributes
Subject
TOE security functions (TSF) Enforces TOE Security Policy (TSP)
Subject
Security attributes
Object/ Information
Security attributes
Resource
Subject
Security attributes
Process
User
Security attributes
Subject
TSF scope of control (TSC)

PREV NEXT

20.4. Recommended Reading and Web Sites

A classic treatment of firewalls is [CHAP00]. Another classic, recently updated, is [CHES03]. [LODI98], [OPPL97], and [BELL94b] are good overview articles on the subject. [WACK02] is an excellent overview of firewall technology and firewall policies. [AUDI04] and [WILS05] provide useful discussions of firewalls.

[GASS88] provides a comprehensive study of trusted computer systems. [PFLE03] and [GOLL99] also provide coverage. [FELT03] and [OPPL05] provide useful discussions of trusted computing.

AUDI04 Audin, G. "Next-Gen Firewalls: What to Expect." *Business Communications Review*, June 2004.

BELL94b Bellovin, S., and Cheswick, W. "Network Firewalls." *IEEE Communications Magazine*, September 1994.

CHAP00 Chapman, D., and Zwicky, E. *Building Internet Firewalls.* Sebastopol, CA: O'Reilly, 2000.

CHES03 Cheswick, W., and Bellovin, S. *Firewalls and Internet Security: Repelling the Wily Hacker*. Reading, MA: Addison-Wesley, 2003.

FELT03 Felten, E. "Understanding Trusted Computing: Will Its Benefits Outweigh Its Drawbacks?" *IEEE Security and Privacy*, May/June 2003.

GASS88 Gasser, M. *Building a Secure Computer System*. New York: Van Nostrand Reinhold, 1988.

GOLL99 Gollmann, D. *Computer Security.* New York: Wiley, 1999.

LODI98 Lodin, S., and Schuba, C. "Firewalls Fend Off Invasions from the Net." *IEEE Spectrum*, February 1998.

OPPL97 Oppliger, R. "Internet Security: Firewalls and Beyond." *Communications of the ACM*, May 1997.

OPPL05 Oppliger, R., and Rytz, R. "Does Trusted Computing Remedy Computer Security Problems?" *IEEE Security and Privacy*, March/April 2005.

PFLE03 Pfleeger, C. *Security in Computing*. Upper Saddle River, NJ: Prentice Hall, 1997.

WACK02 Wack, J.; Cutler, K.; and Pole, J. *Guidelines on Firewalls and Firewall Policy*. NIST Special Publication SP 800-41, January 2002.

WILS05 Wilson, J. "The Future of the Firewall."*Business Communications Review*, May 2005.

Recommended Web Sites

- **Firewall.com:** Numerous links to firewall references and software resources.
- **Trusted Computing Group:** Vendor group involved in developing and promoting trusted computer standards. Site includes white papers, specifications, and vendor links.
- **Common Criteria Portal:** Official Web site of the common criteria project.

PREV NEXT

[Page 645 (continued)]

20.5. Key Terms, Review Questions, and Problems

Key Terms

access control list (ACL)

access matrix

access right

application-level gateway

bastion host

capability ticket

circuit-level gateway

common criteria (CC)

firewall

multilevel security

object

packet-filtering router

reference monitor

stateful inspection firewall

subject

trusted system

Review Questions

20.1 List three design goals for a firewall.

20.2 List four techniques used by firewalls to control access and enforce a security policy.

20.3 What information is used by a typical packet-filtering router?

20.4 What are some weaknesses of a packet-filtering router?

20.5 What is the difference between a packet-filtering router and a stateful inspection firewall?

20.6 What is an application-level gateway?

20.7 What is a circuit-level gateway?

20.8 What are the differences among the three configurations of Figure 20.2?

20.9 In the context of access control, what is the difference between a subject and an object?

20.10 What is the difference between an access control list and a capability ticket?

20.11 What are the two rules that a reference monitor enforces?

20.12 What properties are required of a reference monitor?

20.13 What are the common criteria?

Problems

20.1 As was mentioned in Section 20.1, one approach to defeating the tiny fragment attack is to enforce a minimum length of the transport header that must be contained in the first fragment of an IP packet. If the first fragment is rejected, all subsequent fragments can be rejected. However, the nature of IP is such that fragments may arrive out of order. Thus, an intermediate fragment may pass through the filter before the initial fragment is rejected. How can this situation be handled?

20.2 In an IPv4 packet, the size of the payload in the first fragment, in octets, is equal to Total Length (4 x IHL). If this value is less than the required minimum (8 octets for TCP), then this fragment and the entire packet are rejected. Suggest an alternative method of achieving the same result using only the Fragment Offset field.

20.3 RFC 791, the IPv4 protocol specification, describes a reassembly algorithm that results in new fragments overwriting any overlapped portions of previously received fragments. Given such a reassembly implementation, an attacker could construct a series of packets in which the lowest (zero-offset) fragment would contain innocuous data (and thereby be passed by administrative packet filters), and in which some subsequent packet having a nonzero offset would overlap TCP header information (destination port, for instance) and cause it to be modified. The second packet would be passed through most filter implementations because it does not have a zero fragment offset. Suggest a method that could be used by a packet filter to counter this attack.

20.4 The necessity of the "no read up" rule for a multilevel secure system is fairly obvious. What is the importance of the "no write down" rule?

20.5 In Figure 20.5 one link of the Trojan horse copy-and-observe-later chain is broken. There are two other possible angles of attack by Drake: Drake logging on and attempting to read the string directly, and Drake assigning a security level of sensitive to the back-pocket file. Does the reference monitor prevent these attacks?

Appendix A. Standards and Standards-Setting Organizations

There are some dogs who wouldn't debase what are to them sacred forms. A very fine, very serious German Shepherd I worked with, for instance, grumbled noisily at other dogs when they didn't obey. When training him to retrieve, at one point I set the dumbbell on its end for the fun of it. He glared disapprovingly at the dumbbell and at me, then pushed it carefully back into its proper position before picking it up and returning with it, rather sullenly.

Adam's Task: Calling Animals by Name, Vicki Hearne

An important concept that recurs frequently in this book is standards. This appendix provides some background on the nature and relevance of standards and looks at the key organizations involved in developing standards for networking and communications.

A.1. The Importance of Standards

It has long been accepted in the telecommunications industry that standards are required to govern the physical, electrical, and procedural characteristics of communication equipment. In the past, this view has not been embraced by the computer industry. Whereas communication equipment vendors recognize that their equipment will generally interface to and communicate with other vendors' equipment, computer vendors have traditionally attempted to monopolize their customers. The proliferation of computers and distributed processing has made that an untenable position. Computers from different vendors must communicate with each other and, with the ongoing evolution of protocol standards, customers will no longer accept special-purpose protocol conversion software development. The result is that standards now permeate all the areas of technology discussed in this book.

There are a number of advantages and disadvantages to the standards-making process. The principal advantages of standards are as follows:

- A standard assures that there will be a large market for a particular piece of equipment or software. This encourages mass production and, in some cases, the use of large-scale-integration (LSI) or very-large-scale-integration (VLSI) techniques, resulting in lower costs.
- A standard allows products from multiple vendors to communicate, giving the purchaser more flexibility in equipment selection and use.

The principal disadvantages of standards are as follows:

- A standard tends to freeze the technology. By the time a standard is developed, subjected to review and compromise, and promulgated, more efficient techniques are possible.
- There are multiple standards for the same thing. This is not a disadvantage of standards per se, but of the current way things are done. Fortunately, in recent years the various standards-making organizations have begun to cooperate more closely. Nevertheless, there are still areas where multiple conflicting standards exist.

A.2. Internet Standards and the Internet Society

Many of the protocols that make up the TCP/IP protocol suite have been standardized or are in the process of standardization. By universal agreement, an organization known as the Internet Society is responsible for the development and publication of these standards. The Internet Society is a professional membership organization that oversees a number of boards and task forces involved in Internet development and standardization.

This section provides a brief description of the way in which standards for the TCP/IP protocol suite are developed.

The Internet Organizations and RFC Publication

The Internet Society is the coordinating committee for Internet design, engineering, and management. Areas covered include the operation of the Internet itself and the standardization of protocols used by end systems on the Internet for interoperability. Three organizations under the Internet Society are responsible for the actual work of standards development and publication:

- **Internet Architecture Board (IAB):** Responsible for defining the overall architecture of the Internet, providing guidance and broad direction to the IETF
- **Internet Engineering Task Force (IETF):** The protocol engineering and development arm of the Internet
- **Internet Engineering Steering Group (IESG):** Responsible for technical management of IETF activities and the Internet standards process

Working groups chartered by the IETF carry out the actual development of new standards and protocols for the Internet. Membership in a working group is voluntary; any interested party may participate. During the development of a specification, a working group will make a draft version of the document available as an Internet Draft, which is placed in the IETF's "Internet Drafts" online directory. The document may remain as an Internet Draft for up to six months, and interested parties may review and comment on the draft. During that time, the IESG may approve publication of the draft as an RFC (Request for Comment). If the draft has not progressed to the status of an RFC during the six-month period, it is withdrawn from the directory. The working group may subsequently publish a revised version of the draft.

The IETF is responsible for publishing the RFCs, with approval of the IESG. The RFCs are the working notes of the Internet research and development community. A document in this series may be on essentially any topic related to computer communications and may be anything from a meeting report to the specification of a standard.

The work of the IETF is divided into eight areas, each with an area director and each composed of numerous working groups. Table A.1 shows the IETF areas and their focus.

Table A.1. IETF Areas

(This item is displayed on page 650 in the print version)

IETF Area	Theme	Example Working Groups
General	IETF processes and procedures	Policy Framework Process for Organization of Internet Standards
Applications	Internet applications	Web-related protocols (HTTP) EDI-Internet integration LDAP
Internet	Internet infrastructure	IPv6 PPP extensions
Operations and management	Standards and definitions for network operations	SNMPv3 Remote Network Monitoring
Routing	Protocols and management for routing information	Multicast routing OSPF QoS routing
Security	Security protocols and technologies	Kerberos IPSec X.509 S/MIME TLS
Transport	Transport layer protocols	Differentiated services IP telephony NFS RSVP

User services	Methods to improve the quality of information available to users of the Internet	Responsible Use of the Internet User services FYI documents

The Standardization Process

The decision of which RFCs become Internet standards is made by the IESG, on the recommendation of the IETF. To become a standard, a specification must meet the following criteria:

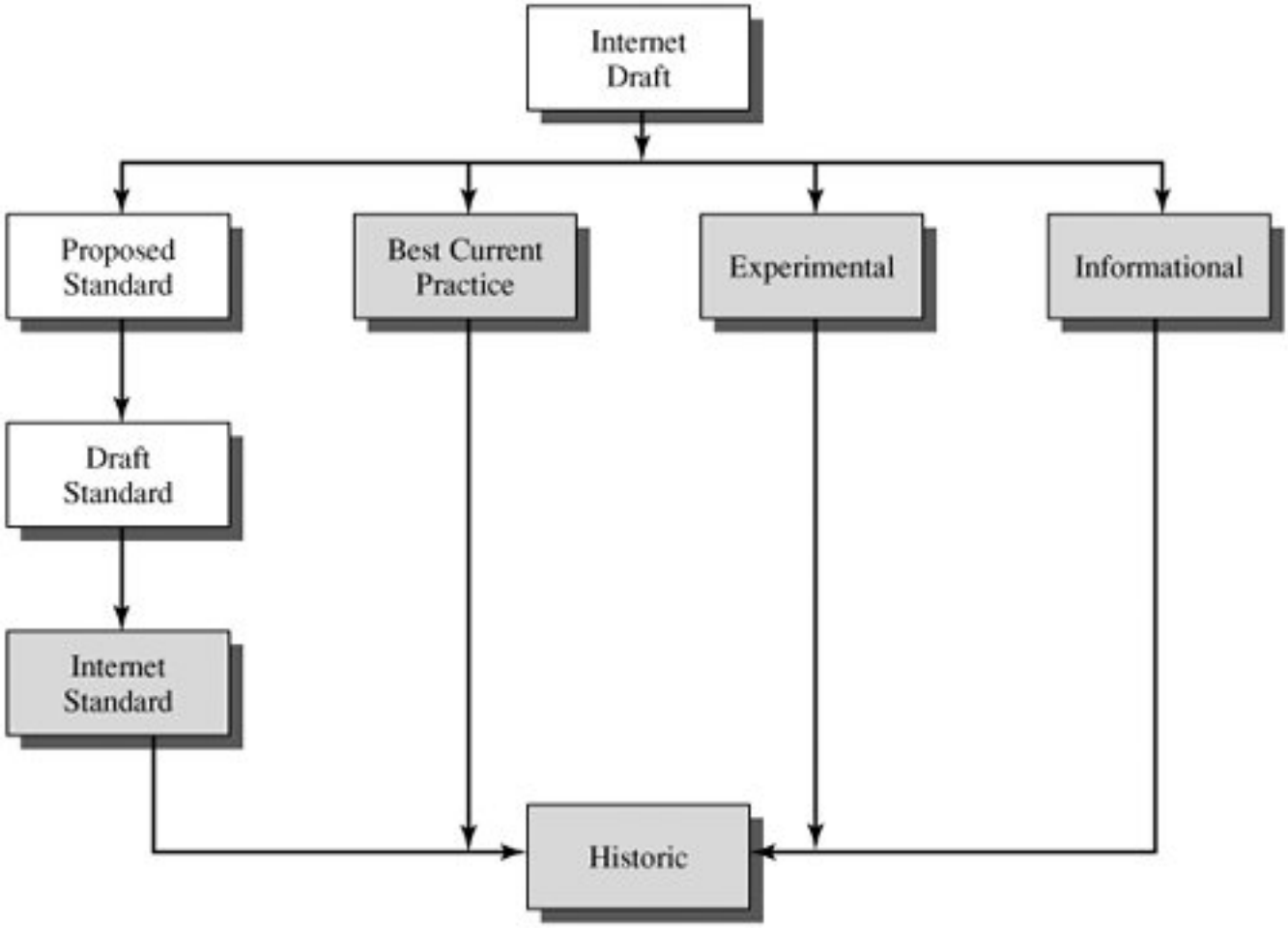
- Be stable and well understood
- Be technically competent
- Have multiple, independent, and interoperable implementations with substantial operational experience
- Enjoy significant public support
- Be recognizably useful in some or all parts of the Internet

The key difference between these criteria and those used for international standards from ITU is the emphasis here on operational experience.

The left-hand side of Figure A.1 shows the series of steps, called the *standards track*, that a specification goes through to become a standard; this process is defined in RFC 2026. The steps involve increasing amounts of scrutiny and testing. At each step, the IETF must make a recommendation for advancement of the protocol, and the IESG must ratify it. The process begins when the IESG approves the publication of an Internet Draft document as an RFC with the status of Proposed Standard.

Figure A.1. Internet RFC Publication Process

(This item is displayed on page 651 in the print version)



The white boxes in the diagram represent temporary states, which should be occupied for the minimum practical time. However, a document must remain a Proposed Standard for at least six months and a Draft Standard for at least four months to allow time for review and comment. The gray boxes represent long-term states that may be occupied for years.

For a specification to be advanced to Draft Standard status, there must be at least two independent and interoperable implementations from which adequate operational experience has been obtained.

After significant implementation and operational experience has been obtained, a specification may be elevated to Internet Standard. At this point, the Specification is assigned an STD number as well as an RFC number.

Finally, when a protocol becomes obsolete, it is assigned to the Historic state.

Internet Standards Categories

All Internet standards fall into one of two categories:

- **Technical specification (TS):** A TS defines a protocol, service, procedure, convention, or format. The bulk of the Internet standards are TSs.
- **Applicability statement (AS):** An AS specifies how, and under what circumstances, one or more TSs may be applied to support a particular Internet capability. An AS identifies one or more TSs that are relevant to the capability, and may specify values or ranges for particular

parameters associated with a TS or functional subsets of a TS that are relevant for the capability.

Other RFC Types

There are numerous RFCs that are not destined to become Internet standards. Some RFCs standardize the results of community deliberations about statements of principle or conclusions about what is the best way to perform some operations or IETF process function. Such RFCs are designated as Best Current Practice (BCP). Approval of BCPs follows essentially the same process for approval of Proposed Standards. Unlike standards-track documents, there is not a three-stage process for BCPs; a BCP goes from Internet draft status to approved BCP in one step.

A protocol or other specification that is not considered ready for standardization may be published as an Experimental RFC. After further work, the specification may be resubmitted. If the specification is generally stable, has resolved known design choices, is believed to be well understood, has received significant community review, and appears to enjoy enough community interest to be considered valuable, then the RFC will be designated a Proposed Standard.

Finally, an Informational Specification is published for the general information of the Internet community.

A.3. National Institute of Standards and Technology

The National Institute of Standards and Technology (NIST), part of the U.S. Commerce Department, issues standards and guidelines for use by U.S. government departments and agencies. These standards and guidelines are issued in the form of Federal Information Processing Standards (FIPS). NIST develops FIPS when there are compelling federal government requirements such as for security and interoperability and there are no acceptable industry standards or solutions.

- NIST announces the proposed FIPS in the *Federal Register* for public review and comment. At the same time that the proposed FIPS is announced in the *Federal Register*, it is also announced on NIST's Web site. The text and associated specifications, if applicable, of the proposed FIPS are posted on the NIST Web site.
- A 90-day period is provided for review and for submission of comments on the proposed FIPS to NIST. The date by which comments must be submitted to NIST is specified in the *Federal Register* and in the other announcements.
- Comments received in response to the *Federal Register* notice and to the other notices are reviewed by NIST to determine if modifications to the proposed FIPS are needed.
- A detailed justification document is prepared, analyzing the comments received and explaining whether modifications were made, or explaining why recommended changes were not made.
- NIST submits the recommended FIPS, the detailed justification document, and recommendations as to whether the standard should be compulsory and binding for Federal government use, to the Secretary of Commerce for approval.
- A notice announcing approval of the FIPS by the Secretary of Commerce is published in the *Federal Register*, and on NIST's Web site.

Although NIST standards are developed for U.S. government use, many of them are widely used in industry. AES and DES are prime examples.

Appendix B. Projects for Teaching Cryptography and Network Security

---

Analysis and observation, theory and experience must never disdain or exclude each other; on the contrary, they support each other.

On War, Carl Von Clausewitz

Many instructors believe that research or implementation projects are crucial to the clear understanding of cryptography and network security. Without projects, it may be difficult for students to grasp some of the basic concepts and interactions among components. Projects reinforce the concepts introduced in the book, give the student a greater appreciation of how a cryptographic algorithm or protocol works, and can motivate students and give them confidence that they are capable of not only understanding but implementing the details of a security capability.

In this text, I have tried to present the concepts of cryptography and network security as clearly as possible and have provided numerous homework problems to reinforce those concepts. However, many instructors will wish to supplement this material with projects. This appendix provides some guidance in that regard and describes support material available in the instructor's supplement. The support material covers five types of projects:

- Research projects
- Programming projects
- Laboratory exercises
- Writing assignments
- Reading/report assignments

B.1. Research Projects

An effective way of reinforcing basic concepts from the course and for teaching students research skills is to assign a research project. Such a project could involve a literature search as well as an Internet search of vendor products, research lab activities, and standardization efforts. Projects could be assigned to teams or, for smaller projects, to individuals. In any case, it is best to require some sort of project proposal early in the term, giving the instructor time to evaluate the proposal for appropriate topic and appropriate level of effort. Student handouts for research projects should include

- A format for the proposal
- A format for the final report
- A schedule with intermediate and final deadlines
- A list of possible project topics

The students can select one of the listed topics or devise their own comparable project. The instructor's supplement includes a suggested format for the proposal and final report as well as a list of fifteen possible research topics.

B.2. Programming Projects

The programming project is a useful pedagogical tool. There are several attractive features of stand-alone programming projects that are not part of an existing security facility:

1. The instructor can choose from a wide variety of cryptography and network security concepts to assign projects.

2. The projects can be programmed by the students on any available computer and in any appropriate language; they are platform and language independent.

3. The instructor need not download, install, and configure any particular infrastructure for stand-alone projects.

There is also flexibility in the size of projects. Larger projects give students more a sense of achievement, but students with less ability or fewer organizational skills can be left behind. Larger projects usually elicit more overall effort from the best students. Smaller projects can have a higher concepts-to-code ratio, and because more of them can be assigned, the opportunity exists to address a variety of different areas.

Again, as with research projects, the students should first submit a proposal. The student handout should include the same elements listed in Section A.1. The instructor's manual includes a set of twelve possible programming projects.

The following individuals have supplied the research and programming projects suggested in the instructor's manual: Henning Schulzrinne of Columbia University; Cetin Kaya Koc of Oregon State University; and David M. Balenson of Trusted Information Systems and George Washington University.

[Page 655 (continued)]

B.3. Laboratory Exercises

Professor Sanjay Rao and Ruben Torres of Purdue University have prepared a set of laboratory exercises that are part of the instructor's supplement. These are implementation projects designed to be programmed on Linux but could be adapted for any Unix environment. These laboratory exercises provide realistic experience in implementing security functions and applications.

[Page 655 (continued)]

B.4. Writing Assignments

Writing assignments can have a powerful multiplier effect in the learning process in a technical discipline such as cryptography and network security. Adherents of the Writing Across the Curriculum (WAC) movement (**http://wac.colostate.edu**) report substantial benefits of writing assignments in facilitating learning. Writing assignments lead to more detailed and complete thinking about a particular topic. In addition, writing assignments help to overcome the tendency of students to pursue a subject with a minimum of personal engagement, just learning facts and problem-solving techniques without obtaining a deep understanding of the subject matter.

[Page 656]

The instructor's supplement contains a number of suggested writing assignments, organized by chapter. Instructors may ultimately find that this is the most important part of their approach to teaching the material. I would greatly appreciate any feedback on this area and any suggestions for additional writing assignments.

B.5. Reading/Report Assignments

Another excellent way to reinforce concepts from the course and to give students research experience is to assign papers from the literature to be read and analyzed. The instructor's supplement includes a suggested list of papers, one or two per chapter, to be assigned. All of the papers are readily available either via the Internet or in any good college technical library. The instructor's supplement also includes a suggested assignment wording.

Glossary

> ***In studying the Imperium, Arrakis, and the whole culture which produced Maud'Dib, many unfamiliar terms occur. To increase understanding is a laudable goal, hence the definitions and explanations given below.***
>
> *Dune*, Frank Herbert

Some of the terms in this glossary are from the *Internet Security Glossary* [RFC 2828]. These are indicated in the glossary by an asterisk.

asymmetric encryption

A form of cryptosystem in which encryption and decryption are performed using two different keys, one of which is referred to as the public key and one of which is referred to as the private key. Also known as public-key encryption.

authentication*

The process of verifying an identity claimed by or for a system entity.

authenticator

Additional information appended to a message to enable the receiver to verify that the message should be accepted as authentic. The authenticator may be functionally independent of the content of the message itself (e.g., a nonce or a source identifier) or it may be a function of the message contents (e.g., a hash value or a cryptographic checksum).

avalanche effect

A characteristic of an encryption algorithm in which a small change in the plaintext or key gives rise to a large change in the ciphertext. For a hash code, the avalanche effect is a characteristic in which a small change in the message gives rise to a large change in the message digest.

bacteria

Program that consumes system resources by replicating itself.

birthday attack

This cryptanalytic attack attempts to find two values in the domain of a function that map to the same value in its range

block chaining

A procedure used during symmetric block encryption that makes an output block dependent not only on the current plaintext input block and key, but also on earlier input and/or output. The effect of block chaining is that two instances of the same plaintext input block will produce different ciphertext blocks, making cryptanalysis more difficult.

block cipher

A symmetric encryption algorithm in which a block of plaintext bits (typically 64 or 128) is transformed as a whole into a ciphertext block of the same length.

byte

A sequence of eight bits. Also referred to as an *octet.*

cipher

An algorithm for encryption and decryption. A cipher replaces a piece of information (an element in plaintext) with another object, with the intent to conceal meaning. Typically, the replacement rule is governed by a secret key.

ciphertext

The output of an encryption algorithm; the encrypted form of a message or data.

code

An unvarying rule for replacing a piece of information (e.g., letter, word, phrase) with another object, not necessarily of the same sort. Generally, there is no intent to conceal meaning. Examples include the ASCII character code (each character is represented by 7 bits) and frequency-shift keying (each binary value is represented by a particular frequency).

---

computationally secure

Secure because the time and/or cost of defeating the security are too high to be feasible.

confusion

A cryptographic technique that seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible. This is achieved by the use of a complex scrambling algorithm that depends on the key and the input.

conventional encryption

Symmetric encryption.

covert channel

A communications channel that enables the transfer of information in a way unintended by the

designers of the communications facility.

cryptanalysis

The branch of cryptology dealing with the breaking of a cipher to recover information, or forging encrypted information that will be accepted as authentic.

cryptographic checksum

An authenticator that is a cryptographic function of both the data to be authenticated and a secret key. Also referred to as a message authentication code (MAC).

cryptography

The branch of cryptology dealing with the design of algorithms for encryption and decryption, intended to ensure the secrecy and/or authenticity of messages.

cryptology

The study of secure communications, which encompasses both cryptography and cryptanalysis.

decryption

The translation of encrypted text or data (called ciphertext) into original text or data (called plaintext). Also called deciphering.

differential cryptanalysis

A technique in which chosen plaintexts with particular XOR difference patterns are encrypted. The difference patterns of the resulting ciphertext provide information that can be used to determine the encryption key.

diffusion

A cryptographic technique that seeks to obscure the statistical structure of the plaintext by spreading out the influence of each individual plaintext digit over many ciphertext digits.

digital signature

An authentication mechanism that enables the creator of a message to attach a code that acts as a signature. The signature is formed by taking the hash of the message and encrypting the message with the creator's private key. The signature guarantees the source and integrity of the message.

digram

A two-letter sequence. In English and other languages, the relative frequency of various digrams in plaintext can be used in the cryptanalysis of some ciphers. Also called *digraph*.

discretionary access control*

An access control service that enforces a security policy based on the identity of system entities and their authorizations to access system resources. (See: access control list, identity-based security policy, mandatory access control.) This service is termed "discretionary" because an entity might have access rights that permit the entity, by its own volition, to enable another entity to access some resource.

divisor

One integer is said to be a devisor of another integer if there is no remainder on division.

encryption

The conversion of plaintext or data into unintelligible form by means of a reversible translation, based on a translation table or algorithm. Also called enciphering.

---

firewall

A dedicated computer that interfaces with computers outside a network and has special security precautions built into it in order to protect sensitive files on computers within the network. It is used to service outside network, especially Internet, connections and dial-in lines.

greatest common divisor

The greatest common divisor of two integers, *a* and *b*, is the largest positive integer that divides both *a* and *b*. One integer is said to divide another integer if there is no remainder on division.

hash function

A function that maps a variable-length data block or message into a fixed-length value called a hash code. The function is designed in such a way that, when protected, it provides an authenticator to the data or message. Also referred to as a message digest.

honeypot

A decoy system designed to lure a potential attacker away from critical systems. A form of intrusion detection.

initialization vector

A random block of data that is used to begin the encryption of multiple blocks of plaintext, when a block-chaining encryption technique is used. The IV serves to foil known-plaintext attacks.

intruder

An individual who gains, or attempts to gain, unauthorized access to a computer system or to gain unauthorized privileges on that system.

intrusion detection system

A set of automated tools designed to detect unauthorized access to a host system.

Kerberos

The name given to Project Athena's code authentication service.

key distribution center

A system that is authorized to transmit temporary session keys to principals. Each session key is transmitted in encrypted form, using a master key that the key distribution center shares with the target principal.

logic bomb

Logic embedded in a computer program that checks for a certain set of conditions to be present on the system. When these conditions are met, it executes some function resulting in unauthorized actions.

mandatory access control

A means of restricting access to objects based on fixed security attributes assigned to users and to files and other objects. The controls are mandatory in the sense that they cannot be modified by users or their programs.

man-in-the-middle attack

A form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data in order to masquerade as one or more of the entities involved in a communication.

master key

A long-lasting key that is used between a key distribution center and a principal for the purpose of encoding the transmission of session keys. Typically, the master keys are distributed by noncryptographic means. Also referred to as a key-encrypting key.

meet-in-the-middle attack

This is a cryptanaltytic attack that attempts to find a value in each of the range and domain of the composition of two functions such that the forward mapping of one through the first function is the same as the inverse image of the other through the second functionquite literally meeting in the middle of the composed function.

message authentication

A process used to verify the integrity of a message.

message authentication code (MAC)

Cryptographic checksum.

---

message digest

Hash function.

modular arithmetic

A kind of integer arithmetic that reduces all numbers to one of a fixed set [0 ... *n* 1] for some number *n*. Any integer outside this range is reduced to one in this range by taking the remainder after division by *n*.

mode of operation

A technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream.

multilevel security

A capability that enforces access control across multiple levels of classification of data.

multiple encryption

Repeated use of an encryption function, with different keys, to produce a more complex mapping from plaintext to ciphertext.

nibble

A sequence of four bits.

nonce

An identifier or number that is used only once.

one-way function

A function that is easily computed, but the calculation of its inverse is infeasible.

password*

A secret data value, usually a character string, that is used as authentication information. A password is usually matched with a user identifier that is explicitly presented in the authentication process, but in some cases the identity may be implicit.

plaintext

The input to an encryption function or the output of a decryption function.

primitive root

If r and n are relatively prime integers with $n > 0$. and if $\phi(n)$ is the least positive exponent m such that $r^m \equiv 1 \bmod n$, then r is called a primitive root modulo n.

private key

One of the two keys used in an asymmetric encryption system. For secure communication, the private key should only be known to its creator.

pseudorandom number generator

A function that deterministically produces a sequence of numbers that are apparently statistically random.

public key

One of the two keys used in an asymmetric encryption system. The public key is made public, to be used in conjunction with a corresponding private key.

public-key certificate

Consists of a public key plus a User ID of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a certificate authority (CA) that is trusted by the user community, such as a government agency or a financial institution.

public-key encryption

Asymmetric encryption.

public-key infrastructure (PKI)

The set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.

relatively prime

Two numbers are relatively prime if they have no prime factors in common; that is, their only common divisor is 1.

replay attacks

An attack in which a service already authorized and completed is forged by another "duplicate request" in an attempt to repeat authorized commands.

residue

When the integer *a* is divided by the integer *n*, the remainder *r* is referred to as the residue. Equivalently, $r = a \bmod n$.

residue class

All the integers that have the same remainder when divided by n form a residue class (mod n). Thus, for a given remainder r, the residue class (mod n) to which it belongs consists of the integers r, $r \pm n$, $r \pm 2n$, ...

RSA algorithm

A public-key encryption algorithm based on exponentiation in modular arithmetic. It is the only algorithm generally accepted as practical and secure for public-key encryption.

secret key

The key used in a symmetric encryption system. Both participants must share the same key, and this key must remain secret to protect the communication.

security attack*

An assault on system security that derives from an intelligent threat; that is, an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system.

security mechanism

A process (or a device incorporating such a process) that is designed to detect, prevent, or recover from a security attack.

security service

A processing or communication service that enhances the security of the data processing systems and the information transfers of an organization. The services are intended to counter security attacks, and they make use of one or more security mechanisms to provide the service.

security threat*

A potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm. That is, a threat is a possible danger that might exploit a vulnerability.

session key

A temporary encryption key used between two principals.

steganography

Methods of hiding the existence of a message or other data. This is different than cryptography, which hides the meaning of a message but does not hide the message itself.

stream cipher

A symmetric encryption algorithm in which ciphertext output is produced bit-by-bit or byte-by-byte from a stream of plaintext input.

symmetric encryption

A form of cryptosystem in which encryption and decryption are performed using the same key. Also known as conventional encryption.

trapdoor

Secret undocumented entry point into a program, used to grant access without normal methods of access authentication.

trapdoor one-way function

A function that is easily computed, and the calculation of its inverse is infeasible unless certain privileged information is known.

Trojan horse*

A computer program that appears to have a useful function, but also has a hidden and potentially malicious function that evades security mechanisms, sometimes by exploiting legitimate authorizations of a system entity that invokes the program.

trusted system

A computer and operating system that can be verified to implement a given security policy.

---

unconditionally secure

Secure even against an opponent with unlimited time and unlimited computing resources.

virtual private network

Consists of a set of computers that interconnect by means of a relatively unsecure network and that make use of encryption and special protocols to provide security.

virus

Code embedded within a program that causes a copy of itself to be inserted in one or more other programs. In addition to propagation, the virus usually performs some unwanted function.

worm

Program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. In addition to propagation, the worm usually performs some unwanted function.

zombie

A program that secretly takes over another Internet-attached computer and then uses that computer to launch attacks that are difficult to trace to the zombie's creator.

PREV NEXT

References

In matters of this kind everyone feels he is justified in writing and publishing the first thing that comes into his head when he picks up a pen, and thinks his own idea as axiomatic as the fact that two and two make four. If critics would go to the trouble of thinking about the subject for years on end and testing each conclusion against the actual history of war, as I have done, they would undoubtedly be more careful of what they wrote.

On War, Carl von Clausewitz

Abbreviations

ACM	Association for Computing Machinery
IEEE	Institute of Electrical and Electronics Engineers
NIST	National Institute of Standards and Technology

ADAM90 Adams, C., and Tavares, S. "Generating and Counting Binary Bent Sequences." *IEEE Transactions on Information Theory*, 1990.

ADAM94 Adams, C. "Simple and Effective Key Scheduling for Symmetric Ciphers." *Proceedings, Workshop in Selected Areas of Cryptography, SAC' 94*. 1994.

ADLE83 Adleman, L. "The Function Field Sieve." *Lecture Notes in Computer Science 877*, *Springer-Verlag*, 1983.

AGRA02 Agrawal, M.; Keyal, N.; and Saxena, N. "PRIMES is in P." *IIT Kanpur, Preprint*, August 2002. http://www.cse.iitk.ac.in/news/primality.pdf

AKL83 Akl, S. "Digital Signatures: A Tutorial Survey." *Computer*, February 1983.

ALVA90 Alvare, A. "How Crackers Crack Passwords or What Passwords to Avoid." *Proceedings, UNIX Security Workshop II*, August 1990.

ANDE80 Anderson, J. *Computer Security Threat Monitoring and Surveillance*. Fort Washington, PA: James P. Anderson Co., April 1980.

AUDI04 Audin, G. "Next-Gen Firewalls: What to Expect." *Business Communications Review*, June 2004.

AXEL00 Axelsson, S. "The Base-Rate Fallacy and the Difficulty of Intrusion Detection." *ACM Transactions and Information and System Security*, August 2000.

BACE00 Bace, R. *Intrusion Detection*. Indianapolis, IN: Macmillan Technical Publishing, 2000.

BACE01 Bace, R., and Mell, P. *Intrusion Detection Systems*. NIST Special Publication SP 800-31, November 2000.

BARK91 Barker, W. *Introduction to the Analysis of the Data Encryption Standard (DES)*. Laguna Hills, CA: Aegean Park Press, 1991.

BARR03 Barreto, P., and Rijmen, V. "The Whirlpool Hashing Function." *Submitted to NESSIE*, September 2000, revised May 2003.

BAUE88 Bauer, D., and Koblentz, M. "NIDXAn Expert System for Real-Time Network Intrusion Detection." *Proceedings, Computer Networking Symposium*, April 1988.

BELL90 Bellovin, S., and Merritt, M. "Limitations of the Kerberos Authentication System." *Computer Communications Review*, October 1990.

BELL92 Bellovin, S. "There Be Dragons." *Proceedings, UNIX Security Symposium III*, September 1992.

BELL93 Bellovin, S. "Packets Found on an Internet." *Computer Communications Review*, July 1993.

BELL94a Bellare, M, and Rogaway, P. "Optimal Asymmetric EncryptionHow to Encrypt with RSA." *Proceedings, Eurocrypt '94*, 1994.

BELL94b Bellovin, S., and Cheswick, W. "Network Firewalls." *IEEE Communications Magazine*, September 1994.

BELL96a Bellare, M.; Canetti, R.; and Krawczyk, H. "Keying Hash Functions for Message Authentication." *Proceedings, CRYPTO '96*, August 1996; published by Springer-Verlag. An expanded version is available at http://www-cse.ucsd.edu/users/mihir

BELL96b Bellare, M.; Canetti, R.; and Krawczyk, H. "The HMAC Construction." *CryptoBytes*, Spring 1996.

BELL97 Bellare, M., and Rogaway, P. "Collision-Resistant Hashing: Towards Making UOWHF's Practical." *Proceedings, CRYPTO '97*, 1997; published by Springer-Verlag.

BELL00 Bellare, M.; Kilian, J.; and Rogaway, P. "The Security of the Cipher Block Chaining Message Authentication Code." *Journal of Computer and System Sciences*, December 2000.

BERL84 Berlekamp, E. *Algebraic Coding Theory*. Laguna Hills, CA: Aegean Park Press, 1984.

BETH91 Beth, T.; Frisch, M.; and Simmons, G. eds. *Public-Key Cryptography: State of the Art and Future Directions*. New York: Springer-Verlag, 1991.

BIHA93 Biham, E., and Shamir, A. *Differential Cryptanalysis of the Data Encryption Standard*. New York: Springer-Verlag, 1993.

BIHA00 Biham, E., and Shamir, A. "Power Analysis of the Key Scheduling of the AES Candidates" Proceedings, Second AES Candidate Conference, 24 October 2000. http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm

BISH03 Bishop, M. *Computer Security: Art and Science*. Boston: Addison-Wesley, 2003.

BISH05 Bishop, M. *Introduction to Computer Security*. Boston: Addison-Wesley, 2005.

BLAC00 Black, J., and Rogaway, P.; and Shrimpton, T. "CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions." *Advances in CryptologyCRYPTO '00*, 2000.

BLAC02 Black, J., and Rogaway, P. "Black-Box Analysis of the Block-Cipher-Based Hash Function Constructions from PGV." *Advances in CryptologyCRYPTO '02*, 2002.

BLAK99 Blake, I.; Seroussi, G.; and Smart, N. *Elliptic Curves in Cryptography*. Cambridge: Cambridge University Press, 1999.

BLOO70 Bloom, B. "Space/time Trade-Offs in Hash Coding with Allowable Errors." *Communications of the ACM*, July 1970.

BLUM86 Blum, L.; Blum, M.; and Shub, M. "A Simple Unpredictable Pseudo-Random Number Generator." *SIAM Journal on Computing*, No. 2, 1986.

BONE99 Boneh, D. "Twenty Years of Attacks on the RSA Cryptosystem." *Notices of the American Mathematical Society*, February 1999.

BONE02 Boneh, D., and Shacham, H. "Fast Variants of RSA." *CryptoBytes*, Winter/Spring 2002. http://www.rsasecurity.com/rsalabs

BORN03 Bornemann, F. "PRIMES is in P: A Breakthrough for Everyman." *Notices of the American Mathematical Society*, May 2003.

BRIG79 Bright, H., and Enison, R. "Quasi-Random Number Sequences from Long-Period TLP Generator with Remarks on Application to Cryptography." *Computing Surveys*, December 1979.

BRYA88 Bryant, W. *Designing an Authentication System: A Dialogue in Four Scenes*. Project Athena document, February 1988. Available at http://web.mit.edu/kerberos/www/dialogue.html

BURN97 Burn, R. *A Pathway to Number Theory*. Cambridge, England: Cambridge University Press, 1997.

CAMP92 Campbell, K., and Wiener, M. "Proof that DES Is Not a Group." *Proceedings, Crypto '92*, 1992; published by Springer-Verlag.

CASS01 Cass, S. "Anatomy of Malice." *IEEE Spectrum*, November 2001.

CERT01 CERT Coordination Center. "Denial of Service Attacks." June 2001. http://www.cert.org/tech_tips/denial_of_service.html

CHAN02 Chang, R. "Defending Against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial." *IEEE Communications Magazine*, October 2002.

CHAP00 Chapman, D., and Zwicky, E. *Building Internet Firewalls*. Sebastopol, CA: O'Reilly, 2000.

CHEN98 Cheng, P., et al. "A Security Architecture for the Internet Protocol." *IBM Systems Journal*, Number 1, 1998.

CHES97 Chess, D. "The Future of Viruses on the Internet." *Proceedings, Virus Bulletin International Conference*, October 1997.

CHES03 Cheswick, W., and Bellovin, S. *Firewalls and Internet Security: Repelling the Wily Hacker*.

Reading, MA: Addison-Wesley, 2003.

COCK73 Cocks, C. *A Note on Non-Secret Encryption.* CESG Report, November 1973.

COHE94 Cohen, F. *A Short Course on Computer Viruses*. New York: Wiley, 1994.

COPP94 Coppersmith, D. "The Data Encryption Standard (DES) and Its Strength Against Attacks." *IBM Journal of Research and Development*, May 1994.

CORM01 Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2001.

CRAN01 Crandall, R., and Pomerance, C. *Prime Numbers: A Computational Perspective*. New York: Springer-Verlag, 2001.

DAEM99 Daemen, J., and Rijmen, V. *AES Proposal: Rijndael, Version 2.* Submission to NIST, March 1999. http://csrc.nist.gov/encryption/aes

DAEM01 Daemen, J., and Rijmen, V. "Rijndael: The Advanced Encryption Standard." *Dr. Dobb's Journal*, March 2001.

DAEM02 Daemen, J., and Rijmen, V. *The Design of Rijndael: The Wide Trail Strategy Explained*. New York, Springer-Verlag, 2002.

DAMG89 Damgard, I. "A Design Principle for Hash Functions." *Proceedings, CRYPTO '89*, 1989; published by Springer-Verlag.

DAVI89 Davies, D., and Price, W. *Security for Computer Networks*. New York: Wiley, 1989.

DAVI93 Davies, C., and Ganesan, R. "BApasswd: A New Proactive Password Checker." *Proceedings, 16th National Computer Security Conference*, September 1993.

DAWS96 Dawson, E., and Nielsen, L. "Automated Cryptoanalysis of XOR Plaintext Strings." *Cryptologia*, April 1996.

DENN81 Denning, D. "Timestamps in Key Distribution Protocols." *Communications of the ACM*, August 1981.

DENN82 Denning, D. *Cryptography and Data Security*. Reading, MA: Addison-Wesley, 1982.

DENN83 Denning, D. "Protecting Public Keys and Signature Keys." *Computer*, February 1983.

DENN87 Denning, D. "An Intrusion-Detection Model." *IEEE Transactions on Software Engineering*, February 1987.

DESK92 Deskins, W. *Abstract Algebra*. New York: Dover, 1992.

DIFF76a Diffie, W., and Hellman, M. "New Directions in Cryptography." *Proceedings of the AFIPS National Computer Conference*, June 1976.

DIFF76b Diffie, W., and Hellman, M. "Multiuser Cryptographic Techniques." *IEEE Transactions on*

Information Theory, November 1976.

DIFF77 Diffie, W., and Hellman, M. "Exhaustive Cryptanalysis of the NBS Data Encryption Standard." *Computer*, June 1977.

DIFF79 Diffie, W., and Hellman, M. "Privacy and Authentication: An Introduction to Cryptography." *Proceedings of the IEEE*, March 1979.

DIFF88 Diffie, W. "The First Ten Years of Public-Key Cryptography." *Proceedings of the IEEE*, May 1988. Reprinted in [SIMM92].

DOBB96 Dobbertin, H. "The Status of MD5 After a Recent Attack." *CryptoBytes*, Summer 1996.

DORA03 Doraswamy, N., and Harkins, D. *IPSec*. Upper Saddle River, NJ: Prentice Hall, 2003.

EFF98 Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design*. Sebastopol, CA: O'Reilly, 1998

ELGA85 ElGamal, T. "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." *IEEE Transactions on Information Theory*, July 1985.

ELLI70 Ellis, J. *The Possibility of Secure Non-Secret Digital Encryption*. CESG Report, January 1970.

ELLI99 Ellis, J. "The History of Non-Secret Encryption." *Cryptologia*, July 1999.

ENGE80 Enger, N., and Howerton, P. *Computer Security*. New York: Amacom, 1980.

ENGE99 Enge, A. *Elliptic Curves and Their Applications to Cryptography*. Norwell, MA: Kluwer Academic Publishers, 1999.

FEIS73 Feistel, H. "Cryptography and Computer Privacy." *Scientific American*, May 1973.

FEIS75 Feistel, H.; Notz, W.; and Smith, J. "Some Cryptographic Techniques for Machine-to-Machine Data Communications." *Proceedings of the IEEE*, November 1975.

FELT03 Felten, E. "Understanding Trusted Computing: Will Its Benefits Outweigh its Drawbacks?" *IEEE Security and Privacy*, May/June 2003.

FERN99 Fernandes, A. "Elliptic Curve Cryptography." *Dr. Dobb's Journal*, December 1999.

FLUH00 Fluhrer, S., and McGrew, D. "Statistical Analysis of the Alleged RC4 Key Stream Generator." *Proceedings, Fast Software Encryption 2000*, 2000.

FLUH01 Fluhrer, S.; Mantin, I.; and Shamir, A. "Weakness in the Key Scheduling Algorithm of RC4." *Proceedings, Workshop in Selected Areas of Cryptography*, 2001.

FORD95 Ford, W. "Advances in Public-Key Certificate Standards." *ACM SIGSAC Review*, July 1995.

FORR97 Forrest, S.; Hofmeyr, S.; and Somayaji, A. "Computer Immunology." *Communications of the ACM*, October 1997.

FRAN01 Frankel, S. *Demystifying the IPSec Puzzle*. Boston: Artech House, 2001.

FREE93 Freedman, D. "The Goods on Hacker Hoods." *Forbes ASAP*, 13 September 1993.

FUMY93 Fumy, S., and Landrock, P. "Principles of Key Management." *IEEE Journal on Selected Areas in Communications*, June 1993.

GARD72 Gardner, M. *Codes, Ciphers, and Secret Writing*. New York: Dover, 1972.

GARD77 Gardner, M. "A New Kind of Cipher That Would Take Millions of Years to Break." *Scientific American*, August 1977.

GARR01 Garrett, P. *Making, Breaking Codes: An Introduction to Cryptology*. Upper Saddle River, NJ: Prentice Hall, 2001.

GASS88 Gasser, M. *Building a Secure Computer System*. New York: Van Nostrand Reinhold, 1988.

GAUD00 Gaudin, S. "The Omega Files." *Network World*, June 26, 2000.

GILB03 Gilbert, H. and Handschuh, H. "Security Analysis of SHA-256 and Sisters." *Proceedings, CRYPTO '03*, 2003; published by Springer-Verlag.

GOLL99 Gollmann, D. *Computer Security*. New York: Wiley, 1999.

GONG92 Gong, L. "A Security Risk of Depending on Synchronized Clocks." *Operating Systems Review*, January 1992.

GONG93 Gong, L. "Variations on the Themes of Message Freshness and Replay." *Proceedings, IEEE Computer Security Foundations Workshop*, June 1993.

GRAH94 Graham, R.; Knuth, D.; and Patashnik, O. *Concrete Mathematics: A Foundation for Computer Science*. Reading, MA: Addison-Wesley, 1994.

GUTM02 Gutmann, P. "PKI: It's Not Dead, Just Resting." *Computer*, August 2002.

HAMM91 Hamming, R. *The Art of Probability for Scientists and Engineers*. Reading, MA: Addison-Wesley, 1991.

HANK04 Hankerson, D.; Menezes, A.; and Vanstone, S. *Guide to Elliptic Curve Cryptography*. New York: Springer, 2004.

HARL01 Harley, D.; Slade, R.; and Gattiker, U. *Viruses Revealed*. New York: Osborne/McGraw-Hill, 2001.

HEBE92 Heberlein, L.; Mukherjee, B.; and Levitt, K. "Internetwork Security Monitor: An Intrusion-Detection System for Large-Scale Networks." *Proceedings, 15th National Computer Security Conference*, October 1992.

HELD96 Held, G. *Data and Image Compression: Tools and Techniques*. New York: Wiley, 1996.

HELL78 Hellman, M. "An Overview of Public Key Cryptography." *IEEE Communications Magazine*, November 1978.

HERS75 Herstein, I. *Topics in Algebra*. New York: Wiley, 1975.

HEVI99 Hevia, A., and Kiwi, M. "Strength of Two Data Encryption Standard Implementations Under Timing Attacks." *ACM Transactions on Information and System Security*, November 1999.

HEYS95 Heys, H., and Tavares, S. "Avalanche Characteristics of Substitution-Permutation Encryption Networks." *IEEE Transactions on Computers*, September 1995.

HONE01 The Honeynet Project. *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community*. Reading, MA: Addison-Wesley, 2001.

HORO71 Horowitz, E. "Modular Arithmetic and Finite Field Theory: A Tutorial." *Proceedings of the Second ACM Symposium and Symbolic and Algebraic Manipulation*, March 1971.

HUIT98 Huitema, C. *IPv6: The New Internet Protocol*. Upper Saddle River, NJ: Prentice Hall, 1998.

IANS90 I'Anson, C., and Mitchell, C. "Security Defects in CCITT Recommendation X.509The Directory Authentication Framework." *Computer Communications Review*, April 1990.

ILGU93 Ilgun, K. "USTAT: A Real-Time Intrusion Detection System for UNIX." *Proceedings, 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1993.

IWAT03 Iwata, T., and Kurosawa, K. "OMAC: One-Key CBC MAC." *Proceedings, Fast Software Encryption*, FSE '03, 2003.

JAIN91 Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York: Wiley, 1991.

JAKO98 Jakobsson, M.; Shriver, E.; Hillyer, B.; and Juels, A. "A practical secure physical random bit generator." *Proceedings of The Fifth ACM Conference on Computer and Communications Security*, November 1998.

JAVI91 Javitz, H., and Valdes, A. "The SRI IDES Statistical Anomaly Detector." *Proceedings, 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1991.

JONE82 Jones, R. "Some Techniques for Handling Encipherment Keys." *ICL Technical Journal*, November 1982.

JUEN85 Jueneman, R.; Matyas, S.; and Meyer, C. "Message Authentication." *IEEE Communications Magazine*, September 1988.

JUEN87 Jueneman, R. "Electronic Document Authentication." *IEEE Network Magazine*, April 1987.

JUN99 Jun, B., and Kocher, P. *The Intel Random Number Generator*. Intel White Paper, April 22, 1999.

JUNO04 Junod, P., and Vaudenay, S. "Perfect Diffusion Primitives for Block Ciphers: Building Efficient MDS Matrices." *Selected Areas in Cryptography 2004*, Waterloo, Canada, August 910, 2004.

JURI97 Jurisic, A., and Menezes, A. "Elliptic Curves and Cryptography." *Dr. Dobb's Journal*, April 1997.

KAHN96 Kahn, D. *The Codebreakers: The Story of Secret Writing*. New York: Scribner, 1996.

KALI95 Kaliski, B., and Robshaw, M. "The Secure Use of RSA." *CryptoBytes*, Autumn 1995.

KALI96a Kaliski, B., and Robshaw, M. "Multiple Encryption: Weighing Security and Performance." *Dr. $$ $Dobb's Journal*, January 1996.

KALI96b Kaliski, B. "Timing Attacks on Cryptosystems." *RSA Laboratories Bulletin*, January 1996. http://www.rsasecurity.com/rsalabs

KATZ00 Katzenbeisser, S., ed. *Information Hiding Techniques for Steganography and Digital Watermarking*. Boston: Artech House, 2000.

KEHN92 Kehne, A.; Schonwalder, J.; and Langendorfer, H. "A Nonce-Based Protocol for Multiple Authentications" *Operating Systems Review*, October 1992.

KELS98 Kelsey, J.; Schneier, B.; and Hall, C. "Cryptanalytic Attacks on Pseudorandom Number Generators." *Proceedings, Fast Software Encryption*, 1998. http://www.schneier.com/paper-prngs.html

KENT00 Kent, S. "On the Trail of Intrusions into Information Systems." *IEEE Spectrum*, December 2000.

KEPH97a Kephart, J.; Sorkin, G.; Chess, D.; and White, S. "Fighting Computer Viruses." *Scientific American*, November 1997.

KEPH97b Kephart, J.; Sorkin, G.; Swimmer, B.; and White, S. "Blueprint for a Computer Immune System." *Proceedings, Virus Bulletin International Conference*, October 1997.

KITS04 Kitsos, P., and Koufopaviou, O. "Architecture and Hardware Implementation of the Whirlpool Hash Function." *IEEE Transactions on Consumer Electronics*, February 2004.

KLEI90 Klein, D. "Foiling the Cracker: A Survey of, and Improvements to, Password Security." *Proceedings, UNIX Security Workshop II*, August 1990.

KNUD98 Knudsen, L., et al. "Analysis Method for Alleged RC4." *Proceedings, ASIACRYPT '98*, 1998.

KNUT97 Knuth, D. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Reading, MA: Addison-Wesley, 1997.

KNUT98 Knuth, D. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1998.

KOBL92 Koblas, D., and Koblas, M. "SOCKS." *Proceedings, UNIX Security Symposium III*, September 1992.

KOBL94 Koblitz, N. *A Course in Number Theory and Cryptography*. New York: Springer-Verlag, 1994.

KOCH96 Kocher, P. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems." *Proceedings, Crypto '96*, August 1996.

KOCH98 Kocher, P.; Jaffe, J.; and Jun, B. Introduction to Differential Power Analysis and Related Attacks." http://www.cryptography.com/dpa/technical/index.html

KOHL89 Kohl, J. "The Use of Encryption in Kerberos for Network Authentication." *Proceedings, Crypto '89*, 1989; published by Springer-Verlag.

KOHL94 Kohl, J.; Neuman, B.; and Ts'o, T. "The Evolution of the Kerberos Authentication Service." in Brazier, F., and Johansen, D. *Distributed Open Systems*. Los Alamitos, CA: IEEE Computer Society Press, 1994. Available at http://web.mit.edu/kerberos/www/papers.html

KOHN78 Kohnfelder, L. *Towards a Practical Public-Key Cryptosystem*. Bachelor's Thesis, M.I.T., May 1978.

KONH81 Konheim, A. *Cryptography: A Primer*. New York: Wiley, 1981.

KORN96 Korner, T. *The Pleasures of Counting*. Cambridge, England: Cambridge University Press, 1996.

KUMA97 Kumar, I. *Cryptology*. Laguna Hills, CA: Aegean Park Press, 1997.

KUMA98 Kumanduri, R., and Romero, C. *Number Theory with Computer Applications*. Upper Saddle River, NJ: Prentice Hall, 1998.

LAM92a Lam, K., and Gollmann, D. "Freshness Assurance of Authentication Protocols" *Proceedings, ESORICS 92*, 1992; published by Springer-Verlag.

LAM92b Lam, K., and Beth, T. "Timely Authentication in Distributed Systems." *Proceedings, ESORICS 92*, 1992; published by Springer-Verlag.

LAND04 Landau, S. "Polynomials in the Nation's Service: Using Algebra to Design the Advanced Encryption Standard." American Mathematical Monthly, February 2004.

LE93 Le, A.; Matyas. S.; Johnson, D.; and Wilkins, J. "A Public Key Extension to the Common Cryptographic Architecture." *IBM Systems Journal*, No. 3, 1993.

LEHM51 Lehmer, D. "Mathematical Methods in Large-Scale Computing." *Proceedings, 2nd Symposium on Large-Scale Digital Calculating Machinery*, Cambridge: Harvard University Press, 1951.

LEUT94 Leutwyler, K. "Superhack." *Scientific American*, July 1994.

LEVE90 Leveque, W. *Elementary Theory of Numbers*. New York: Dover, 1990.

LEWA00 Lewand, R. *Cryptological Mathematics*. Washington, DC: Mathematical Association of America, 2000.

LEWI69 Lewis, P.; Goodman, A.; and Miller, J. "A Pseudo-Random Number Generator for the System/360." *IBM Systems Journal*, No. 2, 1969.

LIDL94 Lidl, R., and Niederreiter, H. *Introduction to Finite Fields and Their Applications*. Cambridge: Cambridge University Press, 1994.

LIPM00 Lipmaa, H.; Rogaway, P.; and Wagner, D. "CTR Mode Encryption." *NIST First Modes of Operation Workshop*, October 2000. http://csrc.nist.gov/encryption/modes

LODI98 Lodin, S., and Schuba, C. "Firewalls Fend Off Invasions from the Net." *IEEE Spectrum*, February 1998.

LUNT88 Lunt, T., and Jagannathan, R. "A Prototype Real-Time Intrusion-Detection Expert System." *Proceedings, 1988 IEEE Computer Society Symposium on Research in Security and Privacy*, April 1988.

MADS93 Madsen, J. "World Record in Password Checking." *Usenet, comp.security.misc newsgroup*, August 18, 1993.

MANT01 Mantin, I., Shamir, A. "A Practical Attack on Broadcast RC4." *Proceedings, Fast Software Encryption*, 2001.

MARK97 Markham, T. "Internet Security Protocol." *Dr. Dobb's Journal*, June 1997.

MATS93 Matsui, M. "Linear Cryptanalysis Method for DES Cipher." *Proceedings, EUROCRYPT '93*, 1993; published by Springer-Verlag.

MATY91a Matyas, S. "Key Handling with Control Vectors." *IBM Systems Journal*, No.2, 1991.

MATY91b Matyas, S.; Le, A.; and Abraham, D. "A Key-Management Scheme Based on Control Vectors." *IBM Systems Journal*, No. 2, 1991.

MCHU00 McHugh, J.; Christie, A.; and Allen, J. "The Role of Intrusion Detection Systems." *IEEE Software*, September/October 2000.

MEIN01 Meinel, C. "Code Red for the Web." *Scientific American*, October 2001.

MENE97 Menezes, A.; van Oorschot, P.; and Vanstone, S. *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1997.

MERK78 Merkle, R., and Hellman, M. "Hiding Information and Signatures in Trap Door Knapsacks." *IEEE Transactions on Information Theory*, September 1978.

MERK79 Merkle, R. *Secrecy, Authentication, and Public Key Systems*. Ph.D. Thesis, Stanford University, June 1979.

MERK81 Merkle, R., and Hellman, M. "On the Security of Multiple Encryption." *Communications of the ACM*, July 1981.

MERK89 Merkle, R. "One Way Hash Functions and DES." *Proceedings, CRYPTO '89*, 1989; published by Springer-Verlag.

MEYE82 Meyer, C., and Matyas, S. *Cryptography: A New Dimension in Computer Data Security*. New York: Wiley, 1982.

MEYE88 Meyer, C., and Schilling, M. "Secure Program Load with Modification Detection Code." *Proceedings, SECURICOM 88*, 1988.

MILL75 Miller, G. "Riemann's Hypothesis and Tests for Primality." *Proceedings of the Seventh Annual ACM Symposium on the Theory of Computing*, May 1975.

MILL88 Miller, S.; Neuman, B.; Schiller, J.; and Saltzer, J. "Kerberos Authentication and Authorization System." *Section E.2.1, Project Athena Technical Plan*, M.I.T. Project Athena, Cambridge, MA. October 27, 1988.

MIRK04 Mirkovic, J., and Relher, P. "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms." *ACM SIGCOMM Computer Communications Review*, April 2004.

MIST96 Mister, S., and Adams, C. "Practical S-Box Design." *Proceedings, Workshop in Selected Areas of Cryptography, SAC' 96*. 1996.

MIST98 Mister, S., and Tavares, S. "Cryptanalysis of RC4-Like Ciphers." *Proceedings, Workshop in Selected Areas of Cryptography, SAC' 98*. 1998.

MITC90 Mitchell, C.; Walker, M.; and Rush, D. "CCITT/ISO Standards for Secure Message Handling." *IEEE Journal on Selected Areas in Communications*, May 1989.

MITC92 Mitchell, C.; Piper, F.; and Wild, P. "Digital Signatures." In [SIMM92].

MIYA90 Miyaguchi, S.; Ohta, K.; and Iwata, M. "Confirmation that Some Hash Functions Are Not Collision Free." *Proceedings, EUROCRYPT '90*, 1990; published by Springer-Verlag.

MOOR01 Moore, M. "Inferring Internet Denial-of-Service Activity." *Proceedings of the 10th USENIX Security Symposium*, 2001.

MUFT89 Muftic, S. *Security Mechanisms for Computer Networks*. New York: Ellis Horwood, 1989.

MURP90 Murphy, S. "The Cryptanalysis of FEAL-4 with 20 Chosen Plaintexts." *Journal of Cryptology*, No. 3, 1990.

MUSA03 Musa, M.; Schaefer, E.; and Wedig, S. "A Simplified AES Algorithm and Its Linear and Differential Cryptanalyses." *Cryptologia*, April 2003.

MYER91 Myers, L. *Spycomm: Covert Communication Techniques of the Underground*. Boulder, CO: Paladin Press, 1991.

NACH97 Nachenberg, C. "Computer Virus-Antivirus Coevolution." *Communications of the ACM*, January 1997.

NECH92 Nechvatal, J. "Public Key Cryptography." In [SIMM92].

NECH00 Nechvatal, J., et al. *Report on the Development of the Advanced Encryption Standard*. National Institute of Standards and Technology. October 2, 2000.

NEED78 Needham, R., and Schroeder, M. "Using Encryption for Authentication in Large Networks of Computers." *Communications of the ACM*, December 1978.

NEUM90 Neumann, P. "Flawed Computer Chip Sold for Years." *RISKS-FORUM Digest*, Vol.10, No. 54, October 18, 1990.

NEUM93a Neuman, B., and Stubblebine, S. "A Note on the Use of Timestamps as Nonces." *Operating Systems Review*, April 1993.

NEUM93b Neuman, B. "Proxy-Based Authorization and Accounting for Distributed Systems." *Proceedings of the 13th International Conference on Distributed Computing Systems*, May 1993.

NICH96 Nichols, R. *Classical Cryptography Course*. Laguna Hills, CA: Aegean Park Press, 1996.

NING04 Ning, P., et al. "Techniques and Tools for Analyzing Intrusion Alerts." *ACM Transactions on Information and System Security*, May 2004.

NIST97 National Institute of Standards and Technology. "Request for Candidate Algorithm Nominations for the Advanced Encryption Standard." *Federal Register*, September 12, 1997.

ODLY95 Odlyzko, A. "The Future of Integer Factorization." *CryptoBytes*, Summer 1995.

OPPL97 Oppliger, R. "Internet Security: Firewalls and Beyond." *Communications of the ACM*, May 1997.

OPPL05 Oppliger, R., and Rytz, R. "Does Trusted Computing Remedy Computer Security Problems?" *IEEE Security and Privacy*, March/April 2005.

ORE67 Ore, O. *Invitation to Number Theory*. Washington, DC: The Mathematical Association of America, 1967,

ORE76 Ore, O. *Number Theory and Its History*. New York: Dover, 1976.

PARK88 Park, S., and Miller, K. "Random Number Generators: Good Ones are Hard to Find." *Communications of the ACM*, October 1988.

PATR04 Patrikakis, C.; Masikos, M.; and Zouraraki, O. "Distributed Denial of Service Attacks." *The Internet Protocol Journal*, December 2004.

PERL99 Perlman, R. "An Overview of PKI Trust Models." *IEEE Network*, November/December 1999.

PFLE02 Pfleeger, C. *Security in Computing*. Upper Saddle River, NJ: Prentice Hall, 2002.

PFLE03 Pfleeger, C. *Security in Computing*. Upper Saddle River, NJ: Prentice Hall, 1997.

PIAT91 Piattelli-Palmarini, M. "Probability: Neither Rational nor Capricious." *Bostonia*, March 1991.

PIEP03 Pieprzyk, J.; Hardjono, T.; and Seberry, J. *Fundamentals of Computer Security*. New York: Springer-Verlag, 2003.

POHL81 Pohl, I., and Shaw, A. *The Nature of Computation: An Introduction to Computer Science*. Rockville, MD: Computer Science Press, 1981.

POIN02 Pointcheval, D. "How to Encrypt Properly with RSA." *CryptoBytes*, Winter/Spring 2002. http://www.rsasecurity.com/rsalabs

POPE79 Popek, G., and Kline, C. "Encryption and Secure Computer Networks." *ACM Computing Surveys*, December 1979.

PORR92 Porras, P. *STAT: A State Transition Analysis Tool for Intrusion Detection*. Master's Thesis, University of California at Santa Barbara, July 1992.

PREN93a Preneel, B.; Govaerta, R.; and Vandewalle, J. "Hash Functions Based on Block Ciphers: a Synthetic Approach." *Proceedings, Advances in CryptologyCRYPTO '93*, 1993.

PREN93b Preneel, B. "Cryptographic Hash Functions." *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography*, 1993.

PREN96 Preneel, B., and Oorschot, P. "On the Security of Two MAC Algorithms." *Lecture Notes in Computer Science 1561; Lectures on Data Security*, 1999; published by Springer-Verlag.

PREN99 Preneel, B. "The State of Cryptographic Hash Functions." *Proceedings, EUROCRYPT '96*, 1996; published by Springer-Verlag.

PROC01 Proctor, P. *The Practical Intrusion Detection Handbook*. Upper Saddle River, NJ: Prentice Hall, 2001.

RABI78 Rabin, M. "Digitalized Signatures." In *Foundations of Secure Computation*, DeMillo, R.; Dobkin, D.; Jones, A.; and Lipton, R., eds. New York: Academic Press, 1978.

RABI80 Rabin, M. "Probabilistic Algorithms for Primality Testing." *Journal of Number Theory*, December 1980.

RAND55 Rand Corporation. *A Million Random Digits*. New York: The Free Press, 1955. http://www.rand.org/publications/classics/randomdigits

RESC01 Rescorla, E. *SSL and TLS: Designing and Building Secure Systems*. Reading, MA: Addison-Wesley, 2001.

RIBE96 Ribenboim, P. *The New Book of Prime Number Records*. New York: Springer-Verlag, 1996.

RIJM96 Rijmen, V.; Daemen, J.; Preneel, B.; Bosselares, A.; and Win, E. "The Cipher SHARK." *Fast Software Encryption, FSE '96*, 1996.

RITT91 Ritter, T. "The Efficient Generation of Cryptographic Confusion Sequences." *Cryptologia*, vol. 15 no. 2, 1991. http://www.ciphersbyritter.com/ARTS/CRNG2ART.HTM

RIVE78 Rivest, R.; Shamir, A.; and Adleman, L. "A Method for Obtaining Digital Signatures and Public Key Cryptosystems." *Communications of the ACM*, February 1978.

RIVE84 Rivest, R., and Shamir, A. "How to Expose an Eavesdropper." *Communications of the ACM*, April 1984.

ROBS95a Robshaw, M. *Stream Ciphers*. RSA Laboratories Technical Report TR-701, July 1995. http://www.rsasecurity.com/rsalabs

ROBS95b Robshaw, M. *Block Ciphers*. RSA Laboratories Technical Report TR-601, August 1995. http://www.rsasecurity.com/rsalabs

ROBS95c Robshaw, M. *MD2, MD4, MD5, SHA and Other Hash Functions*. RSA Laboratories Technical Report TR-101, July 1995. http://www.rsasecurity.com/rsalabs

RODR02 Rodriguez, A., et al. *TCP/IP Tutorial and Technical Overview*. Upper Saddle River: NJ: Prentice Hall, 2002.

ROSE05 Rosen, K. *Elementary Number Theory and its Applications*. Reading, MA: Addison-Wesley, 2000.

ROSI99 Rosing, M. *Implementing Elliptic Curve Cryptography*. Greenwich, CT: Manning Publications, 1999.

RUBI97 Rubin, A. "An Experience Teaching a Graduate Course in Cryptography." *Cryptologia*, April 1997.

RUEP92 Rueppel, T. "Stream Ciphers." In [SIMM92].

SAFF93 Safford, D.; Schales, D.;, and Hess, D. "The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment." *Proceedings, UNIX Security Symposium IV*, October 1993.

SAUE81 Sauer, C., and Chandy, K. *Computer Systems Performance Modeling*. Englewood Cliffs, NJ: Prentice Hall, 1981.

SCHA96 Schaefer, E. "A Simplified Data Encryption Standard Algorithm." *Cryptologia*, January 1996.

SCHN91 Schnorr, C. "Efficient Signatures for Smart Card." *Journal of Cryptology*, No. 3, 1991.

SCHN96 Schneier, B. *Applied Cryptography*. New York: Wiley, 1996.

SCHN00 Schneier, B. *Secrets and Lies: Digital Security in a Networked World*. New York: Wiley, 2000.

SHAM82 Shamir, A. "A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem." *Proceedings, 23rd IEEE Symposium on the Foundations of Computer Science*, 1982.

SHAM03 Shamir, A., and Tromer, E. "On the Cost of Factoring RSA-1024." *CryptoBytes*, Summer 2003. http://www.rsasecurity.com/rsalabs

SHAN49 Shannon, C. "Communication Theory of Secrecy Systems." *Bell Systems Technical Journal*, No. 4, 1949.

SIMM92 Simmons, G., ed. *Contemporary Cryptology: The Science of Information Integrity*. Piscataway, NJ: IEEE Press, 1992.

SIMM93 Simmons, G. "Cryptology." *Encyclopaedia Britannica*, *Fifteenth Edition*, 1993.

SIMO95 Simovits, M. *The DES: An Extensive Documentation and Evaluation*. Laguna Hills, CA: Aegean Park Press, 1995.

SING99 Singh, S. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. New York: Anchor Books, 1999.

SINK66 Sinkov, A. *Elementary Cryptanalysis: A Mathematical Approach*. Washington, DC: The Mathematical Association of America, 1966.

SMIT97 Smith, R. *Internet Cryptography*. Reading, MA: Addison-Wesley, 1997.

SNAP91 Snapp, S., et al. "A System for Distributed Intrusion Detection." *Proceedings, COMPCON Spring '91*, 1991.

SPAF92a Spafford, E. "Observing Reusable Password Choices." *Proceedings, UNIX Security Symposium III*, September 1992.

SPAF92b Spafford, E. "OPUS: Preventing Weak Password Choices." *Computers and Security*, No. 3, 1992.

STAL02 Stallings, W. "The Advanced Encryption Standard." *Cryptologia*, July 2002.

STAL04 Stallings, W. *Computer Networking with Internet Protocols and Technology*. Upper Saddle River, NJ: Prentice Hall, 2004.

STAL06 Stallings, W. "The Whirlpool Secure Hash Function." *Cryptologia*, January 2006.

STEI88 Steiner, J.; Neuman, C.; and Schiller, J. "Kerberos: An Authentication Service for Open Networked Systems." *Proceedings of the Winter 1988 USENIX Conference*, February 1988.

STEP93 Stephenson, P. "Preventive Medicine." *LAN Magazine*, November 1993.

STER92 Sterling, B. *The Hacker Crackdown: Law and Disorder on the Electronic Frontier*. New York: Bantam, 1992.

STIN02 Stinson, D. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 2002.

STOL88 Stoll, C. "Stalking the Wily Hacker." *Communications of the ACM*, May 1988.

STOL89 Stoll, C. *The Cuckoo's Egg*. New York: Doubleday, 1989.

SZOR05 Szor, P. *The Art of Computer Virus Research and Defense*. Reading, MA: Addison-Wesley, 2005.

THOM84 Thompson, K. "Reflections on Trusting Trust (Deliberate Software Bugs)." *Communications of the ACM*, August 1984.

TIME90 Time, Inc. *Computer Security, Understanding Computers Series*. Alexandria, VA: Time-Life Books, 1990.

TIPP27 Tippett, L. *Random Sampling Numbers*. Cambridge, England: Cambridge University Press, 1927.

TSUD92 Tsudik, G. "Message Authentication with One-Way Hash Functions." *Proceedings, INFOCOM '92*, May 1992.

TUCH79 Tuchman, W. "Hellman Presents No Shortcut Solutions to DES." *IEEE Spectrum*, July 1979.

TUNG99 Tung, B. *Kerberos: A Network Authentication System*. Reading, MA: Addison-Wesley, 1999.

VACC89 Vaccaro, H., and Liepins, G. "Detection of Anomalous Computer Session Activity." *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1989.

VANO90 van Oorschot, P., and Wiener, M. "A Known-Plaintext Attack on Two-Key Triple Encryption." *Proceedings, EUROCRYPT '90*, 1990; published by Springer-Verlag.

VANO94 van Oorschot, P., and Wiener, M. "Parallel Collision Search with Application to Hash Functions and Discrete Logarithms." *Proceedings, Second ACM Conference on Computer and Communications Security*, 1994.

VIJA02 Vijayan, J. "Denial-of-Service Attacks Still a Threat." *ComputerWorld*, April 8, 2002.

VOYD83 Voydock, V., and Kent., S. "Security Mechanisms in High-Level Network Protocols." *Computing Surveys*, June 1983.

WACK02 Wack, J.; Cutler, K.; and Pole, J. *Guidelines on Firewalls and Firewall Policy*. NIST Special Publication SP 800-41, January 2002.

WAGN00 Wagner, D., and Goldberg, I. "Proofs of Security for the UNIX Password Hashing Algorithm." *Proceedings, ASIACRYPT '00*, 2000.

WANG05 Wang, X.; Yin, Y.; and Yu, H. "Finding Collisions in the Full SHA-1. *Proceedings, Crypto '05*, 2005; published by Springer-Verlag.

WAYN96 Wayner, P. *Disappearing Cryptography*. Boston: AP Professional Books, 1996.

WEBS86 Webster, A., and Tavares, S. "On the Design of S-Boxes." *Proceedings, Crypto '85*, 1985; published by Springer-Verlag.

WIEN90 Wiener, M. "Cryptanalysis of Short RSA Secret Exponents." *IEEE Transactions on Information*

Theory, vol. IT-36, 1990.

WILS05 Wilson, J. "The Future of the Firewall." *Business Communications Review*, May 2005.

WOO92a Woo, T., and Lam, S. "Authentication for Distributed Systems." *Computer*, January 1992.

WOO92b Woo, T., and Lam, S. "'Authentication' Revisited." *Computer*, April 1992.

YUVA79 Yuval, G. "How to Swindle Rabin." *Cryptologia*, July 1979.

ZENG91 Zeng. K.; Yang, C.; Wei, D.; and Rao, T. "Pseudorandom Bit Generators in Stream-Cipher Cryptography." *Computer*, February 1991.

ZIV77 Ziv, J., and Lempel, A. "A Universal Algorithm for Sequential Data Compression." *IEEE Transactions on Information Theory*, May 1977.

PREV NEXT

[Page InsideFrontCover]

Inside Front Cover

THE WILLIAM STALLINGS BOOKS ON COMPUTER AND DATA COMMUNICATIONS TECHNOLOGY

DATA AND COMPUTER COMMUNICATIONS, SEVENTH EDITION

A comprehensive survey that has become the standard in the field, covering (1) data communications, including transmission, media, signal encoding, link control, and multiplexing; (2) communication networks, including circuit- and packet-switched, frame relay, ATM, and LANs; (3) the TCP/IP protocol suite, including IPv6, TCP, MIME, and HTTP, as well as a detailed treatment of network security. **Received the 2000 Text and Academic Authors Association (TAA) award for long-term excellence in a Computer Science Textbook**. ISBN 0-13-100681-9

COMPUTER NETWORKS WITH INTERNET PROTOCOLS AND TECHNOLOGY

An up-to-date survey of developments in the area of Internet-based protocols and algorithms. Using a top-down approach, this book covers applications, transport layer, Internet QoS, Internet routing, data link layer and computer networks, security, and network management. ISBN 0-13-141098-9

COMPUTER ORGANIZATION AND ARCHITECTURE, SEVENTH EDITION

A unified view of this broad field. Covers fundamentals such as CPU, control unit, microprogramming, instruction set, I/O, and memory. Also covers advanced topics such as RISC, superscalar, and parallel organization. **Fourth and fifth editions received the TAA award for the best Computer Science and Engineering Textbook of the year**. ISBN 0-13-185644-8

WIRELESS COMMUNICATIONS AND NETWORKS, Second Edition

A comprehensive, state-of-the art survey. Covers fundamental wireless communications topics, including antennas and propagation, signal encoding techniques, spread spectrum, and error correction techniques. Examines satellite, cellular, wireless local loop networks and wireless LANs, including Bluetooth and 802.11. Covers Mobile IP and WAP. ISBN 0-13-191835-4

BUSINESS DATA COMMUNICATIONS, FIFTH EDITION

A comprehensive presentation of data communications and telecommunications from a business perspective. Covers voice, data, image, and video communications and applications technology and includes a number of case studies. ISBN 0-13-144257-0

OPERATING SYSTEMS, FIFTH EDITION

A state-of-the art survey of operating system principles. Covers fundamental technology as well as contemporary design issues, such as threads, microkernels, SMPs, real-time systems, multiprocessor scheduling, distributed systems, clusters, security, and object-oriented design. **Fourth edition received the TAA award for the best Computer Science and Engineering Textbook of 2002.** ISBN 0-13-147954-7

NETWORK SECURITY ESSENTIALS, SECOND EDITION

A tutorial and survey on network security technology. The book covers important network security tools and applications, including S/MIME, IP Security, Kerberos, SSL/TLS, SET, and X509v3. In addition, methods for countering hackers and viruses are explored. ISBN 0-13-035128-8

LOCAL AND METROPOLITAN AREA NETWORKS, SIXTH EDITION

An in-depth presentation of the technology and architecture of local and metropolitan area networks. Covers topology, transmission media, medium access control, standards, internetworking, and network management. Provides an up-to-date coverage of LAN/MAN systems, including Fast Ethernet, Fibre Channel, and wireless LANs, plus LAN QoS. **Received the 2001 TAA award for long-term excellence in a Computer Science Textbook**. ISBN 0-13-012939-9

ISDN AND BROADBAND ISDN, WITH FRAME RELAY AND ATM: FOURTH EDITION

An in-depth presentation of the technology and architecture of integrated services digital networks (ISDN). Covers the integrated digital network (IDN), xDSL, ISDN services and architecture, signaling system no. 7 (SS7) and provides detailed coverage of the ITU-T protocol standards. Also provides detailed coverage of protocols and congestion control strategies for both frame relay and ATM. ISBN 0-13-973744-8

HIGH-SPEED NETWORKS AND INTERNETS, SECOND EDITION

A state-of-the art survey of high-speed networks. Topics covered include TCP congestion control, ATM traffic management, Internet traffic management, differentiated and integrated services, Internet routing protocols and multicast routing protocols, resource reservation and RSVP, and lossless and lossy compression. Examines important topic of self-similar data traffic. ISBN 0-13-03221-0

PREV NEXT

[Page InsideBackCover]

Inside Back Cover

Acronyms

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
AH	Authentication Header
ANSI	American National Standards Institute
CBC	Cipher Block Chaining
CC	Common Criteria
CESG	Communications-Electronics Security Group
CFB	Cipher Feedback
CMAC	Cipher-Based Message Authentication Code
CRT	Chinese Remainder Theorem
DDoS	Distributed Denial of Service
DES	Data Encryption Standard
DoS	Denial of Service
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ECB	Electronic Codebook
ESP	Encapsulating Security Payload
FIPS	Federal Information Processing Standard
IAB	Internet Architecture Board
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPSec	IP Security
ISO	International Organization for Standardization
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
IV	Initialization Vector
KDC	Key Distribution Center
LAN	Local Area Network

MAC	Message Authentication Code
MIC	Message Integrity Code
MIME	Multipurpose Internet Mail Extension
MD5	Message Digest, Version 5
MTU	Maximum Transmission Unit
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OFB	Output Feedback
PCBC	Propagating Cipher Block Chaining
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
PRNG	Pseudorandom Number Generator
RFC	Request for Comments
RNG	Random Number Generator
RSA	Rivest-Shamir-Adelman
SET	Secure Electronic Transaction
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
S/MIME	Secure MIME
SNMP	Simple Network Management Protocol
SNMPv3	Simple Network Management Protocol Version 3
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
WAN	Wide Area Network

Index

Index

Index

Index

Index

Index

Index

PREV NEXT

Index

PREV NEXT

Index

Index

Index

Iterated tunneling

PREV NEXT

Index

Index

Index

Index

Index

Index

Index

Index

Index

PREV NEXT

Index

Index

Index

Index

Index

Index