cryptography topic of providing **message integrity** (also known as message authentication). Along with message integrity, we will discuss two related topics in this section: digital signatures and end-point authentication.

We define the message integrity problem using, once again, Alice and Bob. Suppose Bob receives a message (which may be encrypted or may be in plaintext) and he believes this message was sent by Alice. To authenticate this message, Bob needs to verify:

1. The message indeed originated from Alice.
2. The message was not tampered with on its way to Bob.

We'll see in Sections 8.4 through 8.7 that this problem of message integrity is a critical concern in just about all secure networking protocols.

As a specific example, consider a computer network using a link-state routing algorithm (such as OSPF) for determining routes between each pair of routers in the network (see Chapter 4). In a link-state algorithm, each router needs to broadcast a link-state message to all other routers in the network. A router's link-state message includes a list of its directly connected neighbors and the direct costs to these neighbors. Once a router receives link-state messages from all of the other routers, it can create a complete map of the network, run its least-cost routing algorithm, and configure its forwarding table. One relatively easy attack on the routing algorithm is for Trudy to distribute bogus link-state messages with incorrect link-state information. Thus the need for message integrity—when router B receives a link-state message from router A, router B should verify that router A actually created the message and, further, that no one tampered with the message in transit.

In this section, we describe a popular message integrity technique that is used by many secure networking protocols. But before doing so, we need to cover another important topic in cryptography—cryptographic hash functions.

8.3.1 Cryptographic Hash Functions

As shown in Figure 8.7, a hash function takes an input, m, and computes a fixed-size string $H(m)$ known as a hash. The Internet checksum (Chapter 3) and CRCs (Chapter 4) meet this definition. A **cryptographic hash function** is required to have the following additional property:

- It is computationally infeasible to find any two different messages x and y such that $H(x) = H(y)$.

Informally, this property means that it is computationally infeasible for an intruder to substitute one message for another message that is protected by the hash function. That is, if $(m, H(m))$ are the message and the hash of the message created

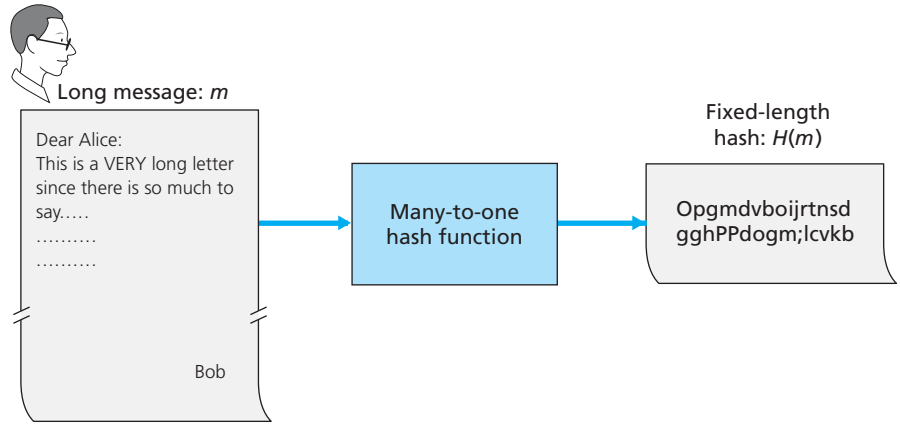

Figure 8.7 ♦ Hash functions

by the sender, then an intruder cannot forge the contents of another message, *y*, that has the same hash value as the original message.

Let's convince ourselves that a simple checksum, such as the Internet checksum, would make a poor cryptographic hash function. Rather than performing 1s complement arithmetic (as in the Internet checksum), let us compute a checksum by treating each character as a byte and adding the bytes together using 4-byte chunks at a time. Suppose Bob owes Alice $100.99 and sends an IOU to Alice consisting of the text string "`IOU100.99BOB`." The ASCII representation (in hexadecimal notation) for these letters is `49, 4F, 55, 31, 30, 30, 2E, 39, 39, 42, 4F, 42`.

Figure 8.8 (top) shows that the 4-byte checksum for this message is B2 C1 D2 AC. A slightly different message (and a much more costly one for Bob) is shown in the bottom half of Figure 8.8. The messages "`IOU100.99BOB`" and "`IOU900.19BOB`" have the *same* checksum. Thus, this simple checksum algorithm violates the requirement above. Given the original data, it is simple to find another set of data with the same checksum. Clearly, for security purposes, we are going to need a more powerful hash function than a checksum.

The MD5 hash algorithm of Ron Rivest [RFC 1321] is in wide use today. It computes a 128-bit hash in a four-step process consisting of a padding step (adding a one followed by enough zeros so that the length of the message satisfies certain conditions), an append step (appending a 64-bit representation of the message length before padding), an initialization of an accumulator, and a final looping step in which the message's 16-word blocks are processed (mangled) in four rounds. For a description of MD5 (including a C source code implementation) see [RFC 1321].