used either with MD5 or SHA-1. HMAC actually runs data and the authentication key through the hash function twice [Kaufman 1995; RFC 2104].

There still remains an important issue. How do we distribute the shared authentication key to the communicating entities? For example, in the link-state routing algorithm, we would somehow need to distribute the secret authentication key to each of the routers in the autonomous system. (Note that the routers can all use the same authentication key.) A network administrator could actually accomplish this by physically visiting each of the routers. Or, if the network administrator is a lazy guy, and if each router has its own public key, the network administrator could distribute the authentication key to any one of the routers by encrypting it with the router's public key and then sending the encrypted key over the network to the router.

8.3.3 Digital Signatures

Think of the number of the times you've signed your name to a piece of paper during the last week. You sign checks, credit card receipts, legal documents, and letters. Your signature attests to the fact that you (as opposed to someone else) have acknowledged and/or agreed with the document's contents. In a digital world, one often wants to indicate the owner or creator of a document, or to signify one's agreement with a document's content. A **digital signature** is a cryptographic technique for achieving these goals in a digital world.

Just as with handwritten signatures, digital signing should be done in a way that is verifiable and nonforgeable. That is, it must be possible to prove that a document signed by an individual was indeed signed by that individual (the signature must be verifiable) and that *only* that individual could have signed the document (the signature cannot be forged).

Let's now consider how we might design a digital signature scheme. Observe that when Bob signs a message, Bob must put something on the message that is unique to him. Bob could consider attaching a MAC for the signature, where the MAC is created by appending his key (unique to him) to the message, and then taking the hash. But for Alice to verify the signature, she must also have a copy of the key, in which case the key would not be unique to Bob. Thus, MACs are not going to get the job done here.

Recall that with public-key cryptography, Bob has both a public and private key, with both of these keys being unique to Bob. Thus, public-key cryptography is an excellent candidate for providing digital signatures. Let us now examine how it is done.

Suppose that Bob wants to digitally sign a document, m. We can think of the document as a file or a message that Bob is going to sign and send. As shown in Figure 8.10, to sign this document, Bob simply uses his private key, K_B^-, to compute $K_B^-(m)$. At first, it might seem odd that Bob is using his private key (which, as we saw in Section 8.2, was used to decrypt a message that had been encrypted

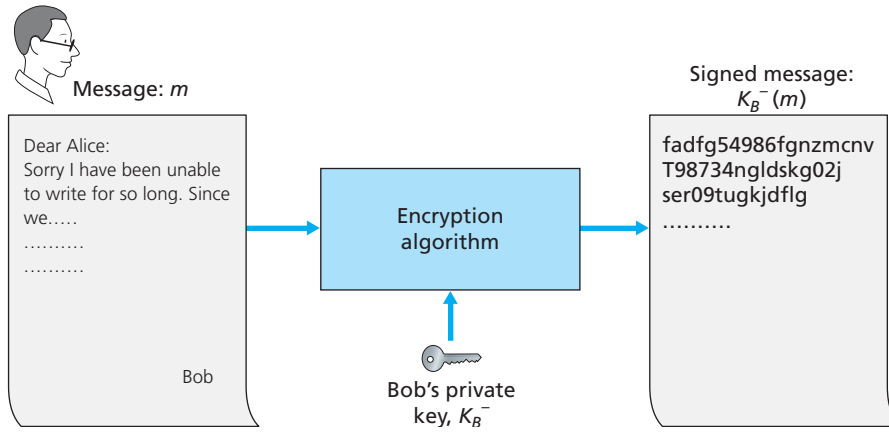

Figure 8.10 ♦ Creating a digital signature for a document

with his public key) to sign a document. But recall that encryption and decryption are nothing more than mathematical operations (exponentiation to the power of e or d in RSA; see Section 8.2) and recall that Bob's goal is not to scramble or obscure the contents of the document, but rather to sign the document in a manner that is verifiable and nonforgeable. Bob's digital signature of the document is $K_B^-(m)$.

Does the digital signature $K_B^-(m)$ meet our requirements of being verifiable and nonforgeable? Suppose Alice has m and $K_B^-(m)$. She wants to prove in court (being litigious) that Bob had indeed signed the document and was the only person who could have possibly signed the document. Alice takes Bob's public key, K_B^+, and applies it to the digital signature, $K_B^-(m)$, associated with the document, m. That is, she computes $K_B^+(K_B^-(m))$, and voilà, with a dramatic flurry, she produces m, which exactly matches the original document! Alice then argues that only Bob could have signed the document, for the following reasons:

- Whoever signed the message must have used the private key, K_B^-, in computing the signature $K_B^-(m)$, such that $K_B^+(K_B^-(m)) = m$.
- The only person who could have known the private key, K_B^-, is Bob. Recall from our discussion of RSA in Section 8.2 that knowing the public key, K_B^+, is of no help in learning the private key, K_B^-. Therefore, the only person who could know K_B^- is the person who generated the pair of keys, (K_B^+, K_B^-), in the first place, Bob. (Note that this assumes, though, that Bob has not given K_B^- to anyone, nor has anyone stolen K_B^- from Bob.)

It is also important to note that if the original document, m, is ever modified to some alternate form, m', the signature that Bob created for m will not be valid for m', since $K_B^+(K_B^-(m))$ does not equal m'. Thus we see that digital signatures also provide message integrity, allowing the receiver to verify that the message was unaltered as well as the source of the message.

One concern with signing data by encryption is that encryption and decryption are computationally expensive. Given the overheads of encryption and decryption, signing data via complete encryption/decryption can be overkill. A more efficient approach is to introduce hash functions into the digital signature. Recall from Section 8.3.2 that a hash algorithm takes a message, m, of arbitrary length and computes a fixed-length "fingerprint" of the message, denoted by $H(m)$. Using a hash function, Bob signs the hash of a message rather than the message itself, that is, Bob calculates $K_B^-(H(m))$. Since $H(m)$ is generally much smaller than the original message m, the computational effort required to create the digital signature is substantially reduced.

In the context of Bob sending a message to Alice, Figure 8.11 provides a summary of the operational procedure of creating a digital signature. Bob puts his original long message through a hash function. He then digitally signs the resulting hash

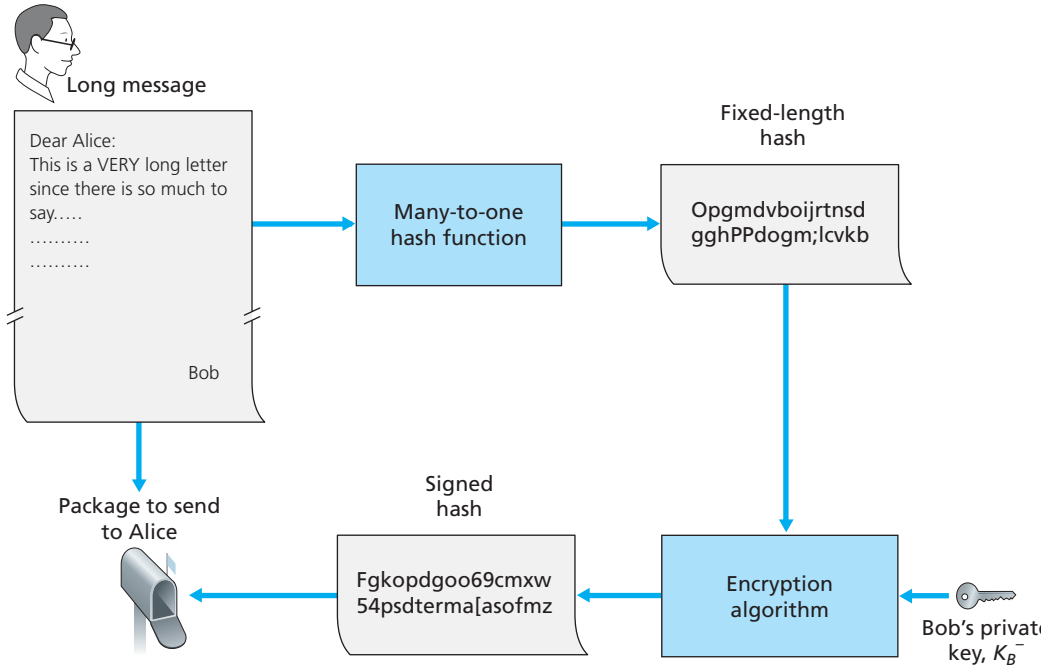

Figure 8.11 ♦ Sending a digitally signed message

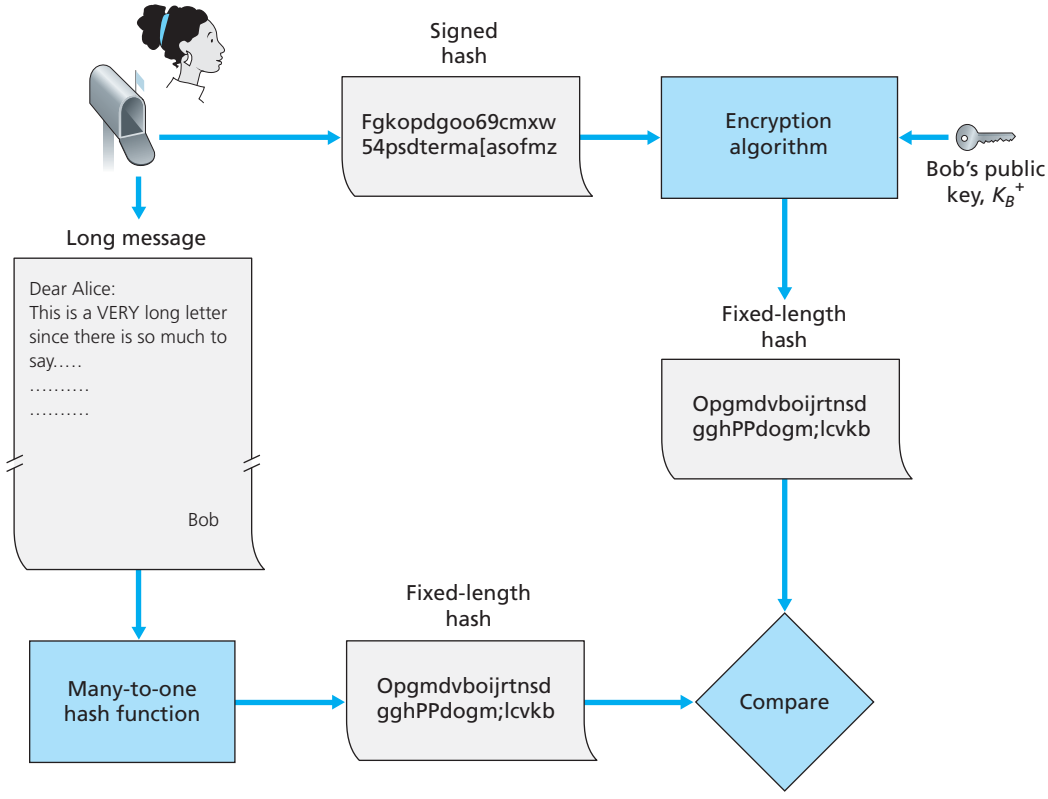

Figure 8.12 ♦ Verifying a signed message

with his private key. The original message (in cleartext) along with the digitally signed message digest (henceforth referred to as the digital signature) is then sent to Alice. Figure 8.12 provides a summary of the operational procedure of the signature. Alice applies the sender's public key to the message to obtain a hash result. Alice also applies the hash function to the cleartext message to obtain a second hash result. If the two hashes match, then Alice can be sure about the integrity and author of the message.

Before moving on, let's briefly compare digital signatures with MACs, since they have parallels, but also have important subtle differences. Both digital signatures and MACs start with a message (or a document). To create a MAC out of the message, we append an authentication key to the message, and then take the hash of the result. Note that neither public key nor symmetric key encryption is involved in creating the MAC. To create a digital signature, we first take the hash of the message and then encrypt the message with our private key (using public key cryptography).

Thus, a digital signature is a "heavier" technique, since it requires an underlying Public Key Infrastructure (PKI) with certification authorities as described below. We'll see in Section 8.4 that PGP—a popular secure e-mail system—uses digital signatures for message integrity. We've seen already that OSPF uses MACs for message integrity. We'll see in Sections 8.5 and 8.6 that MACs are also used for popular transport-layer and network-layer security protocols.

Public Key Certification

An important application of digital signatures is **public key certification**, that is, certifying that a public key belongs to a specific entity. Public key certification is used in many popular secure networking protocols, including IPsec and SSL.

To gain insight into this problem, let's consider an Internet-commerce version of the classic "pizza prank." Alice is in the pizza delivery business and accepts orders over the Internet. Bob, a pizza lover, sends Alice a plaintext message that includes his home address and the type of pizza he wants. In this message, Bob also includes a digital signature (that is, a signed hash of the original plaintext message) to prove to Alice that he is the true source of the message. To verify the signature, Alice obtains Bob's public key (perhaps from a public key server or from the e-mail message) and checks the digital signature. In this manner she makes sure that Bob, rather than some adolescent prankster, placed the order.

This all sounds fine until clever Trudy comes along. As shown in Figure 8.13, Trudy is indulging in a prank. She sends a message to Alice in which she says she is Bob, gives Bob's home address, and orders a pizza. In this message she also includes her (Trudy's) public key, although Alice naturally assumes it is Bob's public key. Trudy also attaches a digital signature, which was created with her own (Trudy's) private key. After receiving the message, Alice applies Trudy's public key (thinking that it is Bob's) to the digital signature and concludes that the plaintext message was indeed created by Bob. Bob will be very surprised when the delivery person brings a pizza with pepperoni and anchovies to his home!

We see from this example that for public key cryptography to be useful, you need to be able to verify that you have the actual public key of the entity (person, router, browser, and so on) with whom you want to communicate. For example, when Alice wants to communicate with Bob using public key cryptography, she needs to verify that the public key that is supposed to be Bob's is indeed Bob's.

Binding a public key to a particular entity is typically done by a **Certification Authority (CA)**, whose job is to validate identities and issue certificates. A CA has the following roles:

1. A CA verifies that an entity (a person, a router, and so on) is who it says it is. There are no mandated procedures for how certification is done. When dealing with a CA, one must trust the CA to have performed a suitably rigorous identity verification. For example, if Trudy were able to walk into the Fly-by-Night CA

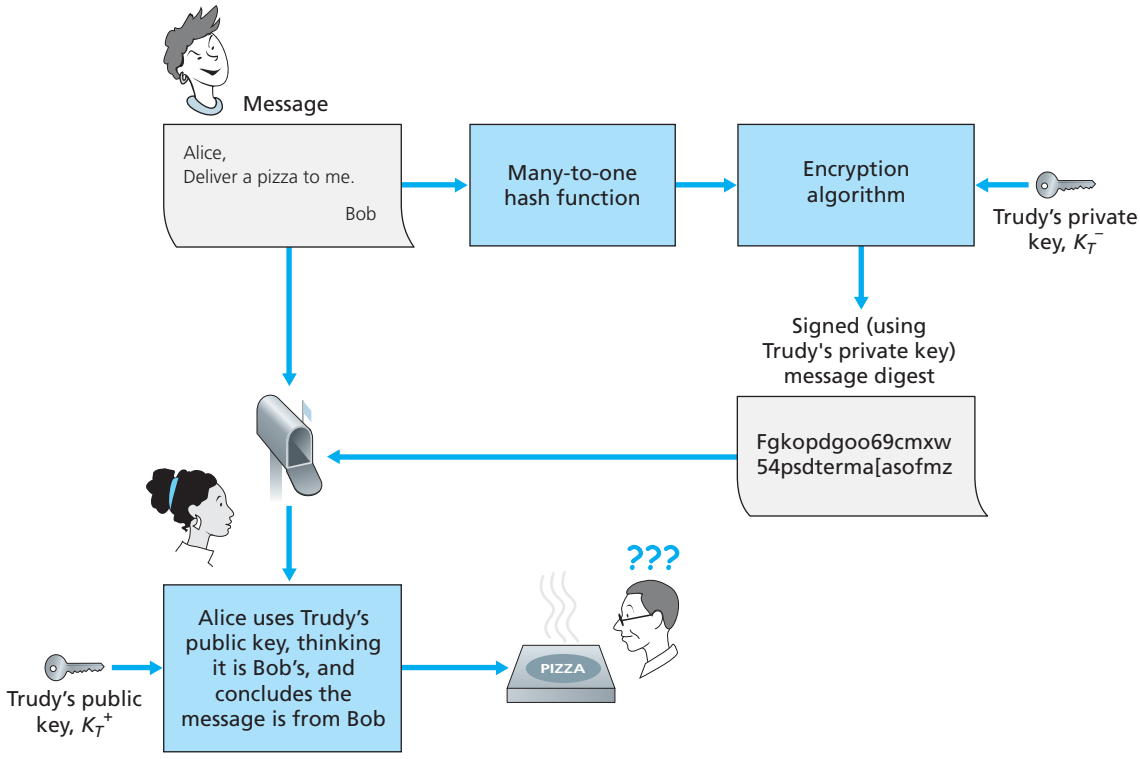

Figure 8.13 ♦ Trudy masquerades as Bob using public key cryptography

and simply announce "I am Alice" and receive certificates associated with the identity of Alice, then one shouldn't put much faith in public keys certified by the Fly-by-Night CA. On the other hand, one might (or might not!) be more willing to trust a CA that is part of a federal or state program. You can trust the identity associated with a public key only to the extent to which you can trust a CA and its identity verification techniques. What a tangled web of trust we spin!

2. Once the CA verifies the identity of the entity, the CA creates a **certificate** that binds the public key of the entity to the identity. The certificate contains the public key and globally unique identifying information about the owner of the public key (for example, a human name or an IP address). The certificate is digitally signed by the CA. These steps are shown in Figure 8.14.

Let us now see how certificates can be used to combat pizza-ordering pranksters, like Trudy, and other undesirables. When Bob places his order he also sends his CA-signed certificate. Alice uses the CA's public key to check the validity of Bob's certificate and extract Bob's public key.

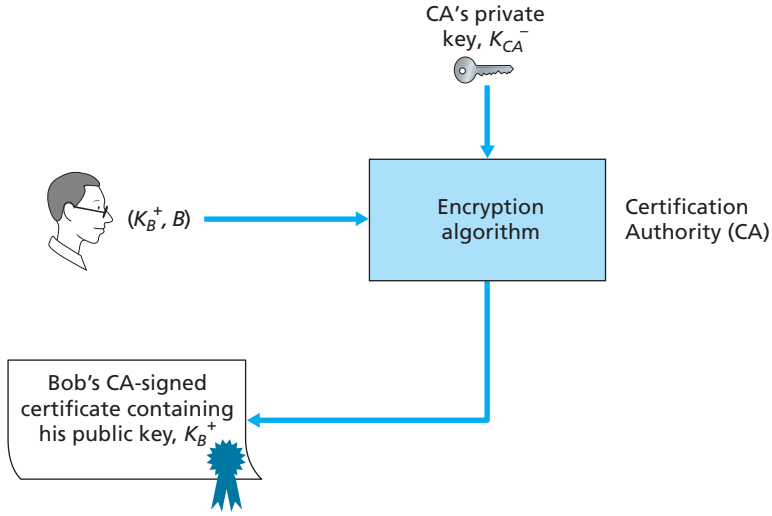

Figure 8.14 ♦ Bob has his public key certified by the CA

Both the International Telecommunication Union (ITU) and the IETF have developed standards for CAs. ITU X.509 [ITU 2005a] specifies an authentication service as well as a specific syntax for certificates. [RFC 1422] describes CA-based key management for use with secure Internet e-mail. It is compatible with X.509 but goes beyond X.509 by establishing procedures and conventions for a key management architecture. Table 8.4 describes some of the important fields in a certificate.

Field Name	Description
Version	Version number of X.509 specification
Serial number	CA-issued unique identifier for a certificate
Signature	Specifies the algorithm used by CA to sign this certificate
Issuer name	Identity of CA issuing this certificate, in distinguished name (DN) [RFC 4514] format
Validity period	Start and end of period of validity for certificate
Subject name	Identity of entity whose public key is associated with this certificate, in DN format
Subject public key	The subject's public key as well indication of the public key algorithm (and algorithm parameters) to be used with this key

Table 8.4 ♦ Selected fields in an X.509 and RFC 1422 public key