Since passwords are so widely used, we might suspect that protocol *ap3.0* is fairly secure. If so, we'd be wrong! The security flaw here is clear. If Trudy eavesdrops on Alice's communication, then she can learn Alice's password. Lest you think this is unlikely, consider the fact that when you Telnet to another machine and log in, the login password is sent unencrypted to the Telnet server. Someone connected to the Telnet client or server's LAN can possibly sniff (read and store) all packets transmitted on the LAN and thus steal the login password. In fact, this is a well-known approach for stealing passwords (see, for example, [Jimenez 1997]). Such a threat is obviously very real, so *ap3.0* clearly won't do.

8.4.4 Authentication Protocol *ap3.1*

Our next idea for fixing *ap3.0* is naturally to encrypt the password. By encrypting the password, we can prevent Trudy from learning Alice's password. If we assume that Alice and Bob share a symmetric secret key, K_{A-B}, then Alice can encrypt the password and send her identification message, "`I am Alice`," and her encrypted password to Bob. Bob then decrypts the password and, assuming the password is correct, authenticates Alice. Bob feels comfortable in authenticating Alice since Alice not only knows the password, but also knows the shared secret key value needed to encrypt the password. Let's call this protocol *ap3.1*.

While it is true that *ap3.1* prevents Trudy from learning Alice's password, the use of cryptography here does not solve the authentication problem. Bob is subject to a **playback attack**: Trudy need only eavesdrop on Alice's communication, record the encrypted version of the password, and play back the encrypted version of the password to Bob to pretend that she is Alice. The use of an encrypted password in *ap3.1* doesn't make the situation manifestly different from that of protocol *ap3.0* in Figure 8.17.

8.4.5 Authentication Protocol *ap4.0*

The failure scenario in Figure 8.17 resulted from the fact that Bob could not distinguish between the original authentication of Alice and the later playback of Alice's original authentication. That is, Bob could not tell if Alice was live (that is, was currently really on the other end of the connection) or whether the messages he was receiving were a recorded playback of a previous authentication of Alice. The very *(very)* observant reader will recall that the three-way TCP handshake protocol needed to address the same problem—the server side of a TCP connection did not want to accept a connection if the received SYN segment was an old copy (retransmission) of a SYN segment from an earlier connection. How

did the TCP server side solve the problem of determining whether the client was really live? It chose an initial sequence number that had not been used in a very long time, sent that number to the client, and then waited for the client to respond with an ACK segment containing that number. We can adopt the same idea here for authentication purposes.

A **nonce** is a number that a protocol will use only once in a lifetime. That is, once a protocol uses a nonce, it will never use that number again. Our *ap4.0* protocol uses a nonce as follows:

1. Alice sends the message "`I am Alice`" to Bob.
2. Bob chooses a nonce, R, and sends it to Alice.
3. Alice encrypts the nonce using Alice and Bob's symmetric secret key, K_{A-B}, and sends the encrypted nonce, $K_{A-B}(\mathrm{R})$, back to Bob. As in protocol *ap3.1*, it is the fact that Alice knows K_{A-B} and uses it to encrypt a value that lets Bob know that the message he receives was generated by Alice. The nonce is used to ensure that Alice is live.
4. Bob decrypts the received message. If the decrypted nonce equals the nonce he sent Alice, then Alice is authenticated.

Protocol *ap4.0* is illustrated in Figure 8.18. By using the once-in-a-lifetime value, *R,* and then checking the returned value, $K_{A-B}(R)$, Bob can be sure that Alice is both who she says she is (since she knows the secret key value needed to encrypt *R)* and live (since she has encrypted the nonce, *R,* that Bob just created).

The use of a nonce and symmetric key cryptography forms the basis of *ap4.0*. A natural question is whether we can use a nonce and public key cryptography

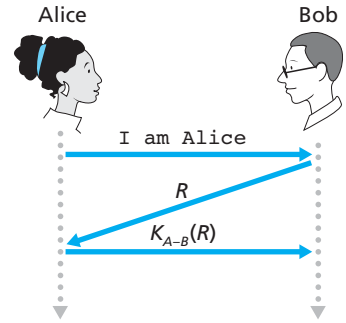

Figure 8.18 ♦ Protocol *ap4.0* and a failure scenario

(rather than symmetric key cryptography) to solve the authentication problem. This issue is explored in the problems at the end of the chapter.

8.5 Securing E-Mail

In previous sections, we examined fundamental issues in network security, including symmetric key and public key cryptography, end-point authentication, key distribution, message integrity, and digital signatures. We are now going to examine how these tools are being used to provide security in the Internet.

Interestingly, it is possible to provide security services in any of the top four layers of the Internet protocol stack. When security is provided for a specific application-layer protocol, the application using the protocol will enjoy one or more security services, such as confidentiality, authentication, or integrity. When security is provided for a transport-layer protocol, all applications that use that protocol enjoy the security services of the transport protocol. When security is provided at the network layer on a host-to-host basis, all transport-layer segments (and hence all application-layer data) enjoy the security services of the network layer. When security is provided on a link basis, then the data in all frames traveling over the link receive the security services of the link.

In Sections 8.5 through 8.8, we examine how security tools are being used in the application, transport, network, and link layers. Being consistent with the general structure of this book, we begin at the top of the protocol stack and discuss security at the application layer. Our approach is to use a specific application, e-mail, as a case study for application-layer security. We then move down the protocol stack. We'll examine the SSL protocol (which provides security at the transport layer), IPsec (which provides security at the network layer), and the security of the IEEE 802.11 wireless LAN protocol.

You might be wondering why security functionality is being provided at more than one layer in the Internet. Wouldn't it suffice simply to provide the security functionality at the network layer and be done with it? There are two answers to this question. First, although security at the network layer can offer "blanket coverage" by encrypting all the data in the datagrams (that is, all the transport-layer segments) and by authenticating all the source IP addresses, it can't provide user-level security. For example, a commerce site cannot rely on IP-layer security to authenticate a customer who is purchasing goods at the commerce site. Thus, there is a need for security functionality at higher layers as well as blanket coverage at lower layers. Second, it is generally easier to deploy new Internet services, including security services, at the higher layers of the protocol stack. While waiting for security to be broadly deployed at the network layer, which is probably still many years in the future, many application developers