In the first message in Figure 10.1, Alice identifies herself and she sends information regarding the crypto parameters that she prefers (crypto algorithms, key lengths, etc.), along with her nonce, R_A. In message two, Bob selects from Alice's crypto parameters and returns his selections, along with his nonce, R_B. In message three, Alice sends her Diffie-Hellman value, and in message four, Bob responds with his Diffie-Hellman value, his certificate, and S_B, which consists of a signed hash value. At this point, Alice is able to compute the key K, and in the final message, she sends an encrypted block that contains her identity, her certificate, and her signed value S_A.

In Figure 10.1, the signatures are intended to provide mutual authentication. Note that the nonce R_A is Alice's challenge to Bob, and S_B is Bob's response. That is, the nonce R_A provides replay protection, and only Bob can give the correct response since a signature is required (assuming, of course, that his private key has not been compromised). A similar argument shows that Alice is authenticated in the final message. So, SSH provides mutual authentication. The security of SSH authentication, the security of the key K, and some other quirks of SSH are considered further in the homework problems at the end of this chapter.

10.3 SSL

The mythical "socket layer" lives between the application layer and the transport layer in the Internet protocol stack, as illustrated in Figure 10.2. In practice, SSL most often deals with Web browsing, in which case the application layer protocol is HTTP and the transport layer protocol is TCP.

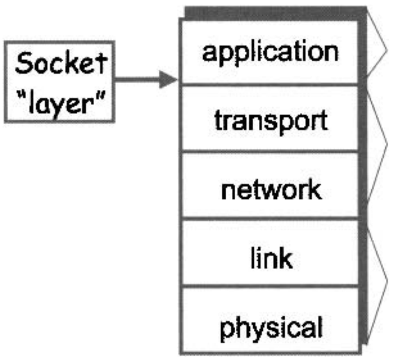

Figure 10.2: Socket Layer

SSL is the protocol of choice for the vast majority of secure transactions over the Internet. For example, suppose that you want to buy a book at amazon.com. Before you provide your credit card information, you want to be sure you are dealing with Amazon, that is, you must authenticate Amazon. Generally, Amazon doesn't care who you are, as long as you have money. As a result, the authentication need not be mutual.

After you are satisfied that you are dealing with Amazon, you will provide private information, such as your credit card number, your address, and so on. You probably want this information protected in transit—in most cases, you want both confidentiality (to protect your privacy) and integrity protection (to assure the transaction is received correctly).

The general idea behind SSL is illustrated in Figure 10.3. In this protocol, Alice (the client) informs Bob (the server) that she wants to conduct a secure transaction. Bob responds with his certificate. Alice then needs to verify the signature on the certificate. Assuming the signature verifies, Alice will be confident that she has Bob's certificate, although she cannot yet be certain that she's talking to Bob. Then Alice will encrypt a symmetric key K_{AB} with Bob's public key and send the encrypted key to Bob. This symmetric key is used to encrypt and integrity protect subsequent communications.

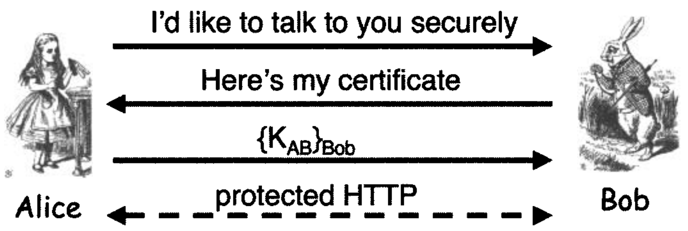

Figure 10.3: Too-Simple Protocol

The protocol in Figure 10.3 is not useful as it stands. For one thing, Bob is not explicitly authenticated and the only way Alice could possibly know she is talking to Bob is by checking to see that the encrypted data decrypts correctly. This is not a desirable situation in any security protocol. Also note that Alice is not authenticated to Bob at all, but in most cases, this is reasonable for transactions on the Internet.

In Figure 10.4, we've given a reasonably complete view of the basic SSL protocol. In this protocol,

$$\begin{aligned} S &= \text{the pre-master secret} \\ K &= h(S, R_A, R_B) \\ \text{msgs} &= \text{shorthand for "all previous messages"} \\ \text{CLNT} &= \text{literal string} \\ \text{SRVR} &= \text{literal string} \end{aligned}$$

where h is a secure hash function. The actual SSL protocol is more complex than what appears in Figure 10.4 but this simplified version is sufficient for our purposes. The complete SSL specification can be found at [271].

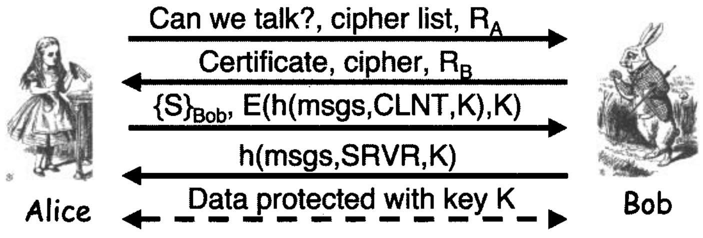
Can we talk?, cipher list, R_A
Certificate, cipher, R_B
{S}_Bob, E(h(msgs,CLNT,K),K)
h(msgs,SRVR,K)
Data protected with key K
Alice
Bob

Figure 10.4: Simplified SSL

Next, we briefly discuss each message in the simplified SSL protocol given in Figure 10.4. In the first message, Alice informs Bob that she would like to establish an SSL connection, and she passes a list of ciphers that she supports, along with a nonce R_A. In the second message, Bob responds with his certificate, he selects one of the ciphers from the cipher list that Alice sent in message one, and he sends a nonce R_B.

In the third message, Alice sends the so-called pre-master secret S, which she randomly generated, along with a hash that is encrypted with the key K. In this hash, "msgs" includes all previous messages and CLNT is a literal string.[2] The hash is used as an integrity check to verify that the previous messages have been received correctly.

In the fourth message, Bob responds with a similar hash. By computing this hash herself, Alice can verify that Bob received the messages correctly, and she can authenticate Bob, since only Bob could have decrypted S, which is required to generate the key K. At this point, Alice has authenticated Bob, and Alice and Bob have established a shared session key K, which they can use to encrypt and integrity protect subsequent messages.

In reality, more than one key is derived from the hash $h(S, R_A, R_B)$. In fact, the following six quantities are generated from this hash.

- Two encryption keys, one for messages sent from the client to server, and one for messages sent from the server to the client.
- Two integrity keys, used in the same way as the encryption keys.
- Two initialization vectors (IVs), one for the client and one for the server.

[2]In this context, "msg" has nothing to do with the list of ingredients at a Chinese restaurant.

In short, different keys are used in each direction. This could help to prevent certain types of attacks where Trudy tricks Bob into doing something that Alice should have done, or vice versa.

The attentive reader may wonder why $h(\text{msgs}, \text{CLNT}, K)$ is encrypted in messages three and four. In fact, this adds no security, although it does add extra work, so it could be considered a minor flaw in the protocol.

In the SSL protocol of Figure 10.4, Alice, the client, authenticates Bob, the server, but not vice versa. With SSL, it is possible for the server to authenticate the client. If this is desired, Bob sends a "certificate request" in message two. However, this feature is generally not used, particularly in e-commerce situations, since it requires users to have valid certificates. If the server wants to authenticate the client, the server could instead require that the client enter a valid password, in which case the resulting authentication is outside the scope of the SSL protocol.

10.3.1 SSL and the Man-in-the-Middle

Hopefully, SSL prevents the man-in-the-middle, or MiM, attack illustrated in Figure 10.5. But what mechanism in SSL prevents this attack? Recall that Bob's certificate must be signed by a certificate authority. If Trudy sends her own certificate instead of Bob's, the attack will fail when Alice attempts to verify the signature on the certificate. Or, Trudy could make a bogus certificate that says "Bob," keep the private key for herself, and sign the certificate herself. Again, this will not pass muster when Alice tries to verify the signature on "Bob's" certificate (which is really Trudy's certificate). Finally, Trudy could simply send Bob's certificate to Alice, and Alice would verify the signature on this certificate. However, this is not an attack since it would not break the protocol—Alice would authenticate Bob, and Trudy would be left out in the cold.

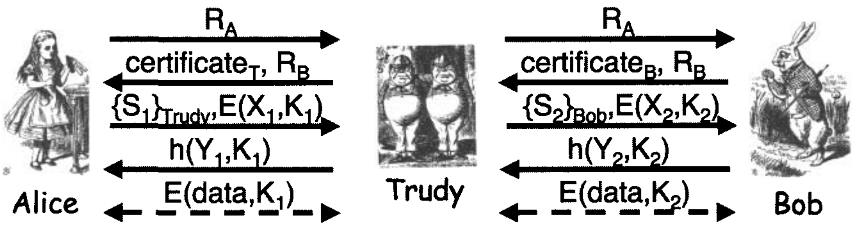

Figure 10.5: Man-in-the-Middle Attack on SSL

However, the real world is not so kind to poor Alice. Typically, SSL is used in a Web browsing session. Then, what happens when Trudy attempts a MiM attack by sending a bogus certificate to Alice? The signature on the certificate is not valid so the attack should fail. However, Alice does not personally check the signature on the certificate—her browser does. And what does Alice's browser do when it detects a problem with a certificate? As you probably know from experience, the browser provides Alice with a warning. Does Alice heed the warning? If she's like most users, Alice ignores the warning and allows the connection to proceed.[3] Note that when Alice ignores this warning, she's opened the door to the MiM attack in Figure 10.5. Finally, it's important to realize that while this attack is a very real threat, it's not due to a flaw in the SSL protocol. Instead, it's caused by a flaw in human nature, making a patch much more problematic.

10.3.2 SSL Connections

An SSL *session* is established as shown in Figure 10.4. This session establishment protocol is relatively expensive, since public key operations are involved.

SSL was originally developed by Netscape, specifically for use in Web browsing. The application layer protocol for the Web is HTTP, and two versions of it are in common usage, HTTP 1.0 and HTTP 1.1. With version 1.0, it is not uncommon for a Web browser to open multiple parallel connections so as to improve performance. Due to the public key operations, there would be significant overhead if a new SSL session was established for each of these HTTP connections. The designers of SSL were aware of this issue, so they included an efficient protocol for opening new SSL *connections* provided that an SSL session already exists. The idea is simple—after establishing one SSL session, Alice and Bob share a session key K, which can then be used to establish new connections, thereby avoiding expensive public key operations.

The SSL connection protocol appears in Figure 10.6. The protocol is similar to the SSL session establishment protocol, except that the previously established session key K is used instead of the public key operation that are used in the session protocol.

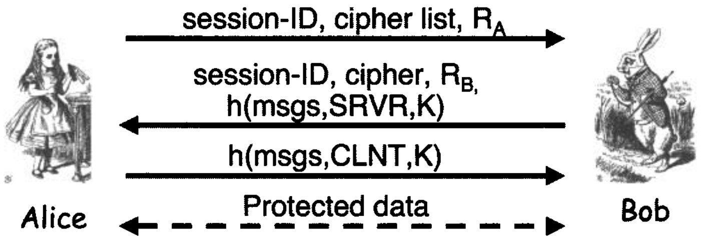
session-ID, cipher list, R_A
session-ID, cipher, R_B, h(msgs,SRVR,K)
h(msgs,CLNT,K)
Protected data
Alice
Bob

Figure 10.6: SSL Connection Protocol

[3]If possible, Alice would probably disable the warning so that she'd never get this annoying "error" message again.

The bottom line here is that in SSL, one (expensive) session is required, but then we can create any number of (cheap) connections. This is a useful feature that was designed to improve the performance of the protocol when used with HTTP 1.1.

10.3.3 SSL Versus IPSec

In the next section, we'll discuss IPSec, which is short for Internet Protocol Security. The purpose of IPSec is similar to that of SSL, namely, security over the network. However, the implementation of the two protocols is very different. For one thing, SSL is relatively simple, while IPSec is relatively complex.

It might seem logical to discuss IPSec in detail before contrasting it with SSL. However, we might get so lost in the weeds with IPSec that we'd completely lose sight of SSL. So instead of waiting until after we discuss IPSec to contrast the two protocols, we'll do so beforehand. You might consider this a partial preview of IPSec.

The most obvious difference between SSL and IPSec is that the two protocols operate at different layers of the protocol stack. SSL (and its twin,[4] the IEEE standard known as TLS), both live at the socket layer. As a result, SSL resides in user space. IPSec, on the other hand, lives at the network layer and is therefore not directly accessible from user space—it's in the domain of the operating system. When viewed from a high level, this is the fundamental distinction between SSL and IPSec.

Both SSL and IPSec provide encryption, integrity protection, and authentication. SSL is relatively simple and well designed, whereas IPSec is complex and, as a result, includes some significant flaws.

Since IPSec is part of the OS, it must be built-in at that level. In contrast, SSL is part of user space, so it requires nothing special of the OS. IPSec also requires no changes to applications, since all of the security magically happens at the network layer. On the other hand, developers have to make a conscious decision to use SSL.

SSL was built for Web application early on, and its primary use remains secure Web transactions. IPSec is often used to secure a virtual private network, or VPN, an application that creates a secure tunnel between the endpoints. Also, IPSec is required in IP version 6 (IPv6), so if IPv6 ever takes over the world, IPSec will be ubiquitous.

There is, understandably, a reluctance to retrofit applications for SSL. There is, also understandably, a reluctance to use IPSec due to its complexity (which creates some challenging implementation issues). The net result is that the Net is less secure than it should be.

[4]They are fraternal twins, not identical twins.