31.2 TRANSPORT LAYER SECURITY

Transport Layer Security (TLS) was designed to provide security at the transport layer. TLS was derived from a security protocol called Secure Sockets Layer (SSL), designed by Netscape to provide security on the WWW. TLS is a nonproprietary version of SSL designed by IETF. For transactions on the Internet, a browser needs the following:

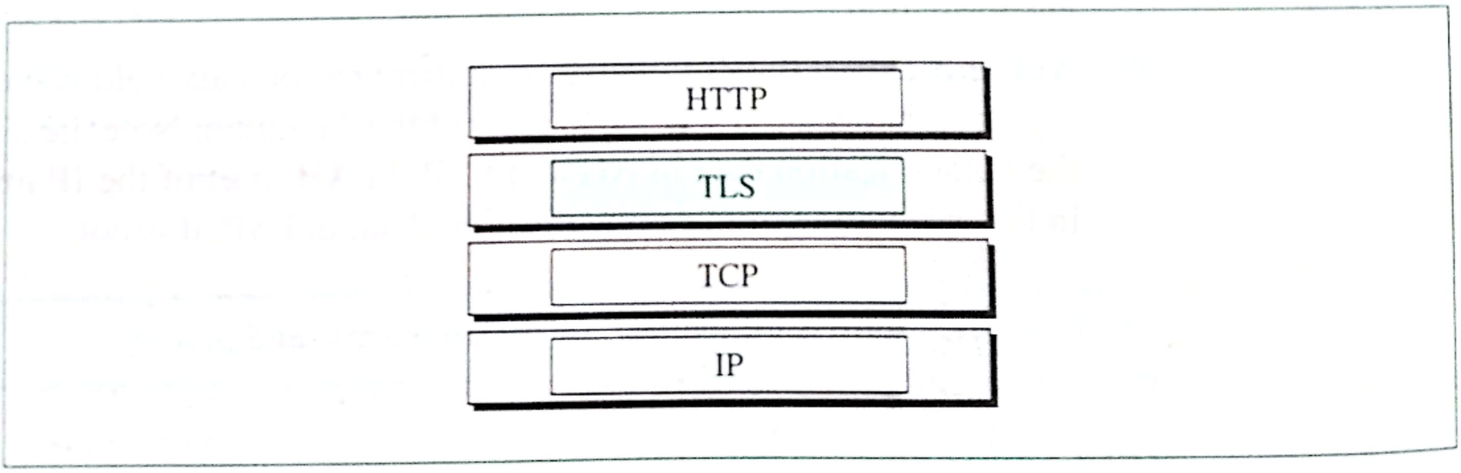
1. The customer needs to be sure that the server belongs to the actual vendor, not an imposter. For example, a customer does not want to give an imposter her credit card number. In other words, the server must be authenticated.
2. The customer needs to be sure that the contents of the message are not modified during transition. A bill for $100 must not be changed to $1000. The integrity of the message must be preserved.
3. The customer needs to be sure that an imposter does not intercept sensitive information such as a credit card number. There is a need for privacy.

There are other optional security aspects that can be added to the above list. For example, the vendor may need to authenticate the customer. TLS can provide additional features to cover these aspects of security.

Position of TLS

TLS lies between the application layer and the transport layer (TCP), as shown in Figure 31.5.

Figure 31.5 *Position of TLS*



The application layer protocol, in this case HTTP, uses the services of TLS, and TLS uses the services of the transport layer.

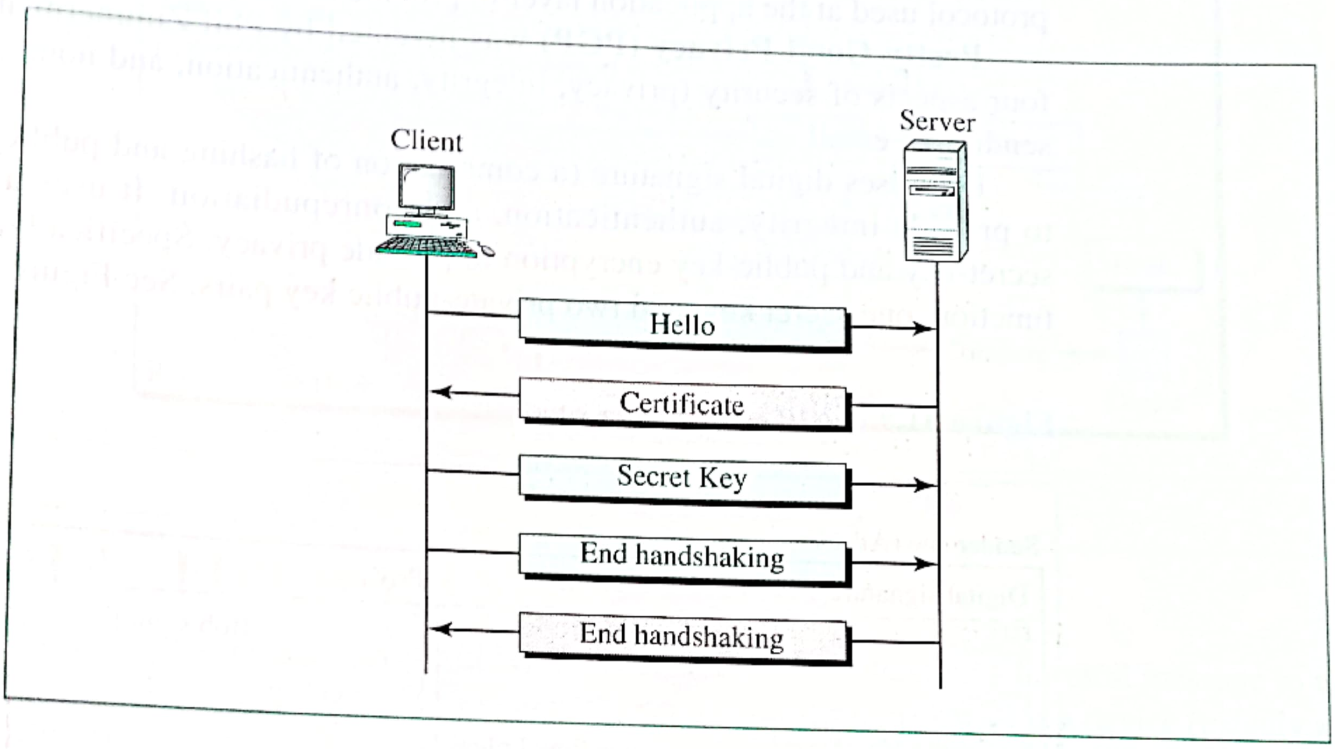
Two Protocols

TLS is actually two protocols: the handshake protocol and the data exchange (sometimes called the record) protocol.

Handshake Protocol

The **handshake protocol** is responsible for negotiating security, authenticating the server to the browser, and (optionally) defining other communication parameters. The handshake protocol defines the exchange of a series of messages between the browser and server. We discuss a simplified version, as shown in Figure 31.6.

Figure 31.6 *Handshake protocol*



1. The browser sends a *hello* message that includes the TLS version and some preferences.
2. The server sends a *certificate* message that includes the public key of the server. The public key is certified by some certification authority, which means that the public key is encrypted by a CA private key. The browser has a list of CAs and their public keys. It uses the corresponding key to decrypt the certificate and finds the server public key. This also authenticates the server because the public key is certified by the CA.
3. The browser generates a secret key, encrypts it with the server public key, and sends it to the server.
4. The browser sends a message, encrypted by the secret key, to inform the server that handshaking is terminating from the browser side.
5. The server decrypts the secret key using its private key and decrypts the message using the secret key. It then sends a message, encrypted by the secret key, to inform the browser that handshaking is terminating from the server side.

Note that handshaking uses the public key for two purposes: to authenticate the server and to encrypt the secret key, which is used in the data exchange protocol.

Data Exchange Protocol

The **data exchange** (record) **protocol** uses the secret key to encrypt the data for secrecy and to encrypt the message digest for integrity. The details and specification of algorithms are agreed upon during the handshake phase.