# Secure Sockets Layer (SSL)

SSL is a security protocol that was developed by Netscape Communications Corporation, along with RSA Data Security, Inc. The primary goal of the SSL protocol is to provide a private channel between communicating applications, which ensures privacy of data, authentication of the partners and integrity.

## SSL Overview

SSL provides an alternative to the standard TCP/IP socket API that has security implemented within it. Hence, in theory it is possible to run any TCP/IP application in a secure way without changing the application. In practice, SSL is only widely implemented for HTTP connections, but Netscape Communications Corp. has stated an intention to employ it for other application types, such as NNTP and Telnet, and there are several such implementations freely available on the Internet. IBM, for example, is using SSL to enhance security for TN3270 sessions in its Host On-Demand and eNetwork Communications Server products.

SSL is composed of two layers:

- At the lower layer, a protocol for transferring data using a variety of predefined cipher and authentication combinations, called the *SSL Record Protocol*. Figure 202 on page 332 illustrates this, and contrasts it with a standard HTTP socket connection. Note that this diagram shows SSL as providing a simple socket interface, on which other applications can be layered. In reality, current implementations have the socket interface embedded within the application and do not expose an API that other applications can use.

- On the upper layer, a protocol for initial authentication and transfer of encryption keys, called the *SSL Handshake Protocol*.

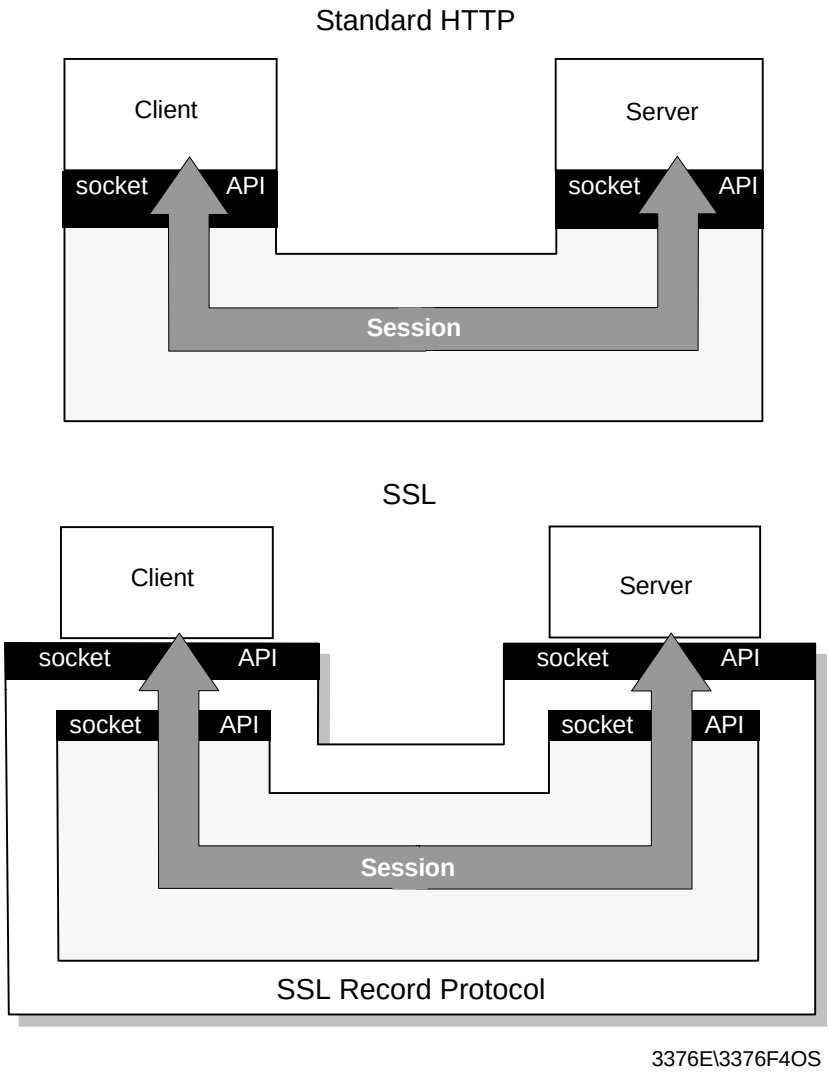


*Figure 202. SSL - Comparison of Standard and SSL Sessions*

An SSL session is initiated as follows:

- On the client (browser) the user requests a document with a special URL that commences https: instead of http:, either by typing it into the URL input field, or by clicking on a link.
- The client code recognizes the SSL request and establishes a connection through TCP port 443 to the SSL code on the server.
- The client then initiates the SSL handshake phase, using the SSL Record Protocol as a carrier. At this point there is no encryption or integrity checking built in to the connection.

The SSL protocol addresses the following security issues:

***Privacy***

After the symmetric key is established in the initial handshake, the messages are encrypted using this key.

***Integrity***

Messages contain a message authentication code (MAC) ensuring the message integrity.

***Authentication***

During the handshake, the client authenticates the server using an asymmetric or public key. It can also be based on certificates.

SSL requires each message to be encrypted and decrypted and therefore has a high performance and resource overhead.

### Differences between SSL V2.0 and SSL V3.0

There is a backward compatibility between SSL V2.0 and SSL V3.0. An SSL V3.0 server implementation should be able accept the connection request from an SSL V2.0 client. The main differences between SSL V2.0 and SSL V3.0 are as follows:

- SSL V2.0 does not support client authentication.
- SSL V3.0 supports more ciphering types in the CipherSpec.

## SSL Protocol

The SSL protocol is located at the top of the transport layer. SSL is also a layered protocol itself. It simply takes the data from the application layer, reformats it and transmits it to the transport layer. SSL handles a message as follows:

***Sender***

Performs the following tasks:

- Takes the message from upper layer
- Fragments the data to manageable blocks
- Optionally compresses the data
- Applies a Message Authentication Code (MAC)
- Encrypts the data
- Transmits the result to the lower layer

***Receiver***

Performs the following tasks:

- Takes the data from lower layer
- Decrypts
- Verifies the data with the negotiated MAC key
- Decompresses the data if compression was used
- Reassembles the message
- Transmits the message to the upper layer

An SSL session works in different states. These states are *session* and *connection* states. The SSL handshake protocol (please see 5.7.2.2, “SSL Handshake Protocol” on page 335) coordinates the states of the client and the server. In addition, there are read and write states defined to coordinate the encryption according to the change cipher spec messages.

When either party sends a change cipher spec message, it changes the pending write state to current write state. Again, when either party receives a change cipher spec message, it changes the pending read state to the current read state.

The session state includes the following components:

***Session identifier***
An arbitrary byte sequence chosen by the server to identify an active or resumable session state.

***Peer certificate***
Certificate of the peer. This field is optional; it can be empty.

***Compression method***
The compression algorithm.

***Cipher spec***
Specifies data encryption algorithm (such as null, DES) and a MAC algorithm.

***Master secret***
48-byte shared secret between the client and the server.

***Is resumable***
A flag indicating whether the session can be used for new connections.

The connection state includes the following components:

***Server and client random***
An arbitrary byte sequence chosen by the client and server for each connection.

***Server write MAC secret***
The secret used for MAC operations by the server.

***Client write MAC secret***
The secret used for MAC operations by the client.

***Server write key***
The cipher key for the server to encrypt the data and the client to decrypt the data.

***Client write key***
The cipher key for the client to encrypt the data and the server to decrypt the data.

***Initialization vectors***
Initialization vectors store the encryption information.

***Sequence numbers***
A sequence number indicates the number of the message transmitted since the last change cipher spec message. Both the client and the server maintain sequence numbers.

## Change Cipher Spec Protocol

The change cipher spec protocol is responsible for sending change cipher spec messages. At any time, the client can request to change current cryptographic parameters such as handshake key exchange. Following the change cipher spec notification, the client sends a handshake key exchange and if available, certificate verify messages, and the server sends a change cipher spec message after processing the key exchange message. After that, the newly agreed keys will be used until the next change cipher spec request. The change cipher spec message is sent after the hello messages during the negotiation.

## SSL Handshake Protocol

The SSL Handshake Protocol allows the client and server to determine the required parameters for an SSL connection such as protocol version, cryptographic algorithms, optional client or server authentication, and public-key encryption methods to generate shared secrets. During this process all handshake messages are forwarded to the SSL Record Layer to be encapsulated into special SSL messages. Figure 203 illustrates an SSL handshake process.

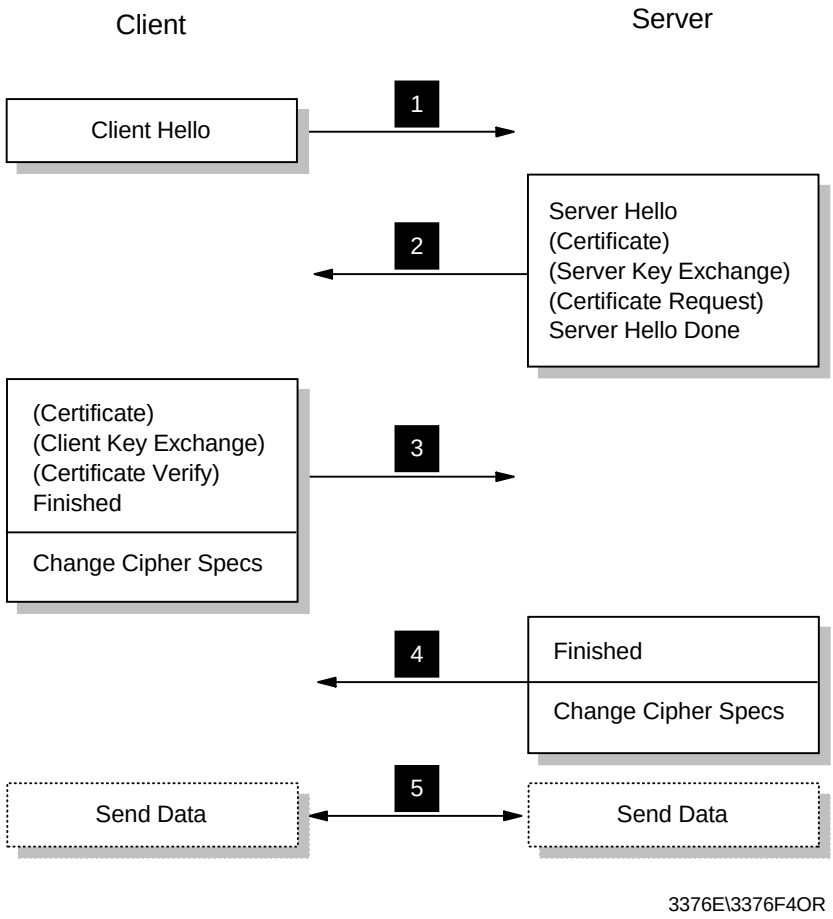


*Figure 203. SSL - Handshake Process*

1. The client sends a connection request with a client hello message. This message includes:
   - Desired version number.
   - Time information (the current time and date in standard UNIX 32-bit format).
   - Optionally session-ID. If it is not specified the server will try to resume previous sessions or return an error message
   - Cipher suites. (List of the cryptographic options supported by the client. These are authentication modes, key exchange methods, encryptions and MAC algorithms.)
   - Compression methods supported by the client.
   - A random value.
2. The server evaluates the parameters sent by the client hello message and returns a server hello message that includes the following parameters which were selected by the server to be used for the SSL session:

- Version number
- Time information (the current time and date in standard UNIX 32-bit format)
- Session ID
- Cipher suite
- Compression method
- A random value

Following the server hello message the server sends the following messages:

- Server certificate if the server is required to be authenticated
- A server key exchange message if there is no certificate available or the certificate is for signing only
- A certificate request if the client is required to be authenticated

Finally, the server sends a server hello done message and begins to wait for the client response.

3. The client sends the following messages:

   - If the server has sent a certificate request, the client must send a certificate or a no certificate message.
   - If the server has sent a server key exchange message, the client sends a client key exchange message based on the public key algorithm determined with the hello messages.
   - If the client has sent a certificate, the client verifies the server certificate and sends a certificate verify message indicating the result.

   The client then sends a finished message indicating the negotiation part is completed. The client also sends a change cipher spec message to generate shared secrets. It should be noted that this is not controlled by the handshake protocol, the change cipher spec protocol manages this part of the operation.

4. The server sends a finished message indicating the negotiation part is completed. The server then sends the change cipher spec message.

5. Finally the session partners separately generate an encryption key, the master key from which they derive the keys to use in the encrypted session that follows. The Handshake protocol changes the state to the connection state. All data taken from the application layer is transmitted as special messages to the other party.

There is significant additional overhead in starting up an SSL session compared with a normal HTTP connection. The protocol avoids some of this overhead by allowing the client and server to retain session key information and to resume that session without negotiating and authenticating a second time.

Following the handshake, both session partners have generated a master key. From that key they generate other session keys, which are used in the symmetric-key encryption of the session data and in the creation of message digests. The first message encrypted in this way is the finished message from the server. If the client can interpret the finished message, it means:

- Privacy has been achieved, because the message is encrypted using a symmetric-key bulk cipher (such as DES or RC4).
- The message integrity is assured, because it contains a Message Authentication Code (MAC), which is a message digest of the message itself plus material derived from the master key.
- The server has been authenticated, because it was able to derive the master key from the pre-master key. As this was sent using the server's public key, it

could only have been decrypted by the server (using its private key). Note that this relies on the integrity of the server's public key certificate.

### SSL Record Protocol

Once the master key has been determined, the client and server can use it to encrypt application data. The SSL record protocol specifies a format for these messages. In general they include a message digest to ensure that they have not been altered and the whole message is encrypted using a symmetric cipher. Usually this uses the RC2 or RC4 algorithm, although DES, triple-DES and IDEA are also supported by the specification.

The U.S. National Security Agency (NSA), a department of the United States federal government imposes restrictions on the size of the encryption key that can be used in software exported outside the U.S. These rules are currently under review, but the present effect is to limit the key to an effective size of 56 bits. The RC2 and RC4 algorithms achieve this by using a key in which all but 56 bits are set to a fixed value. International (export) versions of software products have this hobbled security built into them. SSL checks for mismatches between the export and nonexport versions in the negotiation phase of the handshake. For example, if a U.S. browser tries to connect with SSL to an export server, they will agree on export-strength encryption. See 5.2.7, “Export/Import Restrictions on Cryptography” on page 279 for more information on recent changes of U.S. export regulations of cryptographic material.

## Transport Layer Security (TLS)

The Transport Layer Security 1.0 protocol is based on SSL. At the time of writing, the TLS 1.0 protocol is not a standard protocol. (Please refer to current TLS draft document for more information about SSL.) There are not significant differences between SSL 3.0 and TLS 1.0. They can interoperate with some modifications on the message formats.

## Secure Multipurpose Internet Mail Extension (S-MIME)

Secure Multipurpose Internet Mail Extension (S-MIME) can be thought of as a very specific SSL-like protocol. S-MIME is an application-level security construct, but its use is limited to protecting e-mail via encryption and digital signatures. It relies on public key technology, and uses X.509 certificates to establish the identities of the communicating parties. S-MIME can be implemented in the communicating end systems; it is not used by intermediate routers or firewalls.