Kipp E.B. Hickman
Internet Draft   Netscape Communications Corp
April 1995 (Expires 10/95)

The SSL Protocol
<draft-hickman-netscape-ssl-00.txt>

1. STATUS OF THIS MEMO

2. ABSTRACT

This document specifies the Secure Sockets Layer (SSL) protocol, a
security protocol that provides privacy over the Internet. The protocol
allows client/server applications to communicate in a way that cannot be
eavesdropped. Server's are always authenticated and clients are optionally
authenticated.

3. INTRODUCTION

The SSL Protocol is designed to provide privacy between two
communicating applications (a client and a server). Second, the protocol is
designed to authenticate the server, and optionally the client. SSL requires
a reliable transport protocol (e.g. TCP) for data transmission and
reception.

The advantage of the SSL Protocol is that it is application protocol
independent. A "higher level" application protocol (e.g. HTTP, FTP,
TELNET, etc.) can layer on top of the SSL Protocol transparently. The
SSL Protocol can negotiate an encryption algorithm and session key as
well as authenticate a server before the application protocol transmits or
receives its first byte of data. All of the application protocol data is
transmitted encrypted, ensuring privacy.

The SSL protocol provides "channel security" which has three basic
properties:

  The channel is private. Encryption is used for all messages after a
simple handshake is used to define a secret key. Symmetric
cryptography is used for data encryption (e.g. DES, RC4, etc.)

  The channel is authenticated. The server endpoint of the conversation
is always authenticated, while the client endpoint is optionally

authenticated. Asymmetric cryptography is used for authentication

(e.g. Public Key Cryptography).

   The channel is reliable. The message transport includes a message
integrity check (using a MAC). Secure hash functions (e.g. MD2,
MD5) are used for MAC computations.

The SSL protocol is actually composed of two protocols. At the lowest
level, layered on top of some reliable transport protocol, is the "SSL
Record Protocol". The SSL Record Protocol is used for encapsulation of
all transmitted and received data, including the SSL Handshake Protocol,
which is used to establish security parameters.
4. SSL Record Protocol Specification

4.1 SSL Record Header Format

In SSL, all data sent is encapsulated in a record, an object which is
composed of a header and some non-zero amount of data. Each record
header contains a two or three byte length code. If the most significant bit
is set in the first byte of the record length code then the record has no
padding and the total header length will be 2 bytes, otherwise the record
has padding and the total header length will be 3 bytes. The record header
is transmitted before the data portion of the record.

Note that in the long header case (3 bytes total), the second most
significant bit in the first byte has special meaning. When zero, the record
being sent is a data record. When one, the record being sent is a security
escape (there are currently no examples of security escapes; this is
reserved for future versions of the protocol). In either case, the length code
describes how much data is in the record.

The record length code does not include the number of bytes consumed by
the record header (2 or 3). For the 2 byte header, the record length is
computed by (using a "C"-like notation):

RECORD-LENGTH = ((byte[0] & 0x7f) << 8)) | byte[1];

Where byte[0] represents the first byte received and byte[1] the second
byte received. When the 3 byte header is used, the record length is
computed as follows (using a "C"-like notation):

RECORD-LENGTH = ((byte[0] & 0x3f) << 8)) | byte[1];
IS-ESCAPE = (byte[0] & 0x40) != 0;
PADDING = byte[2];

The record header defines a value called PADDING. The PADDING
value specifies how many bytes of data were appended to the original
record by the sender. The padding data is used to make the record length
be a multiple of the block ciphers block size when a block cipher is used
for encryption.

The sender of a "padded" record appends the padding data to the end of its
normal data and then encrypts the total amount (which is now a multiple

of the block cipher's block size). The actual value of the padding data is
unimportant, but the encrypted form of it must be transmitted for the
receiver to properly decrypt the record. Once the total amount being

transmitted is known the header can be properly constructed with the
PADDING value set appropriately.

The receiver of a padded record decrypts the entire record data (sans
record length and the optional padding) to get the clear data, then subtracts
the PADDING value from the RECORD-LENGTH to determine the
final RECORD-LENGTH. The clear form of the padding data must be
discarded.

4.1.1 SSL Record Data Format

The data portion of an SSL record is composed of three components
(transmitted and received in the order shown):

```
MAC-DATA[MAC-SIZE]
ACTUAL-DATA[N]
PADDING-DATA[PADDING]
```

ACTUAL-DATA is the actual data being transmitted (the message
payload). PADDING-DATA is the padding data sent when a block cipher
is used and padding is needed. Finally, MAC-DATA is the "Message
Authentication Code".

When SSL records are sent in the clear, no cipher is used.Consequently the
amount of PADDING-DATA will be zero and the amount of MAC-
DATA will be zero. When encryption is in effect, the PADDING-DATA
will be a function of the cipher block size. The MAC-DATA is a function
of the CIPHER-CHOICE (more about that later).

The MAC-DATA is computed as follows:

```
MAC-DATA = HASH[ SECRET, ACTUAL-DATA, PADDING-DATA,
SEQUENCE-NUMBER ]
```

Where the SECRET data is fed to the hash function first, followed by the
ACTUAL-DATA, which is followed by the PADDING-DATA which is
finally followed by the SEQUENCE-NUMBER. The SEQUENCE-
NUMBER is a 32 bit value which is presented to the hash function as four
bytes, with the first byte being the most significant byte of the sequence
number, the second byte being the next most significant byte of the
sequence number, the third byte being the third most significant byte, and
the fourth byte being the least significant byte (that is, in network byte
order or "big endian" order).

MAC-SIZE is a function of the digest algorithm being used. For MD2 and
MD5 the MAC-SIZE will be 16 bytes (128 bits).

The SECRET value is a function of which party is sending the message. If
the client is sending the message then the SECRET is the CLIENT-
WRITE-KEY (the server will use the SERVER-READ-KEY to verify

the MAC). If the client is receiving the message then the SECRET is the
CLIENT-READ-KEY (the server will use the SERVER-WRITE-KEY
to
generate the MAC).

The SEQUENCE-NUMBER is a counter which is incremented by both
the sender and the receiver. For each transmission direction, a pair of
counters is kept (one by the sender, one by the receiver). Every time a
message is sent by a sender the counter is incremented. Sequence numbers
are 32 bit unsigned quantities and must wrap to zero after incrementing
past 0xFFFFFFFF.

The receiver of a message uses the expected value of the sequence number
as input into the MAC HASH function (the HASH function is chosen from
the CIPHER-CHOICE). The computed MAC-DATA must agree bit for
bit with the transmitted MAC-DATA. If the comparison is not identity
then the record is considered damaged, and it is to be treated as if an "I/O
Error" had occurred (i.e. an unrecoverable error is asserted and the
connection is closed).

A final consistency check is done when a block cipher is used and the
protocol is using encryption. The amount of data present in a record
(RECORD-LENGTH))must be a multiple of the cipher's block size. If
the received record is not a multiple of the cipher's block size then the
record is considered damaged, and it is to be treated as if an "I/O Error"
had occurred (i.e. an unrecoverable error is asserted and the connection is
closed).

The SSL Record Layer is used for all SSL communications, including
handshake messages, security escapes and application data transfers. The
SSL Record Layer is used by both the client and the server at all times.

For a two byte header, the maximum record length is 32767 bytes. For the
three byte header, the maximum record length is 16383 bytes. The SSL
Handshake Protocol messages are constrained to fit in a single SSL Record
Protocol record. Application protocol messages are allowed to consume
multiple SSL Record Protocol record's.

Before the first record is sent using SSL all sequence numbers are
initialized to zero. The transmit sequence number is incremented after
every message sent, starting with the CLIENT-HELLO and SERVER-
HELLO messages.

5. SSL Handshake Protocol Specification

5.1 SSL Handshake Protocol Flow

The SSL Handshake Protocol is used to negotiate security enhancements
to data sent using the SSL Record Protocol. The security enhancements
consist of authentication, symmetric encryption, and message integrity.

The SSL Handshake Protocol has two major phases. The first phase is
used to establish private communications. The second phase is used for
client authentication.

5.1.1 **Phase 1**

The first phase is the initial connection phase where both parties
communicate their "hello" messages. The client initiates the conversation
by sending the CLIENT-HELLO message. The server receives the

CLIENT-HELLO message and processes it responding with the
SERVER-HELLO message.

At this point both the client and server have enough information to know
whether or not a new "master key" is needed (The master key is used for
production of the symmetric encryption session keys). When a new master
key is not needed, both the client and the server proceed immediately to
phase 2.

When a new master key is needed, the SERVER-HELLO message will
contain enough information for the client to generate it. This includes the
server's signed certificate (more about that later), a list of bulk cipher
specifications (see below), and a connection-id (a connection-id is a
randomly generated value generated by the server that is used by the client
and server during a single connection). The client generates the master key
and responds with a CLIENT-MASTER-KEY message (or an ERROR
message if the server information indicates that the client and server
cannot agree on a bulk cipher).

It should be noted here that each SSL endpoint uses a pair of ciphers per
connection (for a total of four ciphers). At each endpoint, one cipher is
used for outgoing communications, and one is used for incoming
communications. When the client or server generate a session key, they
actually generate two keys, the SERVER-READ-KEY (also known as the
CLIENT-WRITE-KEY) and the SERVER-WRITE-KEY (also known
as the CLIENT-READ-KEY). The master key is used by the client and
server to generate the various session keys (more about that later).

Finally, the server sends a SERVER-VERIFY message to the client after
the master key has been determined. This final step authenticates the
server, because only a server which has the appropriate public key can
know the master key.

5.1.2 **Phase 2**

The second phase is the client authentication phase. The server has already
been authenticated by the client in the first phase, so this phase is primarily
used to authenticate the client. In a typical scenario, the server will require
authentication of the client and send a REQUEST-CERTIFICATE
message. The client will answer in the positive if it has the needed
information, or send an ERROR message if it does not. This protocol
specification does not define the semantics of an ERROR response to a
server request (e.g., an implementation can ignore the error, close the
connection, etc. and still conform to this specification). In addition, it is
permissable for a server to cache client authentication information with the
"session-id" cache. The server is not required to re-authenticate the client
on every connection.

When a party is done authenticating the other party, it sends its finished
message. For the client, the CLIENT-FINISHED message contains the
encrypted form of the CONNECTION-ID for the server to verify. If the
verification fails, the server sends an ERROR message.

Once a party has sent its finished message it must continue to listen to its
peers messages until it too receives a finished message. Once a party has

both sent a finished message and received its peers finished message, the
SSL handshake protocol is done. At this point the application protocol
begins to operate (Note: the application protocol continues to be layered
on the SSL Record Protocol).

5.2 Typical Protocol Message Flow

The following sequences define several typical protocol message flows for
the SSL Handshake Protocol. In these examples we have two principals in
the conversation: the client and the server. We use a notation commonly
found in the literature [10]. When something is enclosed in curly braces
"{something}key" then the something has been encrypted using "key".

5.2.1 Assuming no session-identifier

```
client-hello          C -> S: challenge, cipher_specs
server-hello          S -> C: connection-id, server_certificate,
                                  cipher_specs
client-master-key     C -> S: {master_key}server_public_key
client-finish         C -> S: {connection-id}client_write_key
server-verify         S -> C: {challenge}server_write_key
server-finish         S -> C: {new_session_id}server_write_key
```

5.2.2 Assuming a session-identifier was found by both client & server

```
client-hello          C -> S: challenge, session_id, cipher_specs
server-hello          S -> C: connection-id, session_id_hit
client-finish         C -> S: {connection-id}client_write_key
server-verify         S -> C: {challenge}server_write_key
server-finish         S -> C: {session_id}server_write_key
```

5.2.3 Assuming a session-identifier was used and client authentication
is used

```
client-hello          C -> S: challenge, session_id, cipher_specs
server-hello          S -> C: connection-id, session_id_hit
client-finish         C -> S: {connection-id}client_write_key
server-verify         S -> C: {challenge}server_write_key
request-certificate   S -> C: {auth_type,challenge'}
                                  server_write_key
client-certificate    C -> S: {cert_type,client_cert,
                                  response_data}client_write_key
server-finish         S -> C: {session_id}server_write_key
```

In this last exchange, the response_data is a function of the auth_type.

5.3 Errors

Error handling in the SSL connection protocol is very simple. When an
error is detected, the detecting party sends a message to the other party.
Errors that are not recoverable cause the client and server to abort the
secure connection. Servers and client are required to "forget" any session-
identifiers associated with a failing connection.

The SSL Handshake Protocol defines the following errors:

NO-CIPHER-ERROR
This error is returned by the client to the server when it cannot find
a cipher or key size that it supports that is also supported by the
server. This error is not recoverable.

NO-CERTIFICATE-ERROR
When a REQUEST-CERTIFICATE message is sent, this error may
be returned if the client has no certificate to reply with. This error
is recoverable (for client authentication only).

BAD-CERTIFICATE-ERROR
This error is returned when a certificate is deemed bad by the
receiving party. Bad means that either the signature of the
certificate was bad or that the values in the certificate were
inappropriate (e.g. a name in the certificate did not match the
expected name). This error is recoverable (for client authentication
only).

UNSUPPORTED-CERTIFICATE-TYPE-ERROR
This error is returned when a client/server receives a certificate
type that it can't support. This error is recoverable (for client
authentication only).

5.4 SSL Handshake Protocol Messages

The SSL Handshake Protocol messages are encapsulated using the SSL
Record Protocol and are composed of two parts: a single byte message
type code, and some data. The client and server exchange messages until
both ends have sent their "finished" message, indicating that they are
satisfied with the SSL Handshake Protocol conversation. While one end
may be finished, the other may not, therefore the finished end must
continue to receive SSL Handshake Protocol messages until it receives a
"finished" message from its peer.

After the pair of session keys has been determined by each party, the
message bodies are encrypted. For the client, this happens after it verifies
the session-identifier or creates a new master key and has sent it to the
server. For the server, this happens after the session-identifier is found to
be good, or the server receives the client's master key message.

The following notation is used for SSLHP messages:

```
char MSG-EXAMPLE
char FIELD1
char FIELD2
char THING-MSB
char THING-LSB
char THING-DATA[(MSB<<8)|LSB];
...
```

This notation defines the data in the protocol message, including the
message type code. The order is presented top to bottom, with the top most
element being transmitted first, and the bottom most element transferred
last.

For the "THING-DATA" entry, the MSB and LSB values are actually
THING-MSB and THING-LSB (respectively) and define the number of
bytes of data actually present in the message. For example, if THING-
MSB were zero and THING-LSB were 8 then the THING-DATA array
would be exactly 8 bytes long. This shorthand is used below.

Length codes are unsigned values, and when the MSB and LSB are
combined the result is an unsigned value. Unless otherwise specified
lengths values are "length in bytes".

5.5 Client Only Protocol Messages

There are several messages that are only generated by clients. These
messages are never generated by correctly functioning servers. A client
receiving such a message closes the connection to the server and returns an
error status to the application through some unspecified mechanism.

5.5.1 CLIENT-HELLO (Phase 1; Sent in the clear)

```
char MSG-CLIENT-HELLO
char CLIENT-VERSION-MSB
char CLIENT-VERSION-LSB
char CIPHER-SPECS-LENGTH-MSB
char CIPHER-SPECS-LENGTH-LSB
char SESSION-ID-LENGTH-MSB
char SESSION-ID-LENGTH-LSB
char CHALLENGE-LENGTH-MSB
char CHALLENGE-LENGTH-LSB
char CIPHER-SPECS-DATA[(MSB<<8)|LSB]
char SESSION-ID-DATA[(MSB<<8)|LSB]
char CHALLENGE-DATA[(MSB<<8)|LSB]
```

When a client first connects to a server it is required to send the CLIENT-
HELLO message. The server is expecting this message from the client as
its first message. It is an error for a client to send anything else as its
first
message.

The client sends to the server its SSL version, its cipher specs (see below),
some challenge data, and the session-identifier data. The session-identifier

data is only sent if the client found a session-identifier in its cache for the
server, and the SESSION-ID-LENGTH will be non-zero. When there is
no session-identifier for the server SESSION-ID-LENGTH must be zero.
The challenge data is used to authenticate the server. After the client and
server agree on a pair of session keys, the server returns a SERVER-
VERIFY message with the encrypted form of the CHALLENGE-DATA.

Also note that the server will not send its SERVER-HELLO message
until it has received the CLIENT-HELLO message. This is done so that
the server can indicate the status of the client's session-identifier back to
the client in the server's first message (i.e. to increase protocol efficiency
and reduce the number of round trips required).

The server examines the CLIENT-HELLO message and will verify that it
can support the client version and one of the client cipher specs. The
server can optionally edit the cipher specs, removing any entries it doesn't

choose to support. The edited version will be returned in the SERVER-
HELLO message if the session-identifier is not in the server's cache.

The CIPHER-SPECS-LENGTH must be greater than zero and a
multiple of 3. The SESSION-ID-LENGTH must either be zero or 16.
The CHALLENGE-LENGTH must be greater than or equal to 16 and
less than or equal to 32.

This message must be the first message sent by the client to the server.
After the message is sent the client waits for a SERVER-HELLO
message. Any other message returned by the server (other than ERROR)
is disallowed.

5.5.2 CLIENT-MASTER-KEY (Phase 1; Sent primarily in the clear)

```
char MSG-CLIENT-MASTER-KEY
char CIPHER-KIND[3]
char CLEAR-KEY-LENGTH-MSB
char CLEAR-KEY-LENGTH-LSB
char ENCRYPTED-KEY-LENGTH-MSB
char ENCRYPTED-KEY-LENGTH-LSB
char KEY-ARG-LENGTH-MSB
char KEY-ARG-LENGTH-LSB
char CLEAR-KEY-DATA[MSB<<8|LSB]
char ENCRYPTED-KEY-DATA[MSB<<8|LSB]
char KEY-ARG-DATA[MSB<<8|LSB]
```

The client sends this message when it has determined a master key for the
server to use. Note that when a session-identifier has been agreed upon,
this message is not sent.

The CIPHER-KIND field indicates which cipher was chosen from the
server's CIPHER-SPECS.

The CLEAR-KEY-DATA contains the clear portion of the MASTER-
KEY. The CLEAR-KEY-DATA is combined with the SECRET-KEY-
DATA (described shortly) to form the MASTER-KEY, with the

SECRET-KEY-DATA being the least significant bytes of the final
MASTER-KEY. The ENCRYPTED-KEY-DATA contains the secret
portions of the MASTER-KEY, encrypted using the server's public key.
The encryption block is formatted using block type 2 from PKCS#1 [5].
The data portion of the block is formatted as follows:

char SECRET-KEY-DATA[SECRET-LENGTH]

SECRET-LENGTH is the number of bytes of each session key that is
being transmitted encrypted. The SECRET-LENGTH plus the CLEAR-
KEY-LENGTH equals the number of bytes present in the cipher key (as
defined by the CIPHER-KIND). It is an error if the SECRET-LENGTH
found after decrypting the PKCS#1 formatted encryption block doesn't
match the expected value. It is also an error if CLEAR-KEY-LENGTH is
non-zero and the CIPHER-KIND is not an export cipher.

If the key algorithm needs an argument (for example, DES-CBC's
initialization vector) then the KEY-ARG-LENGTH fields will be non-

Hickman   [page 9]

zero and the KEY-ARG-DATA will contain the relevant data. For the
SSL_CK_RC2_128_CBC_WITH_MD5,
SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5,
SSL_CK_IDEA_128_CBC_WITH_MD5,
SSL_CK_DES_64_CBC_WITH_MD5 and
SSL_CK_DES_192_EDE3_CBC_WITH_MD5 algorithms the KEY-ARG
data must be present and be exactly 8 bytes long.

Client and server session key production is a function of the CIPHER-
CHOICE:

SSL_CK_RC4_128_WITH_MD5
SSL_CK_RC4_128_EXPORT40_WITH_MD5
SSL_CK_RC2_128_CBC_WITH_MD5
SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5
SSL_CK_IDEA_128_CBC_WITH_MD5

KEY-MATERIAL-0 = MD5[MASTER-KEY, "0", CHALLENGE,
CONNECTION-ID]
KEY-MATERIAL-1 = MD5[MASTER-KEY, "1", CHALLENGE,
CONNECTION-ID]

CLIENT-READ-KEY = KEY-MATERIAL-0[0-15]
CLIENT-WRITE-KEY = KEY-MATERIAL-1[0-15]

Where KEY-MATERIAL-0[0-15] means the first 16 bytes of the KEY-
MATERIAL-0 data, with KEY-MATERIAL-0[0] becoming the most
significant byte of the CLIENT-READ-KEY.

Data is fed to the MD5 hash function in the order shown, from left to
right: first the MASTER-KEY, then the "0" or "1", then the
CHALLENGE and then finally the CONNECTION-ID.

Note that the "0" means the ascii zero character (0x30), not a zero value.
"1" means the ascii 1 character (0x31). MD5 produces 128 bits of output
data which are used directly as the key to the cipher algorithm (The most
significant byte of the MD5 output becomes the most significant byte of
the key material).

SSL_CK_DES_64_CBC_WITH_MD5

 KEY-MATERIAL-0 = MD5[MASTER-KEY, CHALLENGE,
CONNECTION-ID]

 CLIENT-READ-KEY = KEY-MATERIAL-0[0-7]
 CLIENT-WRITE-KEY = KEY-MATERIAL-0[8-15]

For DES-CBC, a single 16 bytes of key material are produced using MD5.
The first 8 bytes of the MD5 digest are used as the CLIENT-READ-KEY
while the remaining 8 bytes are used as the CLIENT-WRITE-KEY. The
initialization vector is provided in the KEY-ARG-DATA. Note that the
raw key data is not parity adjusted and that this step must be performed
before the keys are legitimate DES keys.

SSL_CK_DES_192_EDE3_CBC_WITH_MD5

```
 KEY-MATERIAL-0 = MD5[ MASTER-KEY, "0", CHALLENGE,
CONNECTION-ID ]
 KEY-MATERIAL-1 = MD5[ MASTER-KEY, "1", CHALLENGE,
CONNECTION-ID ]
 KEY-MATERIAL-2 = MD5[ MASTER-KEY, "2", CHALLENGE,
CONNECTION-ID ]

 CLIENT-READ-KEY-0 = KEY-MATERIAL-0[0-7]
 CLIENT-READ-KEY-1 = KEY-MATERIAL-0[8-15]
 CLIENT-READ-KEY-2 = KEY-MATERIAL-1[0-7]
 CLIENT-WRITE-KEY-0 = KEY-MATERIAL-1[8-15]
 CLIENT-WRITE-KEY-1 = KEY-MATERIAL-2[0-7]
 CLIENT-WRITE-KEY-2 = KEY-MATERIAL-2[8-15]
```

Data is fed to the MD5 hash function in the order shown, from left to
right: first the MASTER-KEY, then the "0", "1" or "2", then the
CHALLENGE and then finally the CONNECTION-ID. Note that the
"0" means the ascii zero character (0x30), not a zero value. "1" means the
ascii 1 character (0x31). "2" means the ascii 2 character (0x32).

A total of 6 keys are produced, 3 for the read side DES-EDE3 cipher and 3
for the write side DES-EDE3 function. The initialization vector is
provided in the KEY-ARG-DATA. The keys that are produced are not
parity adjusted. This step must be performed before proper DES keys are
usable.

Recall that the MASTER-KEY is given to the server in the CLIENT-
MASTER-KEY message. The CHALLENGE is given to the server by
the client in the CLIENT-HELLO message. The CONNECTION-ID is
given to the client by the server in the SERVER-HELLO message. This
makes the resulting cipher keys a function of the original session and the
current session. Note that the master key is never directly used to encrypt
data, and therefore cannot be easily discovered.

The CLIENT-MASTER-KEY message must be sent after the CLIENT-
HELLO message and before the CLIENT-FINISHED message. The
CLIENT-MASTER-KEY message must be sent if the SERVER-
HELLO message contains a SESSION-ID-HIT value of 0.

5.5.3 CLIENT-CERTIFICATE (Phase 2; Sent encrypted)

```
char MSG-CLIENT-CERTIFICATE
char CERTIFICATE-TYPE
char CERTIFICATE-LENGTH-MSB
char CERTIFICATE-LENGTH-LSB
char RESPONSE-LENGTH-MSB
char RESPONSE-LENGTH-LSB
char CERTIFICATE-DATA[MSB<<8|LSB]
char RESPONSE-DATA[MSB<<8|LSB]
```

This message is sent by one an SSL client in response to a server
REQUEST-CERTIFICATE message. The CERTIFICATE-DATA
contains data defined by the CERTIFICATE-TYPE value. An ERROR

message is sent with error code NO-CERTIFICATE-ERROR when this
request cannot be answered properly (e.g. the receiver of the message has

no registered certificate).

CERTIFICATE-TYPE is one of:

SSL_X509_CERTIFICATE
The CERTIFICATE-DATA contains an X.509 (1988) [3] signed
certificate.

The RESPONSE-DATA contains the authentication response data. This
data is a function of the AUTHENTICATION-TYPE value sent by the
server.

When AUTHENTICATION-TYPE is
SSL_AT_MD5_WITH_RSA_ENCRYPTION then the RESPONSE-
DATA contains a digital signature of the following components (in the
order shown):

   the KEY-MATERIAL-0
   the KEY-MATERIAL-1 (only if defined by the cipher kind)
   the KEY-MATERIAL-2 (only if defined by the cipher kind)
   the CERTIFICATE-CHALLENGE-DATA (from the
REQUEST-CERTIFICATE message)
   the server's signed certificate (from the SERVER-HELLO
message)

The digital signature is constructed using MD5 and then encrypted using
the clients private key, formatted according to PKCS#1's digital signature
standard [5]. The server authenticates the client by verifying the digital
signature using standard techniques. Note that other digest functions are
supported. Either a new AUTHENTICATION-TYPE can be added, or
the algorithm-id in the digital signature can be changed.

This message must be sent by the client only in response to a REQUEST-
CERTIFICATE message.

5.5.4 CLIENT-FINISHED (Phase 2; Sent encrypted)

char MSG-CLIENT-FINISHED
char CONNECTION-ID[N-1]

The client sends this message when it is satisfied with the server. Note that
the client must continue to listen for server messages until it receives a
SERVER-FINISHED message. The CONNECTION-ID data is the
original connection-identifier the server sent with its SERVER-HELLO
message, encrypted using the agreed upon session key.

"N" is the number of bytes in the message that was sent, so "N-1" is the
number of bytes in the message without the message header byte.

For version 2 of the protocol, the client must send this message after it has
received the SERVER-HELLO message. If the SERVER-HELLO
message SESSION-ID-HIT flag is non-zero then the CLIENT-
FINISHED message is sent immediately, otherwise the CLIENT-
FINISHED message is sent after the CLIENT-MASTER-KEY message.

There are several messages that are only generated by servers. The
messages are never generated by correctly functioning clients.

5.6.1 SERVER-HELLO (Phase 1; Sent in the clear)

```
char MSG-SERVER-HELLO
char SESSION-ID-HIT
char CERTIFICATE-TYPE
char SERVER-VERSION-MSB
char SERVER-VERSION-LSB
char CERTIFICATE-LENGTH-MSB
char CERTIFICATE-LENGTH-LSB
char CIPHER-SPECS-LENGTH-MSB
char CIPHER-SPECS-LENGTH-LSB
char CONNECTION-ID-LENGTH-MSB
char CONNECTION-ID-LENGTH-LSB
char CERTIFICATE-DATA[MSB<<8|LSB]
char CIPHER-SPECS-DATA[MSB<<8|LSB]
char CONNECTION-ID-DATA[MSB<<8|LSB]
```

The server sends this message after receiving the clients CLIENT-
HELLO message. The server returns the SESSION-ID-HIT flag
indicating whether or not the received session-identifier is known by the
server (i.e. in the server's session-identifier cache). The SESSION-ID-
HIT flag will be non-zero if the client sent the server a session-identifier
(in the CLIENT-HELLO message with SESSION-ID-LENGTH != 0)
and the server found the client's session-identifier in its cache. If the
SESSION-ID-HIT flag is non-zero then the CERTIFICATE-TYPE,
CERTIFICATE-LENGTH and CIPHER-SPECS-LENGTH fields will
be zero.

The CERTIFICATE-TYPE value, when non-zero, has one of the values
described above (see the information on the CLIENT-CERTIFICATE
message).

When the SESSION-ID-HIT flag is zero, the server packages up its
certificate, its cipher specs and a connection-id to send to the client. Using
this information the client can generate a session key and return it to the
server with the CLIENT-MASTER-KEY message.

When the SESSION-ID-HIT flag is non-zero, both the server and the
client compute a new pair of session keys for the current session derived
from the MASTER-KEY that was exchanged when the SESSION-ID
was created. The SERVER-READ-KEY and SERVER-WRITE-KEY
are derived from the original MASTER-KEY keys in the same manner as
the CLIENT-READ-KEY and CLIENT-WRITE-KEY:

```
SERVER-READ-KEY = CLIENT-WRITE-KEY
SERVER-WRITE-KEY = CLIENT-READ-KEY
```

Note that when keys are being derived and the SESSION-ID-HIT flag is
set and the server discovers the client's session-identifier in the servers
cache, then the KEY-ARG-DATA is used from the time when the

SESSION-ID was established. This is because the client does not send
new KEY-ARG-DATA (recall that the KEY-ARG-DATA is sent only in
the CLIENT-MASTER-KEY message).

The CONNECTION-ID-DATA is a string of randomly generated bytes
used by the server and client at various points in the protocol. The
CLIENT-FINISHED message contains an encrypted version of the
CONNECTION-ID-DATA. The length of the CONNECTION-ID must
be between 16 and than 32 bytes, inclusive.

The CIPHER-SPECS-DATA define a cipher type and key length (in bits)
that the receiving end supports. Each SESSION-CIPHER-SPEC is 3
bytes long and looks like this:

char CIPHER-KIND-0
char CIPHER-KIND-1
char CIPHER-KIND-2

Where CIPHER-KIND is one of:

SSL_CK_RC4_128_WITH_MD5
SSL_CK_RC4_128_EXPORT40_WITH_MD5
SSL_CK_RC2_128_CBC_WITH_MD5
SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5
SSL_CK_IDEA_128_CBC_WITH_MD5
SSL_CK_DES_64_CBC_WITH_MD5
SSL_CK_DES_192_EDE3_CBC_WITH_MD5

This list is not exhaustive and may be changed in the future.

The SSL_CK_RC4_128_EXPORT40_WITH_MD5 cipher is an RC4
cipher where some of the session key is sent in the clear and the rest is sent
encrypted (exactly 40 bits of it). MD5 is used as the hash function for
production of MAC's and session key's. This cipher type is provided to
support "export" versions (i.e. versions of the protocol that can be
distributed outside of the United States) of the client or server.

An exportable implementation of the SSL Handshake Protocol will have
secret key lengths restricted to 40 bits. For non-export implementations
key lengths can be more generous (we recommend at least 128 bits). It is
permissible for the client and server to have a non-intersecting set of
stream ciphers. This, simply put, means they cannot communicate.

Version 2 of the SSL Handshake Protocol defines the
SSL_CK_RC4_128_WITH_MD5 to have a key length of 128 bits. The
SSL_CK_RC4_128_EXPORT40_WITH_MD5 also has a key length of
128 bits. However, only 40 of the bits are secret (the other 88 bits are sent
in the clear by the client to the server).

The SERVER-HELLO message is sent after the server receives the
CLIENT-HELLO message, and before the server sends the SERVER-
VERIFY message.
5.6.2 SERVER-VERIFY (Phase 1; Sent encrypted)

char MSG-SERVER-VERIFY

```
char CHALLENGE-DATA[N-1]
```

The server sends this message after a pair of session keys (SERVER-
READ-KEY and SERVER-WRITE-KEY) have been agreed upon either
by a session-identifier or by explicit specification with the CLIENT-
MASTER-KEY message. The message contains an encrypted copy of the
CHALLENGE-DATA sent by the client in the CLIENT-HELLO
message.

"N" is the number of bytes in the message that was sent, so "N-1" is the
number of bytes in the CHALLENGE-DATA without the message
header byte.

This message is used to verify the server as follows. A legitimate server
will have the private key that corresponds to the public key contained in
the server certificate that was transmitted in the SERVER-HELLO
message. Accordingly, the legitimate server will be able to extract and
reconstruct the pair of session keys (SERVER-READ-KEY and
SERVER-WRITE-KEY). Finally, only a server that has done the
extraction and decryption properly can correctly encrypt the
CHALLENGE-DATA. This, in essence, "proves" that the server has the
private key that goes with the public key in the server's certificate.

The CHALLENGE-DATA must be the exact same length as originally
sent by the client in the CLIENT-HELLO message. Its value must match
exactly the value sent in the clear by the client in the CLIENT-HELLO
message. The client must decrypt this message and compare the value
received with the value sent, and only if the values are identical is the
server to be "trusted". If the lengths do not match or the value doesn't
match then the connection is to be closed by the client.

This message must be sent by the server to the client after either detecting
a session-identifier hit (and replying with a SERVER-HELLO message
with SESSION-ID-HIT not equal to zero) or when the server receives the
CLIENT-MASTER-KEY message. This message must be sent before
any Phase 2 messages or a SERVER-FINISHED message.

[5.6.3](#) SERVER-FINISHED (Phase 2; Sent encrypted)

```
char MSG-SERVER-FINISHED
char SESSION-ID-DATA[N-1]
```

The server sends this message when it is satisfied with the clients security
handshake and is ready to proceed with transmission/reception of the
higher level protocols data. The SESSION-ID-DATA is used by the client
and the server at this time to add entries to their respective session-
identifier caches. The session-identifier caches must contain a copy of the
MASTER-KEY sent in the CLIENT-MASTER-KEY message as the
master key is used for all subsequent session key generation.

"N" is the number of bytes in the message that was sent, so "N-1" is the
number of bytes in the SESSION-ID-DATA without the message header
byte.

This message must be sent after the SERVER-VERIFY message.

5.6.4 REQUEST-CERTIFICATE (Phase 2; Sent encrypted)

char MSG-REQUEST-CERTIFICATE
char AUTHENTICATION-TYPE
char CERTIFICATE-CHALLENGE-DATA[N-2]

A server may issue this request at any time during the second phase of the
connection handshake, asking for the client's certificate. The client
responds with a CLIENT-CERTIFICATE message immediately if it has
one, or an ERROR message (with error code NO-CERTIFICATE-
ERROR) if it doesn't. The CERTIFICATE-CHALLENGE-DATA is a
short byte string (whose length is greater than or equal to 16 bytes and less
than or equal to 32 bytes) that the client will use to respond to this
message.

The AUTHENTICATION-TYPE value is used to choose a particular
means of authenticating the client. The following types are defined:

SSL_AT_MD5_WITH_RSA_ENCRYPTION

The SSL_AT_MD5_WITH_RSA_ENCRYPTION type requires that the
client construct an MD5 message digest using information as described
above in the section on the CLIENT-CERTIFICATE message. Once the
digest is created, the client encrypts it using its private key (formatted
according to the digital signature standard defined in PKCS#1). The server
authenticates the client when it receives the CLIENT-CERTIFICATE
message.

This message may be sent after a SERVER-VERIFY message and before
a SERVER-FINISHED message.

5.7 Client/Server Protocol Messages

These messages are generated by both the client and the server.

5.7.1 ERROR (Sent clear or encrypted)

char MSG-ERROR
char ERROR-CODE-MSB
char ERROR-CODE-LSB

This message is sent when an error is detected. After the message is sent,
the sending party shuts the connection down. The receiving party records
the error and then shuts its connection down.

This message is sent in the clear if an error occurs during session key
negotiation. After a session key has been agreed upon, errors are sent
encrypted like all other messages.

Appendix A: ASN.1 Syntax For Certificates

Certificates are used by SSL to authenticate servers and clients. SSL
Certificates are based largely on the X.509 [3] certificates. An X.509
certificate contains the following information (in ASN.1 [1] notation):

```
X.509-Certificate ::= SEQUENCE {
  certificateInfo CertificateInfo,
  signatureAlgorithm AlgorithmIdentifier,
  signature BIT STRING
}

CertificateInfo ::= SEQUENCE {
  version [0] Version DEFAULT v1988,
  serialNumber CertificateSerialNumber,
  signature AlgorithmIdentifier,
  issuer Name,
  validity Validity,
  subject Name,
  subjectPublicKeyInfo SubjectPublicKeyInfo
}

Version ::= INTEGER { v1988(0) }

CertificateSerialNumber ::= INTEGER

Validity ::= SEQUENCE {
  notBefore UTCTime,
  notAfter UTCTime
}

SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm AlgorithmIdentifier,
  subjectPublicKey BIT STRING
}

AlgorithmIdentifier ::= SEQUENCE {
  algorithm OBJECT IDENTIFIER,
  parameters ANY DEFINED BY ALGORITHM OPTIONAL
}
```

For SSL's purposes we restrict the values of some of the X.509 fields:

   The X.509-Certificate::signatureAlgorithm and
CertificateInfo::signature fields must be identical in value.

   The issuer name must resolve to a name that is deemed acceptable by
the application using SSL. How the application using SSL does this is
outside the scope of this memo.

Certificates are validated using a few straightforward steps. First, the
signature on the certificate is checked and if invalid, the certificate is
invalid (either a transmission error or an attempted forgery occurred).
Next, the CertificateInfo::issuer field is verified to be an issuer that the
application trusts (using an unspecified mechanism). The
CertificateInfo::validity field is checked against the current date and
verified.

Finally, the CertificateInfo::subject field is checked. This check is
optional and depends on the level of trust required by the application using
SSL.

Appendix B: Attribute Types and Object Identifiers

SSL uses a subset of the X.520 selected attribute types as well as a
few specific object identifiers. Future revisions of the SSL protocol
may include support for more attribute types and more object
identifiers.

B.1 Selected attribute types

commonName { attributeType 3 }
The common name contained in the distinguished name contained
within a certificate issuer or certificate subject.

countryName { attributeType 6 }
The country name contained in the distinguished name contained
within a certificate issuer or certificate subject.

localityName { attributeType 7 }
The locality name contained in the distinguished name contained
within a certificate issuer or certificate subject.

stateOrProvinceName { attributeType 8 }
The state or province name contained in the distinguished name
contained within a certificate issuer or certificate subject.

organizationName { attributeType 10 }
The organization name contained in the distinguished name contained
within a certificate issuer or certificate subject.

organizationalUnitName { attributeType 11 }
The organizational unit name contained in the distinguished name
contained within a certificate issuer or certificate subject.

B.2 Object identifiers

md2withRSAEncryption { ... pkcs(1) 1 2 }
The object identifier for digital signatures that use both MD2 and RSA
encryption. Used by SSL for certificate signature verification.

md5withRSAEncryption { ... pkcs(1) 1 4 }
The object identifier for digital signatures that use both MD5 and RSA
encryption. Used by SSL for certificate signature verification.

rc4 { ... rsadsi(113549) 3 4 }
The RC4 symmetric stream cipher algorithm used by SSL for bulk
encryption.

Appendix C: Protocol Constant Values

This section describes various protocol constants. A special value needs
mentioning - the IANA reserved port number for "https" (HTTP using
SSL). IANA has reserved port number 443 (decimal) for "https". IANA
has also reserved port number 465 for "ssmtp" and port number 563 for
"snntp".

C.1 **Protocol Version Codes**

```
#define SSL_CLIENT_VERSION 0x0002
#define SSL_SERVER_VERSION 0x0002
```

C.2 **Protocol Message Codes**

The following values define the message codes that are used by version
2 of the SSL Handshake Protocol.

```
#define SSL_MT_ERROR                    0
#define SSL_MT_CLIENT_HELLO             1
#define SSL_MT_CLIENT_MASTER_KEY        2
#define SSL_MT_CLIENT_FINISHED          3
#define SSL_MT_SERVER_HELLO             4
#define SSL_MT_SERVER_VERIFY            5
#define SSL_MT_SERVER_FINISHED          6
#define SSL_MT_REQUEST_CERTIFICATE      7
#define SSL_MT_CLIENT_CERTIFICATE       8
```

C.3 **Error Message Codes**

The following values define the error codes used by the ERROR message.

```
#define SSL_PE_NO_CIPHER                        0x0001
#define SSL_PE_NO_CERTIFICATE                   0x0002
#define SSL_PE_BAD_CERTIFICATE                  0x0004
#define SSL_PE_UNSUPPORTED_CERTIFICATE_TYPE     0x0006
```

C.4 **Cipher Kind Values**

The following values define the CIPHER-KIND codes used in the
CLIENT-HELLO and SERVER-HELLO messages.

```
#define SSL_CK_RC4_128_WITH_MD5
        0x01,0x00,0x80
#define SSL_CK_RC4_128_EXPORT40_WITH_MD5
        0x02,0x00,0x80
#define SSL_CK_RC2_128_CBC_WITH_MD5
        0x03,0x00,0x80
#define SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5
        0x04,0x00,0x80
#define SSL_CK_IDEA_128_CBC_WITH_MD5
        0x05,0x00,0x80
#define SSL_CK_DES_64_CBC_WITH_MD5
        0x06,0x00,0x40
#define SSL_CK_DES_192_EDE3_CBC_WITH_MD5
        0x07,0x00,0xC0
```

C.5 **Certificate Type Codes**

The following values define the certificate type codes used in the
SERVER-HELLO and CLIENT-CERTIFICATE messages.

```
#define SSL_CT_X509_CERTIFICATE         0x01
```

[C.6](#) Authentication Type Codes

The following values define the authentication type codes used in the
REQUEST-CERTIFICATE message.

#define SSL_AT_MD5_WITH_RSA_ENCRYPTION 0x01

[C.7](#) Upper/Lower Bounds

The following values define upper/lower bounds for various protocol
parameters.

```
#define SSL_MAX_MASTER_KEY_LENGTH_IN_BITS      256
#define SSL_MAX_SESSION_ID_LENGTH_IN_BYTES     16
#define SSL_MIN_RSA_MODULUS_LENGTH_IN_BYTES    64
#define SSL_MAX_RECORD_LENGTH_2_BYTE_HEADER    32767
#define SSL_MAX_RECORD_LENGTH_3_BYTE_HEADER    16383
```

[C.8](#) Recommendations

Because protocols have to be implemented to be of value, we recommend
the following values for various operational parameters. This is only a
recommendation, and not a strict requirement for conformance to the
protocol.

Session-identifier Cache Timeout

Session-identifiers are kept in SSL clients and SSL servers. Session-
identifiers should have a lifetime that serves their purpose (namely,
reducing the number of expensive public key operations for a single
client/server pairing). Consequently, we recommend a maximum session-
identifier cache timeout value of 100 seconds. Given a server that can
perform N private key operations per second, this reduces the server load
for a particular client by a factor of 100.

Appendix D: Attacks

In this section we attempt to describe various attacks that might be used
against the SSL protocol. This list is not guaranteed to be exhaustive. SSL
was defined to thwart these attacks.

[D.1](#) Cracking Ciphers

SSL depends on several cryptographic technologies. RSA Public Key
encryption [5] is used for the exchange of the session key and client/server
authentication. Various cryptographic algorithms are used for the session
cipher. If successful cryptographic attacks are made against these
technologies then SSL is no longer secure.

Attacks against a specific communications session can be made by
recording the session, and then spending some large number of compute
cycles to crack either the session key or the RSA public key until the
communication can be seen in the clear. This approach is easier than
cracking the cryptographic technologies for all possible messages. Note
that SSL tries to make the cost of such of an attack greater than the

benefits gained from a successful attack, thus making it a waste of
money/time to perform such an attack.

There have been many books [9] and papers [10] written on cryptography.
This document does not attempt to reference them all.

D.2 Clear Text Attack

A clear text attack is done when the attacker has an idea of what kind of
message is being sent using encryption. The attacker can generate a data
base whose keys are the encrypted value of the known text (or clear text),
and whose values are the session cipher key (we call this a "dictionary").
Once this data base is constructed, a simple lookup function identifies the
session key that goes with a particular encrypted value. Once the session
key is known, the entire message stream can be decrypted. Custom
hardware can be used to make this cost effective and very fast.

Because of the very nature of SSL clear text attacks are possible. For
example, the most common byte string sent by an HTTP client application
to an HTTP server is "GET". SSL attempts to address this attack by using
large session cipher keys. First, the client generates a key which is larger
than allowed by export, and sends some of it in the clear to the server (this
is allowed by United States government export rules). The clear portion of
the key concatenated with the secret portion make a key which is very
large (for RC4, exactly 128 bits).

The way that this "defeats" a clear text attack is by making the amount of
custom hardware needed prohibitively large. Every bit added to the length
of the session cipher key increases the dictionary size by a factor of 2. By
using a 128 bit session cipher key length the size of the dictionary required
is beyond the ability of anyone to fabricate (it would require more atoms to
construct than exist in the entire universe). Even if a smaller dictionary is
to be used, it must first be generated using the clear key bits. This is a time
consumptive process and also eliminates many possible custom hardware
architectures (e.g. static prom arrays).

The second way that SSL attacks this problem is by using large key
lengths when permissible (e.g. in the non-export version). Large key sizes
require larger dictionaries (just one more bit of key size doubles the size of
the dictionary). SSL attempts to use keys that are 128 bits in length.

Note that the consequence of the SSL defense is that a brute force attack
becomes the cheapest way to attack the key. Brute force attacks have well
known space/time tradeoffs and so it becomes possible to define a cost of
the attack. For the 128 bit secret key, the known cost is essentially infinite.
For the 40 bit secret key, the cost is much smaller, but still outside the
range of the "random hacker".

D.3 Replay

The replay attack is simple. A bad-guy records a communication session
between a client and server. Later, it reconnects to the server, and plays
back the previously recorded client messages. SSL defeats this attack
using a "nonce" (the connection-id) which is "unique" to the connection. In
theory the bad-guy cannot predict the nonce in advance as it is based on a

set of random events outside the bad-guys control, and therefore the bad-guy cannot respond properly to server requests.

A bad-guy with large resources can record many sessions between a client and a server, and attempt to choose the right session based on the nonce the server sends initially in its SERVER-HELLO message. However, SSL nonces are at least 128 bits long, so a bad-guy would need to record approximately 2^64 nonces to even have a 50% chance of choosing the right session. This number is sufficiently large that one cannot economically construct a device to record 2^64 messages, and therefore the odds are overwhelmingly against the replay attack ever being successful.

D.4 The Man In The Middle

The man in the middle attack works by having three people in a communications session: the client, the server, and the bad guy. The bad guy sits between the client and the server on the network and intercepts traffic that the client sends to the server, and traffic that the server sends to
the client.

The man in the middle operates by pretending to be the real server to the client. With SSL this attack is impossible because of the usage of server certificates. During the security connection handshake the server is required to provide a certificate that is signed by a certificate authority. Contained in the certificate is the server's public key as well as its name and the name of the certificate issuer. The client verifies the certificate by first checking the signature and then verifying that the name of the issuer is somebody that the client trusts.

In addition, the server must encrypt something with the private key that goes with the public key mentioned in the certificate. This in essence is a single pass "challenge response" mechanism. Only a server that has both the certificate and the private key can respond properly to the challenge.

If the man in the middle provides a phony certificate, then the signature check will fail. If the certificate provided by the bad guy is legitimate, but for the bad guy instead of for the real server, then the signature will pass but the name check will fail (note that the man in the middle cannot forge certificates without discovering a certificate authority's private key).

Finally, if the bad guy provides the real server's certificate then the signature check will pass and the name check will pass. However, because the bad guy does not have the real server's private key, the bad guy cannot properly encode the response to the challenge code, and this check will fail.

In the unlikely case that a bad guy happens to guess the response code to the challenge, the bad guy still cannot decrypt the session key and therefore cannot examine the encrypted data.

Appendix E: Terms

Application Protocol
An application protocol is a protocol that normally layers directly on
top of TCP/IP. For example: HTTP, TELNET, FTP, and SMTP.

Authentication
Authentication is the ability of one entity to determine the identity of
another entity. Identity is defined by this document to mean the
binding between a public key and a name and the implicit ownership
of the corresponding private key.

Bulk Cipher
This term is used to describe a cryptographic technique with certain
performance properties. Bulk ciphers are used when large quantities of
data are to be encrypted/decrypted in a timely manner. Examples
include RC2, RC4, and IDEA.

Client
In this document client refers to the application entity that is initiates a
connection to a server.

CLIENT-READ-KEY
The session key that the client uses to initialize the client read cipher.
This key has the same value as the SERVER-WRITE-KEY.

CLIENT-WRITE-KEY
The session key that the client uses to initialize the client write cipher.
This key has the same value as the SERVER-READ-KEY.

MASTER-KEY
The master key that the client and server use for all session key
generation. The CLIENT-READ-KEY, CLIENT-WRITE-KEY,
SERVER-READ-KEY and SERVER-WRITE-KEY are generated
from the MASTER-KEY.

MD2
MD2 [8] is a hashing function that converts an arbitrarily long data
stream into a digest of fixed size. This function predates MD5 [7]
which is viewed as a more robust hash function [9].

MD5
MD5 [7] is a hashing function that converts an arbitrarily long data
stream into a digest of fixed size. The function has certain properties
that make it useful for security, the most important of which is it's
inability to be reversed.

Nonce
A randomly generated value used to defeat "playback" attacks. One
party randomly generates a nonce and sends it to the other party. The
receiver encrypts it using the agreed upon secret key and returns it to
the sender. Because the nonce was randomly generated by the sender
this defeats playback attacks because the replayer can't know in
advance the nonce the sender will generate. The receiver denies
connections that do not have the correctly encrypted nonce.

Non-repudiable Information Exchange
When two entities exchange information it is sometimes valuable to
have a record of the communication that is non-repudiable. Neither
party can then deny that the information exchange occurred. Version 2
of the SSL protocol does not support Non-repudiable information
exchange.

Public Key Encryption
Public key encryption is a technique that leverages asymmetric ciphers.
A public key system consists of two keys: a public key and a private
key. Messages encrypted with the public key can only be decrypted
with the associated private key. Conversely, messages encrypted with
the private key can only be decrypted with the public key. Public key
encryption tends to be extremely compute intensive and so is not
suitable as a bulk cipher.

Privacy
Privacy is the ability of two entities to communicate without fear of
eavesdropping. Privacy is often implemented by encrypting the
communications stream between the two entities.

RC2, RC4
Proprietary bulk ciphers invented by RSA (There is no good reference
to these as they are unpublished works; however, see [9]). RC2 is
block cipher and RC4 is a stream cipher.

Server
The server is the application entity that responds to requests for
connections from clients. The server is passive, waiting for requests
from clients.

Session cipher
A session cipher is a "bulk" cipher that is capable of encrypting or
decrypting arbitrarily large amounts of data. Session ciphers are used
primarily for performance reasons. The session ciphers used by this
protocol are symmetric. Symmetric ciphers have the property of using
a single key for encryption and decryption.

Session identifier
A session identifier is a random value generated by a client that
identifies itself to a particular server. The session identifier can be
thought of as a handle that both parties use to access a recorded secret
key (in our case a session key). If both parties remember the session
identifier then the implication is that the secret key is already known
and need not be negotiated.

Session key
The key to the session cipher. In SSL there are four keys that are called
session keys: CLIENT-READ-KEY, CLIENT-WRITE-KEY,
SERVER-READ-KEY, and SERVER-WRITE-KEY.

SERVER-READ-KEY
The session key that the server uses to initialize the server read cipher.
This key has the same value as the CLIENT-WRITE-KEY.

SERVER-WRITE-KEY
The session key that the server uses to initialize the server write cipher.
This key has the same value as the CLIENT-READ-KEY.

Symmetric Cipher
A symmetric cipher has the property that the same key can be used for
decryption and encryption. An asymmetric cipher does not have this
behavior. Some examples of symmetric ciphers: IDEA, RC2, RC4.

References

[1] CCITT. Recommendation X.208: "Specification of Abstract Syntax
Notation One (ASN.1). 1988.

[2] CCITT. Recommendation X.209: "Specification of Basic Encoding
Rules for Abstract Syntax Notation One (ASN.1). 1988.

[3] CCITT. Recommendation X.509: "The Directory - Authentication
Framework". 1988.

[4] CCITT. Recommendation X.520: "The Directory - Selected Attribute
Types". 1988.

[5] RSA Laboratories. PKCS #1: RSA Encryption Standard, Version 1.5,
November 1993.

[6] RSA Laboratories. PKCS #6: Extended-Certificate Syntax Standard,
Version 1.5, November 1993.

[7] R. Rivest. RFC 1321: The MD5 Message Digest Algorithm. April
1992.

[8] R. Rivest. RFC 1319: The MD2 Message Digest Algorithm. April
1992.

[9] B. Schneier. Applied Cryptography: Protocols, Algorithms, and Source
Code in C, Published by John Wiley & Sons, Inc. 1994.

[10] M. Abadi and R. Needham. Prudent engineering practice for
cryptographic protocols. 1994.

Patent Statement

This version of the SSL protocol relies on the use of patented public key
encryption technology for authentication and encryption. The Internet
Standards Process as defined in RFC 1310 requires a written statement
from the Patent holder that a license will be made available to applicants
under reasonable terms and conditions prior to approving a specification as
a Proposed, Draft or Internet Standard.

The Massachusetts Institute of Technology and the Board of Trustees of
the Leland Stanford Junior University have granted Public Key Partners
(PKP) exclusive sub-licensing rights to the following patents issued in the

United States, and all of their corresponding foreign patents:

Cryptographic Apparatus and Method ("Diffie-Hellman")
          No. 4,200,770

Public Key Cryptographic Apparatus and Method ("Hellman-Merkle")
          No. 4,218,582

Cryptographic Communications System and Method ("RSA")
          No. 4,405,829

Exponential Cryptographic Apparatus and Method ("Hellman-Pohlig")
          No. 4,424,414

These patents are stated by PKP to cover all known methods of practicing
the art of Public Key encryption, including the variations collectively
known as ElGamal.

Public Key Partners has provided written assurance to the Internet Society
that parties will be able to obtain, under reasonable, nondiscriminatory
terms, the right to use the technology covered by these patents. This
assurance is documented in RFC 1170 titled "Public Key Standards and
Licenses". A copy of the written assurance dated April 20, 1990, may be
obtained from the Internet Assigned Number Authority (IANA).

The Internet Society, Internet Architecture Board, Internet Engineering
Steering Group and the Corporation for National Research Initiatives take
no position on the validity or scope of the patents and patent applications,
nor on the appropriateness of the terms of the assurance. The Internet
Society and other groups mentioned above have not made any
determination as to any other intellectual property rights which may apply
to the practice of this standard. Any further consideration of these matters
is the user's own responsibility.

Security Considerations

This entire document is about security.

Author's Address

Kipp E.B. Hickman
Netscape Communications Corp.
501 East Middlefield Rd.
Mountain View, CA 94043
kipp@netscape.com