SSL 0.2 Protocol Specification

This Protocol Specification Was Revised On November 29th, 1994:

- a fundamental correction to the client-certificate authentication protocol,
- the removal of the username/password messages,
- corrections in some of the cryptographic terminology,
- the addition of a MAC to the messages [see section 1.2],
- the allowance for different kinds of message digest algorithms.

This Document Was Revised On December 22nd, 1994:

- The spec now defines the order the clear key data and secret key data are combined to produce the master key.
- The spec now explicitly states the size of the MAC instead of making the reader figure it out.
- The spec is more clear on the actual values used to produce the session read and write keys.
- The spec is more clear on how many bits of the session key are used after they are produced from the hash function.

This Document Was Revised On January 17th, 1995:

- Defined the category to be informational.
- Clarified ordering of data elements in various places.
- Defined DES-CBC cipher kind and key construction.
- Defined DES-EDE3-CBC cipher kind and key construction.

- Fixed bug in definition of CIPHER-CHOICE in CLIENT-MASTER-KEY message. The previous spec erroneously indicated that the CIPHER-CHOICE was an index into the servers CIPHER-SPECS-DATA array, when it was actually supposed to be the CIPHER-KIND value chosen by the client.
- Clarified the values of the KEY-ARG-DATA.

THIS DOCUMENT WAS REVISED ON FEBRUARY 9TH, 1995:

The spec has been clarified to indicate the byte order of sequence numbers when they are being applied to the MAC hash function.

- The spec now defines the acceptable length range of the CONNECTION-ID parameter (sent by the server in the SERVER-HELLO message).
- Simplified the specification of the CIPHER-KIND data. The spec language has been changed yet the format remains compatible with all existing implementations. The CIPHER-KIND information is now a three byte value which defines the type of cipher and the length of the key. The key length is no longer separable from the CIPHER-KIND.
- Explained how the KEY-ARG-DATA is retained with the SESSION-ID when the session-identifier cache is used.

Experimental Kipp E.B. Hickman
Request For Comments: XXXX Netscape Communications Corp.
Category: Informational Last Update: Feb. 9th, 1995

The SSL Protocol

Status of this Memo

This is a **DRAFT** specification.

This RFC specifies a security protocol for the Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

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.

Motivation

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.
- The channel is authenticated. The server endpoint of the conversation is always authenticated, while the client endpoint is optionally authenticated.
- The channel is reliable. The message transport includes a message integrity check (using a MAC).

1. SSL Record Protocol Specification

1.1 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.

1.2 SSL Record Data Format

```
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

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.

2. SSL Handshake Protocol Specification

2.1 SSL Handshake Protocol Flow

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.

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. 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.

Phase 2

The second phase is the 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 something from the client and send a request. 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).

When a party is done authenticating the other party, it sends its **finished** message. For the client, the `CLIENT-`

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.