

The SSL Protocol  
Version 3.0

<draft-ietf-tls-ssl-version3-00.txt>

Status of this memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet- Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or made obsolete by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as work in progress.

To learn the current status of any Internet-Draft, please check the lid-abstracts.txt listing contained in the Internet Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

Abstract

This document specifies Version 3.0 of the Secure Sockets Layer (SSL V3.0) protocol, a security protocol that provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

## Table of Contents

	Status of this memo	1
	Abstract	1
	Table of Contents	2
1.	Introduction	4
2.	Goals	4
3.	Goals of this document	5
4.	Presentation language	5
4.1	Basic block size	5
4.2	Miscellaneous	6
4.3	Vectors	6
4.4	Numbers	7
4.5	Enumerateds	7
4.6	Constructed types	8
4.6.1	Variants	8
4.7	Cryptographic attributes	9
4.8	Constants	10
5.	SSL protocol	10
5.1	Session and connection states	10
5.2	Record layer	12
5.2.1	Fragmentation	12
5.2.2	Record compression and decompression	13
5.2.3	Record payload protection and the CipherSpec	13
5.2.3.1	Null or standard stream cipher	14
5.2.3.2	CBC block cipher	15
5.3	Change cipher spec protocol	16
5.4	Alert protocol	16
5.4.1	Closure alerts	17
5.4.2	Error alerts	17
5.5	Handshake protocol overview	18
5.6	Handshake protocol	20
5.6.1	Hello messages	21
5.6.1.1	Hello request	21
5.6.1.2	Client hello	21
5.6.1.3	Server hello	24
5.6.2	Server certificate	25
5.6.3	Server key exchange message	25
5.6.4	Certificate request	27
5.6.5	Server hello done	27
5.6.6	Client certificate	28
5.6.7	Client key exchange message	28
5.6.7.1	RSA encrypted premaster secret message	28
5.6.7.2	FORTEZZA key exchange message	29
5.6.7.3	Client Diffie-Hellman public value	30
5.6.8	Certificate verify	30
5.6.9	Finished	31
5.7	Application data protocol	32
6.	Cryptographic computations	32
6.1	Asymmetric cryptographic computations	32
6.1.1	RSA	32
6.1.2	Diffie-Hellman	33
6.1.3	FORTEZZA	33

6.2	Symmetric cryptographic calculations and the CipherSpec	33
6.2.1	The master secret	33
6.2.2	Converting the master secret into keys and MAC	33
6.2.2.1	Export key generation example	35
A.	Protocol constant values	36
A.1	Reserved port assignments	36
A.1.1	Record layer	36
A.2	Change cipher specs message	37
A.3	Alert messages	37
A.4	Handshake protocol	37
A.4.1	Hello messages	38
A.4.2	Server authentication and key exchange messages	39
A.5	Client authentication and key exchange messages	40
A.5.1	Handshake finalization message	41
A.6	The CipherSuite	41
A.7	The CipherSpec	42
B.	Glossary	44
C.	CipherSuite definitions	47
D.	Implementation Notes	49
D.1	Temporary RSA keys	49
D.2	Random Number Generation and Seeding	49
D.3	Certificates and authentication	50
D.4	CipherSuites	50
D.5	FORTEZZA	50
D.5.1	Notes on use of FORTEZZA hardware	50
D.5.2	FORTEZZA Ciphersuites	51
D.5.3	FORTEZZA Session resumption	51
E.	Version 2.0 Backward Compatibility	52
E.1	Version 2 client hello	52
E.2	Avoiding man-in-the-middle version rollback	53
F.	Security analysis	55
F.1	Handshake protocol	55
F.1.1	Authentication and key exchange	55
F.1.1.1	Anonymous key exchange	55
F.1.1.2	RSA key exchange and authentication	56
F.1.1.3	Diffie-Hellman key exchange with authentication	57
F.1.1.4	FORTEZZA	57
F.1.2	Version rollback attacks	57
F.1.3	Detecting attacks against the handshake protocol	58
F.1.4	Resuming sessions	58
F.1.5	MD5 and SHA	58
F.2	Protecting application data	59
F.3	Final notes	59
G.	Patent Statement	60
	References	61
	Authors	62

## 1. Introduction

The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. The protocol is composed of two layers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP[TCP]), is the SSL Record Protocol. The SSL Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the SSL Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. One advantage of SSL is that it is application protocol independent. A higher level protocol can layer on top of the SSL Protocol transparently. The SSL protocol provides connection security that has three basic properties:

- The connection is private. Encryption is used after an initial handshake to define a secret key. Symmetric cryptography is used for data encryption (e.g., DES[DES], RC4[RC4], etc.)
- The peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA[RSA], DSS[DSS], etc.).
- The connection is reliable. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA, MD5, etc.) are used for MAC computations.

## 2. Goals

The goals of SSL Protocol v3.0, in order of their priority, are:

1. Cryptographic security  
SSL should be used to establish a secure connection between two parties.
2. Interoperability  
Independent programmers should be able to develop applications utilizing SSL 3.0 that will then be able to successfully exchange cryptographic parameters without knowledge of one another's code.

Note: It is not the case that all instances of SSL (even in the same application domain) will be able to successfully connect. For instance, if the server supports a particular hardware token, and the client does not have access to such a token, then the connection will not succeed.

3. Extensibility  
SSL seeks to provide a framework into which new public key and bulk encryption methods can be incorporated as necessary. This will also accomplish two sub-goals: to prevent the need

to create a new protocol (and risking the introduction of possible new weaknesses) and to avoid the need to implement an entire new security library.

#### 4. Relative efficiency

Cryptographic operations tend to be highly CPU intensive, particularly public key operations. For this reason, the SSL protocol has incorporated an optional session caching scheme to reduce the number of connections that need to be established from scratch. Additionally, care has been taken to reduce network activity.

### 3. Goals of this document

The SSL Protocol Version 3.0 Specification is intended primarily for readers who will be implementing the protocol and those doing cryptographic analysis of it. The spec has been written with this in mind, and it is intended to reflect the needs of those two groups. For that reason, many of the algorithm-dependent data structures and rules are included in the body of the text (as opposed to in an Appendix), providing easier access to them.

This document is not intended to supply any details of service definition nor interface definition, although it does cover select areas of policy as they are required for the maintenance of solid security.

### 4. Presentation language

This document deals with the formatting of data in an external representation. The following very basic and somewhat casually defined presentation syntax will be used. The syntax draws from several sources in its structure. Although it resembles the programming language "C" in its syntax and XDR [XDR] in both its syntax and intent, it would be risky to draw too many parallels. The purpose of this presentation language is to document SSL only, not to have general application beyond that particular goal.

#### 4.1 Basic block size

The representation of all data items is explicitly specified. The basic data block size is one byte (i.e. 8 bits). Multiple byte data items are concatenations of bytes, from left to right, from top to bottom. From the bytestream a multi-byte item (a numeric in the example) is formed (using C notation) by:

```
value = (byte[0] << 8*(n-1)) | (byte[1] << 8*(n-2)) | ...
| byte[n-1];
```

This byte ordering for multi-byte values is the commonplace network byte order or big endian format.

# 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.