SSL/TLS Strong Encryption: An Introduction

The nice thing about standards is that there are so many to choose from. And if you really don't like all the standards you just have to wait another year until the one arises you are looking for.

-- A. Tanenbaum, "Introduction to Computer Networks"

As an introduction this chapter is aimed at readers who are familiar with the Web, HTTP, and Apache, but are not security experts. It is not intended to be a definitive guide to the SSL protocol, nor does it discuss specific techniques for managing certificates in an organization, or the important legal issues of patents and import and export restrictions. Rather, it is intended to provide a common background to mod_ssl users by pulling together various concepts, definitions, and examples as a starting point for further exploration.

The presented content is mainly derived, with permission by the author, from the article Introducing SSL and Certificates using SSLeay[1] from Frederick J. Hirsch[2], of The Open Group Research Institute, which was published in Web Security: A Matter of Trust[3], World Wide Web Journal, Volume 2, Issue 3, Summer 1997. Please send any postive feedback to Frederick Hirsch (the original article author) and all negative feedback to Ralf S. Engelschall (the `mod_ssl` author).

Topics

Cryptographic Techniques

Understanding SSL requires an understanding of cryptographic algorithms, message digest functions (aka. one-way or hash functions), and digital signatures. These techniques are the subject of entire books (see for instance [AC96]) and provide the basis for privacy, integrity, and authentication.

Cryptographic Algorithms

Suppose Alice wants to send a message to her bank to transfer some money. Alice would like the message to be private, since it will include information such as her account number and transfer amount. One solution is to use a cryptographic algorithm, a technique that would transform her message into an encrypted form, unreadable except by those it is intended for. Once in this form, the message may only be interpreted through the use of a secret key. Without the key the message is useless: good cryptographic algorithms make it so difficult for intruders to decode the original text that it isn't worth their effort.

There are two categories of cryptographic algorithms: conventional and public key.

Conventional cryptography
: also known as symmetric cryptography, requires the sender and receiver to share a key: a secret piece of information that may be used to encrypt or decrypt a message. If this key is secret, then nobody other than the sender or receiver may read the message. If Alice and the bank know a secret key, then they may send each other private messages. The task of privately choosing a key before communicating, however, can be problematic.

Public key cryptography
: also known as asymmetric cryptography, solves the key exchange problem by defining an algorithm which uses two keys, each of which may be used to encrypt a message. If one key is used to encrypt a message then the other must be used to decrypt it. This makes it possible to

> receive secure messages by simply publishing one key (the public key) and keeping the other secret (the private key).

Anyone may encrypt a message using the public key, but only the owner of the private key will be able to read it. In this way, Alice may send private messages to the owner of a key-pair (the bank), by encrypting it using their public key. Only the bank will be able to decrypt it.

Message Digests

Although Alice may encrypt her message to make it private, there is still a concern that someone might modify her original message or substitute it with a different one, in order to transfer the money to themselves, for instance. One way of guaranteeing the integrity of Alice's message is to create a concise summary of her message and send this to the bank as well. Upon receipt of the message, the bank creates its own summary and compares it with the one Alice sent. If they agree then the message was received intact.

A summary such as this is called a *message digest*, *one-way function* or *hash function*. Message digests are used to create short, fixed-length representations of longer, variable-length messages. Digest algorithms are designed to produce unique digests for different messages. Message digests are designed to make it too difficult to determine the message from the digest, and also impossible to find two different messages which create the same digest -- thus eliminating the possibility of substituting one message for another while maintaining the same digest.

Another challenge that Alice faces is finding a way to send the digest to the bank securely; when this is achieved, the integrity of the associated message is assured. One way to to this is to include the digest in a digital signature.

Digital Signatures

When Alice sends a message to the bank, the bank needs to ensure that the message is really from her, so an intruder does not request a transaction involving her account. A *digital signature*, created by Alice and included with the message, serves this purpose.

Digital signatures are created by encrypting a digest of the message, and other information (such as a sequence number) with the sender's private key. Though anyone may *decrypt* the signature using the public key, only the signer knows the private key. This means that only they may have signed it. Including the digest in the signature means the signature is only good for that message; it also ensures the integrity of the message since no one can change the digest and still sign it.

To guard against interception and reuse of the signature by an intruder at a later date, the signature contains a unique sequence number. This protects the bank from a fraudulent claim from Alice that she did not send the message -- only she could have signed it (non-repudiation).

Certificates

Although Alice could have sent a private message to the bank, signed it, and ensured the integrity of the message, she still needs to be sure that she is really communicating with the bank. This means that she needs to be sure that the public key she is using corresponds to the bank's private key. Similarly, the bank also needs to verify that the message signature really corresponds to Alice's signature.

If each party has a certificate which validates the other's identity, confirms the public key, and is signed by a trusted agency, then they both will be assured that they are communicating with whom they think they are. Such a trusted agency is called a *Certificate Authority*, and certificates are used for authentication.

Certificate Contents

A certificate associates a public key with the real identity of an individual, server, or other entity, known as the subject. As shown in Table 1, information about the subject includes identifying information (the distinguished name), and the public key. It also includes the identification and signature of the Certificate Authority that issued the certificate, and the period of time during which the certificate is valid. It may have additional information (or extensions) as well as administrative information for the Certificate Authority's use, such as a serial number.

Table 1: Certificate Information

Subject	Distinguished Name, Public Key
Issuer	Distinguished Name, Signature
Period of Validity	Not Before Date, Not After Date
Administrative Information	Version, Serial Number
Extended Information	Basic Contraints, Netscape Flags, etc.

A distinguished name is used to provide an identity in a specific context -- for instance, an individual might have a personal certificate as well as one for their identity as an employee. Distinguished names are defined by the X.509 standard [X509], which defines the fields, field names, and abbreviations used to refer to the fields (see Table 2).

Table 2: Distinguished Name Information

DN Field	Abbrev.	Description	Example
Common Name	CN	Name being certified	CN=Joe Average
Organization or Company	O	Name is associated with this organization	O=Snake Oil, Ltd.
Organizational Unit	OU	Name is associated with this organization unit, such as a department	OU=Research Institute
City/Locality	L	Name is located in this City	L=Snake City
State/Province	ST	Name is located in this State/Province	ST=Desert
Country	C	Name is located in this Country (ISO code)	C=XZ

A Certificate Authority may define a policy specifying which distinguished field names are optional, and which are required. It may also place requirements upon the field contents, as may users of certificates. As an example, a Netscape browser requires that the Common Name for a certificate representing a server has a name which matches a wildcard pattern for the domain name of that server, such as `*.snakeoil.com`.

The binary format of a certificate is defined using the ASN.1 notation [X208] [PKCS]. This notation defines how to specify the contents, and encoding rules define how this information is translated into binary form. The binary encoding of the certificate is defined using Distinguished Encoding Rules (DER), which are based on the more general Basic Encoding Rules (BER). For those transmissions which cannot handle binary, the binary form may be translated into an ASCII form by using Base64 encoding [MIME]. This encoded version is called PEM encoded (the name comes from "Privacy Enhanced Mail"), when placed between begin and end delimiter lines as illustrated in the following example.

Example of a PEM-encoded certificate (snakeoil.crt)

```
-----BEGIN CERTIFICATE-----
```

```
MIIC7jCCAlegAwIBAgIBATANBgkqhkiG9w0BAQQFADCBqTELMAkGA1UEBhMCWFkx
FTATBgNVBAgTDFNuYWtlIERlc2VydDETMBEGA1UEBxMKU25ha2UgVG93bjEXMBUG
A1UEChMOU25ha2UgT2lsLCBMdGQxHjAcBgNVBAsTFUNlcnRpZmljYXRlIEF1dGhv
cml0eTEVMBMGA1UEAxMMU25ha2UgT2lsIENBMR4wHAYJKoZIhvcNAQkBFg9jYUBz
bmFrZW9pbC5kb20wHhcNOTgxMDIxMDg1ODM2WhcNOTkxMDIxMDg1ODM2WjCBpzEL
MAkGA1UEBhMCWFkxFTATBgNVBAgTDFNuYWtlIERlc2VydDETMBEGA1UEBxMKU25h
a2UgVG93bjEXMBUGA1UEChMOU25ha2UgT2lsLCBMdGQxFzAVBgNVBAsTDldlYnNl
cnZlciBUZWFtMRkwFwYDVQQDExB3d3cuc25ha2VvaWwuZG9tMR8wHQYJKoZIhvcN
AQkBFhB3d3dAc25ha2VvaWwuZG9tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQDH9Ge/s2zcH+da+rPTx/DPRp3xGjHZ4GG6pCmvADIEtBtKBFAcZ64n+Dy7Np8b
vKR+yy5DGQiijsH1D/j8HlGE+q4TZ8OFk7BNBFazHxFbYI4OKMiCxdKzdif1yfaa
lWoANFlAzlSdbxeGVHoT0K+gT5w3UxwZKv2DLbCTzLZyPwIDAQABoyYwJDAPBgNV
HRMECDAGAQH/AgEAMBEGCWCGSAGG+EIBAQQEAwIAQDANBgkqhkiG9w0BAQQFAAOB
gQAZUIHAL4D09oE6Lv2k56Gp38OBDuILvwLg1v1KL8mQR+KFjghCrtpqaztZqcDt
2q2QoyulCgSzHbEGmi0EsdkPfg6mp0penssIFePYNI+/8u9HT4LuKMJX15hxBam7
dUHzICxBVC1lnHyYGjDuAMhe396lYAn8bCld1/L4NMGBCQ==
-----END CERTIFICATE-----
```

Certificate Authorities

By first verifying the information in a certificate request before granting the certificate, the Certificate Authority assures the identity of the private key owner of a key-pair. For instance, if Alice requests a personal certificate, the Certificate Authority must first make sure that Alice really is the person the certificate request claims.

Certificate Chains

A Certificate Authority may also issue a certificate for another Certificate Authority. When examining a certificate, Alice may need to examine the certificate of the issuer, for each parent Certificate Authority, until reaching one which she has confidence in. She may decide to trust only certificates with a limited chain of issuers, to reduce her risk of a "bad" certificate in the chain.

Creating a Root-Level CA

As noted earlier, each certificate requires an issuer to assert the validity of the identity of the certificate subject, up to the top-level Certificate Authority (CA). This presents a problem: Since this is who vouches for the certificate of the top-level authority, which has no issuer? In this unique case, the certificate is "self-signed", so the issuer of the certificate is the same as the subject. As a result, one must exercise extra care in trusting a self-signed certificate. The wide publication of a public key by the root authority reduces the risk in trusting this key -- it would be obvious if someone else publicized a key claiming to be the authority. Browsers are preconfigured to trust well-known certificate authorities.

A number of companies, such as Thawte[6] and VeriSign[7] have established themselves as Certificate Authorities. These companies provide the following services:

- Verifying certificate requests
- Processing certificate requests
- Issuing and managing certificates

It is also possible to create your own Certificate Authority. Although risky in the Internet environment, it may be useful within an Intranet where the organization can easily verify the identities of individuals and servers.

Certificate Management

Establishing a Certificate Authority is a responsibility which requires a solid administrative, technical, and management framework. Certificate Authorities not only issue certificates, they also manage them -- that is, they determine how long certificates are valid, they renew them, and they keep lists of certificates that have already been issued but are no longer valid (Certificate Revocation Lists, or CRLs). Say Alice is entitled to a certificate as an employee of a company. Say too, that the certificate needs to be revoked when Alice leaves the company. Since certificates are objects that get passed around, it is impossible to tell from the certificate alone that it has been revoked. When examining certificates for validity, therefore, it is necessary to contact the issuing Certificate Authority to check CRLs -- this is not usually an automated part of the process.

Note

If you use a Certificate Authority that is not configured into browsers by default, it is necessary to load the Certificate Authority certificate into the browser, enabling the browser to validate server certificates signed by that Certificate Authority. Doing so may be dangerous, since once loaded, the browser will accept all certificates signed by that Certificate Authority.

Secure Sockets Layer (SSL)

The Secure Sockets Layer protocol is a protocol layer which may be placed between a reliable connection-oriented network layer protocol (e.g. TCP/IP) and the application protocol layer (e.g. HTTP). SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity, and encryption for privacy.

The protocol is designed to support a range of choices for specific algorithms used for cryptography, digests, and signatures. This allows algorithm selection for specific servers to be made based on legal, export or other concerns, and also enables the protocol to take advantage of new algorithms. Choices are negotiated between client and server at the start of establishing a protocol session.

Table 4: Versions of the SSL protocol

Version	Source	Description	Browser Support
SSL v2.0	Vendor Standard (from Netscape Corp.) [SSL2]	First SSL protocol for which implementations exists	- NS Navigator 1.x/2.x - MS IE 3.x - Lynx/2.8+OpenSSL
SSL v3.0	Expired Internet Draft (from Netscape Corp.) [SSL3]	Revisions to prevent specific security attacks, add non-RSA ciphers, and support for certificate chains	- NS Navigator 2.x/3.x/4.x - MS IE 3.x/4.x - Lynx/2.8+OpenSSL
TLS v1.0	Proposed Internet Standard (from IETF) [TLS1]	Revision of SSL 3.0 to update the MAC layer to HMAC, add block padding for block ciphers, message order standardization and more alert messages.	- Lynx/2.8+OpenSSL

There are a number of versions of the SSL protocol, as shown in Table 4. As noted there, one of the benefits in SSL 3.0 is that it adds support of certificate chain loading. This feature allows a server to pass a server certificate along with issuer certificates to the browser. Chain loading also permits the browser to validate the server certificate, even if Certificate Authority certificates are not installed for the intermediate issuers, since they are included in the certificate chain. SSL 3.0 is the basis for the Transport Layer Security [TLS] protocol standard, currently in development by the Internet Engineering Task Force (IETF).

Session Establishment

The SSL session is established by following a handshake sequence between client and server, as shown in Figure 1. This sequence may vary, depending on whether the server is configured to provide a server certificate or request a client certificate. Though cases exist where additional handshake steps are required for management of cipher information, this article summarizes one common scenario: see the SSL specification for the full range of possibilities.

Note

Once an SSL session has been established it may be reused, thus avoiding the performance penalty of repeating the many steps needed to start a session. For this the server assigns each SSL session a unique session identifier which is cached in the server and which the client can use on forthcoming connections to reduce the handshake (until the session identifer expires in the cache of the server).

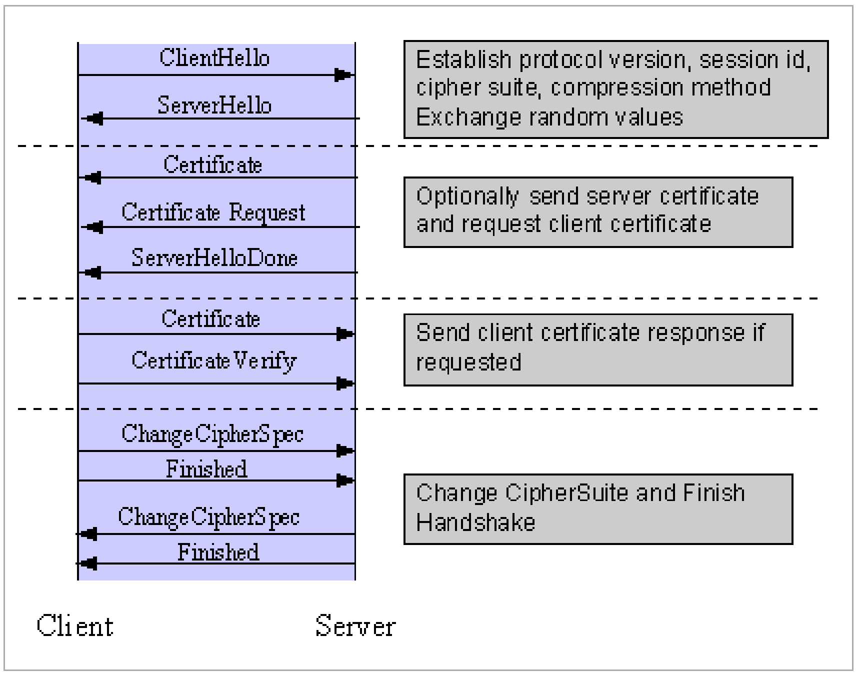
ClientHello
ServerHello
Establish protocol version, session id, cipher suite, compression method Exchange random values
Certificate
Certificate Request
ServerHelloDone
Optionally send server certificate and request client certificate
Certificate
Certificate Verify
Send client certificate response if requested
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished
Change CipherSuite and Finish Handshake
Client
Server

***Figure 1**: Simplified SSL Handshake Sequence*

The elements of the handshake sequence, as used by the client and server, are listed below:

1. Negotiate the Cipher Suite to be used during data transfer
2. Establish and share a session key between client and server
3. Optionally authenticate the server to the client
4. Optionally authenticate the client to the server

The first step, Cipher Suite Negotiation, allows the client and server to choose a Cipher Suite supportable by both of them. The SSL3.0 protocol specification defines 31 Cipher Suites. A Cipher Suite is defined by the following components:

- Key Exchange Method
- Cipher for Data Transfer
- Message Digest for creating the Message Authentication Code (MAC)

These three elements are described in the sections that follow.

Key Exchange Method

The key exchange method defines how the shared secret symmetric cryptography key used for application data transfer will be agreed upon by client and server. SSL 2.0 uses RSA key exchange only, while SSL 3.0 supports a choice of key exchange algorithms including the RSA key exchange when certificates are used, and Diffie-Hellman key exchange for exchanging keys without certificates and without prior communication between client and server.

One variable in the choice of key exchange methods is digital signatures -- whether or not to use them, and if so, what kind of signatures to use. Signing with a private key provides assurance against a man-in-the-middle-attack during the information exchange used in generating the shared key [AC96, p516].

Cipher for Data Transfer

SSL uses the conventional cryptography algorithm (symmetric cryptography) described earlier for encrypting messages in a session. There are nine choices, including the choice to perform no encryption:

- No encryption
- Stream Ciphers
  - RC4 with 40-bit keys
  - RC4 with 128-bit keys
- CBC Block Ciphers
  - RC2 with 40 bit key
  - DES with 40 bit key
  - DES with 56 bit key
  - Triple-DES with 168 bit key
  - Idea (128 bit key)
  - Fortezza (96 bit key)

Here "CBC" refers to Cipher Block Chaining, which means that a portion of the previously encrypted cipher text is used in the encryption of the current block. "DES" refers to the Data Encryption Standard [AC96, ch12], which has a number of variants (including DES40 and 3DES_EDE). "Idea" is one of the best and cryptographically strongest available algorithms, and "RC2" is a proprietary algorithm from RSA DSI [AC96, ch13].

Digest Function

The choice of digest function determines how a digest is created from a record unit. SSL supports the following:

- No digest (Null choice)
- MD5, a 128-bit hash
- Secure Hash Algorithm (SHA-1), a 160-bit hash

The message digest is used to create a Message Authentication Code (MAC) which is encrypted with the message to provide integrity and to prevent against replay attacks.

Handshake Sequence Protocol

The handshake sequence uses three protocols:

- The *SSL Handshake Protocol* for performing the client and server SSL session establishment.
- The *SSL Change Cipher Spec Protocol* for actually establishing agreement on the Cipher Suite for the session.
- The *SSL Alert Protocol* for conveying SSL error messages between client and server.

These protocols, as well as application protocol data, are encapsulated in the *SSL Record Protocol*, as shown in Figure 2. An encapsulated protocol is transferred as data by the lower layer protocol, which does not examine the data. The encapsulated protocol has no knowledge of the underlying protocol.

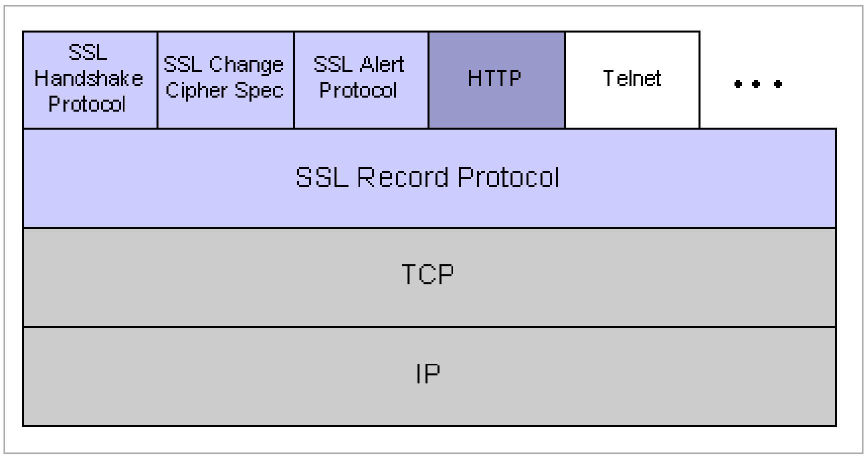

***Figure 2**: SSL Protocol Stack*

The encapsulation of SSL control protocols by the record protocol means that if an active session is renegotiated the control protocols will be transmitted securely. If there were no session before, then the Null cipher suite is used, which means there is no encryption and messages have no integrity digests until the session has been established.

Data Transfer

The SSL Record Protocol, shown in Figure 3, is used to transfer application and SSL Control data between the client and server, possibly fragmenting this data into smaller units, or combining multiple higher level protocol data messages into single units. It may compress, attach digest signatures, and encrypt these units before transmitting them using the underlying reliable transport protocol (Note: currently all major SSL implementations lack support for compression).

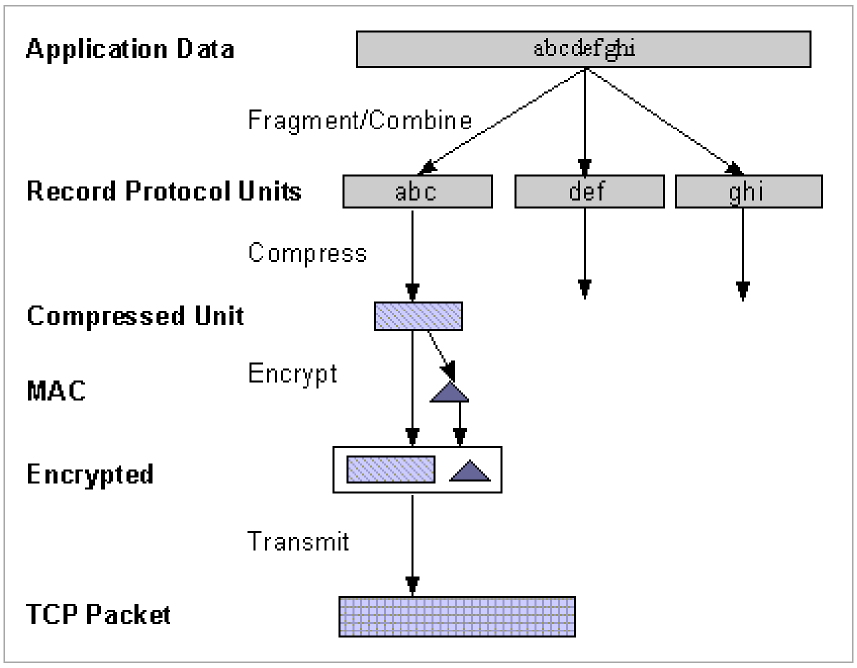

***Figure 3**: SSL Record Protocol*

Securing HTTP Communication

One common use of SSL is to secure Web HTTP communication between a browser and a webserver. This case does not preclude the use of non-secured HTTP. The secure version is mainly plain HTTP over SSL (named HTTPS), but with one major difference: it uses the URL scheme `https` rather than `http` and a different server port (by default 443). This mainly is what `mod_ssl` provides to you for the Apache webserver...

References

[AC96]
Bruce Schneier, *Applied Cryptography*, 2nd Edition, Wiley, 1996. See http://www.counterpane.com/ for various other materials by Bruce Schneier.

[X208]
ITU-T Recommendation X.208, *Specification of Abstract Syntax Notation One (ASN.1)*, 1988. See for instance http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.208-198811-I.

[X509]
ITU-T Recommendation X.509, *The Directory - Authentication Framework*. See for instance http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.509.

[PKCS]
Public Key Cryptography Standards (PKCS), RSA Laboratories Technical Notes, See http://www.rsasecurity.com/rsalabs/pkcs/.

[MIME]
N. Freed, N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, RFC2045. See for instance http://ietf.org/rfc/rfc2045.txt.

[SSL2]
Kipp E.B. Hickman, *The SSL Protocol*, 1995. See http://www.netscape.com/eng/security/SSL_2.html.

[SSL3]
Alan O. Freier, Philip Karlton, Paul C. Kocher, *The SSL Protocol Version 3.0*, 1996. See http://www.netscape.com/eng/ssl3/draft302.txt.

[TLS1]
Tim Dierks, Christopher Allen, *The TLS Protocol Version 1.0*, 1999. See http://ietf.org/rfc/rfc2246.txt.

URI References

[1] http://home.earthlink.net/~fjhirsch/Papers/wwwj/article.html
[2] http://home.earthlink.net/~fjhirsch/
[3] http://www.ora.com/catalog/wjsum97/
[6] http://www.thawte.com/
[7] http://www.verisign.com/