

SSL

SSL

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide secure communications on the Internet for such things as web browsing, e-mail, Internet faxing, instant messaging and other data transfers. There are slight differences between SSL and TLS, but they are essentially the same

Description

The TLS protocol allows applications to communicate across a network in a way designed to prevent eavesdropping, tampering, and message forgery. TLS provides endpoint authentication and communications privacy over the Internet using cryptography. Typically, only the server is authenticated (i.e., its identity is ensured) while the client remains unauthenticated; this means that the end user (whether an individual or an application, such as a Web browser) can be sure with whom it is communicating. The next level of security — in which both ends of the "conversation" are sure with whom they are communicating — is known as mutual authentication. Mutual authentication requires public key infrastructure (PKI) deployment to clients unless TLS-PSK or the Secure Remote Password (SRP) protocol are used, which provide strong mutual authentication without needing to deploy a PKI.

TLS involves three basic phases:

- a) Peer negotiation for algorithm support.
- b) Key exchange and authentication.
- c) Symmetric cipher encryption and message authentication.

During the first phase, the client and server negotiate cipher suites, which determine the ciphers to be used, the key exchange and authentication algorithms, as well as the message authentication codes (MACs). The key exchange and authentication algorithms are typically public key algorithms, or as in TLS-PSK preshared keys could be used. The message authentication codes are made up from cryptographic hash functions using the HMAC construction for TLS, and a non-standard pseudorandom function for SSL.

Typical algorithms could be:

1. For key exchange: RSA, Diffie-Hellman, ECDH, SRP, PSK
2. For authentication: RSA, DSA, ECDSA
3. Symmetric ciphers: RC4, Triple DES, AES or Camellia. In older versions of SSL the ciphers RC2, IDEA and DES were also used.
4. For cryptographic hash function: HMAC-MD5 or HMAC-SHA are used for TLS, MD5 and SHA for SSL, while older versions of SSL also used MD2 and MD4.

How it works

SSL handshake with two way authentication with certificates. (Accuracy disputed.)

SSL handshake with two way authentication with certificates. (Accuracy disputed.)

A TLS client and server negotiate a stateful connection by using a handshaking procedure. During this handshake, the client and server agree on various parameters used to establish the connection's security.

- The handshake begins when a client connects to a TLS-enabled server requesting a secure connection, and presents a list of supported ciphers and hash functions.
- From this list, the server picks the strongest cipher and hash function that it also supports and notifies the client of the decision.

SSL

- The server sends back its identification in the form of a digital certificate. The certificate usually contains the server name, the trusted certificate authority (CA), and the server's public encryption key.

The client may contact the server that issued the certificate (the trusted CA as above) and confirm that the certificate is authentic before proceeding.

- In order to generate the session keys used for the secure connection, the client encrypts a random number with the server's public key, and sends the result to the server. Only the server can decrypt it (with its private key): this is the one fact that makes the keys hidden from third parties, since only the server and the client have access to this data.
- From the random number, both parties generate key material for encryption and decryption.

This concludes the handshake and begins the secured connection, which is encrypted and decrypted with the key material until the connection closes.

If any one of the above steps fails, the TLS handshake fails, and the connection is not created.

TLS handshake in detail

The TLS protocol exchanges records, which encapsulate the data to be exchanged. Each record can be compressed, padded, appended with a message authentication code (MAC), or encrypted, all depending on the state of the connection. Each record has a content type field that specifies the record, a length field, and a TLS version field.

When the connection starts, the record encapsulates another protocol, the handshake protocol, which has content type 22.

A simple connection example follows:

- A Client sends a ClientHello message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and compression methods.
- The Server responds with a ServerHello, containing the chosen protocol version, a random number, cipher suite, and compression method from the choices offered by the client.
- The Server sends its Certificate (depending on the selected cipher suite, this may be omitted by the Server).
- These certificates are currently X.509, but there is also a draft specifying the use of OpenPGP based certificates.
- The server may request a certificate from the client, so that the connection can be mutually authenticated, using a CertificateRequest.
- The Server sends a ServerHelloDone message, indicating it is done with handshake negotiation.
- The Client responds with a ClientKeyExchange message, which may contain a PreMasterSecret, public key, or nothing. (Again, this depends on the selected cipher.)
- The Client and Server then use the random numbers and PreMasterSecret to compute a common secret, called the "master secret". All other key data is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed "pseudorandom function".

SSL

- The Client now sends a ChangeCipherSpec message, essentially telling the Server, "Everything I tell you from now on will be encrypted." Note that the ChangeCipherSpec is itself a record-level protocol, and has type 20, and not 22.
- Finally, the Client sends an encrypted finished message, containing a hash and MAC over the previous handshake messages.
- The Server will attempt to decrypt the Client's finished message, and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
- Finally, the Server sends a ChangeCipherSpec and its encrypted finished message, and the Client performs the same decryption and verification.
- At this point, the "handshake" is complete and the Application protocol is enabled, with content type of 23. Application messages exchanged between Client and Server will be encrypted.

Support for virtual servers

TLS does not provide a mechanism for a client to tell a server the name of the server it is contacting. It is often desirable for clients to provide this information to facilitate secure connections to servers that host multiple Virtual Servers sharing a single IP address.

In order to provide the server name, RFC 4366 Transport Layer Security (TLS) Extensions allow clients to include a "server_name" extension in the extended client hello. The TLS server, in response, should provide the appropriate certificate for the requested Virtual Server.

Security

TLS/SSL have a variety of security measures:

- The client may use the certificate authority's (CA's) public key to validate the CA's digital signature on the server certificate. If the digital signature can be verified, the client accepts the server certificate as a valid certificate issued by a trusted CA.
- The client verifies that the issuing CA is on its list of trusted CAs.
- The client checks the server's certificate validity period. The authentication process stops if the current date and time fall outside of the validity period.
- To protect against Man-in-the-middle attacks, the client compares the actual DNS name of the server to the DNS name on the certificate.[citation needed] Browser-dependent, not defined by TLS.
- Protection against a downgrade of the protocol to a previous (less secure) version or a weaker cipher suite.
- Numbering all the Application records with a sequence number, and using this sequence number in the message authentication codes (MACs).
- Using a message digest enhanced with a key (so only a key-holder can check the MAC). This is specified in RFC 2104. TLS only.
- The message that ends the handshake ("Finished") sends a hash of all the exchanged handshake messages seen by both parties.
- The pseudorandom function splits the input data in half and processes each one with a different hashing algorithm (MD5 and SHA-1), then XORs them together. This provides protection even if one of these algorithms is found to be vulnerable. TLS only.
- SSL v3 improved upon SSL v2 by adding SHA-1 based ciphers, and support for certificate authentication. Additional improvements in SSL v3 include better handshake protocol flow and increased resistance to man-in-the-middle attacks.

SSL v2 is flawed in a variety of ways:

- ❑ Identical cryptographic keys are used for message authentication and encryption.
- ❑ MACs are unnecessarily weakened in the "export mode" required by U.S. export restrictions (symmetric key length was limited to 40 bits in Netscape and Internet Explorer).
- ❑ SSL v2 has a weak MAC construction and relies solely on the MD5 hash function.
- ❑ SSL v2 does not have any protection for the handshake, meaning a Man-in-the-middle downgrade attack can go undetected.
- ❑ SSL v2 uses the TCP connection close to indicate the end of data. This means that truncation attacks are possible: the attacker simply forges a TCP FIN, leaving the recipient unaware of an illegitimate end of data message (SSL v3 fixes this problem by having an explicit closure alert).

SSL v2 is disabled by default in Internet Explorer 7, Mozilla Firefox 2, Opera 9 and Safari. Support for SSL v2 (and weak 40-bit and 56-bit ciphers) will be removed completely from the upcoming Opera 9.5 (code-named Kestrel).]

Applications

TLS runs on layers beneath application protocols such as HTTP, FTP, SMTP, NNTP, and XMPP and above a reliable transport protocol, TCP for example. While it can add security to any protocol that uses reliable connections (such as TCP), it is most commonly used with HTTP to form HTTPS. HTTPS is used to secure World Wide Web pages for applications such as electronic commerce and asset management. SMTP is also an area in which TLS has been growing and is specified in RFC 3207. These applications use public key certificates to verify the identity of endpoints.

An increasing number of client and server products support TLS natively, but many still lack support. As an alternative, users may wish to use standalone TLS products like Stunnel. Wrappers such as Stunnel rely on being able to obtain a TLS connection immediately, by simply connecting to a separate port reserved for the purpose. For example, by default the TCP port for HTTPS is 443, to distinguish it from HTTP on port 80.

TLS can also be used to tunnel an entire network stack to create a VPN, as is the case with OpenVPN. Many vendors now marry TLS's encryption and authentication capabilities with authorization. There has also been substantial development since the late 1990s in creating client technology outside of the browser to enable support for client/server applications. When compared against traditional IPsec VPN technologies, TLS has some inherent advantages in firewall and NAT traversal that make it easier to administer for large remote-access populations.

TLS is also increasingly being used as the standard method for protecting SIP application signaling. TLS can be used to provide authentication and encryption of the SIP signalling associated with VoIP and other SIP-based applications.

History and development

Early research efforts toward transport layer security included the Secure Network Programming (SNP) API, which in 1993 explored the approach of having a secure transport layer API closely resembling sockets, to facilitate retrofitting preexisting network applications with security measures. The SNP project received the 2004 ACM Software System Award.

The SSL protocol was originally developed by Netscape. Version 1.0 was never publicly released; version 2.0 was released in 1994 but "contained a number of security flaws which ultimately led to the design of SSL version 3.0", which was released in 1996 (Rescorla 2001). This later served as the basis for TLS version 1.0, an Internet Engineering Task Force (IETF) standard protocol first defined in RFC 2246 in January 1999. Visa, MasterCard, American Express and many leading financial institutions have endorsed SSL for commerce over the Internet.[citation needed]

SSL operates in modular fashion. It is extensible by design, with support for forward and backward compatibility and negotiation between peers.

Early short keys

Some early implementations of SSL used 40-bit symmetric keys because of US government restrictions on the export of cryptographic technology. The US government explicitly imposed a 40-bit keyspace, which was small enough to be broken by brute-force search by law enforcement agencies wishing to read the encrypted traffic, while still presenting obstacles to less-well-funded attackers.[citations needed] A similar limitation applied to Lotus Notes in export versions. After several years of public controversy, a series of lawsuits, and eventual US government recognition of cryptographic products with longer key sizes produced outside the US, the authorities relaxed some aspects of the export restrictions. The 40-bit key size limitation has mostly gone away, and modern implementations use 128-bit (or longer) keys for symmetric key ciphers.[citations needed]

Standards

The first definition of TLS appeared in:

- RFC 2246: "The TLS Protocol Version 1.0".

The current approved version is 1.1, which is specified in

- RFC 4346: "The Transport Layer Security (TLS) Protocol Version 1.1".

The next version is proposed:

- INTERNET DRAFT - The Transport Layer Security (TLS) Protocol Version 1.2 (published March 2008, expires September 2008)

Other RFCs subsequently extended TLS, including:

- RFC 2595: "Using TLS with IMAP, POP3 and ACAP". Specifies an extension to the IMAP, POP3 and ACAP services that allow the server and client to use transport-layer security to provide private, authenticated communication over the Internet.
- RFC 2712: "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)". The 40-bit ciphersuites defined in this memo appear only for the purpose of documenting the fact that those ciphersuite codes have already been assigned.
- RFC 2817: "Upgrading to TLS Within HTTP/1.1", explains how to use the Upgrade mechanism in HTTP/1.1 to initiate Transport Layer Security (TLS) over an existing TCP connection. This allows unsecured and secured HTTP traffic to share the same well known port (in this case, http: at 80 rather than https: at 443).
- RFC 2818: "HTTP Over TLS", distinguishes secured traffic from insecure traffic by the use of a different 'server port'.
- RFC 3207: "SMTP Service Extension for Secure SMTP over Transport Layer Security". Specifies an extension to the SMTP service that allows an SMTP server and client to use transport-layer security to provide private, authenticated communication over the Internet.
- RFC 3268: "AES Ciphersuites for TLS". Adds Advanced Encryption Standard (AES) ciphersuites to the previously existing symmetric ciphers.
- RFC 3546: "Transport Layer Security (TLS) Extensions", adds a mechanism for negotiating protocol extensions during session initialisation and defines some extensions. Made obsolete by RFC 4366.
- RFC 3749: "Transport Layer Security Protocol Compression Methods", specifies the framework for compression methods and the DEFLATE compression method.

SSL

- RFC 3943: "Transport Layer Security (TLS) Protocol Compression Using Lempel-Ziv-Stac (LZS)".
- RFC 4132: "Addition of Camellia Cipher Suites to Transport Layer Security (TLS)".
- RFC 4162: "Addition of SEED Cipher Suites to Transport Layer Security (TLS)".
- RFC 4279: "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", adds three sets of new ciphersuites for the TLS protocol to support authentication based on pre-shared keys.
- RFC 4347: "Datagram Transport Layer Security" specifies a TLS variant that works over datagram protocols (such as UDP).
- RFC 4366: "Transport Layer Security (TLS) Extensions" describes both a set of specific extensions, and a generic extension mechanism.
- RFC 4492: "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)".
- RFC 4507: "Transport Layer Security (TLS) Session Resumption without Server-Side State".
- RFC 4680: "TLS Handshake Message for Supplemental Data".
- RFC 4681: "TLS User Mapping Extension".
- RFC 4785: "Pre-Shared Key (PSK) Ciphersuites with NULL Encryption for Transport Layer Security (TLS)".

Implementation

Programmers may use the OpenSSL, NSS, or GnuTLS libraries for SSL/TLS functionality. Microsoft Windows includes an implementation of SSL and TLS as part of its Secure Channel package. Delphi programmers may use a library called Indy.

TLS 1.1

As noted above, TLS 1.1 is the current approved version of the TLS protocol. TLS 1.1 clarifies some ambiguities and adds a number of recommendations, but remains very similar to TLS 1.0. A full list of differences is provided in RFC 4346 (Section 1.1).

Cryptography Portal

- Certificate authority
- Digital Certificate
- Extended Validation Certificate
- SSL acceleration
- Datagram Transport Layer Security
- Multiplexed Transport Layer Security
- X.509
- Virtual private network
- SEED
- Server gated cryptography

Software

- OpenSSL: a free (and very popular) implementation (BSD license with some extensions).
- GnuTLS: a free GPL licensed implementation.
- JSSE: a Java implementation included in the Java Runtime Environment
- Network Security Services (NSS): FIPS 140 validated open source library
- Crypt::OpenSSL: Perl wrapper modules for OpenSSL.
- System.Net.Security: a Microsoft implementation for the .NET Common Language Runtime.
- Mono.Security a free software implementation for the Common Language Runtime.
- yaSSL: a free implementation (GPL and commercial licenses available).