SSL/TLS in Detail

Updated: July 31, 2003

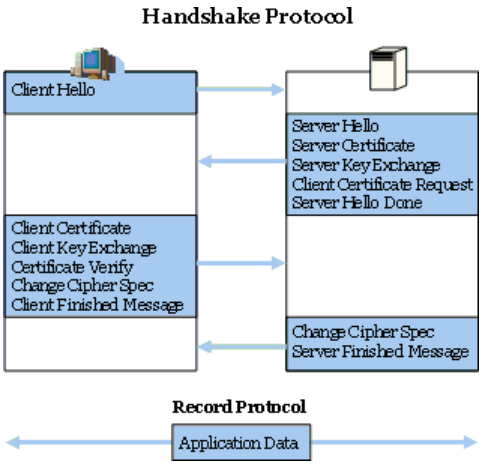
Applies To: Windows Server 2003 with SP1

This section provides a detailed explanation of the SSL/TLS protocol, specifically the handshake protocol, its associated messages and alerts, and the record protocol.

The Handshake Protocol

The handshake protocol is a series of sequenced messages that negotiate the security parameters of a data transfer session. Figure 2 illustrates the message sequence in the handshake protocol.

Figure 2. Handshake Protocol Messages



Initial Client Message to Server

Client Hello. The client initiates a session by sending a Client Hello message to the server. The Client Hello message contains:

- **Version Number.** The client sends the version number corresponding to the highest version it supports. Version 2 is used for SSL 2.0, version 3 for SSL 3.0, and version 3.1 for TLS. Although the IETF RFC for TLS is TLS version 1.0, the protocol uses 3.1 in the version field to indicate that it is a higher level (newer and with more functionality) than SSL 3.0.
- **Randomly Generated Data.** ClientRandom[32], the random value, is a 4-byte number that consists of the client's date and time plus a 28-byte randomly generated number that will ultimately be used with the server random value to generate a master secret from which the encryption keys will be derived.
- **Session Identification (if any).** The sessionID is included to enable the client to resume a previous session. Resuming a previous session can be useful, because creating a new session requires processor-intensive public key operations that can be avoided by resuming an existing session with its established session keys. Previous session information, identified by the sessionID, is stored in the respective client and server session caches.
- **Cipher Suite.** The A list of cipher suites available on the client. An example of a cipher suite is TLS_RSA_WITH_DES_CBC_SHA, where TLS is the protocol version, RSA is the algorithm that will be used for the key exchange, DES_CBC is the encryption algorithm (using a 56-bit key in CBC mode), and SHA is the hash function.

- **Compression Algorithm.** The requested compression algorithm (none currently supported).

The following is an example of a Client Hello message:

```
ClientVersion 3,1
ClientRandom[32]
SessionID: None (new session)
Suggested Cipher Suites:
   TLS_RSA_WITH_3DES_EDE_CBC_SHA
   TLS_RSA_WITH_DES_CBC_SHA
Suggested Compression Algorithm: NONE
```

Server Response to Client

Server Hello. The server responds with a Server Hello message. The Server Hello message includes:

- **Version Number.** The server sends the highest version number supported by both sides. This is the lower of: the highest version number the server supports and the version sent in the Client Hello message.
- **Randomly Generated Data.** ServerRandom[32], the Random Value, is a 4-byte number of the server's date and time plus a 28-byte randomly generated number that will be ultimately used with the client random value to generate a master secret from which the encryption keys will be derived
- **Session Identification (if any).** This can be one of three choices.
    - New session ID – The client did not indicate a session to resume so a new ID is generated. A new session ID is also generated when the client indicates a session to resume but the server can't or won't resume that session. This latter case also results in a new session ID.
    - Resumed Session ID– The id is the same as indicated in the client hello. The client indicated a session ID to resume and the server is willing to resume that session.
    - Null – this is a new session, but the server is not willing to resume it at a later time so no ID is returned.
- **Cipher Suite.** The server will choose the strongest cipher that both the client and server support. If there are no cipher suites that both parties support, the session is ended with a "handshake failure" alert.
- **Compression Algorithm.** Specifies the compression algorithm to use (none currently supported).

The following is an example of a Server Hello Message:

```
Version 3,1
ServerRandom[32]
SessionID: bd608869f0c629767ea7e3ebf7a63bdcffb0ef58b1b941e6b0c044acb6820a77
Use Cipher Suite:
TLS_RSA_WITH_3DES_EDE_CBC_SHA
Compression Algorithm: NONE
```

Server Certificate. The server sends its certificate to the client. The server certificate contains the server's public key. The client will use this key to authenticate the server and to encrypt the premaster secret.

The client also checks the name of the server in the certificate to verify that it matches the name the client used to connect. If the user typed www.contoso.com as the URL in the browser, the certificate should contain a subject name of www.contoso.com or *.contoso.com. Internet Explorer will warn the user if these names do not match.

Server Key Exchange. This is an optional step in which the server creates and sends a temporary key to the client. This key can be used by the client to encrypt the Client Key Exchange message later in the process. The step is only required when the public key algorithm does not provide the key

material necessary to encrypt the Client Key Exchange message, such as when the server's certificate does not contain a public key.

Client Certificate Request. This is an optional step in which the server requests authentication of the client. This step might be used for Web sites (such as a banking Web site) where the server must confirm the identity of the client before providing sensitive information.

Server Hello Done. This message indicates that the server is finished and awaiting a response from the client.

Client Response to Server

Client Certificate. If the server sent a Client Certificate Request, the client sends its certificate to the server for client authentication. The client's certificate contains the client's public key.

Client Key Exchange. The client sends a Client Key Exchange message after computing the premaster secret using both random values. The premaster secret is encrypted by the public key from the server's certificate before being transmitted to the server. Both parties will compute the master secret locally and derive the session key from it.

If the server can decrypt this data and complete the protocol, the client is assured that the server has the correct private key. This step is crucial to prove the authenticity of the server. Only the server with the private key that matches the public key in the certificate can decrypt this data and continue the protocol negotiation.

This message will also include the protocol version. The server will verify that it matches the original value sent in the client hello message. This measure guards against rollback attacks. Rollback attacks work by manipulating the message in order to cause the server and the client to use a less secure, earlier version of the protocol.

Certificate Verify. This message is sent only if the client previously sent a Client Certificate message. The client is authenticated by using its private key to sign a hash of all the messages up to this point. The recipient verifies the signature using the public key of the signer, thus ensuring it was signed with the client's private key.

Change Cipher Spec. This message notifies the server that all messages that follow the Client Finished message will be encrypted using the keys and algorithms just negotiated.

Client Finished. This message is a hash of the entire conversation to provide further authentication of the client. This message is the first message that the record layer encrypts and hashes.

Server Final Response to Client

Change Cipher Spec Message. This message notifies the client that the server will begin encrypting messages with the keys just negotiated.

Server Finished Message. This message is a hash of the entire exchange to this point using the session key and the MAC secret. If the client is able to successfully decrypt this message and validate the contained hashes, it is assured that the SSL/TLS handshake was successful, and the keys computed on the client machine match those computed on the server.

The Alert Sub-Protocol

The alert sub-protocol is a component of the Handshake protocol that includes event-driven alert messages that can be sent from either party. Following an alert message the session is either ended or the recipient is given the choice of whether or not to end the session. The alerts are defined in the TLS specification in RFC 2246. Table 1 lists the alert messages and their descriptions.

Table 1. Event-Driven Alert Messages from the Alert Sub-Protocol

Alert Message	Fatal?	Description
unexpected_message	Yes	Inappropriate message
bad_record_mac	Yes	Incorrect MAC
decryption_failed	Yes	Unable to decrypt TLSCiphertext correctly
record_overflow	Yes	Record is more than 2^14+1024 bytes

handshake_failure	Yes	Unacceptable security parameters
bad_certificate	Yes	There is a problem with the Certificate
unsupported_certificate	No	Certificate is unsupported
certificate_revoked	No	Certificate has been revoked
certificate_expired	No	Certificate has expired
certificate_unknown	No	Certificate is unknown
illegal_parameter	Yes	Security parameters violated
unknown_ca	Yes	CA unknown
access_denied	Yes	Sender does not want to negotiate
decode_error	Yes	Unable to decode message
decrypt_error	Yes	Handshake cryptographic operation failed
export_restriction	Yes	Not in compliance with export regulations
protocol_version	Yes	Protocol version not supported by both parties
insufficient_security	Yes	Security requirements no met
internal_error	Yes	Error not related to protocol
user_canceled	Yes	Not related to protocol failure
no_renegotiation	No	Negotiation request rejected.

The Record Protocol

The record protocol receives the data from the application layer and:

- Fragments the data into blocks or reassembles fragmented data into its original structure.
- Numbers the sequence of data blocks in the message to protect against attacks that attempt to reorder data.
- Compresses or decompresses the data using the compression algorithm negotiated in the handshake protocol.
- Encrypts or decrypts the data using the encryption keys and cryptographic algorithm negotiated during the handshake protocol.
- Applies an HMAC (or a MAC for SSL 3.0) to outgoing data. Computes the HMAC and verifies that it is identical to the value transmitted in order to check data integrity when a message is received.

Once the record protocol has completed its operations on the data, it sends the data to the TCP/IP transport layer for transmission. If the data is incoming, it is sent to the appropriate process for reception.