How TLS Works – An Overview Based on RFC 2246

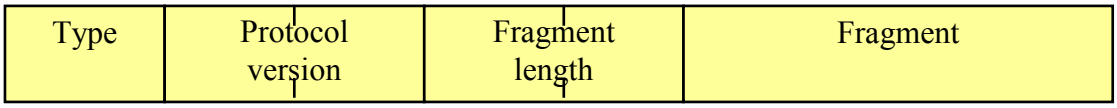
Integers are transmitted in network / big-endian order (MSB first).

TLS Record Layer

- Receives data from above.
- May combine multiple client messages of the same type into a single record; may fragment a client message across multiple records.
- Maximum content size 2^14 bytes.

Record structure:

- ContentType (1 byte);
- ProtocolVersion (2 bytes: major, minor);
- length (unsigned integer 16-bits);
- fragment (content data bytes: number of bytes is length specified)



Protocol version: TLS 1.0 is 3,1 (SSL 3.0 is 3,0 and TLS 1.0 is a minor modification to SSL 3.0)

Note (p17) different record types (from above) may be interleaved.

Note: Because records do not correspond to message boundaries from above, length field cannot be used to determine size of individual messages from above; hence layer above must define either fixed-size messages or use its own length field.

Record Types

- Handshake
- Alert
- Change cipher spec
- Application data
- Additional record types may be defined
    - Implementation should ignore unknown record types

Compression

The fragment (data content) of a record is compressed – the new record has the same syntax but length will change and the fragment will also change (unless the compression method is null). Compression may not expand data by more than 1024 bytes. Length may not exceed 2^14 + 1024.

- Null compression is the default (and the only supported compression method)

Payload protection

Depending upon the current active cipherspec, the fragment is encrypted, including a MAC. Resulting length may not exceed 2^14 + 2048.

- MAC is computed before encryption using HMAC. MAC key is MAC_write_secret. Data used for MAC is:
    - Record sequence number
    - The entire compressed record (content type, protocol version, length and fragment).
- Null encryption (with no MAC) is supported.
- For block ciphers, padding followed by padded length is added to the fragment to make the message a multiple of the block size; the padding may be more than the minimum needed, so as to help frustrate message length analysis. The last byte of the message before encryption is the padding length; the padded bytes are filled with the padding length value also. Maximum padding is 255 bytes. If the padding is zero bytes, the padded length of 0 must be added to the data before it (all, including the padding) is encrypted.

TLS Handshake

TLS handshake messages establish the following state information for the session. The information may be renegotiated within the session, changing the encryption method or the keys.

- Session identifier – chosen by server
- Peer certificate X509v3 certificate of peer (may be null)
- Compression method
- Cipher spec: cipher and MAC algorithms.
- Master secret: 48 bytes shared between client and server
- IsResumable: flag whether session can be resumed on a new connection.

Change cipher spec message type

A single-byte message with the value 1. Transmitted under the current settings, it causes the sender and receiver to update to the pending settings for subsequent data. The pending settings are what has been negotiated so far by other handshake messages. The pending settings are then reinitialised to the default settings (see section 6.1 of the RFC)

Alert message type – error reporting

- AlertLevel (byte): warning or fatal; fatal causes immediate termination of connection.
    - If warning received, receiver may treat it as fatal and close the connection.
- AlertDescription: an error/warning detail code as a single byte. Many types including:
    - Close_notify: notifies other party that this party will not send any more data on the connection; use Alert level warning. The other party must discard pending writes and send its own close_notify and close the connection.

Handshake message type: Negotiating a session

ContentType is HandShake
HandShake record syntax (contained as the fragment(s) within TLS records of type HandShake):

- HandshakeType (byte): Particular type of handshake message

- Length (unsigned 24-bit integer).
- Body

Hello request

Server sends to client to request session renegotiation (asynchronous – may be sent at any time). Client may refuse to renegotiate. Server may close connection with fatal error if client does not respond with client hello (thus renegotiating the session).

  - Hello request is not included in the messages that are verified by the hash in the finished messages – it is not part of the renegotiation but only a request for renegotiation.
- Empty message body.

Client hello

First message from client on connect; may be sent in response to hello request; may be sent asynchronously for client to initiate renegotiation of session.

- ProtocolVersion: Highest version supported by client (i.e. 3,1 for TLS 1.0)
- Random data:
  - Gmt_unix_time: 32-bit time seconds since jan 1 1970 GMT.
  - Random-bytes (28 bytes).
- SessionID (0-32 bytes): 0 if requesting a new session; otherwise, ID of a session requested to be resumed.
    - Length (8 bits)
    - Opaque data (0-32 bytes)
- Cipher suite vector: Each cipher suite is represented by a 2-byte code, in order of preference. Code 0 (null encryption) may not be negotiated, but is the initial state.
    - Length (16 bits)
    - Vector data content (length bytes; i.e. length/2 items) [see page 7 of RFC]
  - Server will select a listed method, or if no acceptable choice is given, return handshake failure alert and close connection.
- Compression method vector: Each method is represented by a single byte; only null is currently defined.
    - Length (1 byte) followed by data.
  - Vector must contain null and all clients must support null compression.
- Additional data for forward compatibility: this is the only handshake message that allows additional data to be included; that data must be ignored but must be included in the finishing hashes.
- Response to client hello must be server hello – any other handshake message is a fatal error.

Server hello

Only sent if server is able to select a suitable match from client vectors for encryption algorithm and compression

- ProtocolVersion (lower of client requested version and server's highest supported version)
- Random – different from and independent of clienthello.Random
- Session ID:
    - Zero: session will not be cached and cannot be resumed.
    - Non-zero: session may be resumed.
        - If client gave session ID and server has returned that same session ID then this means that resume has been accepted and parties must proceed directly to finished messages.
- CipherSuite: 2-byte code representing selected cipher suite
- CompressionMethod: 1-byte code representing select compression method – currently only null compression is defined.

Server certificate

Server must send certificate whenever key exchange method is not anonymous. Server certificate message always follows server hello message. Certificate type must be appropriate for the selected cipher suite key exchange algorithm. Certificates will generally be X509. IETF PKIX working group defines certificates.

- Certificate_list: the message contains a sequence (chain) of X.509v3 certificates as a vector with a three-byte length (specifying the length of the certificate chain in bytes). First certificate must be sender's certificate, and following certificates must directly certify the one before them. The last certificate in the chain would be the self-signed certificate of a recognised CA – this may be omitted since the public key of the CA must be known by independent means anyhow.

Server key exchange

Only used when the certificate does not provide suitable information for the client to accomplish key exchange. This message will then contain a public key used to encrypt the premaster secret or a key for Diffie-Hellman key exchange.

    - RSA export restriction: (Page 40) US low prohibits use of RSA keys greater than 512 bits for RSA key exchange. In case a host's public key is larger than 512 bits, it can create a temporary shorter key and sign it with the larger key and use it for key exchange.
- Contents of this message depend upon the key exchange algorithm. The contents are parameters for key exchange
- A digital signature of those parameters is included in the message.

Certificate request

Server may request client to send it a certificate.

- Certificate_types_list: A vector of certificate type codes (single byte codes representing different types of certificates), with a 1-byte length field. These are the types of certificate that the server will accept.
- Certificate_authorities_list: A vector of identifiers of CA's that the server will accept - these may represent root CA's or subordinate CA's. The vector contains DistinguishedName objects as defined by X.509.

Server hello done

Last message from the server in this sequence. Server now listens for client response. Client should check information provided by server and then proceed with client's part of the key exchange.

- Empty message body

Client certificate

If the server requested a certificate, the client should send this message containing the client certificates. If there are no suitable certificates, the client should send a message containing the certificates.

Client key exchange

The content of this message depends upon the key exchange method.
RSA key exchange:

- ProtocolVersion: This is the latest protocol version supported by the client; the server must check that it matches the version used in the client hello message. This provides protection against version roll-back attacks (in which an attacker fools the client and server into using SSL v2.0 so the attacker can exploit the weaknesses of the earlier protocol version).
- Random: 46 bytes of securely generated random data.
- The above 48-byte premaster secret is encrypted using the RSA key provided by the server. (The server will decrypt the premaster secret to use in calculating the master secret.)

For Diffie-Hellman key exchange, the message contains public values for the key exchange, unless these were already sent in the client certificate.

Certificate verify

If the client sent a certificate that has signing capability (can be used to compute digital signatures) then this message can be sent to verify the signature. The message contains the digital signature of all the handshake messages sent or received prior to this message (i.e. the concatenation of the messages from the client hello up to but excluding the current message).

[Change cipher spec]

At this point the client should send the change cipher spec message. This message is not a handshake message.

Finished

This message is sent immediately after a change cipher spec message. This is the first message that is protected with the algorithms, keys and secrets that have just been negotiated. The contents of this message must be checked to verify the negotiation.

- Verify_data: 12 bytes of verification data. The verification data is obtained by applying a hashing function to the master secret, a label string to distinguish client and server finished messages, and two message digests calculated from the handshake messages exchanged so far. (Strictly, the 'hashing' function is a pseudo-random function that generates a stream of pseudo-random data from the supplied input. It is uses both MD5 and SHA, which should keep it secure even if one of those two hashing methods was cryptographically broken in the future.)

Master secret

The master secret is calculated from the pre-master secret and the random values specified by the client and the server in their respective hello messages. This ensures that the master secret is generated by both parties. Therefore, the same master secret cannot be reused in a new session, so a replay attack will fail because the server would generate a new random value and the master secret will be different for the new session.

Mandatory cipher suite

The required cipher suite is:

- Diffie-Hellman key exchange using ephemeral parameters.
- DSS digital signing certificate.
- Triple DES (EDE) encryption with Cipher block chaining.
    - Triple DES EDE: Encrypt(k1); decrypt(k2); encrypt(k3) so that if k1=k2=k3 the result is the same as encrypting once with key k1.
    - CBC: Before encryption, each block is XOR'd with the previous encrypted block (or with an initialisation vector for the first block).