Record Format

SSL runs on top of TCP. Recall that TCP is a stream protocol that erases packet boundaries. In order to provide for in-band signalling in a stream protocol, SSL partitions the stream into records. Each record has a five byte header followed by a payload as shown in the following diagram.

Content Type	1 byte
Version	2 bytes
Length n	2 bytes
Payload	n bytes

Synchronization is maintained by the length field which indicates not only the payload length but also the start of the next record. When SSL switches to encrypted mode, only the payload is encrypted. The header is always sent in the clear to maintain synchronization.

RFC 2246 defines the following content types. (See RFC 2246 page 17.)

Content Type	Decimal Code
Change Cipher Spec	20
Alert Message	21
Handshake Protocol	22
Application Data	23

Handshake Protocol

SSL defines a handshake protocol for sending connection setup messages. The handshake protocol uses content type 22 (0x16) in the first byte of the record header. Handshake messages have no alignment relative to the record protocol and may be split across non-contiguous records. The handshake header and data are shown in the following diagram.

Handshake Type	1 byte
Handshake Length m	3 bytes
Handshake Payload	m bytes

Typical HTTPS Transaction

The following table shows what happens during a typical HTTPS session. Messages in angle brackets are encrypted.

Client		Server
Client Hello	$\longrightarrow$	
	$\longleftarrow$	Server Hello
	$\longleftarrow$	Certificate
	$\longleftarrow$	Server Hello Done
Client Key Exchange	$\longrightarrow$	
Change Cipher Spec	$\longrightarrow$	
⟨ Finished ⟩	$\longrightarrow$	
	$\longleftarrow$	Change Cipher Spec
	$\longleftarrow$	⟨ Finished ⟩
⟨ HTTP Get ⟩	$\longrightarrow$	
	$\longleftarrow$	⟨ HTTP Response ⟩
	$\longleftarrow$	⟨ Close Notify ⟩
⟨ Close Notify ⟩	$\longrightarrow$	

Certificates

A certificate is a collection of type-length-value objects (TLVs) with the property that TLVs can be nested. In other words, the V of a TLV can itself contain more TLVs. Nested TLVs are indicated by a type field T of either 0x30 (SEQUENCE) or 0x31 (SET). Certificates use nested TLVs to organize certificate values into groups. The following diagram is a general outline of how certificate data is organized (not all TLVs are shown).

```
SEQUENCE
| SEQUENCE                  Certificate Info
| | INTEGER                 Serial Number
| | SEQUENCE                Certificate Signature Algorithm
| | SEQUENCE                Issuer
| | SEQUENCE                Validity
| | SEQUENCE                Subject
| | SEQUENCE                Subject Public Key Info
| | | SEQUENCE
| | | | OBJECT ID           Subject Public Key Algorithm
| | | | NULL
| | | BIT STRING            Subject Public Key
| | | | SEQUENCE
| | | | | INTEGER           Modulus (n in RSA encryption}
| | | | | INTEGER           Exponent (e in RSA encryption
| SEQUENCE
| | OBJECT ID               Certificate Signature Algorithm
| | NULL
| BIT STRING                Certificate Signature Value
```

The client computer uses the certificate key to encrypt its pre-master secret. The encrypted pre-master secret is then sent to the server. The certificate key is not used for any other type of encryption. After the connection is established, application data is encrypted using keys derived from the pre-master secret.

Self-signed certificates can be checked by encrypting the signature value using the public key found in the certificate itself. The result is then compared to a hash of the Certificate Info data.

RSA encryption

The client computer uses RSA encryption to securely transmit the client's 48-byte pre-master secret to the web server.

Let n and e be RSA public keys where n is the modulus and e is the exponent. Let P be a plaintext such that bit-wise numerically $P < n$. Then ciphertext C is obtained from P by raising P to the power e modulo n.

$$C = P^e \bmod n$$

Only the originator of the public keys n and e can decrypt C and obtain P.

There is also a padding requirement that ensures a strong cipher. The padding is specified in RFC 3447 page 25 as follows.

$$P = \text{0x00} \mid \text{0x02} \mid U \mid \text{0x00} \mid M$$

M is the actual message to encrypt. U is a sequence of non-zero random bytes. The vertical bar is a concatenation operator. The length of U is chosen such that the bit length of P is the same as n. The number of bytes in U cannot be less than eight. If n is 1,024 bits long (128 bytes) then M cannot be longer than $128 - 11 = 117$ bytes. Note that the leading 0x00 ensures that $P < n$ since n must have its most significant bit set.

References

A Layman's Guide to a Subset of ASN.1, BER, and DER

FIPS Publication 180-4, Secure Hash Standard

RFC 1321 The MD5 Message-Digest Algorithm

RFC 2104 HMAC: Keyed-Hashing for Message Authentication

RFC 2246 The TLS Protocol Version 1.0

RFC 3447 Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1