SSL Handshake Phase:

1. 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.

2. The server responds with a **ServerHello** message, containing the chosen protocol version, a random number, and a cipher suite and compression method from the choices offered by the client. The server may also send a *session id* as part of the message to perform a resumed handshake.
3. The server sends its **Certificate** message (depending on the selected cipher suite, this may be omitted by the server).
4. The server requests a certificate from the client, so that the connection can be mutually authenticated, using a **CertificateRequest** message.
5. The server sends a **ServerHelloDone** message, indicating it is done with handshake negotiation.

6. The client responds with a **Certificate** message, which contains the client's certificate.
7. The client sends a **ClientKeyExchange** message, which may contain a *PreMasterSecret*, public key, or nothing. (Again, this depends on the selected cipher.) This *PreMasterSecret* is encrypted using the public key of the server certificate.
8. The client sends a **CertificateVerify** message, which is a signature over the previous handshake messages using the client's certificate's private key. This signature can be verified by using the client's certificate's public key. This lets the server know that the client has access to the private key of the certificate and thus owns the certificate.
9. The client and server then use the random numbers and *PreMasterSecret* to compute a common secret, called the "master secret". All other key data for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.
10. The client now sends a **ChangeCipherSpec** record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption was negotiated)." The ChangeCipherSpec is itself a record-level protocol and has type 20 and not 22.
11. Finally, the client sends an encrypted **Finished** message, containing a hash and MAC over the previous handshake messages.
12. 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.

13. Finally, the server sends a **ChangeCipherSpec**, telling the client, "Everything I tell you from now on will be authenticated (and encrypted if encryption was negotiated)."
14. The server sends its own encrypted **Finished** message.
15. The client performs the same decryption and verification.