SSL/TLS

Applied Cryptography

NETS1035 FALL 2020

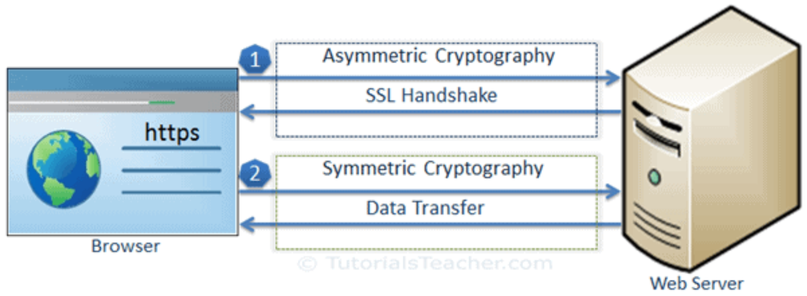
Secure Sockets Layer

- Deprecated in 2015, Secure Sockets Layer was a scheme to add cryptography to network communications developed by Netscape for their Navigator web browser in 1994 and first released for broad use as SSL v3 in 1996
- Dr. Taher ElGamal, who was chief scientist at Netscape at the time, is regarded as being the father of SSL, along with being the creator of a signature scheme which later was used as the basis for DSA and other ElGamal variants still in use today
- All versions of SSL are considered vulnerable to attack and are deprecated
- It was replaced by Transport Layer Security (TLS), newer versions of which are based on SSL 3.0, the name was changed because Microsoft wanted that
- Confusingly, the term SSL is often used to mean TLS which is not technically correct, TLS is secure, SSL is not

Transport Layer Security

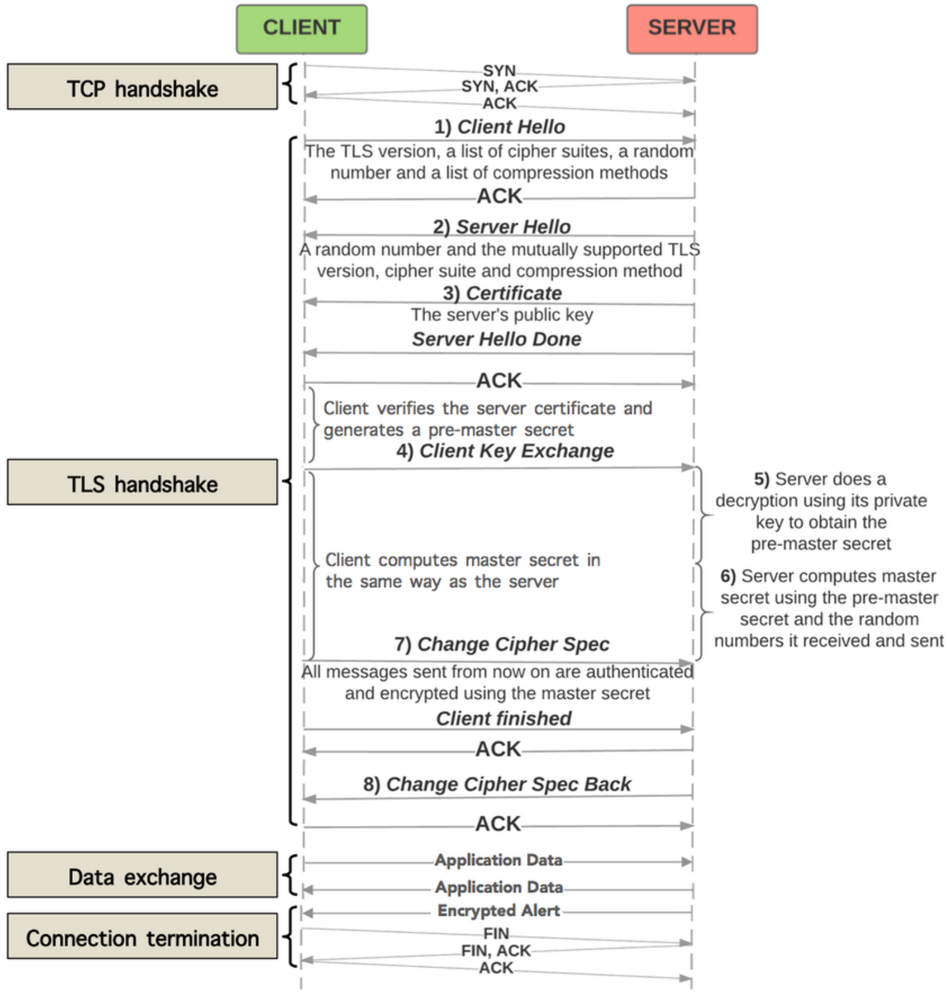
- TLS 1.0 was developed as part of the Secure Data Network Systems (SDNS) joint initiative begun in 1986 by the NSA, National Bureau of Standards, Defense Communications Agency, and a dozen companies, to research a security design for use on the public internet
- TLS was first published in 1995, but TLS version 1.0 was not released until 1999
- It was based not on the SP4 protocol that came from the SDNS efforts, but on SSL 3.0, and by rights should have been called SSL 4, but Microsoft had issues with allowing a Netscape creation to become the IETF standard, so it was renamed from SSL to TLS
- It is regularly referred to as SSL/TLS or TLS/SSL today and the current version is 1.3
- Like SSL before it, TLS uses asymmetric encryption and public key certificates in a protocol handshake to establish trust before allowing the exchange of application data using a symmetric encryption method

Cipher Selection



https://www.tutorialsteacher.com/https/how-ssl-works

- During SSL/TLS handshake, the client sends the server a list of ciphers they are willing to use
- The server selects a cipher based on a preference list in the configuration of the service software
- The server notifies the client of the selected cipher and the client will use it
- As ciphers and modes get compromised or become computationally insecure, the software package maintainers update their cipher preference defaults and system/network/ security administrators must update their running services to remain secure
- Simply updating software on existing machines does not always update configuration files that have been modified since package installation, manual intervention is often required to remove deprecated ciphers

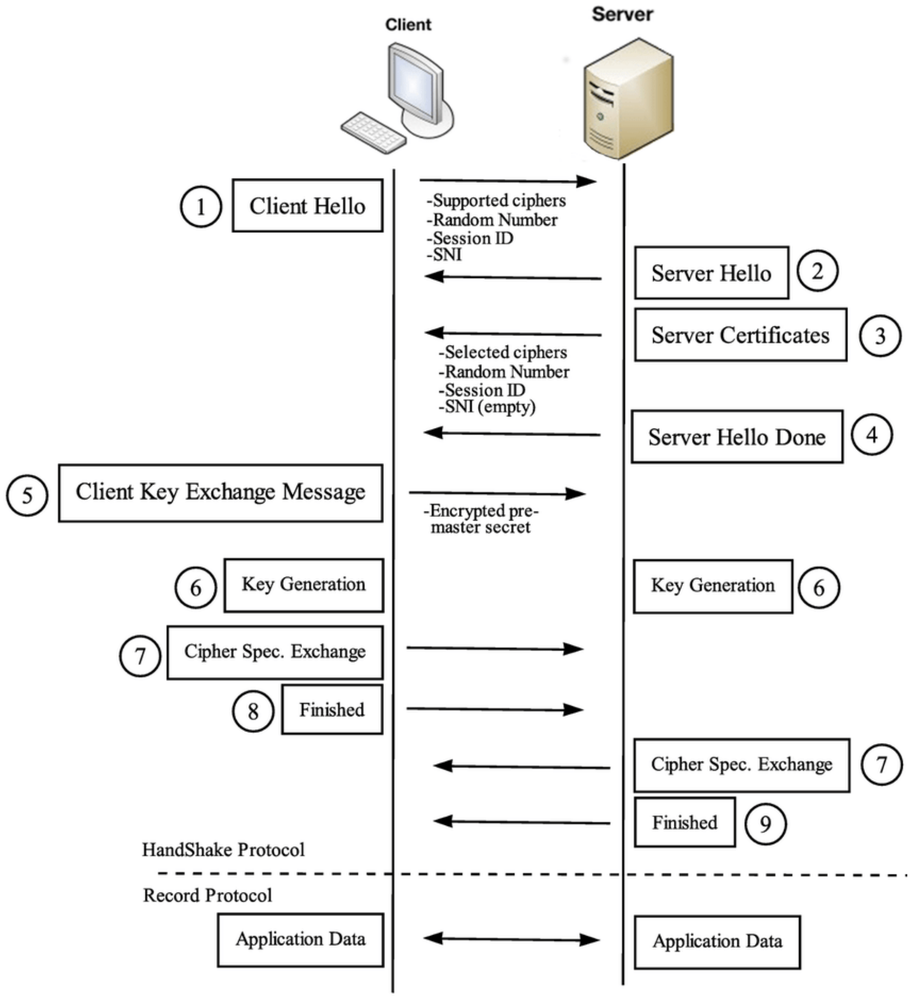


https://www.researchgate.net/figure/HTTPS-message-sequence-diagram-with-detailed-TLS-handshaking-steps_fig1_306187575

Where does TLS fit in the connection?

3 stages of a TLS-secured connection

- TLS protocol rides on top of a TCP connection, so the first stage is to establish a TCP connection between 2 endpoint applications such as a web browser and a web server
- The TLS handshake protocol is stage 2 and must be completed prior to the 2 endpoint applications exchanging application data
- TLS uses public key certificates to establish identities and trust, verifies server identity and may also verify client identity with small additions to the handshake
- TLS uses asymmetric encryption to securely agree on a random session encryption key for the 2 parties
- Stage 3 begins when the TLS handshake completes and the 2 endpoint applications switch to using symmetric encryption with the agreed-upon session key for speed and efficiency

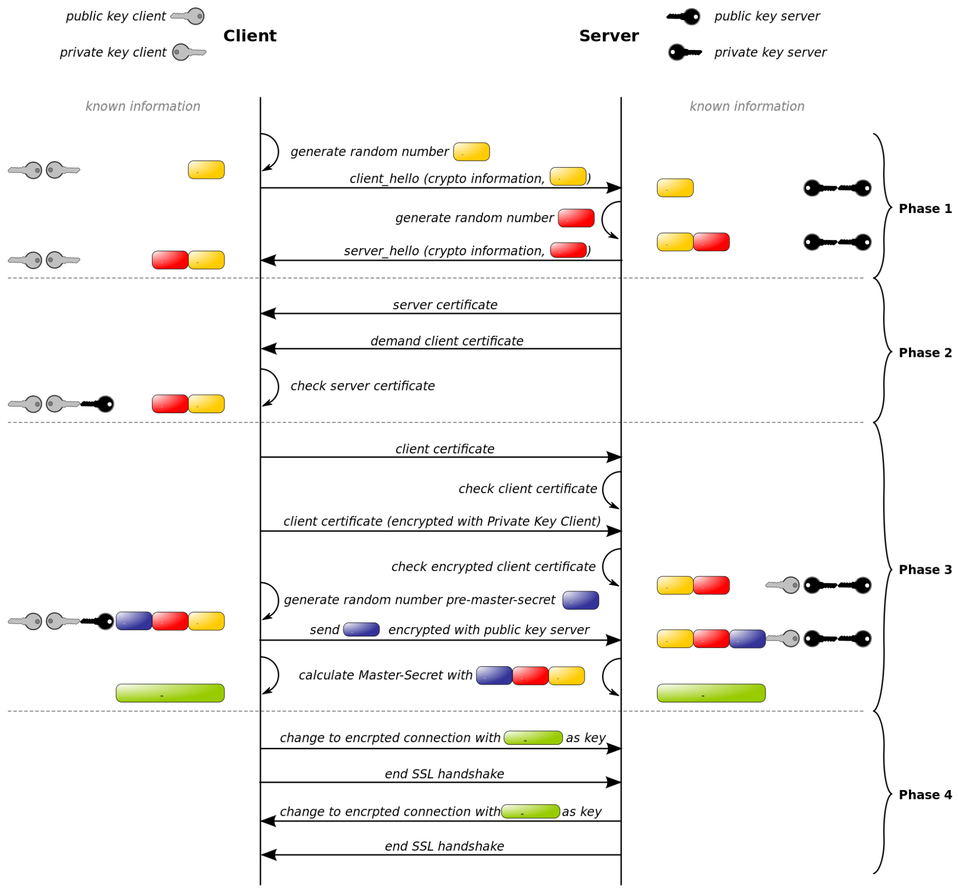


https://www.researchgate.net/publication/298065605_A_multi-level_framework_to_identify_HTTPS_services

TLS Handshake
Unauthenticated Client

- Provides a mechanism to create a temporary random key known to both parties in a connection whose content is not exposed if the packets are captured
- Once the key is known to both parties, the connection will stop using asymmetric encryption and will start using higher speed, more efficient symmetric encryption for the duration of the connection

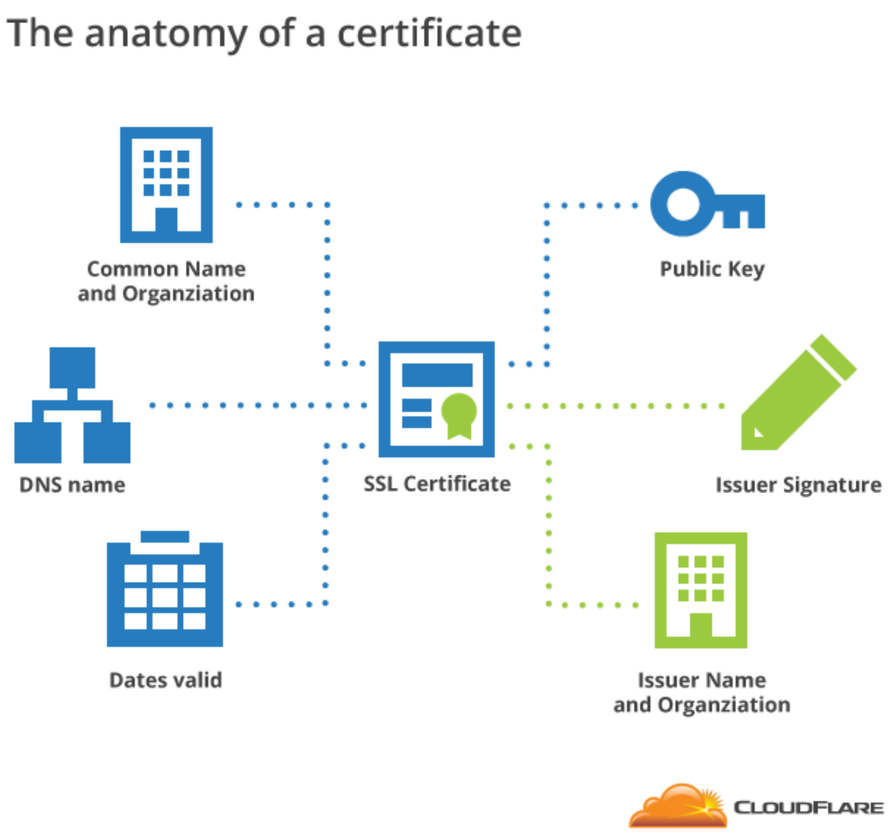
TLS Handshake
2-way Authenticated



https://en.wikipedia.org/wiki/File:Ssl_handshake_with_two_way_authentication_with_certificates.svg

- Same as unauthenticated client but includes the optional sending and verification of the client's certificate
- Not normally used for web service
- Commonly used to establish longer-duration connections for remote trusted parties (e.g. VPN connections, employee portal access applications, etc.)

Server or SSL/TLS Certificates



https://blog.cloudflare.com/how-to-build-your-own-public-key-infrastructure/

- The FQDN of the server is the certificate's subject
- Normally includes additional identity information such as organization name, address, email address, etc.
- Contain the public key used to decrypt messages from the server and encrypt messages that can only be decrypted by the server
- May be self-signed (only valid for encryption, not authentication), or may include a signature from the CA that is guaranteeing this certificate is valid
- Generally stored using base64 encoding with a header and footer, called PEM (Privacy-Enhanced Mail) format, but may be stored in other formats with other file extensions on Microsoft operating systems - see http://www.gtopia.org/blog/2010/02/der-vs-crt-vs-cer-vs-pem-certificates/

OpenSSL

- Openssl includes many tools for testing and diagnosing TLS/SSL connections
- It includes a diagnostic server mode, and a client mode
- You can trivially create a verbose server process that listens by default on port 4433

```
openssl s_server -cert certfile -key keyfile
```

- You can connect to the server using a verbose client to show the handshake and cipher negotiation

```
openssl s_client -connect host:port
```

- These are much easier to use for configuration debugging than something like wireshark

```
#   SSL Engine Switch:
#   Enable/Disable SSL for this virtual host.
SSLEngine on

#   A self-signed (snakeoil) certificate can be created by installing
#   the ssl-cert package. See
#   /usr/share/doc/apache2/README.Debian.gz for more info.
#   If both key and certificate are stored in the same file, only the
#   SSLCertificateFile directive is needed.
SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

#   Server Certificate Chain:
#   Point SSLCertificateChainFile at a file containing the
#   concatenation of PEM encoded CA certificates which form the
#   certificate chain for the server certificate. Alternatively
#   the referenced file can be the same as SSLCertificateFile
#   when the CA certificates are directly appended to the server
#   certificate for convinience.
#SSLCertificateChainFile /etc/apache2/ssl.crt/server-ca.crt

#   Certificate Authority (CA):
#   Set the CA certificate verification path where to find CA
#   certificates for client authentication or alternatively one
#   huge file containing all of them (file must be PEM encoded)
#   Note: Inside SSLCACertificatePath you need hash symlinks
#         to point to the certificate files. Use the provided
#         Makefile to update the hash symlinks after changes.
#SSLCACertificatePath /etc/ssl/certs/
#SSLCACertificateFile /etc/apache2/ssl.crt/ca-bundle.crt

#   Certificate Revocation Lists (CRL):
#   Set the CA revocation path where to find CA CRLs for client
#   authentication or alternatively one huge file containing all
#   of them (file must be PEM encoded)
#   Note: Inside SSLCARevocationPath you need hash symlinks
#         to point to the certificate files. Use the provided
#         Makefile to update the hash symlinks after changes.
#SSLCARevocationPath /etc/apache2/ssl.crl/
#SSLCARevocationFile /etc/apache2/ssl.crl/ca-bundle.crl

#   Client Authentication (Type):
#   Client certificate verification type and depth.  Types are
#   none, optional, require and optional_no_ca.  Depth is a
#   number which specifies how deeply to verify the certificate
#   issuer chain before deciding the certificate is not valid.
#SSLVerifyClient require
#SSLVerifyDepth  10

#   SSL Engine Options:
#   Set various options for the SSL engine.
#   o FakeBasicAuth:
#     Translate the client X.509 into a Basic Authorisation.  This means that
#     the standard Auth/DBMAuth methods can be used for access control.  The
#     user name is the `one line' version of the client's X.509 certificate.
#     Note that no password is obtained from the user. Every entry in the user
#     file needs this password: `xxj31ZMTZzkVA'.
#   o ExportCertData:
#     This exports two additional environment variables: SSL_CLIENT_CERT and
#     SSL_SERVER_CERT. These contain the PEM-encoded certificates of the
#     server (always existing) and the client (only existing when client
#     authentication is used). This can be used to import the certificates
#     into CGI scripts.
#   o StdEnvVars:
#     This exports the standard SSL/TLS related `SSL_*' environment variables.
#     Per default this exportation is switched off for performance reasons,
#     because the extraction step is an expensive operation and is usually
#     useless for serving static content. So one usually enables the
#     exportation for CGI and SSI requests only.
#   o OptRenegotiate:
#     This enables optimized SSL connection renegotiation handling when SSL
#     directives are used in per-directory context.
#SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire
```

Configuring websites for TLS/SSL

Apache2

- Apache2 keeps it config files in /etc/apache by default
- The websites served are normally defined in site files
- Default ssl site config file in /etc/apache2/sites-available/default-ssl.conf has most commonly used directives and their default settings
- Also has lots of comments on what the defaults are, what the choices are, and when and how to change them
- The apache website has the rest of the documentation
- Simple TLS protected websites can mostly use the defaults and just need a certificate/key specified for the website

Configuring Nginx for TLS/SSL

- Nginx has a different configuration file syntax, but the settings required are mostly the same as for Apache2
- The Nginx default configuration directory is /etc/nginx
- There are many ways to set up https under nginx, some more scalable than others
- See https://www.techrepublic.com/article/how-to-enable-ssl-on-nginx/ for an example of how to set up TLS for nginx servers