Lecture 14: Transport-Level Security

Stephen Huang

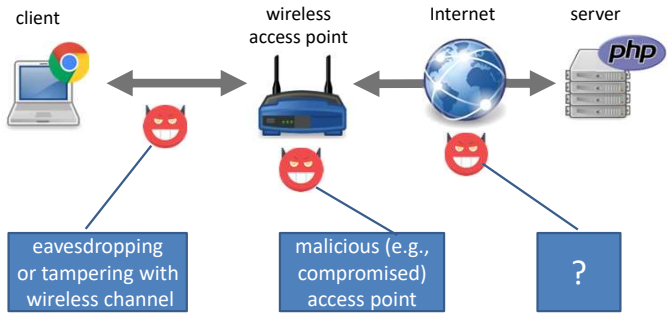
Content

1. Secure Sockets Layer (SSL)
2. Transport Layer Security (TLS)
3. HTTP over SSL/TLS
4. Conclusion

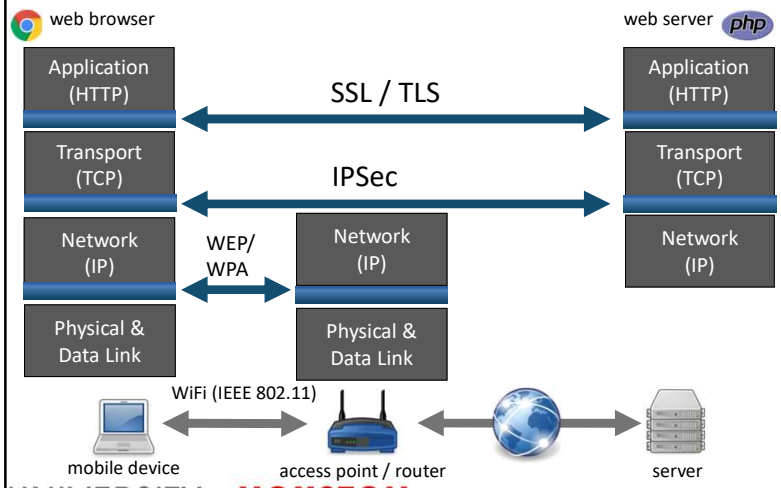
Although SSL implementations are still around, it has been deprecated by IETF and is disabled by most corporations offering TLS software

Review

- Communication Threats in Practice



Review



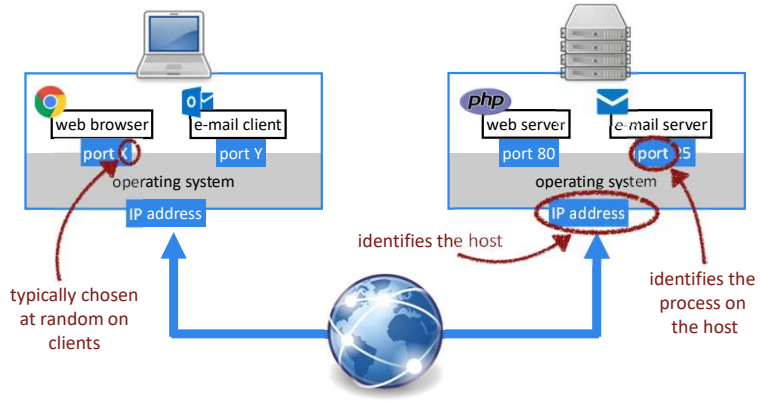
1. Secure Sockets Layer (SSL)

- End-to-end security between two applications, such as a client and a server.
- The Transport Layer of OSI transparently transfers data between end users, providing reliable data transfer services to the upper layers.
- Transportation Layer Protocols
  - Transmission Control Protocol(TCP)
  - User Datagram Protocol (UDP)

UNIVERSITY of HOUSTON 5

5

IP & Network Ports



UNIVERSITY of HOUSTON 6

6

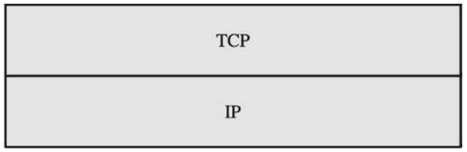
Secure Sockets Layer (SSL)

- End-to-end security between two applications (~TCP ports)
- Endpoint applications can implement it without the help of the operating systems or any intermediate devices
- Developed by Netscape for securing HTTP
  → HTTPS = HTTP over SSL
  - "father of SSL": Taher Elgamal
- Versions 1.0 (1994) and 2.0 (1995) had serious security issues
- Version 3.0 (1996)
  - complete redesign
  - very widely used, not just for HTTP (e.g., FTP, POP3, IMAP)
- Improved SSL 3.0 was standardized by IETF as TLS 1.0 in 1999.

UNIVERSITY of HOUSTON 7

7

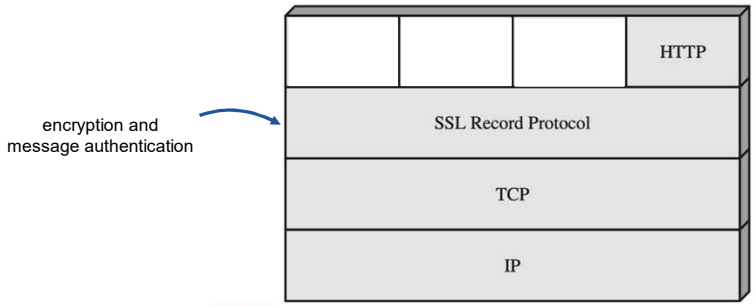
SSL Overview



UNIVERSITY of HOUSTON 8

8

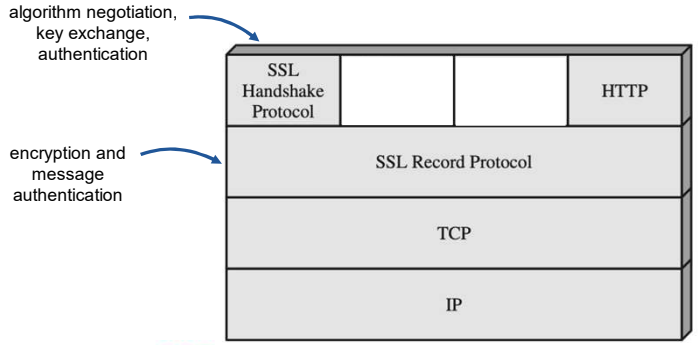
SSL Overview



UNIVERSITY of HOUSTON 9

9

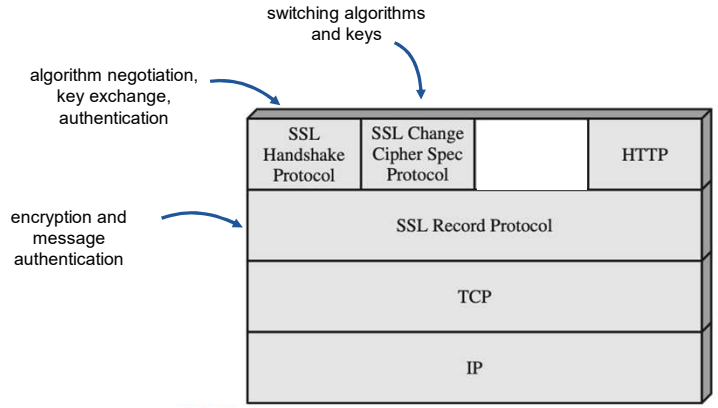
SSL Overview



UNIVERSITY of HOUSTON 10

10

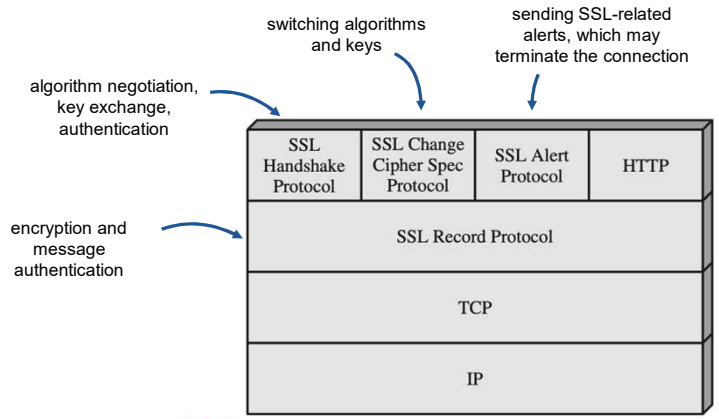
SSL Overview



UNIVERSITY of HOUSTON 11

11

SSL Overview

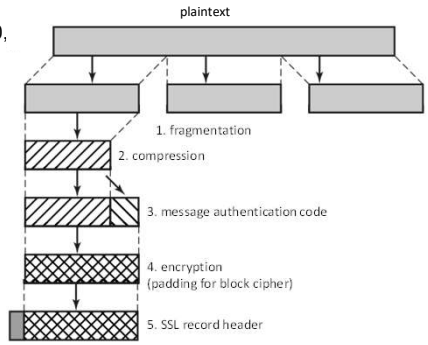


UNIVERSITY of HOUSTON 12

12

SSL Record Protocol

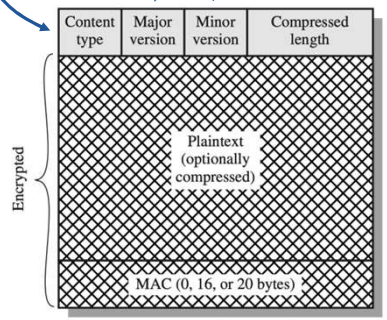
- Security services
  - **confidentiality**: symmetric-key encryption (AES GCM, Salsa20, …)
  - **integrity**: message authentication codes based on symmetric-key cryptography (HMAC-SHA256, …)
- Additional services
  - **fragmentation**: fragment application data into records of at most 16,384 bytes
  - **lossless compression**: optional (default is no compression)



SSL Record Format

specifies higher-level protocol (Handshake, ChangeCihpherSpec, Alert, or application)

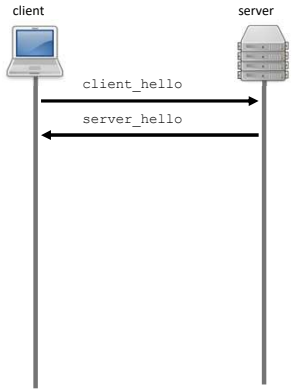
specifies SSL/TLS protocol version (*example:* SSL 3.0 → major = 3, minor = 0 TLS 1.0 → major = 3, minor = 1)



SSL Handshake Protocol

Phase 1: establish security capabilities

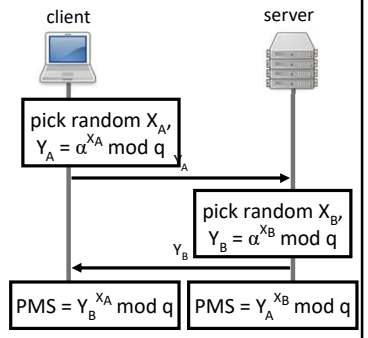
- client_hello:
  - highest SSL version supported by the client
  - nonce (timestamp + random value)
  - cipher suite: list of key-exchange methods, as well as encryption and MAC algorithms
  - compression method: list of supported compression algorithms
- server_hello:
  - highest SSL version supported by both the client and the server
  - nonce
  - chosen cipher suite and compression method



Key Exchange Methods

Goal: exchange or agree on a pre-master secret (PMS)

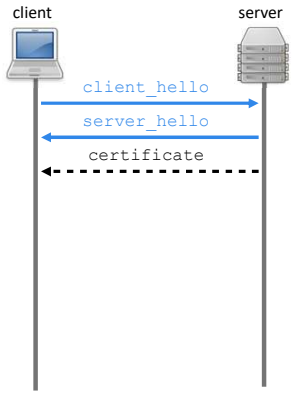
- RSA
  - client generates pre-master secret (PMS)
  - sends PMS to the server encrypted using RSA public-key encryption
- Diffie-Hellman protocol
  - anonymous D-H: basic D-H with no authentication
  - fixed D-H: D-H parameters of the server (X_A and Y_A) are fixed, and Y_A is contained in a digital certificate
  - ephemeral D-H: D-H with authentication



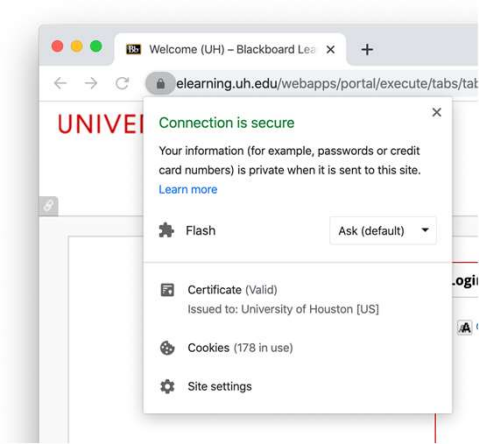
SSL Handshake Protocol: Phase 2

Phase 2: server authentication and key exchange

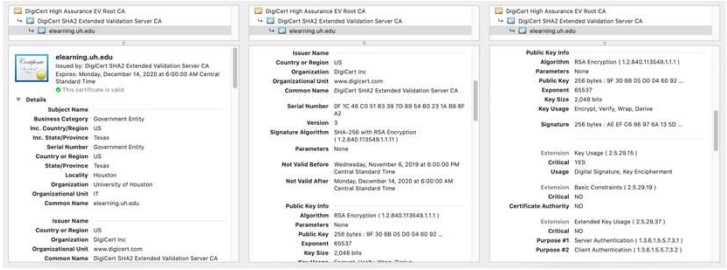
- `certificate` (optional): X.509 certificate (may be a chain)



X.509 Certificate in Practice



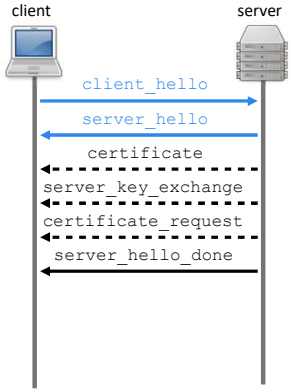
X.509 Certificate in Practice



SSL Handshake Protocol: Phase 2

Phase 2:
server authentication and key exchange

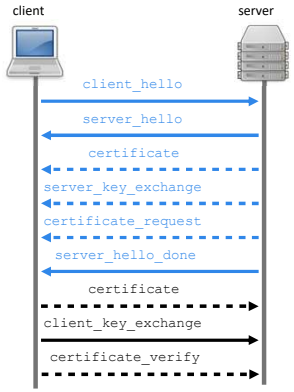
- `certificate` (optional): X.509 certificate (may be a chain)
- `server_key_exchange` (optional):
  - parameters for Anonymous or Ephemeral Diffie-Hellman exchange
  - public-key for RSA exchange if the certificate contains only a signing key
  - signed by the server (together with the nonces)
- `certificate_request` (optional): ask client for an X.509 certificate
- `server_hello_done`: server is finished



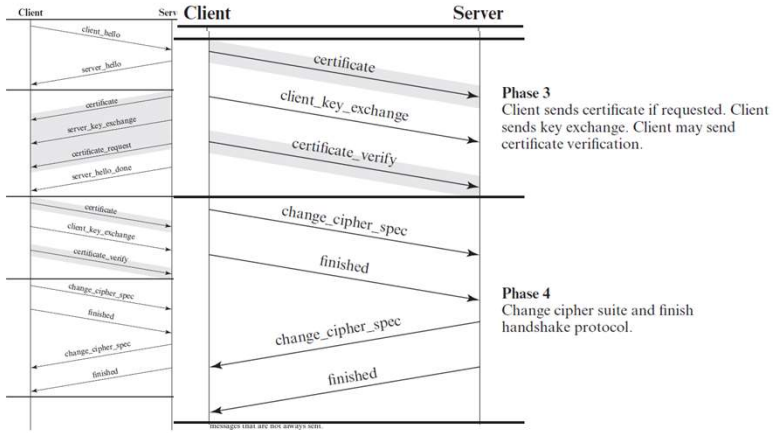
SSL Handshake Protocol: Phase 3

Phase 3: client authentication and key exchange

- `certificate` (optional): X.509 certificate if the server asked for a client certificate
- `client_key_exchange`:
  - pre-master secret encrypted using public RSA key of the server
  - parameters for D-H exchange
- `certificate_verify` (optional): digital signature of all previous handshake messages



Handshake Protocol Action



Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

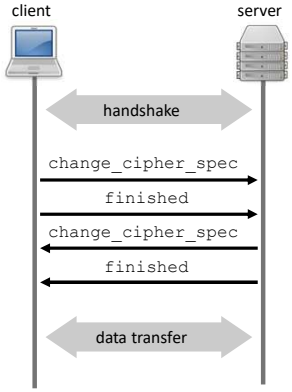
Key Generation

- Reminder: pre-master secret
  - RSA exchange → generated by the client and sent to the server
  - D-H exchange → generated using the D-H protocol
- Master secret
  - from pre-master secret and the nonces
  - generated using HMAC with SHA hash function
- Keys and secrets
  - for message-authentication key, encryption key, and IV
  - for both directions (client sending and server sending)
  - generated from master secret and the nonces using HMAC with SHA hash

SSL Change Cipher Spec Protocol

Same for client and server

- `change_cipher_spec`: signals that the communication party is switching to the negotiated cryptographic algorithms and keys
- `finished`: hash value computed from the master secret and all handshake messages using HMAC with SHA hash function

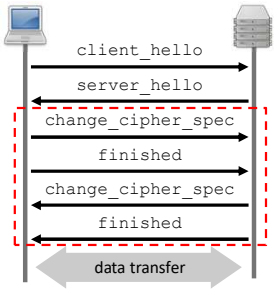


Session Resume

- When a client and server establish an SSL connection for the first time, they establish a shared key called the `master_secret`.
- The `master_secret` is then used to create all the bulk encryption keys to protect the traffic.
- The `master_secret` is almost invariably established using a public key algorithm.
- SSL contains a "session resumption" feature that skips this time-consuming step if they have already established a `master_secret` previously.

Session Resume

- Authentication and key exchange are complex & result needs to be reusable
- Session
  - association between a client and a server
  - cipher suite, compression method, and master secret
- Connection
  - within a session
  - keys and IVs for encryption and message authentication
- Session ID: identifies a session
  - sent in ClientHello → may specify an existing session to be resumed
  - sent in ServerHello → server can accept resume by sending the same ID



- Session resume skips all messages in the Handshake after the ClientHello and ServerHello messages
  - new keys and IVs are generated from the nonces in the Hello messages

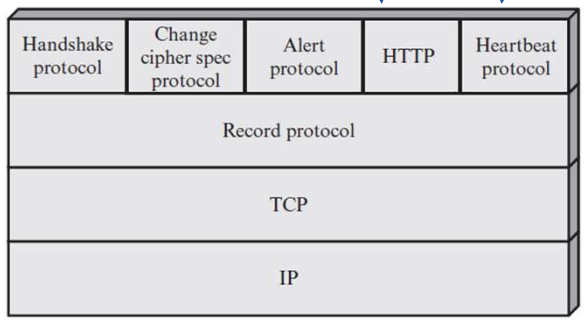
2. Transport Layer Security (TLS)

- In 1999, IETF standardized TLS 1.0 in RFC 2246
  - only minor differences compared to SSL 3.0:
    - pseudorandom function for generating keys and MAC is based on HMAC
    - variable length padding (may prevent traffic analysis)
    - other minor changes
- TLS 1.1 (RFC 4346) and 1.2 (RFC 5246), released in 2006 and 2008
  - minor changes to the protocol and updated cipher suites
- SSL 3.0 was deprecated in June 2015 by the IETF
- TLS 1.3 (RFC 8446) released in August 2018
  - changes improving security and updated cipher suites (e.g., separating key agreement from cipher suites, removing MD5, adding ChaCha20)

TLS Protocol Stack

provides the transfer service for web interaction and can operate on top of TLS.

to detect the failure, shutdown or unavailability of nodes belonging to a network cluster



TLS 1.2

Handshake

- The client sends the server information, including the highest version of SSL that it supports and a list of the cipher suites that it supports.
  - The cipher suite information includes cryptographic algorithms and key sizes.
  - Extensions: `supported_versions`, `key_share`, `pre_sared_key`

Handshake

- **Server hello**: The server chooses the highest version of SSL and the best cipher suite that both the client and server support and sends this information.
- **Certificate for Server Authentication**: The server sends the client a certificate (optional)
- **Server key exchange**: The server sends the client a server key exchange message if the public key information from the Certificate is not sufficient for key exchange (optional).

Handshake

- **Certificate Request**: The server sends the client a certificate request (optional).
- **Server Hello Done**: The server tells the client that it is finished with its initial negotiation messages.
- **Certificate**: The client sends its certificate if requested by the server.

Handshake

- **Client key exchange**: The client generates information used to create a key to use for symmetric encryption.
  - For RSA, the client then encrypts this key information with the server's public key and sends it to the server.
  - For cipher suites based on Diffie-Hellman (DH), this message contains the client's DH public key.

Handshake

- **Certificate Verify**: This message is sent by the client when the client presents a certificate (optional).
  - The client sends information that it digitally signs using a cryptographic hash function.
  - When the server decrypts this information with the client's public key, the server is able to authenticate the client.

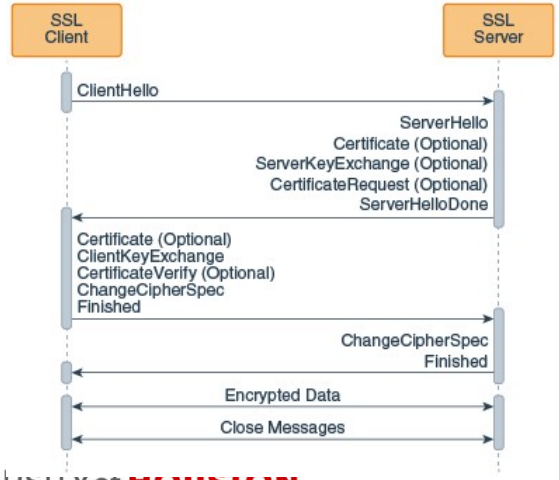
Handshake

- **Change Cipher Spec**: The client sends a message telling the server to change to encrypted mode.
- **Finished:** The client tells the server that it is ready for secure data communication to begin.
- **Change Cipher Spec**: The server sends a message telling the client to change to encrypted mode.
- **Finished**.

Handshake

- **Encrypted Data**: The client and the server communicate using the symmetric encryption algorithm and the cryptographic hash function negotiated and using the secret key that the client sent to the server during the client key exchange.
- **Close Messages**: At the end of the connection, each side sends a close_notify.

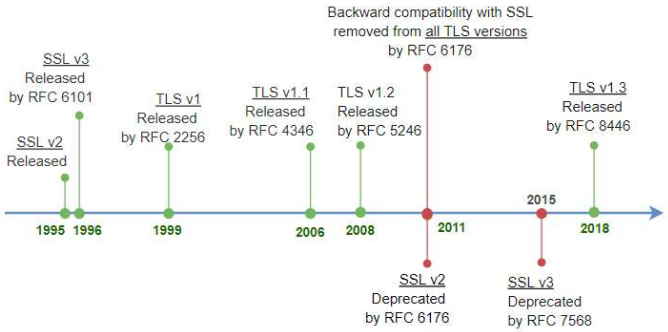
TLS 1.2



Example Vulnerabilities

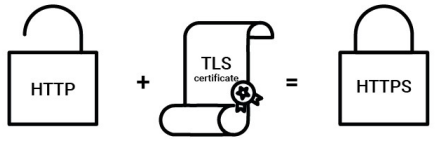
- Logjam (2015)
  - "export-grade" 512-bit versions of the Diffie-Hellman key exchange were implemented in the 1990s because of US export restriction on cryptography
  - active attacker may trick the client and server into using a weak key exchange
    → attacker can recover the PMS and, hence, all keys and IVs
- Sweet32 (2016)
  - 64-bit block ciphers (*e.g.*, 3DES) in CBC mode
    → after only 2^{32} blocks, repeated cipher blocks are very likely
  - if C_i and C_j are equal, attacker can compute $P_i \oplus P_j = C_{i-1} \oplus C_{j-1}$
- Implementation errors
  - these are not vulnerabilities in the protocol
  - *examples:* OpenSSL Heartbleed (2014), Apple `goto fail;` (2014)

Timeline



3. HTTPS

- HTTPS = Hypertext Transfer Protocol Secure, or
- HTTP over SSL/TLS.

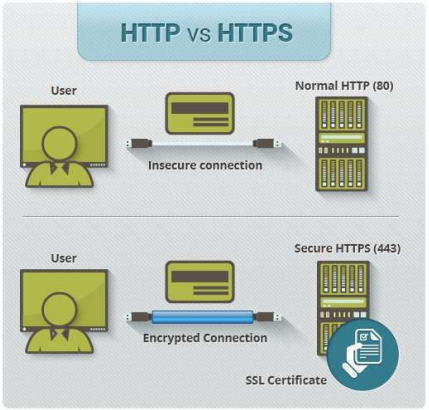


HTTP over SSL/TLS

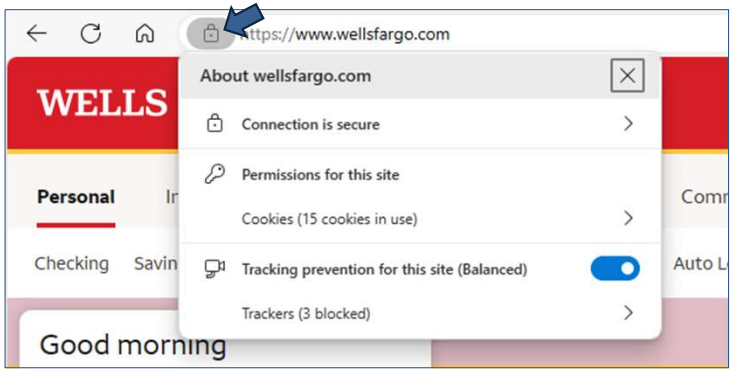
- Between the web browser and server:
  HTTP client = SSL client,
  HTTP server = SSL server
- Conventions
  - URL: `https://` instead of `http://`
  - default TCP port is 443 instead of 80
- HTTP request may be sent after SSL Finished messages
- Protected information
  - URL, contents of the document, browser forms, cookies, headers
  - page served over HTTPS may include elements retrieved using HTTP

HTTP vs HTTPS

- More than 2 million SSL certificates are typically issued daily.
- Secure HTTPS Pages account for 93.2% of Chrome Users' Surfing Time.



HTTPS



4. SSL/TLS: Conclusion

- Security between a client and a server applications (~TCP ports).
- Provides
  - confidentiality and integrity
  - authentication (typically for the server)
  - some protections against traffic analysis
- Very widely used: HTTPS, FTPS, SMTPS, IMAP / POP over SSL
- Versions currently in use: TLS 1.2 and TLS 1.3
- Multiple vulnerabilities have been discovered
  - most of them are not practical
  - addressed in implementation or minor protocol revisions

Next Topic

- Transport-Layer Security
- Application-Layer Protocols