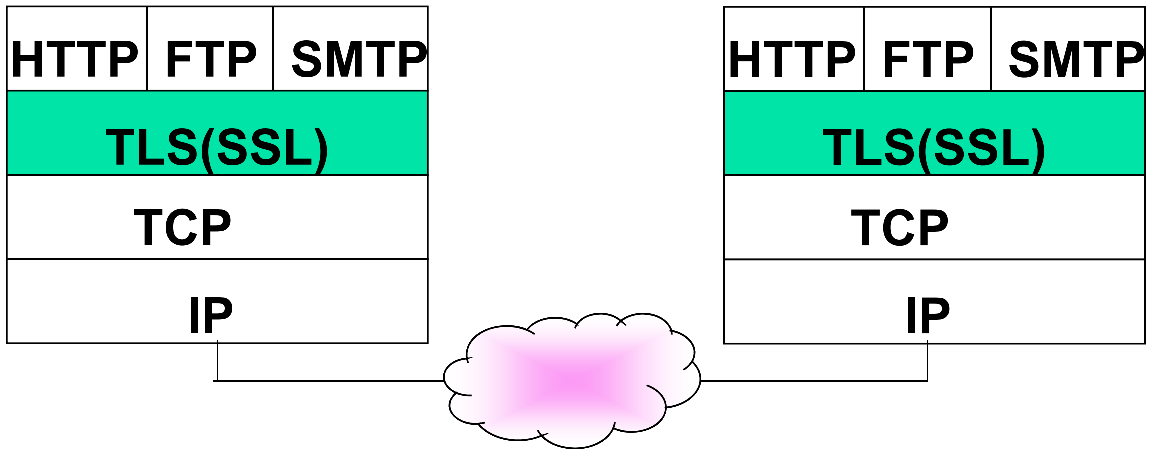
TLS/SSL

2019. 5. 7

TLS

- It is most widely used transport-layer security protocol.
- It can be applied to any applications which are working on TCP/IP, such as web application, email, etc.

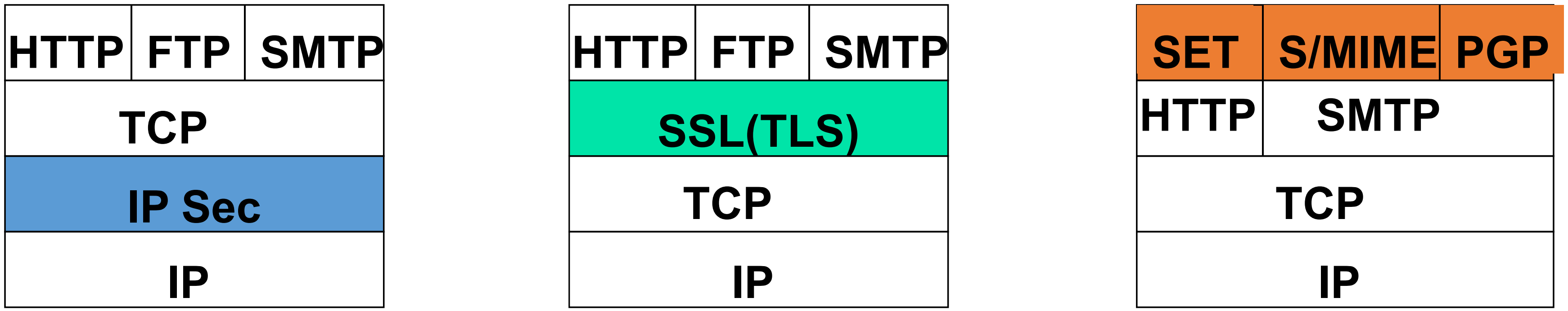


Comparison of security protocols at other layers

network-layer security protocol

transport-layer security protocol

application-specific security protocol



Brief history

- SSL v1
  - Designed by Netscape, never deployed
- SSL v2
  - Deployed in Netscape Navigator 1.1 in 1995
- SSL v3
  - Substantial overhaul, fixing security flaws, publicly reviewed (RFC 6101)

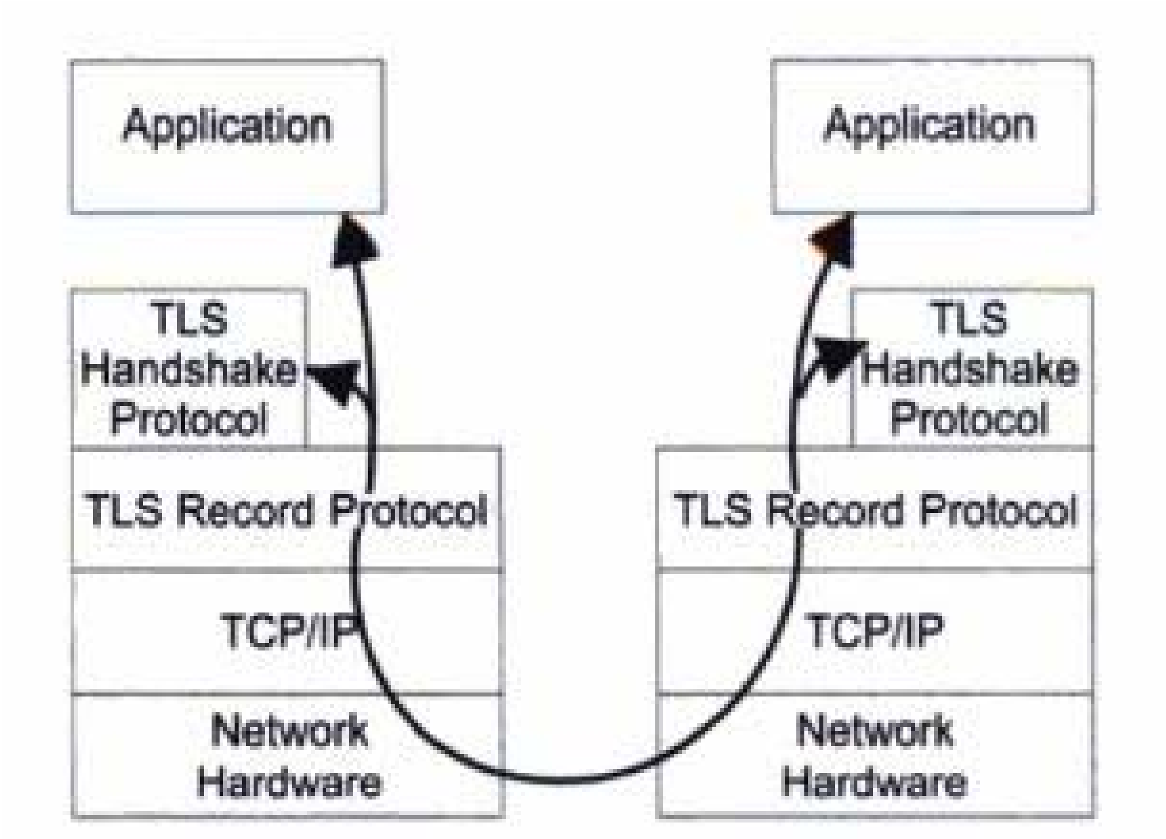
TLS(Transport Layer Security)

- TLS 1.0
  - IETF standard (RFC 2246) in 1999
  - SSLv3 with little tweak
- TLS 1.1
  - Update from TLS 1.0 (RFC 4346) in 2006
- TLS 1.2
  - RFC 5246 in 2008
- TLS 1.3
  - Published in 2018 Aug.

What TLS can do

- TLS provides secure communication channel over TCP
- Suppose that you want to buy a book at amazon.com
  - You want to be sure you are dealing with Amazon (server authentication)
  - Your credit card information must be protected in transit (confidentiality and/or integrity)
  - As long as you have money, Amazon does not care who you are (client authentication optinally)
  - So, no need for mutual authentication

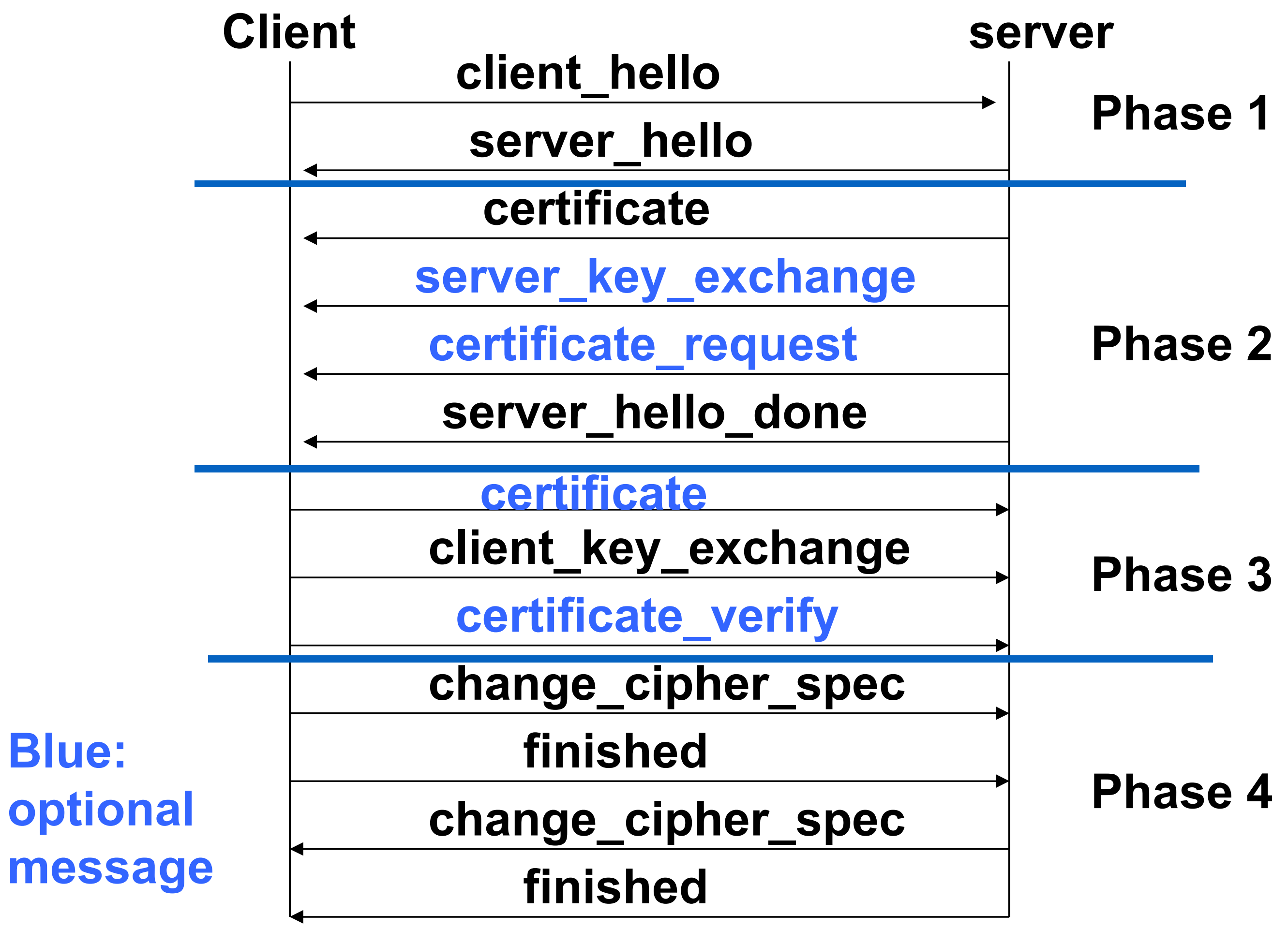
TLS layers


Application
Application
TLS Handshake Protocol
TLS Handshake Protocol
TLS Record Protocol
TLS Record Protocol
TCP/IP
TCP/IP
Network Hardware
Network Hardware

TLS procedure

- Handshake
  - Authenticate server
  - Exchange parameters to compute keys
- Keys computation
- Secure data exchange
  - Fragment into TLS records (append MAC and encryption)
- Session termination
  - Special messages to securely close connection

TLS Handshake Protocol



Phase 1: establish security capabilities

- {client, server}_hello message
  - Version: the highest SSL version
  - Random
    - 32-bit timestamp
    - 28 bytes random number
  - Session ID
  - Cipher suite
    - client_hello: Ciphers are listed in decreasing order of preference
    - server_hello: chosen cipher
  - Compression method

Cipher Suite

- Cipher suite
  - (key exchange methods, cipher spec)
- Key exchange methods
  - RSA: encrypt key with receiver's public key
  - Fixed Diffie-Hellman
    - Server's certificate contains DH public parameters signed by CA. The client provides its DH public parameters either in a certificate or in a key exchange message.
  - Ephemeral Diffie-Hellman
    - Certificate contains server's public key.
    - DH public parameters are signed using the server's private key.
  - Anonymous Diffie-Hellman
    - Each side sends its DH public parameters to the other without authentication.

Phase 2: Server authentication and key exchange

- S → C: certificate
  - RSA: Certificate contains server's public key
  - Fixed DH: Certificate contains DH public parameters signed by CA.
  - Ephemeral DH: Certificate contains DH public key, plus signature
- S → C: server_key_exchange
  - Anonymous DH: $\{g, p, g^s\}$
  - Ephemeral DH: $\{g, p, g^s\}$ + signature of $\{g, p, g^s\}$
  - RSA: if server's key is only for a signature-only key, the server create a temporary RSA public/private keys and send the temporary public key
- S → C: certificate_request
  - Certificate_type (RSA or DSS for key exchange)
  - List of acceptable certificate authorities
- S → C: server_hello_done, no parameters
- A signature is created by computing hash(client_rand || server_rand || server parameters) and encrypting it with the sender's private key.

Phase 3

- After phase 2, client has all values required to generate the session key
- C → S: certificate
  - If server requested a certificate
- C → S: client_key_exchange
  - RSA: client generates 48 byte pre-master secret, encrypts it with server's public key or temporary RSA key from a server_key_exchange message.
  - Ephemeral or anonymous DH: client's DH public parameters
  - Fixed DH: null (certificate contained client's DH key)
- C → S: certificate_verify
  - Only used if client sent certificate with signing key K_C
  - CertificateVerify.signature.md5_hash = {MD5(master secret || pad2 || MD5(handshake messages || master secret || pad1))}$_{K_C}$-1

Phase 4

- After phase 3, client and server share master secret computed from pre-master secret, and authenticated each other
- Phase 4: Finish
- C → S: change_cipher_spec
  - Copies the pending Cipher spec in the current CipherSpec.
- C → S: finished
  - MD5(master_secret || pad2 || MD5(handshake messages || Sender || master_secret || pad1)) ||
    SHA-1(master_secret || pad2 || SHA-1(handshake messages || Sender || master_secret || pad1))
  - pad1 and pad2 are the values defined earlier for the MAC
  - Handshake messages contains all messages up to now
- S → C: change_cipher_spec
- S → C: finished

Key computation

- Client and server perform DH calculation to create the shared pre-master secret (PS) when they chose DH key exchange.
- Master secret (MS) created from pre-master secret (PS), Client random (CR), Server random (SR)
  - MS = MD5(PS || SHA-1(‘A’ || PS || CR || SR)) ||
    MD5(PS || SHA-1(‘BB’ || PS || CR || SR)) ||
    MD5(PS || SHA-1(‘CCC’ || PS || CR || SR))
- CipherSpec requires client & server MAC key, client & server encryption key, client & server IV, generated from MS:
  - MD5(MS || SHA-1(‘A’ || MS || SR || CR)) ||
    MD5(MS || SHA-1(‘BB’ || MS || SR || CR)) ||
    MD5(MS || SHA-1(‘CCC’ || MS || SR || CR)) ||
    MD5(MS || SHA-1(‘DDDD’ || MS || SR || CR)) || …

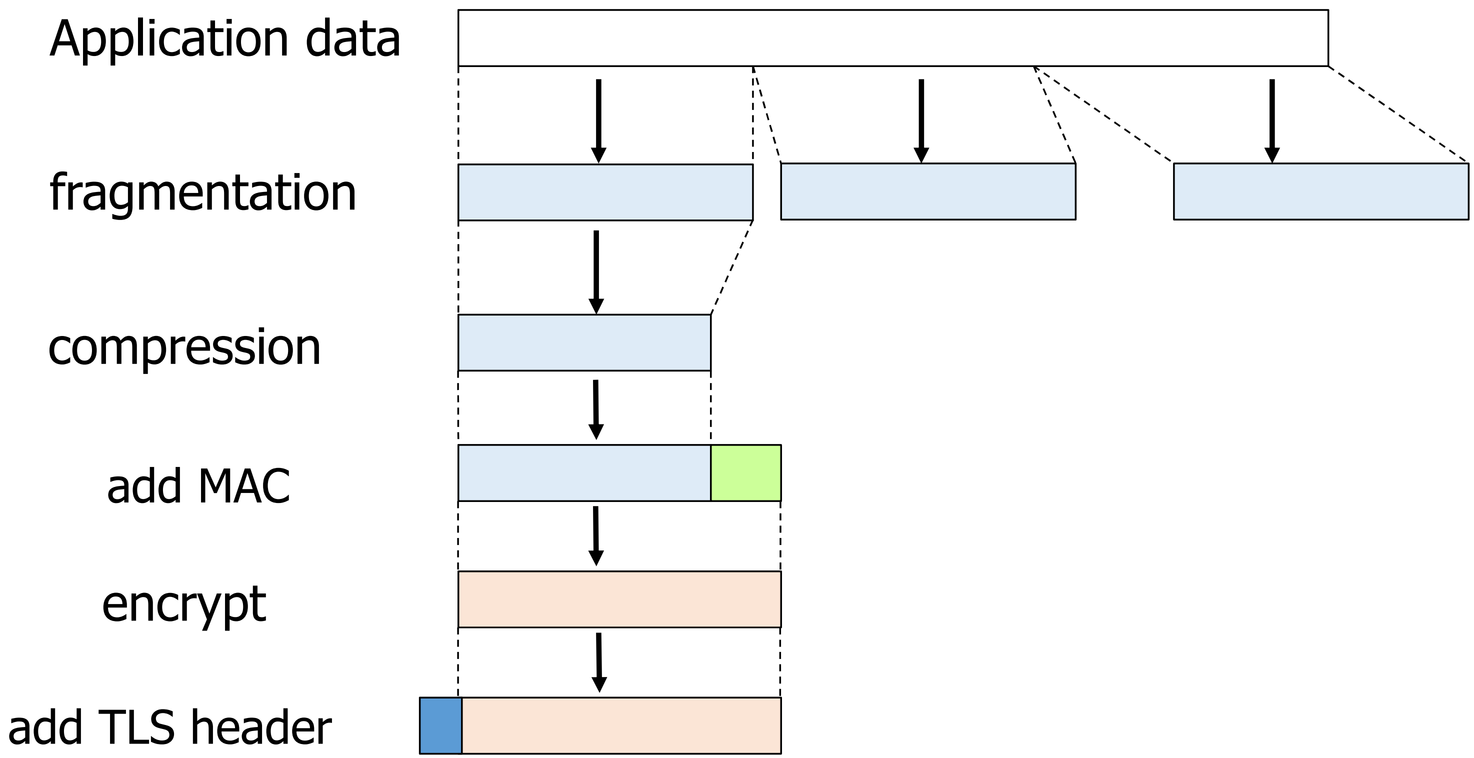
Sample TLS Handshake

- Client has no certificate, only server authenticated
- C → S: client_hello
- S → C: server_hello
  - Ephemeral DH key exchange, RC4 encryption, MD5-based MAC
- S → C: Server certificate, containing RSA public key
  - Client checks validity + verifies URL matches certificate
- S → C: Server_key_exchange: g, p, g^S, $\{H(g, p, g^S)\}_{K_S^{-1}}$
- S → C: server_hello_done
- C → S: client_key_exchange: g^C
- C → S: change_cipher_spec
- C → S: finished
- S → C: change_cipher_spec
- S → C: finished

Key computation

- In the previous example, compute pre-master secret from $\{g, p, g^A, g^B\}$.
- Compute master secret from pre-master secret.
- From client nonce(R_A), server nonce(R_B), and master secret, compute the following 4 keys and 2 IVs.
    - client MAC key
    - server MAC key
    - client encryption key
    - server encryption key
    - client initialization vector (IV)
    - server initialization vector (IV)

From application data to TLS record



TLS record format