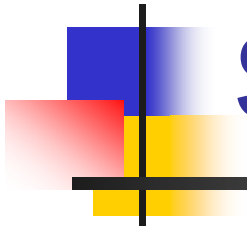


SSL/TLS

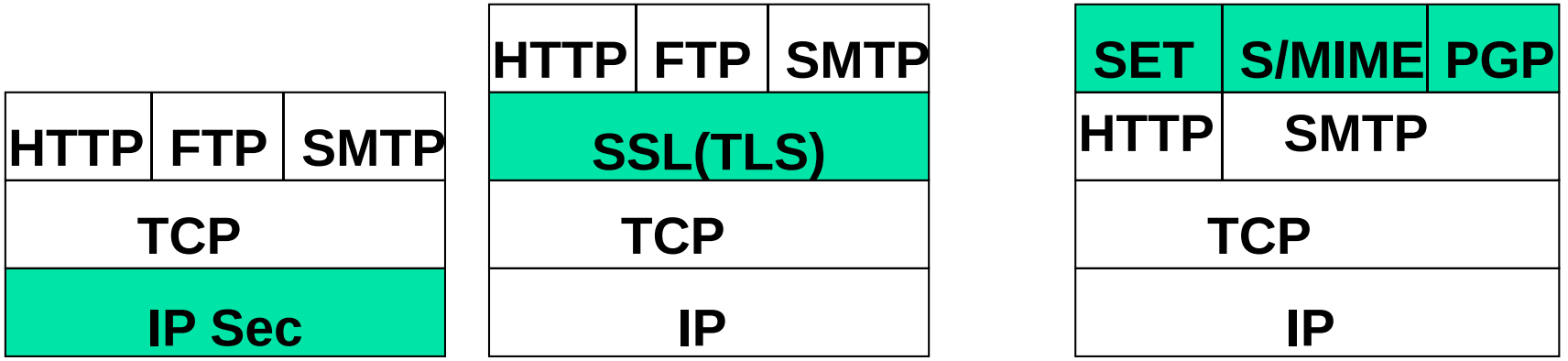
Web security requirements

- Security requirements
  - Secrecy to prevent eavesdroppers to learn sensitive information
  - Entity authentication
  - Message authentication and integrity to prevent message alteration / injection

Web security approach

- IP Security
  - Apply security at the IP layer
  - So, it is transparent to all end users and applications above IP
  - Most general approach
- SSL(Secure Sockets Layer protocol)
  - Security above TCP
- Application-specific security
  - Dedicated to specific application
  - embedded within the particular application

Comparison of web security approaches



SSL 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 protocol)

- 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
  - Currently working draft

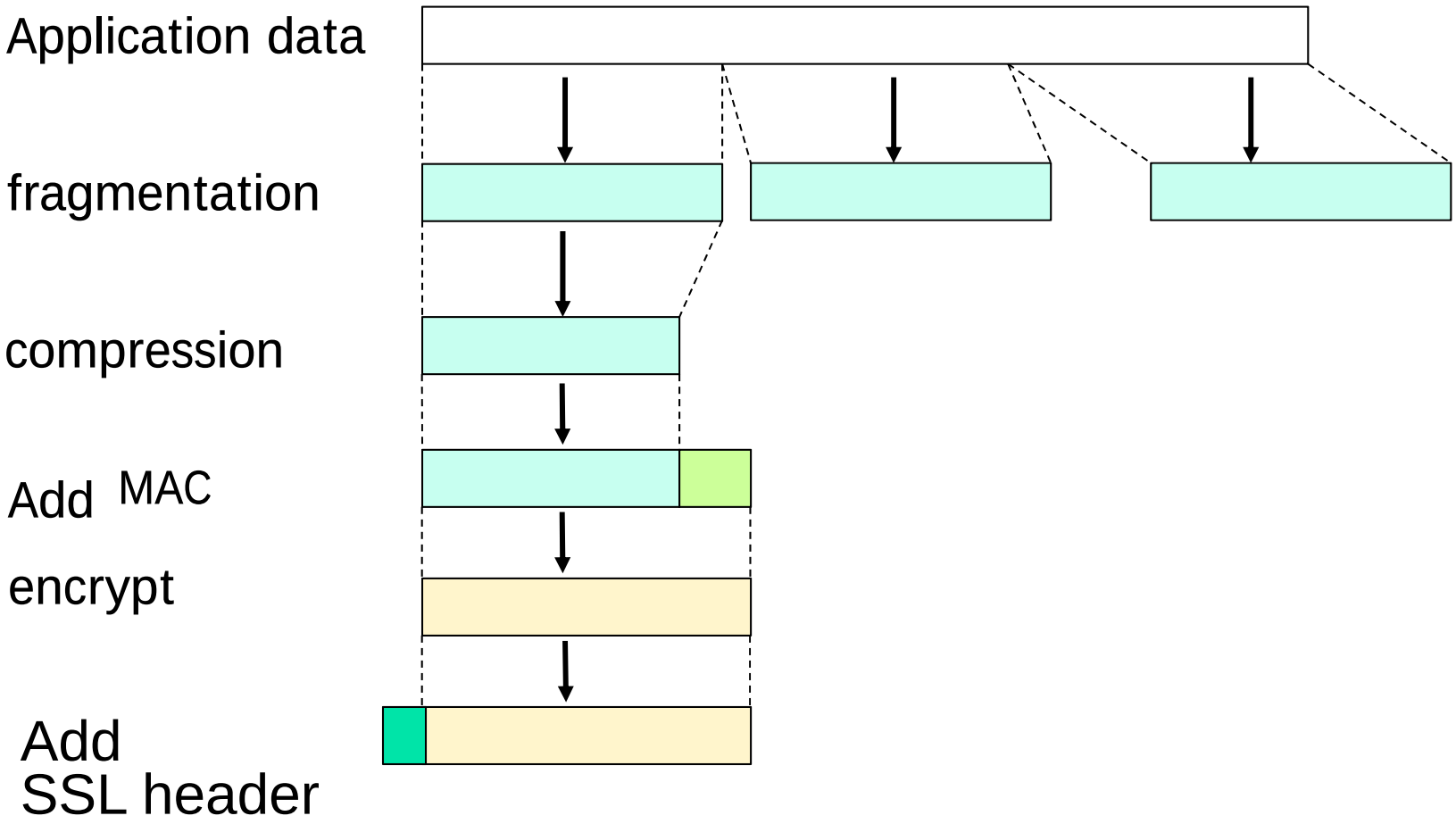
What SSL can do

- In a word, SSL provides secure communication channel.
- 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)
  - So, no need for mutual authentication

What SSL can’t do?

- Parameter tampering attacks that modify the values of URL or other parameters in a request at the browser.
- Scripting or SQL injection attacks.
- Attack on data stored on the client computer.
- Attack on data stored on the server computer
- Data after it reaches the server and is decrypted.

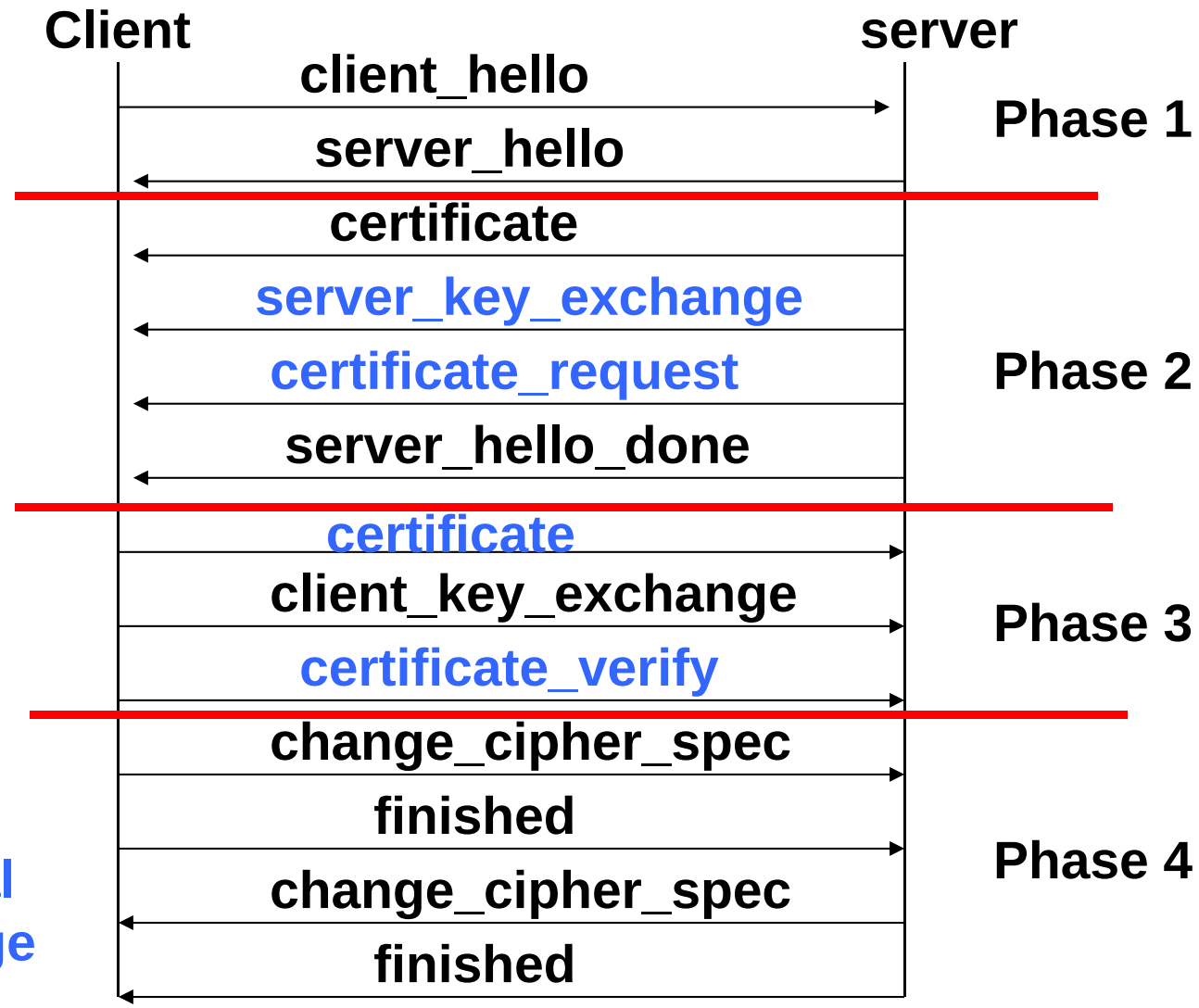
From application data to SSL record



SSL procedure

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

SSL Handshake Protocol



Blue: optional message

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.

- Cipher spec
  - Cipher Algorithm (RC4, RC2, DES, 3DES, DES40, IDEA)
  - MAC Algorithm (MD5, SHA-1)
  - Cipher Type (stream or block)
  - Is Exportable (true or false)
  - Hash size (0 or 16 bytes for MD5, 20 bytes for SHA-1)
  - IV size: IV size for Cipher Block Chaining(CBC) encryption

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
  - Cert_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

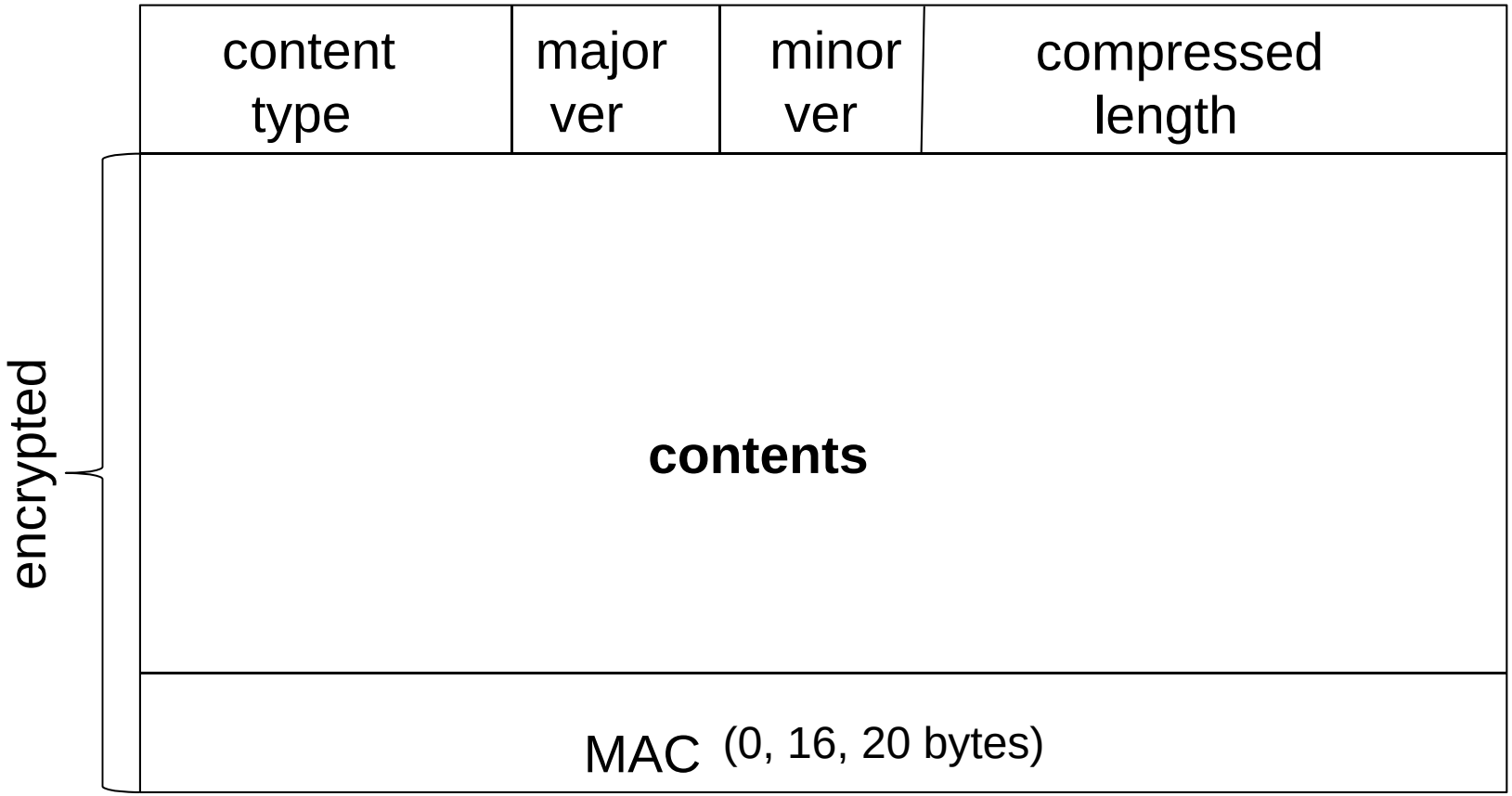
Phase 4

- 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

Cryptographic computation

- Client and server perform DH calculation to create the shared pre-master secret (PS).
- 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)) || …

SSL record format



SSL Record format

- SSL Record = Content type || Major version || Minor version || Length ||{ Data || MAC(K', Data) }$_K$
- MAC(K', Data) = hash(K' || pad2 || hash(K' || pad1 || seq_num || compressed_type || length || Data))
  - hash: MD5 or SHA-1
  - pad1 = 0x363636…
  - pad2 = 0x5C5C5C…
  - seq_num: sequence number for message

Sample SSL session

- 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)