

Chapter 8

Network Security

A S M Touhidul Hasan, Ph.D.
Assistant Professor,
Department of CSE, UAP

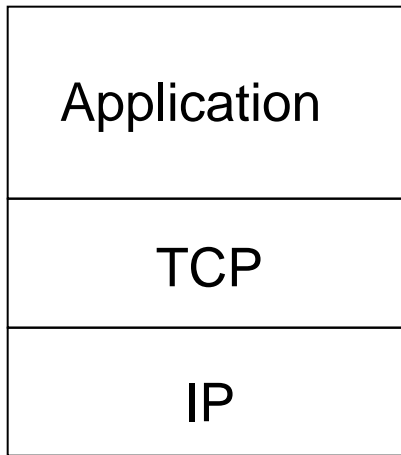
Chapter 8 Roadmap of Lecture 24

8.5 Securing TCP connections: SSL

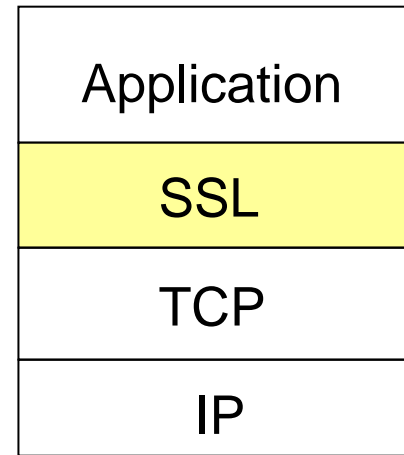
SSL: Secure Sockets Layer

- ❖ widely deployed security protocol
 - supported by almost all browsers, web servers
 - https
 - billions \$/year over SSL
- ❖ mechanisms: [Woo 1994], implementation: Netscape
- ❖ variation -TLS: transport layer security, RFC 2246
- ❖ provides
 - *confidentiality*
 - *integrity*
 - *authentication*
- ❖ original goals:
 - Web e-commerce transactions
 - encryption (especially credit-card numbers)
 - Web-server authentication
 - optional client authentication
 - minimum hassle in doing business with new merchant
- ❖ available to all TCP applications
 - secure socket interface

SSL and TCP/IP



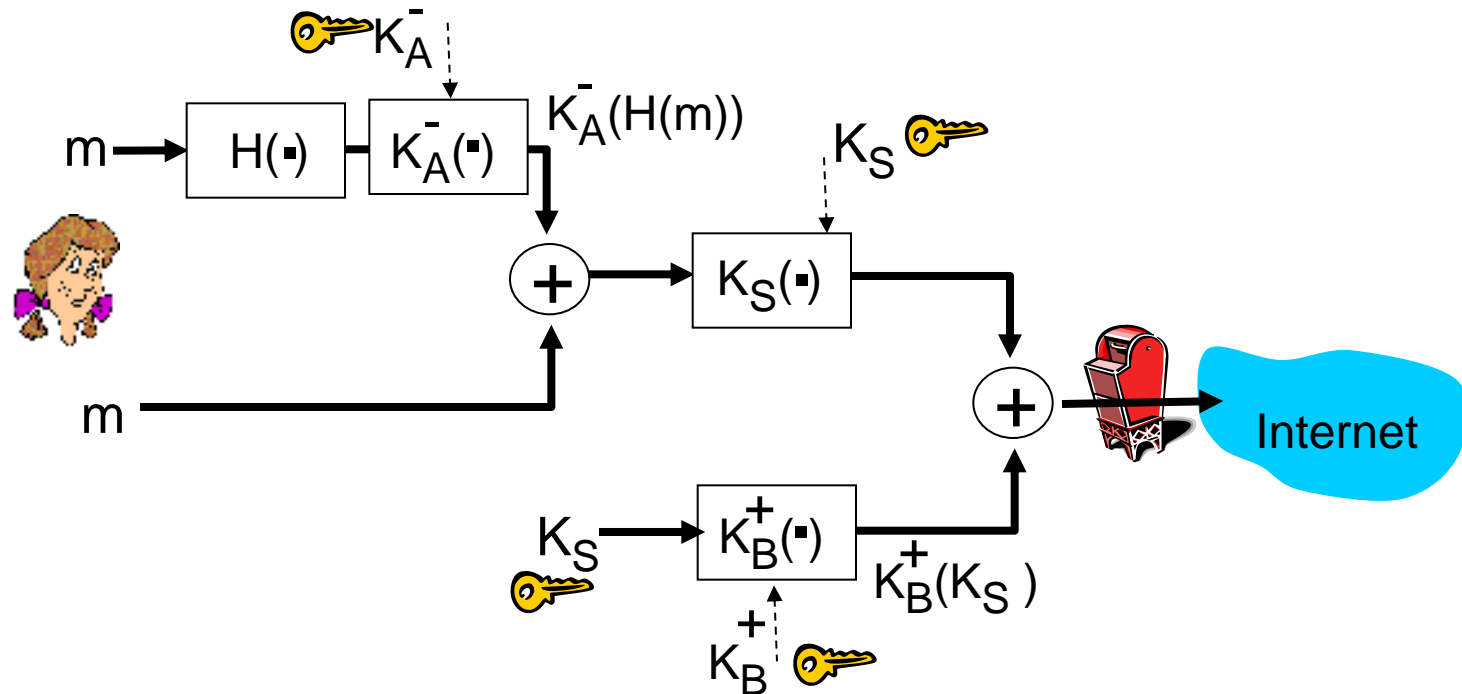
normal application



application with SSL

- ❖ SSL provides application programming interface (API) to applications
- ❖ C and Java SSL libraries/classes readily available

Could do something like PGP:

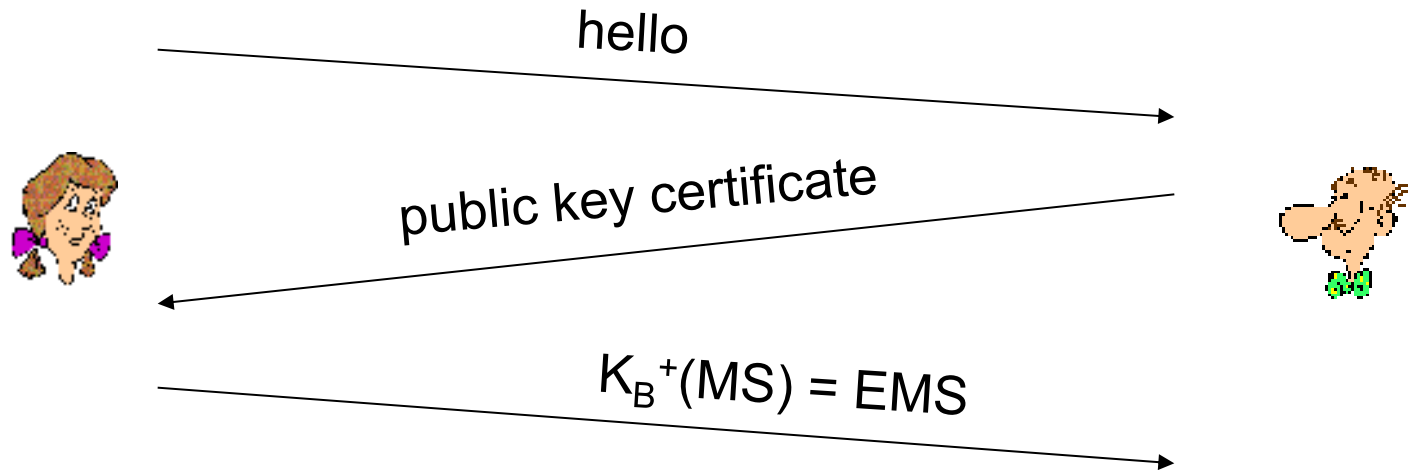


- ❖ but want to send byte streams & interactive data
- ❖ want set of secret keys for entire connection
- ❖ want certificate exchange as part of protocol: handshake phase

Toy SSL: a simple secure channel

- ❖ *handshake*: Alice and Bob use their certificates, private keys to authenticate each other and exchange shared secret
- ❖ *key derivation*: Alice and Bob use shared secret to derive set of keys
- ❖ *data transfer*: data to be transferred is broken up into series of records
- ❖ *connection closure*: special messages to securely close connection

Toy: a simple handshake



MS: master secret

EMS: encrypted master secret

Toy: key derivation

- ❖ considered bad to use same key for more than one cryptographic operation
 - use different keys for message authentication code (MAC) and encryption
- ❖ four keys:
 - K_c = encryption key for data sent from client to server
 - M_c = MAC key for data sent from client to server
 - K_s = encryption key for data sent from server to client
 - M_s = MAC key for data sent from server to client
- ❖ keys derived from key derivation function (KDF)
 - takes master secret and (possibly) some additional random data and creates the keys

Toy: data records

- ❖ why not encrypt data in constant stream as we write it to TCP?
 - where would we put the MAC? If at end, no message integrity until all data processed.
 - e.g., with instant messaging, how can we do integrity check over all bytes sent before displaying?
- ❖ instead, break stream in series of records
 - each record carries a MAC
 - receiver can act on each record as it arrives
- ❖ issue: in record, receiver needs to distinguish MAC from data
 - want to use variable-length records



Toy: sequence numbers

- ❖ *problem:* attacker can capture and replay record or re-order records
- ❖ *solution:* put sequence number into MAC:
 - $MAC = MAC(M_x, \text{sequence} || \text{data})$
 - note: no sequence number field
- ❖ *problem:* attacker could replay all records
- ❖ *solution:* use nonce

Toy: control information

- ❖ *problem*: truncation attack:
 - attacker forges TCP connection close segment
 - one or both sides thinks there is less data than there actually is.
- ❖ *solution*: record types, with one type for closure
 - type 0 for data; type 1 for closure
- ❖ $MAC = MAC(M_x, \text{sequence}||\text{type}||\text{data})$

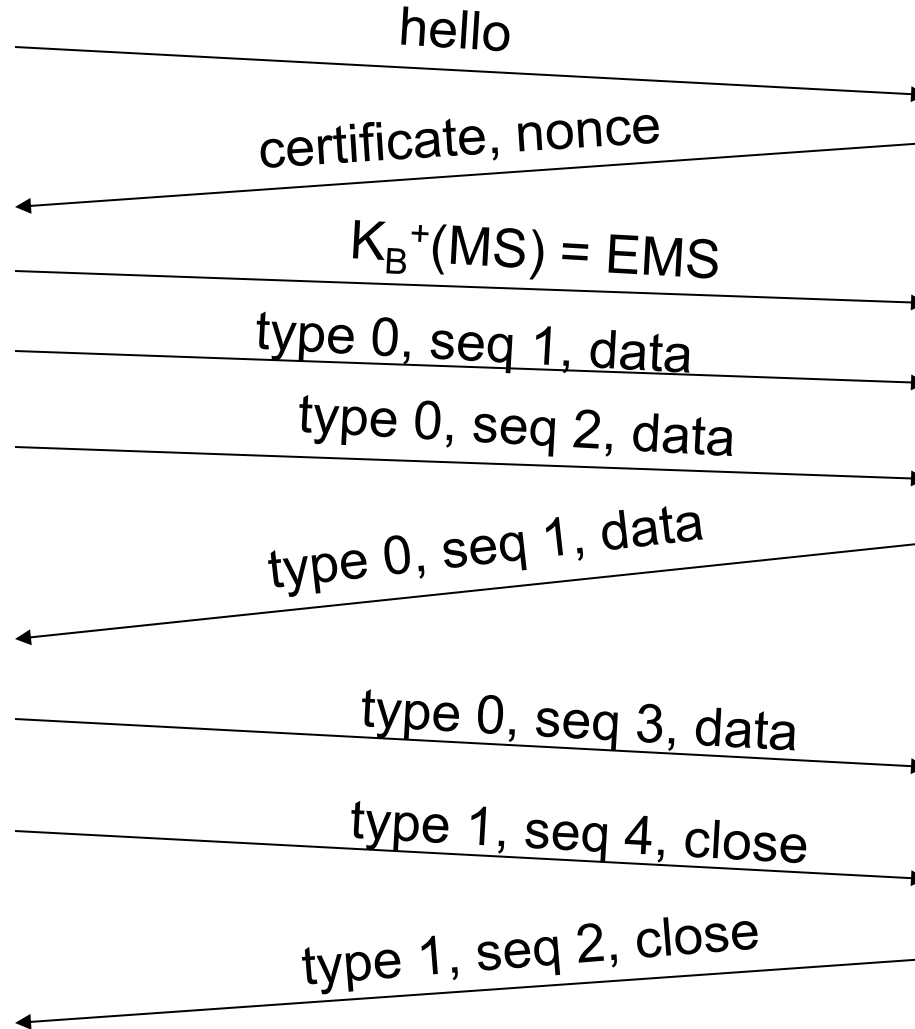


Toy SSL: summary



bob.com

encrypted



Toy SSL isn't complete

- ❖ how long are fields?
- ❖ which encryption protocols?
- ❖ want negotiation?
 - allow client and server to support different encryption algorithms
 - allow client and server to choose together specific algorithm before data transfer

SSL cipher suite

❖ cipher suite

- public-key algorithm
- symmetric encryption algorithm
- MAC algorithm

❖ SSL supports several cipher suites

❖ negotiation: client, server agree on cipher suite

- client offers choice
- server picks one

common SSL symmetric ciphers

- DES – Data Encryption Standard: block
- 3DES – Triple strength: block
- RC2 – Rivest Cipher 2: block
- RC4 – Rivest Cipher 4: stream

SSL Public key encryption

- RSA

Real SSL: handshake (I)

Purpose

1. server authentication
2. negotiation: agree on crypto algorithms
3. establish keys
4. client authentication (optional)

Real SSL: handshake (2)

1. client sends list of algorithms it supports, along with client nonce
2. server chooses algorithms from list; sends back: choice + certificate + server nonce
3. client verifies certificate, extracts server's public key, generates pre_master_secret, encrypts with server's public key, sends to server
4. client and server independently compute encryption and MAC keys from pre_master_secret and nonces
5. client sends a MAC of all the handshake messages
6. server sends a MAC of all the handshake messages

Real SSL: handshaking (3)

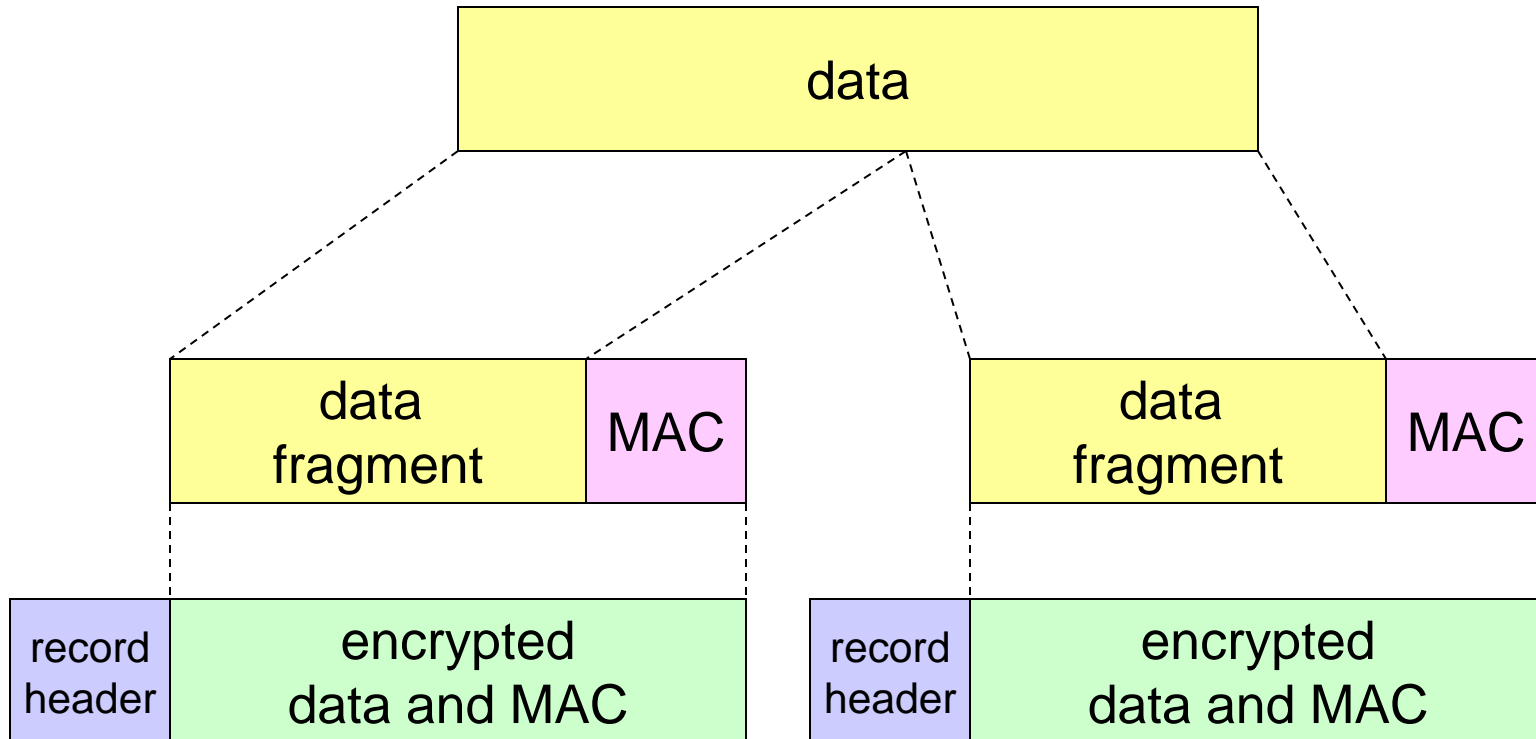
last 2 steps protect handshake from tampering

- ❖ client typically offers range of algorithms, some strong, some weak
- ❖ man-in-the middle could delete stronger algorithms from list
- ❖ last 2 steps prevent this
 - last two messages are encrypted

Real SSL: handshaking (4)

- ❖ why two random nonces?
- ❖ suppose Trudy sniffs all messages between Alice & Bob
- ❖ next day, Trudy sets up TCP connection with Bob, sends exact same sequence of records
 - Bob (Amazon) thinks Alice made two separate orders for the same thing
 - solution: Bob sends different random nonce for each connection. This causes encryption keys to be different on the two days
 - Trudy's messages will fail Bob's integrity check

SSL record protocol

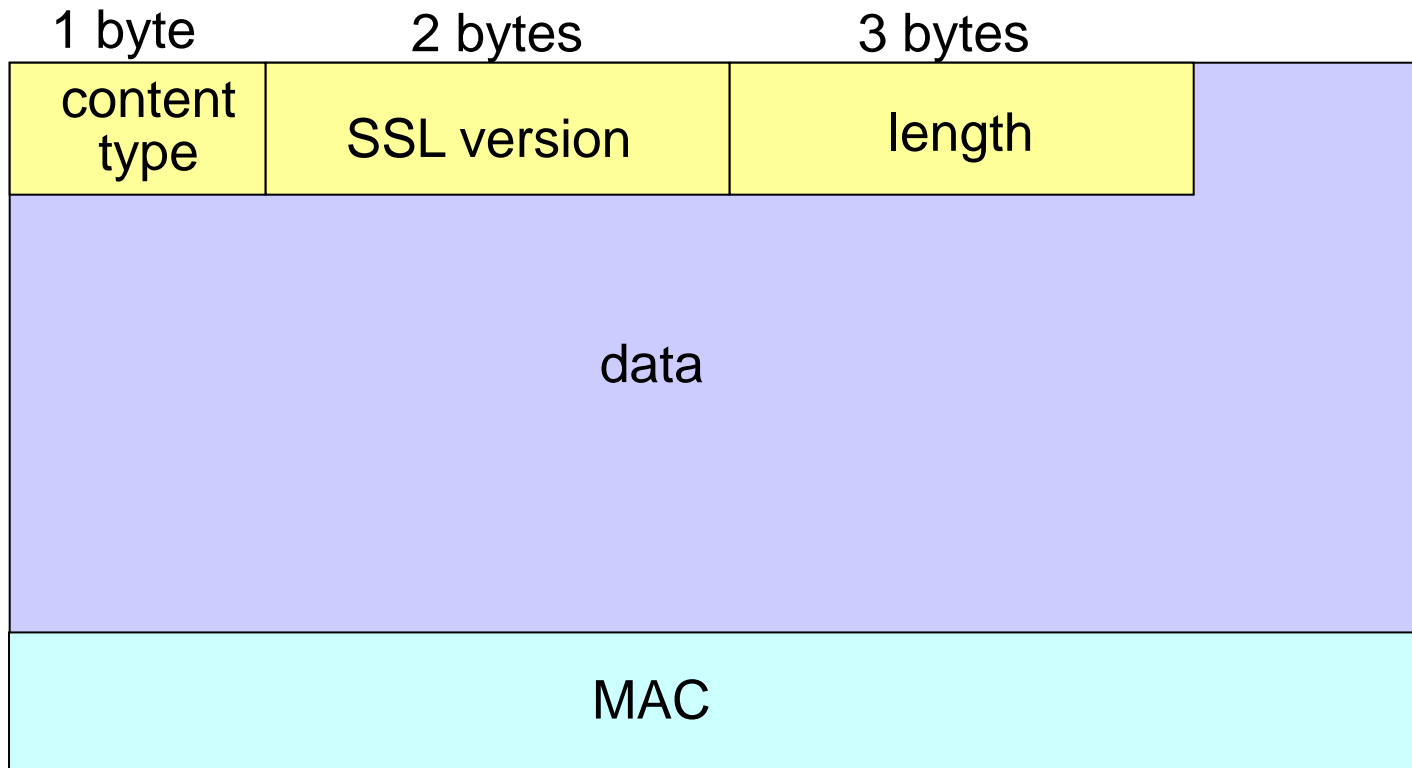


record header: content type; version; length

MAC: includes sequence number, MAC key M_x

fragment: each SSL fragment 2^{14} bytes (~16 Kbytes)

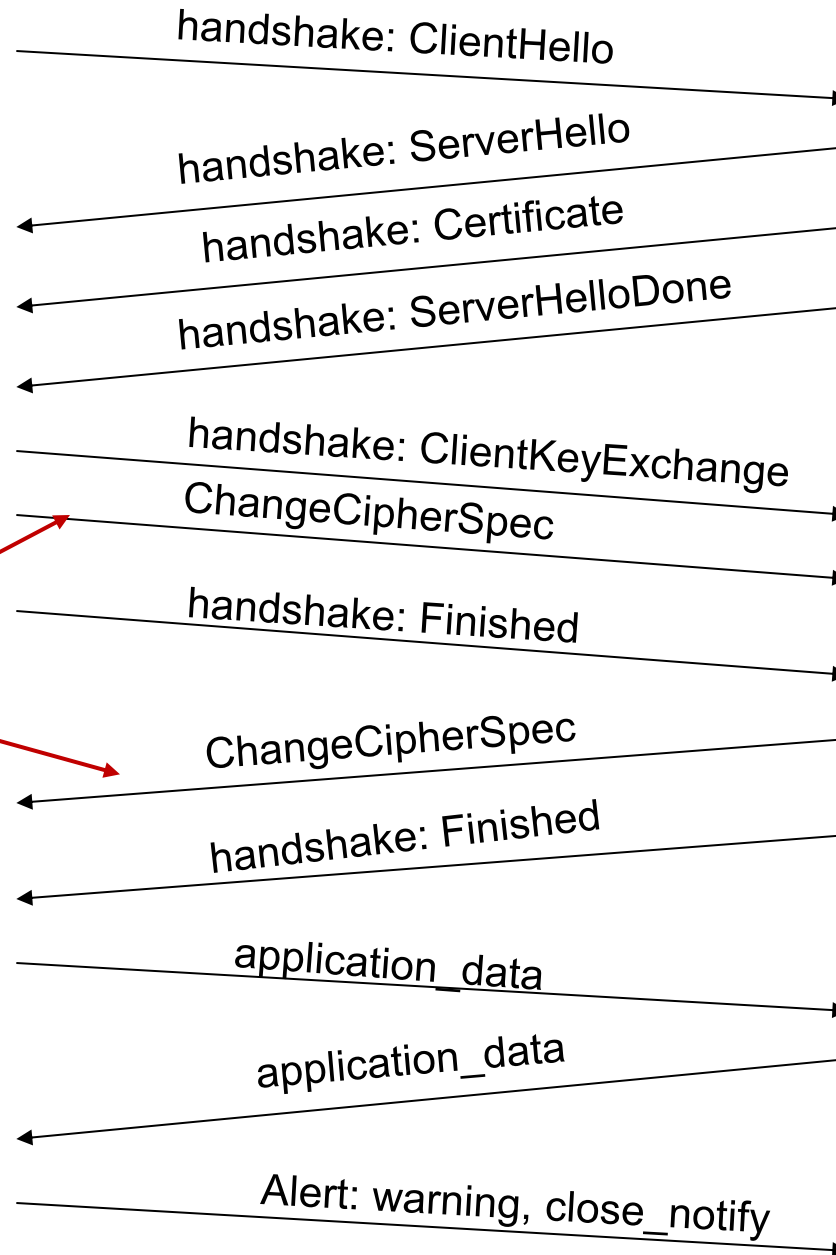
SSL record format



data and MAC encrypted (symmetric algorithm)

Real SSL connection

*everything
henceforth
is encrypted*



TCP FIN follows

Key derivation

- ❖ client nonce, server nonce, and pre-master secret input into pseudo random-number generator.
 - produces master secret
- ❖ master secret and new nonces input into another random-number generator: “key block”
 - because of resumption: TBD
- ❖ key block sliced and diced:
 - client MAC key
 - server MAC key
 - client encryption key
 - server encryption key
 - client initialization vector (IV)
 - server initialization vector (IV)