Announcements

About scores

- Likely that we need to grade on a curve
- Don't worry too much about the absolute score: Just try to study as hard as you can
- Will adjust problem difficulties for the final

This Thursday: ***No class!***

- Enjoy Thanksgiving!

Lecture 15

Transport Layer Security/ Secure Socket Layer (TLS/SSL)

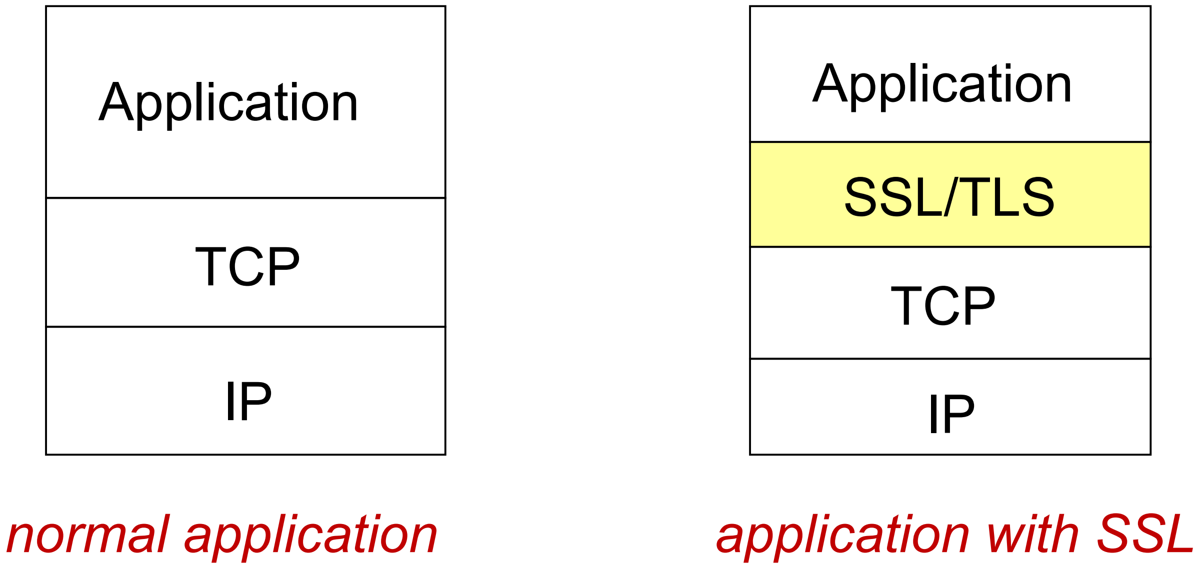
(Chapter 9 in KPS)

[lecture slides are adapted from previous slides by Prof. Gene Tsudik]

SSL: Secure Sockets Layer & TLS: Transport Layer Security

- Most widely deployed security protocol
  - supported by almost all browsers, web servers
  - the "s" in https
  - billions $/year over SSL
- Current version:
  - TLS=Transport Layer Security
  - TLS 1.3: https://tools.ietf.org/html/rfc8446
- 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/TLS and TCP/IP

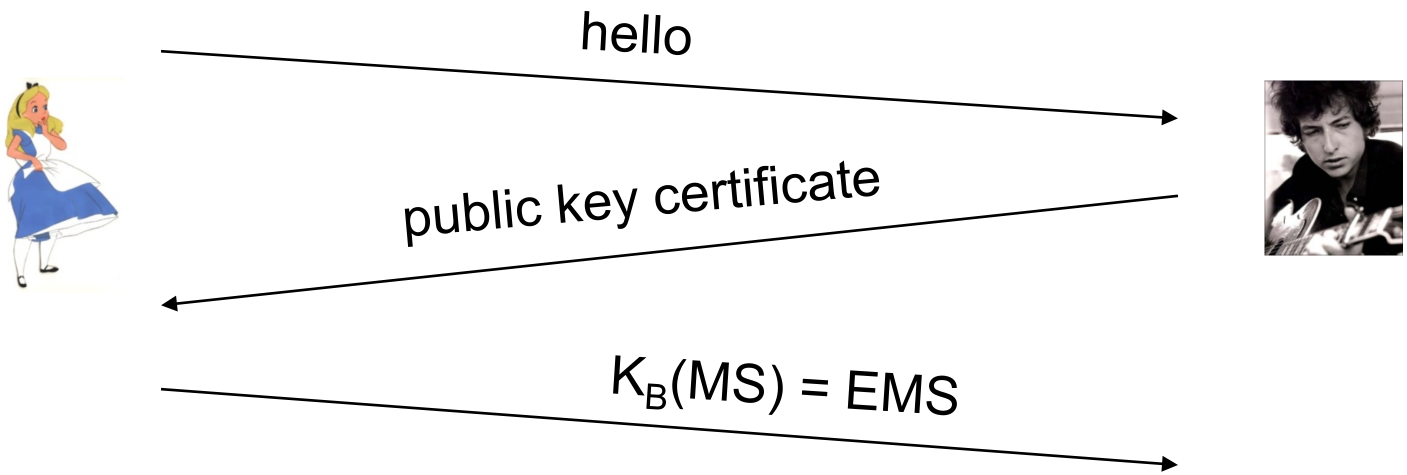


- SSL/TLS provides application programming interface (API) to applications
- C and Java SSL/TLS libraries/classes readily available

Toy SSL/TLS: a Simple Secure Channel

- *handshake:* Alice and Bob use their certificates, private keys to authenticate each other and exchange a 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 messages in a session 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

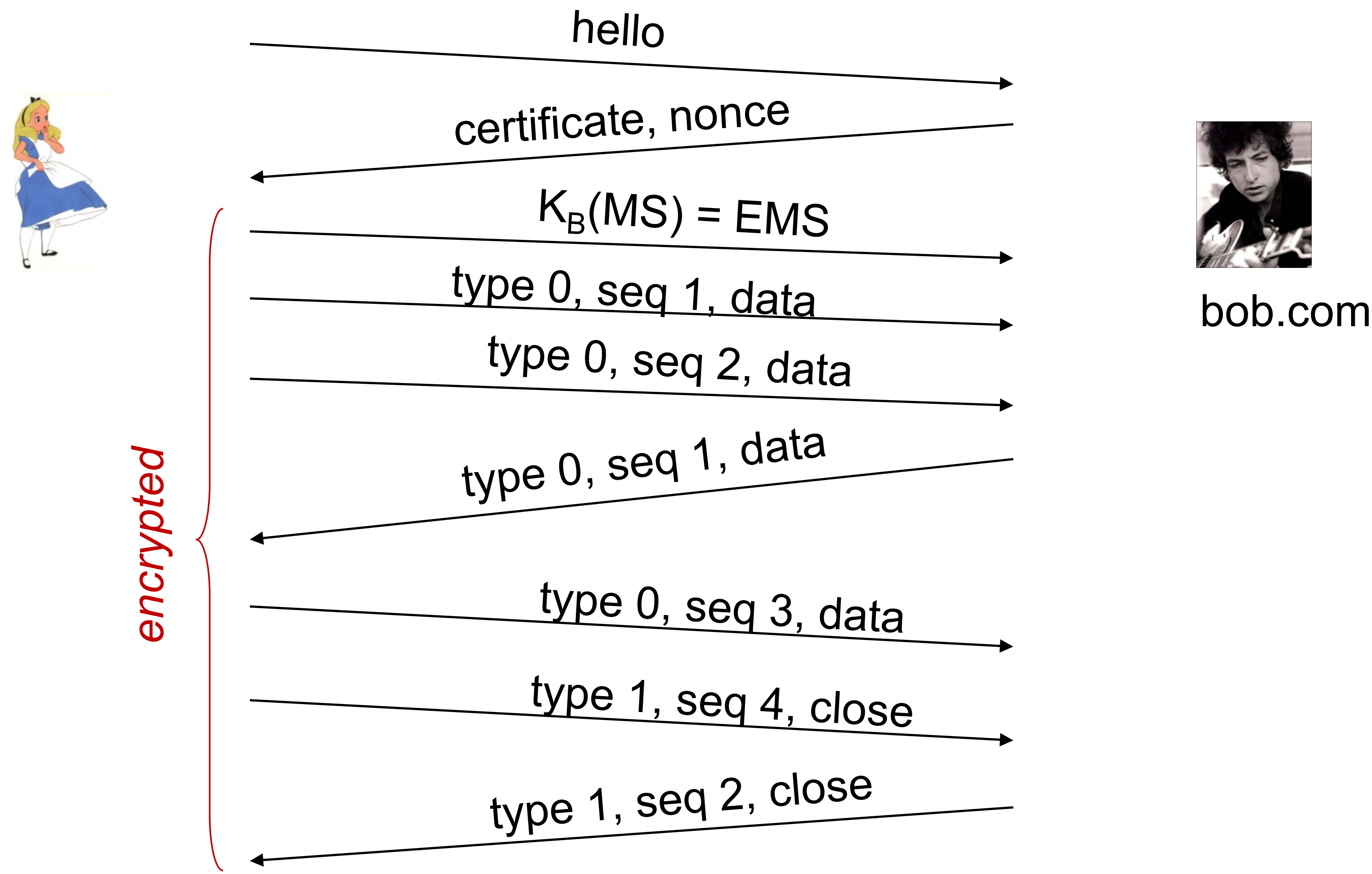
Toy: Sequence Numbers

- *problem:* attacker can capture and replay or re-order records
- *solution:* put sequence number into MAC:
  - MAC = MAC(M_x, sequence||data)
  - note: no sequence number field, M_x = MAC key

Toy: Control Information

- *problem:* truncation attack:
  - attacker forges TCP connection close segment
  - one sides thinks there is less data than there actually is
- *solution:* record types, with special type for closure
  - type 0 for data; type1for closure
- MAC = MAC(M_x, sequence||type||data)

Toy SSL/TLS: Summary



Toy SSL/TLS isn't complete

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

SSL/TLS Cipher Suite

- cipher suite
  - public-key algorithm
  - symmetric encryption algorithm
  - MAC algorithm
- SSL/TLS supports multiple cipher suites
- negotiation: client, server agree on a cipher suite
  - client offers choice
  - server picks one

Common SSL/TLS symmetric ciphers
- AES
- 3DES

SSL/TLS Public key encryption
- RSA
- DH
- EC-DH
- DSA

- MAC
  - SHA-256, SHA=128, etc.

Real SSL/TLS: Handshake (1)

Purpose

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

Real SSL/TLS: Handshake (2)

1. client sends a list of algorithms it supports, along with a client nonce
2. server chooses algorithms from list; sends back: choice + own 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 both nonces
5. client sends a MAC of all handshake messages
6. server sends a MAC of all the handshake messages

Real SSL/TLS: Handshake (3)

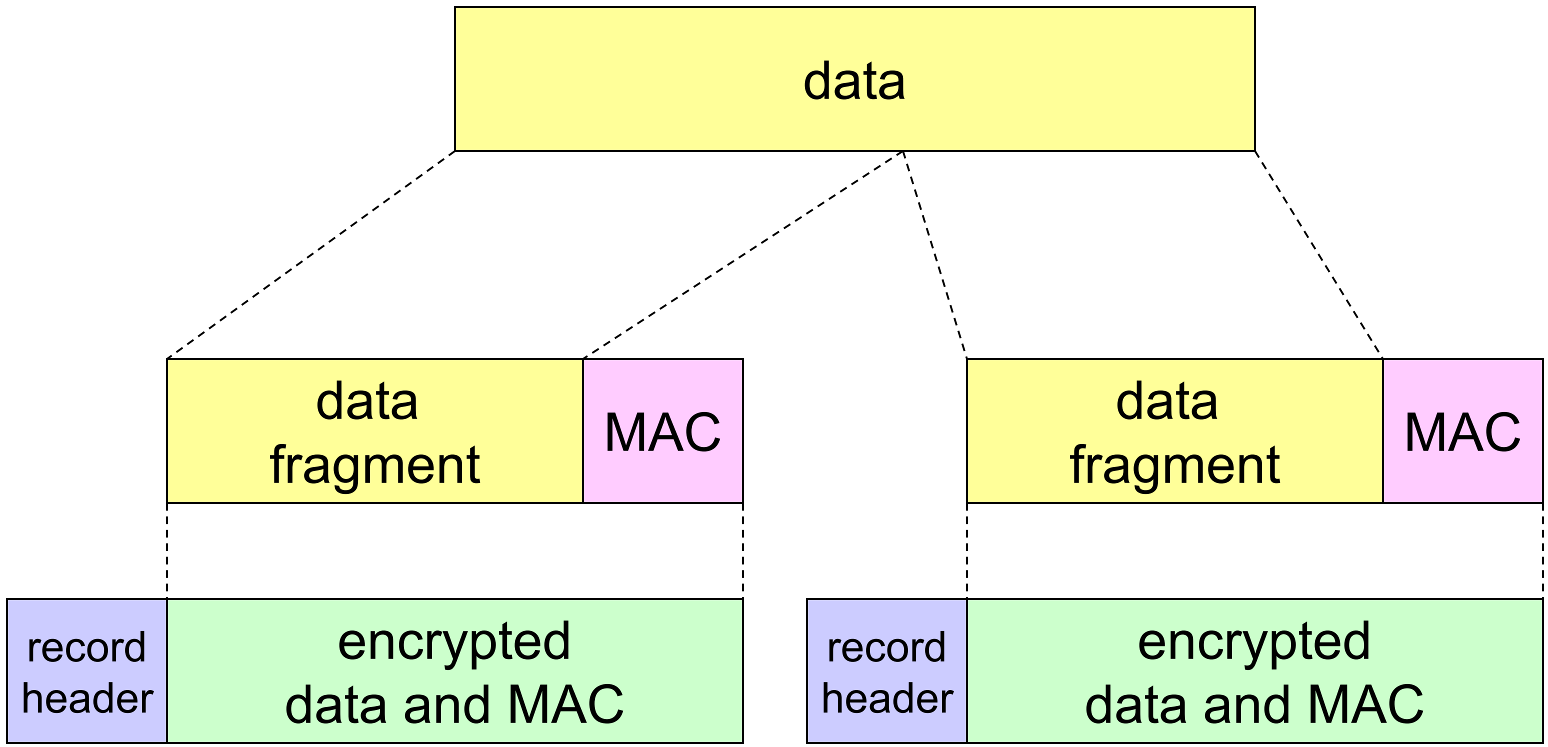
last 2 steps protect handshake from tampering

- 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/TLS: Handshake (4)

- why two random nonces?
- suppose Eve sniffs all messages between Alice & Bob
- next day, Eve 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
  - Eve's messages will fail Bob's integrity check

SSL/TLS Record Protocol

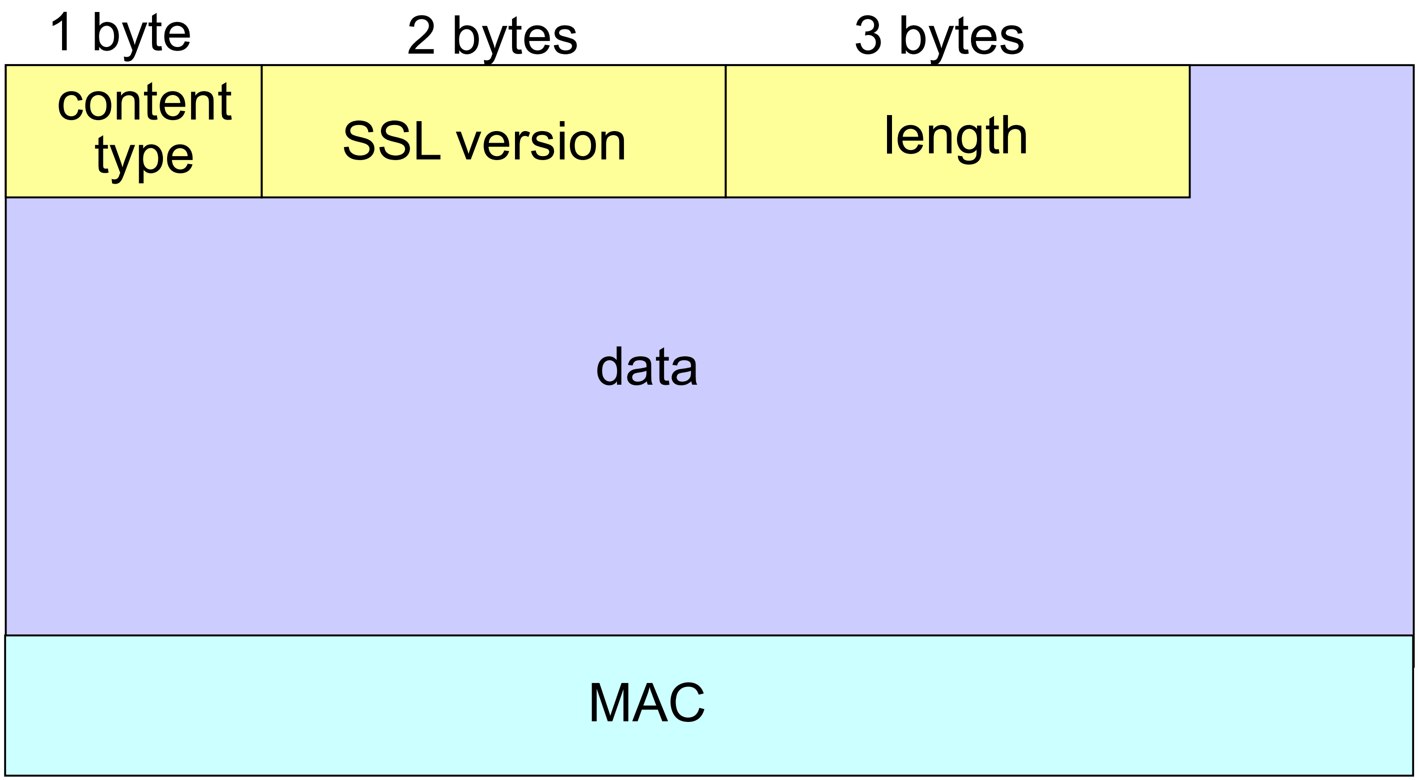


record header: content type; version; length

MAC: includes sequence number, computer with MAC key M_x

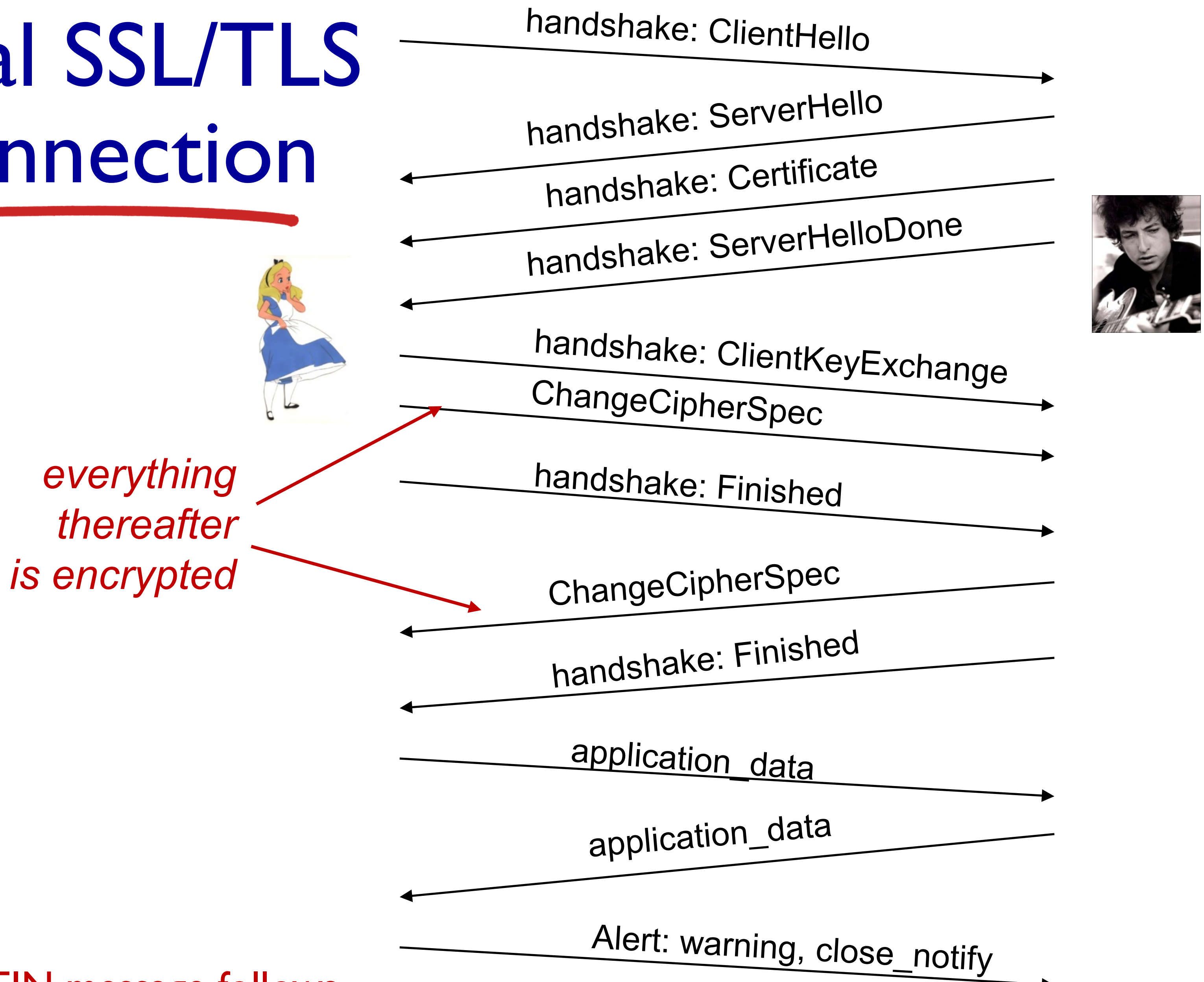
fragment: each SSL fragment 2^{14} bytes (~16 Kbytes)

SSL/TLS Record Format



data and MAC encrypted (symmetric algorithm)

Real SSL/TLS Connection



TCP FIN message follows

Key Derivation

- client nonce, server nonce, and pre-master secret input into pseudo random-number generator (PRG).
  - produces master secret
- master secret and new nonces input into another random-number generator: "key block"

- key block used to derive separate:
  - client MAC key
  - server MAC key
  - client encryption key
  - server encryption key
  - client initialization vector (IV)
  - server initialization vector (IV)