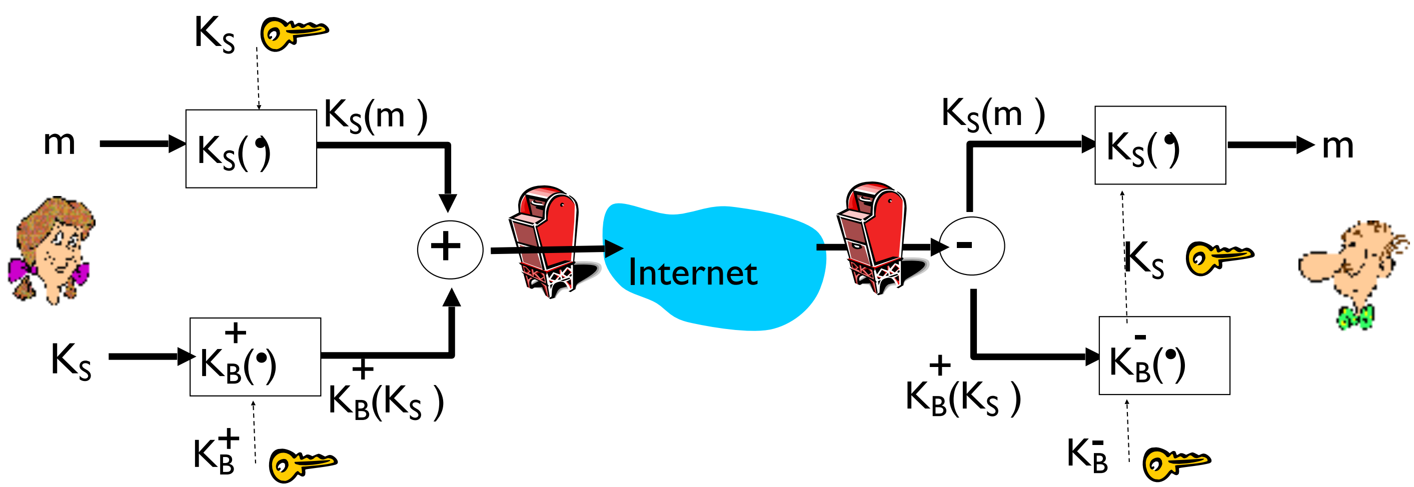
Encryptions in Different Network Layers

- Application Layer
  - Secure Email
- Transport Layer
  - TLS
- Network Layer
  - IP SEC
- Physical Layer
  - IEEE 802.11 WiFi Security
  - 4G/5G Security

Why do we need security in different layers?

Secure e-mail (Confidentiality)

- Alice wants to send confidential e-mail, m, to Bob.

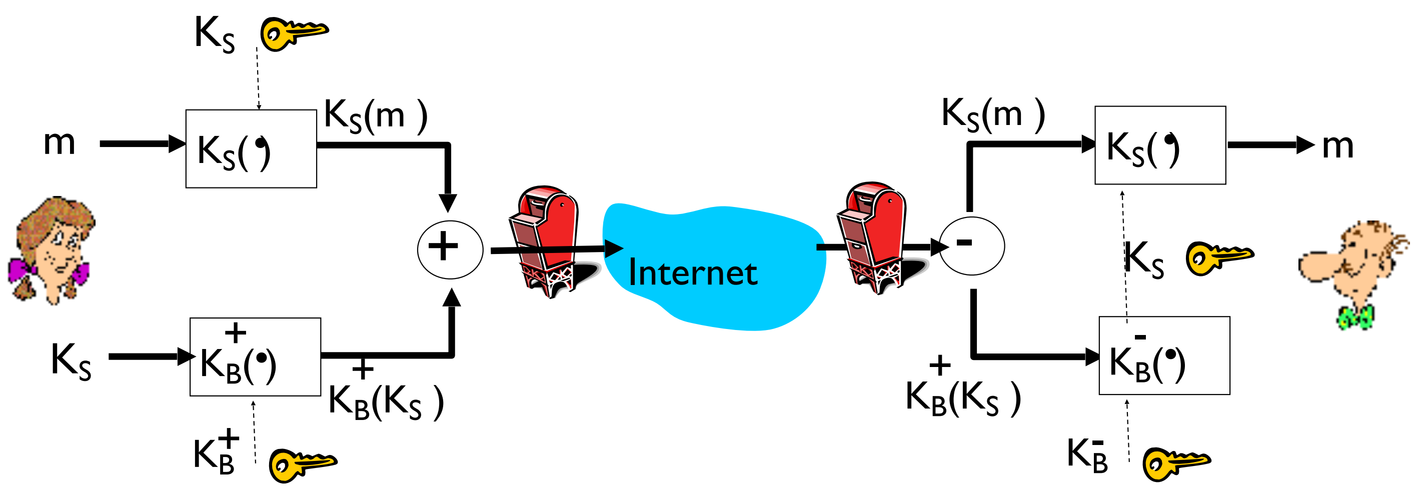


Alice:

- generates random *symmetric* private key, K_S
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key
- sends both $K_S(m)$ and $K_B(K_S)$ to Bob

Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.

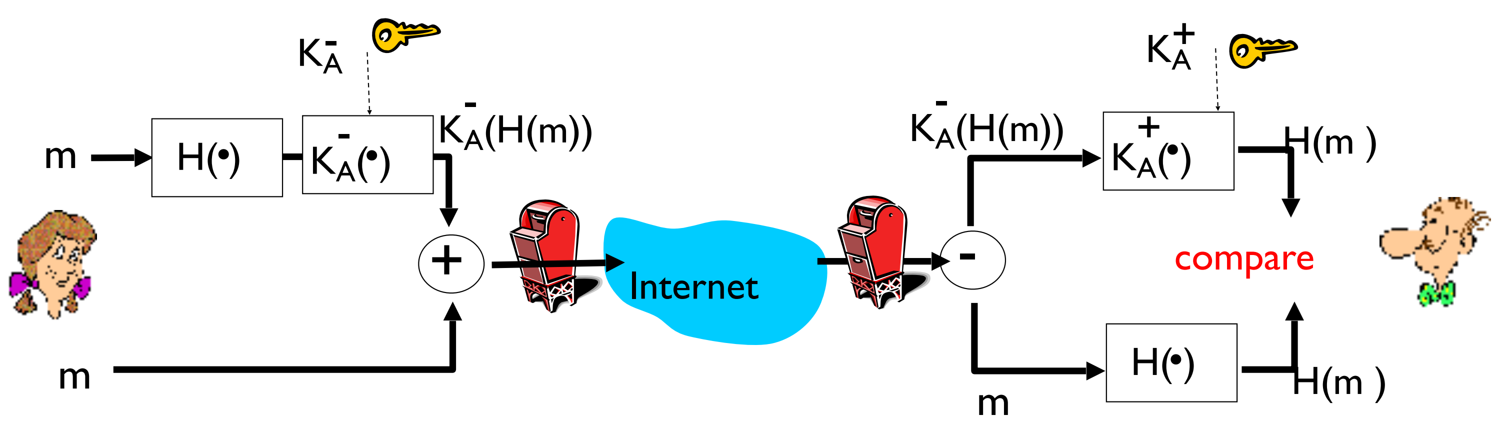


Bob:

- uses his private key to decrypt and recover K_S
- uses K_S to decrypt $K_S(m)$ to recover m

Secure e-mail (Sender Authentication)

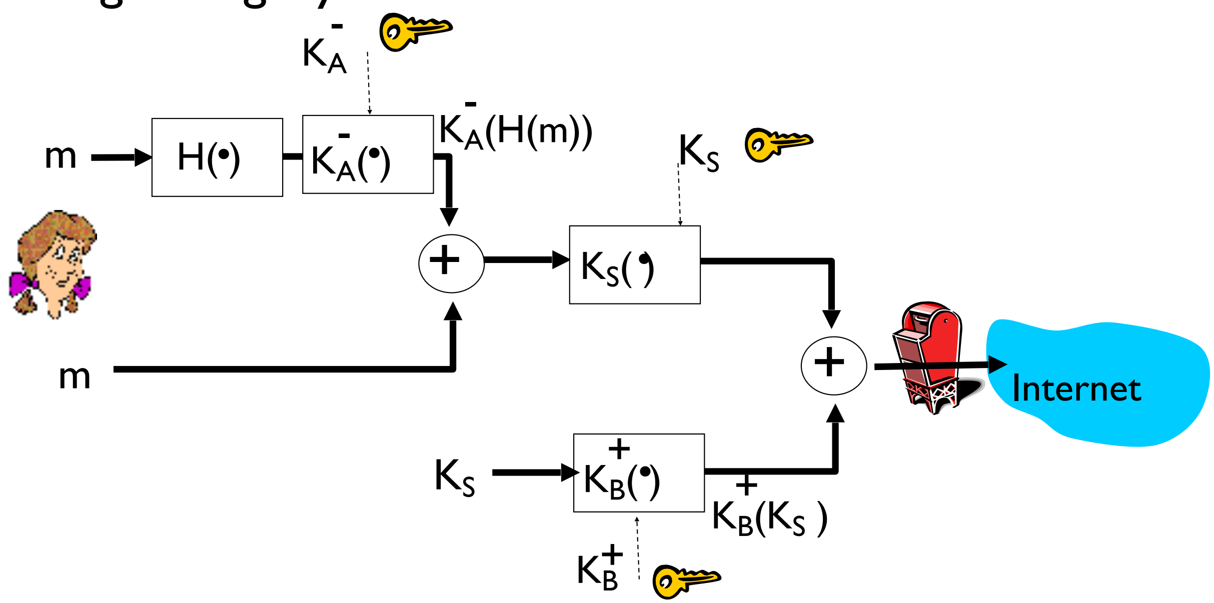
- Alice wants to provide sender authentication message integrity



- Alice digitally signs message
- sends both message (in the clear) and digital signature

Secure e-mail (Message Integrity)

- Alice wants to provide secrecy, sender authentication, message integrity.



Alice uses three keys: her private key, Bob's public key, newly created symmetric key

Phil Zimmermann

- Creator of Pretty Good Privacy (PGP) in 1991, the most widely used email encryption software in the world
- PGP uses
  - MD5 or SHA for Hash function
  - CAST, triple-DES, or IDEA for symmetric key encryption
  - RSA for public key

A PGP Signed Message

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash:  SHA1
Bob:
Can I see you tonight?
Passionately yours, Alice
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset:  noconv
yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----
```

A Secret PGP Message


```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=lKhnMikLo0+1/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp
-----END PGP MESSAGE
```

Encryptions in Different Network Layers

- Application Layer
  - Secure Email
- Transport Layer
  - TLS
- Network Layer
  - IP SEC
- Physical Layer
  - IEEE 802.11 WiFi Security
  - 4G/5G Security

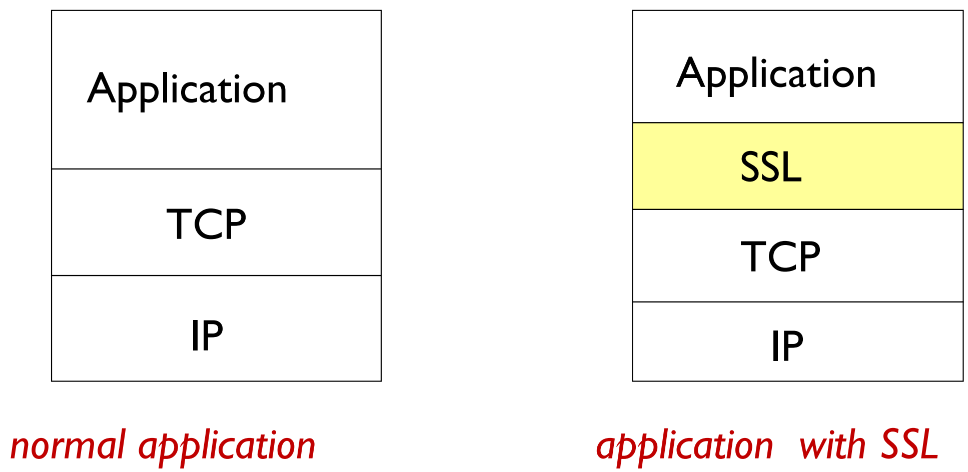
Why do we need security in different layers?

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

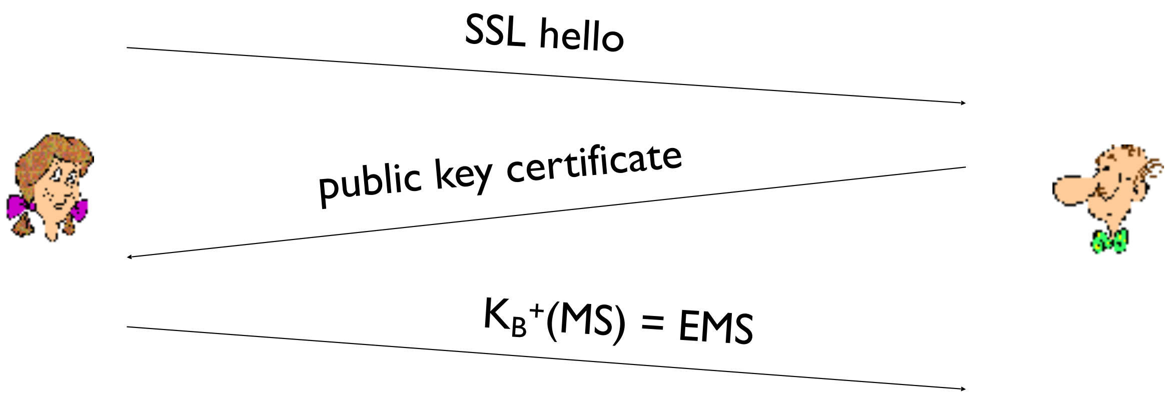


- SSL provides application programming interface (API) to applications
- C and Java SSL libraries/classes readily available

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:
  - $\mathbf{K_c}$ **=** encryption key for data sent from client to server
  - $\mathbf{M_c}$ **=** MAC key for data sent from client to server
  - $\mathbf{K_s}$ **=** encryption key for data sent from server to client
  - $\mathbf{M_s}$ **=** 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

length	data	MAC

Toy: Sequence Numbers

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

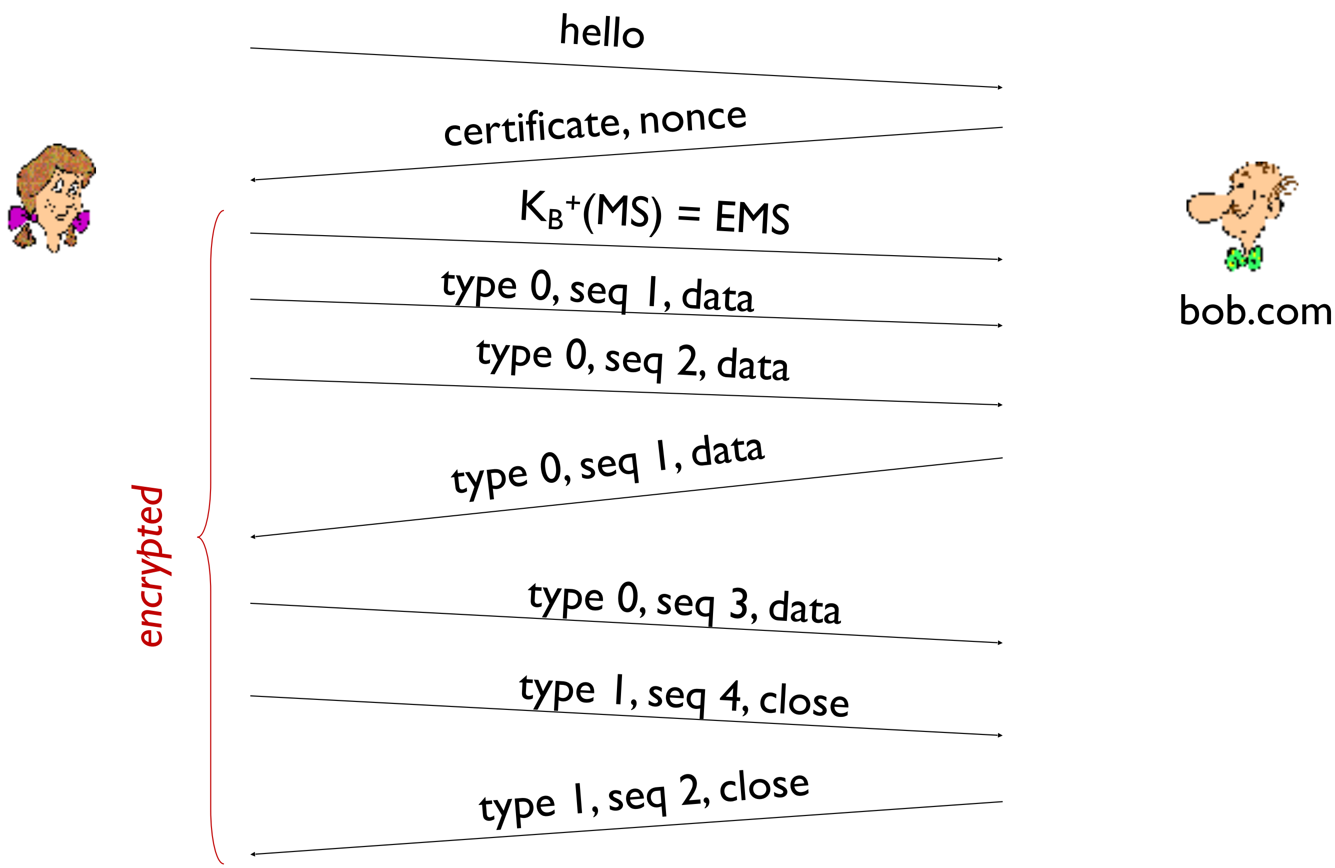
- *Problem:* attacker could replay all records
- *Solution:* use nonce

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, sequence||type||data)

length	type	data	MAC

Toy SSL: summary



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)

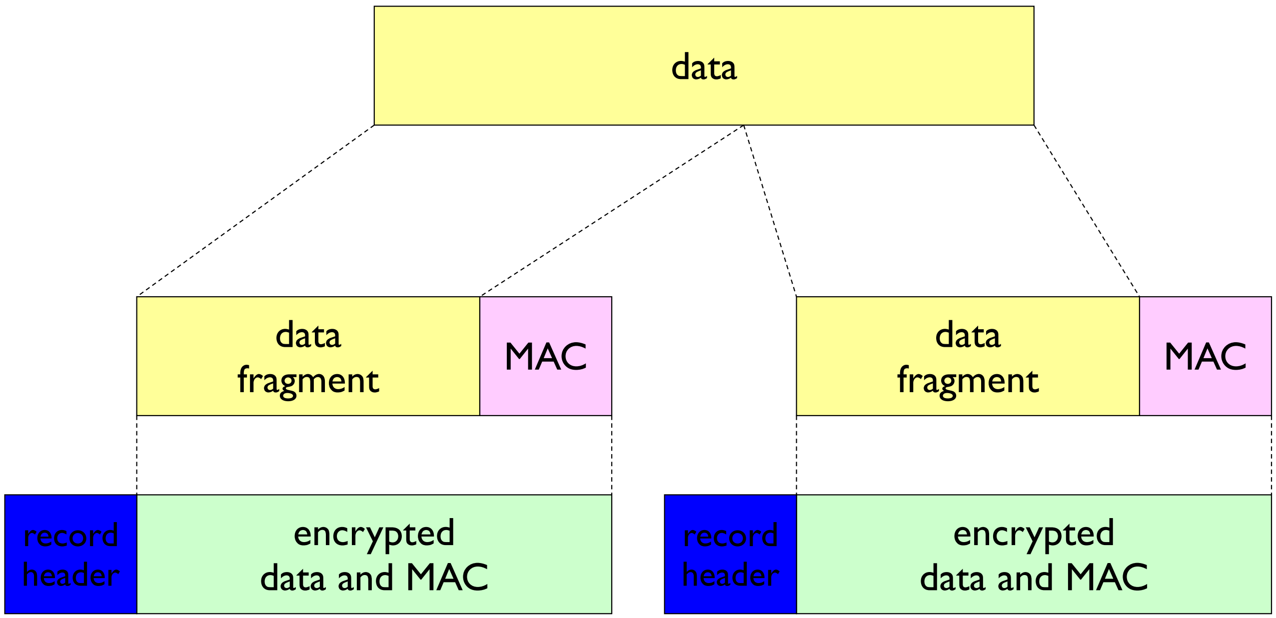
Last 2 steps protect handshake from tampering

- 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

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

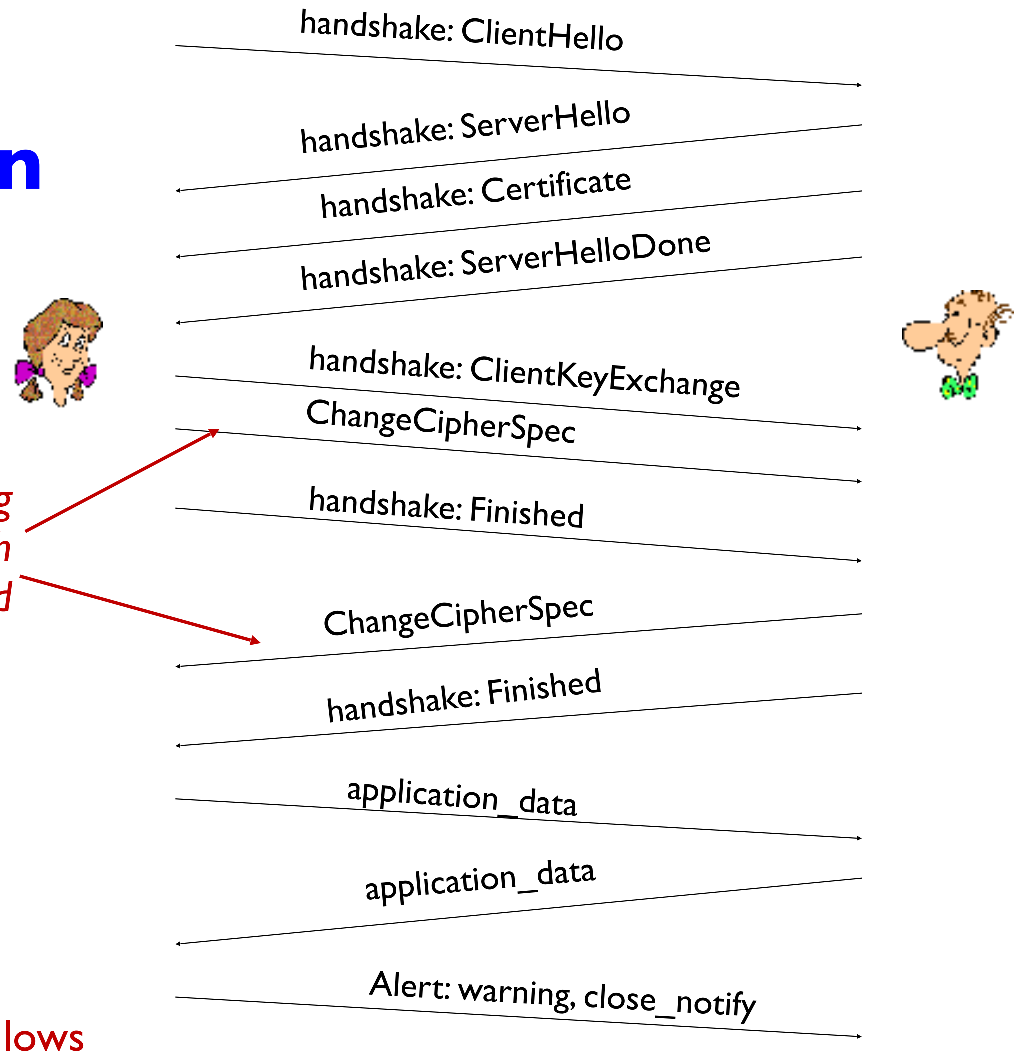
SSL Record Protocol



record header: content type; version; length

MAC: includes sequence number, MAC key M_x

Real SSL connection



everything henceforth is encrypted

TCP FIN follows