# CSCI 7000-001 — Communication Security for Applications

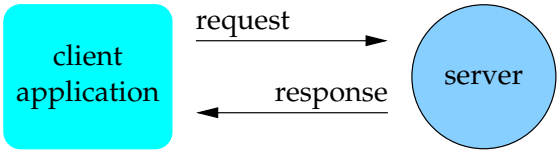
October 9, 2001

## 1 Outline

- intro to distributed computing: client-server computing
- transport-layer security
- secure socket layer

## 2 Client-server computing

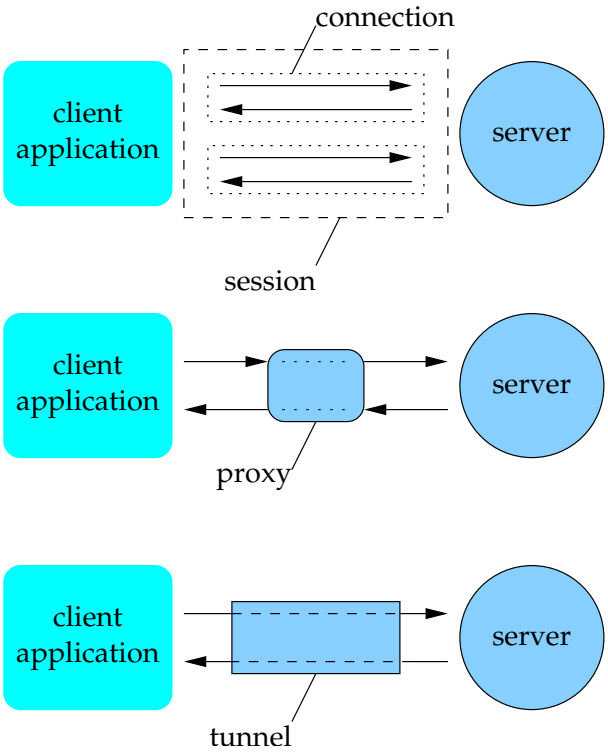
Simple client-server architecture:




Examples:

- X11 (remote display)
- databases
- web
- ...

# 3 Anatomy of the client-server architecture




**connection** request/response

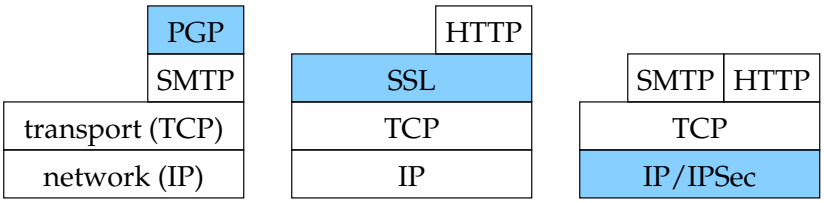
**session** sequence of connections

**proxy** active relay. May modify parts of the protocol, redirect requests to different servers or even serve them locally (e.g., a proxy cache)

**tunnel** passive relay

# 4 Security across layers

We can secure different communication layers




Examples:

**PGP** application-level security

**SSL** transport-level security

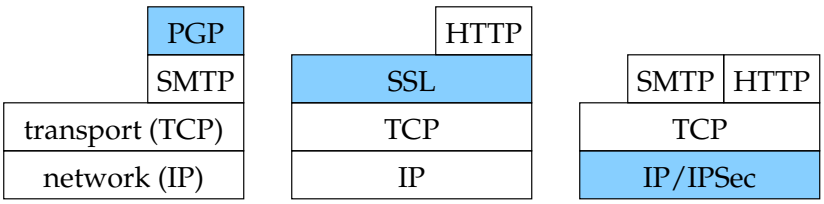
**IPSec** network-level security

# 5 SSL

- originally developed by Netscape Corp.
- version 3 designed with public review/input, published as an Internet draft
- IETF working group: transport layer security (TSL)

# 6 SSL architecture

SSL is made of two layers of protocols:




# 7 SSL Record Protocol

- provides basic security services to higher-level protocols
- *connections*:

  - transient
  - peer-to-peer relationships
  - every connection is associated with a session

- *session*:
  - association between a client and a server
  - may span multiple connections
  - created by the Handshake Protocol
  - define a set of cryptographic security parameters

# 8 Session parameters

**session identifier** an arbitrary sequence chosen by the server

**peer certificate** an X509.v3 certificate of the peer (may be null)

**compression method** algorithm used to compress data before encryption

**cipher spec** bulk encryption algorithm (e.g., DES, IDEA, etc., or null)

**master secret** 48-byte secret shared between client and server

**is resumable** flag indicating whether this session can be used to initiate new connections

# 9 Connection parameters

**server and client random** byte sequences chosen by server and client

**server write MAC secret** secret key used to verify MACs for data sent by the server

**client write MAC secret** secret key used to verify MACs for data sent by the client

**server write key** key for conventional (bulk) encryption of data sent by the server

**client write key** key for conventional (bulk) encryption of data sent by the client

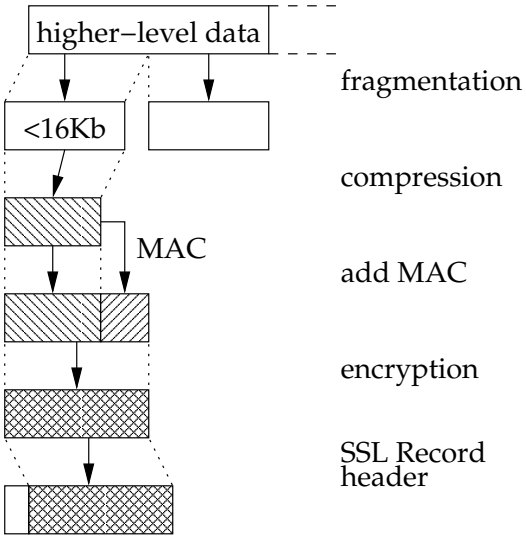
**initialization vector** init vector for CBC data encryption mode. Initialized by the Handshake Protocol

**sequence number** 64 bit record (block) identifier

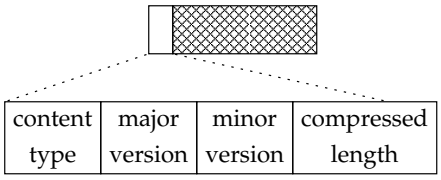
# 10 SSL Record Protocol

SSL Record Protocol provides:

- *confidentiality*: shared key used for conventional encryption
- *integrity*: shared key used for MACs



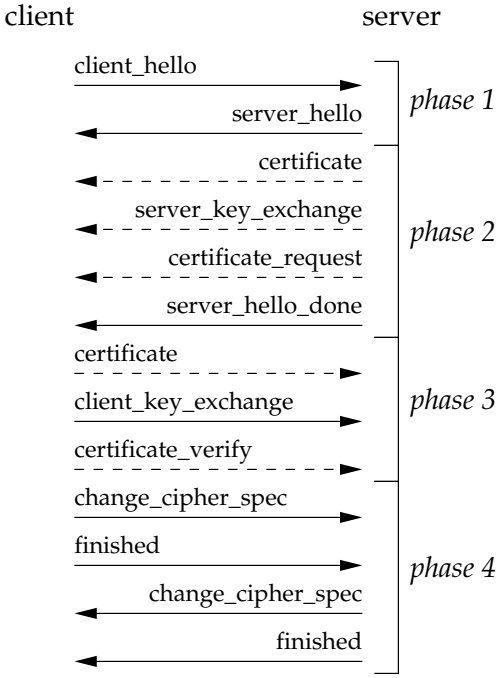

# 11 SSL Record Header



# 12 SSL Handshake Protocol

- run before any application data is transmitted
- provides mutual authentication
- establishes secret encryption keys
- establishes secret MAC keys

# 13 SSL Handshake Protocol




# 14 SSL Handshake Protocol: Phase 1

*Phase 1* initiates protocol and establishes security capabilities. The client sends a *client_hello message* with the following parameters:

- *version*: highest SSL version understood by client

- *random* 32-bit timestamp and 28 bytes generated by a secure random number generator (these values are nonces used during key exchange)
- *session id* variable-length identifier: $0$ means that clients wants to initiate a new session; $\neq 0$ means that client wants to update parameters of existing connection
- *cipher suite* list of combinations of crypto algorithms supported by client, in decreasing order of preference:
  - key exchange methods (e.g., RSA, fixed Diffie-Hellman, anonymous Diffie-Hellman, etc.)
  - bulk encryption and MAC specification, including specifications for cipher algorithm (3DES, IDEA, etc.), MAC algorithm (MD5 or SHA1), cipher type, is-exportable, hash size (bytes: 0, 16 for MD5, or 20 for SHA1), init vector.
- *compression methods* list of compression algorithms supported by client

Server replies with a *server_hello message* with the same parameters.

# 15 SSL Handshake Protocol: Phase 2

*Phase 2* performs server authentication and key exchange:

- server sends a *certificate message* containing a chain of X.509 certificates (optional, not necessary for anonymous Diffie-Hellman)
- server sends a *server_key_exchange message*. Not necessary for fixed Diffie-Hellman or RSA key exchange.
- a non anonymous server sends an *certificate_request message*, specifying the signature schema as well as a list of accepted certification authorities
- server sends a *server_done message* to close up this phase

# 16 SSL Handshake Protocol: Phase 3

*Phase 3* performs client authentication and key exchange:

- client sends a *certificate message*, if server asked
- client sends a *client_key_exchange message*.
- client sends *certificate_verify message*

# 17 SSL Handshake Protocol: Phase 4

*Phase 3* wraps up.

- client sends a *change_cipher_spec message*
- client sends a *finished message* to verify that the key-exchange and authentication process was successful
- server sends a *change_cipher_spec message*
- server sends a *finished message*

*finished message* are essentially hashes of the secret keys and of the whole handshake protocol (excluding this message, of course).

# 18 SSL in practice: setup

The first step is to set up keys and certificates:

- creating your own certification authority
- creating a key pair and a certification request
- creating certificate: signing