

# SSL

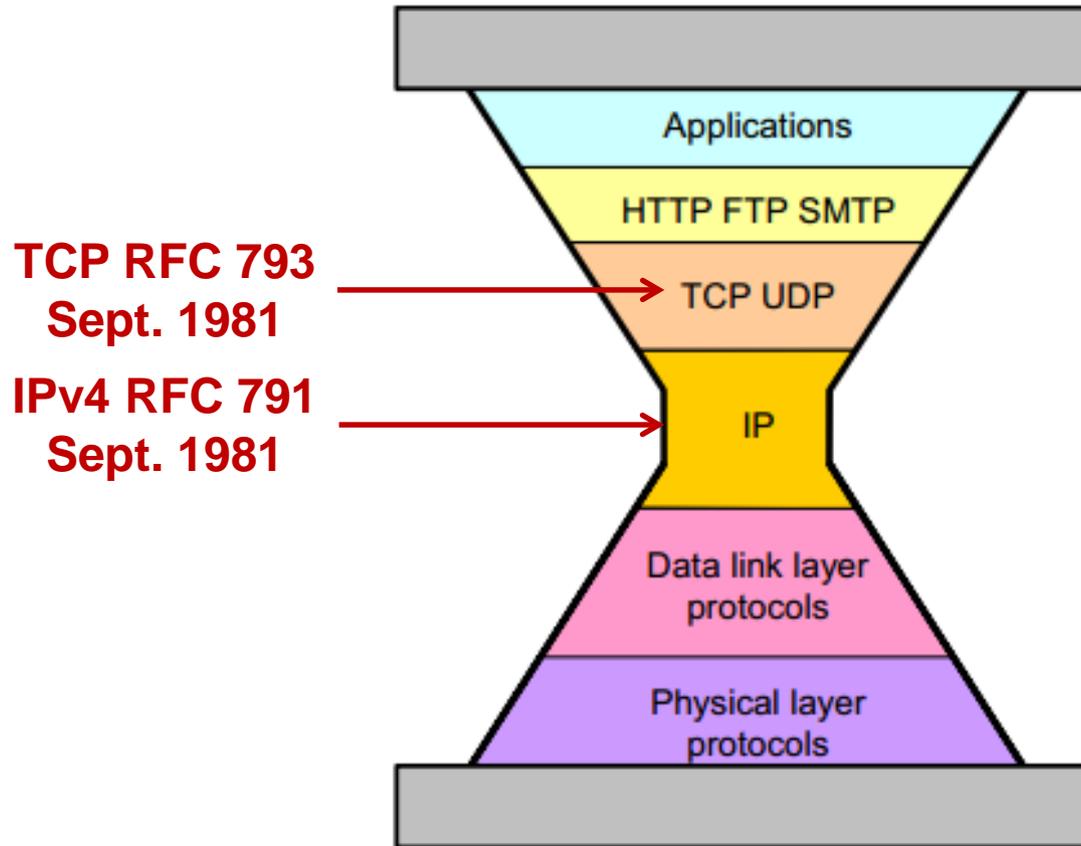
## Secure Sockets Layer

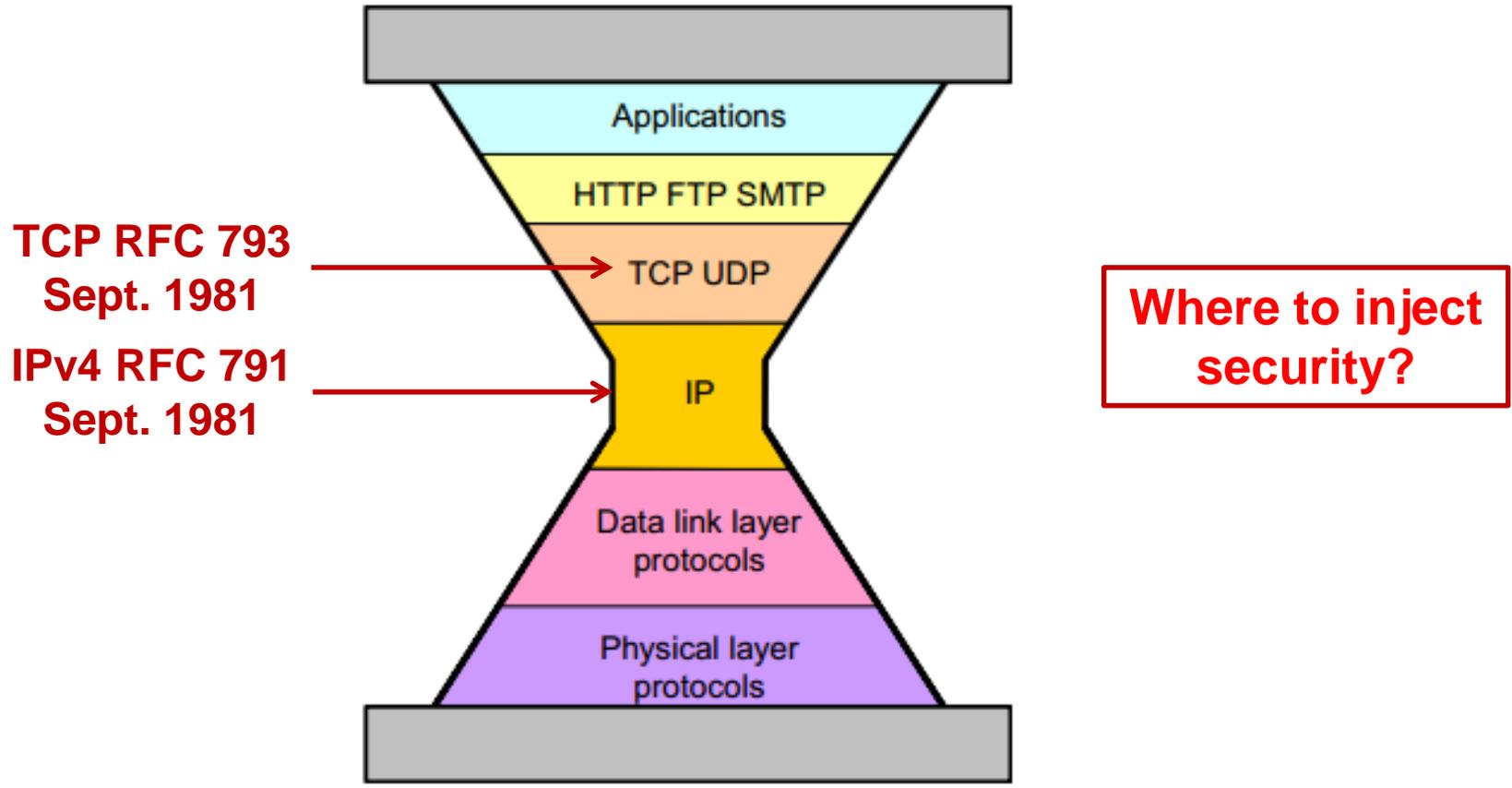
Prof. Ravi Sandhu  
Executive Director and Endowed Chair

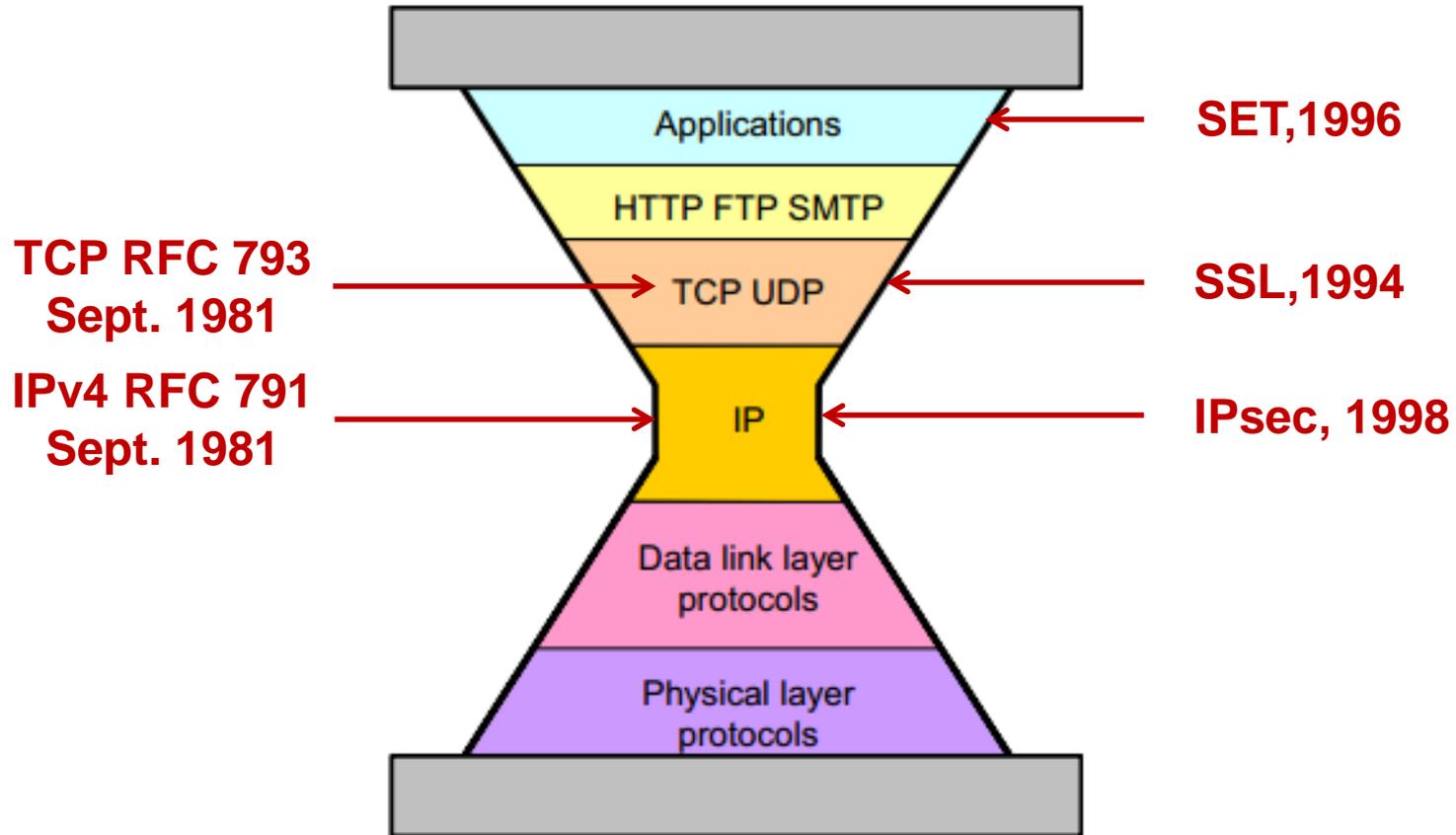
Lecture 10

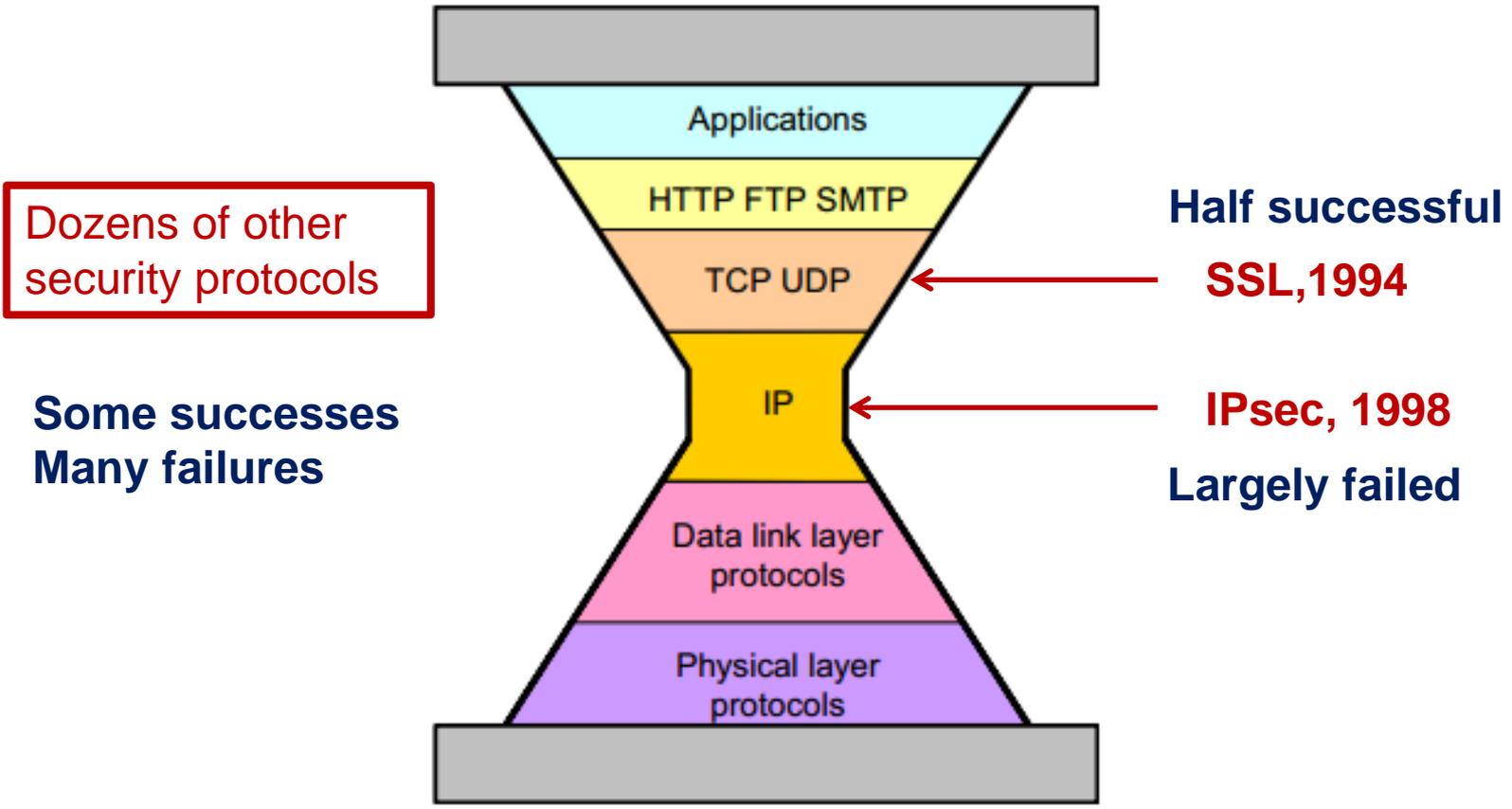
ravi.utsa@gmail.com  
www.profsandhu.com

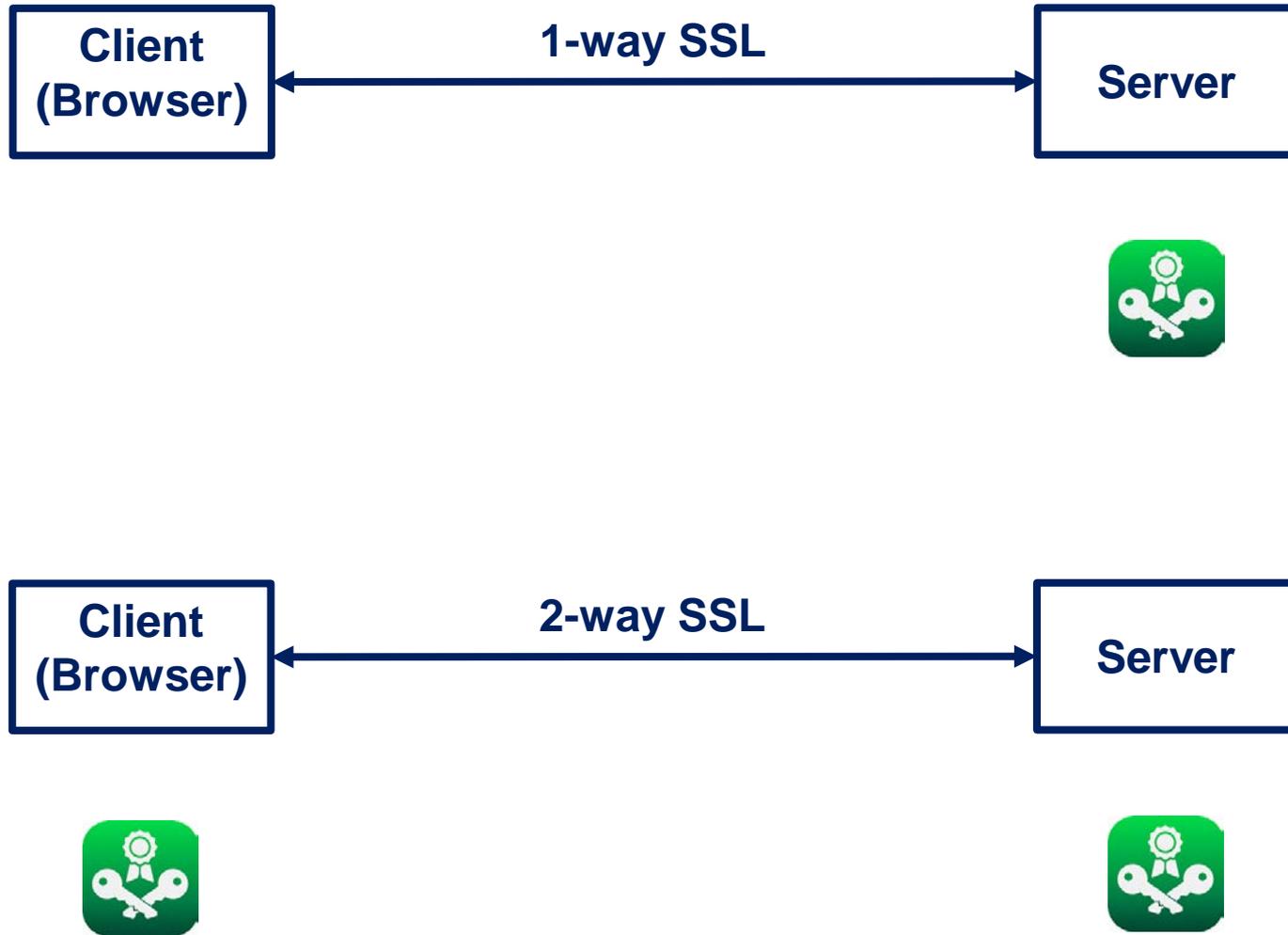
# Internet Security Protocols

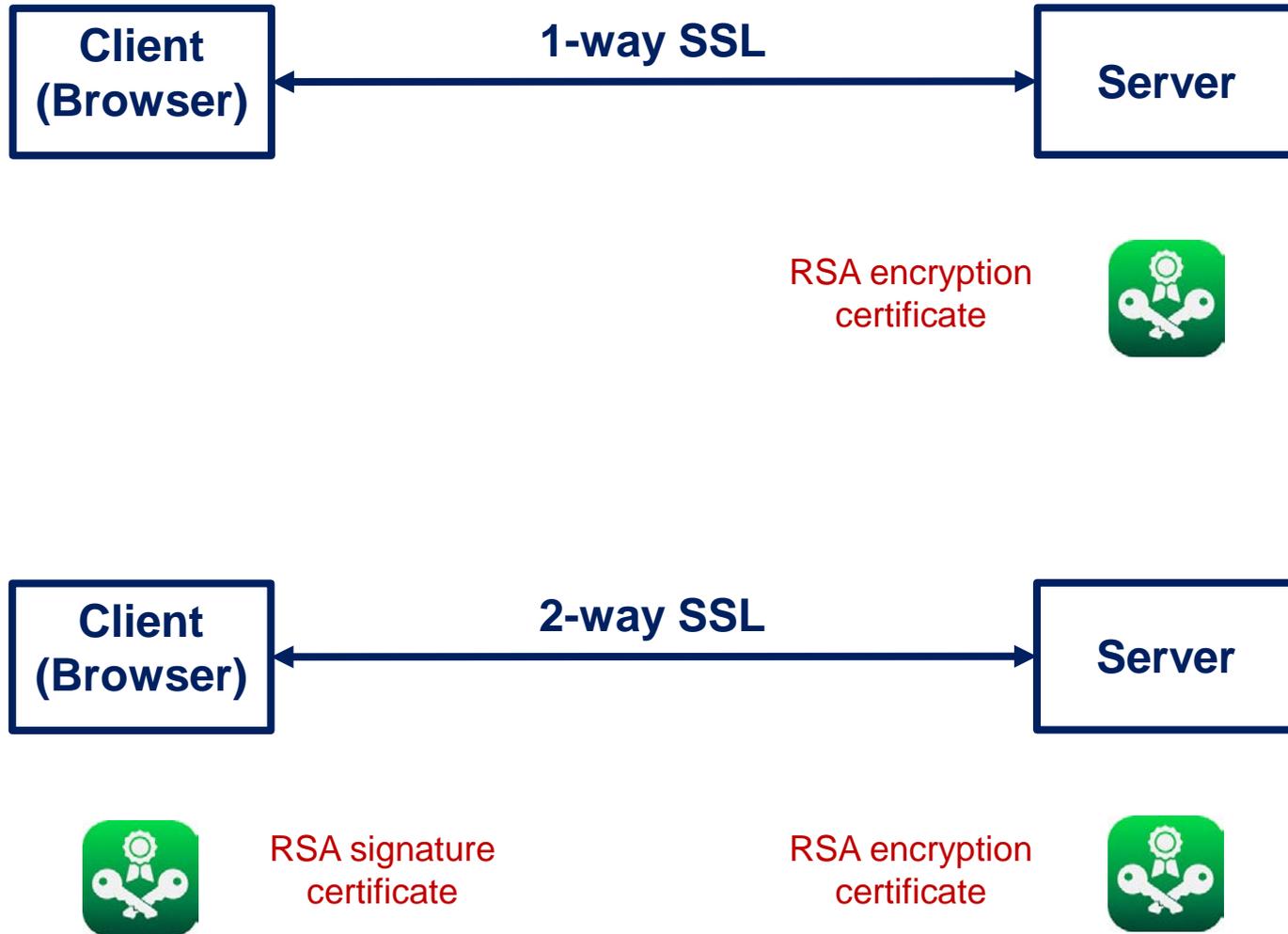


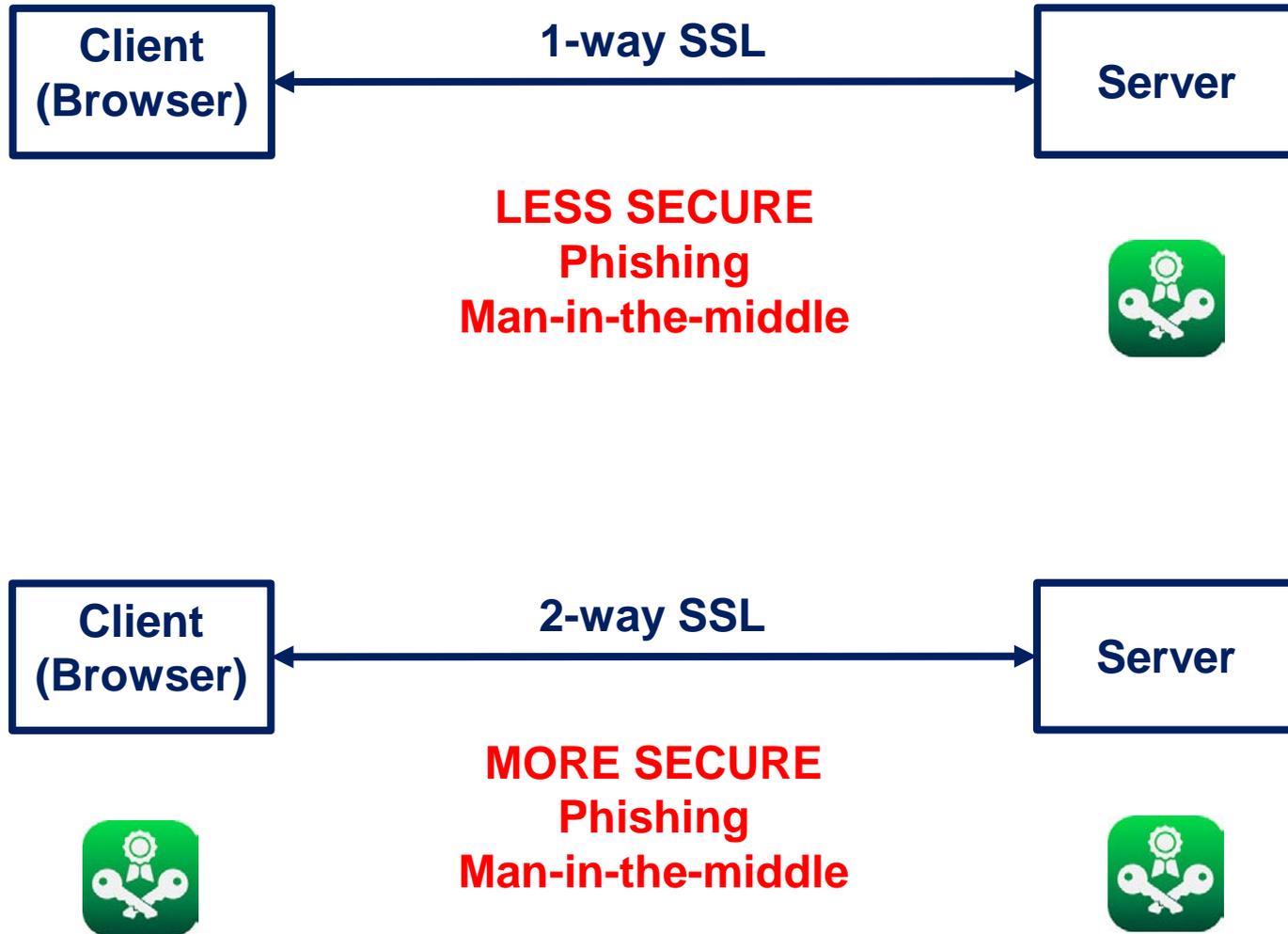


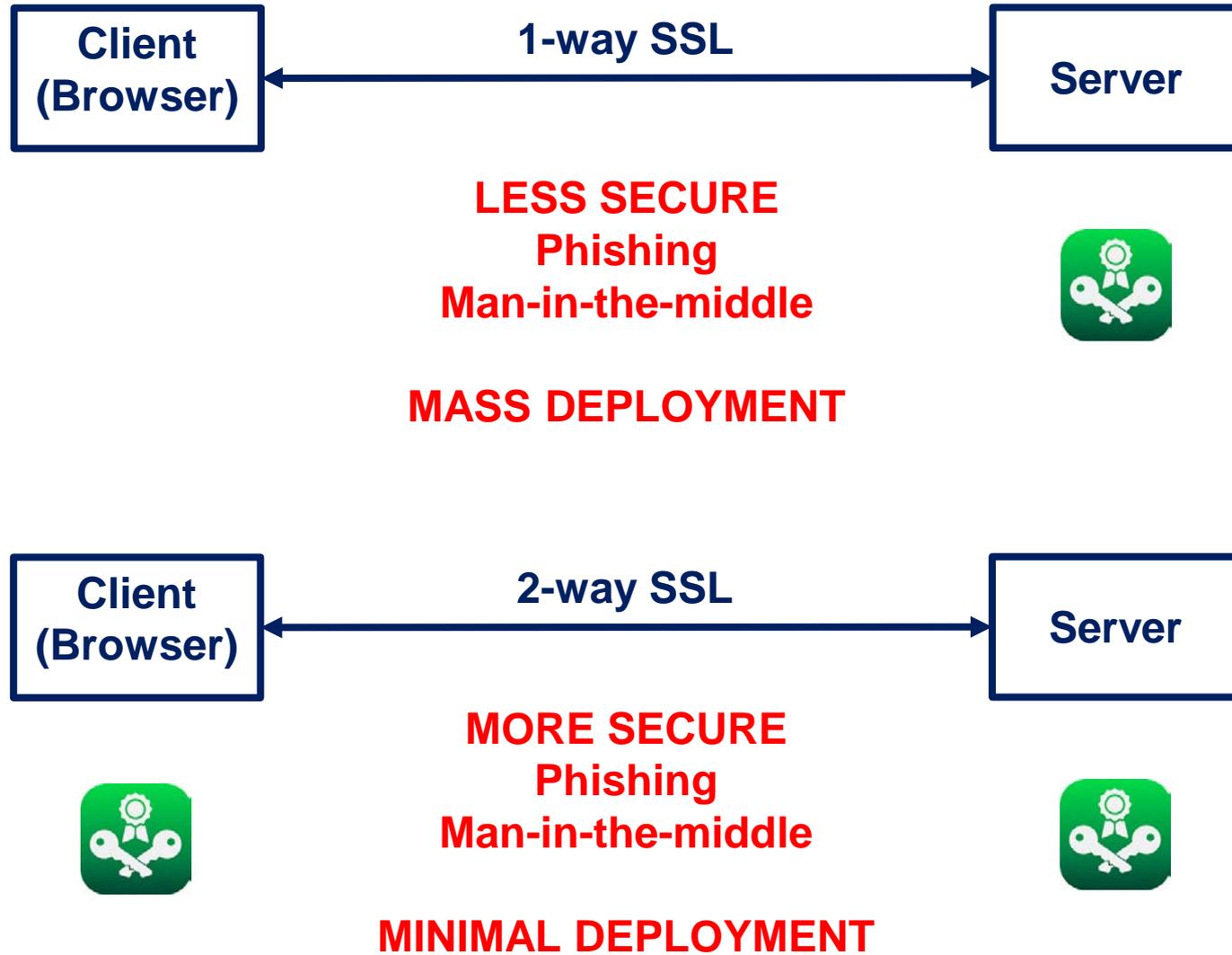












- Client-less trumps client-full
- Start-ups (SSL) trump committees (IPSEC)

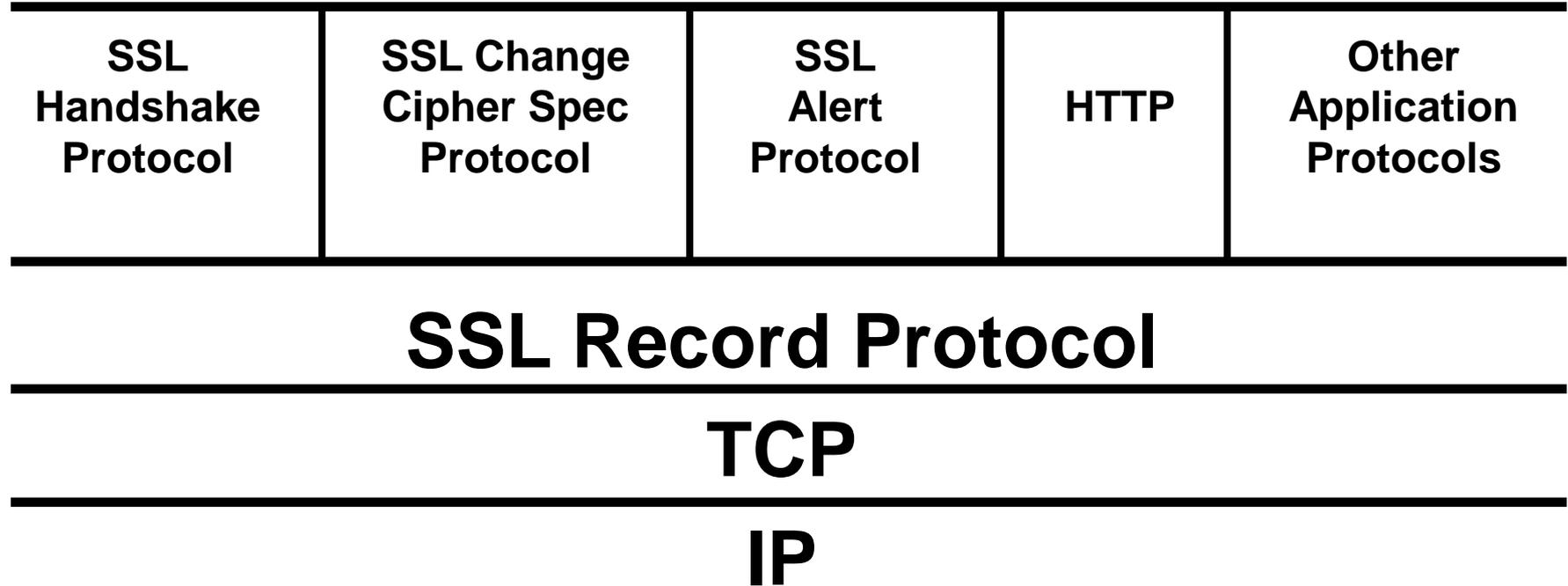
# SSL Details

- layered on top of TCP
- SSL versions 1.0, 2.0, 3.0, 3.1
- Netscape protocol
- later refitted as IETF standard TLS (Transport Layer Security)
- TLS 1.0 very close to SSL 3.1

- application protocol independent
- does not specify how application protocols add security with SSL
  - ❖ how to initiate SSL handshaking
  - ❖ how to interpret certificates
- left to designers of upper layer protocols to figure out

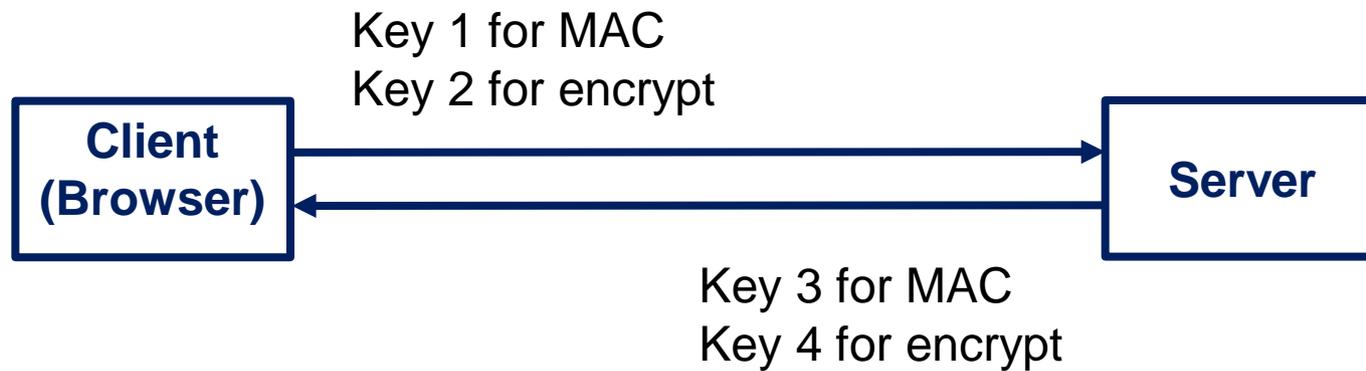
- https 443
- smtp 465
- snntp 563
- sldap 636
- spop3 995
- ftp-data 889
- ftps 990
- imaps 991
- telnets 992
- ircs 993

- peer entity authentication
- data confidentiality
- data authentication and integrity
- compression/decompression
- generation/distribution of session keys
  - ❖ integrated into protocol
- security parameter negotiation



- Handshake protocol: complicated
  - ❖ embodies key exchange & authentication
  - ❖ runs in plaintext
  - ❖ 10 message types
- Change Cipher Spec protocol: straightforward
  - ❖ single 1 byte message with value 1
  - ❖ could be considered part of handshake protocol
  - ❖ transitions from plaintext to encrypted and mac'ed
- Record protocol: straightforward
  - ❖ fragment, compress, MAC, encrypt
  - ❖ uses 4 symmetric keys
- Alert protocol: straightforward
  - ❖ 2 byte messages
  - ❖ 1 byte alert level- fatal or warning; 1 byte alert code

➤ 4 symmetric keys



- 4 steps by sender (reversed by receiver)
  - ❖ Fragmentation
  - ❖ Compression
  - ❖ MAC
  - ❖ Encryption

- each SSL record contains
  - ❖ content type: 8 bits, only 4 defined
    - change\_cipher\_spec
    - alert
    - handshake
    - application\_data
  - ❖ protocol version number: 8 bits major, 8 bits minor
  - ❖ length: max 16K bytes (actually  $2^{14}+2048$ )
  - ❖ data payload: optionally compressed and encrypted
  - ❖ message authentication code (MAC)

- initially SSL session has null compression and cipher algorithms
- both are set by the handshake protocol at beginning of session
- handshake protocol may be repeated during the session

- SSL session negotiated by handshake protocol
  - ❖ session ID
    - chosen by server
  - ❖ X.509 public-key certificate of peer
    - possibly null
  - ❖ compression algorithm
  - ❖ cipher spec
    - encryption algorithm
    - message digest algorithm
  - ❖ master secret
    - 48 byte shared secret
  - ❖ is resumable flag
    - can be used to initiate new connections
    - each session is created with one connection, but additional connections within the session can be further created

- connection end: client or server
- client and server random: 32 bytes each
- keys generated from master secret, client/server random
  - ❖ client\_write\_MAC\_secret    server\_write\_MAC\_secret
  - ❖ client\_write\_key            server\_write\_key
  - ❖ client\_write\_IV             server\_write\_IV
- compression state
- cipher state: initially IV, subsequently next feedback block
- sequence number: starts at 0, max  $2^{64}-1$

- 4 parts to state
  - ❖ current read state
  - ❖ current write state
  - ❖ pending read state
  - ❖ pending write state
- handshake protocol
  - ❖ initially current state is empty
  - ❖ either pending state can be made current and reinitialized to empty

- Type: 1 byte
  - ❖ 10 message types defined
- length: 3 bytes
- content

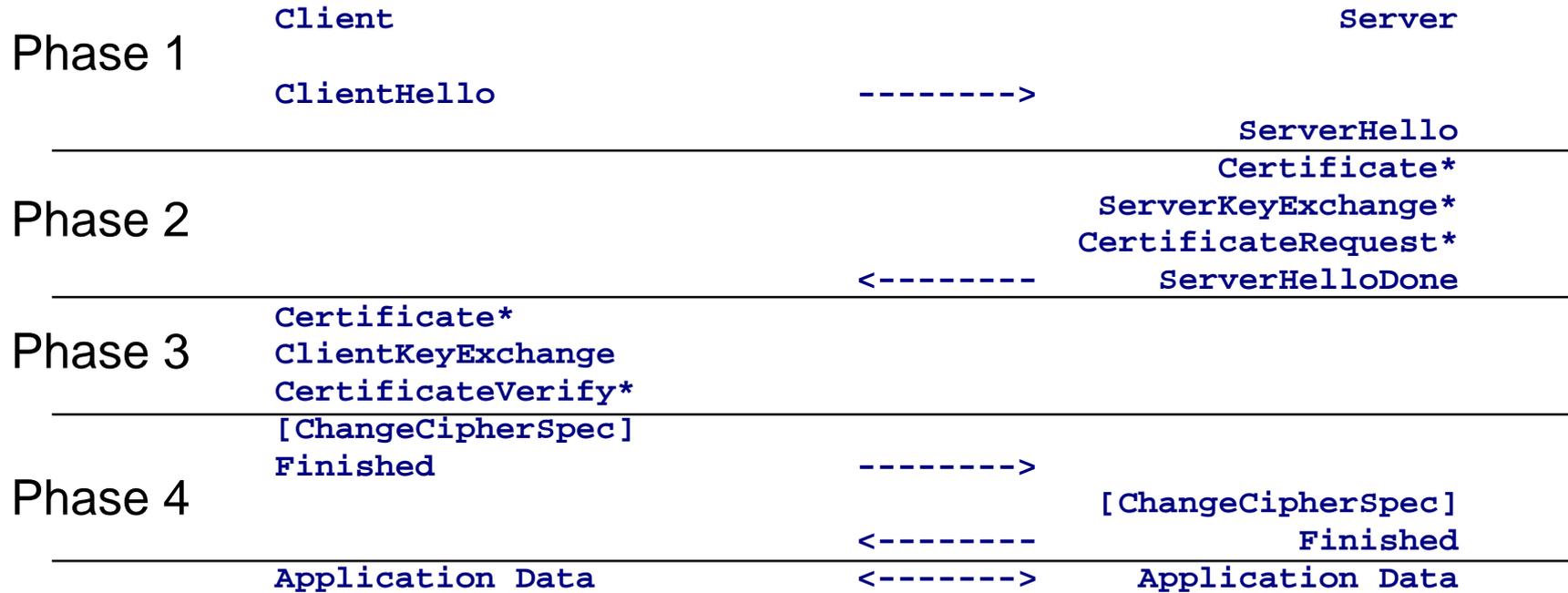


Fig. 1 - Message flow for a full handshake

\* Indicates optional or situation-dependent messages that are not always sent.

- Phase 1:
  - ❖ Establish security capabilities
- Phase 2:
  - ❖ Server authentication and key exchange
- Phase 3:
  - ❖ Client authentication and key exchange
- Phase 4:
  - ❖ Finish

- these handshake messages must occur in order
- optional messages can be eliminated
- 10th message
  - ❖ hello\_request
  - ❖ can be sent anytime from server to client to request client to start handshake protocol to renegotiate session
- change\_cipher\_spec is a separate 1 message protocol
  - ❖ functionally just like a message in the handshake protocol

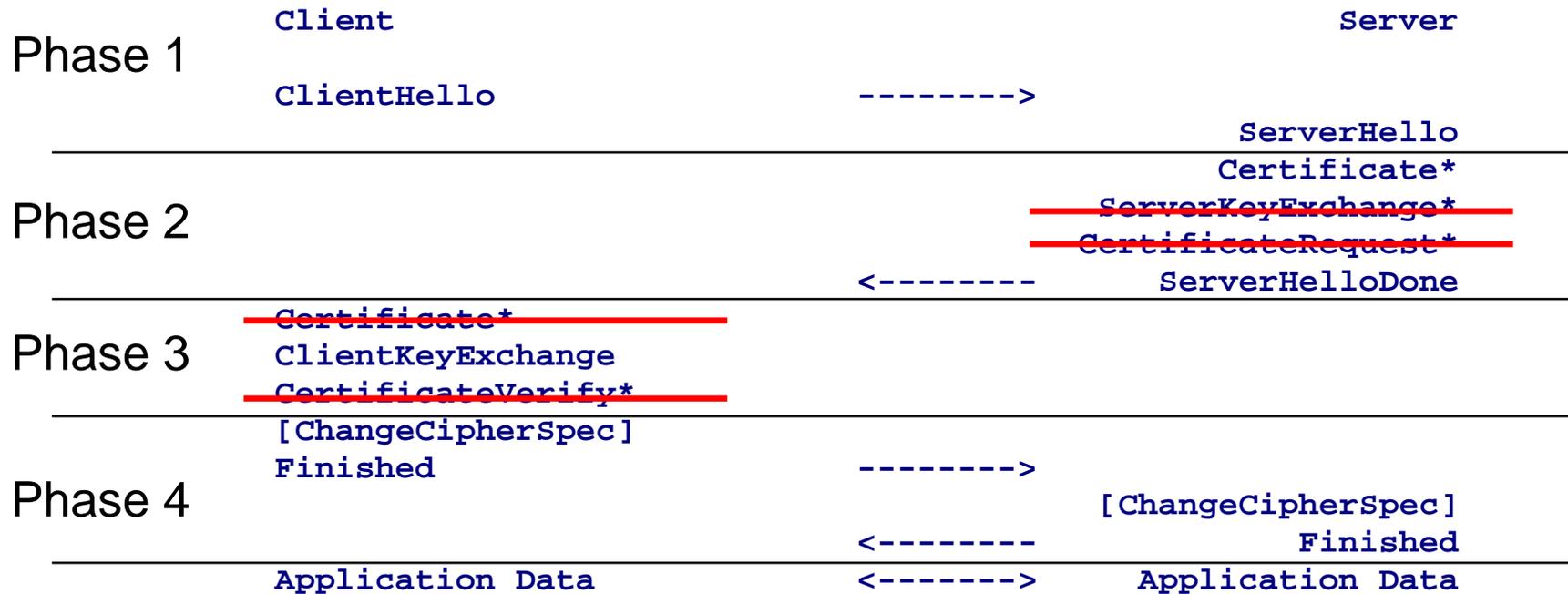


Fig. 1 - Message flow for a full handshake

\* Indicates optional or situation-dependent messages that are not always sent.

- Establish security capabilities
- client hello message
  - ❖ 4 byte timestamp, 28 byte random value
  - ❖ session ID:
    - non-zero for new connection on existing session
    - zero for new connection on new session
  - ❖ client version: highest version
  - ❖ cipher\_suite list: ordered list
    - key exchange method, encryption method, MAC method
  - ❖ compression list: ordered list
- server hello message
  - ❖ 32 byte random value
  - ❖ session ID:
    - new or reuse
  - ❖ version
    - lower of client suggested and highest supported
  - ❖ cipher\_suite list: single choice
  - ❖ compression list: single choice

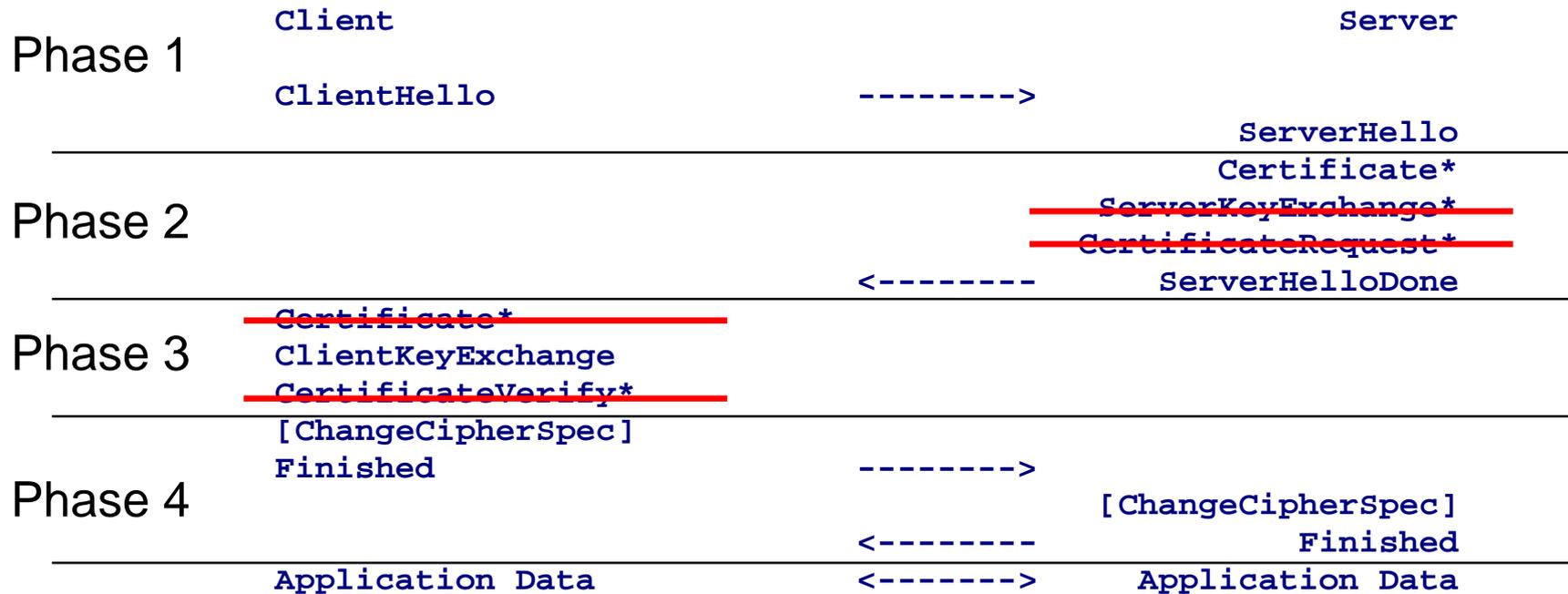


Fig. 1 - Message flow for a full handshake

\* Indicates optional or situation-dependent messages that are not always sent.

- Server authentication and key exchange
- certificate message
  - ❖ server's X.509v3 certificate followed by optional chain of certificates
  - ❖ required for RSA
- server done message
  - ❖ ends phase 2, always required

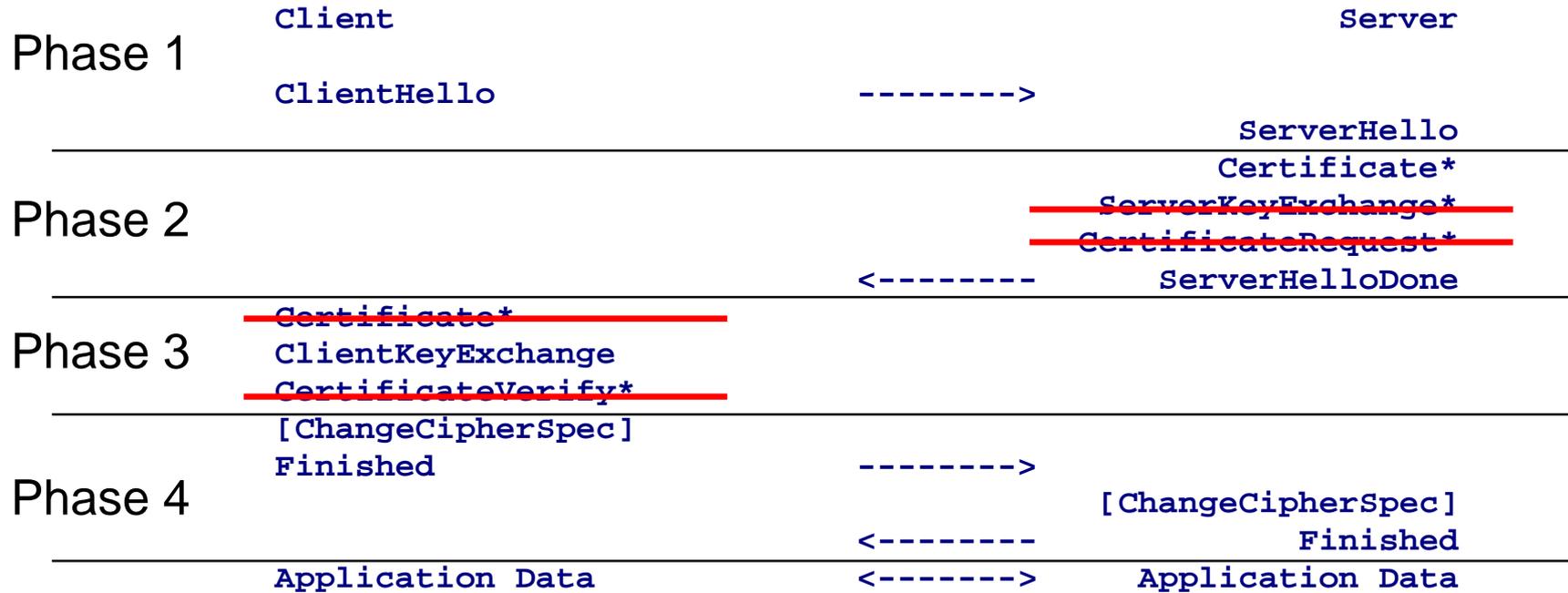
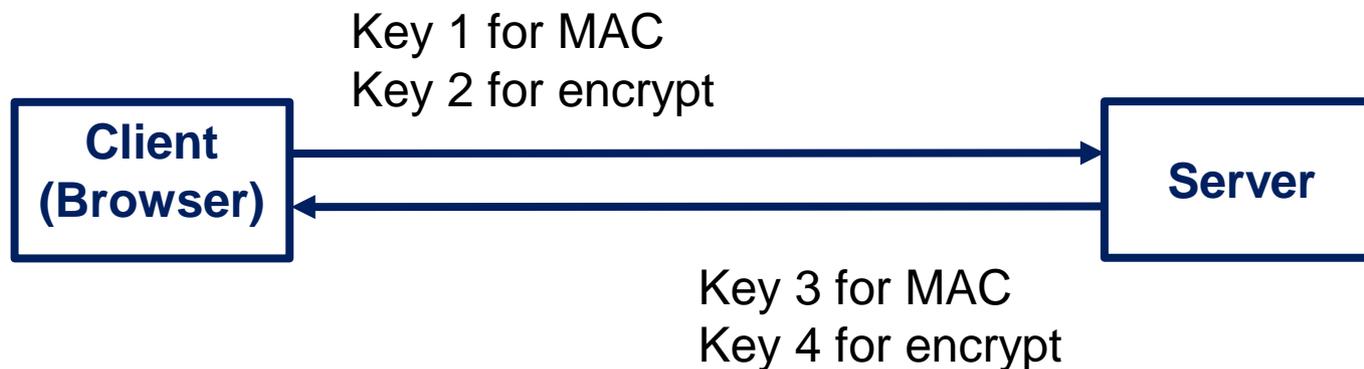


Fig. 1 - Message flow for a full handshake

\* Indicates optional or situation-dependent messages that are not always sent.

- Client authentication and key exchange
- client key exchange message
  - ❖ client generates 48-byte pre-master secret, encrypts with server's RSA public key
- client and server compute 48 byte master secret
  - ❖ using 48-byte pre-master secret, ClientHello.random, ServerHello.random
- client and server compute 4 symmetric keys from master secret



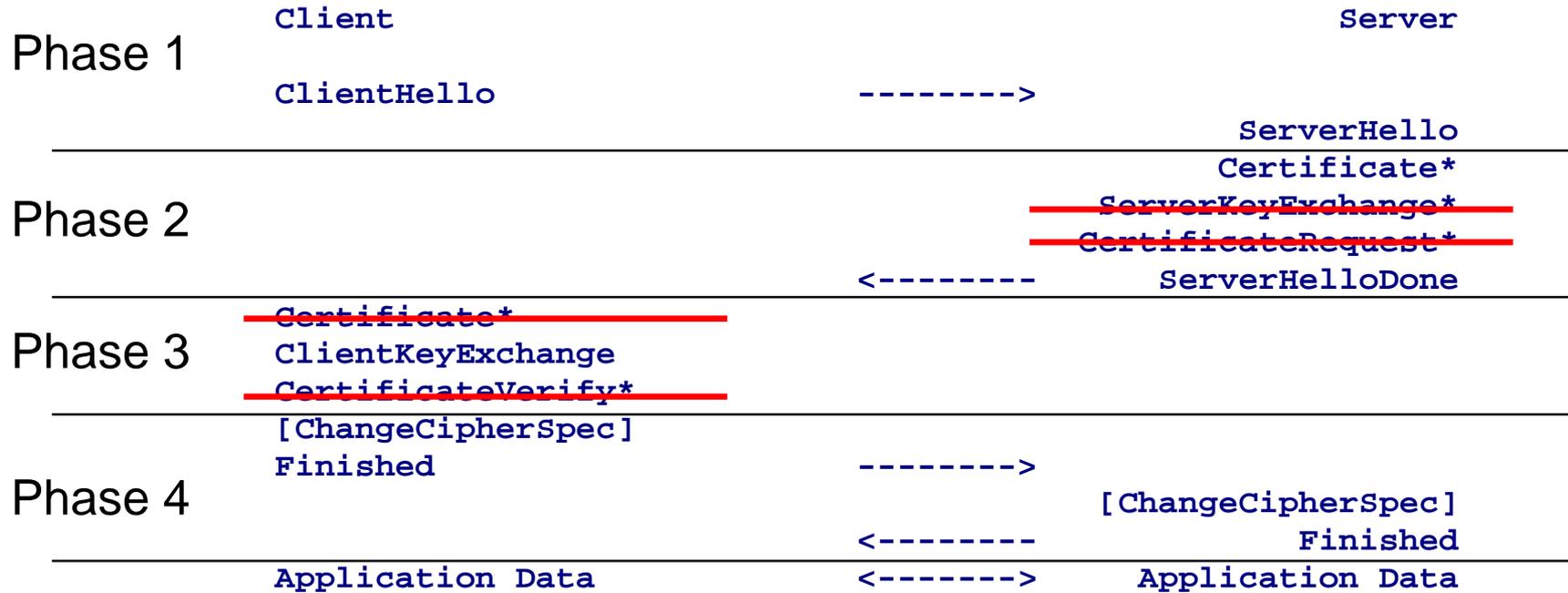


Fig. 1 - Message flow for a full handshake

\* Indicates optional or situation-dependent messages that are not always sent.

- Finish and move to record protocol
- change cipher spec message
  - ❖ not considered part of handshake protocol but in some sense is part of it
  - ❖ 1 byte message protected by current state
  - ❖ copies pending state to current state
- Finished message
  - ❖ sent under new algorithms and keys
  - ❖ content is MAC of all previous messages with master secret and constant “client finished” or “server finished”

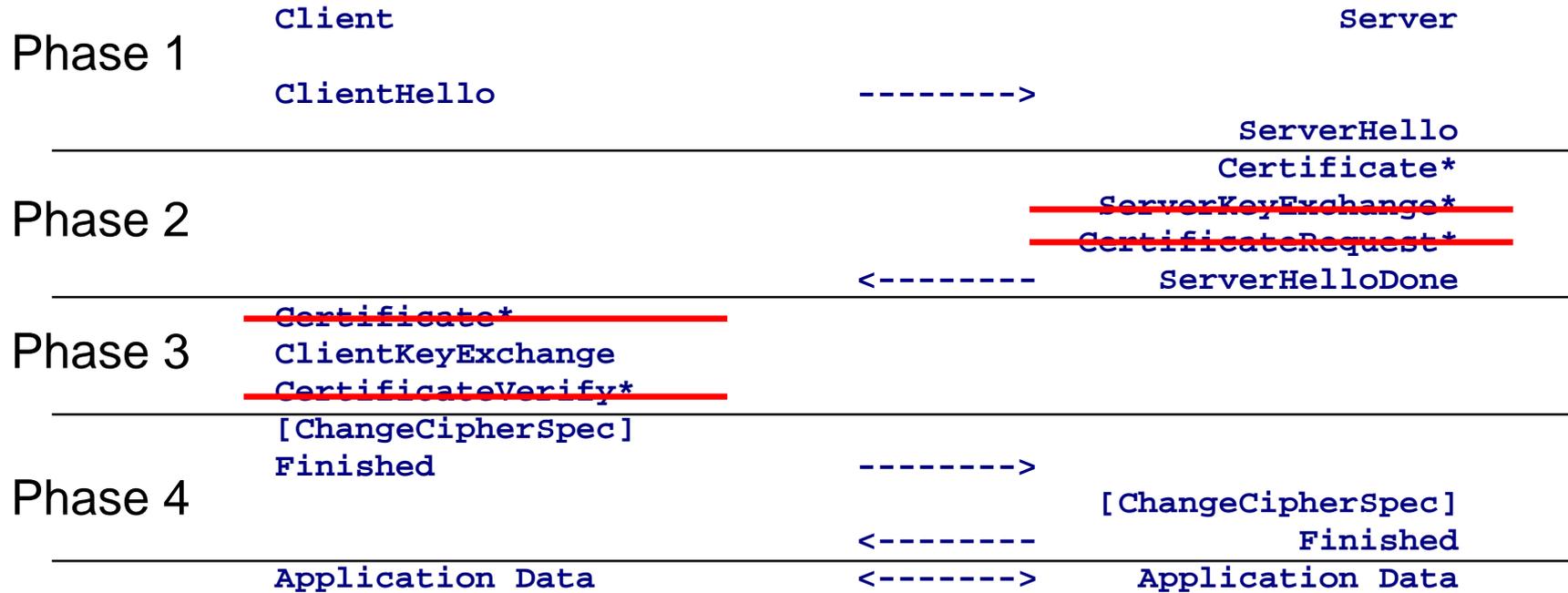


Fig. 1 - Message flow for a full handshake

\* Indicates optional or situation-dependent messages that are not always sent.

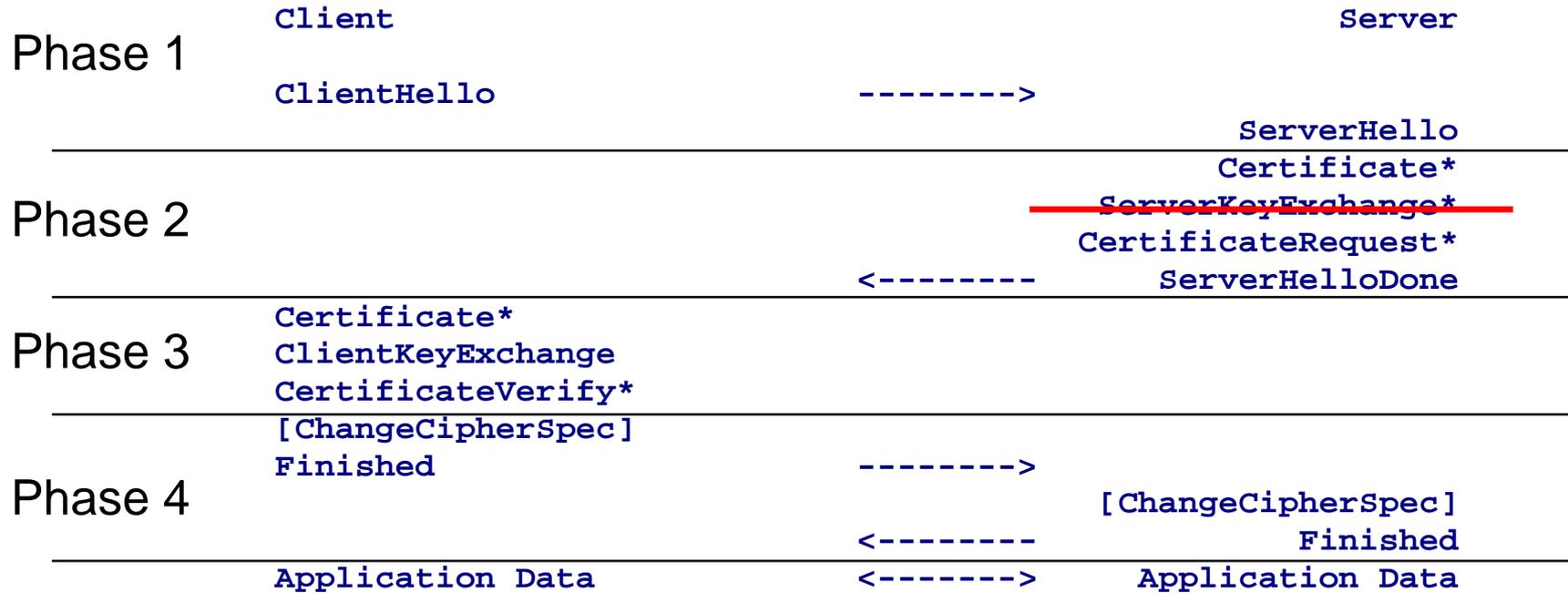


Fig. 1 - Message flow for a full handshake

\* Indicates optional or situation-dependent messages that are not always sent.

- Server authentication and key exchange
- certificate message
  - ❖ server's X.509v3 certificate followed by optional chain of certificates
  - ❖ required for RSA
- **certificate request message**
  - ❖ **request a certificate from client**
  - ❖ **specifies Certificate Type and Certificate Authorities**
- server done message
  - ❖ ends phase 2, always required

- Client authentication and key exchange
- **certificate message**
  - ❖ **client's X.509v3 certificate followed by optional chain of certificates**
- client key exchange message
  - ❖ client generates 48-byte pre-master secret, encrypts with server's RSA public key
- **certificate verify message**
  - ❖ **signs hash of master secret (established by key exchange) and all handshake messages so far**
- client and server compute 48 byte master secret
  - ❖ using 48-byte pre-master secret, ClientHello.random, ServerHello.random
- client and server compute 4 symmetric keys from master secret

- 2 byte alert messages
  - ❖ 1 byte level
    - fatal or warning
  - ❖ 1 byte
    - alert code

## Warning or fatal

```
close_notify(0),  
unexpected_message(10),  
bad_record_mac(20),  
decryption_failed(21),  
record_overflow(22),  
decompression_failure(30),  
handshake_failure(40),  
bad_certificate(42),  
unsupported_certificate(43),  
certificate_revoked(44),  
certificate_expired(45),  
certificate_unknown(46),  
illegal_parameter(47),  
unknown_ca(48),  
access_denied(49),  
decode_error(50),  
decrypt_error(51),  
export_restriction(60),  
protocol_version(70),  
insufficient_security(71),  
internal_error(80),  
user_canceled(90),  
no_renegotiation(100),
```

- always fatal
  - ❖ unexpected\_message
  - ❖ bad\_record\_mac
  - ❖ decompression\_failure
  - ❖ handshake\_failure
  - ❖ illegal\_parameter

# SSL Man-in-the-Middle (MITM) Attack



## SSL Lock Icon Evolution by Browser

IE	 v5,6	 v7,8	 v9	
Firefox:	 v2	 v3,4 osx	 v3,4 win	 v3,4 linux
Chrome:				
Safari:	 osx	 win		
Opera:				
Konqueror:				

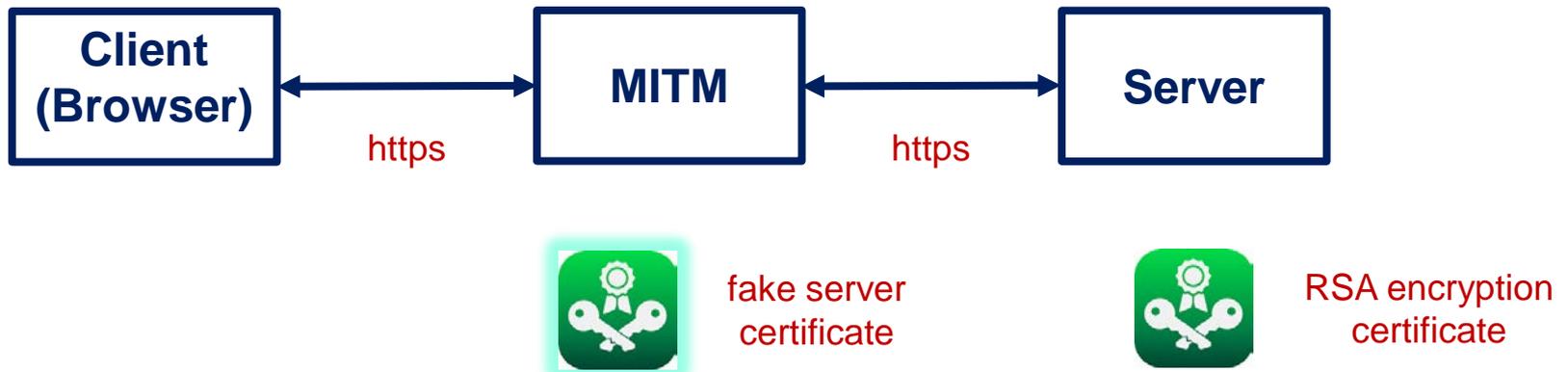
<http://elie.im/blog/>

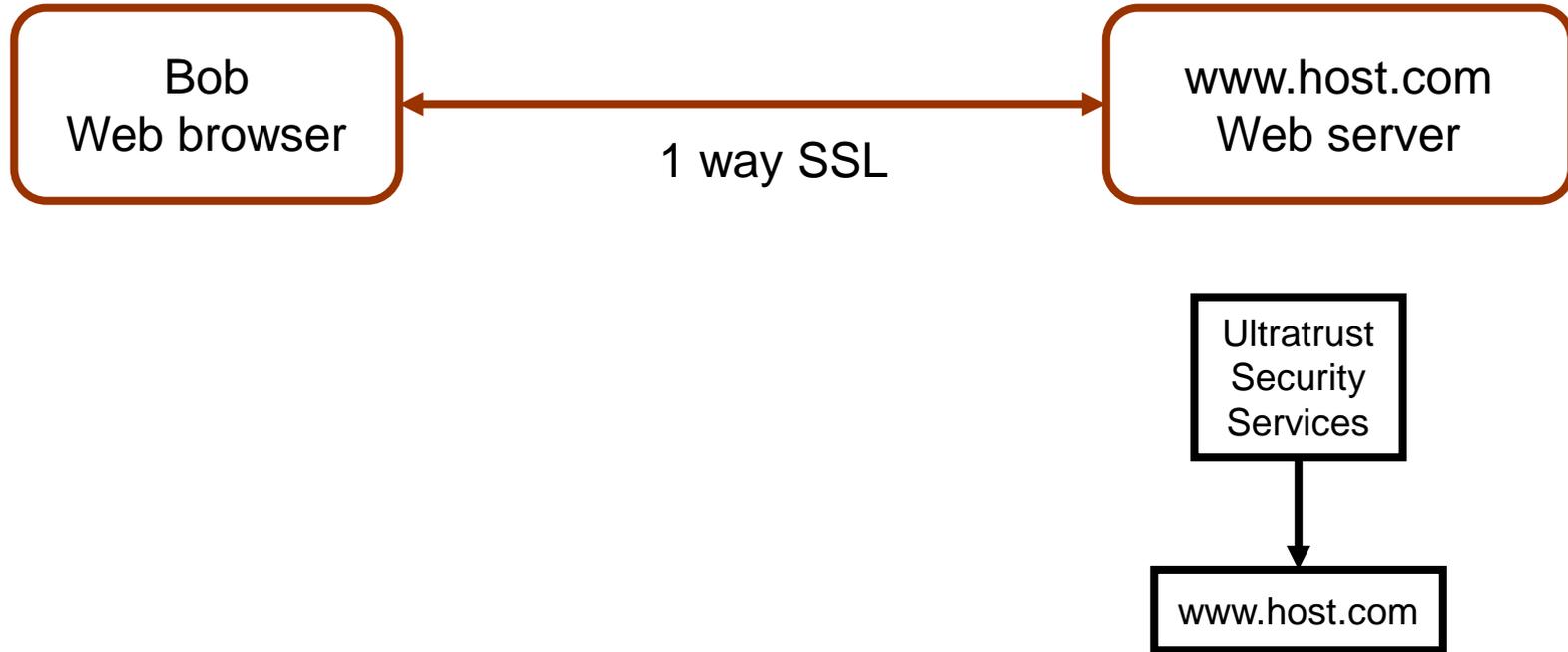


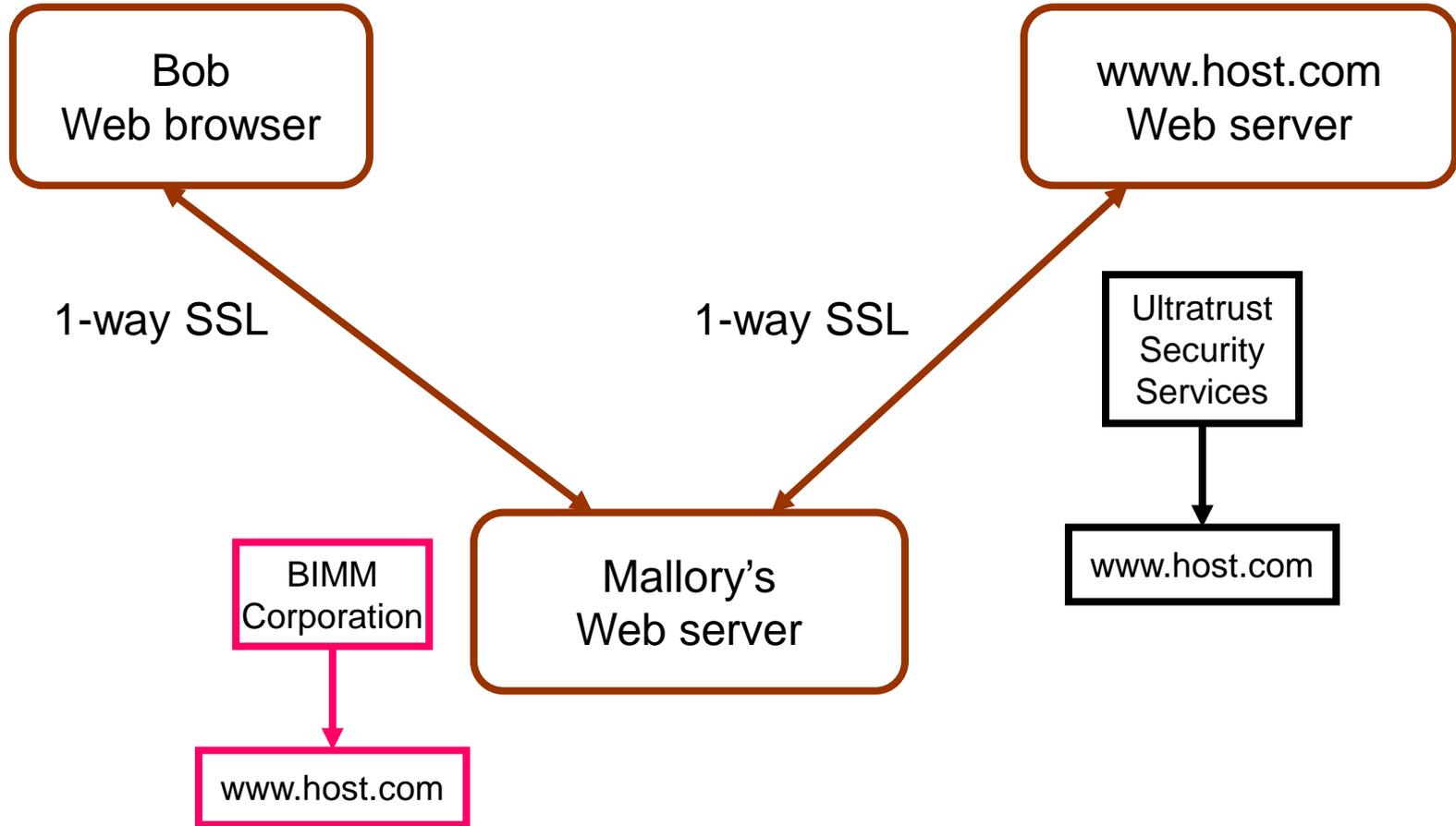
RSA encryption  
certificate

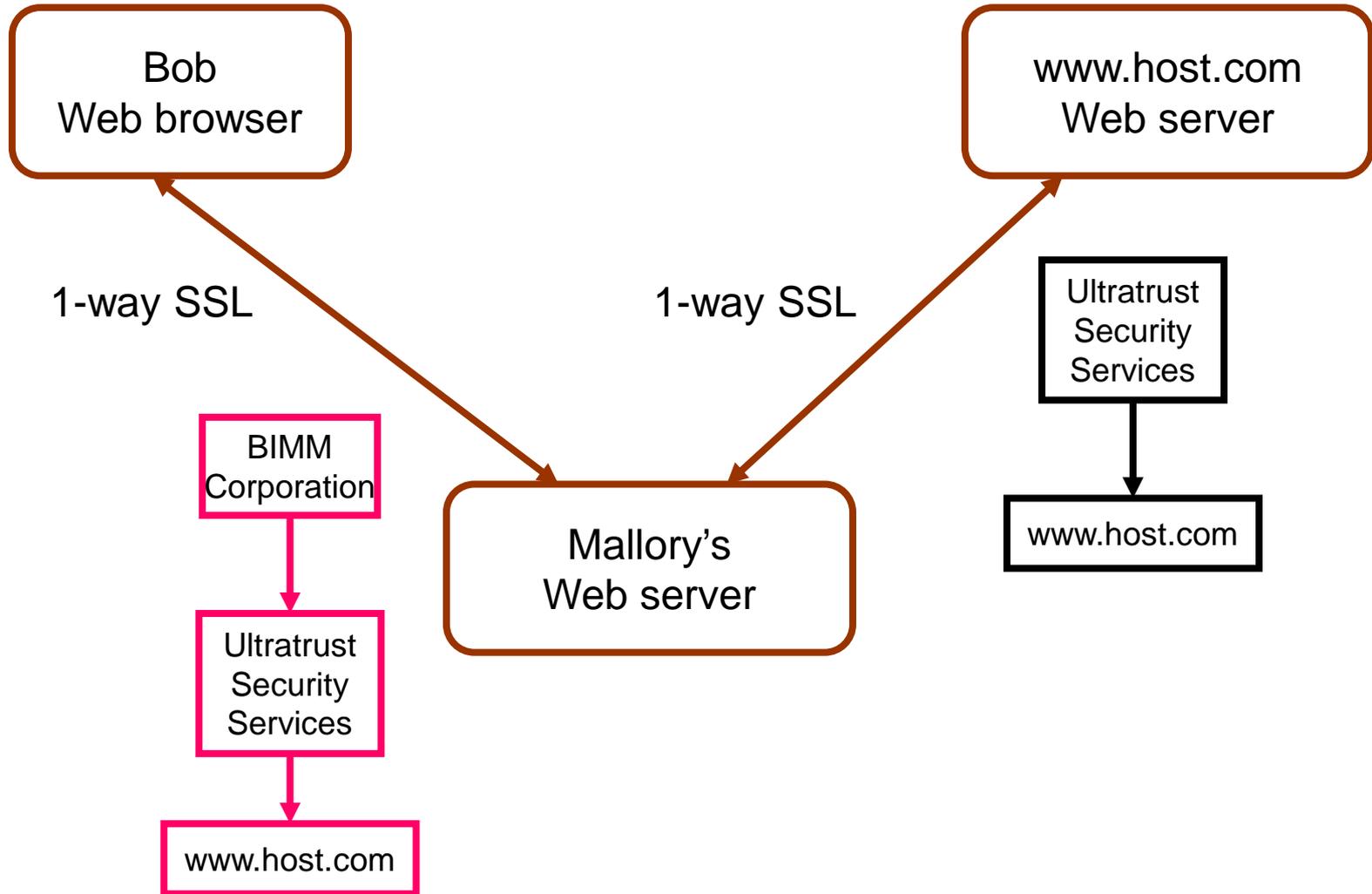


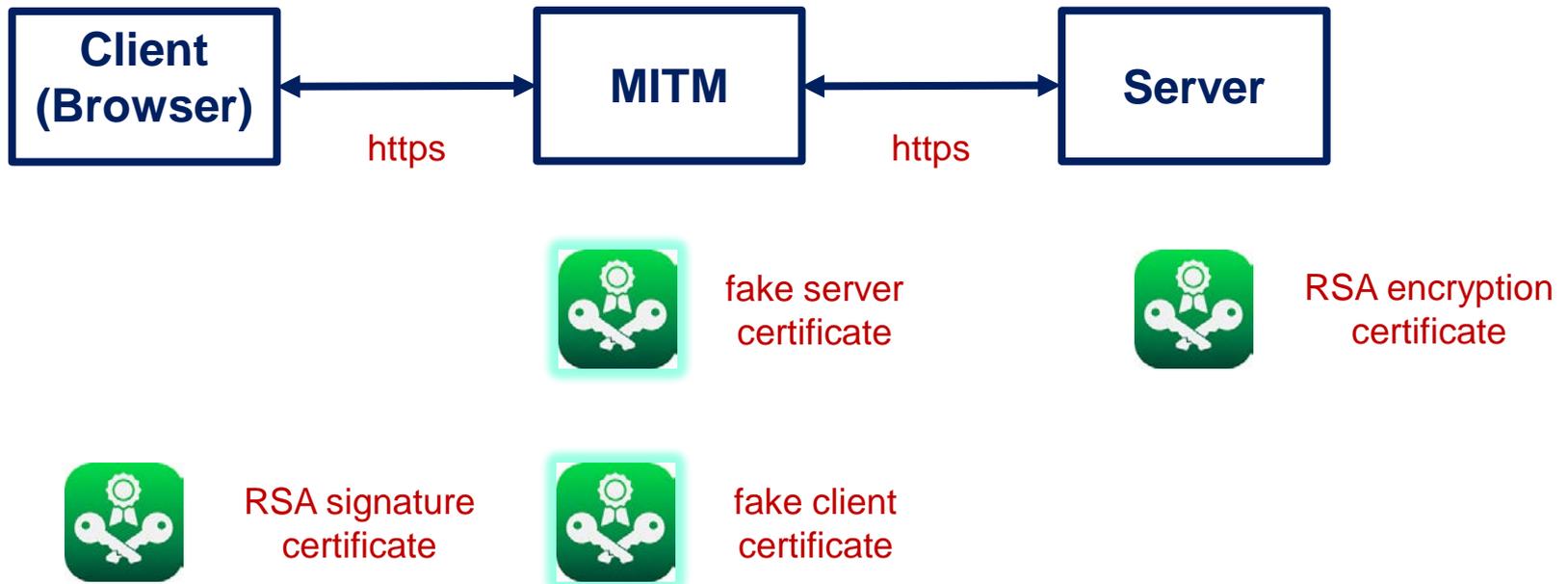
RSA encryption  
certificate











# OpenSSL Heartbleed Attack

### HOW THE HEARTBLEED BUG WORKS:

