CSCE 465 Computer & Network Security

Instructor: Abner Mendoza

SSL/TLS

Roadmap

- Overview
- The SSL/TLS Record Protocol
- The SSL/TLS Handshake and Other Protocols

Protocols

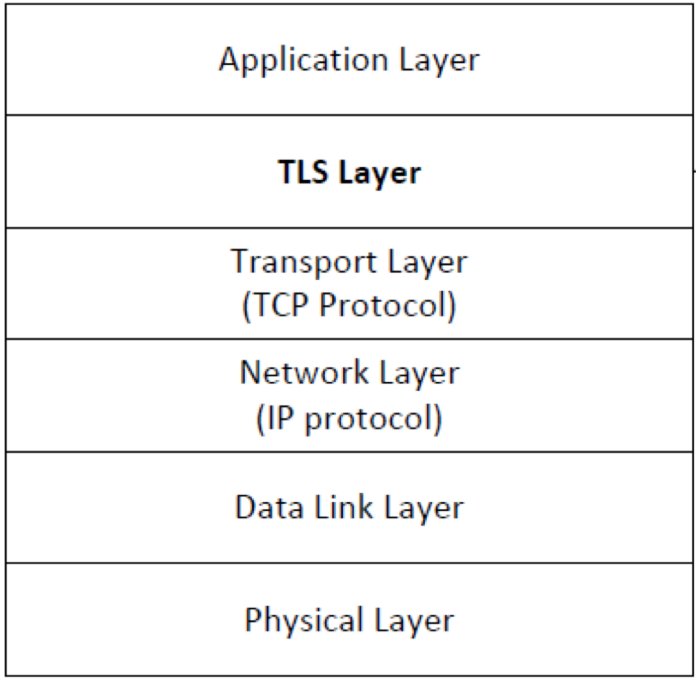
- Goal: application independent security
  - Originally for HTTP, but now used for many applications
  - Each application has an assigned TCP port, e.g., https (HTTP over SSL) uses port 443
- Secure Sockets Layer (SSL)
  - the de facto standard for web-based security
  - v3 was developed with public review
- Transport Layer Security (TLS)
  - TLS v1.0 very close to SSL v3.1
  - Currently, TLS 1.2 (SSL 3.3)

Overview of TLS

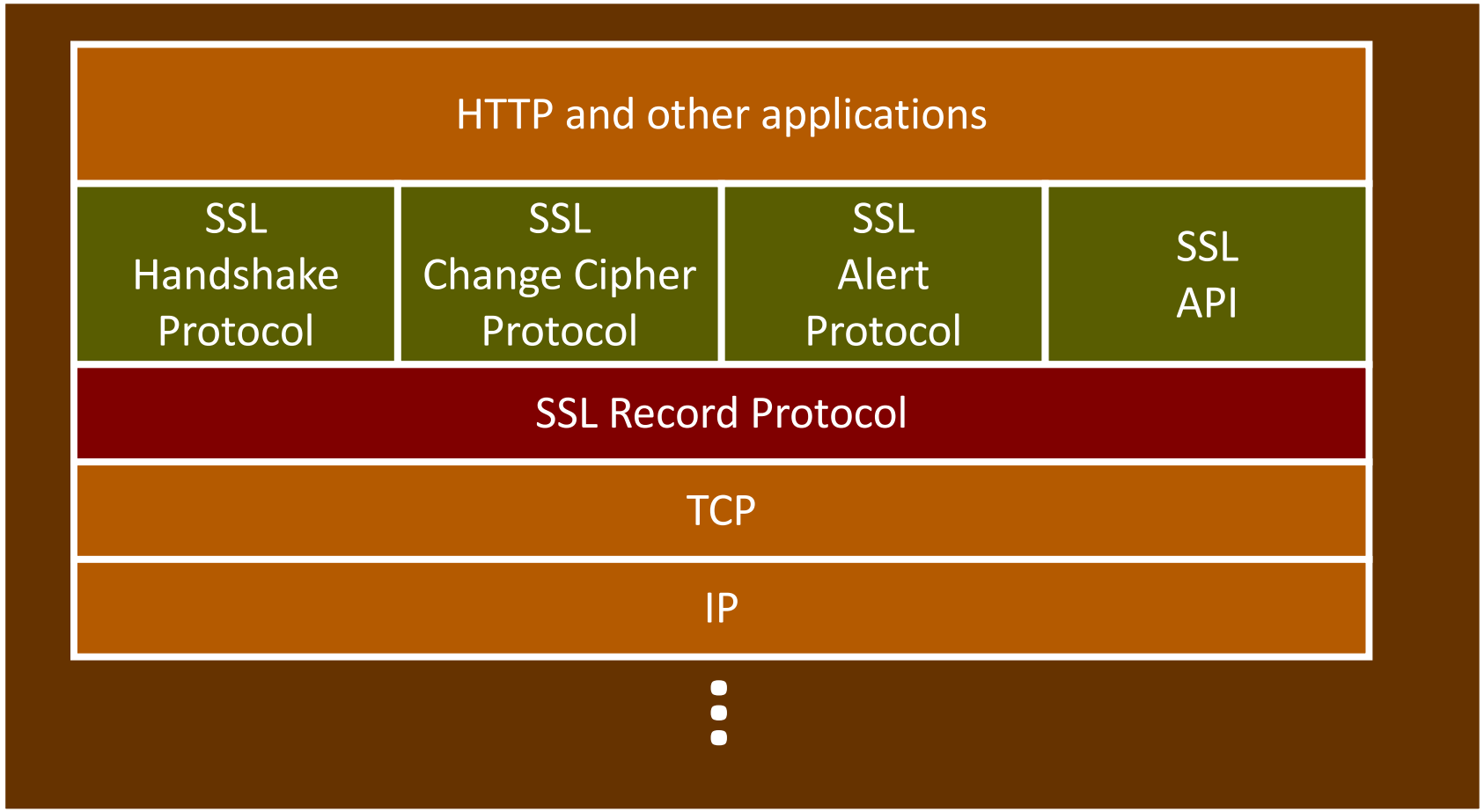
- Transport Layer Security (TLS) is a protocol that provides a secure channel between two communicating applications. The secure channel has 3 properties:
  - Confidentiality: Nobody other than the two ends of the channel can see the actual content of the data transmitted.
  - Integrity: Channel can detect any changes made to the data during transmission
  - Authentication: At least one end of the channel needs to be authenticated, so the other end knows who it is talking to.

TLS Layer

- TLS sits between the Transport and Application layer
  - Unprotected data is given to TLS by Application layer
  - TLS handles encryption, decryption and integrity checks
  - TLS gives protected data to Transport layer



SSL Architecture



- Relies on TCP for reliable communication

Architecture (Cont'd)

- Handshake protocol: establishment of a session key
- Change Cipher protocol: start using the previously-negotiated encryption / message authentication
- Alert protocol: notification (warnings or fatal exceptions)
- Record protocol: protected (encrypted, authenticated) communication between client and server

SSL Services

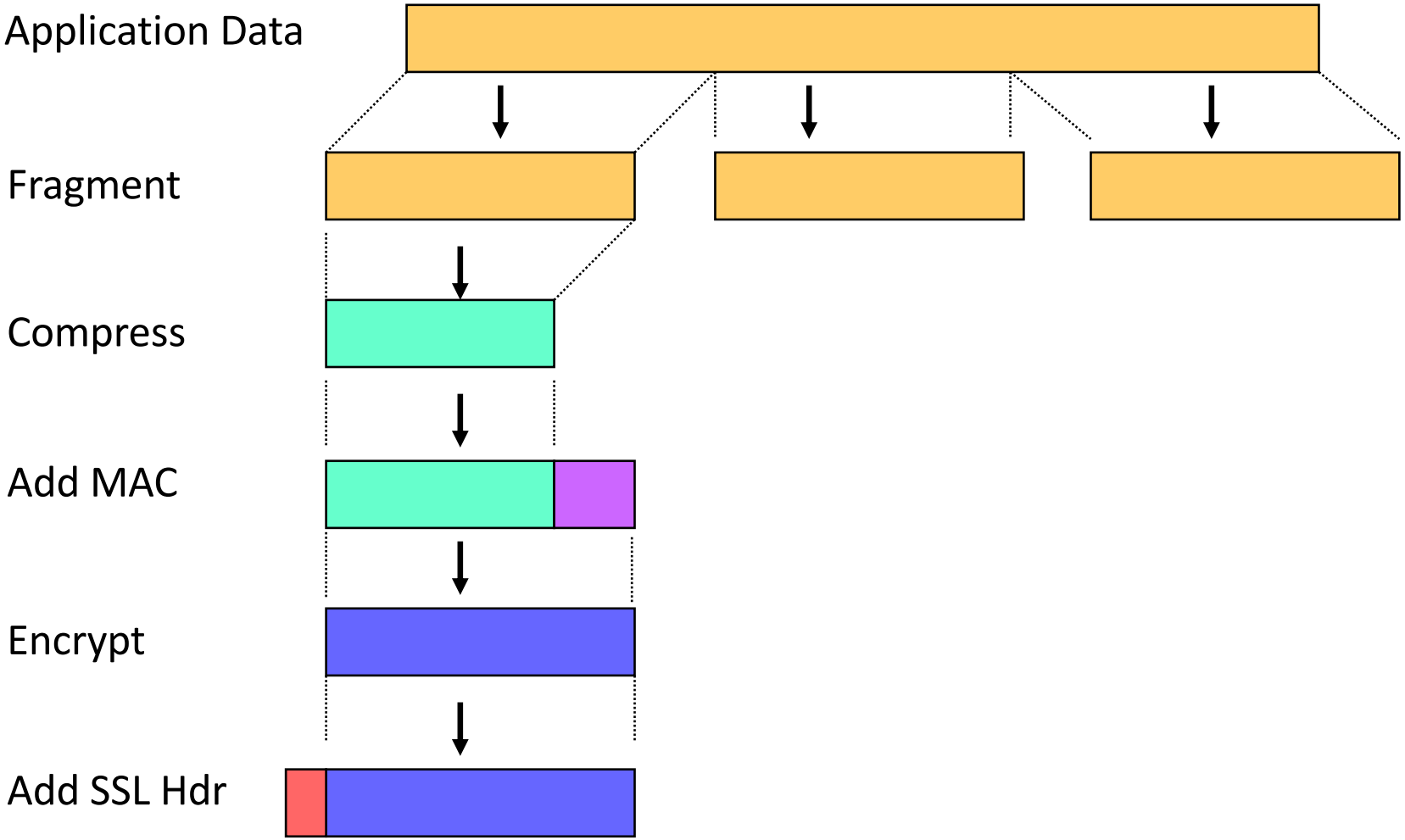
- Peer authentication
- Negotiation of security parameters
- Generation / distribution of session keys
- Data confidentiality
- Data integrity
- Authentication

The SSL Record Protocol

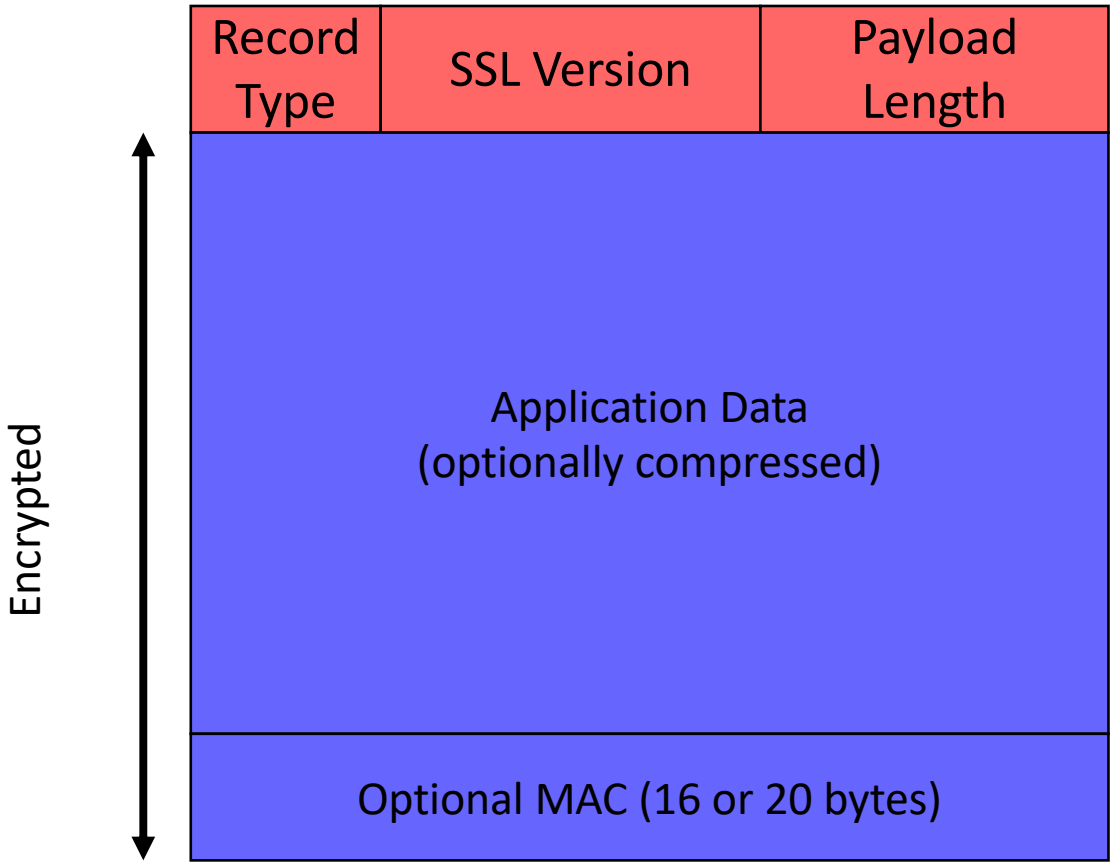
Protocol Steps

1. Fragment data stream into records
   - each with a maximum length of 2^{14} (=16K) bytes
2. Compress each record
3. Create message authentication code for each record
4. Encrypt each record

Steps… (cont'd)

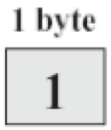

Application Data
Fragment
Compress
Add MAC
Encrypt
Add SSL Hdr

SSL Record Format

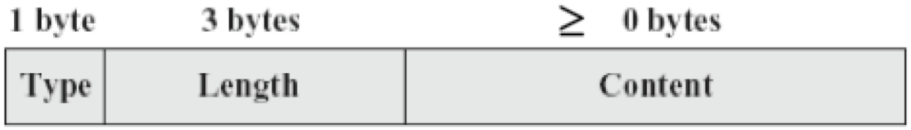


- There is, unfortunately, some version number silliness between v2 and v3; see text for (ugly) details

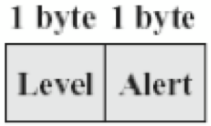
Possible Record “Payloads”



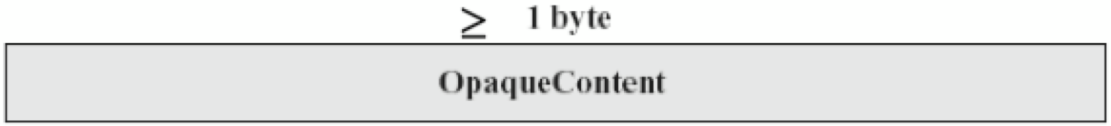
(a) Change Cipher Spec Protocol



(c) Handshake Protocol



(b) Alert Protocol



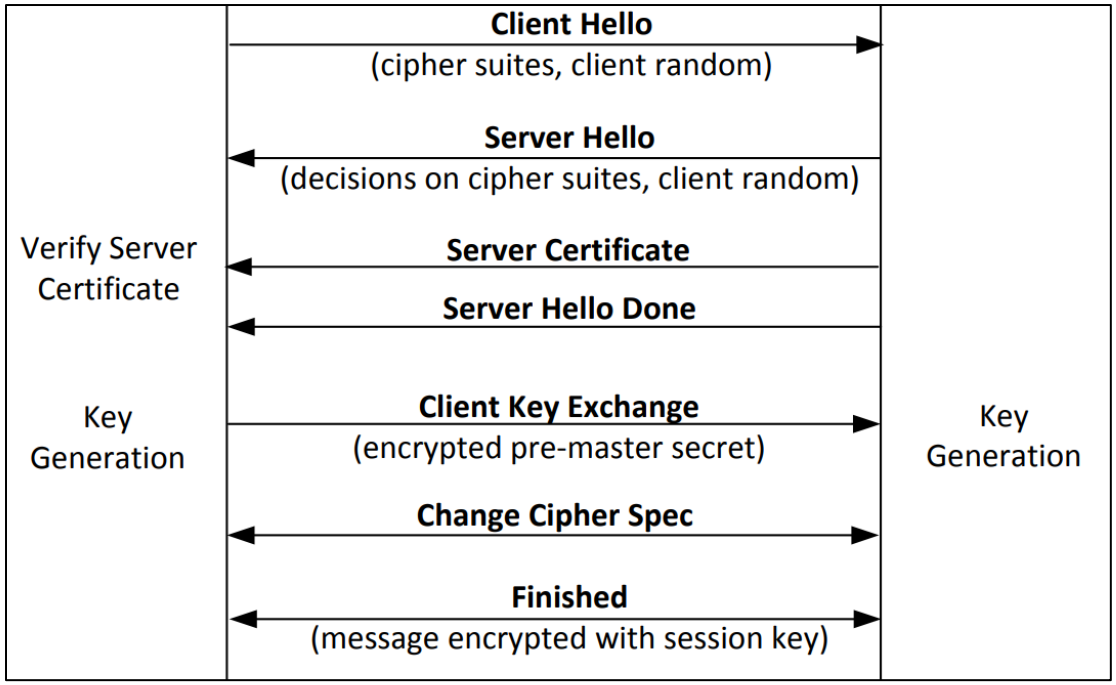
(d) Other Upper-Layer Protocol (e.g., HTTP)

Handshake Protocol

TLS Handshake

- Before a client and server can communicate securely, several things need to be set up first:
  - Encryption algorithm and key
  - MAC algorithm
  - Algorithm for key exchange
- These cryptographic parameters need to be agreed upon by the client and server
- This is the primary purpose of the handshake protocol

TLS Handshake Protocol



Phases of Protocol

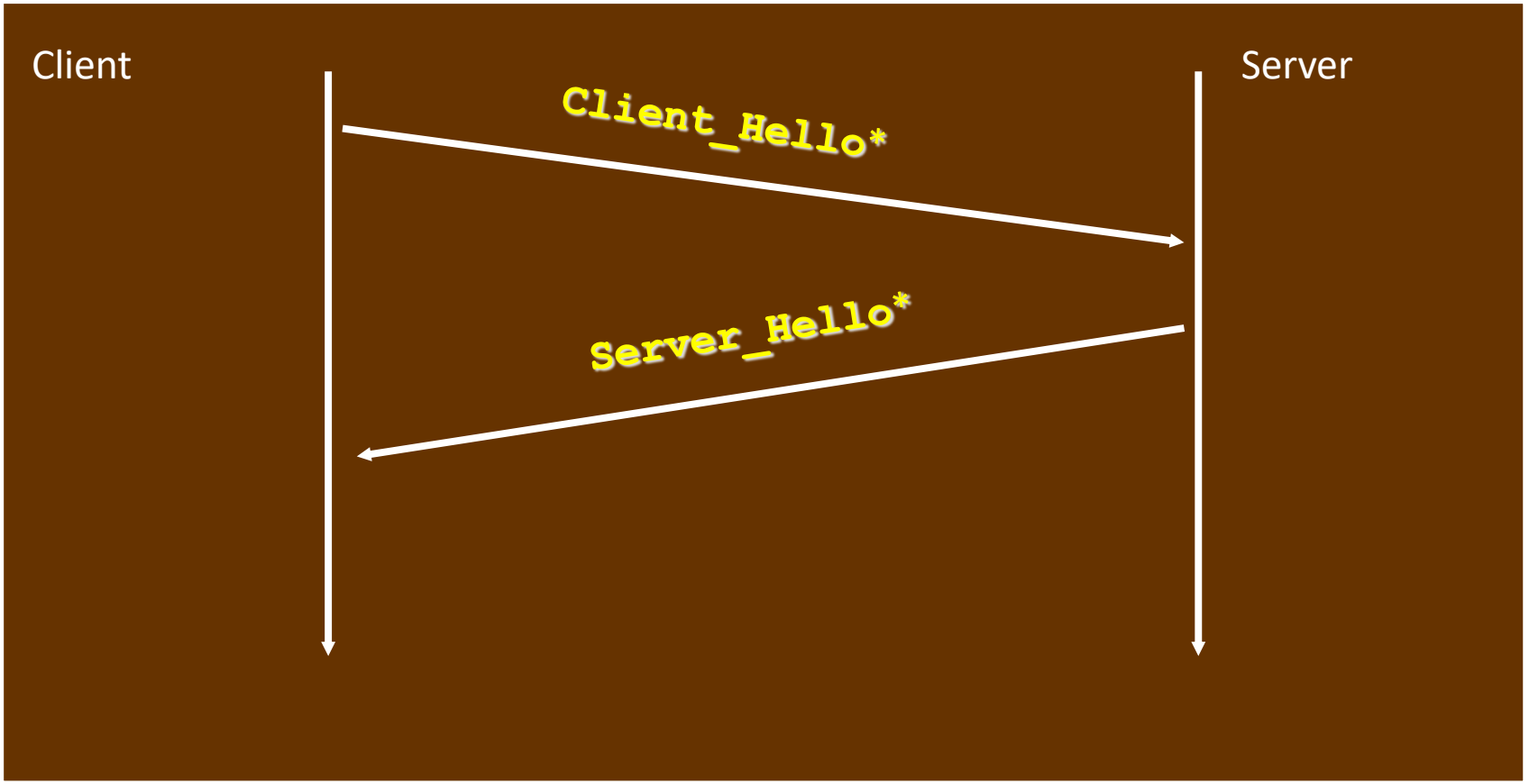
I. Establish security capabilities
   - version of SSL to use
   - cipher + parameters to use

II. Authenticate server (optional), and perform key exchange

III. Authenticate client (optional), and perform key exchange

IV. Finish up

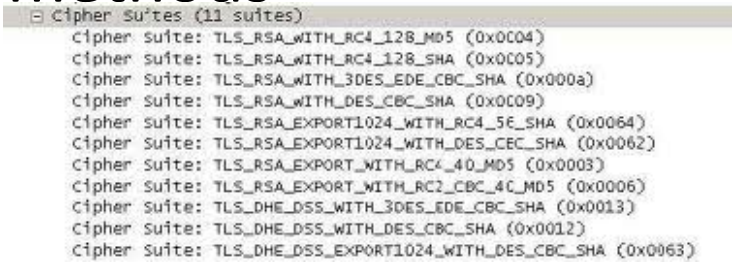
I. Establish Security Capabilities



- Messages marked with * are mandatory

`Client_Hello` Message

- Transmitted in plaintext
- Contents
  - highest SSL version understood by client
  - R_C: a 4-byte timestamp + 28-byte random number
  - session ID: 0 for a new session, non-zero for a previous session
  - list of supported cryptographic algorithms
  - list of supported compression methods



`Server_Hello` Message

- Also transmitted in plaintext
- Contents
  - minimum of (highest version supported by server, highest version supported by client)
  - R_S: 4-byte timestamp and 28-byte random number
  - session ID
  - a cryptographic choice selected from the client's list
  - a compression method selected from the client's list

II. Server Auth. / Key Exchange

- The Server_Certificate message is optional, but almost always used in practice

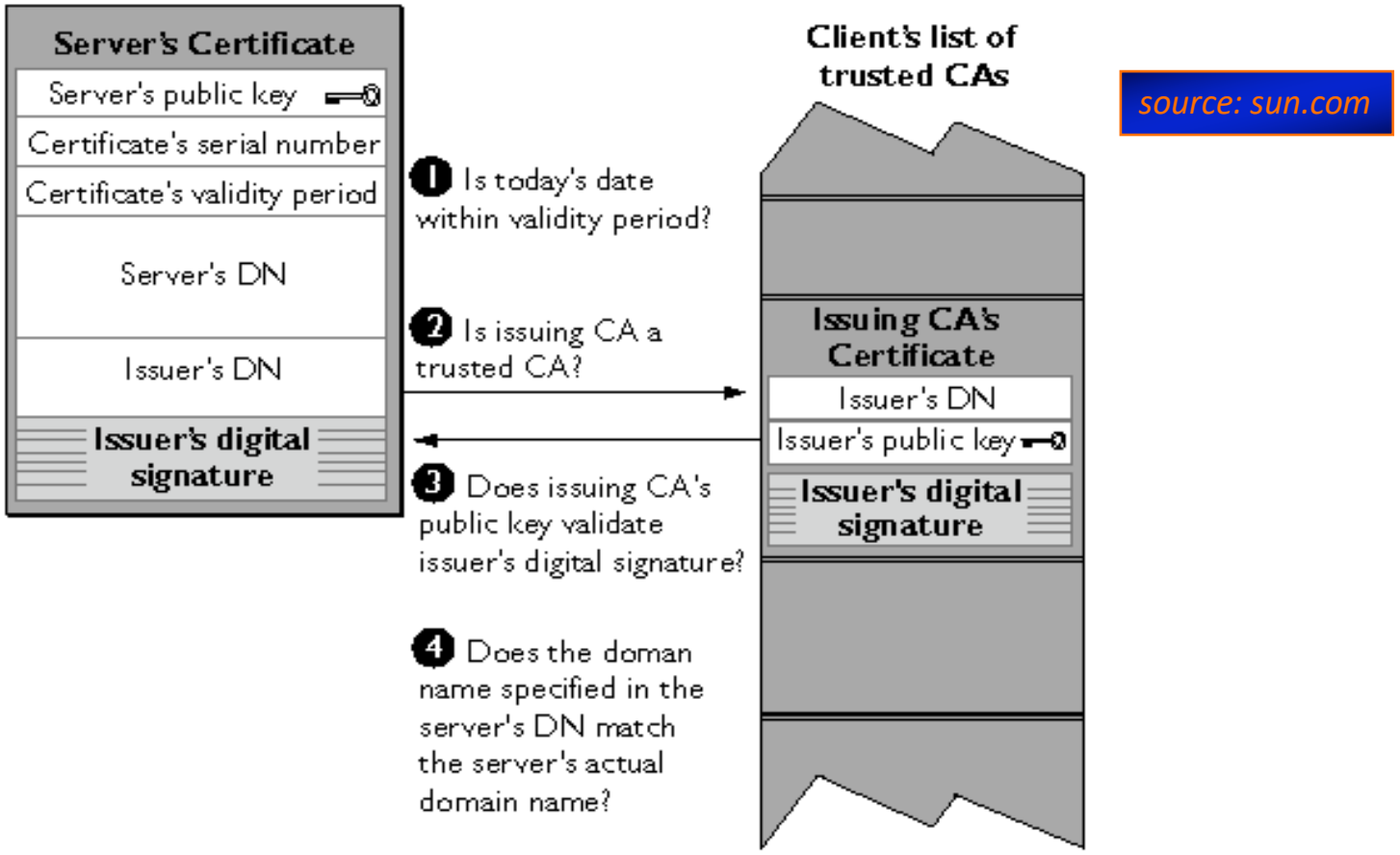
`Server_Certificate` Message

- Contains a certificate with server's public key, in X.509 format
  - or, a chain of certificates if required
- The server certificate is necessary for any key exchange method except for anonymous Diffie-Hellman

Certificate Verification

- The client first does a validation check of the certificate
  - Check expiration date, signature validity, etc.
  - Hostname and certificate's common name match
- The client needs to have the singing CA's public-key certificate.

Authenticating the Server



- Step #4: Domain name in certificate must match domain name of server (not part of SSL protocol, but clients should check this)

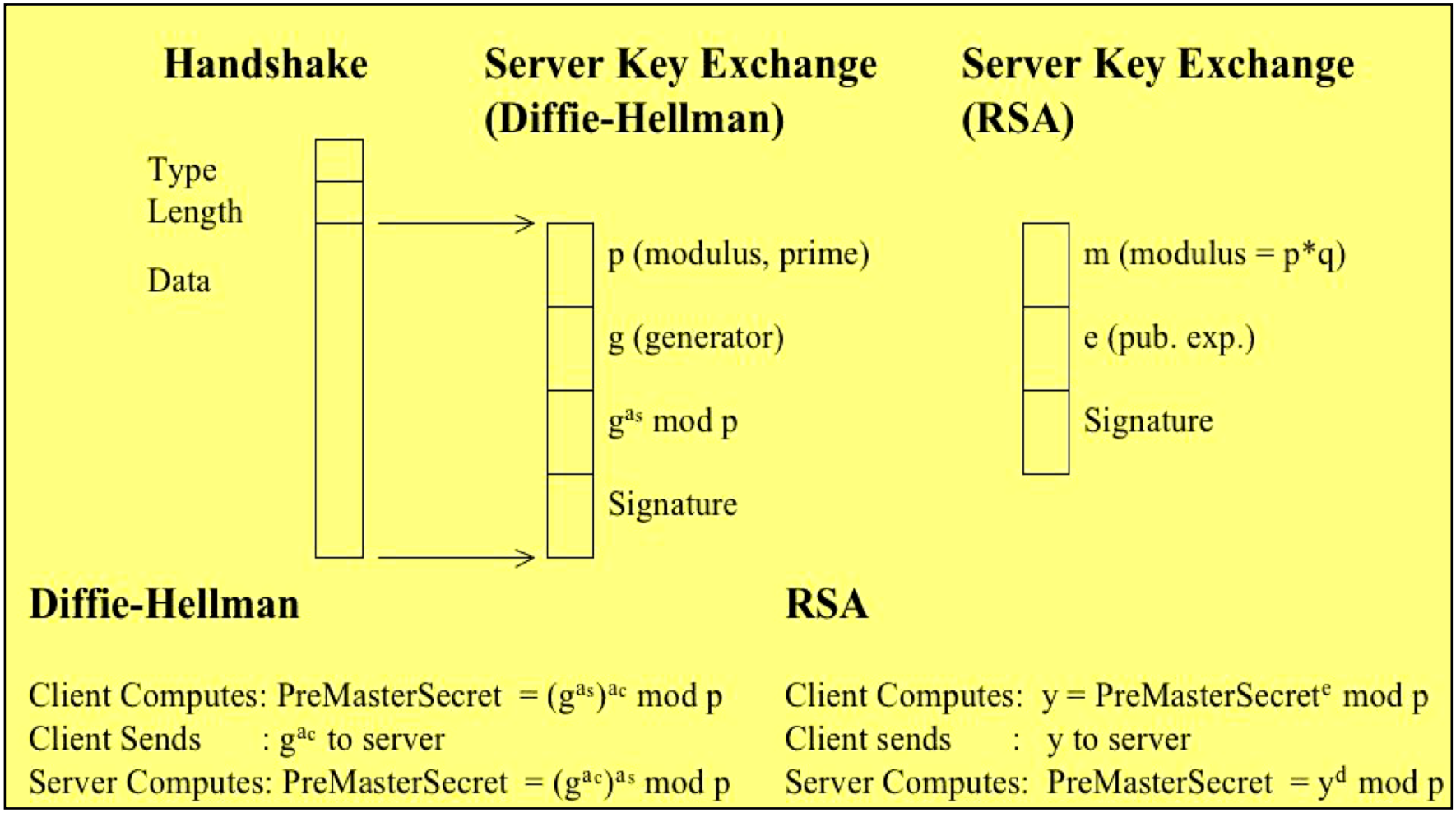
Key Exchange Methods Supported

- RSA (server must have a certificate)
- Ephemeral Public Key
  - public keys are exchanged, signed using long-term RSA keys
- (Fixed Diffie-Hellman
  - server provides the D-H public parameters in a certificate
  - client responds with D-H public key either in a certificate, or in a key exchange message
- Anonymous Diffie-Hellman)

`Server_Key_Exchange` Message

- Needed for…
  - anonymous D-H
  - ephemeral public key

Server Key Exchange

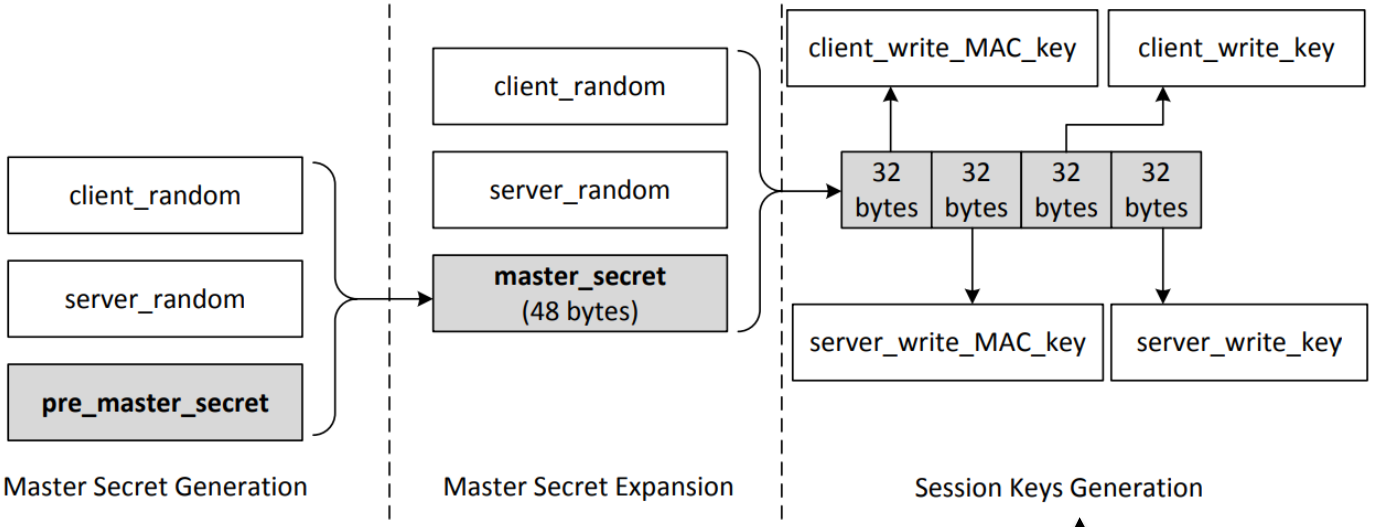


Key Generation and Exchange

- Although public-key algorithms can be used to encrypt data, it is much more expensive than secret-key algorithms.
  - TLS uses PKI for key exchange.
  - After that, server and client switch to secret-key encryption algorithm

- The entire key generation consists of three steps:
  - Step 1: Generating pre-master secret
  - Step 2: Generating master secret
  - Step 3: Generating session keys

Key Generation and Exchange



These keys are used to protect an SSL session

`Client_Certificate_Request` Msg.

- Normally not used, because in most applications
  - only the server is authenticated
  - client is authenticated at the application layer, if needed
- Two parameters
  - certificate type accepted, e.g., RSA/signature only, DSS/signature only, …
  - list of certificate authorities recognized (i.e., trusted third parties)

III. Client Auth. / Key Exchange

`Client_Certificate` Message

- Contains a certificate, or chain of certificates if needed

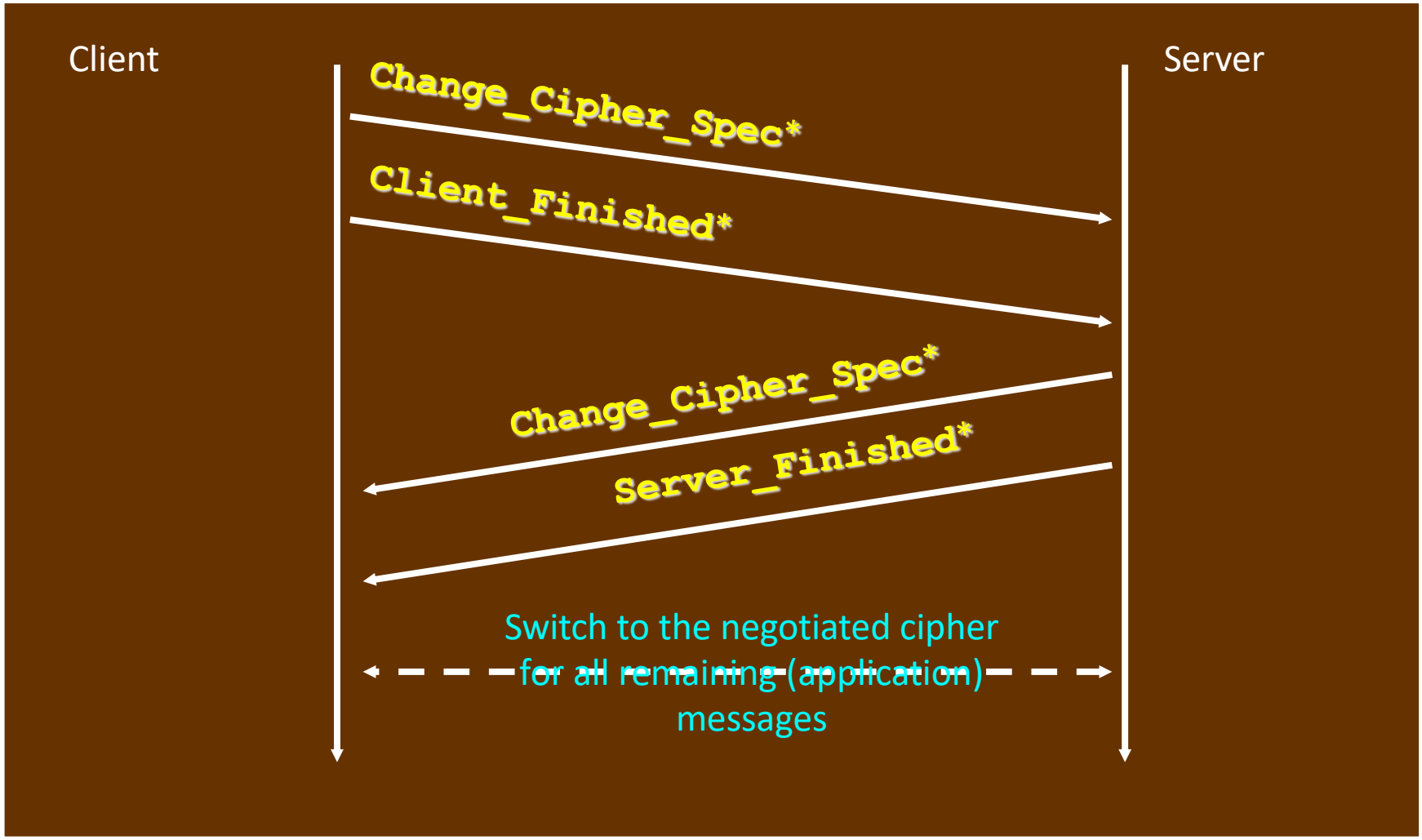
`Client_Key_Exchange` Message

- If using RSA, the pre-master secret S, encrypted with the server's public key
- If using D-H, the client's public key

`Client_Certificate_Verify` Msg

- Proves the client is the valid owner of a certificate (i.e., knows the corresponding private key)
- Only sent following any client certificate that has signing capability

IV. Finish Up


Client
Server
Change_Cipher_Spec*
Client_Finished*
Change_Cipher_Spec*
Server_Finished*
Switch to the negotiated cipher
for all remaining (application)
messages

`Change_Cipher_Spec` Msg

- Confirms the change of the current state of the session to a newly-negotiated set of cryptographic parameters
- **`Finished`** Messages
  - keyed hash of the previous handshake messages to prevent man-in-the-middle-attacks from succeeding

“Abbreviated” Protocol Possible

- Allows resumption of a previously-established session
  - does not require authentication of server or client
  - does not exchange keys
- Details omitted

Network Traffics During TLS Handshake

Since TLS runs top of TCP, a TCP connection needs to be established before the handshake protocol. This is how the packet exchange looks between a client and server during a TLS handshake protocol captured using Wireshark:

No.	Source	Destination	Protocol	Info
1	10.0.2.45	10.0.2.35	TCP	59930 –> 11110 [SYN] Seq=0 Win=14600 Len=0 MSS=1460...
2	10.0.2.35	10.0.2.45	TCP	11110 –> 59930 [SYN, ACK] Seq=0 Ack=1 Win=14480...
3	10.0.2.45	10.0.2.35	TCP	59930 –> 11110 [ACK] Seq=1 Ack=1 Win=14720 Len=0...
4	10.0.2.45	10.0.2.35	TLSv1.2	Client Hello
6	10.0.2.35	10.0.2.45	TLSv1.2	Server Hello, Certificate, Server Hello Done
8	10.0.2.45	10.0.2.35	TLSv1.2	Client Key Exchange, Change Cipher Spec, Finished
9	10.0.2.35	10.0.2.45	TLSv1.2	New Session Ticket, Change Cipher Spec, Finished

Creating the "Master" Secret

- The master secret is a one-time (per session) 48-byte (= 16+16+16) value
- Parameters
  - the pre-master secret S has previously been communicated using RSA or D-H
  - the client nonce R_c
  - the server nonce R_s
- Computation: K =

  MD5 (S | SHA-1("A" | S | R_c | R_s)) |
  MD5 (S | SHA-1("BB" | S | R_c | R_s)) |
  MD5 (S | SHA-1("CCC" | S | R_c | R_s))

Cryptographic Parameters

- Generated from
  - the master secret K
  - Rc
  - Rs
- Values to be generated
  - client authentication and encryption keys
  - server authentication and encryption keys
  - client encryption IV
  - server encryption IV

Alert Protocol Examples

- Type 1: **`Fatal_Alert`**
  - ex.: **`Unexpected_Message`**, **`Bad_MAC`**, etc.
  - connection is immediately terminated
- Type 2: **`Warning`**
  - ex.: **`No_Certificate`**, **`Close_Notify`**

Summary

1. SSL is the de facto authentication/encryption protocol standard for HTTP
   – becoming popular for many other protocols as well
2. Allows negotiation of cryptographic methods and parameters