Computer and Network Security: Case Study-SSL/TLS

Kameswari Chebrolu

Securing Internet Layers

- Security can be applied at different layers of the protocol stack
- Security protocols at different layers
  - WEP/WPA at link layer, (802.11) IPsec at network layer; SSL/TLS at transport; PGP/SSH at application layer
- Case-Study: SSL/TLS at Transport layer
  - SSL: Secure Socket Layer
  - TLS: Transport Layer Security

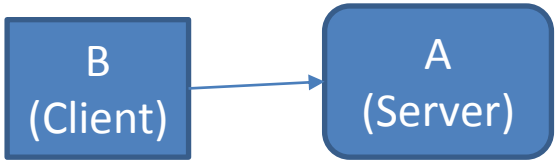
Securing Transport Layer

- SSL: originated at Netscape
  - V1 (internal use), v2 (buggy) , v3 (popular but not secure; 2014 poodle attack)
- TSL: standardized by IETF
  - Not compatible with SSL
  - V1.0, v1.1. v1.2 (v1.3 in pipeline)
  - Evolution accounts for security fixes, newer protocols, removing support for weak protocols
- Henceforth will use SSL/TSL interchangeably

What is SSL?

- Cryptographic protocol that authenticates a server to a client
- Optionally can also
  - Authenticate client to the server
  - Provide confidentiality and integrity
- Runs on top of TCP to provide a secure channel to application layer protocols
  - Web browsing (HTTP), Email (SMTP/IMAP/POP); VOIP
  - E.g. https://www.cse.iitb.ac.in

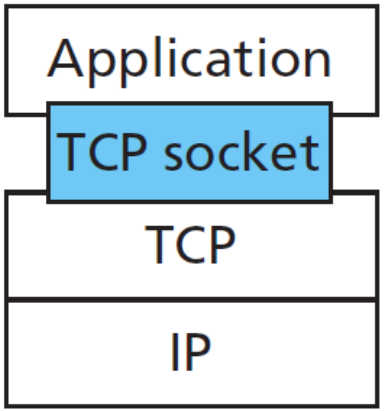
Recap: Need for it?



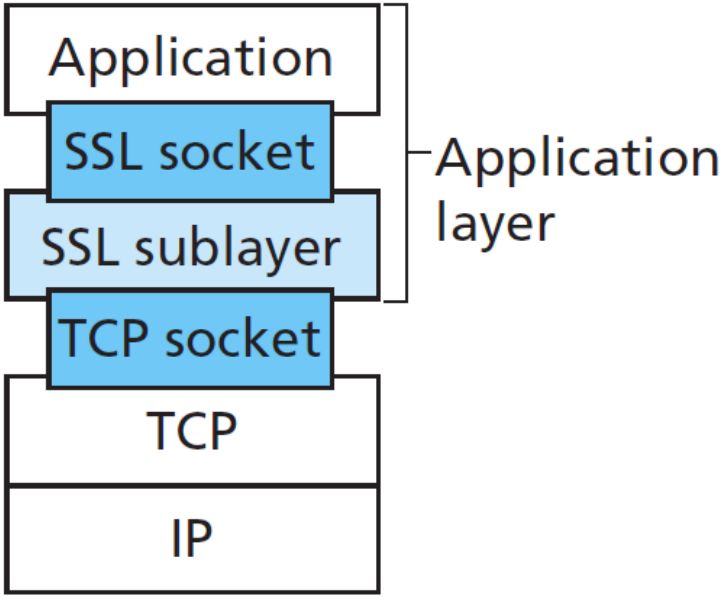
- Players: Bob: client; Alice: e-commerce website; Mallory: Malicious attacker
- Confidentiality: Prevents Mallory from getting Bob's bank/credit card info
- Integrity: Prevent Mallory from modifying Bob's order (1 TV to 10 TVs)
- Authentication: Ensures Bob is talking with real Alice and not Mallory pretending to be Alice
  - Else Mallory can steal Bob's personal details

Implementation

- No kernel (OS) level changes
- Applications need to use SSL API
- Why TCP?
  TCP will handle losses → SSL is simpler



TCP API



TCP enhanced with SSL

Steps

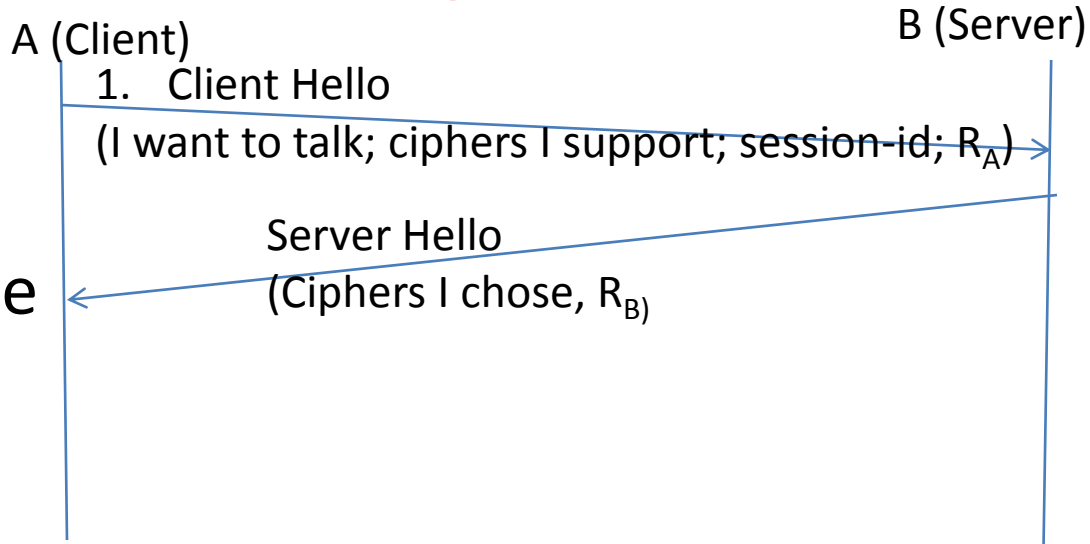
- Handshake + Key Derivation
- Data Transfer
- Alerts/Connection Closure

- **Focus: Creating a new session**

Handshake: Step-1

TCP 3-way handshake

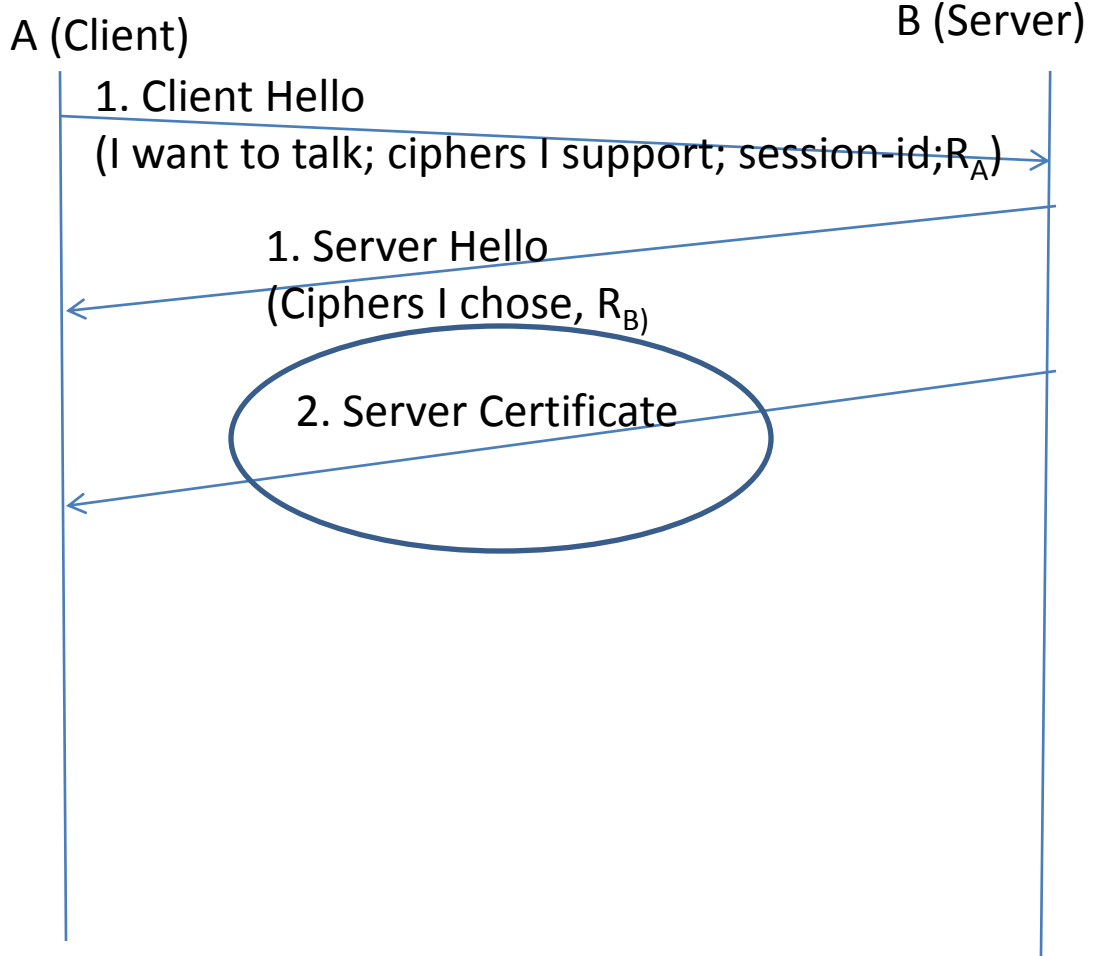
Precedes the first message



1. Client sends list of ciphers it supports; session id 0; nonce R_A
   - In case of session reuse; session id set to previous session's
2. Server sends ciphers it chose and a server nonce R_B
   - E.g: AES for symmetric key, RSA for public key, HMAC for MAC

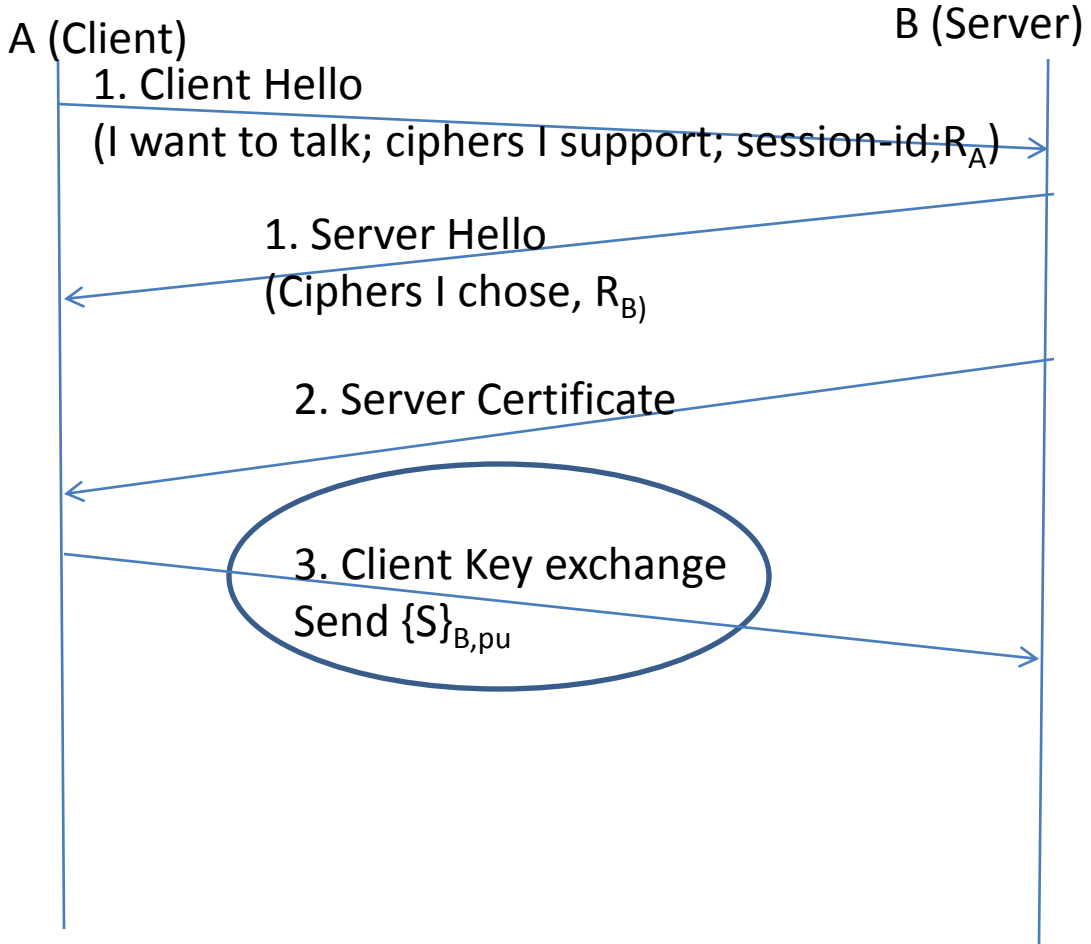
Handshake: Step-2

- Server sends its certificate which is verified by the client
- Server authenticated?
  - NO



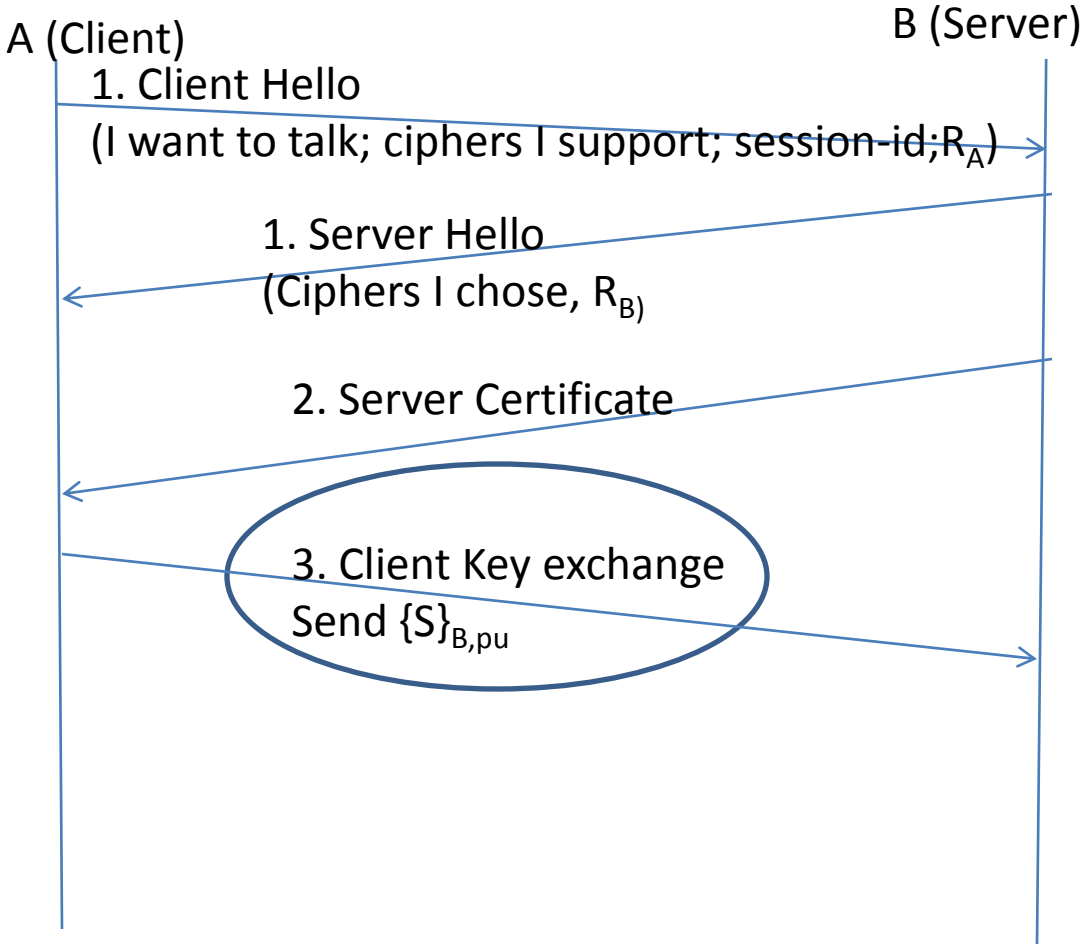
Handshake: Step-3

- Client chooses a random number S (pre-master secret key), encrypts it with server's public key and sends to server
- Client and server computer master key K = f (S, R_A, R_B)
  - f is a HMAC style hash function



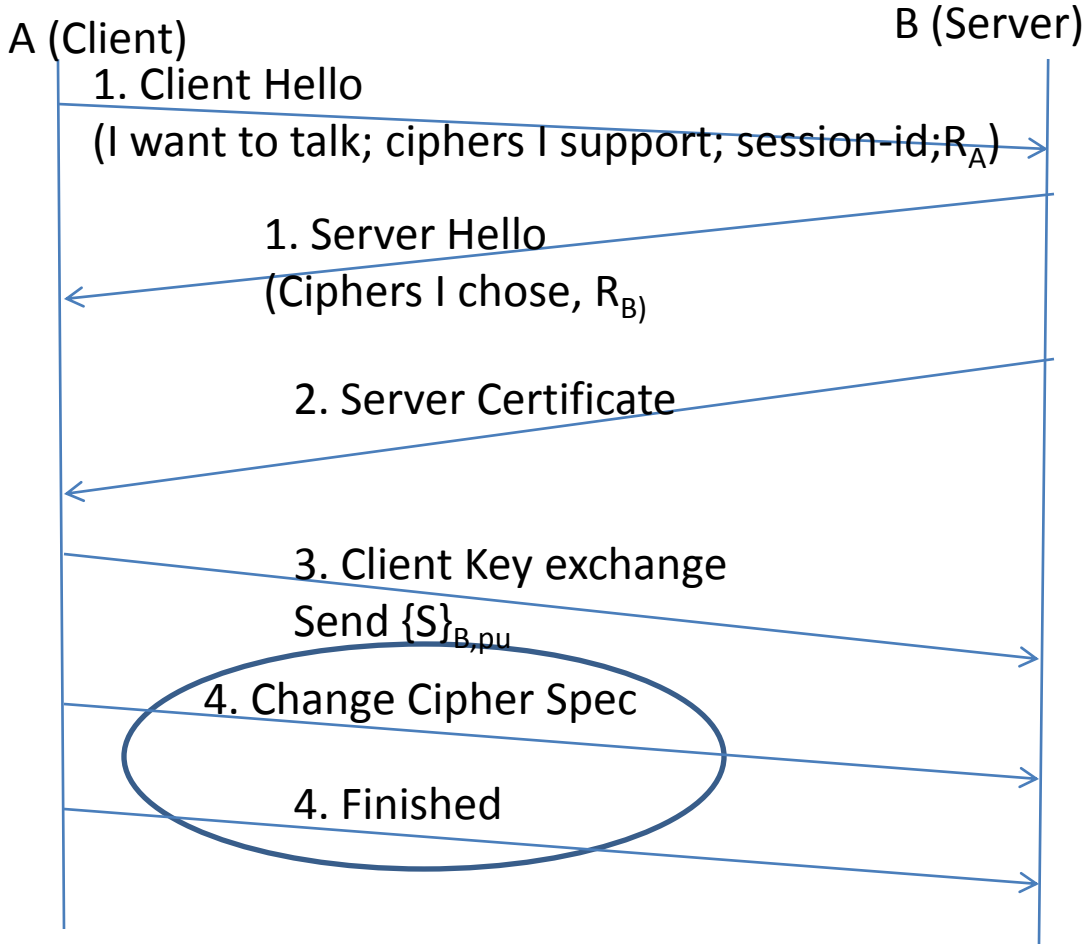
Handshake: Step-3

- Based on master secret key K, six secret keys are derived
  - Two Initialization vectors for encryption (C to S and S to C)
  - Two secret keys for encryption (C to S and S to C)
  - Two secret keys for MAC (C to S and S to C)
- Server authenticated?
  - NO



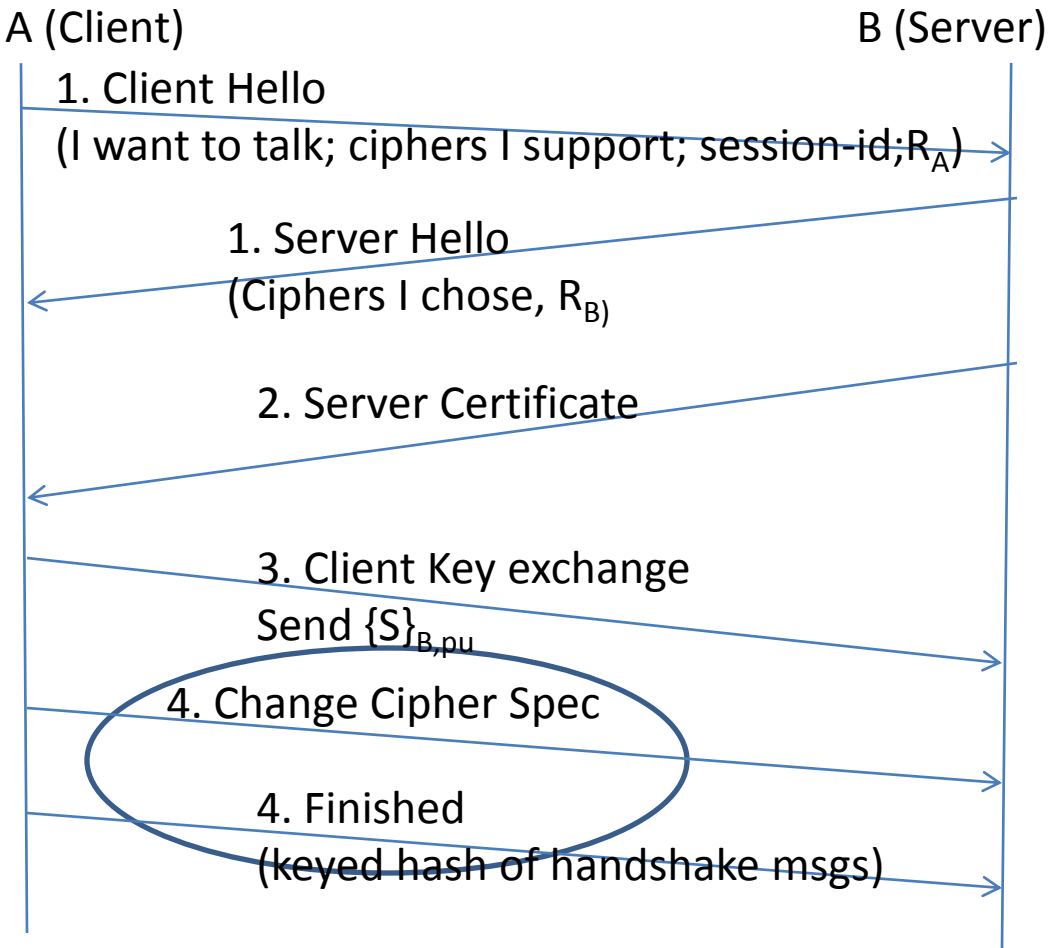
Handshake: Step-4

- Client sends a ‘change cipher spec’ message
  - Now on (i.e. everything post this) will be encrypted/integrity protected with chosen ciphers and derive keys



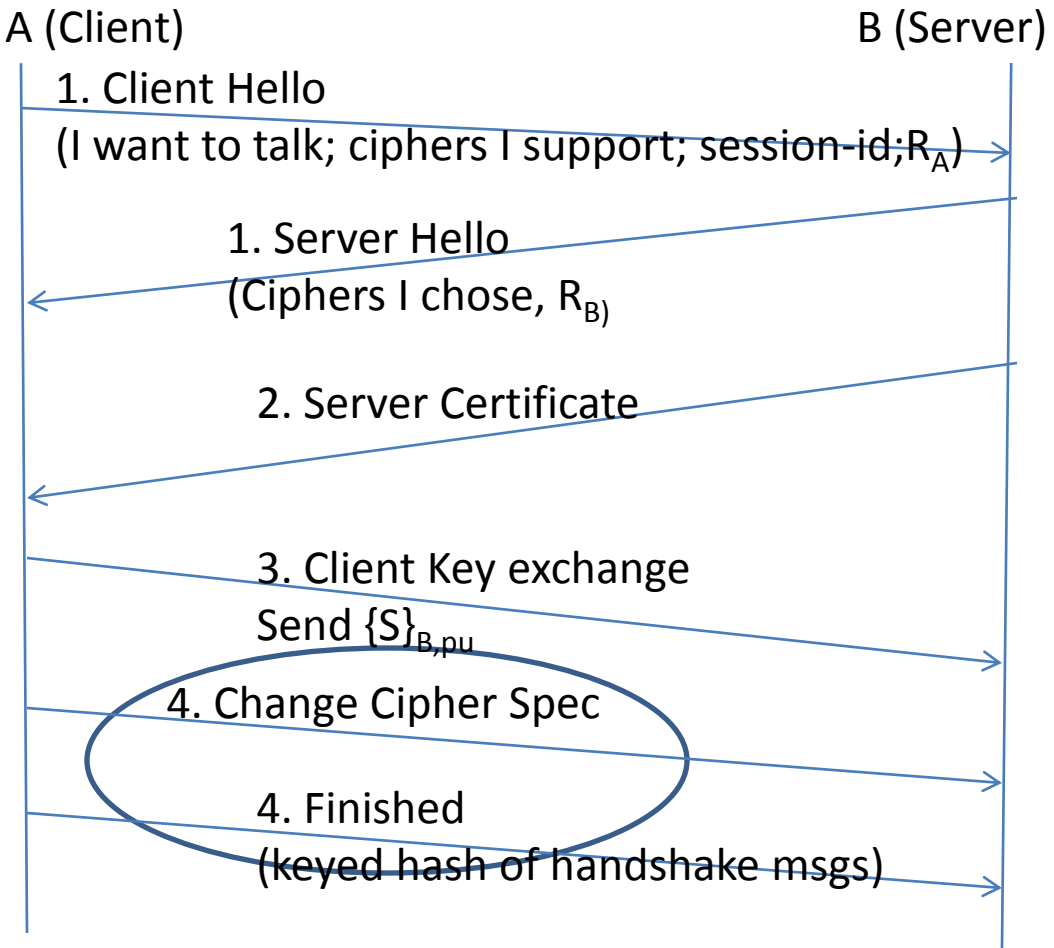
Handshake: Step-4

- Finished message includes a HMAC style hash of (master secret + all handshake messages exchanged so far + const)
  - Proves client knows the master key and no tampering of handshake messages



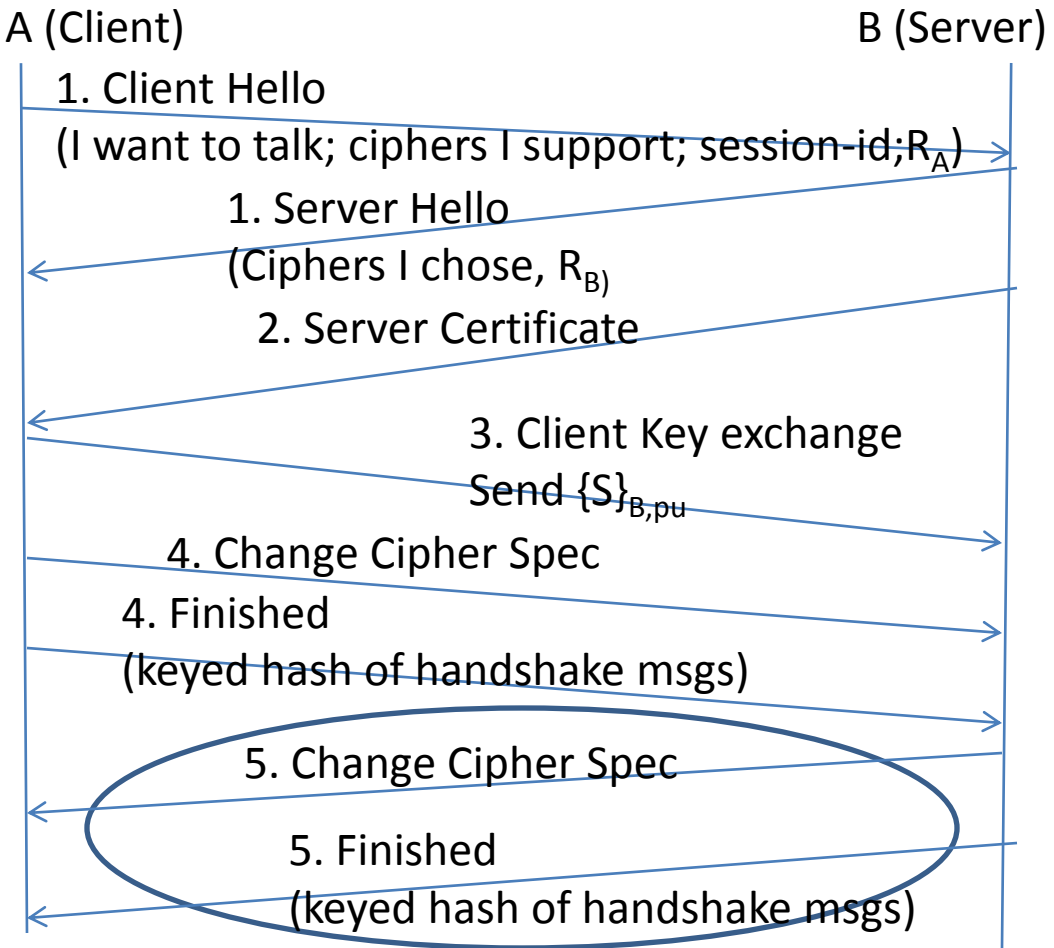
Handshake: Step-4

- Integrity via hash important to prevent tampering of messages
  - E.g. change AES to DES in chosen ciphers (step 1)
  - SSLv2 did not have this message



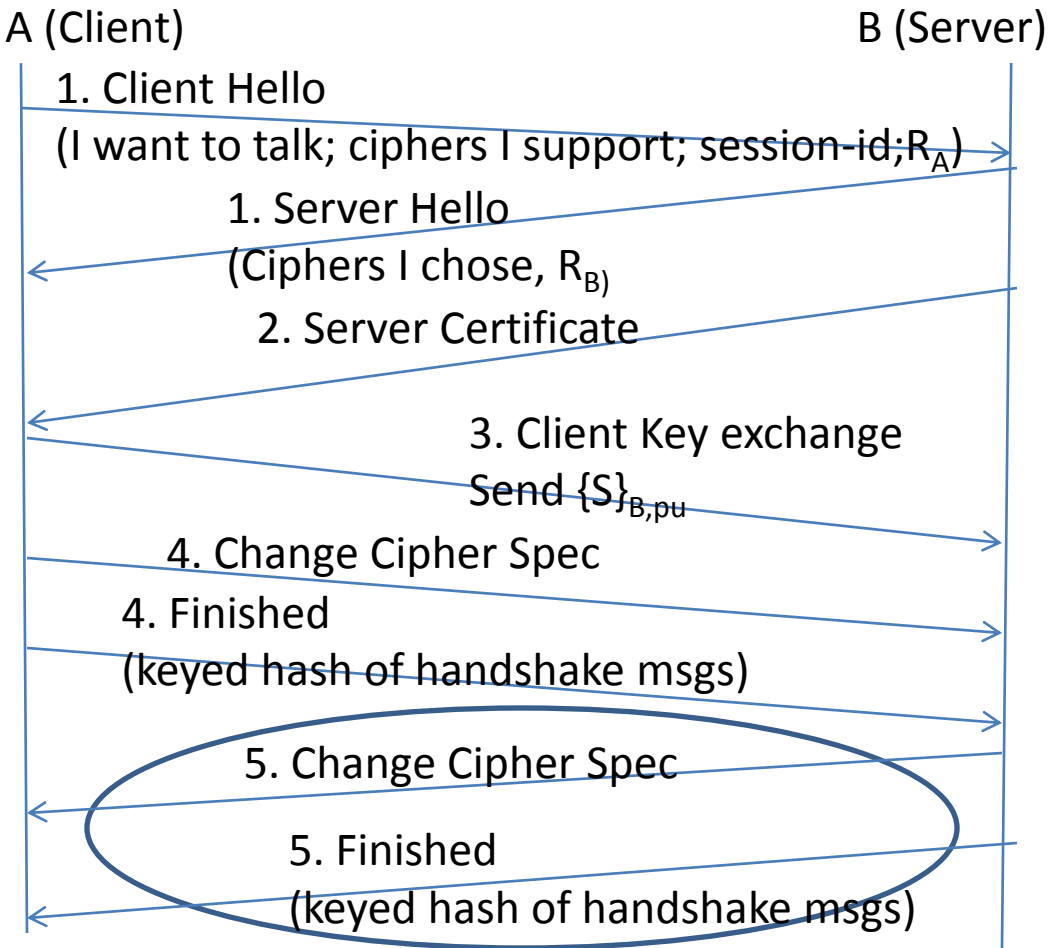
Handshake: Step-5

- Server also confirms change of cipher spec
  - Now on chosen ciphers and derive keys will be used
- Verifies computation of keyed hash



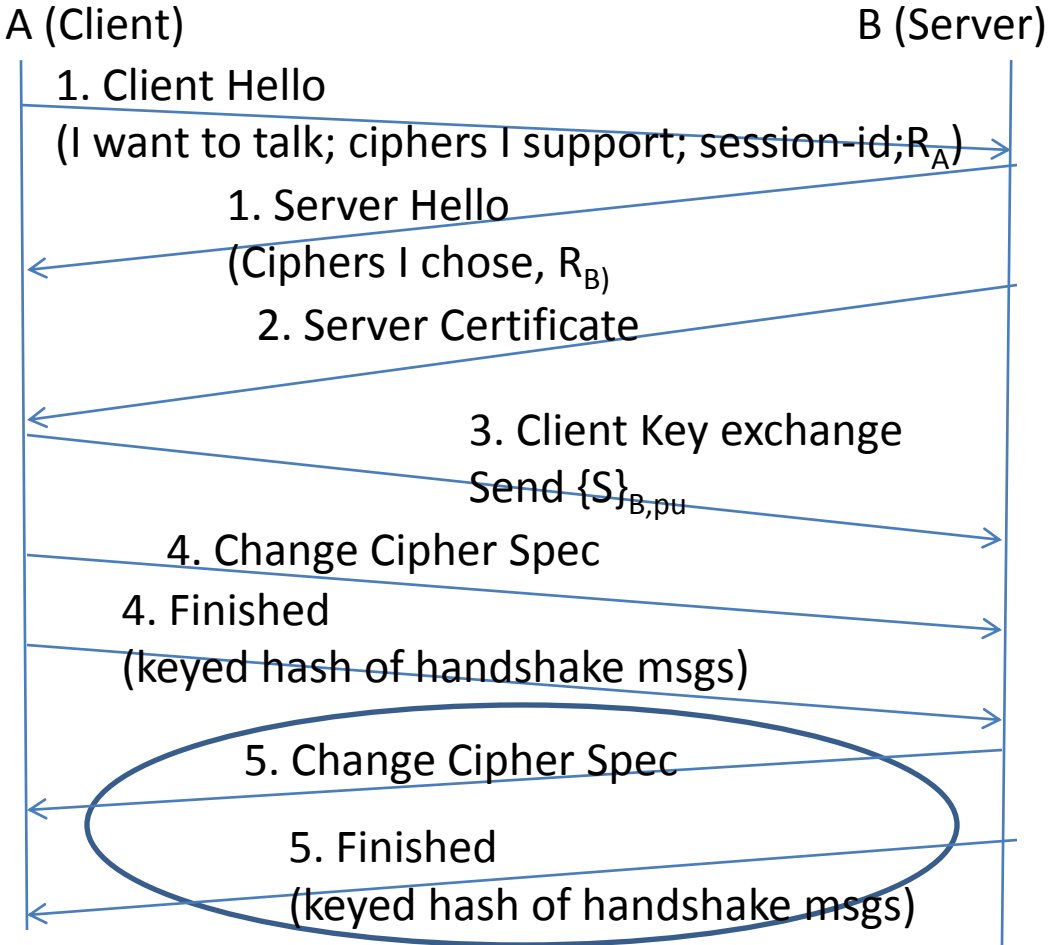
Handshake: Step-5

- Server sends its own hash of (master secret + all handshake messages exchanged so far + const) in finished message
- Client verifies the keyed hash from server.



Handshake: Step-5

- Authentication Complete after step-5?
  - Yes
- Data transfer begins
  - Data protected by keys derived from master key



Data Transfer

- Alice/Bob have all necessary keys (derived secret keys) for encryption and integrity
- SSL encrypt app data on fly and pass to TCP?
  - Where to append MAC?
- Break data and place in records
- Append MAC to record
- Encrypt (record+MAC)

Record format

Is this Secure?

- Subject to MITM attack
- Assume each TCP segment contains one record
- Mallory can reverse order of TCP segments (and modify TCP checksum) sent by Alice
- TCP will pass the two records to Bob's SSL; Bob's SSL integrity check is passed
- **Application receives data out of order**

Fix

- Maintain a sequence number 0 and increment for each record sent
- No need to include sequence number in record but include in MAC calculation
- MAC = hash of (record, MAC key and seq.no)
- Any alternation of TCP packet → integrity fail at SSL

Steps

- Handshake + Key Derivation
- Data Transfer
- **Alerts/Connection Closure/Connection Resume**

Records

Record format

- Four types
  - Handshake
    - E.g. Client Hello, Certificate, Finished etc
  - Change cipher spec
  - Alerts
    - Some are warnings (continue or abort) and some are fatal (abort)
  - Application data

Example

Not the entire list

Alert Code	Alert Message	Description
0	close_notify	Notifies the recipient that the sender will not send any more messages on this connection.
10	unexpected_message	Received an inappropriate message This alert should never be observed in communication between proper implementations. This message is always fatal.
20	bad_record_mac	Received a record with an incorrect **MAC**. This message is always fatal.
21	decryption_failed	Decryption of a TLSCiphertext record is decrypted in an invalid way: either it was not an even multiple of the block length or its padding values, when checked, were not correct. This message is always fatal.
22	record_overflow	Received a TLSCiphertext record which had a length more than **2^14+2048** bytes, or a record decrypted to a TLSCompressed record with more than **2^14+1024 bytes**. This message is always fatal.
30	decompression_failure	Received improper input, such as data that would expand to excessive length, from the decompression function. This message is always fatal.
40	handshake_failure	Indicates that the sender was unable to negotiate an acceptable set of security parameters given the options available. This is a fatal error.
42	bad_certificate	There is a problem with the certificate, for example, a certificate is corrupt, or a certificate contains signatures that cannot be verified.

Connection Closure

- How to end SSL connection? Send TCP FIN?
- Truncation attack (fixed in SSL v3)
  - Mallory can send the TCP FIN and end an SSL session in middle
- Fix: Handled by Alert protocol (close notify)

Sessions and Connections

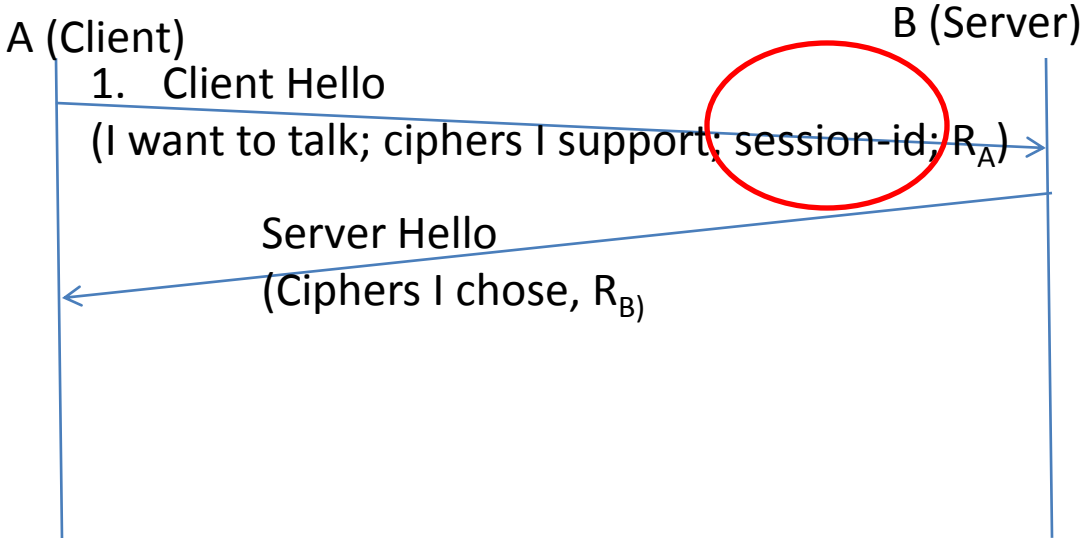
- Saw how to open a new session and close a session
- Session can have many connections (TCP)
  - E.g. Open one TCP connection for each object in the web page
- Should each connection have a new pre-master secret?
  - Very expensive operation (decryption of pre-master key)
- New connections: Create new master key but
  - Reuse same pre-master key and
  - Choose two fresh nonces from client and server
  - E.g. New connection setup 3ms vs new session setup 45ms (Apostolopus et.al)

Details

- Session State: pre-master key, negotiate ciphers and session ID
- Connection state: two nonces, master key, six derived keys and two sequence numbers (for each direction)

Session Resumption

- A can choose a new session-id, then public key portion (pre-master key) has to happen again
- Else, session-id is reused
  - Only nonces are exchanged



Summary

- SSL/TSL: a security protocol that puts together all the principles learnt
  - Challenge/response; integrity check; different derived keys, use of seq.nos, explicit messages for connection closure
- Also takes into account set-up overhead
  - Session resumption
- Open SSL: open source software that implements TLS/SSL