Web Protocols

Kameswari Chebrolu

Department of CSE, IIT Bombay

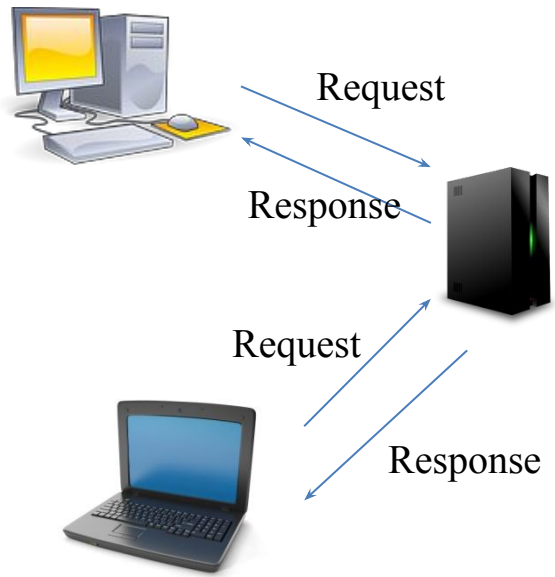
HTTP status ranges in a nutshell:
1xx: hold on
2xx: here you go
3xx: go away
4xx: you messed up
5xx: I messed up

Background: Overall Outline

- ~~What constitutes a webpage?~~
- ~~What goes on inside a Browser?~~
  - ~~What standard security mechanisms implemented?~~
- **How do client and server communicate?**
  - **HTTP/HTTPs protocol**
  - Session Management via cookies and tokens
- How does a web server process requests and generate responses?
  - Static vs Dynamic content

Hyper Text Transfer Protocol (HTTP)

- Protocol employed by Web applications
- Based on client-server model
  - Client (browser) requests web objects
  - Server responds with status code and requested object (if present)
- Operates over TCP, server port 80 (http), 443 (https)
- Stateless protocol i.e. no user information stored across requests

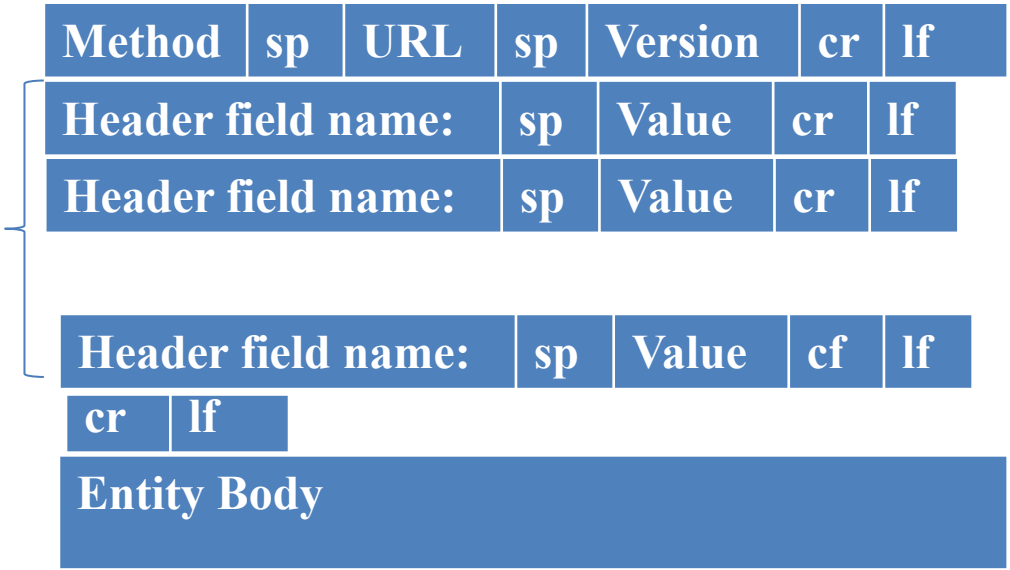


HTTP Message Format

- Two types of messages: Request and Response
- User types http://www.example.com/page.html in browser
- Request Message

```
GET /page.html HTTP/1.1
Host: www.example.com
User-agent: Mozilla/5.0
Connection: close
Accept-language: En
(blank line)
```

Header Lines



Methods

Method	Description
GET	Request for a web object
HEAD	Request for header fields (no body); Useful for debugging, get time of last modification etc
PUT	updating or creating a resource; considered idempotent.
POST	submitting data to be processed, often used when user fills forms
DELETE	Remove object
TRACE	Asks server to echo incoming request; Useful for debugging
CONNECT	Used to facilitate secure connection when using Proxy servers
OPTIONS	Query server about its properties or that of an object
PATCH	Similar to PUT except permits partial updating of an object

POST vs GET

- POST requests often involve changing state at server
  - E.g. creating user accounts, changing user details, invoking system commands, or executing arbitrary code
  - POST based attacks often associated with altering databases or executing commands/code!
- GET requests, on the other hand, just display data
  - GET based attacks often associated with exfiltrating data

Web Forms

- Allows users to upload content to web server in the form of name-value pairs
- Can be achieved both via GET and POST
  - GET (preferable for querying database) http://www.example.com/form.php?name=Chotu
  - POST: name, value is included in the request body
    - Preferable when there is a resulting action, e.g. sending email or writing to database
      - Accidental navigation that triggers GETs does not mess things up!

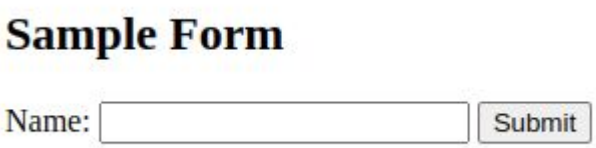
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sample Form</title>
</head>
<body>

    <h2>Sample Form</h2>

    <form action="/form.php" method="post">
        <!-- Text input for the user's name -->
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>

        <!-- Submit button to send the form data -->
        <input type="submit" value="Submit">
    </form>

</body>
</html>
```



```
POST /form.php HTTP/1.1
Host: example.com
Content-Type:
application/x-www-form-urlencoded
Content-Length: 10

name=Chotu
```

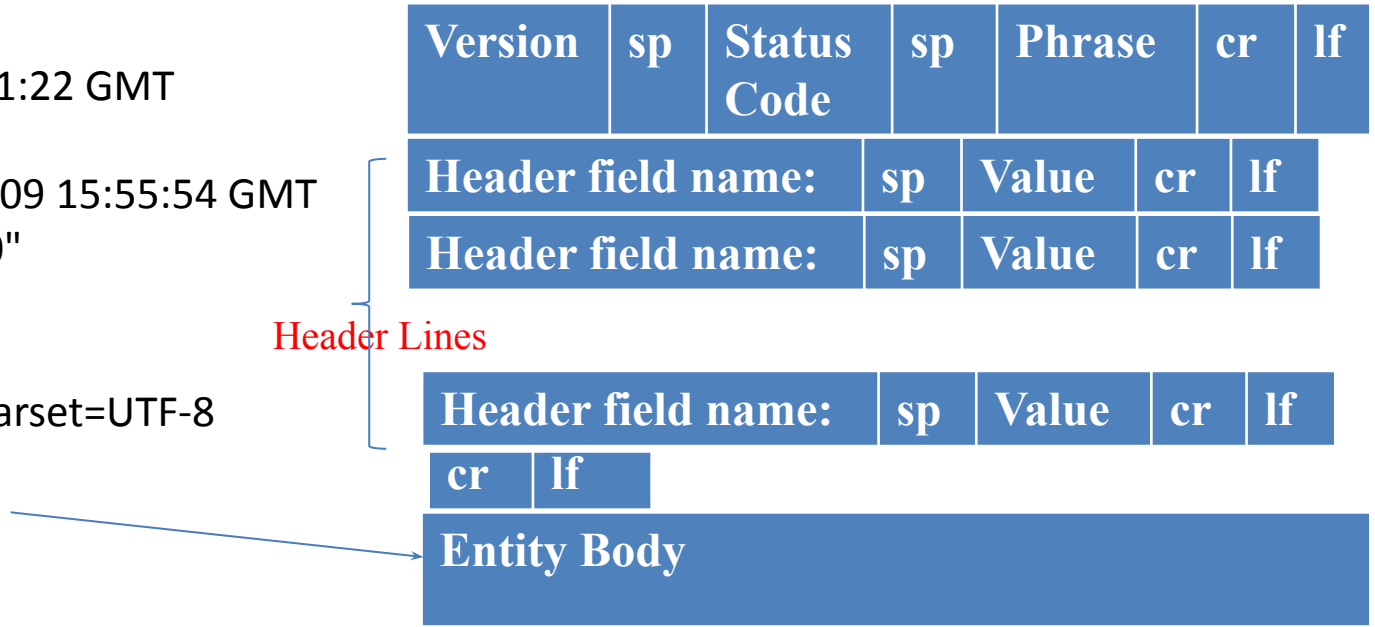
Explanation

- An action attribute set to "/form.php
  - Indicates the URL (endpoint) where the form data will be sent
- Method attribute set to "post," indicating use HTTP POST method
- Label and input elements create a text input field for user to enter name
- Input element with type="submit" creates a submit button
- When user enters name and clicks "Submit" button, a POST request is sent to the "/forms.php" URL

Response Message Format

HTTP/1.1 200 OK
Date: Sat, 02 Nov 2013 12:31:22 GMT
Server: Apache
Last-Modified: Fri, 02 Jan 2009 15:55:54 GMT
ETag: "424c001-fd-f9d8d680"
Accept-Ranges: bytes
Content-Length: 253
Content-Type: text/html; charset=UTF-8

data data data data data ...



Sample Status Code and Phrases

Status Code	Phrase	Description
200	OK	Request successful, information enclosed
301	Moved Permanently	Object moved; new url under Location:
400	Bad Request	Request could not be understood
404	Not Found	Requested object not found on server
503	Service Unavailable	Server is currently unavailable (overloaded)
505	HTTP Version not supported	Server does not support the HTTP version

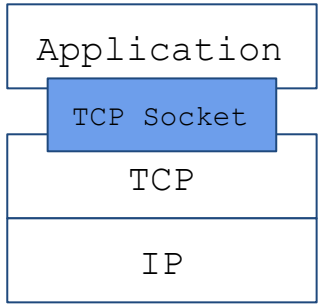
Hypertext Transfer Protocol Secure (HTTPs)

- A secure extension of HTTP
  - URLs that use HTTPS begin with "https://"
    - https://www.example.com
  - Browser connects over TCP to server listening on port 443 (as against 80 for http)
- Provides a secure communication channel between client and server
- Many modern web browsers encourage the use of HTTPs by displaying a padlock icon

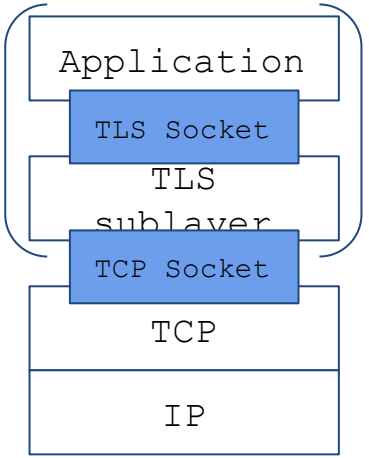
- Ensures the following:
  - Provides confidentiality and integrity of information transferred
    - No one can sniff or tamper the data exchanged
  - Provides Authentication i.e. ensures users are communicating with the intended website
    - Achieved via digital certificates issued by trusted Certificate Authorities (CAs)
  - Guards against various security threats, including man-in-the-middle attacks

Implementation

- No changes to HTTP Protocol
- Implemented at transport layer on top of TCP protocol
- Application needs to use SSL(or TLS) API instead of regular socket API



TCP API



Application Layer

TCP enhanced with TLS

Securing Transport Layer

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

What is SSL/TLS?

- Cryptographic protocol that authenticates a server to client
  - Can be used in email, VOIP applications as well!
  - Also provides confidentiality and integrity
- Optionally can also
  - Authenticate client to server
- Provides End-to-end security
  - Anything below application sees only encrypted content
    - No router, switch, wifi-AP etc can look at application content
  - Secure even in the presence of an attacker
    - Who owns network
      - Owning → controls routers, switches, Wi-Fi, DNS etc
    - Who can sniff, inject packets

Background

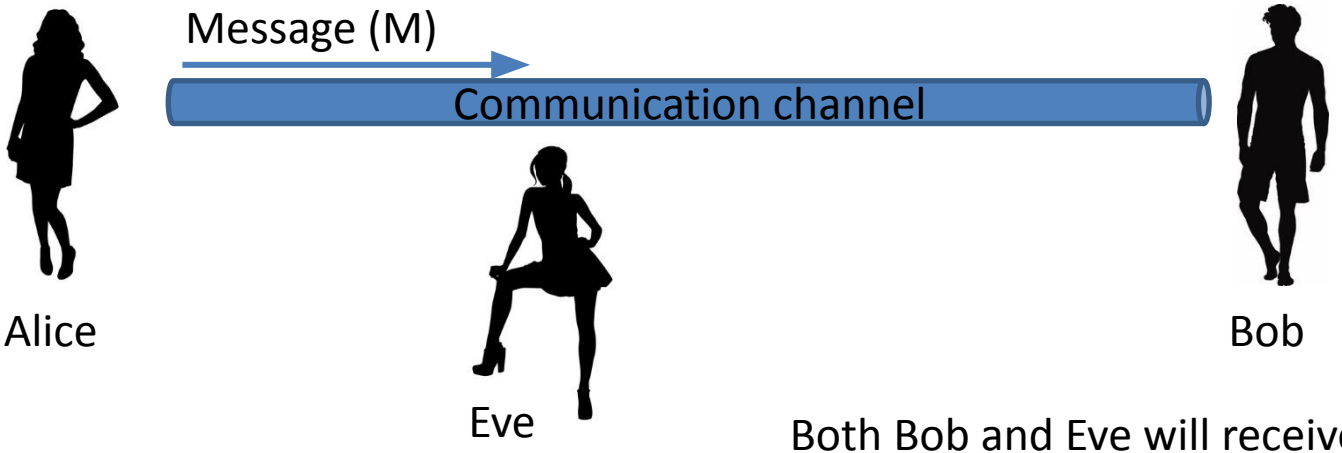
https://xkcd.com/1323/

Players

- Alice (A) and Bob (B) (lovers)
  - In our world: web browser and server
- Eve (E, eavesdropper) (jealous ex)
  - Passive attacker who can listen but not modify messages
- Mallory (M, malicious) or Trudy (T, intruder)
  - Active attacker who can modify, substitute, replay messages
- Goal: Alice and Bob want to communicate **securely** in presence of interlopers like E, M or T
  - Securely: Confidentiality, Integrity (Authentication included)

Confidentiality

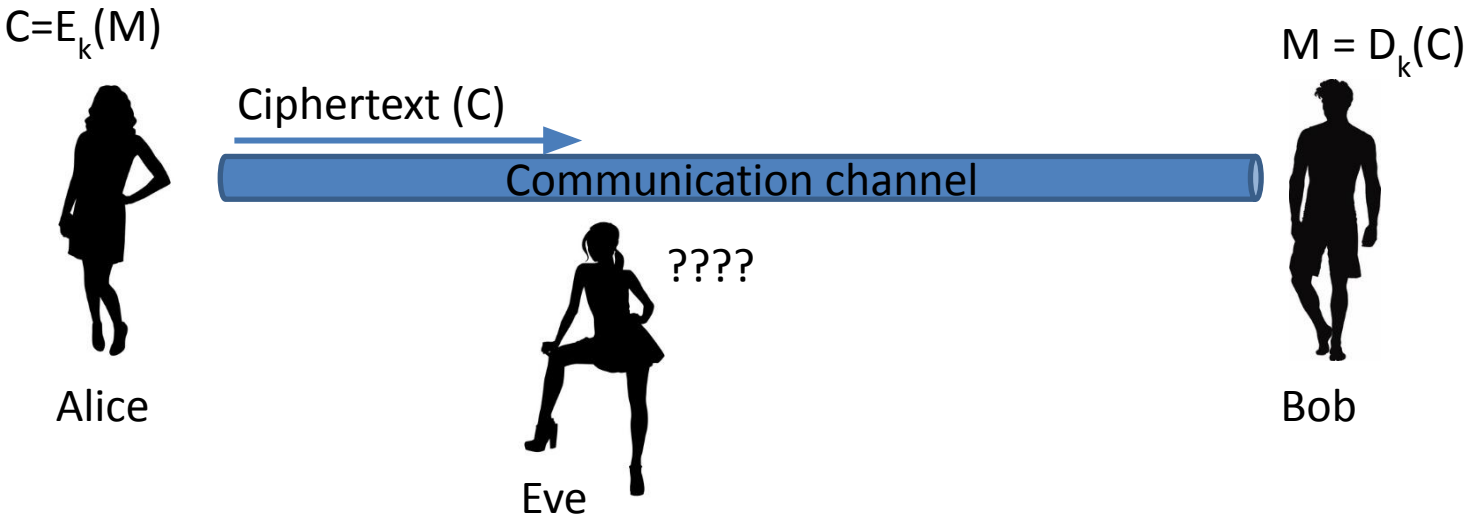
- Information not available or disclosed to unauthorized entities
- Solution: Encryption and Decryption



Both Bob and Eve will receive message

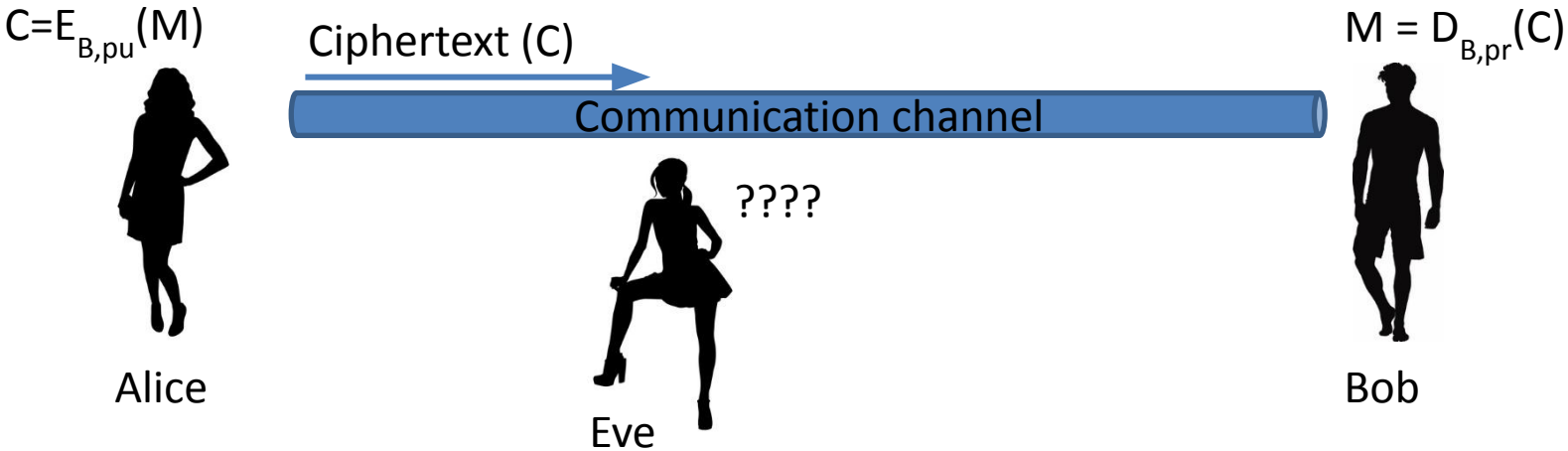
Symmetric Key

- Also called Shared key
  - Alice and Bob share a key k
  - Eve does not know the key but knows the encryption/decryption algorithm



Asymmetric Key

- Also called Public-key
  - Bob has two keys: one public and one secret
  - Bob's public key (B,pu) is public, both Eve and Alice have access to this
  - Bob keeps private key (B,pr) secret
  - Alice encrypts message with Bob's public key



Integrity and Authentication

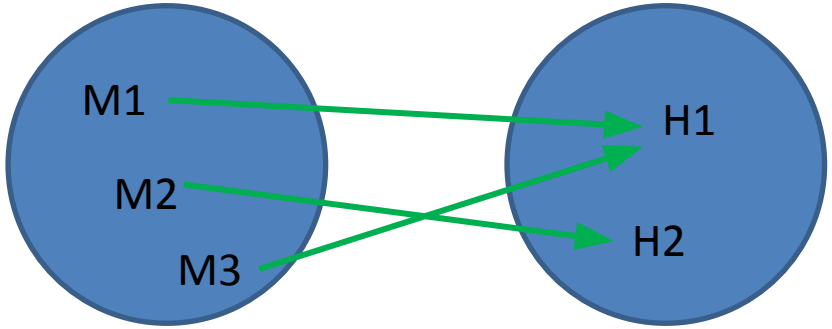
- Integrity (data): Has the data been modified in transit?
- Authentication (source): Am I talking with the right person?

Solutions

- Hashes/Message Digests: Provide Integrity (data)
- Message Authentication Codes (MACs): Provide both Integrity and Authentication
  - Based on Shared key algorithms
- Digital Signatures: Provide both Integrity and Authentication
  - Based on Asymmetric key algorithm

Hashes/Message Digest

- **One way** function used to verify message integrity
- Maps arbitrary length message (*) □ fixed length string (d)
  - h(M) hash of message M; similar to fingerprint
  - No key; algorithm is public!
    - SHA1/2/3; MD4/5
  - Typical values of d: 128, 160, 256, 512 bits

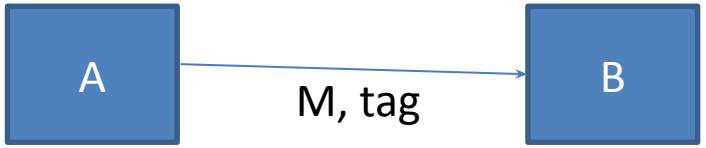


$$h : \{0,1\}^* \longrightarrow \{0,1\}^d$$

Collisions Possible

Message Authentication Codes (MACs)

- Also referred to as keyed hashes
- Provides both Integrity and Authentication
  - M: message, k: secret key shared between A and B
  - A sends message and tag (function of M and k)
  - B verifies received message with tag
    - Matches, accept (authentic + untampered)
    - No match, reject (tampered/unauthentic or corrupted)



tag = F(M,k)
(=MAC)

V(k,M,tag)
= 1 (accept)
= 0 (reject)

MAC vs Hash

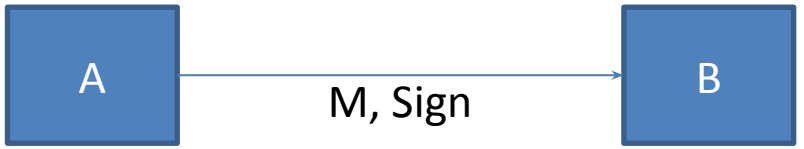
- A virus can modify a file and its hash also (recalculate) → cannot detect tampering
- Virus can modify file but cannot calculate new MAC since it does not know the secret key

MAC vs Digital Signatures

- Example: Software company releasing periodic patches; integrity of patches important
  - How many keys?
  - How to ensure trust? Bind document to author
- Digital Signatures (based on public key systems)
  - Scalable
  - Easy to verify identity
  - Disputes can be resolved by third parties

Digital Signatures

- M: message; A_pu: Public key; A_pr: private key
- A sends message and sign (function of A_pr and M)
- B verifies received message with sign using A_pu
  - Matches, accept (authentic + untampered)
  - No match, reject (tampered/unauthentic or corrupted)

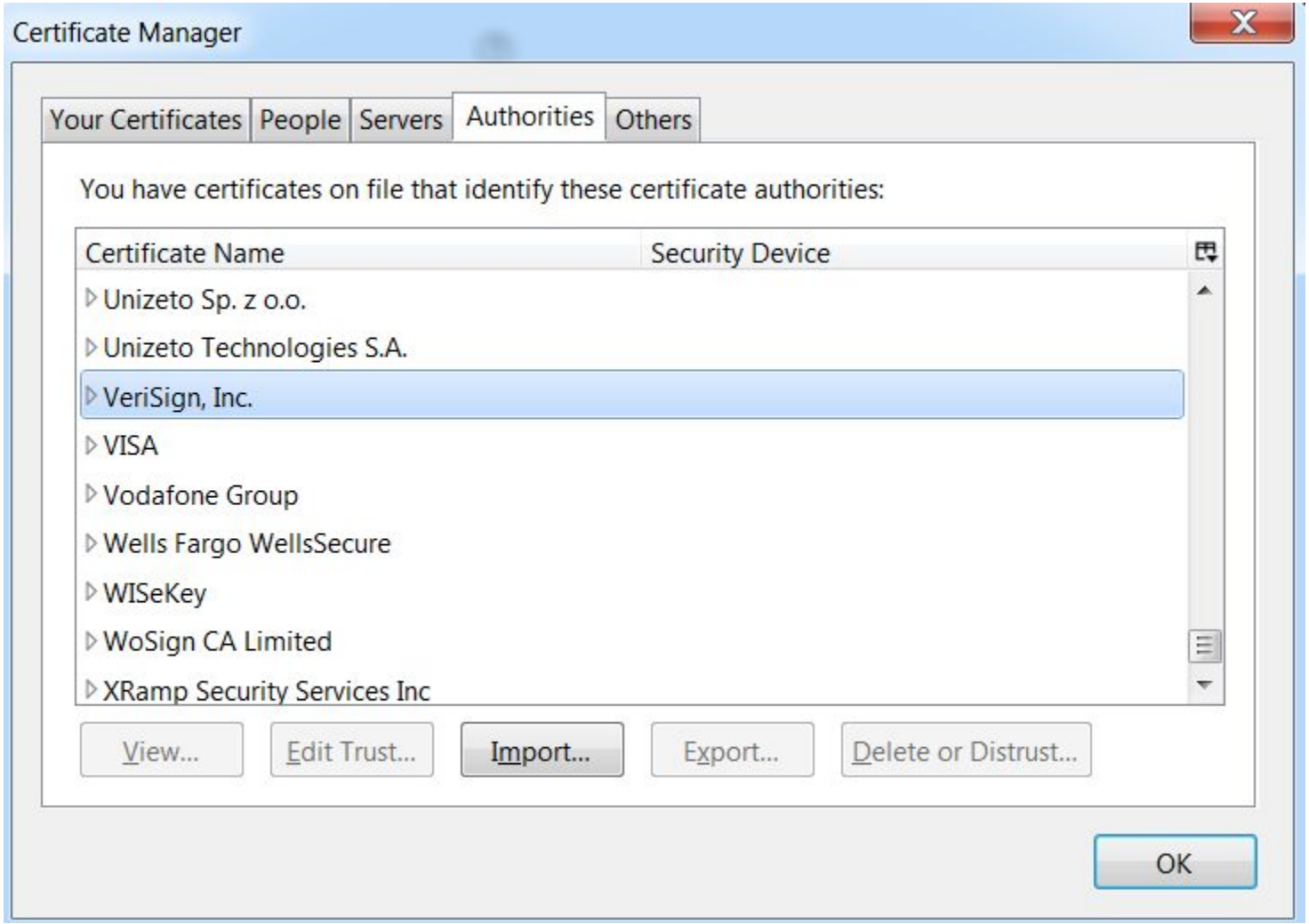


sign = F(A_pr,M)

V(A_pu,M,S)
= 1 (accept)
= 0 (reject)

Digital Certificates

- A trusted entity signs some “data” using its private key
  - Data? [public key, public key owner identifier, expiration time, serial number]
  - This signed document is called digital certificate
  - Trusted Entity is called Certificate Authority
- Example: VeriSign signs a digital certificate for IITB claiming www.iitb.ac.in’s public key is xyz
  - IITB can use this certificate to convince users visiting IITB website that the site is genuine
  - User (Browser i.e.) has Verisign’s public key (hardcoded) and can verify this certificate!

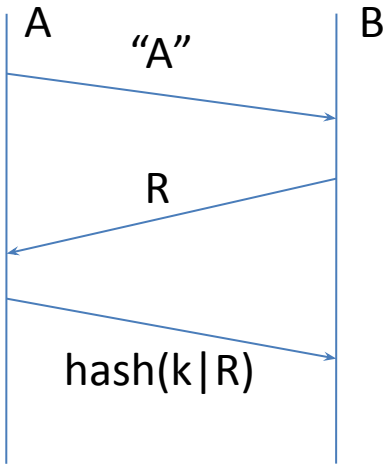

Certificate Manager
Your Certificates
People
Servers
Authorities
Others
You have certificates on file that identify these certificate authorities:
Certificate Name
Security Device
Unizeto Sp. z o.o.
Unizeto Technologies S.A.
VeriSign, Inc.
VISA
Vodafone Group
Wells Fargo WellsSecure
WISeKey
WoSign CA Limited
XRamp Security Services Inc
View...
Edit Trust...
Import...
Export...
Delete or Distrust...
OK

More Details

- www.iitb.ac.in uses http, say IITB wants to move to https
- IITB needs to buy a certificate from a CA
  - CA will sign a doc that links www.iitb.ac.in (domain) with IITB's public key
- When a user types https://www.iitb.ac.in
  - Browser asks IITB's website for a copy of its certificate
  - Browser verifies certificate using public key of CA (public key of CA hardcoded in browser)
  - Checks domain in certificate matches url typed in browser
  - Establishes secure connection to IITB web server using IITB's public key in the digital certificate

Challenge-response

- B wants to verify if it is indeed A it is talking with currently?
  - B throws a challenge in form of R (nonce, random number, e.g. 256 bits)
  - A has to produce a MAC on it with shared key
    - Attacker cannot produce it
    - B also knows the key and can verify



(a)

Using secret/shared key

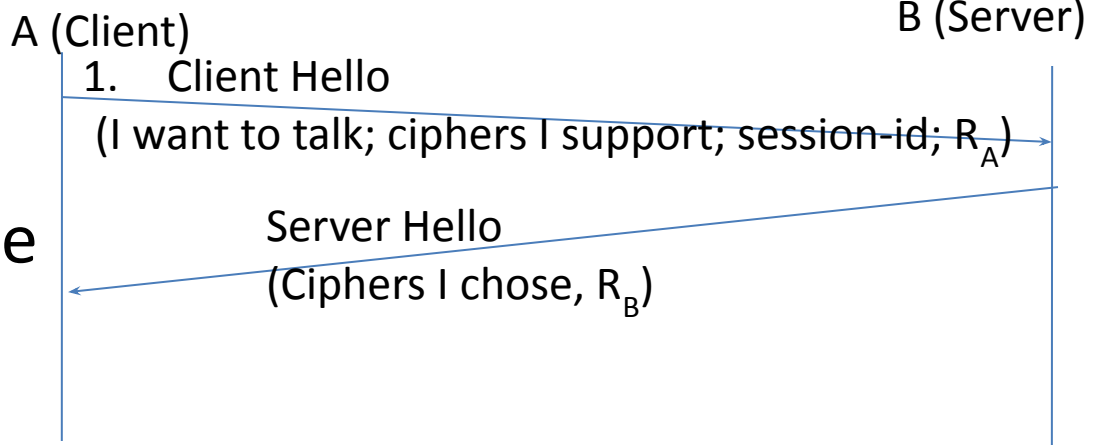
Pieces

- Symmetric key and asymmetric key based encryption and data integrity mechanisms
- Digital certificates that involve digital signatures
- Challenge Response Mechanisms

TLS Handshake: Step-1

TCP 3-way handshake

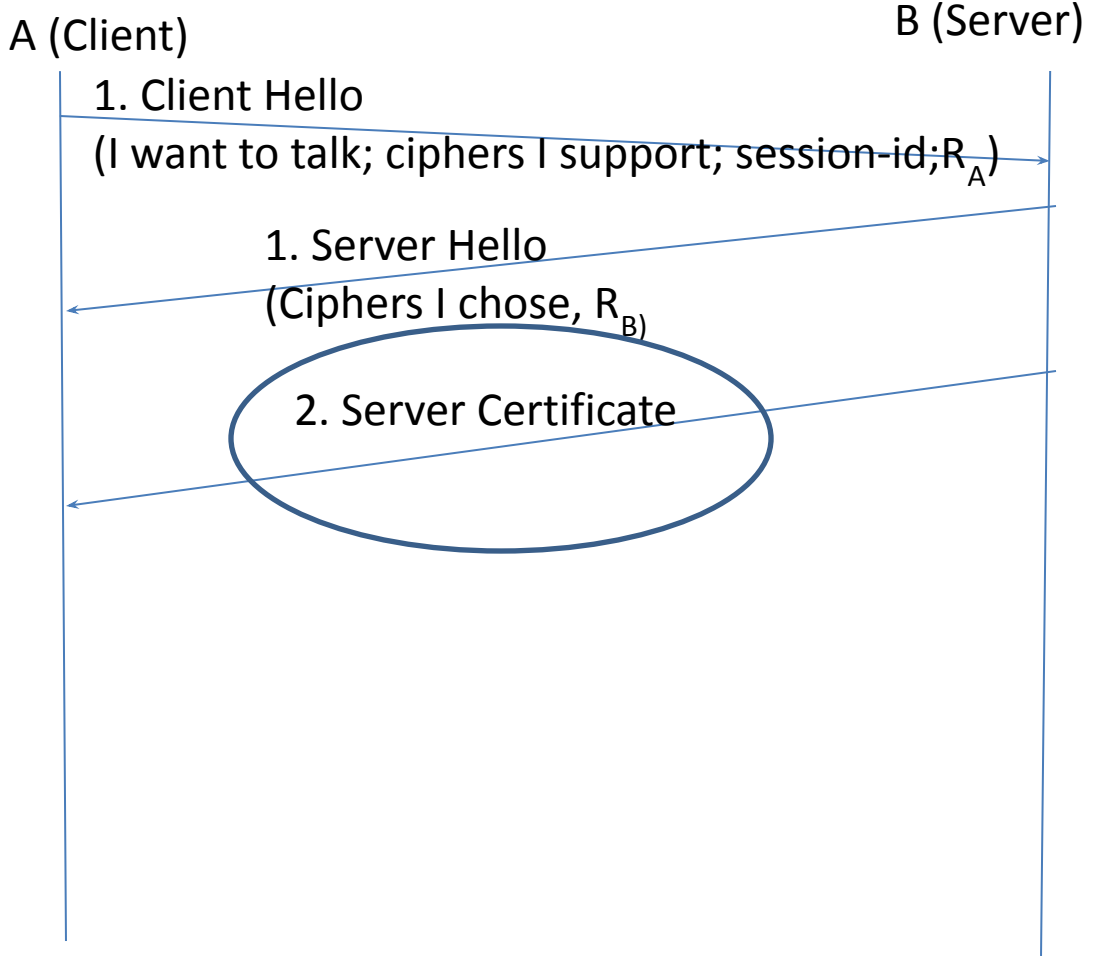
Precedes the first message



- Client sends version; list of ciphers (algorithms) it supports; session id 0; nonce R_A
  - In case of session reuse; session id set to previous session's
- Server sends ciphers it chose (highest version; strongest suite); nonce R_B
  - E.g: AES for symmetric key, RSA for public key, HMAC for MAC
- Both sent in plain text

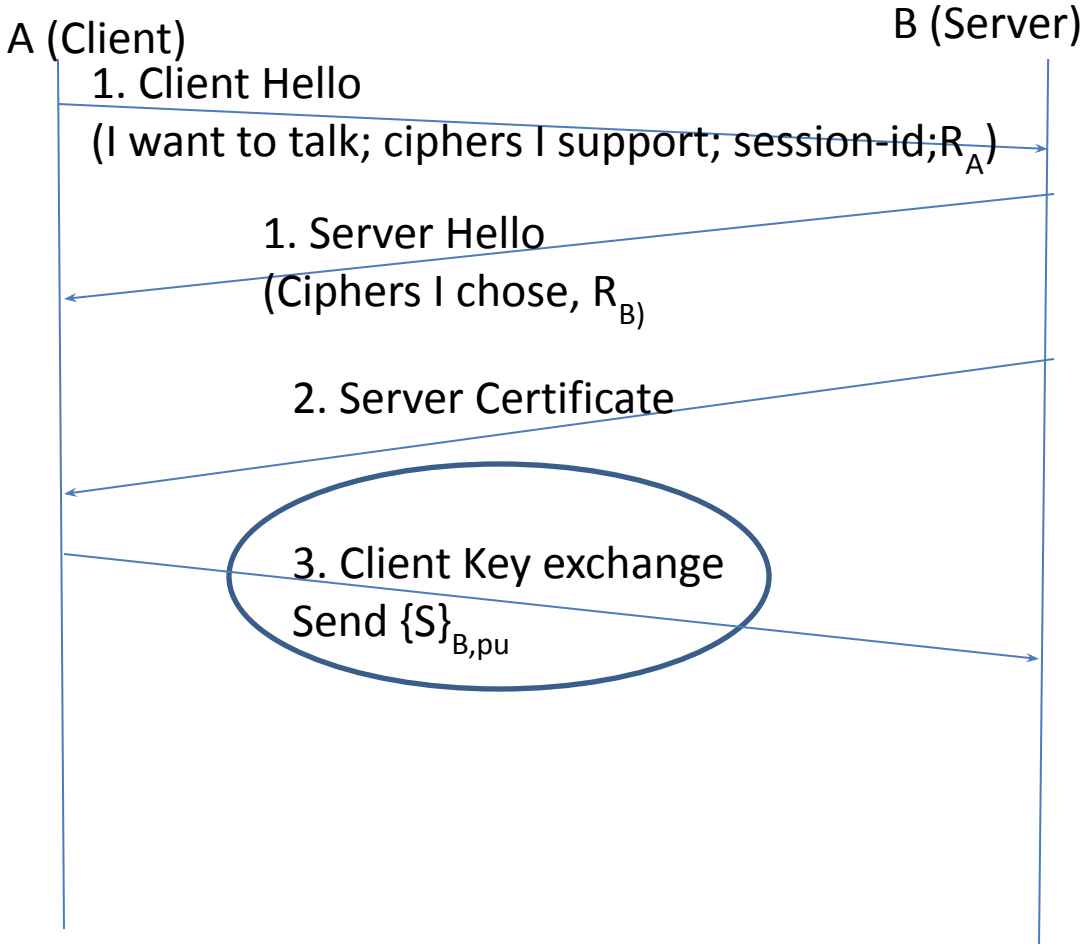
Handshake: Step-2

- Server sends its digital certificate which is verified by the client
  - Using the CA's public key

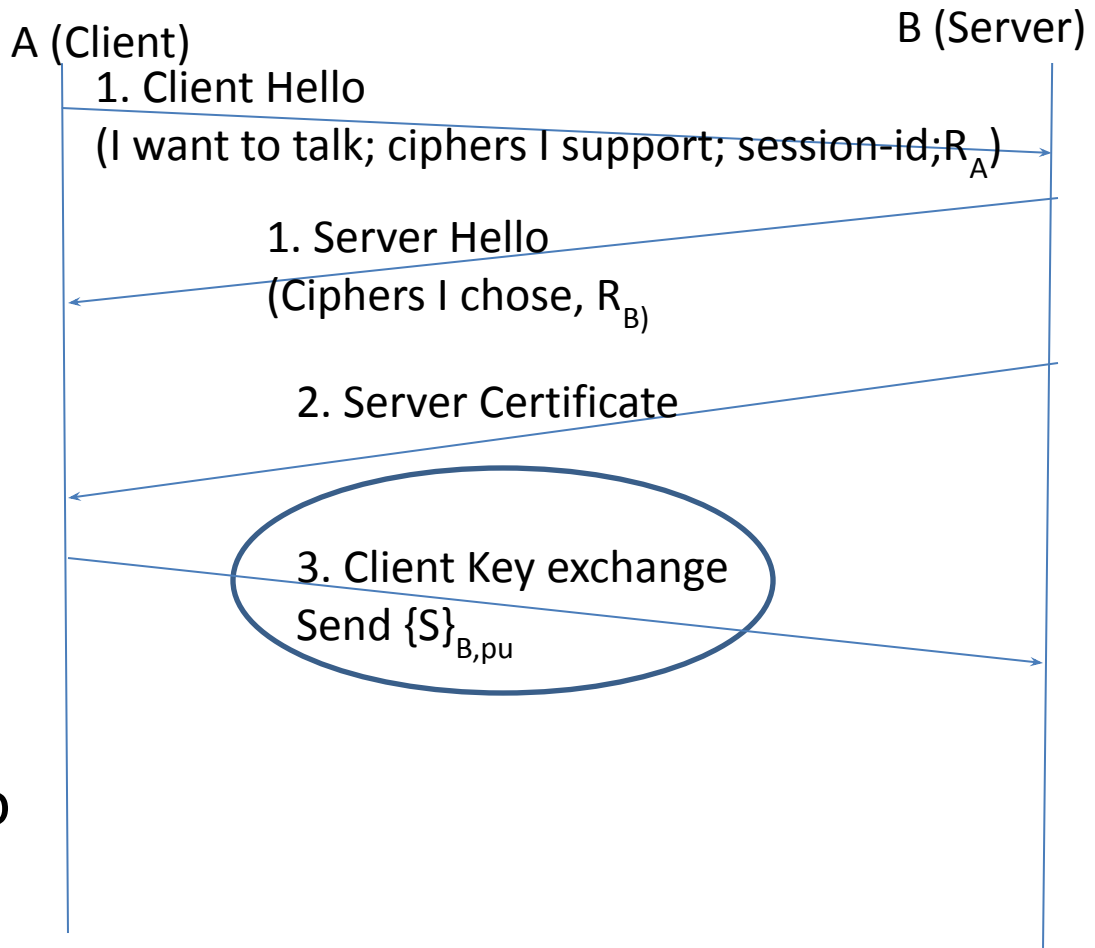


Handshake: Step-3

- 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

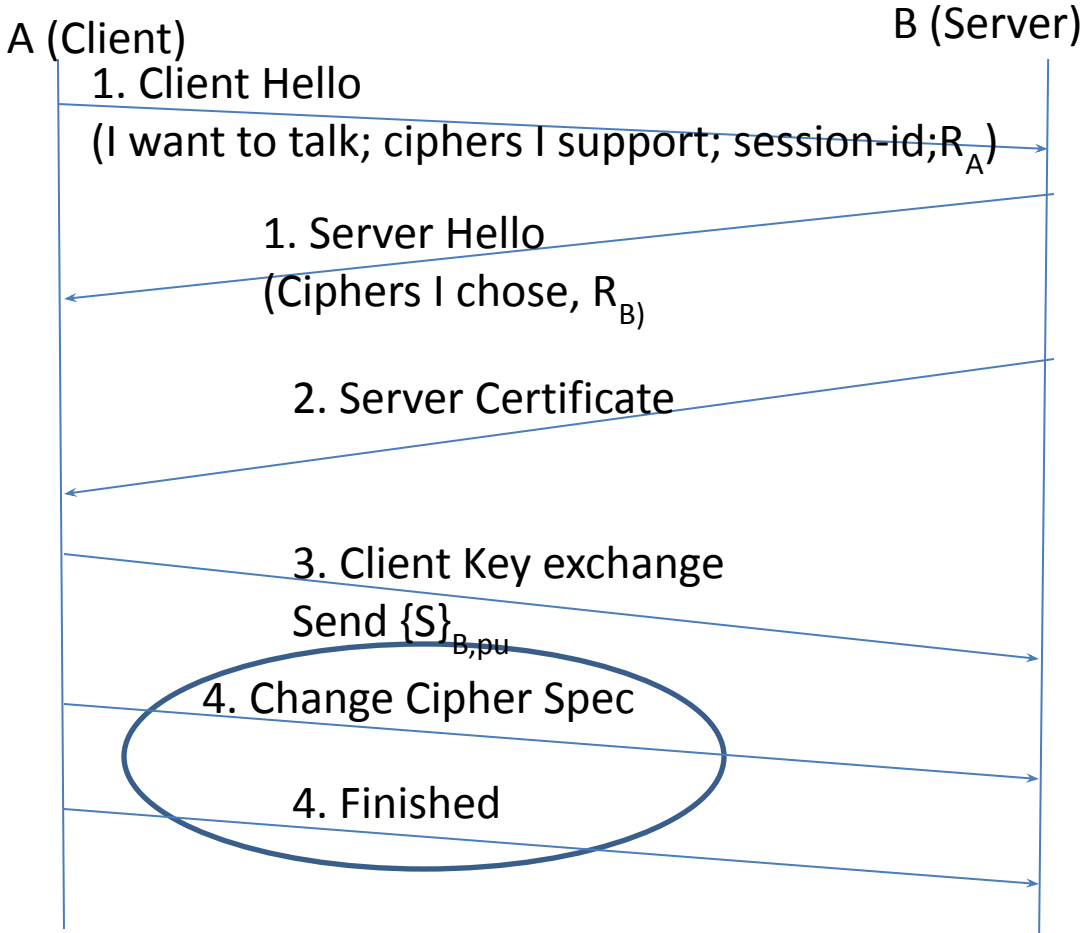


- Based on master secret key K, six secret keys are derived
  - Two Initialization vectors for encryption (C to S and S to C)
  - Two secret keys for encryption (C to S and S to C)
  - Two secret keys for MAC (C to S and S to C)

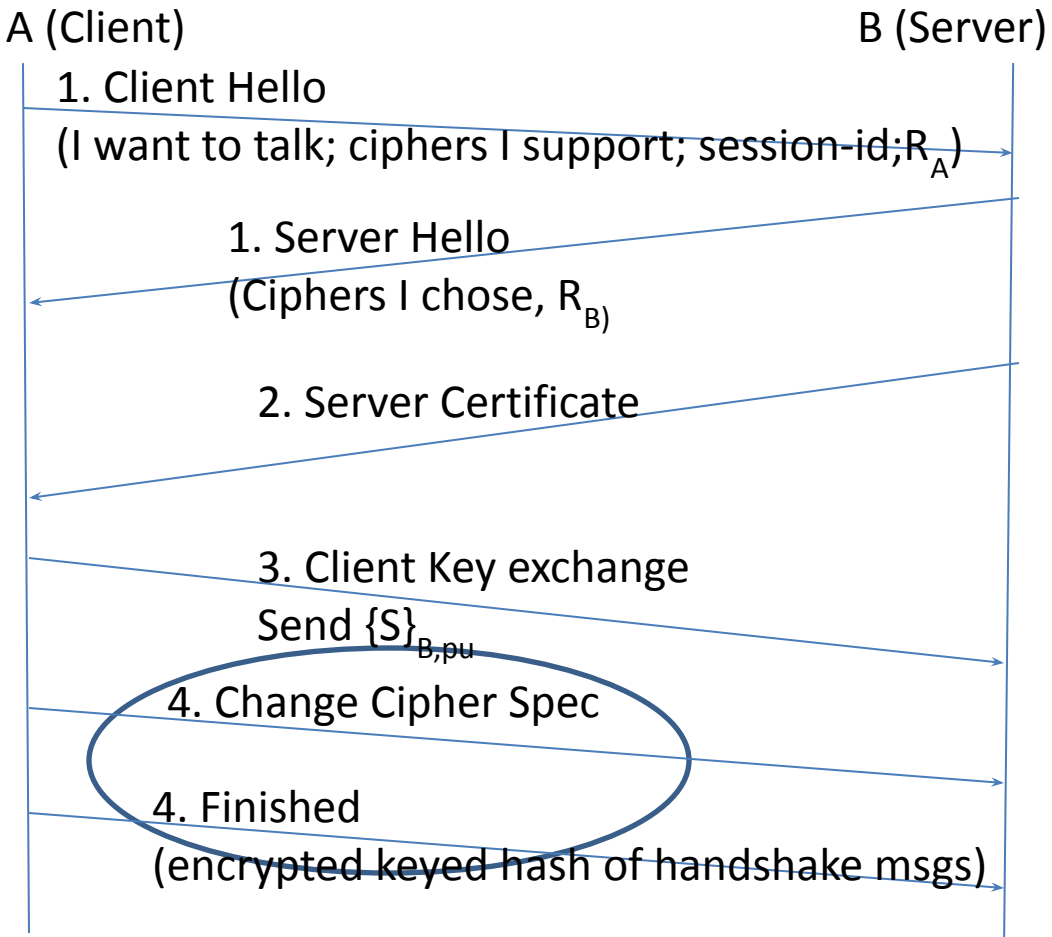


Handshake: Step-4

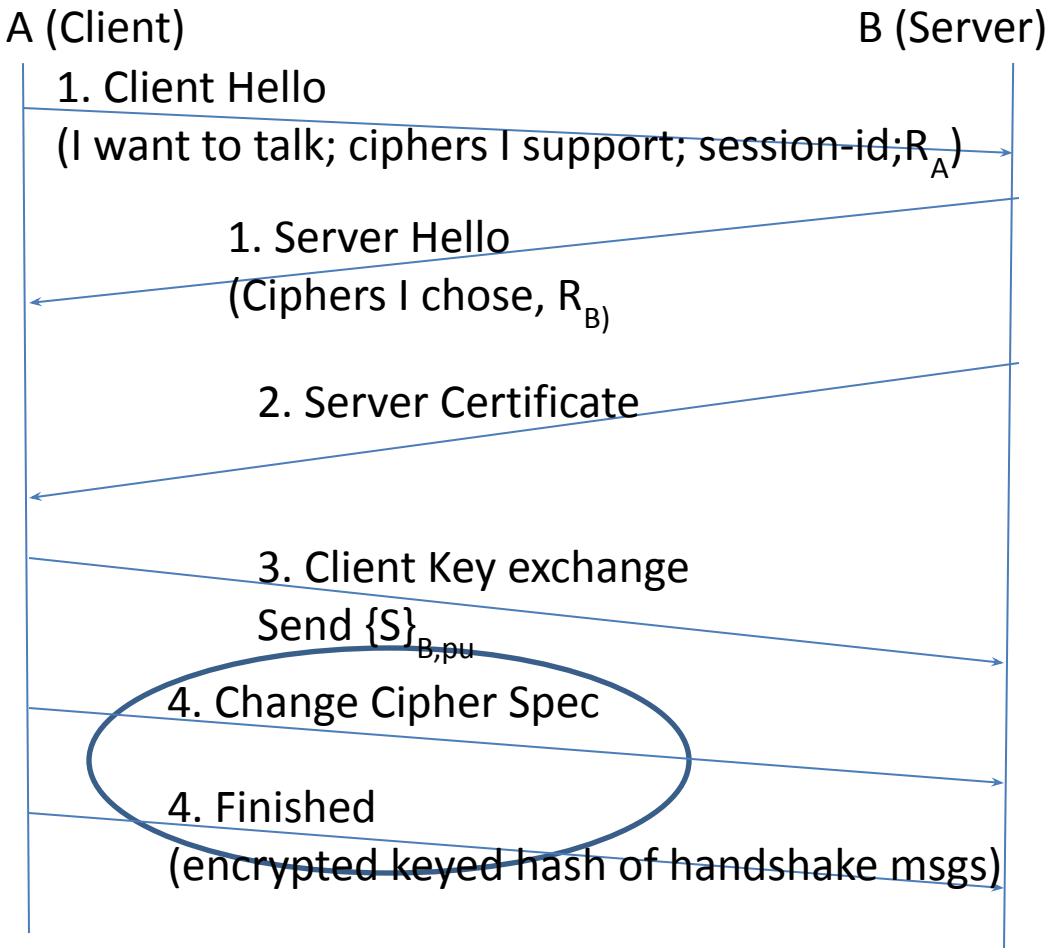
- Client sends a ‘change cipher spec’ message
  - Now on (i.e. everything post this) will be encrypted/integrity protected with chosen ciphers and derived keys



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

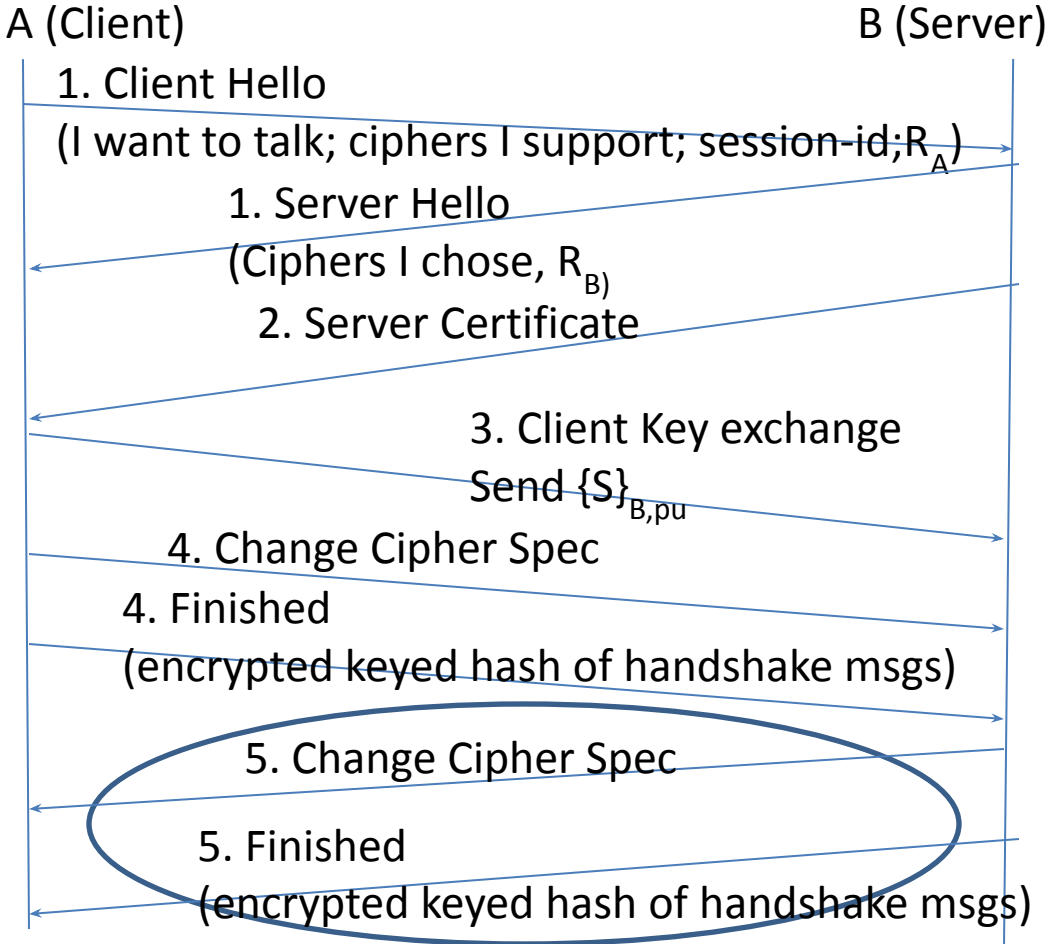


- Integrity via hash important to prevent tampering of messages

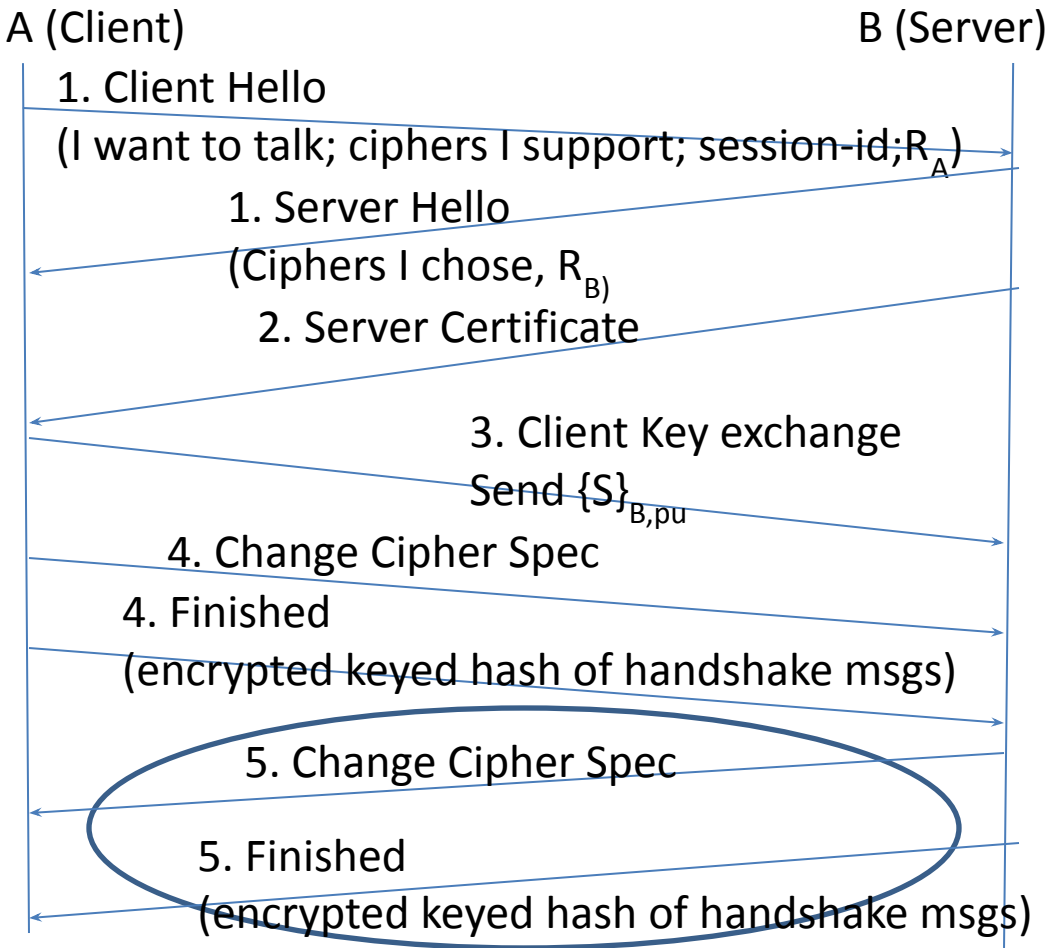


Handshake: Step-5

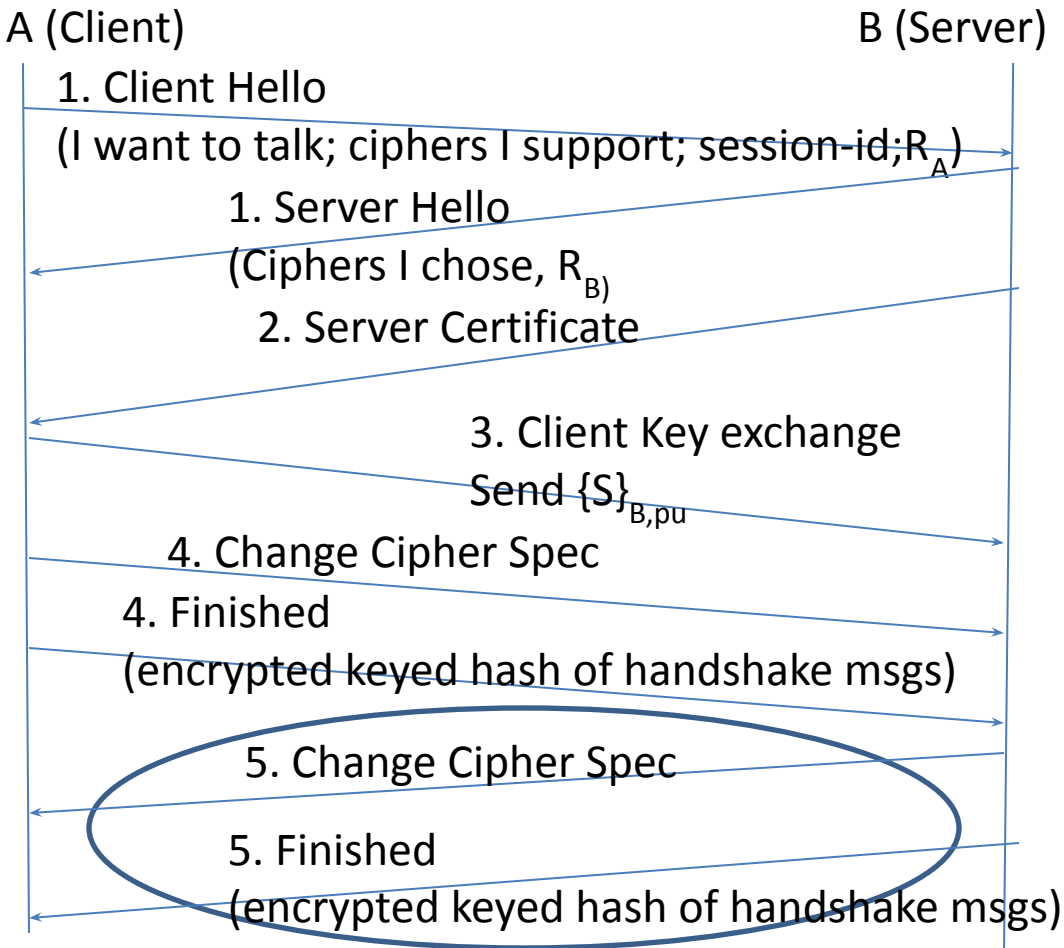
- Server also confirms change of cipher spec
  - Now on chosen ciphers and derive keys will be used
- Verifies computation of keyed hash



- 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



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



Client Authentication?

- So far, client authenticated the server
- How does server authenticate the client?
  - SSL/TLS does support public key based client authentication (this is optional)
    - Clients having public keys not very realistic
  - Common case: Client send password as part of the data (encrypted) that follows authentication

Data Transfer

- Alice/Bob have all necessary keys (derived secret keys) for encryption and integrity
- Break data and place in records
- Append MAC to record
- Encrypt (record+MAC)

Record format

HTTP/2.0

- Second major version developed to address limitations of HTTP/1.1
  - Standardized by the Internet Engineering Task Force (IETF) in May 2015
- Key features:
  - Multiplexing: allows multiple requests and responses to be sent in parallel
    - Helps improve the utilization of network resources and reduces latency
  - Header Compression: efficient header compression algorithm (HPACK)
    - Minimizes amount of data transmitted, resulting in faster page loads times
  - Binary Protocol: Earlier protocol uses textual format; HTTP/2 employs a binary format
    - Compact representation of data, contributing to improved performance

- Server Push: enables servers to push resources to clients before they are explicitly requested
  - Can load related resources (such as stylesheets, images, or scripts) proactively
- Stream Prioritization: enables more important resources to be transmitted first
- Backward Compatible with HTTP/1.1
  - If a client or server doesn't support HTTP/2, communication can still occur using HTTP/1.1
- Many major browsers support HTTP/2 only over secure connections (HTTPS)

HTTP/3.0

- Third major version, developed to address limitations of its predecessor
  - Enhance performance and security
  - On 6 June 2022, IETF published HTTP/3 as a Proposed Standard in RFC 9114
- Uses QUIC (Quick UDP Internet Connections) transport protocol instead of TCP
  - More efficient and faster than TCP
    - Supposedly 3x times faster than HTTP/1.1
  - Operates over UDP (User Datagram Protocol) with built-in support for encryption, flow control, and error correction
- Backward compatible with HTTP/2 and HTTP/1.1
- An evolving standard, slowly gaining adoption

References

- TLS1.0: http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html
- TLS1.2: https://medium.com/@ethicalevil/tls-handshake-protocol-overview-a39e8eee2cf5
- HTTP: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
- HTTP/2.0: https://dl.acm.org/doi/fullHtml/10.1145/2534706.2534721
- The QUIC Transport Protocol: Design and Internet-Scale Deployment: https://dl.acm.org/doi/10.1145/3098822.3098842

Summary

- Basics of HTTP/1.1 Protocol
  - message format, methods and response codes
- HTTPs via TLS
  - Encryption, integrity, challenge/response fundamentals
  - Steps involved via Message exchange
- Features of more recent versions (HTTP/2.0 and HTTP/3.0)