



Tecnologie e applicazioni web

HTTP

Filippo Bergamasco (filippo.bergamasco@unive.it)

<http://www.dais.unive.it/~bergamasco/>

DAIS - Università Ca' Foscari di Venezia

Anno accademico: 2017/2018

Il protocollo alla base del web

HTTP: HyperText Transfer Protocol è il principale protocollo di comunicazione tra i programmi nel web.

- Come abbiamo visto, è stato alla base del suo sviluppo
- Oggi affiancato da altri protocolli e standard (ex. websocket), ma resta sempre fondamentale per il funzionamento del web

Il protocollo alla base del web

HTTP: HyperText Transfer Protocol è il principale protocollo di comunicazione tra i programmi nel web.



Definisce le **regole** sintattiche, semantiche e di sincronizzazione con cui scambiare informazioni

Il protocollo alla base del web

HTTP: HyperText Transfer Protocol è il principale protocollo di comunicazione tra i programmi nel web.



Nato per l'interscambio di ipertesti, non è però limitato a questa specifica tipologia di contenuti

Il protocollo alla base del web

HTTP: **H**yper**T**ext **T**ransfer **P**rotocol è il principale protocollo di comunicazione tra i programmi nel web.

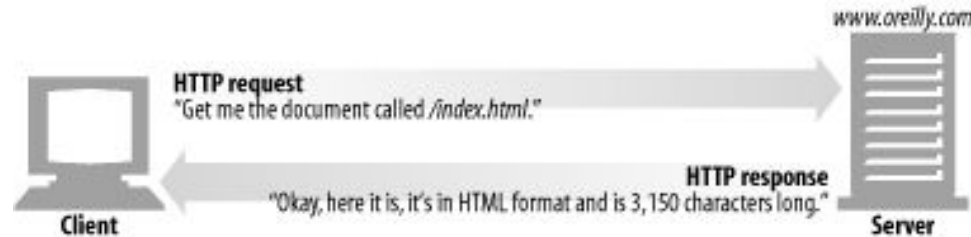


Pensato per il trasferimento (client-server) dei contenuti

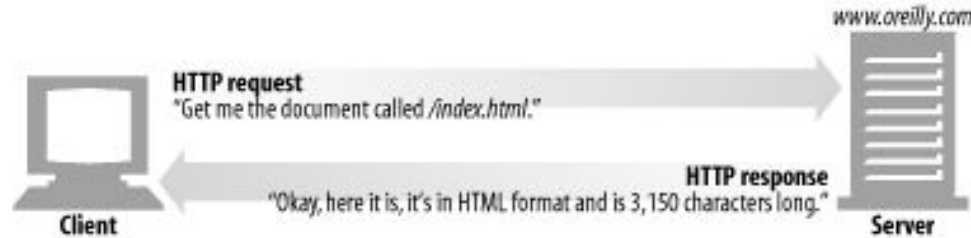
Concetti base

Modello client-server

- HTTP è un protocollo di tipo **request-response** in un modello di comunicazione di tipo **client-server**

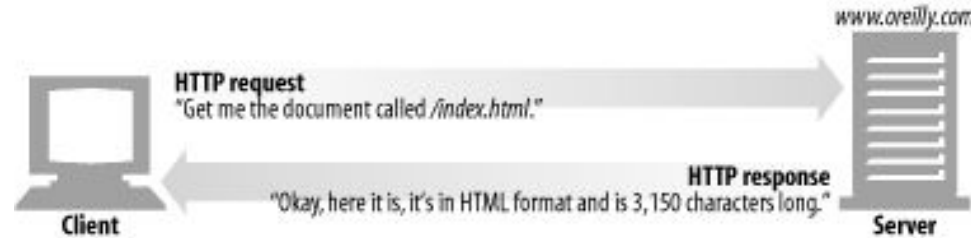


Modello client-server



Un tipico **client** HTTP è ad esempio il vostro web **browser**. La funzionalità base di un browser è quella di chiedere risorse al server (ex pagine html) e visualizzarne il contenuto sullo schermo

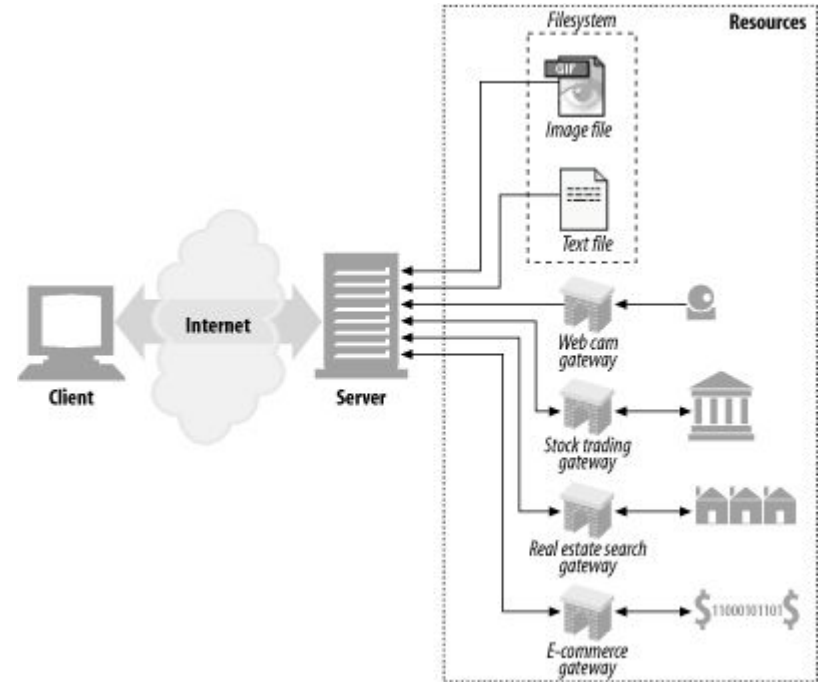
Modello client-server



Un **server** HTTP è un software che si occupa di rispondere alle richieste effettuate da uno o più client.

Risorse

- Il server HTTP ospita (host) delle **risorse**. Le risorse possono essere statiche o dei programmi che generano contenuto on-demand



Tipo delle risorse

- Ciascuna risorsa ha un tipo, detto **MIME** (Multipurpose Internet Mail Extension). Il tipo descrive il formato della risorsa



Tipo delle risorse

Il tipo mime è semplicemente una stringa, formattata in questo modo: `<tipo>/<sottotipo>`

Esempi:

- `text/html` è il tipo mime per un documento HTML
- `text/plain` è il tipo mime per un semplice file di testo con codifica ASCII
- `image/jpeg` è il tipo mime per un'immagine JPEG
- `application/JSON` è il tipo mime per stringhe JSON
- etc.

Identificare le risorse

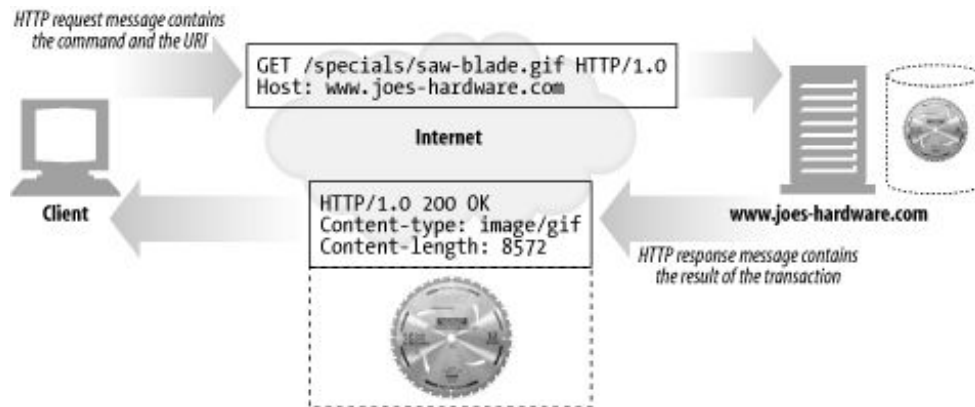
- Ciascuna risorsa ha un nome che la identifica in modo univoco sul web, detto **URI** (Uniform Resource Identifier).

Due differenti tipologie di URI: **URL** e **URN**

- **U**niform **R**esource **L**ocator identifica una risorsa in base al posto (location) in cui si trova.
- **U**niform **R**esource **N**ame serve come nome univoco per la risorsa indipendentemente da dove si trova (non usato in pratica)

Request & Response

HTTP è formato da sequenze di **transazioni** (transactions) che consistono in un **comando** di richiesta (client ➡ server) detto **request** e una risposta (client ← server) detta **response**, formattati in un **messaggio** HTTP



Metodi

- HTTP supporta diversi tipi di comandi, detti **metodi**
 - Ciascun messaggio inviato al server (request) contiene un metodo
 - 5 metodi più comuni
 - GET
 - PUT
 - DELETE
 - POST
 - HEAD

Codici di stato

- Ciascun messaggio di risposta (response) contiene un codice di stato (status code)
 - Codice numerico a 3 cifre per notificare il client se la richiesta è andata a buon fine o se sono necessarie azioni aggiuntive
 - Esempi di status code:
 - **200** (utilizzato per notificare il successo della richiesta)
 - **302** (utilizzato per notificare un redirect)
 - **404** (utilizzato per notificare il client nel caso che la risorsa richiesta non esista)

Versioni del protocollo

- HTTP/0.9
 - Primo prototipo del 1991. Contiene molti bug concettuali e supporta soltanto il metodo GET
- HTTP/1.0
 - Prima versione adottata su larga scala. Risolve molti dei problemi di HTTP/0.9, Aggiunge metodi e tipi MIME

Versioni del protocollo

- HTTP/1.0+
 - Aggiunto supporto per connessioni keep-alive, virtual hosting e proxy
- HTTP/1.1
 - Corregge alcuni problemi delle precedenti versioni e include molte ottimizzazioni che ne migliorano le performance. E' la versione attualmente usata nella maggior parte dei casi

Protocollo in dettaglio

Struttura degli URL

Un Uniform Resource Locator (URL) è un nome standardizzato per definire risorse nel web.

- Solitamente rappresenta l'“entry point” da cui gli utenti iniziano a navigare sul web
- Codifica il nome della risorsa, il luogo dove trovarla e il modo in cui è possibile ottenerla
- Il formato generale è composto da una stringa composta da 9 parti distinte (non tutte obbligatorie)

Struttura degli URL

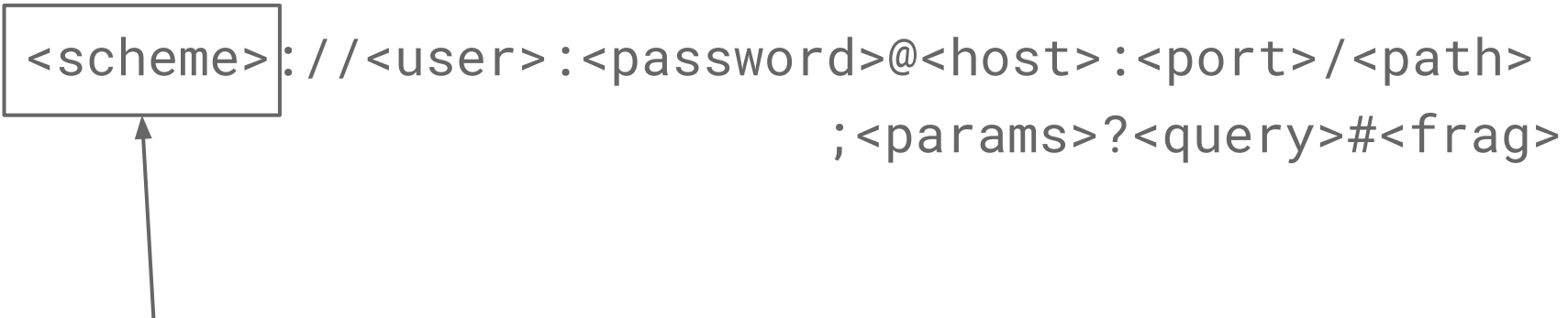
<scheme> : // <user> : <password> @ <host> : <port> / <path>
; <params> ? <query> # <frag>

Ex:

<http://www.joes-hardware.com/seasonal/index-fall.html>

Struttura degli URL

`<scheme>://<user>:<password>@<host>:<port>/<path>
;<params>?<query>#<frag>`

A diagram illustrating the structure of a URL. The text is split across two lines. The first line contains the components from the scheme to the path, and the second line contains the optional parameters, query, and fragment. A rectangular box is drawn around the first part of the first line, specifically around the placeholder for the scheme. An arrow originates from the bottom of this box and points downwards towards the explanatory text below.

Lo schema specifica che protocollo usare quando si accede al server per ottenere la risorsa indicata.

Ex: http, ftp, rtsp, etc.

Struttura degli URL

`<scheme> : // <user> : <password> @ <host> : <port> / <path>
; <params> ? <query> # <frag>`



L'utente e la password possono essere talvolta richiesti su alcuni protocolli per autenticarsi con il server.

Se non indicato, l'utente di default è "anonymous"

Struttura degli URL

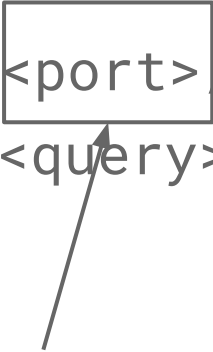
`<scheme>://<user>:<password>@<host>:<port>/<path>
;<params>?<query>#<frag>`



L'indirizzo del server da cui è possibile reperire la risorsa. Può essere un hostname oppure un indirizzo IP. Il valore deve essere sempre specificato

Struttura degli URL


`<scheme>://<user>:<password>@<host>:<port>/<path>
;<params>?<query>#<frag>`



La porta (TCP) su cui il server sta attendendo connessioni. A seconda dello schema specificato, la porta può avere un valore di default e può essere omessa. Ex: per HTTP la porta di default è la 80

Struttura degli URL

`<scheme>://<user>:<password>@<host>:<port>/<path>
;<params>?<query>#<frag>`



Il nome (locale) della risorsa sul server. La sintassi del path è specifico della combinazione schema-server. In HTTP il path può essere diviso in più segmenti separati da “/”. Ex: seg1/seg2/seg3

Struttura degli URL

<scheme>://<user>:<password>@<host>:<port>/<path>
; <params> ?<query> #<frag>



Questa componente permette di associare liste di parametri nome-valore ad un segmento del path. Questo può essere necessario per fornire al server web ulteriori informazioni (parametri) sulla risorsa richiesta

Struttura degli URL

`<scheme>://<user>:<password>@<host>:<port>/<path>
;<params>?<query>#<frag>`



Ex:

`ftp://prep.ai.mit.edu/pub/gnu?type=d`

`http://www.aa.com/hammers;sale=false/index.html;graphics=true`

Struttura degli URL

<scheme> ://<user>:<password>@<host>:<port>/<path>
;<params>?<query>#<frag>



Simile a <params> (e più usato in ambito HTTP),
<query> permette di definire coppie nome-valore per
restringere il contesto della risorsa richiesta

Struttura degli URL

<scheme> : // <user> : <password> @ <host> : <port> / <path>
; <params> ? <query> # <frag>

A diagram illustrating the structure of a URL. The text shows the components of a URL: <scheme> : // <user> : <password> @ <host> : <port> / <path> ; <params> ? <query> # <frag>. A rectangular box is drawn around the ? <query> part, and an arrow points from below towards the box.

Ex: `http://ww.a.com/inventory.html?item=1234`

In questo caso la risorsa `inventory.html` viene ristretta a riferirsi soltanto ad un ipotetico item numero 1234

Struttura degli URL

<scheme>://<user>:<password>@<host>:<port>/<path>
;<params>?<query>#<frag>



Ex: `http://ww.a.com/inventory.html?item=1234&num=3`

Le coppie chiave-valore possono essere separate dal carattere & oppure da ; (meno comune)

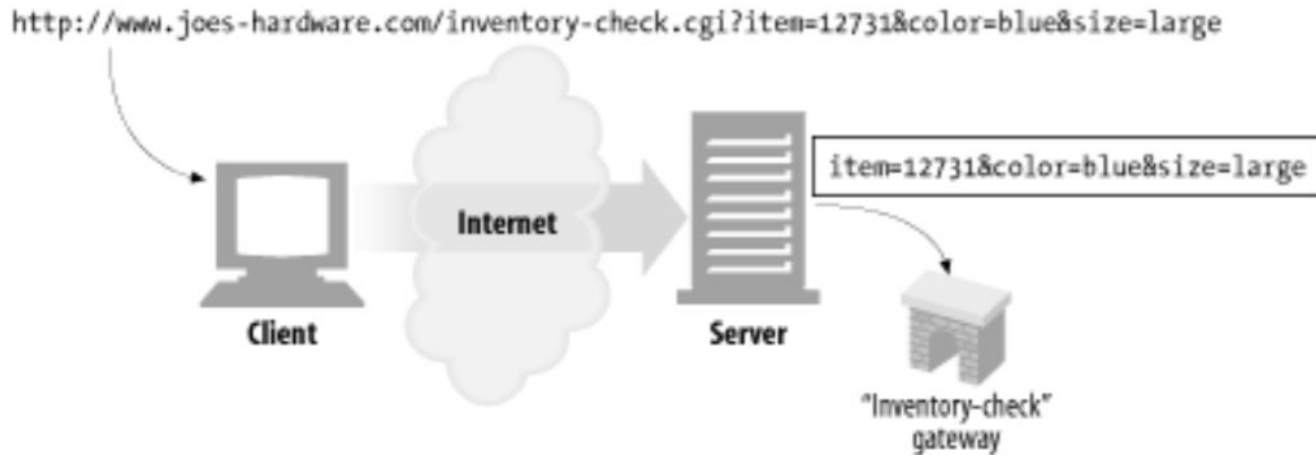
<params> o <query>?

<scheme> : // <user> : <password> @ <host> : <port> / <path>
; <params> ? <query> # <frag>

Che differenza c'è tra <params> e <query>?

- Il primo si riferisce ad un segmento di path, il secondo alla risorsa
- <param> solitamente interessa il server web mentre <query> il possibile gateway che genera la risorsa.

<params> o <query>?



Ex: Il gateway "inventory-check" utilizza la stringa <query> dell'url per effettuare una ricerca nel database riguardo l'articolo 12731, di colore blue e di taglia large

Struttura degli URL

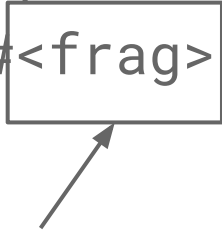
`<scheme>://<user>:<password>@<host>:<port>/<path>
;<params>?<query>#<frag>`

A diagram illustrating the structure of a URL. The text shows the components of a URL: scheme, user, password, host, port, path, params, query, and fragment. The fragment part, represented as <frag> after a hash symbol, is enclosed in a rectangular box. An arrow points from the bottom of this box towards the explanatory text below.

Fragment può essere usato per identificare frammenti all'interno della risorsa specificata. Per esempio, in un file HTML può puntare ad un paragrafo o ad un'immagine

Struttura degli URL

`<scheme>://<user>:<password>@<host>:<port>/<path>
;<params>?<query>#<frag>`

A diagram illustrating the structure of a URL. The text shows the components of a URL: scheme, user, password, host, port, path, params, query, and fragment. The fragment part, represented as <frag> after a hash symbol, is enclosed in a rectangular box. An arrow points from the bottom of this box towards the explanatory text below.

Visto che solitamente il server gestisce intere risorse, e non parte di esse, la parte di fragment non è inviata al server durante una richiesta HTTP. Essa viene usata dal client in fase di visualizzazione

Frammenti di risorsa

`http://www.joes-hardware.com/tools.html#drills`

(a) User selects link to "`http://www.joes-hardware.com/tools.html#drills`"

(Fragment is NOT sent to the server)

(b) Browser makes request to `http://www.joes-hardware.com/tools.html`



(c) Server returns entire HTML page



Browser scrolls down to start at named "drills" fragment

(d) Browser displays HTML page starting with named "drills" fragment

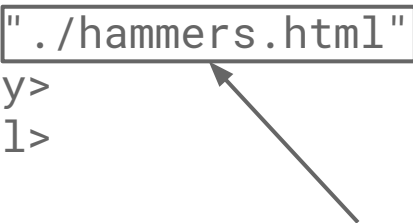
URL assoluti e relativi

Gli URL visti fino ad ora sono detti assoluti perchè contengono al loro interno tutte le informazioni per accedere ad una risorsa.

Specialmente lavorando con pagine HTML, è possibile utilizzare URL incompleti detti relativi. Gli URL relativi possono essere resi assoluti rispetto ad un certo URL di base

URL relativi

```
<html>
<head><title>Joe's Tools</title></head>
<body>
<h1> Tools Page  </h1>
<p> Joe's Hardware Online has the largest selection of <a
href="./hammers.html" />hammers
</body>
</html>
```



Questo URL è relativo. Manca almeno lo schema e l'host a cui richiedere la risorsa

URL relativi

Se assumiamo il seguente URL di base:

`http://www.joes-hardware.com/tools.html`



URL di base

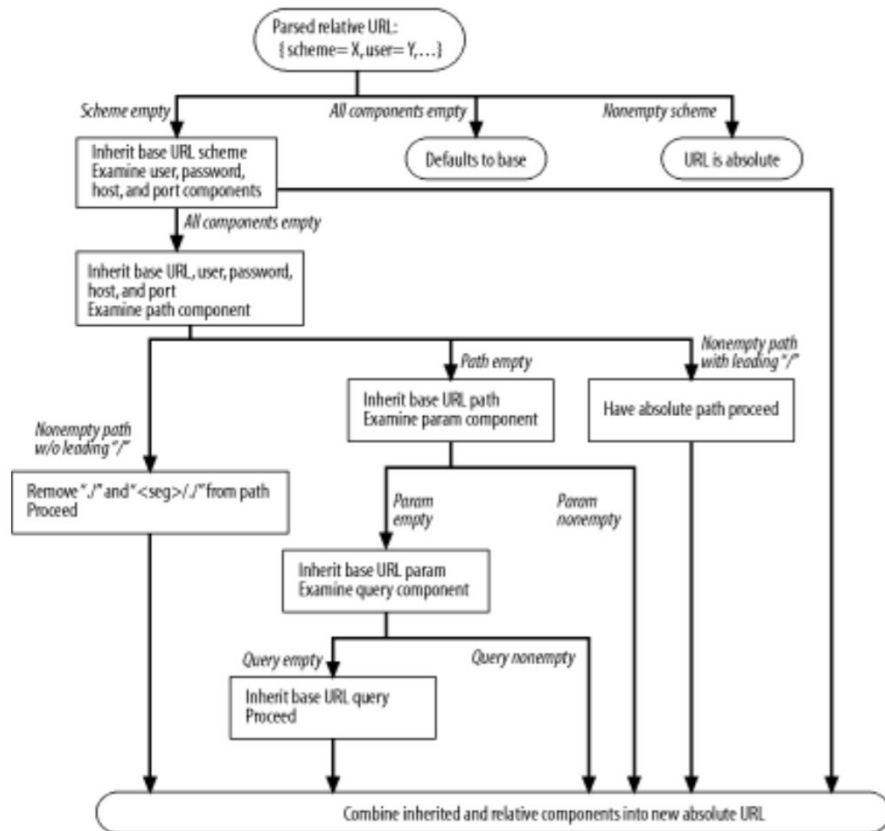
L'URL di base può essere ricavato in 2 modi diversi:

1. In modo esplicito nella risorsa. Ad esempio usando il tag `<base>` in un documento HTML
2. Utilizzando l'URL (assoluto) della risorsa in cui l'URL relativo è inserito (caso più comune)

In generale l'URL relativo e di base possono essere complessi e contenere parametri, username, etc.

I due URL vanno entrambi scomposti prima di essere ricombinati in un URL assoluto

Da URL relativo ad assoluto



- Algoritmo completo di costruzione di un URL assoluto
- Contenuto in RFC 2396

The URL character set

Da specifiche, gli URL possono solo contenere simboli dal set di caratteri **ASCII**.

Se il nome della risorsa contiene caratteri estranei al set ASCII (ex. Lettere accentate, cirilliche, etc) è necessario convertire il nome (escaping) in un set di caratteri valido

Escaping

L'escaping dei caratteri non ammessi in un URL avviene trasformando tali caratteri nella sequenza:

- % + <codice esadecimale 2 cifre>

Il codice esadecimale da inserire dipende dal set di caratteri del browser o della pagina. Di default il set è UTF8

Escaping

Character	From Windows-1252	From UTF-8
space	%20	%20
!	%21	%21
"	%22	%22
#	%23	%23
\$	%24	%24
%	%25	%25
&	%26	%26

URL: caratteri non ammessi

Oltre ai caratteri non ASCII non sono ammessi i seguenti caratteri perché in conflitto con altri utilizzi

Carattere	Restrizione
%	Riservato come carattere di escape
/	Riservato come delimitatore per segmenti del path
.	Riservato come componente del path
..	Riservato come componente del path
#	Riservato per delimitare il <fragment>

URL: caratteri non ammessi

Oltre ai caratteri non ASCII non sono ammessi i seguenti caratteri perché in conflitto con altri utilizzi

Carattere	Restrizione
?	Riservato per delimitare la stringa <query>
:	Riservato per delimitare lo schema, host e porta, user/password
;	Riservato per delimitare i parametri
{ } \ ^ ~ [] `	Riservati per possibili conflitti con alcuni agenti di trasporto (gateways)
@&=	Riservati perché hanno significati speciali in alcuni schemi (non HTTP)

IRI

L'**I**nternationalized **R**esource **I**dentifier (**IRI**) è stato definito nel 2005 per estendere l'URI con caratteri contenuti nell' universal character set

Backward-compatibile attraverso l'escaping standard degli URL. Le applicazioni e i protocolli già creati per supportare gli IRI li utilizzano direttamente.

IRI

Esempio di un IRI:

<https://en.wiktionary.org/wiki/Ῥόδος>

URL corrispondente:

<https://en.wiktionary.org/wiki/%E1%BF%AC%CF%8C%CE%B4%CE%BF%CF%82>

IRI e il phishing

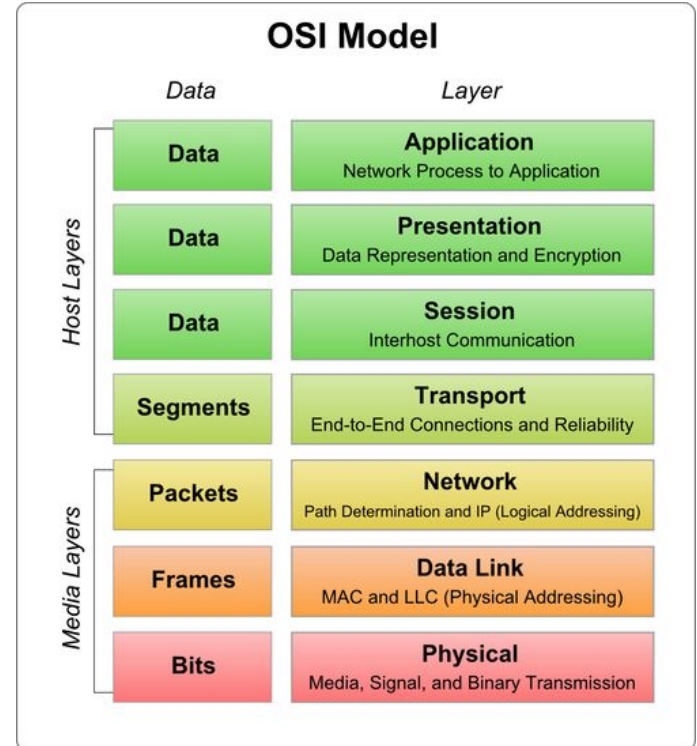
Uno dei problemi principali degli IRI è il cosiddetto Internationalized domain name (IDM) homoglyph attack.

Idea:

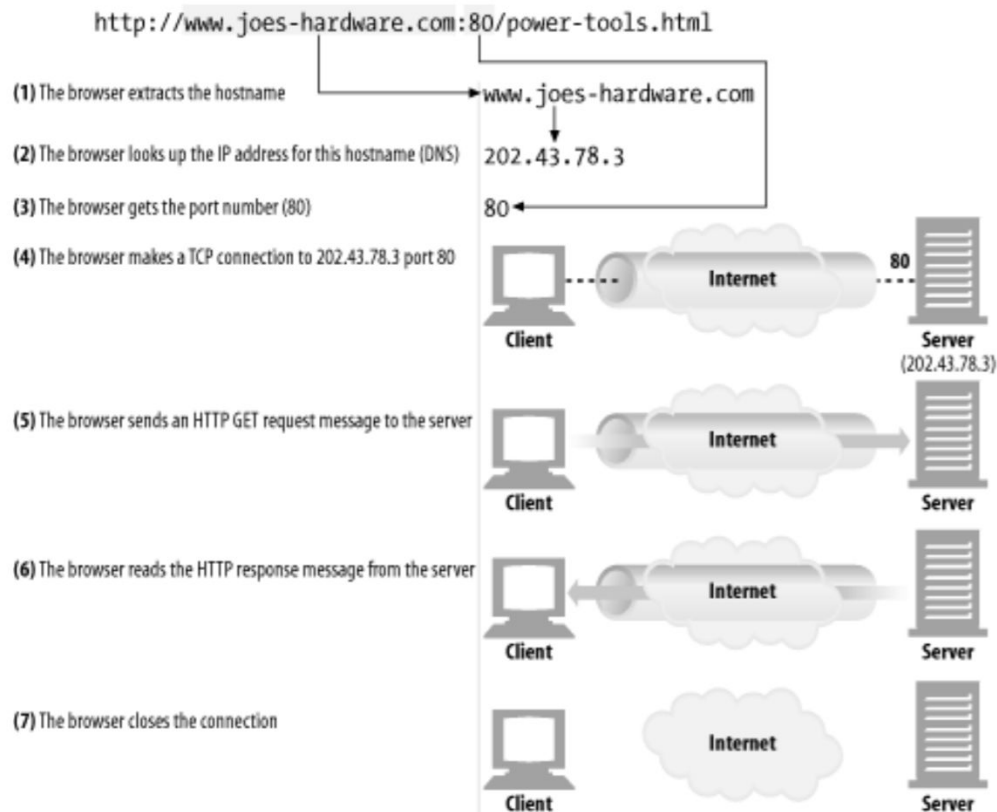
Si utilizzano caratteri simili a quelli ASCII usati comunemente per indirizzare l'utente su un sito malevolo: Ex: paypal.it invece che paypal.it

Connessione HTTP

- HTTP è un protocollo dello strato "Applicazione" nel modello OSI
- Utilizza connessioni TCP/IP per lo scambio di messaggi tra client e server
- L'indirizzo e la porta sono definite all'interno dell'URL



Connessione HTTP



TCP: Concetti base

TCP fornisce un flusso bidirezionale di dati

- Orientato alla connessione
- Affidabile (i dati vengono trasmessi correttamente oppure viene generato un errore)
- I dati arrivano nello stesso ordine con cui sono stati trasmessi (bit pipe)
- Viene effettuato un controllo di flusso per evitare le congestioni

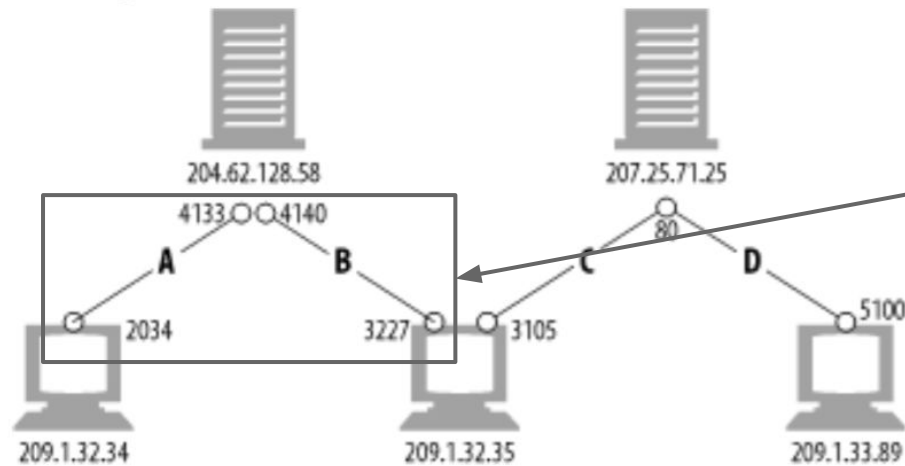
TCP: Concetti base

La connessione TCP tra client e host è definita in modo univoco da 4 valori:

- <ip sorgente> <porta sorgente> <ip destinazione>
<porta destinazione>

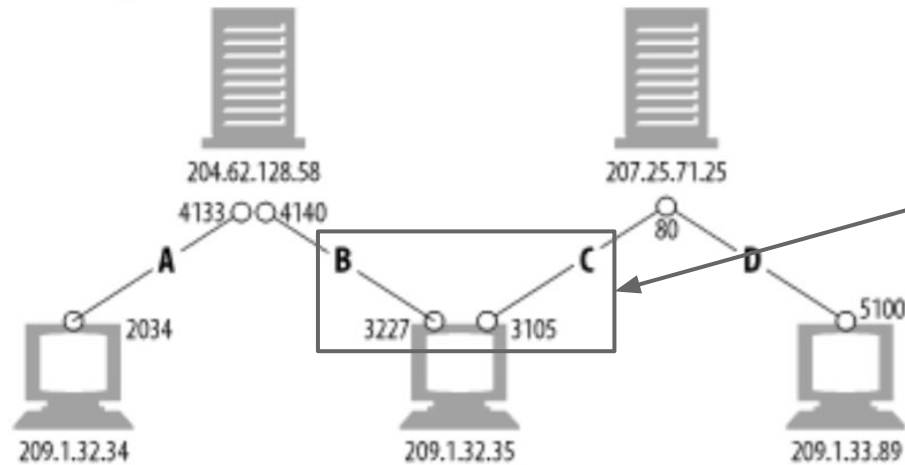
Sono permesse più connessioni da e verso stessi indirizzi ip e porte a patto che ci sia al più **una connessione** con una certa quadrupla di valori

TCP: Concetti base



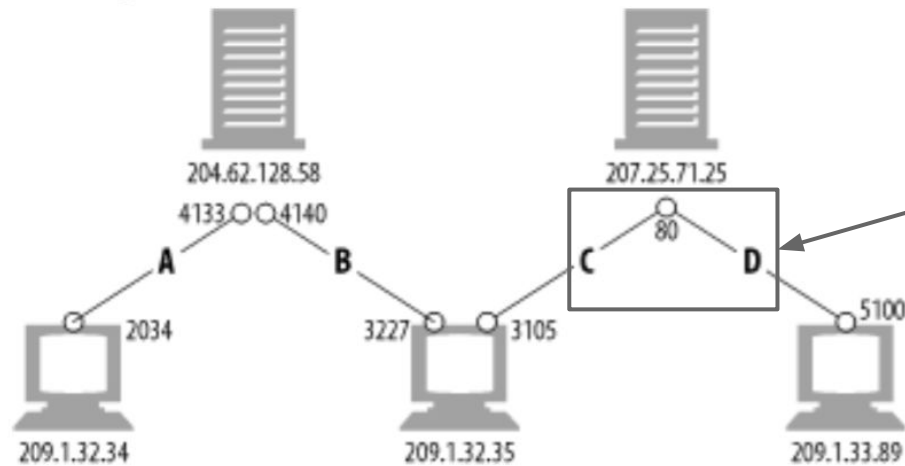
Due connessioni da sorgenti diverse su porte diverse

TCP: Concetti base



B e C condividono
lo stesso indirizzo IP
sorgente

TCP: Concetti base



C e D condividono
la stessa
destinazione sulla
stessa porta

Messaggi HTTP

Una volta che il **client** stabilisce una connessione TCP con il server, il protocollo HTTP prevede lo scambio di almeno 2 **messaggi**:

- Il client invia un messaggio al server con una richiesta (**request**) e riceve dal server un messaggio di risposta (**response**)
- Ciascun messaggio contiene sempre una stringa di testo che ne descrive il significato e il contenuto
- Un messaggio può contenere dei dati da trasferire

Messaggi HTTP

Un messaggio è composto di 3 parti:

Start line	Una stringa di testo, terminata da CRLF, che descrive il messaggio
Header	Molteplici stringhe di testo (ciascuna terminata da CRLF) che definiscono opzioni e proprietà del messaggio. L'header termina con una riga vuota (contenente solo CRLF)
Body	Dati generici (anche binari) facenti parte dell'oggetto del messaggio

Request

I messaggi di tipo request sono inviati dal client al server per richiedere un'azione. Sono composti in questo modo:

Start line	<method> <request-URL> <version>
Header	<name1>: <value1> <name2>: <value2> ...
Body	Dati binari o un semplice CRLF se non sono presenti dati

Metodi

<method> <request-URL> <version>

<name1>: <value1>

<name2>: <value2>

...

Dati binari o un semplice CRLF se non sono presenti dati

L'azione che il client richiede al server di effettuare. I metodi possibili sono ($\frac{1}{3}$):

GET	Chiede al server di inviare una certa risorsa al client
HEAD	Simile a GET ma chiede al server di inviare soltanto l'header della risorsa e non il contenuto Utilizzi: <ul style="list-style-type: none">• Conoscere il tipo di una risorsa• Controllare se una risorsa esiste• Controllare se una risorsa è stata modificata dall'ultimo accesso
PUT	Inverso di GET. Chiede al server di memorizzare una certa risorsa inviata dal client

Metodi

L'azione che il client richiede al server di effettuare. I metodi possibili sono (2/3):

<method> <request-URL> <version>

<name1>: <value1>

<name2>: <value2>

...

Dati binari o un semplice CRLF se non sono presenti dati

POST	Utilizzato per inviare dati generici al server. A differenza di PUT i dati non vanno memorizzati ma utilizzati da software esterni (esempio tipico inviare dati di form HTML)
TRACE	Utilizzato per avere informazioni diagnostiche sulla topologia delle connessioni tra client e server (presenza di gateway, proxy, etc)
OPTIONS	Chiede al server una lista delle funzionalità supportate. Utilizzata talvolta dal client per determinare il modo migliore per accedere alle risorse

Metodi

L'azione che il client richiede al server di effettuare. I metodi possibili sono (3/3):

<method> <request-URL> <version>

<name1>: <value1>

<name2>: <value2>

...

Dati binari o un semplice CRLF se non sono presenti dati

DELETE

Utilizzata dal client per richiedere al server la rimozione di una certa risorsa. Come PUT è quasi sempre necessaria un'autenticazione

LOCK,
MKCOL,
COPY,
MOVE,
etc

Un server può supportare metodi aggiuntivi (estensioni) al di fuori del protocollo HTTP/1.1.
Ex: WebDAV

Request-URL

<method> **<request-URL>** <version>

<name1>: <value1>

<name2>: <value2>

...

Dati binari o un semplice CRLF se non sono presenti dati

Path della risorsa. Può essere l'URL completo o solo la componente di path della risorsa.

Version

<method> <request-URL> **<version>**

<name1>: <value1>

<name2>: <value2>

...

Dati binari o un semplice CRLF se non sono presenti dati

Versione del protocollo HTTP
che il client desidera utilizzare.
Solitamente "HTTP/1.1"

Response

I messaggi di tipo response sono inviati dal server al client in risposta ad un'azione richiesta nel precedente messaggio di request:

Start line	<version> <status> <reason-phrase>
Header	<name1>: <value1> <name2>: <value2> ...
Body	Dati binari o un semplice CRLF se non sono presenti dati

Status

<version> **<status>** <reason-phrase>

<name1>: <value1>

<name2>: <value2>

...

Dati binari o un semplice CRLF se non sono presenti dati

Numero di 3 cifre che descrive il risultato della richiesta. La prima cifra descrive il tipo generico di stato, le ultime due lo stato specifico

Table 3-2. Status code classes

Overall range	Defined range	Category
100-199	100-101	Informational
200-299	200-206	Successful
300-399	300-305	Redirection
400-499	400-415	Client error
500-599	500-505	Server error

Reason-phrase

`<version> <status> <reason-phrase>`

`<name1>: <value1>`

`<name2>: <value2>`

...

Dati binari o un semplice CRLF se non sono presenti dati

Stringa che fornisce una descrizione testuale (human readable) dello status code specificato

Headers

HTTP/1.1 definisce molteplici headers a seconda del tipo di messaggio e di contenuto da trasferire. Si dividono in:

- Generali
- Di richiesta (request)
- Di risposta (response)
- Di entità
- Estensioni. Il protocollo tollera l'utilizzo di header al di fuori dello standard

Headers generali

Alcuni headers generali più importanti

Date	Data e ora di generazione del messaggio (non della risorsa)
Upgrade	Specifica una nuova versione del protocollo che il mittente del messaggio desidera utilizzare
Cache-Control	Utilizzato per dare direttive sul caching
Trailer	Utilizzato per i trasferimenti “Chunked”, cioè che coinvolgono più di un messaggio

Headers request

Alcuni headers più importanti per messaggi di tipo request

Client-IP	Ip address della macchina client
Host	IP address della macchina server (verso cui si invia la richiesta)
Accept	Utilizzato per informare il server sul tipo di files che il client supporta (ex. JPEG/PNG)
Accept-Encoding	Utilizzato per informare il server sul tipo di codifica dei dati supportata (ex. Gzip)
If-Modified-Since	Richiede al server di restituire la risorsa soltanto se è stata modificata successivamente ad una certa data

Headers request

Alcuni headers più importanti per messaggi di tipo request

Authorization	Contiene i dati di autenticazione del client. Ad esempio, in basic access authentication le credenziali “username:password” sono codificate in BASE64 e inviate attraverso questo header
Cookie	Utilizzata dal client per inviare generici “token” al server, utili ad esempio per il meccanismo delle sessioni

Headers response

Alcuni headers più importanti per messaggi di tipo response

WWW-Authenticate	Lista di challenges dal server verso il client per invitarlo ad autenticarsi. Utilizzato nel basic access authentication
Set-Cookie	Utilizzata dal server per chiedere al client di memorizzare un generico token
Server	Nome e la versione del software che agisce da web server

Headers di entità

Alcuni headers più importanti per descrivere i dati (body) di un messaggio HTTP

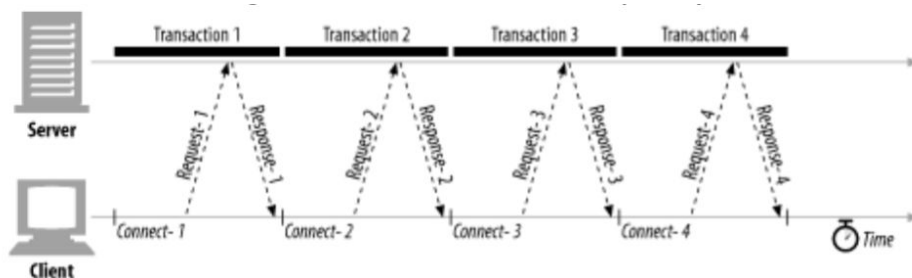
Allow	Lista dei metodi che possono essere richiesti su una certa risorsa
Content-Encoding	Encoding della risorsa (ex. gzip)
Content-Length	Dimensione (numero di bytes) della risorsa inviata nel messaggio.
Content-Type	MIME-type della risorsa
Last-Modified	Data di ultima modifica della risorsa

Connessioni parallele o persistenti

Una transazione HTTP standard prevede, nell'ordine:

- Di instaurare una connessione TCP (client ➡ server)
- Invio di un messaggio request (client ➡ server)
- Invio di un messaggio response (server ➡ client)
- Chiusura della connessione HTTP

Se il contenuto dei messaggi è ridotto, l'overhead della connessione diventa significativo



Connessioni parallele o persistenti

Due possibili soluzioni:

1. Il client instaura più connessioni parallele verso il server
 - Vantaggi: molto rapido, può gestire risorse su server diversi
 - Svantaggi: Elevato utilizzo di risorse del client (memoria), non porta vantaggi se la banda del client è limitata
2. Il client riutilizza la stessa connessione TCP per molteplici messaggi sequenziali. Supportato in HTTP/1.1
 - Vantaggi: Rapido se si inviano molti messaggi di dimensione ridotta

HTTP e la sicurezza

Il protocollo HTTP non prevede in modo intrinseco funzionalità per garantire la sicurezza e la riservatezza dei dati che vengono scambiati tra client e server

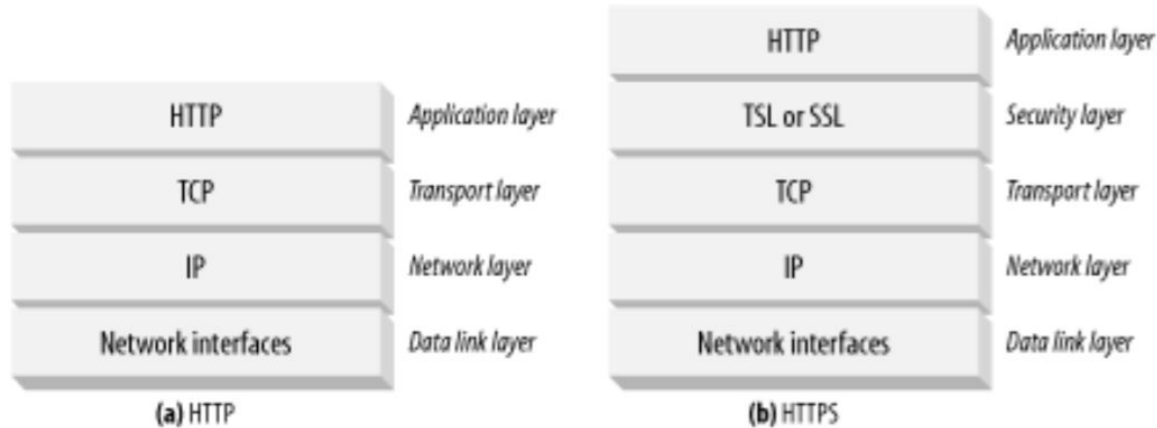
Con l'evoluzione del web moderno, specialmente per l'e-commerce, è nata la necessità di rendere il protocollo sicuro per la gestione di transazioni bancarie, autenticazione, etc.

HTTP e la sicurezza

Chiunque connesso alla rete può potenzialmente leggere il traffico HTTP generato da due entità per rubare dati confidenziali o alterare in modo malevolo il contenuto (man-in-the-middle attack)

La sicurezza in ambito HTTP è realizzata inserendo uno strato intermedio tra HTTP e TCP, chiamato SSL (Secure Socket Layer) o TLS (Transport Layer Security)

HTTPS



Il vantaggio di usare un layer aggiuntivo è che, una volta stabilita una connessione sicura, il protocollo HTTP resta inalterato.

Client e server non devono alterare la loro logica di processo per utilizzare HTTPS

Cosa fornisce HTTPS

Autenticazione del server

- Un client può verificare in modo sicuro di essere connesso al server atteso e non ad un impostore

Autenticazione del client

- Il server può verificare l'identità di un certo client connesso indipendentemente dalle tecniche HTTP di autenticazione

Cosa fornisce HTTPS

Integrità della trasmissione

- E'possibile rilevare se i dati scambiati fra client e server sono stati manomessi da un terzo agente

Cifratura della trasmissione

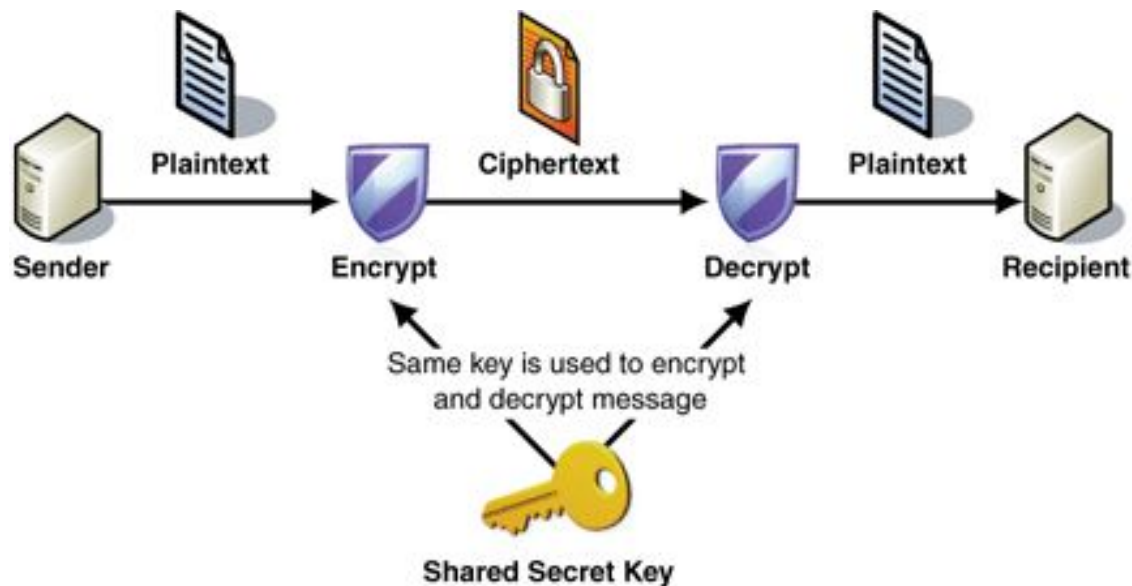
- Tutti i dati trasmessi sono leggibili soltanto dal rispettivo client e server. Non è possibile spiare il traffico in transito, nemmeno se si opera su un canale non sicuro (ex. WiFi non protetto)

HTTPS e crittografia

HTTPS è basato sulle seguenti tecniche di crittografia:

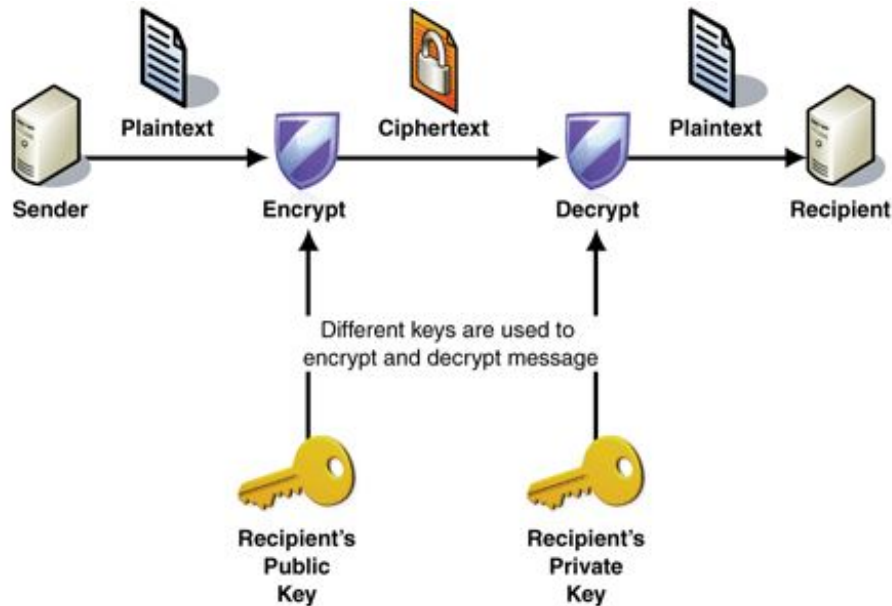
- Crittografia a chiave simmetrica
- Crittografia a chiave asimmetrica
- Firma digitale
- Certificato digitale

Crittografia a chiave simmetrica



Problema: come scambiare le chiavi in modo sicuro?

Crittografia a chiave asimmetrica



Chiave pubblica usata per cifrare. Chiave privata usata per decifrare.
Le chiavi pubbliche possono essere diffuse senza problemi

Firma digitale

SIGNING



VERIFICATION



Chiave privata usata per firmare. Chiave pubblica usata per verificare la firma

Certificato digitale

Problema: Chi garantisce (certifica) che una certa chiave pubblica appartenga effettivamente alla persona che sostiene di essere?

“Un certificato digitale è un documento elettronico che attesta l'associazione univoca tra una chiave pubblica e l'identità di un soggetto”

https://it.wikipedia.org/wiki/Certificato_digitale

Certificato digitale

Un certificato digitale solitamente contiene:

- Il nome del soggetto (persona, server, organizzazione, etc)
- La data di fine validità della firma
- Informazioni su chi attesta (firma) il certificato
- La chiave pubblica del soggetto

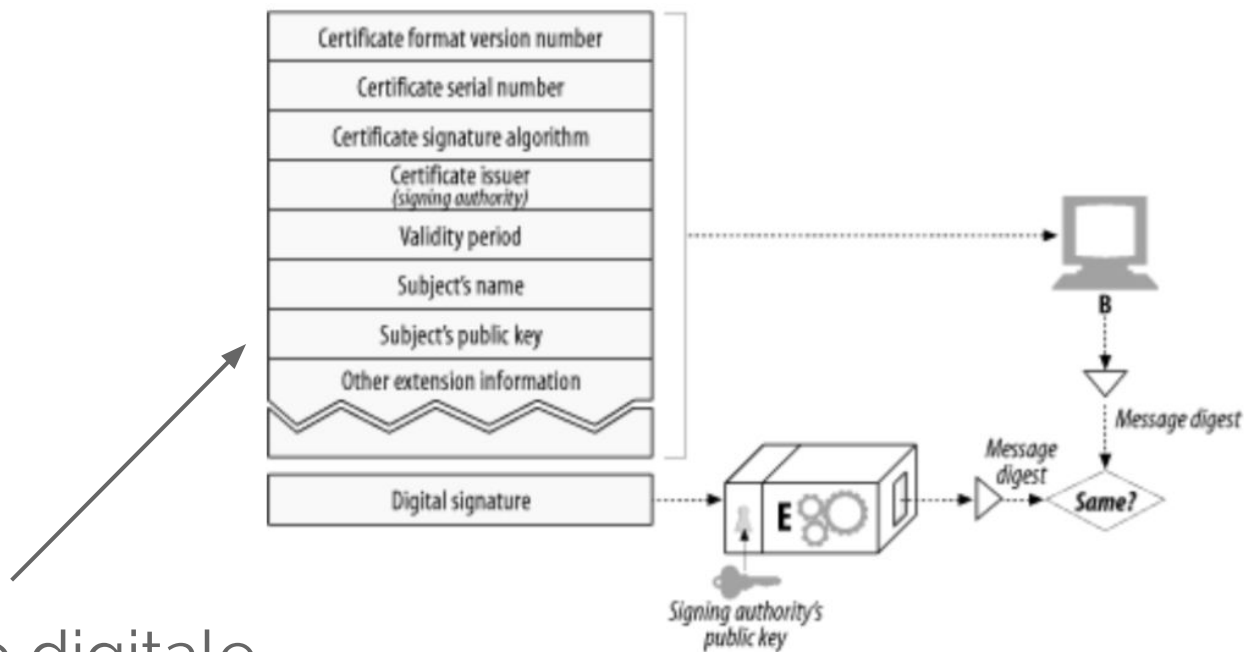
Tutto firmato da una **signing authority** riconosciuta dal client come sicura

HTTPS

Quando il browser effettua una connessione HTTPS ad un server, vengono effettuate le seguenti operazioni dal layer SSL o TLS:

- Richiede il certificato digitale del server
- Dal certificato si ricava il nome della signing authority
- Se la signing authority è nota e pre-installata nel browser, controlla la firma digitale utilizzando la chiave pubblica della stessa

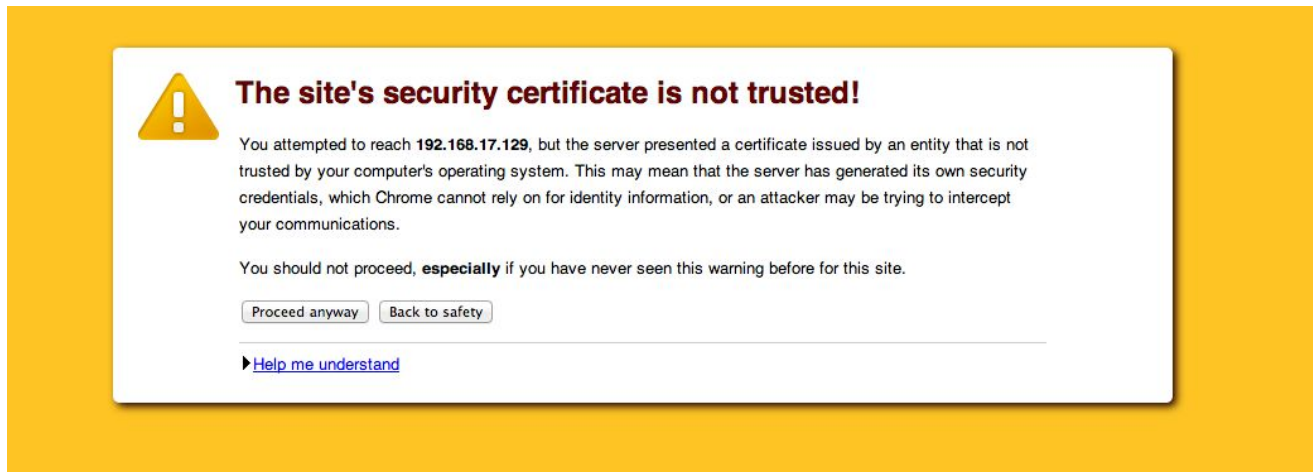
HTTPS



Certificato digitale
scaricato dal server

HTTPS

Se la signing authority non è riconosciuta, o se il certificato è self-signed, il browser avverte l'utente che l'identità del server non può essere verificata

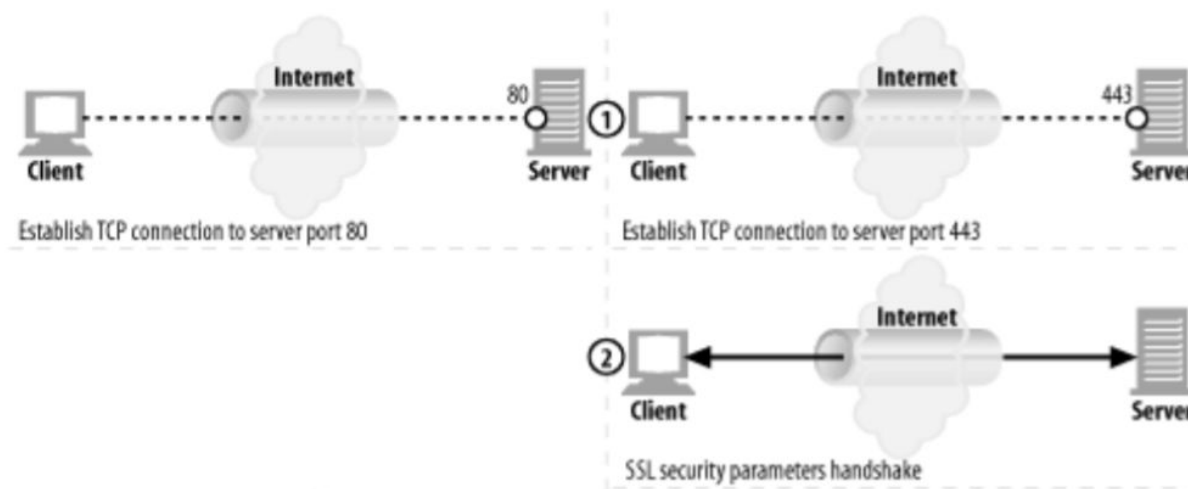


HTTPS

Dopo aver controllato il certificato, il layer SSL/TLS negozia tra client e server i parametri di sicurezza da utilizzare (tipo di algoritmo, versioni, etc), si scambiano le chiavi pubbliche che vengono usate per creare un canale di comunicazione cifrato sopra TCP

A questo punto il protocollo HTTPS opera senza differenze rispetto a HTTP. Il layer SSL/TLS agisce in modo trasparente per cifrare la comunicazione

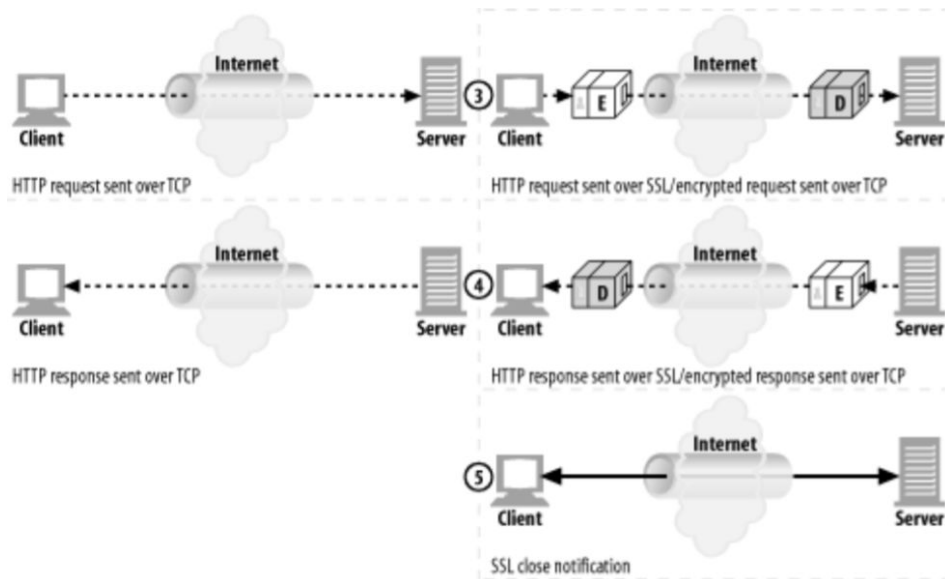
HTTP vs HTTPS



- Si stabilisce una connessione TCP sulla porta 80

- Si stabilisce una connessione TCP sulla porta 443
- Si effettua l'handshake dei parametri di sicurezza

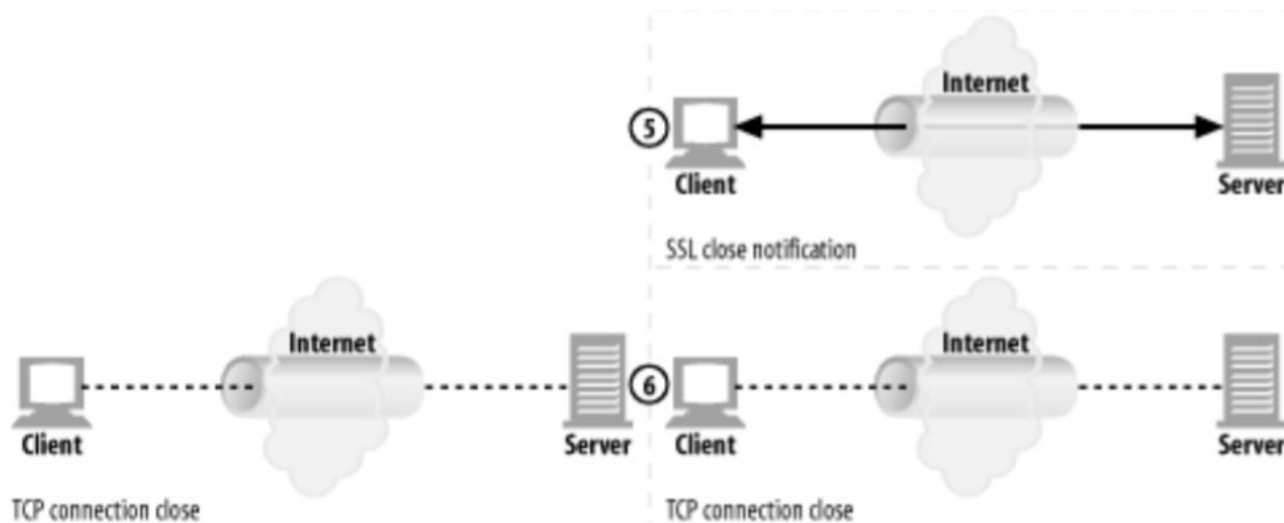
HTTP vs HTTPS



- Request e response sono inviate e ricevute sul canale TCP

- Request e response sono inviate e ricevute in modo cifrato attraverso TCP

HTTP vs HTTPS



- La connessione TCP viene chiusa

- La connessione SSL/TLS viene chiusa
- La connessione TCP viene chiusa