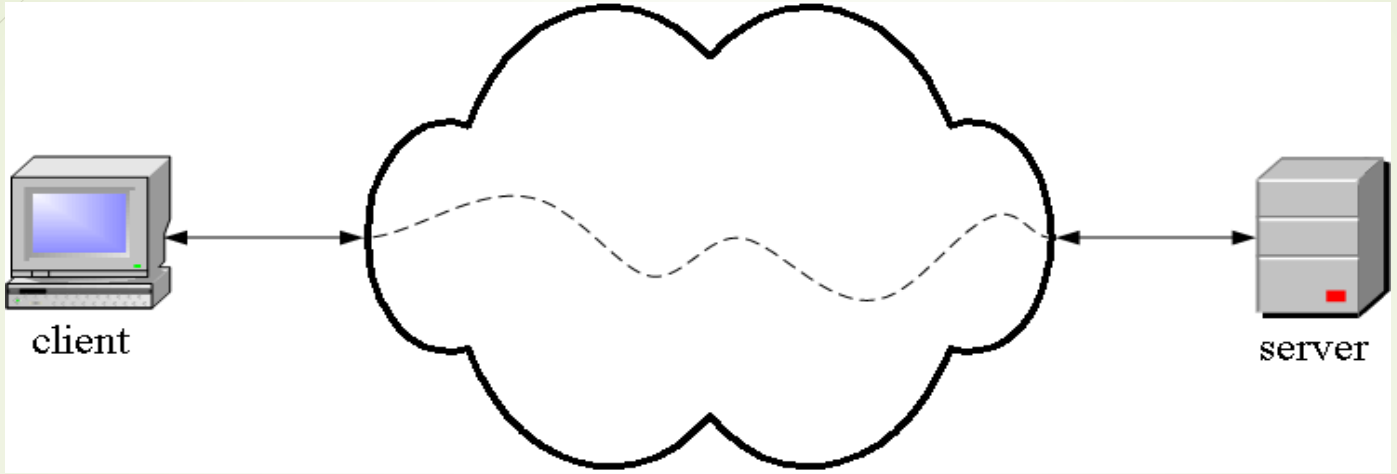
HTTP

Hypertext Transfer Protocol

What is Http?

- Short for **HyperText Transfer Protocol**, **HTTP** is a set of standards that allow users of the World Wide Web to exchange information found on web pages.
- The HTTP protocol is a request/response protocol based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.
- When accessing any web page entering http:// in front of the address tells the browser to communicate over HTTP. For example, the URL for Computer Hope is http://www.myexample.com.
- Protocol for transfer of various data formats between server and client
  - Plaintext
  - Hypertext
  - Images
  - Video
  - Sound

HTTP is an application layer protocol



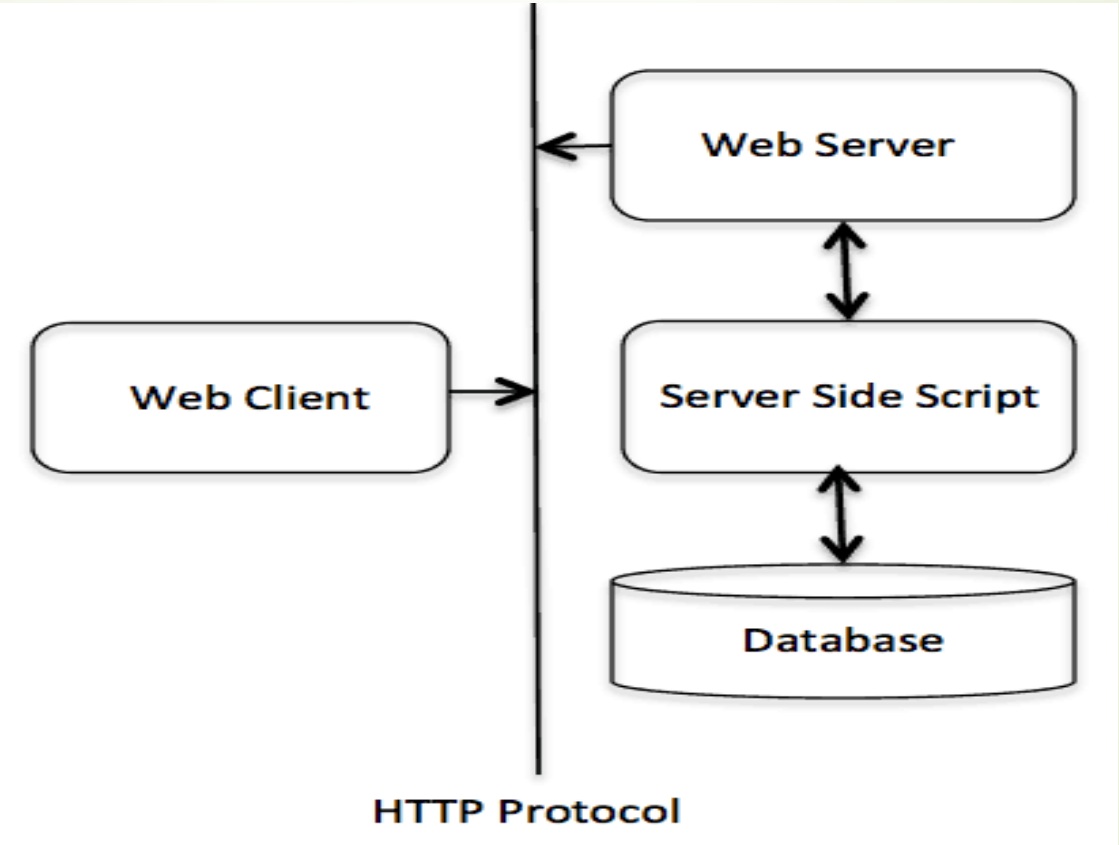
- The Web client and the Web server are application programs
- Application layer programs do useful work like retrieving Web pages, sending and receiving email or transferring files
- Lower layers take care of the communication details
- The client and server send messages and data without knowing anything about the communication network

Why Http?

The protocol we need for information access must provide

- A subset of the file transfer functionality
- The ability to request an index search
- Automatic format negotiation.
- The ability to refer the client to another server

Where Http stand?



Many application layer protocols are used on the Internet, HTTP is only one

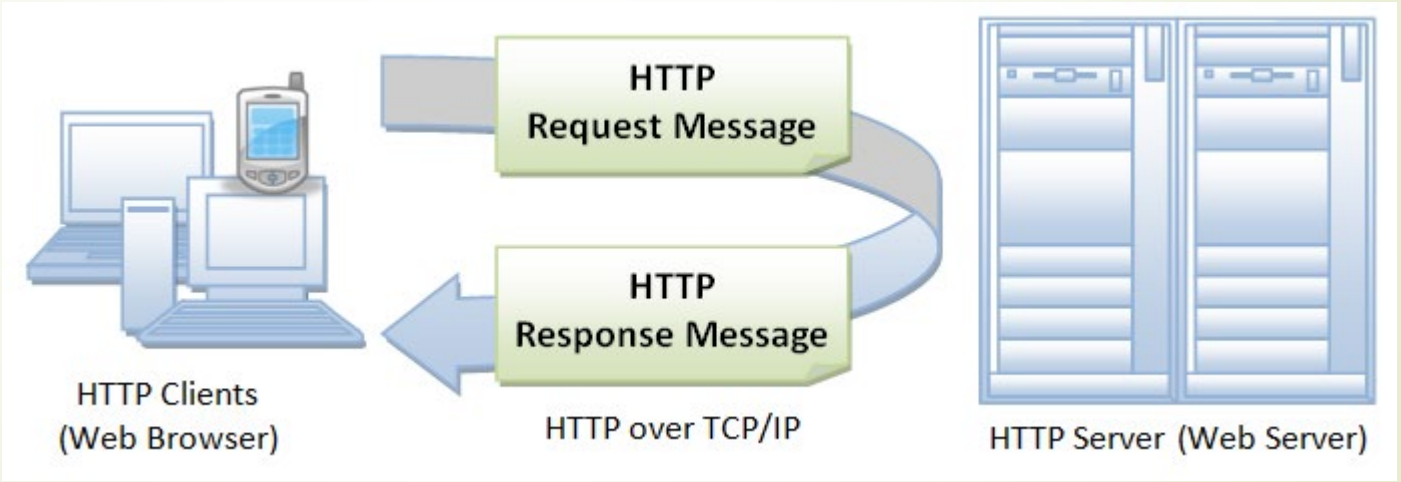
Protocol	Application
HTTP: Hypertext Transfer	Retrieve and view Web pages
FTP: File Transfer	Copy files from client to server or from server to client
SMTP: Simple Mail Transport	Send email
POP: Post Office	Read email

Http 1.0 vs Http 1.1

- Digest authentication
  - In HTTP/1.0, user sent username and password over the network
  - In HTTP/1.1, the client and the server never send the actual username or password over the network
- Persistent connections
  - In HTTP/1.0, if a single page includes inline images, multiple frames, animation, and other external references, to browse this page would require many reconnections
  - In HTTP/1.1 there are multiple request/response transactions per connection
  - Clients can pipeline requests to the server by sending multiple requests at start of session

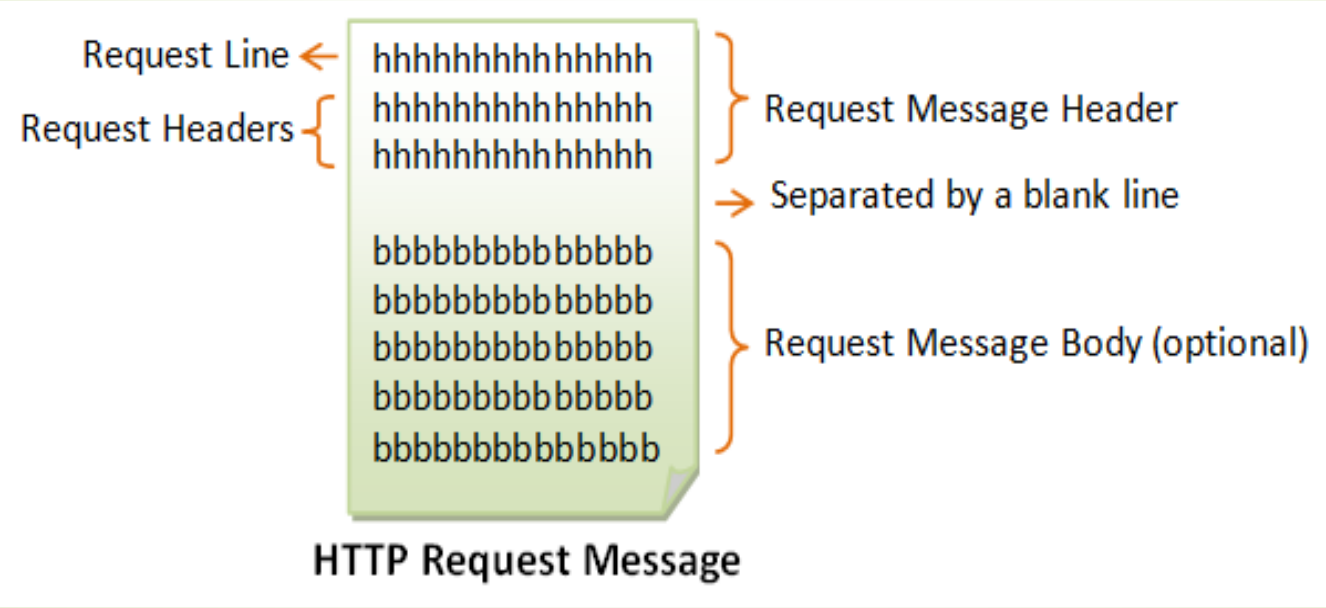
Request/Response formats

- HTTP client and server communicate by sending text messages. The client sends a *request message* to the server. The server, in turn, returns a *response message*.



HTTP Request Message

- The format of an HTTP request message is as follow:



Request Message

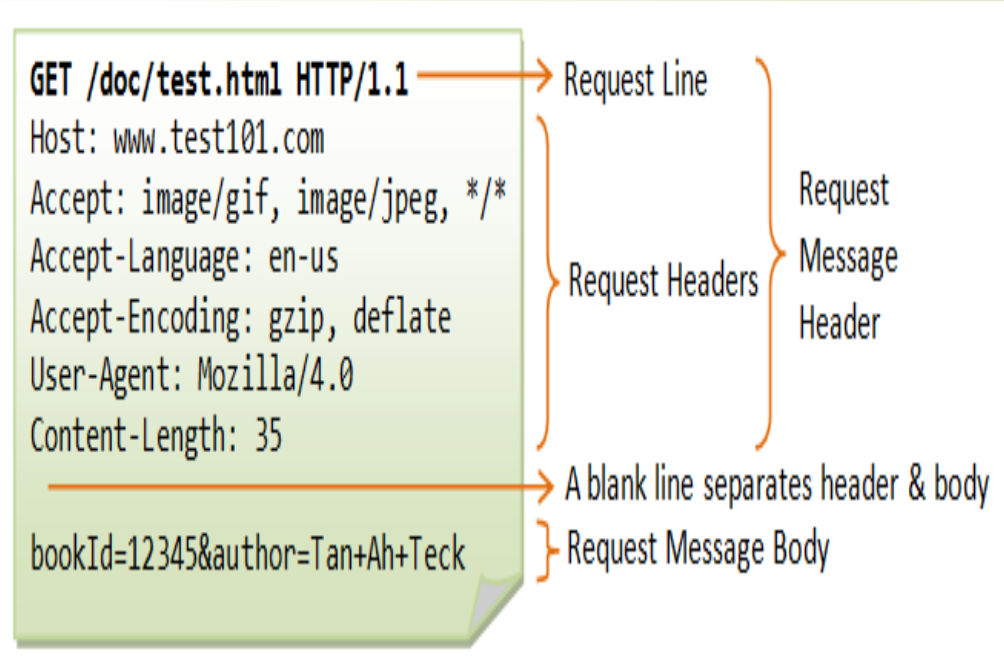
- Request Line:
  - The first line of the header is called the *request line*, followed by optional *request headers*.
  - The request line has the following syntax:

    Request-method-name request-URI HTTP-version

- Request Header:
  - The request headers are in the form of name: value pairs. Multiple values separated by commas, can be specified.
  - The request header has the following syntax:

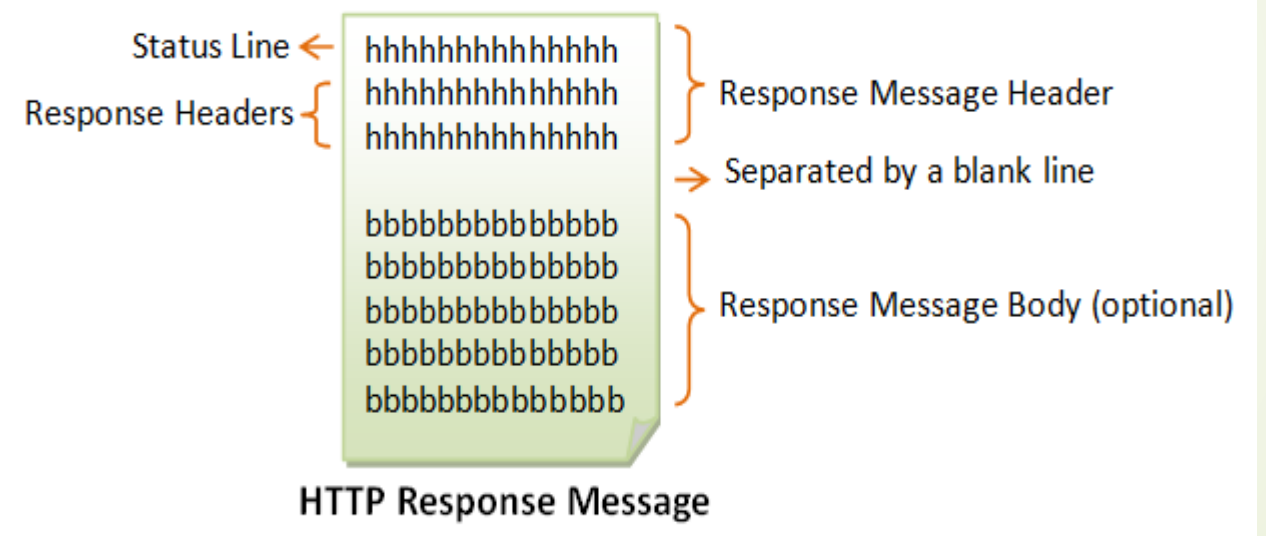
    Request-header-name: request-header-value1, request-header-value2, ...

- Sample Http Request message



HTTP Response Message

- The format of the HTTP response message is as follows:



Response Message

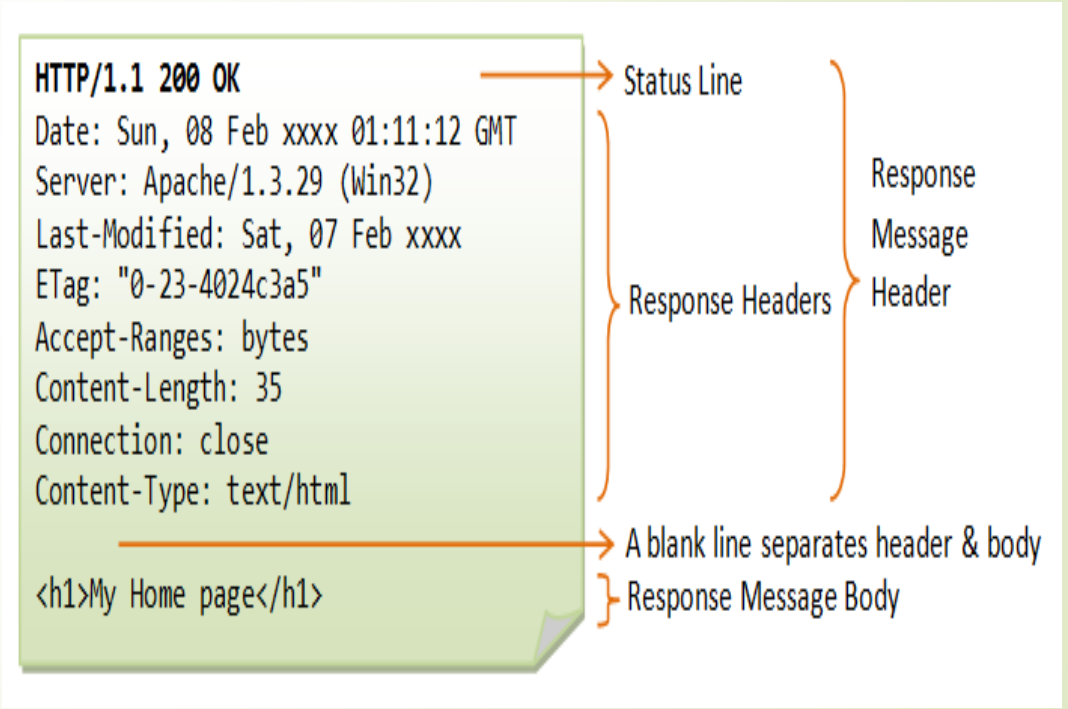
- Status Line:
  - The first line is called the *status line*, followed by optional response *headers*.
  - The request line has the following syntax:

    HTTP-version status-code reason-phrase

    - *HTTP-version*: The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
    - *status-code*: a 3-digit number generated by the server to reflect the outcome of the request.
    - *reason-phrase*: gives a short explanation to the status code.
- Response Header:
  - The response headers are in the form name: value pair
  - can be specified.
  - The request line has the following syntax:

    Response-header-name: response-header-value1, response-header-value2, ...

- Sample Http Response message

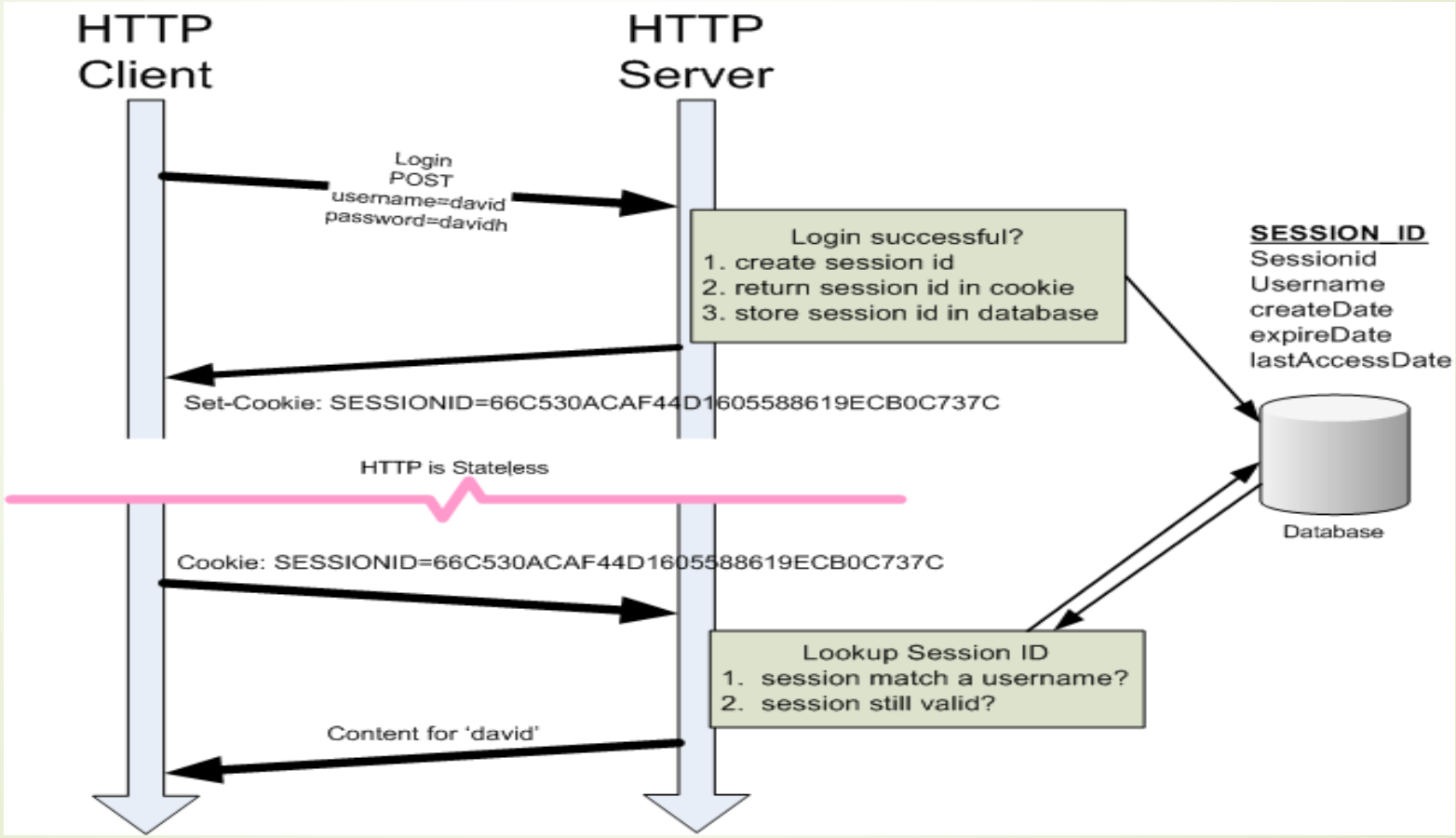


What is a Cookie?

- Quite simply, a cookie is a small text file that is stored by a browser on the user's machine. Cookies are plain text; they contain no executable code.
- A web page or server instructs a browser to store this information and then send it back with each subsequent request based on a set of rules.
- Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items added in the shopping cart in an online store) or to record the user's browsing activity.
- A web server specifies a cookie to be stored by sending an HTTP header called Set-Cookie.
- HttpOnly cookies can only be used when transmitted via HTTP (or HTTPS). They are not accessible through non-HTTP APIs such as JavaScript. This restriction eliminates the threat of cookie theft via cross-site scripting (XSS), while leaving the threats of cross-site tracing (XCT) and cross-site request forgery (CSRF) intact.

- Set-Cookie: name=Alok; expires=Sat, 02 March 2016 23:38:25 GMT

How Cookies Work?



HTTP Request Methods

- **GET**: A client can use the GET request to get a web resource from the server.
- **HEAD**: A client can use the HEAD request to get the header that a GET request would have obtained.
- **POST**: Used to post data up to the web server. For example, customer information, file upload, etc. using HTML forms
- **PUT**: Ask the server to store the data.
- **OPTIONS**: Ask the server to return the list of request methods it supports.

GET Method

- A **GET** request retrieves data from a web server by specifying parameters in the URL portion of the request.
- A client can use the **GET** request method to request (or "get") for a piece of resource from an HTTP server.
- The **GET** method is used to retrieve information from a specified URI. This means that the operation must have no side effects and GET requests can be re-issued without worrying about the consequences.
- One downside of **GET** requests is that they can only supply data in the form of parameters encoded in the URL (known as a **Query String**) or as cookies in the **cookie** request header. Therefore, **GET** cannot be used for uploading files or other operations that require large amounts of data to be sent to the server.

Post Method

- The POST method is used when you want to send some data to the server, for example, file update, form data, etc.
- This method packages the name/value pairs inside the body of the HTTP request, which makes for a cleaner URL.
- POST method is secured for passing sensitive and confidential information to server it will not visible in query parameters in URL and Parameters are not saved in browser history.
- No restrictions on data length. When we are reloading the browser should alert the user that the data are about to be re-submitted, post method cannot be bookmarked.

Put Method

- The PUT method is used to request the server to store the included entity-body at a location specified by the given URL.
- PUT uploads a new resource on the server. If the resource already exists and is different, it is replaced; if it does not exist, it is created.
- If a new resource is created, the origin server MUST inform the user agent via the 201 (Created) response.
- If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.
- If the resource could not be created or modified with the Request-URI, an appropriate error response SHOULD be given that reflects the nature of the problem.

Http Status Codes

- The first line of the response message (i.e., the status line) contains the response status code, which is generated by the server to indicate the outcome of the request.

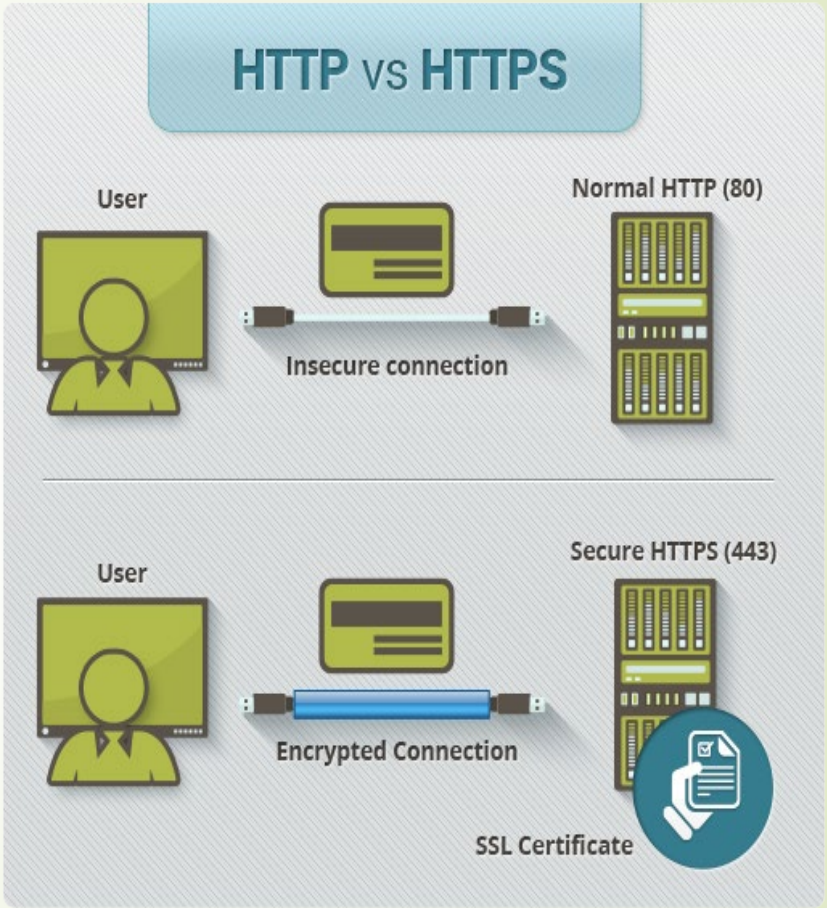
- The status code is a 3-digit number:
  - 1xx (Informational): Request received, server is continuing the process.
  - 2xx (Success): The request was successfully received, understood, accepted and serviced.
  - 3xx (Redirection): Further action must be taken in order to complete the request.
  - 4xx (Client Error): The request contains bad syntax or cannot be understood.
  - 5xx (Server Error): The server failed to fulfill an apparently valid request.

Caching

- HTTP supports caching so that content can be stored locally by the browser and reused when required. By carefully controlling caching, it is possible to reuse static content and prevent the storage of dynamic data.
- The goal of caching is to eliminate the need to send requests in many cases, and to eliminate the need to send full responses in many other cases.
- Browser caching is controlled by the use of the **Cache-Control**, **Last-Modified** and **Expires** response headers.
- The **Cache-Control** header allows a client or server to transmit a variety of directives in either requests or responses. These directives typically override the default caching algorithms.
- The caching directives are specified in a comma-separated list. For example: *Cache-control: no-cache*

HTTPS?

- Hyper Text Transfer Protocol Secure (HTTPS) is the secure version of HTTP, the protocol over which data is sent between your browser and the website that you are connected to. The 'S' at the end of HTTPS stands for 'Secure'.
- It means all communications between your browser and the website are encrypted. HTTPS is often used to protect highly confidential online transactions like online banking and online shopping order forms.



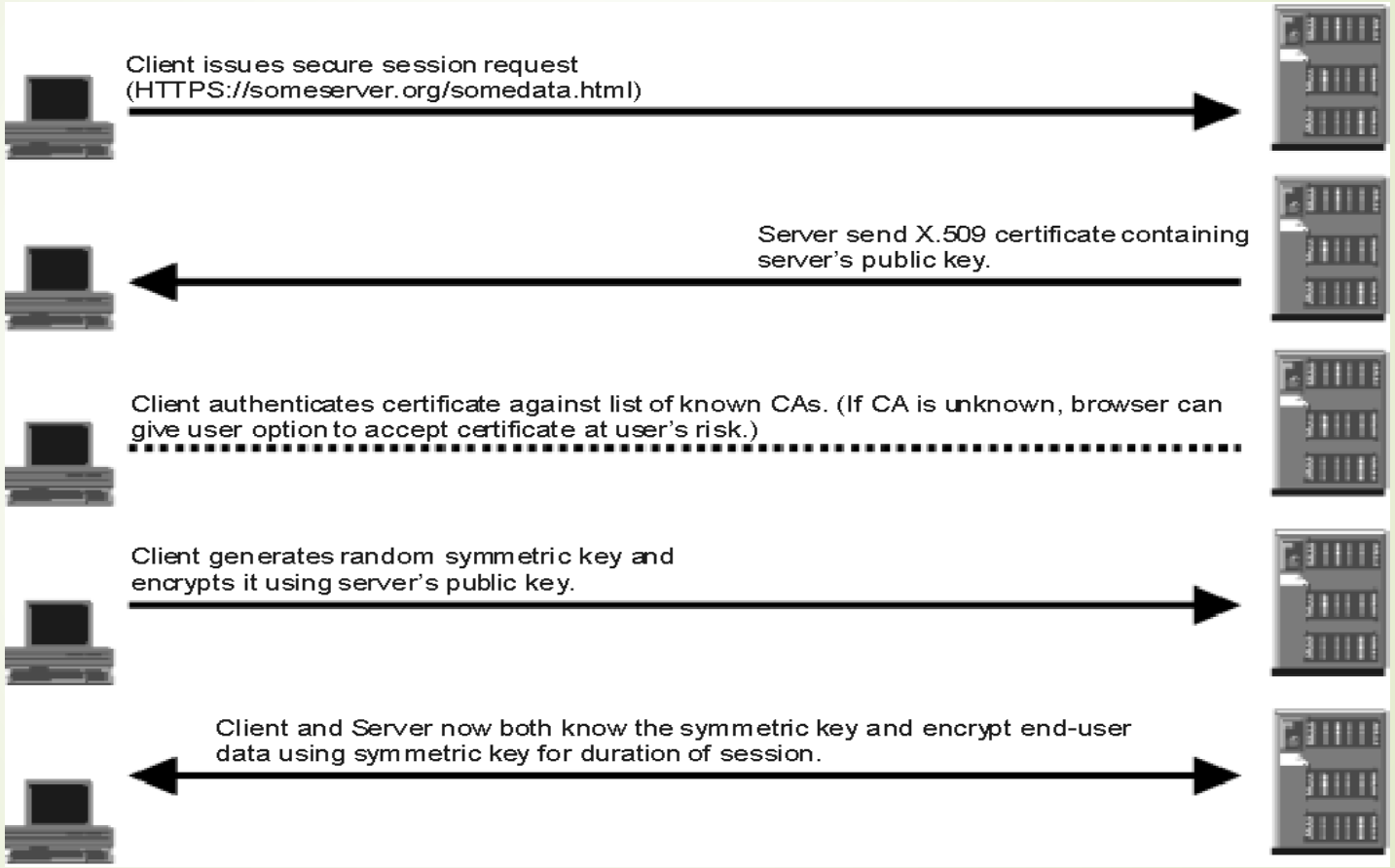
How Does HTTPS Work?

- HTTPS encrypts and decrypts user page requests as well as the pages that are returned by the Web server.
- HTTPS (HTTP over **SSL** or HTTP Secure) is the use of **Secure Socket Layer (SSL)** or Transport Layer Security (TLS) as a sublayer under regular HTTP application layering.
- SSL protocols use what is known as an 'asymmetric' Public Key Infrastructure (PKI) system. An asymmetric system uses two 'keys' to encrypt communications, a 'public' key and a 'private' key.
- Anything encrypted with the public key can only be decrypted by the private key and vice-versa.
- HTTPS and SSL support the use of X.509 digital certificates from the server so that, if necessary, a user can authenticate the sender.

What is a HTTPS certificate?

- When you request a HTTPS connection to a webpage, the website will initially send its SSL certificate to your browser.
- This certificate contains the public key needed to begin the secure session. Based on this initial exchange, your browser and the website then initiate the 'SSL handshake'.
- A consumer's browser begins the SSL handshake process by requesting a secure Web page using the HTTPS protocol.
- The SSL handshake involves the generation of shared secrets to establish a uniquely secure connection between yourself and the website.

SSL Handshake



Thank You!