**Name: Inayat Yousuf**
**Task : SSH, HTTP, HTTPS, APACHE SERVER.**

## SSH: **It is a crytographic network protocol for operting network servies securly over an unsecured newtowk. It is a secure alternative to the non protected login protocols like (telnet, rloign) and insure file tranfer methods like FTP. SSH uses client server architecture. It uses public key cryptography/ asymetric key crypotography to authenticate the remote server.**

The protocol is used in corporate networks for:

- providing secure access for users and automated processes
- interactive and automated file transfers
- issuing remote commands
- managing network infrastructure and other mission-critical system components.

### How does the SSH protocol work

The protocol works in the client-server model, which means that the connection is established by the SSH client connecting to the SSH server. The SSH client drives the connection setup process and uses public key cryptography to verify the identity of the SSH server. After the setup phase the SSH protocol uses strong symmetric encryption and hashing algorithms to ensure the privacy and integrity of the data that is exchanged between the client and server.

### SSH provides strong encryption and integrity protection

Once a connection has been established between the SSH client and server, the data that is transmitted is encrypted according to the parameters negotiated in the setup. During the negotiation the client and server agree on the symmetric encryption algorithm to be used and generate the encryption key that will be used. The traffic between the communicating parties is protected with industry standard strong encryption algorithms (such as AES (Advanced Encryption Standard)), and the SSH protocol also includes a mechanism that ensures the integrity of the transmitted data by using standard hash algorithms (such as SHA-2 (Standard Hashing Algorithm))

# Step 1 — Creating the Key Pair

The first step is to create a key pair on the client machine. This will likely be your local computer. Type the following command into your local command line:

```
ssh-keygen -t ed25519
```

## Step 2 — Specifying Where to Save the Keys

The first prompt from the `ssh-keygen` command will ask you where to save the keys:

you can choose any file name or location by typing it after the prompt and hitting `ENTER`.

## Step 3 — Creating a Passphrase

The second and final prompt from `ssh-keygen` will ask you to enter a passphrase: It's up to you whether you want to use a passphrase, but it is strongly encouraged: the security of a key pair, no matter the encryption scheme, still depends on the fact that it is not accessible to anyone else.

## Step 4 — Copying the Public Key to Your Server

Once the key pair is generated, it's time to place the public key on the server that you want to connect to.

You can copy the public key into the server's `authorized_keys` file with the `ssh-copy-id` command. Make sure to replace the example username and address:

**`ssh-copy-id filename@your_server_address`**

we can copy the file to the local computer to the remote servier.

**scp index.html inayat@address:filepath**

## What is HTTP?

The Hypertext Transfer Protocol (HTTP) is the foundation of the World Wide Web, and is used to load web pages using hypertext links. HTTP is an application layer protocol designed to transfer information between networked devices and runs on top of other layers of the network protocol stack. A typical flow over HTTP involves a client machine making a request to a server, which then sends a response message.

An HTTP request is the way internet communications platforms such as web browsers ask for the information they need to load a website.

Each HTTP request made across the Internet carries with it a series of encoded data that carries different types of information. A typical HTTP request contains:

1. HTTP version type
2. a URL
3. an HTTP method
4. HTTP request headers

5. Optional HTTP body.

**What's an HTTP method?**

An HTTP method, sometimes referred to as an HTTP verb, indicates the action that the HTTP request expects from the queried server. For example, two of the most common HTTP methods are 'GET' and 'POST'; a 'GET' request expects information back in return (usually in the form of a website), while a 'POST' request typically indicates that the client is submitting information to the web server (such as form information, e.g. a submitted username and password).

**What are HTTP request headers?**

HTTP headers contain text information stored in key-value pairs, and they are included in every HTTP request. These headers communicate core information, such as what browser the client is using what data is being requested.

Example of HTTP request headers from Google Chrome's network tab:

**Request Headers**

**:authority: www.google.com**

**:method: GET**

**:path: /**

**:scheme: https**

**:accept: text/html**

**accept-encoding: gzip, deflate, br**

**accept-language en-US, en; q=0.9**

## What's in an HTTP request body?

The body of a request is the part that contains the 'body' of information the request is transferring. The body of an HTTP request contains any information being submitted to the web server, such as a username and password, or any other data entered into a form.

# What's in an HTTP response?

An HTTP response is what web clients (often browsers) receive from an Internet server in answer to an HTTP request. These responses communicate valuable information based on what was asked for in the HTTP request.

A typical HTTP response contains:

1. an HTTP status code
2. HTTP response headers
3. optional HTTP body

**What's an HTTP status code?**

HTTP status codes are 3-digit codes most often used to indicate whether an HTTP request has been successfully completed. Status codes are broken into the following 5 blocks:

1. 1xx Informational
2. 2xx Success
3. 3xx Redirection
4. 4xx Client Error
5. 5xx Server Error

   The "xx" refers to different numbers between 00 and 99.

   Status codes starting with the number '2' indicate a success. For example, after a client requests a web page, the most commonly seen responses have a status code of '200 OK', indicating that the request was properly completed.

   If the response starts with a '4' or a '5' that means there was an error and the webpage will not be displayed. A status code that begins with a '4' indicates a client-side error (It's very common to encounter a '404 NOT FOUND' status code when making a typo in a URL). A status code beginning in '5' means something went wrong on the server side. Status codes can also begin with a '1' or a '3', which indicate an informational response and a redirect, respectively.

**What are HTTP response headers?**

Much like an HTTP request, an HTTP response comes with headers that convey important information such as the language and format of the data being sent in the response body.

**What's in an HTTP response body?**

Successful HTTP responses to 'GET' requests generally have a body which contains the requested information. In most web requests, this is HTML data which a web browser will translate into a web page

# HTTP flow

When a client wants to communicate with a server, either the final server or an intermediate proxy, it performs the following steps:

1. Open a TCP connection: The TCP connection is used to send a request, or several, and receive an answer. The client may open a new connection, reuse an existing connection, or open several TCP connections to the servers.
2. Send an HTTP message: HTTP messages (before HTTP/2) are human-readable. With HTTP/2, these simple messages are encapsulated in frames, making them impossible to read directly, but the principle remains the same.
3. Read the response sent by the server,
4. Close or reuse the connection for further requests.

## HTTP Messages

HTTP messages, as defined in HTTP/1.1 and earlier, are human-readable. In HTTP/2, these messages are embedded into a binary structure, a *frame*, allowing optimizations like compression of headers and multiplexing. Even if only part of the original HTTP message is sent in this version of HTTP, the semantics of each message is unchanged and the client reconstitutes (virtually) the original HTTP/1.1 request. It is therefore useful to comprehend HTTP/2 messages in the HTTP/1.1 format.

# Difference between http:// and https://

In address bar of a browser, have you noticed either *http://* or *https://* at the time of browsing a website? If neither of these are present then most likely, it's *http://* Let's find out the difference…

In short, both of these are protocols using which the information of a particular website is exchanged between Web Server and Web Browser. But what's difference between these two? Well, extra ***s*** is present in *https* and that makes it secure! What a difference A very short and concise difference between *http* and *https* is that *https* is much more secure compared to *http*.

**H**yper**T**ext **T**ransfer **P**rotocol (HTTP is a protocol using which hypertext is transferred over the Web. Due to its simplicity, *http* has been the most widely used protocol for data transfer over the Web but the data (i.e. hypertext) exchanged using *http* isn't as secure as we would like it to be. In fact, hyper-text exchanged using *http* goes as plain text i.e. anyone between the browser and server can read it relatively easy if one intercepts this exchange of data. But why do we need this security over the Web? Think of 'Online shopping' at Amazon or Flipkart. You might have noticed that as soon as we click on the Check-out on these online shopping portals, the address bar gets changed to use *https*. This is done so that the subsequent data transfer (i.e. financial transaction etc.) is made secure. And that's why *https* was introduced so that a secure session is setup first between Server and Browser. In fact, cryptographic protocols such as SSL and/or TLS turn *http* into *https* i.e. **https** = **http** + **cryptographic protocols**. Also, to achieve this security in *https*, Public Key Infrastructure (PKI) is used because public keys can be used by several Web Browsers while private key can be used by the Web Server of that particular website. The distribution of these public keys is done via Certificates which are maintained by the Browser. You can check these certificates in your Browser settings.

Also, another syntactic difference between *http* and *https* is that *http* uses default port 80 while *https* uses default port 443. But it should be noted that this security in *https* is achieved at the cost of processing time because Web Server and Web Browser needs to exchange encryption keys using Certificates before actual data can be transferred. Basically, setting up of a secure session is done before the actual hypertext exchange between server and browser.

**Differences between HTTP and HTTPS**

- In HTTP, URL begins with “http://” whereas URL starts with “https://”
- HTTP uses port number 80 for communication and HTTPS uses 443
- HTTP is considered to be unsecure and HTTPS is secure
- HTTP Works at Application Layer and HTTPS works at Transport Layer
- In HTTP, Encryption is absent and Encryption is present in HTTPS
- HTTP does not require any certificates and HTTPS needs SSL Certificates