**SSL**, or **Secure Sockets Layer**, is an **encryption-based** **Internet security protocol**. It was first developed by Netscape in 1995 for the purpose of ensuring **privacy**, **authentication**, and **data integrity** in Internet communications. SSL is the predecessor to the modern **TLS encryption** used today.

A website that implements **SSL/TLS** has **"HTTPS"** in its URL instead of "**HTTP**."

**Graphical user interface, application, timeline

Description automatically generated**

**How does SSL/TLS work?**

- In order to provide a high degree of privacy, SSL encrypts data that is transmitted across the web. This means that anyone who tries to intercept this data will only see a **garbled** mix of characters that is nearly impossible to decrypt.

- SSL initiates an authentication process called a **handshake** between two communicating devices to ensure that both devices are really who they claim to be.

- SSL also digitally signs data in order to provide data integrity, verifying that the data is not tampered with before reaching its intended recipient.

There have been several iterations of SSL, each more secure than the last. In 1999 SSL was updated to become TLS.

You will have install **SSL** **certificate** in order to protect your website. Follow the steps given in the link mentioned below:

[**https://sucuri.net/guides/how-to-install-ssl-certificate/**](https://sucuri.net/guides/how-to-install-ssl-certificate/)

**SSL handshake process**

An **SSL handshake** is a negotiation between two parties on a network, such as a **browser** and a **web server** in order to establish the details of their connection. It determines what version of **SSL/TLS** will be used in the session, which **cipher suite** will **encrypt communication**, verifies the server (and sometimes also the client), and establishes that a secure connection is in place before transferring data.

The client receives the server's **X. 509 digital certificate**. The client authenticates the server, using a list of known **certificate authorities**. The client generates a random **symmetric key** and encrypts it using the server's public key.

**To setup SSL.**

**SSL (Secure Sockets Layer)** is a **security protocol** used to establish an encrypted link between a **web server** and a **client browser**. **SSL** is commonly used to protect sensitive data such as login credentials, credit card numbers, and other personal information.

Here are the basic steps to set up SSL:

**Obtain an SSL Certificate:** To set up SSL, you first need to obtain an **SSL certificate** from a trusted **certificate authority (CA)**. The certificate will contain information about the domain name, the organization that owns the domain, and the **CA** that issued the certificate.

**Install the SSL Certificate:** Once you have obtained an SSL certificate, you need to install it on your **web server**. The process of installing the **SSL certificate** will vary depending on the **web server** software you are using.

**Configure your Web Server:** After installing the **SSL certificate**, you need to configure your **web server** to use SSL. This typically involves modifying your **web server's configuration file** to enable **SSL** and specify the location of the **SSL certificates.**

**Verify SSL is Working:** Once **SSL** is enabled on your **web server**, you should verify that it is working correctly. You can do this by accessing your website using **HTTPS** instead of **HTTP** and checking that the browser displays a padlock icon and a "Secure" message.

**Renew SSL Certificate:** **SSL certificates** have an expiration date, so it's important to renew them before they expire to ensure that your website remains secure.

Overall, setting up **SSL** involves obtaining an **SSL certificate**, installing it on your **web server**, configuring your web server to use **SSL**, and verifying that **SSL** is working correctly. Following these steps will help ensure that your website is secure and that sensitive data is protected.

**WEB SERVER**

**Web server software** is accessed through the domain names of websites and ensures the delivery of the site's content to the requesting user. The software side is also comprised of several components, with at least an **HTTP server**. The **HTTP server** is able to understand **HTTP** and **URLs**. As **hardwar**e, a **web server** is a computer that stores **web server software** and other files related to a website, such as **HTML documents**, images and **JavaScript** files.

When a **web browser**, like **Google Chrome** or Firefox, needs a file that's hosted on a **web server**, the browser will request the file by HTTP. When the request is received by the web server, the **HTTP server** will accept the request, find the content and send it back to the browser through **HTTP**.

More specifically, when a browser requests a page from a **web server**, the process will follow a series of steps. First, a person will specify a **URL** in a web browser's address bar. The **web browser** will then obtain the **IP address** of the domain name -- either translating the URL through DNS (Domain Name System) or by searching in its **cache**. This will bring the browser to a **web server**. The browser will then request the **specific file** from the **web server** by an **HTTP request**. The web server will respond, sending the browser the requested page, again, through **HTTP**. If the requested page does not exist or if something goes wrong, the web server will respond with an error message. The browser will then be able to display the webpage.

Multiple domains also can be hosted on one web server.

**CLIENT SERVER**

The **client-server** architecture is also termed as a network-computing structure because every request and their associated services are distributed over a network.

**So now the question is how the thing works?**

In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested, process it and deliver the data packets requested back to the client. One special feature is that the server computer has the potential to manage numerous clients at the same time. Also, a single client can connect to numerous servers at a single timestamp, where each server provides a different set of services to that specific client.

Only the server must have a **digital certificate** in **One-way SSL** to establish a secure connection.

Deploy an **Nginx (Port 80)** website over a valid domain name.

Next, you can set up Let's Encrypt SSL, a certificate authority that provides free **SSL/TLS** certificates.

You can also set up a **wildcard domain** and configure Let's Encrypt to understand **wildcard DNS** and **SSL**.

A **wildcard DNS record** is a **record** in a **DNS zone** that will match requests for non-existent domain names. A **wildcard DNS record** is specified by using a \* as the leftmost label (part) of a domain name, e.g. \*. example.com .

Another concept to learn is 𝗠𝘂𝘁𝘂𝗮𝗹 𝗧𝗟𝗦, which is common in most distributed systems. For example, Kubernetes components interact via **mTLS**.

Also, you will encounter this concept in **server mesh** implementations.

**mTLS (Mutual TLS)** is a security protocol that allows both client and server to authenticate each other using certificates.

To learn **Mutual TLS**, you can generate client and server certificates using **OpenSSL**.

Next, you can create a simple **Python Flask API** that requires SSL and a client program that makes a request to the **Flask API** using its certificates.

Finally, you can verify that the request succeeds and that the client and server exchange **SSL/TLS certificates**.

Diagram

Description automatically generated