When **HTTP** is used with **SSL/TLS** (encryption methods), it is called **HTTPS**, which stands for **"Secure HTTP"** . It's still the same HTTP protocol but with added **security**.

SSL: (Secure Sockets Layer)
TLS: (Transport Layer Security)

SSL and **TLS** are encryption protocols used to secure data sent over the internet. They make sure that information exchanged between a user's device (like a computer or smartphone) and a website is private and protected from hackers.

SSL is An older version of the protocol that creates a secure connection between two systems.
TLS is the newer and improved version of SSL, offering stronger security.

HTTPS usually runs on a different port **(port 443)** instead of the regular port used by HTTP (port 80).

SSL (the security layer) can be used in many applications, but it is most commonly used in **web browsers** to **secure** websites.

SSL and **TLS** are not very different from each other; they both work to secure data.

SSL was the original version, created by **Netscape**.
Netscape was a company that created one of the first popular web browsers called **Netscape Navigator** in the 1990s. It played a big role in the early development of the internet. Netscape also developed SSL (Secure Sockets Layer), which became one of the first widely-used methods for securing data on the web. Although **Netscape's** browser is no longer in use, it influenced many internet technologies we use today, including secure connections like **SSL/TLS**.

Later, **SSL** was updated and improved to enhance security features. During this update, the protocol was renamed **TLS** (Transport Layer Security) when it became a standardized method for securing data. Though the name changed, TLS is essentially an improved version of SSL. That's why we often refer to both as **SSL/TLS** when talking about secure internet connections.

SSL/TLS and cryptography

TLS/SSL uses public-key cryptography to verify the identities of the parties communicating. This ensures that the parties (like your browser and a website) are who they say they are. While this authentication can be optional, it is usually required for at least one party, typically the **server**.

After the authentication is successful, **symmetric key cryptography** is used to **encrypt the data** being transmitted, making sure the information shared between the two parties stays private and secure.

TLS/SSL Handshaking

After the **client** (like your browser) and **server** agree to use TLS, they go through a process called a **handshake** to set up a secure connection.

During this handshake, the client and server agree on various parameters used to establish the connection's security.

Once the **handshake** is complete, the secure data transmission can begin. This means all data exchanged between the client and server will be encrypted and protected.

Step 1 (Client to Server)

The handshake starts when the **client** (like your web browser) connects to a **TLS-enabled server** and requests a secure connection. This initial request is known as the **Client Hello** message.

In the **Client Hello** message, the client sends a **list of supported cipher suites**. A **cipher suite** is a specific set of encryption and key exchange algorithms that will be used to secure the connection.

Since different versions of different types of browsers support different cipher suites, the client includes its options to ensure the server can choose a common one for secure communication. This helps both parties agree on the best security settings for their connection.

Step 2 (Server to Client)

In response to the **Client Hello** message, the **server** reviews the list of cipher suites provided by the client and selects one that it also supports. This response is called the **Server Hello** message.

The server generally has a precedence list and chooses the highest one which is also supported by the client browser

Step 3 (Server to Client)

After agreeing on the cipher suite, the **server** sends its **digital certificate** to the client. This certificate serves as a form of identification for the server.

The **digital certificate** contains several important pieces of information:

- **Server Name**: This identifies the server and ensures that the client is connecting to the correct website.
- **Trusted Certificate Authority (CA)**: This is the organization that issued the certificate, confirming that it is valid and trustworthy.
- **Server's Public Encryption Key**: This key is used by the client to encrypt messages that can only be decrypted by the server, establishing secure communication.

Step 4 (Client)

The **client** verifies the **validity of the digital certificate** received from the server before proceeding with the connection.

To confirm its validity, the client checks:

Expiration: The certificate must be within its valid period and not expired.

Trusted Certificate Authority (CA): The certificate should be issued by a CA that the client trusts. This ensures the certificate's authenticity and the server's identity.

The certificate is specified by the X.509 standard

Step 5 (Pre-Master Secret)

After verifying the server's identity, the **client** generates a **Pre-Master Secret key**. This key is a randomly created value used to establish a secure session between the client and server.

The client **encrypts** the Pre-Master Secret using the **server's public key**, ensuring that only the server can decrypt it.

The client then sends the encrypted Pre-Master Secret to the server.

The **server** receives this encrypted message and uses its **private key** to decrypt the Pre-Master Secret. This step is crucial because it ensures that only the server can access the secret key needed for the upcoming secure communication.

Step 6 (Symmetric Encryption Key Generation)

Now that both the **client** and the **server** have the **Pre-Master Secret**, they use it to generate a **symmetric encryption key**.

Using mathematical algorithms(they agreed on before), both parties apply the same processes to the Pre-Master Secret to independently derive identical symmetric keys. These keys are used for **encrypting and decrypting data** during the session.

The symmetric encryption key ensures that all data transmitted between the client and server is encrypted and secure, making it unreadable to anyone else.

Step 7 (Client to Server)

Now that both the **client** and the **server** have the same **symmetric key**, the client sends a **Change Cipher** message to the server.

This message indicates that all future communication between the client and the server will use **symmetric encryption**, marking the end of the initial handshake that involved **asymmetric encryption**.

The **Change Cipher** message tells the server that the switch to symmetric encryption is about to begin, which is faster and more efficient for ongoing communication.

The client also sends a message to inform the server that it has **completed the handshaking protocol** and is ready to start secure data transmission.

Step 8 (Server to Client)

After receiving the **Change Cipher** message from the client, the **server** acknowledges it by sending a confirmation back to the client.

The server also sends a second message to confirm that the **handshaking protocol has successfully completed**.

Data Transmission

Once the **handshaking protocol** is completed, the secure **data transmission** begins between the client and the server.

As previously discussed, the encryption now switches to **symmetric encryption**, which is faster than the earlier **asymmetric encryption** used during the handshake process.

This means that **TLS/SSL** combines the best of both worlds:

- **Public-key (asymmetric) cryptography** is used during the handshake to securely exchange keys and verify identities.
- **Symmetric key cryptography** is used for the actual data transmission, providing both security and speed.