1. Breaking Down the URL and Checking It

When we type `google.com` into the browser's address bar and press Enter, the browser first needs to understand what we want. It breaks down the URL into parts: the protocol (HTTP or HTTPS), the domain name (`google.com`), and any extra details like paths or parameters. The browser also checks if the URL is written correctly. If something is wrong, it will show an error message.

2. Looking Up DNS Records in the Browser

The browser then checks its own storage to see if it already knows the IP address for `google.com`. This storage is called the cache. If the browser finds the address there, it uses it right away, so it doesn't need to look it up again.

3. Checking the Operating System's Cache

If the browser doesn't have the address, it asks the operating system for help. The OS also keeps a cache of DNS records for quick lookups. If the OS has the IP address stored, it sends it back to the browser.

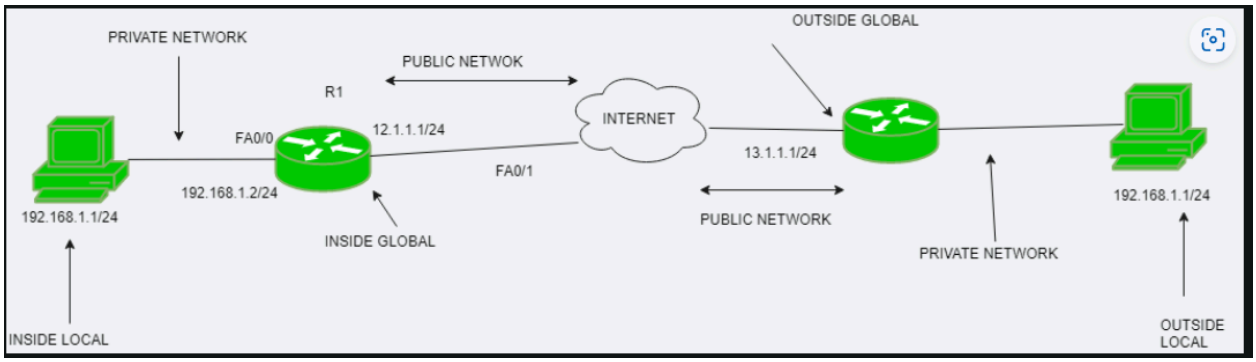
4. Asking the Router for the DNS Record

If the operating system doesn't have the address, the request goes to your router. The router also has a DNS cache. If it has the IP address, it will give it to the browser. If not, the router sends the request to the ISP's DNS server.

5. Mapping IP Addresses with NAT

When accessing a website like google.com from a local network, Network Address Translation (NAT) is responsible for managing IP addresses. A device within the network, which has a private IP address sends a request to the router. The router then replaces this private IP address with its own public IP address and assigns a unique port number. This mapping is recorded in the router's NAT table, which tracks the relationship between the private IP address and the public IP address along with the port number.

When the website google.com responds, it sends the data back to the router's public IP address and port. The router consults its NAT table to find the corresponding private IP address and port for this public IP and port combination. It then translates the address and forwards the data back to the correct device within the local network. This process allows multiple devices on a local network to share a single public IP address while keeping internal IP addresses hidden from the internet, thus enhancing security.



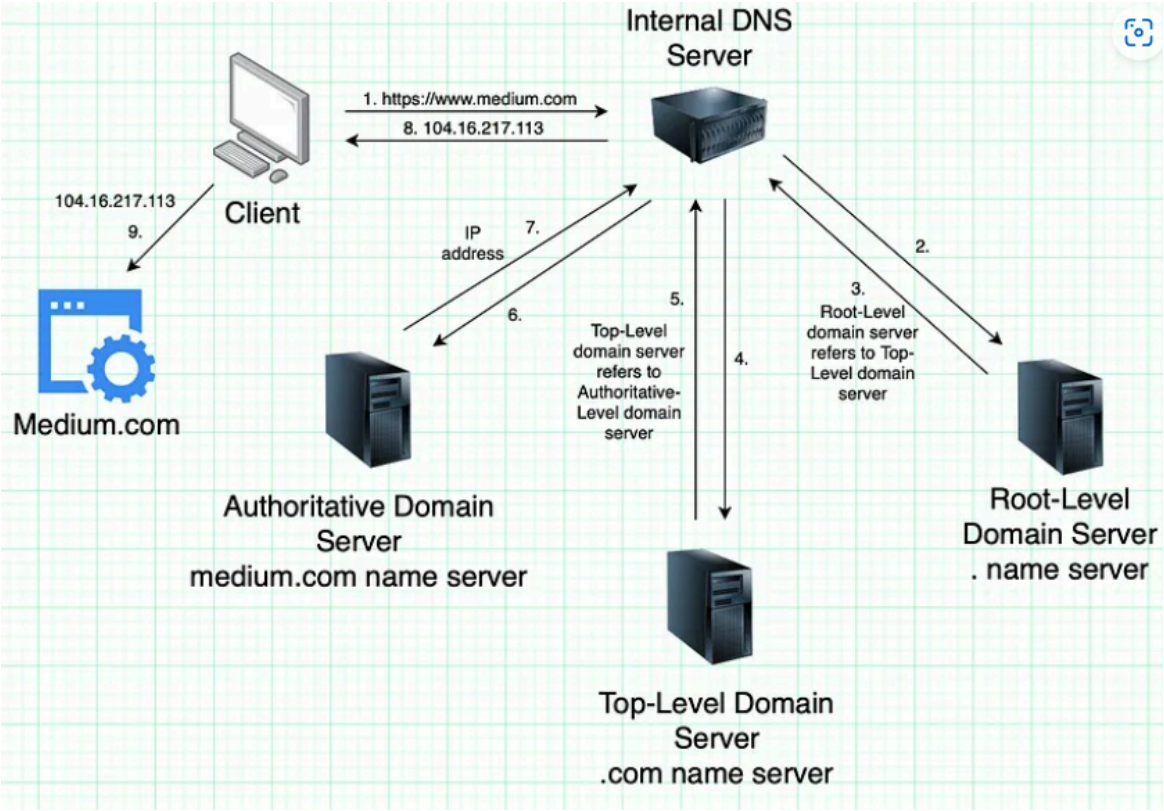
6. Finding the IP Address

If no one has the IP address yet, the ISP's DNS server will start looking for it. It asks several servers to find the answer:

- Asking the Root DNS Server: The ISP's server first asks a root DNS server. These are top-level servers that know where to find information about different domain endings like `.com`.

- Asking the TLD DNS Server: The root server tells the ISP's server to ask the TLD (Top-Level Domain) server for `.com`. This server knows where to find the authoritative servers for `.com` domains.

- Asking Google's DNS Server:The TLD server then directs the request to Google's own DNS server, which has the exact IP address for `google.com`.



7. Getting the IP Address

Google’s DNS server provides the IP address for `google.com`. This address is sent back through the servers to the ISP’s server, then to the router, and finally to the browser.

8. Sending the IP Address to the Browser

The ISP’s DNS server sends the IP address to the router. The router then passes this information to the operating system, and the operating system gives it to the browser.

9. Starting a TCP Connection

With the IP address now known, the browser starts a TCP connection with the server at that IP address. TCP (Transmission Control Protocol) helps ensure that data is sent accurately and in the correct order.

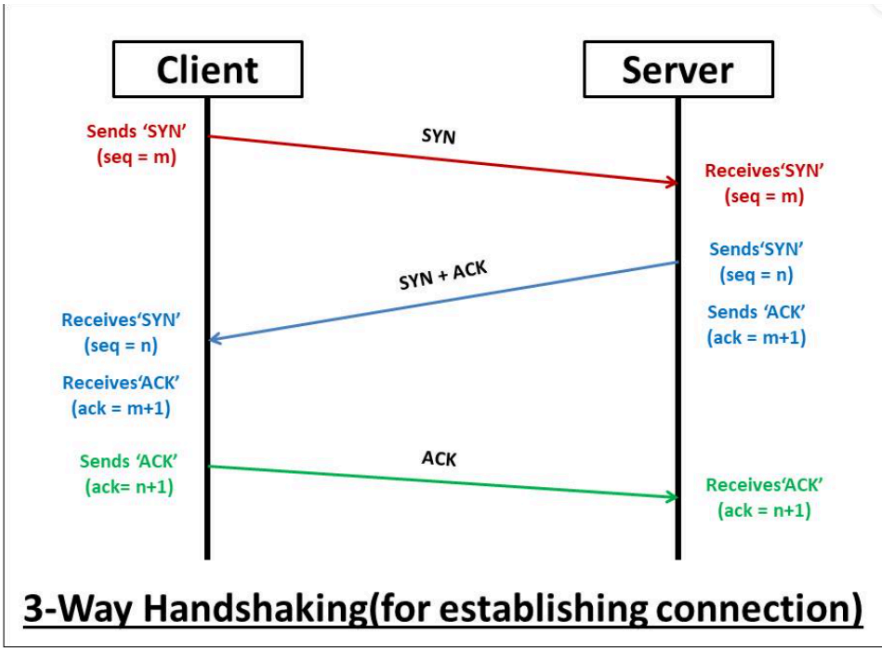
10. Updating the NAT Table

As the connection is being set up, the router updates its NAT table. This table keeps track of which private IP address and port on the network is communicating with the public IP address and port.

11. Establishing the Connection

The TCP connection is established through a process called the three-way handshake:

- SYN Packet: The browser sends a SYN (synchronize) packet to the server to start the connection, which includes a sequence number.
- SYN-ACK Packet: The server replies with a SYN-ACK (synchronize-acknowledge) packet, acknowledging your sequence number and providing its own.
- ACK Packet: Your browser sends an ACK (acknowledge) packet back to the server, finalizing the handshake and completing the connection setup.

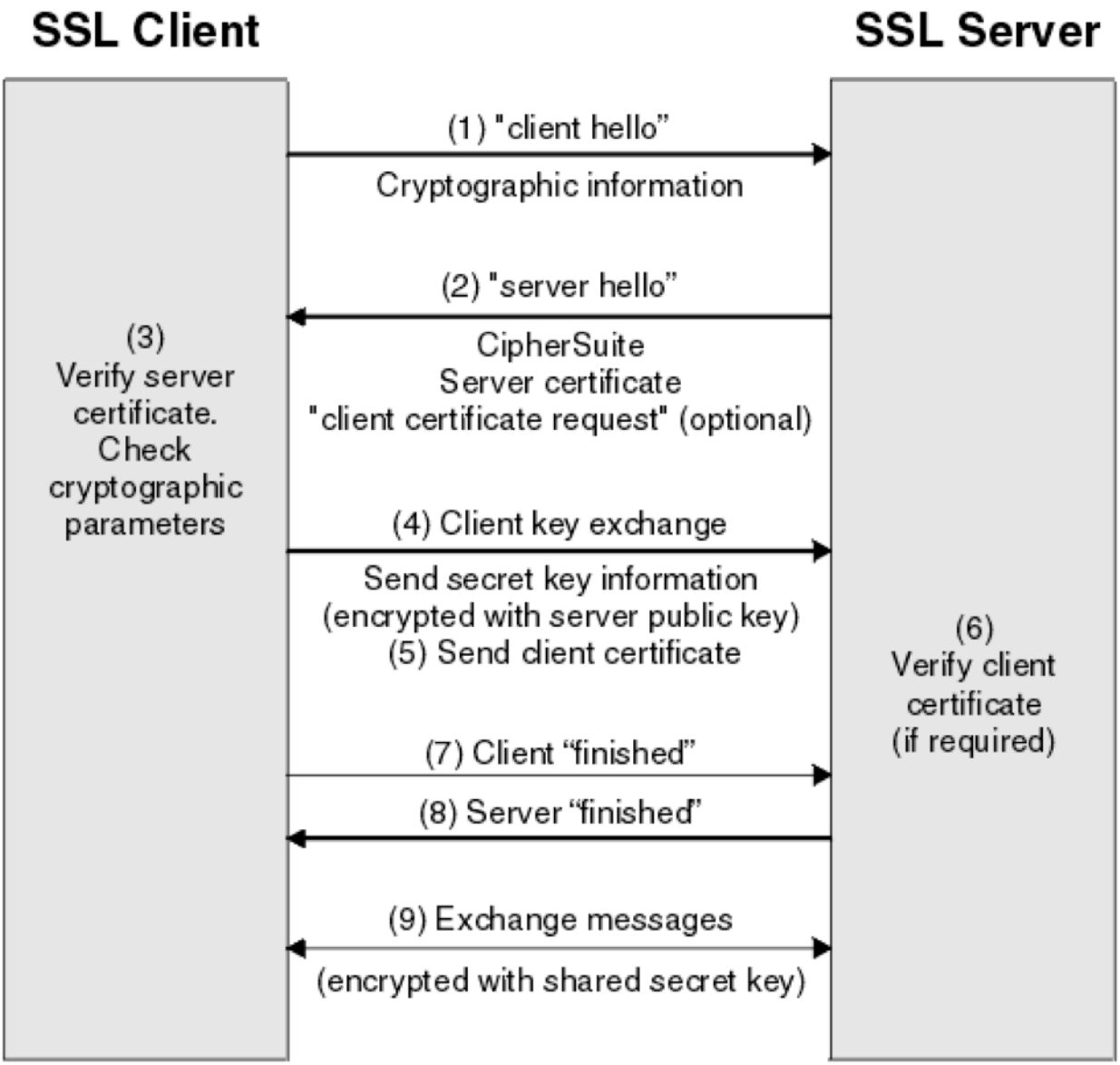


12. Connection is ready

Now that the TCP connection is set up, data can be reliably sent between the browser and the server.

13. Starting the TLS Handshake

To keep the data secure, the browser starts a TLS (Transport Layer Security) handshake. TLS encrypts the data so that no one else can read it.



- **Browser Sends "Hello"**

The browser sends a "Client Hello" message to the server. This message includes details about the encryption methods it supports and other settings.

- **Server Replies “Hello”**

The server responds with a “Server Hello” message. It includes the encryption method it has chosen, along with a digital certificate that verifies its identity.

- **Verifying the Certificate**

The browser checks the server’s certificate to make sure it is valid and comes from a trusted source. This step is crucial to ensure you are connecting to the real Google and not an impostor.

- **Certificate Validation Success**

If the certificate is valid, the TLS handshake continues. If not, the browser will end the connection.

- **Exchanging Keys and Switching to Encryption**
  - ➢ Key Exchange: Both the browser and the server exchange keys to encrypt the communication. The browser then sends a “Change Cipher Spec” message to switch to using encryption.
  - ➢ Change Cipher Spec and Finished: The server replies with its own “Change Cipher Spec” and “Finished” messages. This completes the handshake and secures the communication.

12. Secure Connection Established

With the TLS handshake complete, an encrypted channel is set up between the browser and the server. This ensures that all data sent between them is secure.

13. Sending the Request

The browser now sends a request to the server through this secure channel. For instance, it might be a request to load the homepage of `google.com`.

14. Receiving the Response

- Server Sends Back Data: The server processes the request and sends back an encrypted response with the webpage data.
- Browser Decrypts Data: The browser decrypts the response using the keys exchanged earlier.

15. Displaying the Webpage

- **Receiving the Response**: The browser retrieves the server's HTTP response, which may be encrypted when using HTTPS. If encryption is present, the browser decrypts the response as needed. This response contains HTML code along with additional components such as CSS and JavaScript files.
- **Parsing HTML**: The browser processes the HTML to understand the structure of the webpage, creating elements such as headings, paragraphs, images, and links.
- **Building the DOM**: As the HTML is parsed, the Document Object Model (DOM) is constructed, representing the content and structure of the webpage in a hierarchical tree.
- **Loading CSS**: The browser requests and applies CSS files referenced within the HTML. CSS is responsible for styling the webpage, including defining layout, colors, and fonts.
- **Processing JavaScript**: The browser runs any JavaScript included in the HTML or from external scripts, which adds interactivity and updates the DOM as needed.
- **Rendering the Page**: The browser calculates the layout for each element and draws them on the screen, forming the visible webpage.

- **Handling Additional Resources**: The browser makes separate requests for additional resources like images, videos, or fonts, incorporating these into the page as they are loaded.
- **Interactive Elements**: Once loaded, the page becomes interactive, responding to user inputs such as clicks and scrolls. JavaScript may continue to modify the page based on these interactions.
- **Final Rendering**: The browser completes the rendering process, enabling interaction with a fully loaded and functional webpage.