xName: Parshwa Herwade

PRN: 22510064

BTech Final Year CSE

Cryptography and Network Security Lab (B – 1)

Assignment 11 (Demonstration of SSL using Wireshark)

**Objectives:**

To analyze and understand how **SSL/TLS (Secure Sockets Layer / Transport Layer Security)** ensures secure communication over the internet by capturing and inspecting network packets using **Wireshark**. This includes identifying the SSL/TLS handshake process, encryption mechanisms, and certificate exchange.

**Problem Statement:**Secure communication over the internet is essential to protect data from eavesdropping, tampering, or forgery. SSL/TLS protocols provide encryption, authentication, and integrity for data transmitted between a client and a server.

In this task, you are required to:

1. Set up a secure HTTPS connection (e.g., by visiting an HTTPS-enabled website using a browser).
2. Capture network traffic using Wireshark while the SSL/TLS handshake and data exchange occur.
3. Identify and analyze the following in Wireshark:
   * SSL/TLS handshake process (Client Hello, Server Hello)
   * Server certificate exchange
   * Key exchange and cipher suite negotiation
   * Session keys and encrypted data packets
4. Highlight the encrypted nature of HTTPS traffic and explain what information is still visible (e.g., IP addresses, port numbers, SNI).
5. Optionally, use browser developer tools or import a private key (if available) to decrypt SSL traffic and inspect actual HTTP data inside Wireshark.

The goal is to demonstrate how SSL/TLS protects data during transmission and to gain familiarity with analyzing encrypted traffic using Wireshark.

**Equipment / Tools:**

* **Wireshark** (latest version)
* **Google Chrome / Mozilla Firefox**
* **Operating System:** Windows / Linux / macOS
* **Internet Connection** with access to an HTTPS website  
  (e.g., https://wikipedia.org or any other secure domain)

**Theory:**

Modern secure communication over the Internet relies on a combination of **TCP** and **TLS (Transport Layer Security)**.  
When you access a secure website (HTTPS), two handshakes occur sequentially:

1. **TCP Three-Way Handshake** – establishes a reliable connection.
2. **TLS Handshake** – negotiates encryption, authentication, and integrity parameters.

These steps ensure that both parties can communicate securely, privately, and reliably.

The **TCP three-way handshake** is used to establish a reliable, ordered, and error-checked communication channel between a client and a server.

**Purpose**

* Synchronize sequence numbers between client and server.
* Verify both sides are ready to send and receive data.
* Transition both sides into the **ESTABLISHED** state.

**Transition to TLS Handshake**

After the TCP connection is established, the **TLS handshake** begins.  
TLS (formerly SSL) ensures **confidentiality**, **integrity**, and **authenticity** of data transmitted over TCP.

The **TLS handshake** negotiates encryption parameters, authenticates the server (and optionally the client), and establishes session keys for secure communication.

**Main Steps of the TLS Handshake**

|  |  |  |
| --- | --- | --- |
| Step | Message | Description |
| 1 | Client Hello | Client initiates TLS. Sends supported **TLS versions**, list of **cipher suites**, **compression methods**, and random nonce. |
| 2 | Server Hello | Server selects the TLS version and one **cipher suite** from the list offered by the client. Sends its own random nonce. |
| 3 | Server Certificate | Server sends its **digital certificate** (usually X.509 format), issued by a trusted **Certificate Authority (CA)**. This proves the server’s identity. |
| 4 | Server Key Exchange | Sent when using ephemeral or Diffie–Hellman-based key exchange. Contains parameters for key negotiation. |
| 5 | Server Hello Done | Indicates the server is done with initial setup. |
| 6 | Client Key Exchange | Client sends its **session key material** (encrypted with the server’s public key, if RSA) or performs a **Diffie–Hellman exchange**. |
| 7 | Change Cipher Spec | Both client and server signal that subsequent communication will be encrypted. |
| 8 | Finished Message | Each side verifies that the handshake was successful using the newly established session keys. |

**Cipher Suites**

A **cipher suite** defines the algorithms used for encryption, key exchange, and message authentication during a TLS session.

**Categories of Cipher Suites**

1. **Key Exchange Algorithms**: RSA, ECDHE, DHE
2. **Encryption Algorithms**: AES, ChaCha20, 3DES
3. **Message Authentication Codes (MACs)**: SHA-256, SHA-384
4. **Protocol Version**: TLS 1.2, TLS 1.3, etc.

**Certificates**

Certificates provide **authentication** — they confirm that the server is who it claims to be.

**Key Concepts**

* A certificate is an **X.509** structure that includes:
  + Server’s **public key**,
  + **Issuer (CA)** details,
  + **Validity period**,
  + **Common Name (CN)** or **Subject Alternative Name (SAN)** — usually the website domain.
* Certificates are **digitally signed** by a trusted **Certificate Authority (CA)** such as DigiCert, Let's Encrypt, or GlobalSign.
* The client verifies:
  + Certificate signature validity,
  + Certificate’s expiration,
  + Hostname match.

If all checks pass, the client trusts the server.

**Key Exchange and Encryption**

Depending on the chosen cipher suite:

* **RSA** key exchange → client encrypts a random pre-master secret using the server’s public key.
* **Diffie–Hellman (DHE/ECDHE)** → both sides derive a shared secret using ephemeral parameters.

From this shared secret, session keys for:

* **Encryption**,
* **Integrity (MAC)**, and
* **Session resumption (optional)**  
  are generated.

**TLS Session Establishment**

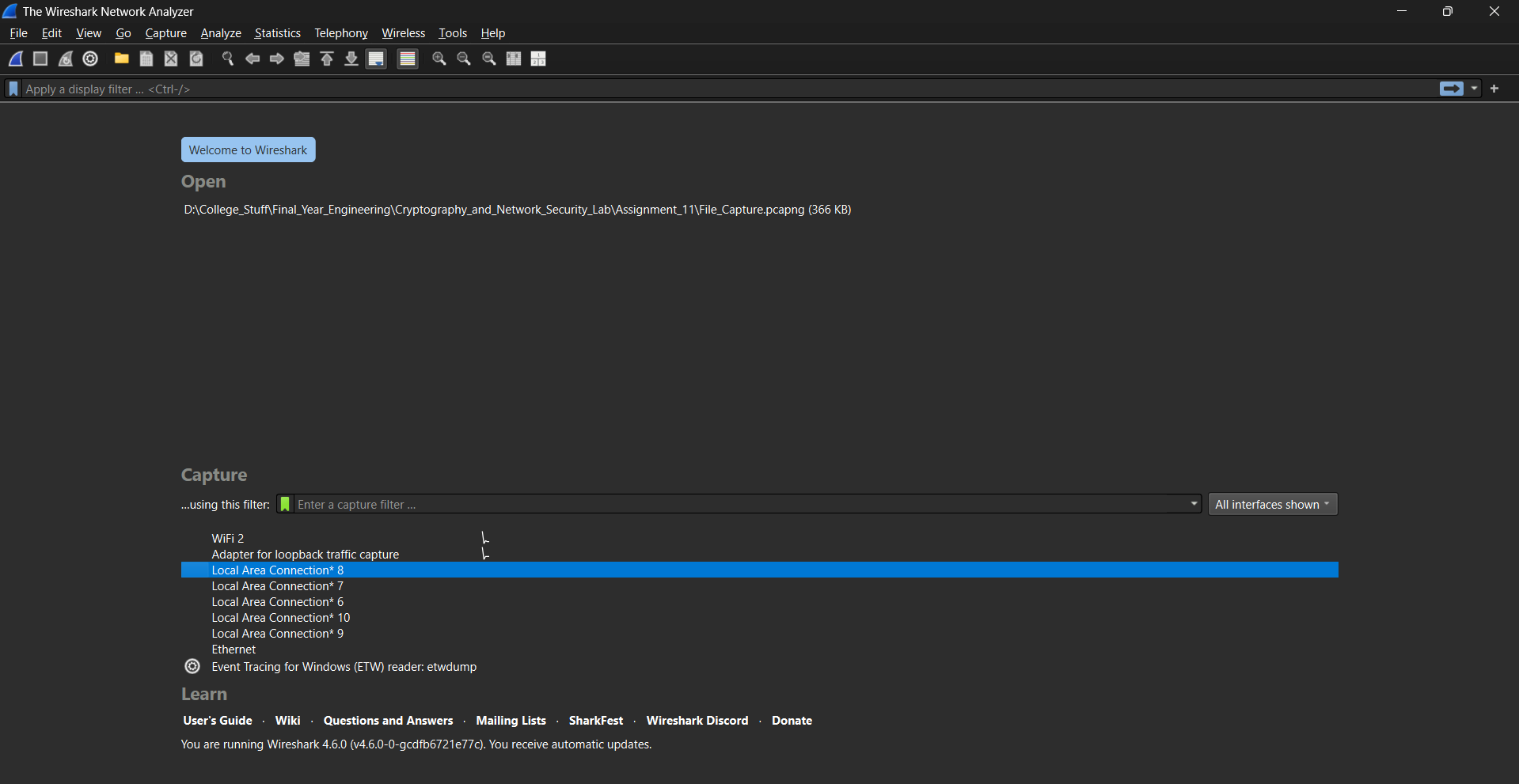
Once both sides send **Change Cipher Spec** and **Finished** messages:

* Both confirm that encryption is now active.
* All future packets are encrypted under the negotiated cipher suite.

**Procedure:**

**Step 1: Install Wireshark**

1. Download Wireshark from the official website:  
   <https://www.wireshark.org/download.html>
2. Choose the correct version for your operating system (Windows / macOS / Linux).
3. During installation, **enable “WinPcap” or “Npcap”** (for Windows) — this is required for packet capture.
4. Once installed, open **Wireshark**.



**Step 2: Verify HTTPS Connectivity**

1. Open your browser (Chrome / Edge / Firefox).
2. Visit a secure website such as:
   * https://www.google.com, or
   * https://www.wikipedia.org
3. Confirm the **“lock” icon** in the address bar — this means HTTPS (TLS) is active.

**Step 3: Start Packet Capture**

1. In Wireshark, select your **active network interface** (usually Wi-Fi or Ethernet).
2. Click **Start Capturing Packets (blue shark fin icon)**.
3. Keep Wireshark running in the background.

**Step 4: Initiate HTTPS Connection**

1. While Wireshark is capturing, visit any HTTPS website again (e.g., https://example.com).
2. This triggers the **TCP handshake** followed by the **TLS handshake**.
3. Wait for the page to load fully (this ensures full TLS negotiation and data exchange are captured).

**Step 5: Stop the Capture**

1. Go back to Wireshark.
2. Click the **red square (Stop Capture)** button.

**Step 6: Apply Filters**

To focus on HTTPS/TLS packets: tls

**Step 7: Locate TCP Handshake**

Filter: tcp.flags.syn == 1

Then scroll through the following 2–3 packets:

* **SYN** (client → server)
* **SYN-ACK** (server → client)
* **ACK** (client → server)

**Step 8: Locate TLS Handshake**

Filter for TLS handshakes: tls.handshakes

You will see packets such as:

* **Client Hello**
* **Server Hello**
* **Certificate**
* **Server Hello Done**
* **Client Key Exchange**
* **Change Cipher Spec**
* **Finished**

**Step 9: Examine Certificate Exchange**

1. In the packet list, find **Server Certificate** packet.
2. Expand *“Transport Layer Security” → “Handshake Protocol: Certificate”*.
3. Expand *“Certificate” → “Certificate Details”*.
4. Observe fields:
   * Subject (server name)
   * Issuer (Certificate Authority)
   * Validity period
   * Public key

**Step 10: Identify Key Exchange**

Find **Client Key Exchange** packet:

* For RSA: the “Pre-Master Secret” is encrypted using the server’s public key.
* For ECDHE: the client and server exchange elliptic curve public parameters.

**Step 11: Observe “Change Cipher Spec”**

This marks the transition from unencrypted to encrypted communication.

**Step 12: Confirm Encrypted Data Exchange**

After handshake completion, the next packets will show:

* Protocol: **TLSv1.2 / TLSv1.3**
* Info: **Application Data**

These packets are encrypted — you will not see readable HTTP content.

**Observations:**

**TCP Handshake Observation**

**Observation:**

* The capture shows three packets involved in establishing a TCP connection between the client and server:
  1. **SYN** (Client → Server)
  2. **SYN-ACK** (Server → Client)
  3. **ACK** (Client → Server)
* The client initiates the connection by sending a SYN packet containing its **initial sequence number (ISN)**.
* The server responds with a SYN-ACK, acknowledging the client’s ISN and providing its own ISN.
* Finally, the client sends an ACK confirming the connection establishment.
* After these three steps, the connection state transitions to **ESTABLISHED**, enabling higher-layer protocols like TLS to start communication.

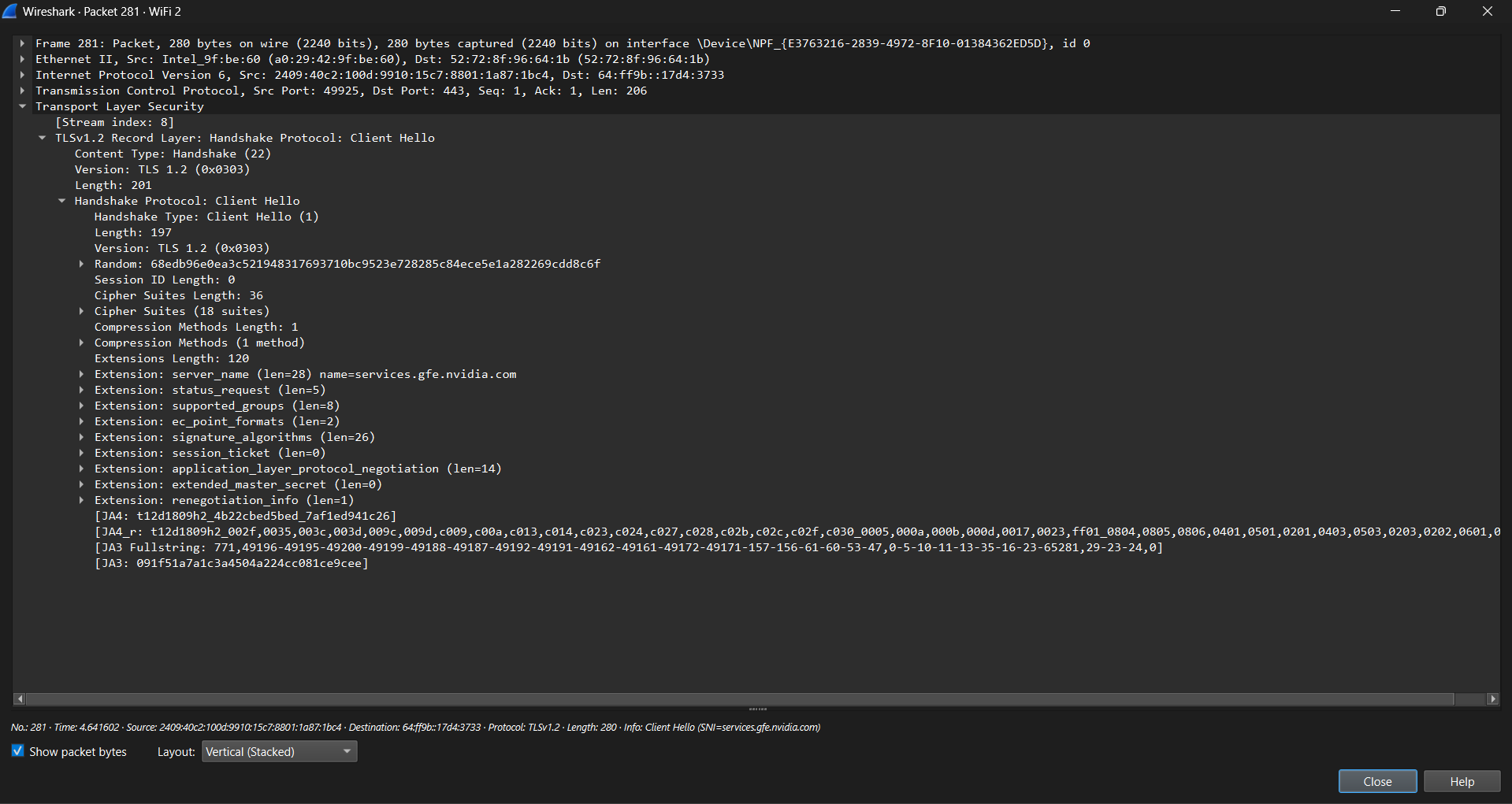
**TLS Handshake Observation**

After the TCP session is established, the TLS handshake begins.

**(a) Client Hello**

**Observation:**

* The client initiates the TLS handshake by sending a **Client Hello** message.
* The packet includes:
  + **Supported TLS versions** (e.g., TLS 1.2, TLS 1.3)
  + **List of Cipher Suites**
  + **Compression methods**
  + **Random number**
  + **SNI (Server Name Indication)** extension indicating the target domain name.

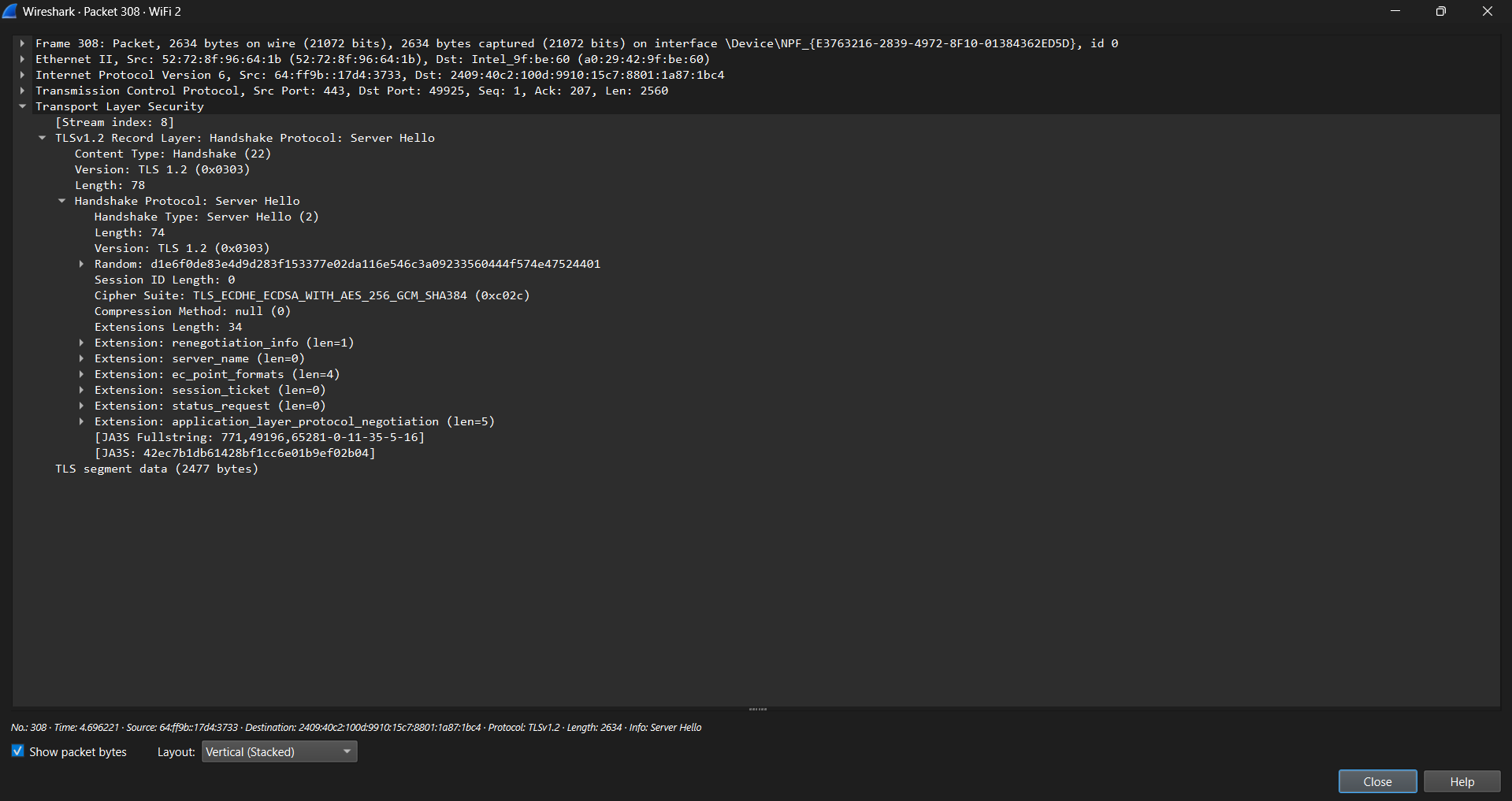


**Inference:**  
The client proposes security parameters and identifies the server it wants to connect to.

**(b) Server Hello**

**Observation:**

* The server replies with a **Server Hello** message.
* It includes:
  + The **chosen TLS version** (e.g., TLS 1.3)
  + The **selected Cipher Suite** from the client’s list
  + The server’s random value.

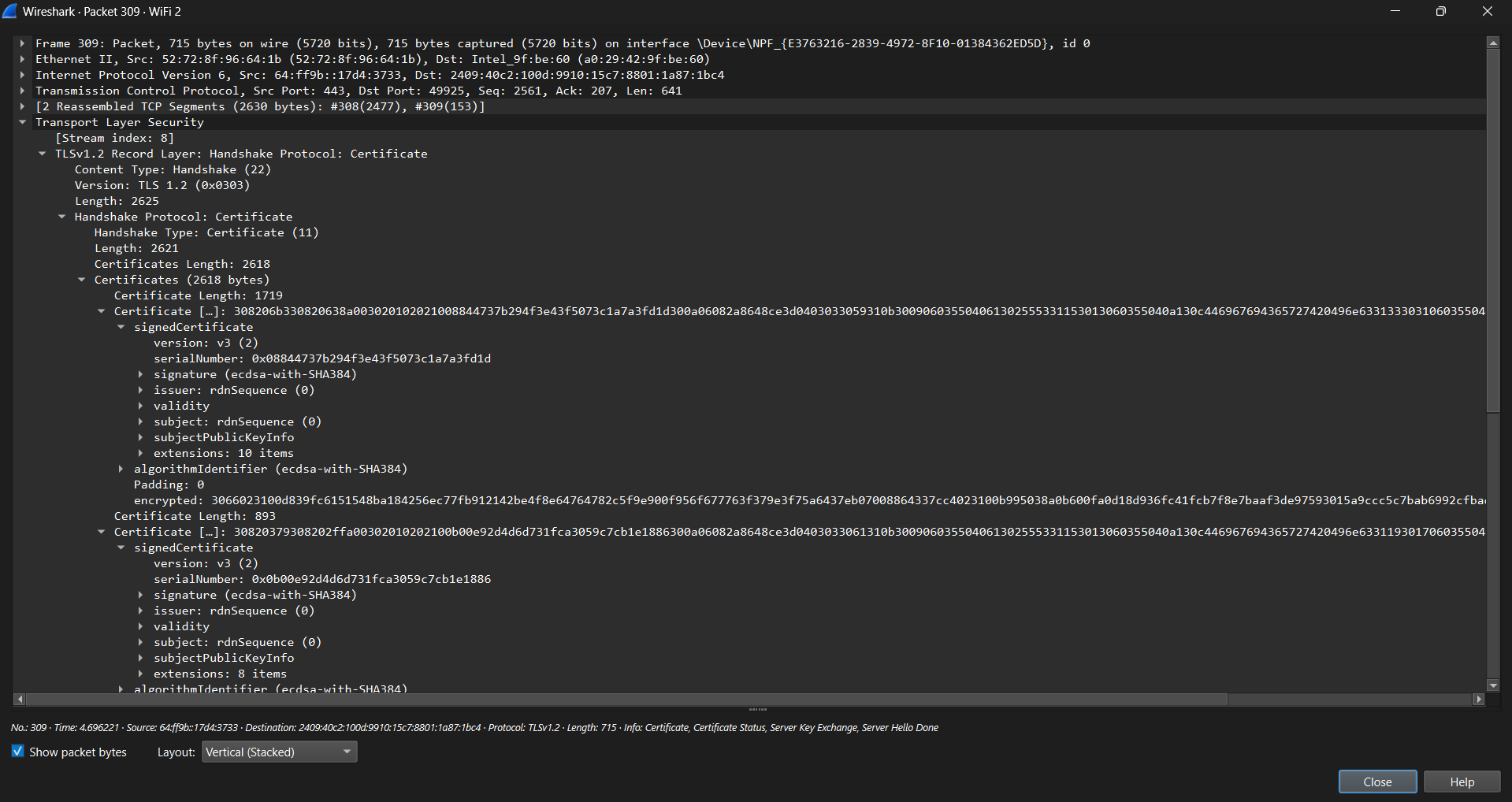


**Inference:**  
The server finalizes the encryption parameters, ensuring both client and server agree on a single cipher suite for the session.

**(c) Server Certificate**

**Observation:**

* The server sends its **digital certificate** (X.509 format).
* The certificate details include:
  + **Subject:** The website’s domain name
  + **Issuer:** Certificate Authority (CA)
  + **Validity period:** Start and expiry dates
  + **Public Key information**



**Inference:**  
The certificate authenticates the server, assuring the client that it is communicating with a legitimate and trusted endpoint.

**(d) Server Hello Done**

**Observation:**

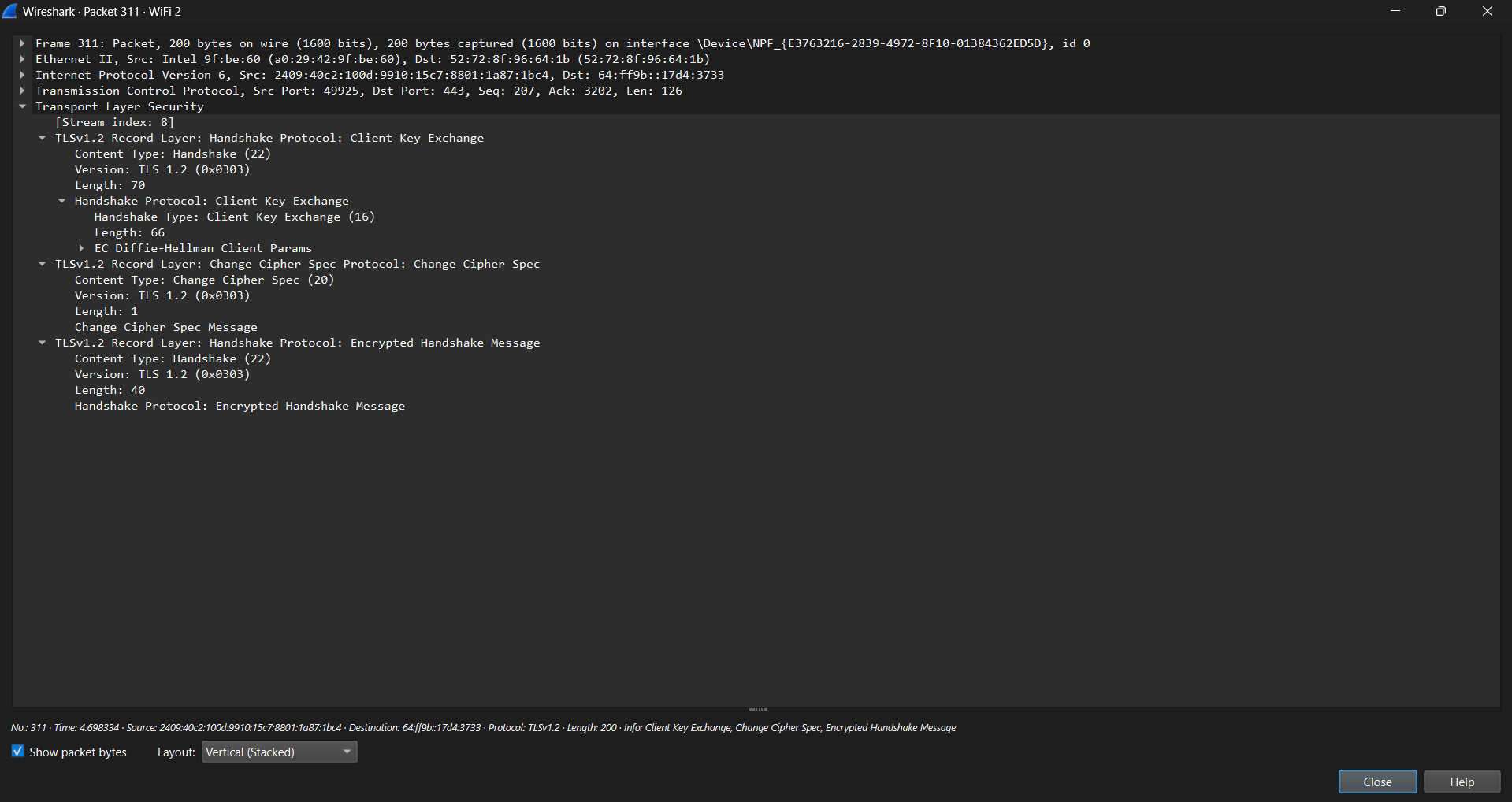
* This packet indicates that the server has completed sending its initial handshake messages.

**Inference:**  
It signals the client to proceed with key exchange

**(e) Client Key Exchange**

**Observation:**

* The client sends its **Client Key Exchange** message.
* Depending on the cipher suite:
  + If **RSA** is used, the **pre-master secret** is encrypted using the server’s public key.
  + If **ECDHE/DHE** is used, the client sends its **public key parameters** for Diffie–Hellman exchange.



**Inference:**  
Both client and server use this information to compute the same **session key**, which will be used for encryption.

**(f) Change Cipher Spec**

**Observation:**

* Both client and server send a **Change Cipher Spec** message to signal that subsequent data will be encrypted using the negotiated session key and cipher suite.

**Inference:**  
This marks the transition from plaintext handshake messages to encrypted communication.

**(g) Finished Messages**

**Observation:**

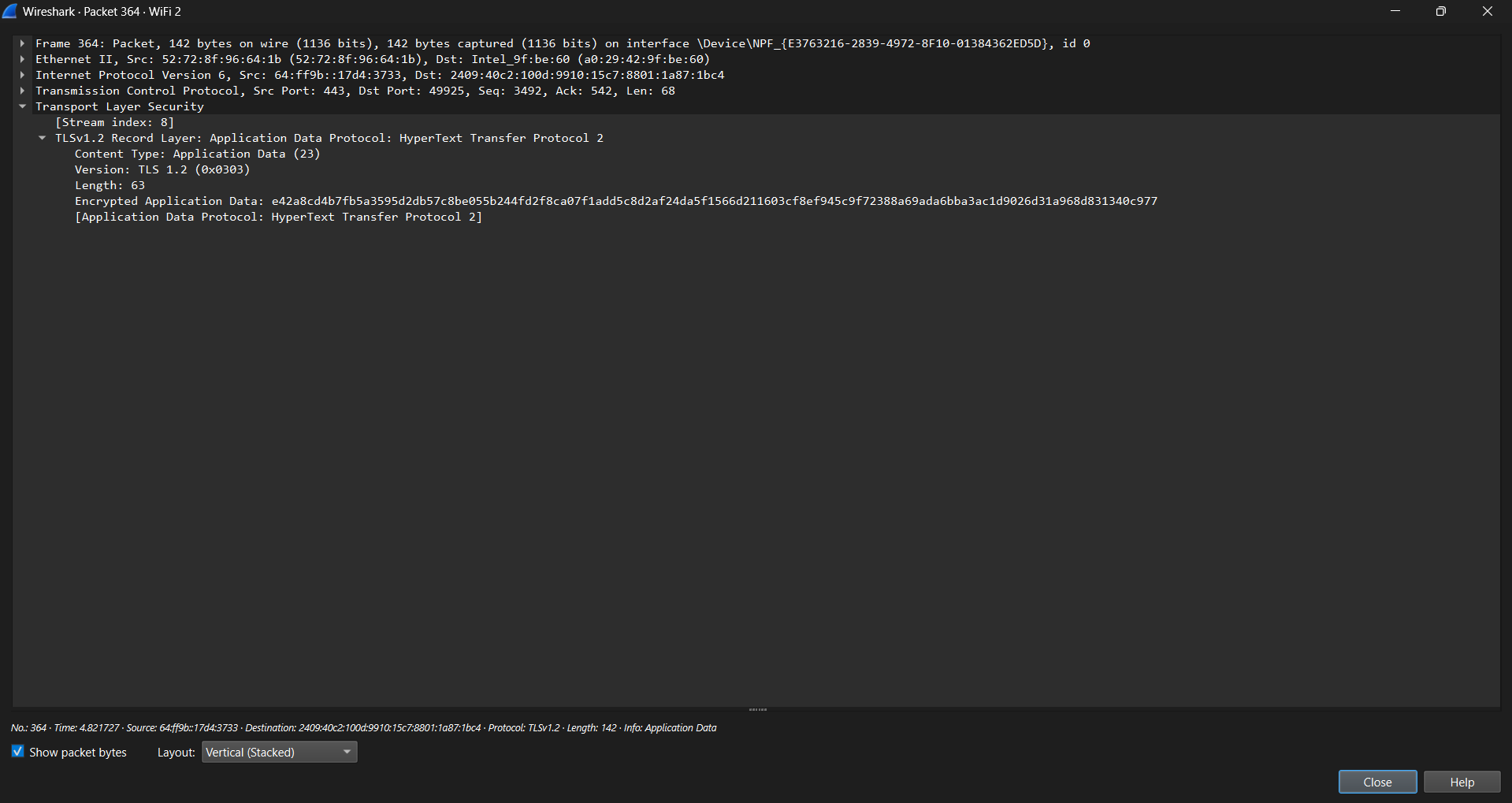
* The **Finished** message is sent by both parties to verify that the handshake was completed successfully.
* It is encrypted with the new session key, confirming that both sides derived the same key material.

**Inference:**  
This concludes the TLS handshake. A secure encrypted channel is now active.

**Encrypted Application Data**

**Observation:**

* Following the handshake, the packets now display:
  + **Protocol:** TLSv1.2 / TLSv1.3
  + **Info:** Application Data
* The payload contents are encrypted and unreadable.
* Wireshark indicates these packets as “Application Data,” confirming encryption.



**Inference:**  
Actual HTTP content (GET/POST requests, webpage data, etc.) is now protected by encryption and cannot be viewed without decryption keys.

**Visible Metadata in Encrypted Communication**

**Observation:**

Even though the data is encrypted, certain fields remain visible in Wireshark:

* **Source and Destination IP addresses**
* **Port numbers** (usually 443 for HTTPS)
* **TLS Version**
* **Cipher Suite used**
* **SNI (Server Name Indication)** — reveals the target domain name (in plaintext inside Client Hello)

**Inference:**  
While TLS hides the content, some metadata is still visible. This allows routing but can also be analyzed for network visibility or censorship.

**Conclusion:**

Through this experiment, we successfully analyzed how **SSL/TLS (Secure Sockets Layer / Transport Layer Security)** ensures secure communication over the internet by encrypting data exchanged between a client and a server. Using **Wireshark**, we captured and inspected the HTTPS traffic to observe the **TLS handshake**, which included the **Client Hello**, **Server Hello**, **certificate exchange**, **cipher suite negotiation**, and **session key establishment** phases.

The analysis demonstrated that:

* SSL/TLS uses **asymmetric cryptography** during the handshake to securely establish a shared session key.
* Once the session key is established, all further communication is **encrypted using symmetric encryption**, providing both speed and security.
* Sensitive data such as **login credentials, form submissions, or messages** were encrypted and thus not visible in Wireshark, confirming data confidentiality.
* Only limited metadata such as **IP addresses**, **port numbers**, **Server Name Indication (SNI)**, and **packet sizes/timings** were visible, which cannot reveal the actual contents of communication.

This demonstrates that **SSL/TLS effectively protects data integrity, confidentiality, and authenticity**, making it the backbone of secure communication over the web. Wireshark proved to be a valuable tool to understand and visualize this secure communication process in action.