Day 3: Ethical Hacking and Cyber Forensics @IITK

Network Forensics with Wireshark and Cryptography Fundamentals

Dr. Renu Mary Daniel

June 11, 2025

Contents

1 Introduction to Wireshark

Wireshark is an indispensable tool in network forensics and analysis. It is a free and open-source network protocol analyzer, widely used for understanding network traffic at a granular level.

- **Functionality:** Wireshark allows users to capture live network traffic and then analyze it in detail, providing deep insights into how data traverses a network.
- **Value:** Its capabilities make it extremely valuable for various tasks, including troubleshooting network issues, investigating security incidents, and gaining a comprehensive understanding of network behavior.

The image below visually summarizes these key aspects of Wireshark, highlighting its role as a network protocol analyzer for detailed packet-level insights.

1.1 Cleansheet

You can download it from here `https://www.comparitech.com/net-admin/wireshark-cheat-sheet/`

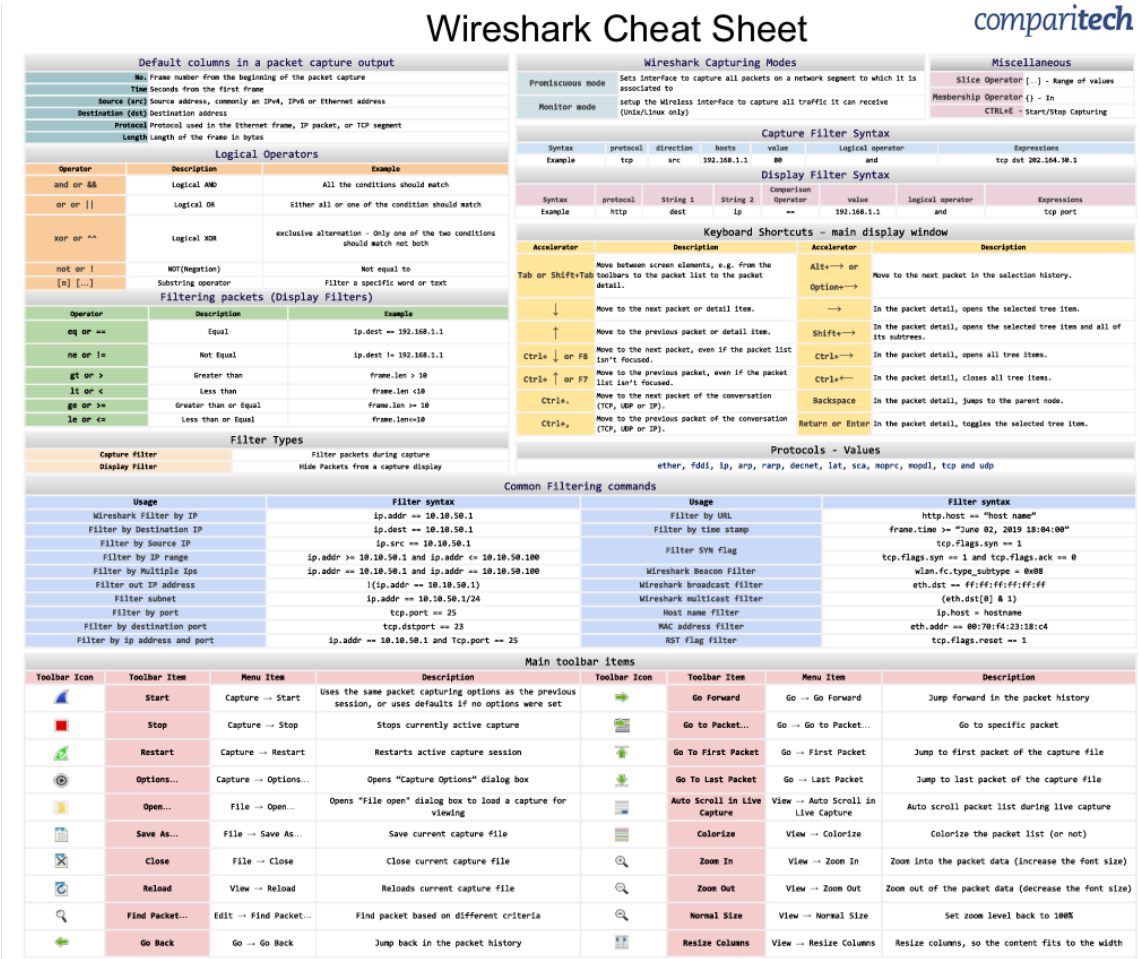

Figure 1: Wireshark Cheat Sheet

2 Wireshark Setup and Packet Capture Scenarios

Before diving into traffic analysis, understanding how Wireshark captures packets is crucial. Network traffic can be captured in various scenarios, often utilizing a **SPAN (Switched Port Analyzer) port** or a **mirror port**. This configuration duplicates traffic from one or more source ports to a dedicated monitor port where your capturing device (e.g., a PC running Wireshark) is connected.

The diagram below illustrates a common setup for network traffic analysis using a SPAN port, where traffic from multiple client ports is copied to a traffic analyzer.

Alternatively, packet capture can be performed using tools like ‘tcpdump‘ or ‘tshark‘ on a dedicated monitoring PC or an IDS (Intrusion Detection System) server for passive monitoring. The following figure shows

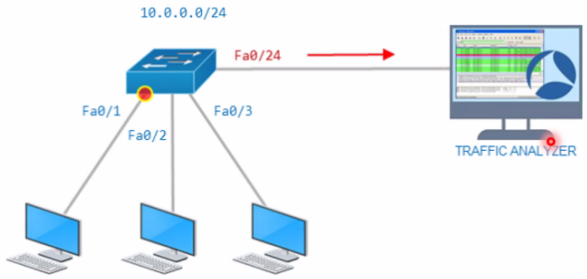

Figure 2: SPAN Port Setup for Traffic Analysis

two primary setups for packet capture, one with a dedicated monitoring PC and another integrating an IDS server.

Figure 3: Packet Capture Scenarios

2.1 Starting and Stopping Capture

The process of starting and stopping packet capture in Wireshark is straightforward using the toolbar icons.

To **start capturing**, click on the blue fin icon, typically located in the toolbar or accessible via 'Capture ¿ Start'. The first image highlights this option in the Wireshark interface.

To **stop capturing**, click on the red square icon, which pauses the live capture and displays the collected packets. The second image points to the stop capture icon.

3 Wireshark Filters

Filters are essential for narrowing down the vast amount of captured traffic to focus on specific packets. Wireshark supports two main types of filters: **Capture Filters** and **Display Filters**.

- **Capture Filters:** Applied before capturing, they restrict which packets are saved to the capture file. Example: 'port 80' to capture only HTTP traffic.

- **Display Filters:** Applied after capturing, they hide packets that don't match the criteria, without discarding them from the capture file. This allows for flexible analysis post-capture.

3.1 Applying Display Filters

Display filters are powerful for on-the-fly analysis. Here are some examples:

To filter packets by a specific IP address, use 'ip.addr == ¡IP_address¿'. The first image shows applying the filter 'ip.addr == 172.16.193.142' in Wireshark.

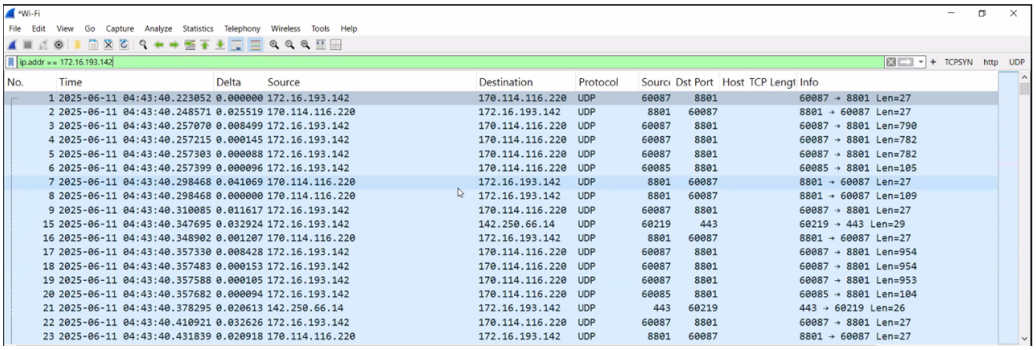

Figure 4: Display Filter: 'ip.addr == 172.16.193.142'

To filter by a source IP address, use 'ip.src == ¡IP_address¿'. The second image demonstrates filtering traffic where '172.16.193.142' is the source.

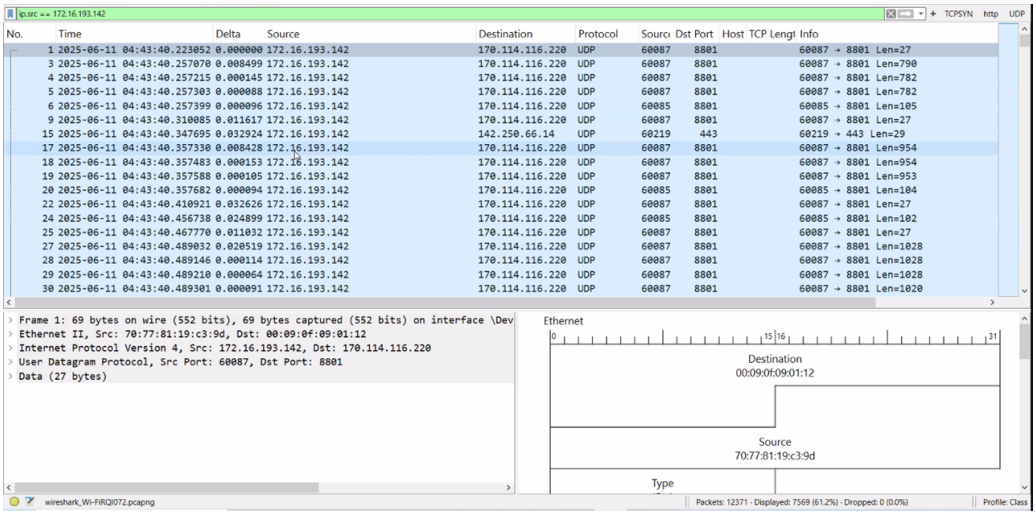

Figure 5: Display Filter: 'ip.src == 172.16.193.142'

4 Customizing Wireshark Columns

Customizing columns in Wireshark can significantly improve visibility and efficiency during analysis. You can add, remove, and reorder columns to display information relevant to your investigation.

4.1 Adding Custom Columns

To add new columns, navigate to 'Edit ¿ Preferences ¿ Appearance ¿ Columns'. The first image shows the Wireshark preferences window, specifically the "Columns" section, where you can manage displayed columns.

For example, you can add 'dst.port' (Destination port) and 'src.port' (Source port) to easily view destination port numbers. This is demonstrated in the second image.

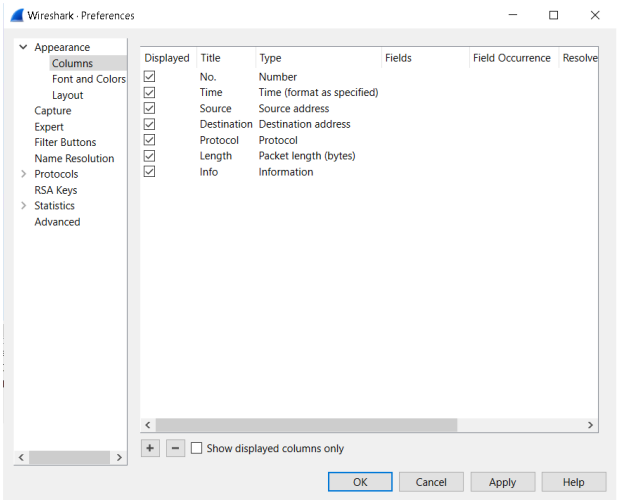

Figure 6: Wireshark initial Column Preferences

Similarly, adding ‘delta‘ (Delta time) can help in analyzing time differences between packets, as shown in the third image.

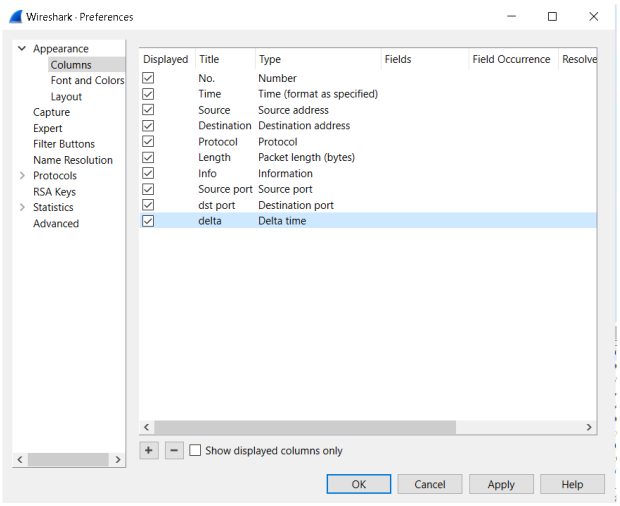

Figure 7: Adding ’delta’ Column

Once added, these columns provide quick insights directly in the main packet display, as observed in the capture below with the ”Delta” column.

4.2 Changing Layout and Reordering Columns

Wireshark’s display layout can be adjusted in ‘Edit ¿ Preferences ¿ Appearance ¿ Layout‘. This allows you to arrange the packet list, packet details, and packet bytes panes according to your preference. The image illustrates the layout options, allowing you to choose how the different Wireshark panes are arranged.

Columns can also be reordered easily by dragging and dropping them in the column headers, further customizing your view for efficient analysis. Observe how columns can be clicked and dragged to different positions, as indicated by the arrows in the screenshot.

5 Wireshark Coloring Rules

Coloring rules visually highlight packets based on specific criteria, making it easier to spot interesting or problematic traffic at a glance. Wireshark comes with predefined rules, and you can create your own. The screenshot displays the ”Coloring Rules” preferences in Wireshark, showing a list of rules and their associated filters.

Common examples include:

- **Bad TCP:** ‘tcp.analysis.flags && !tcp.analysis.window_update‘ - to identify TCP issues.

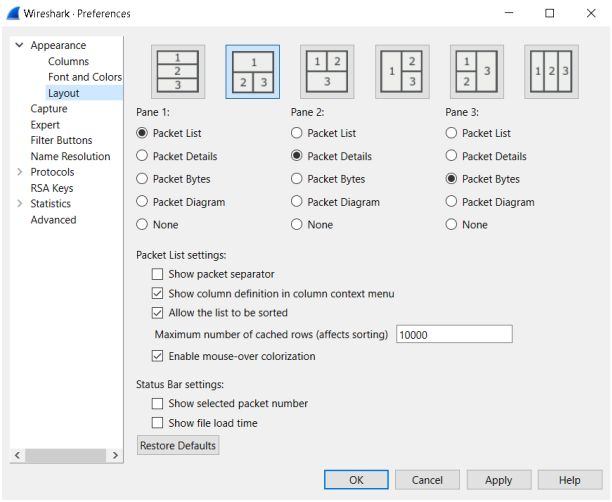

Figure 8: Wireshark Layout Preferences

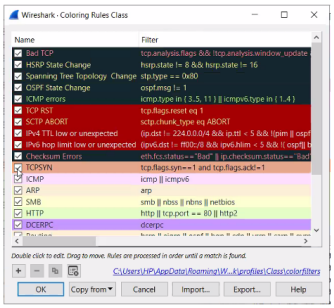

Figure 9: Wireshark Coloring Rules Class

- **TCP SYN:** ‘tcp.flags.syn==1 and tcp.flags.ack!=1‘ - to identify initial connection requests.
- **ICMP errors:** ‘icmp.type in 3..5, 11 —— icmpv6.type in 1..4‘ - to highlight network errors.

6 HTTP Traffic Analysis and Object Export

Analyzing HTTP traffic is a common task in network forensics, especially when investigating web-based activities and potential data exfiltration.

6.1 Capturing and Inspecting HTTP Login

To capture HTTP login traffic, you can start Wireshark, apply a display filter for ‘http‘, and then attempt a login on an HTTP website. The capture log below shows a typical HTTP login sequence, including a GET request for the login page and a POST request after credentials have been submitted.

For sensitive information like usernames and passwords transmitted over unencrypted HTTP, you can easily find them in the packet details of a POST request. In the captured HTTP POST request shown, the form items for ‘uname‘ (username) and ‘pass‘ (password) are clearly visible in plaintext within the ”HTML Form URL Encoded” section.

6.2 Following HTTP Stream and Exporting Objects

To reconstruct a full HTTP conversation or view the entire content transferred, you can ”Follow HTTP Stream.” This concatenates all TCP segments belonging to a particular HTTP conversation into a single readable stream.

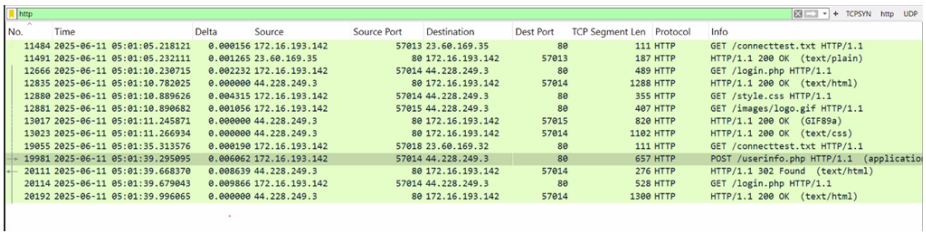

Figure 10: HTTP Login Capture in Wireshark

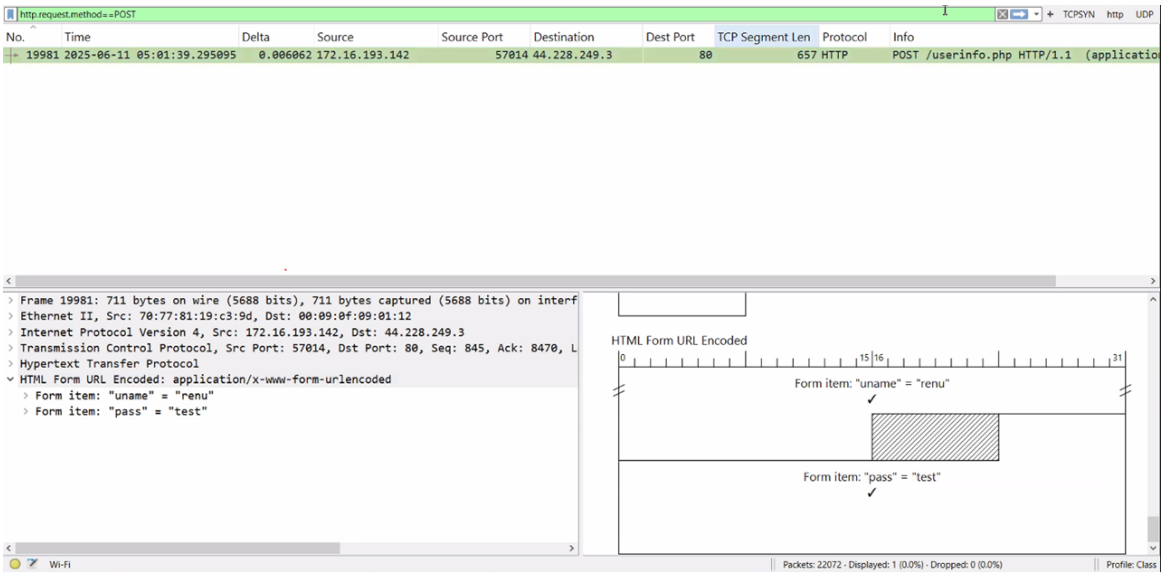

Figure 11: Inspecting HTTP POST for Credentials

The screenshot illustrates how using "Follow HTTP Stream" displays the entire conversational flow, including headers and content, for a specific HTTP transaction.

Wireshark also allows you to extract files or objects transferred over HTTP. This is useful for retrieving downloaded malware, documents, or images. Navigate to 'File ¿ Export Objects ¿ HTTP'. The image shows the Wireshark menu path to export HTTP objects, allowing you to save files found within the captured HTTP traffic.

7 TCP 3-Way Handshake Analysis

The TCP (Transmission Control Protocol) 3-Way Handshake is a fundamental process for establishing a connection between a client and a server. It ensures that both ends are ready to send and receive data.

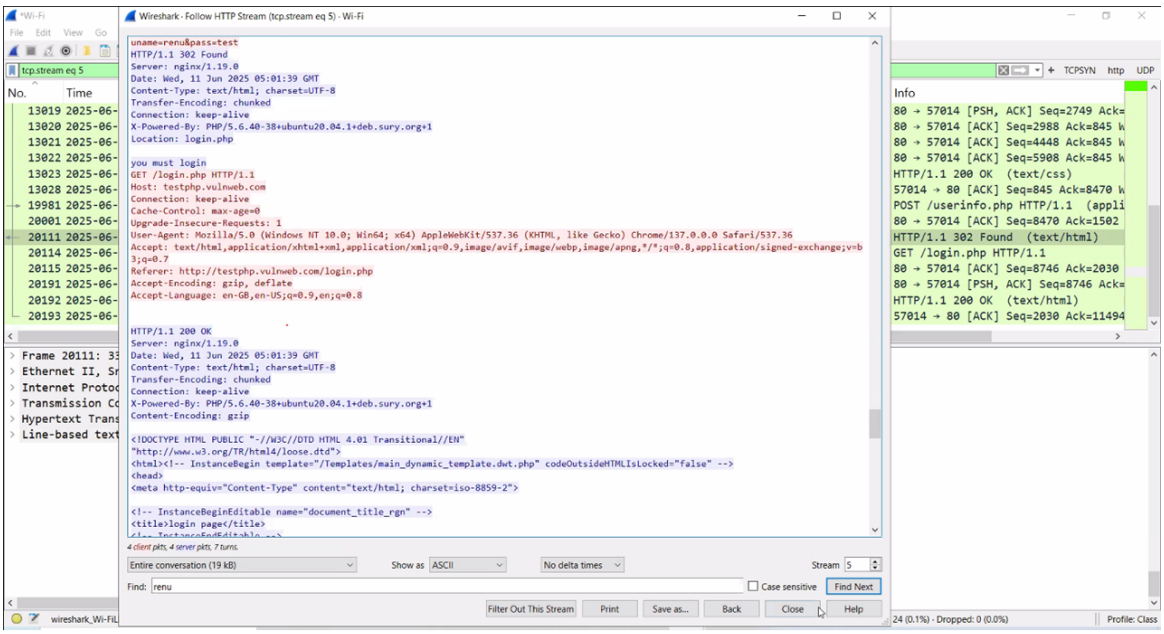

Figure 12: Following an HTTP Stream

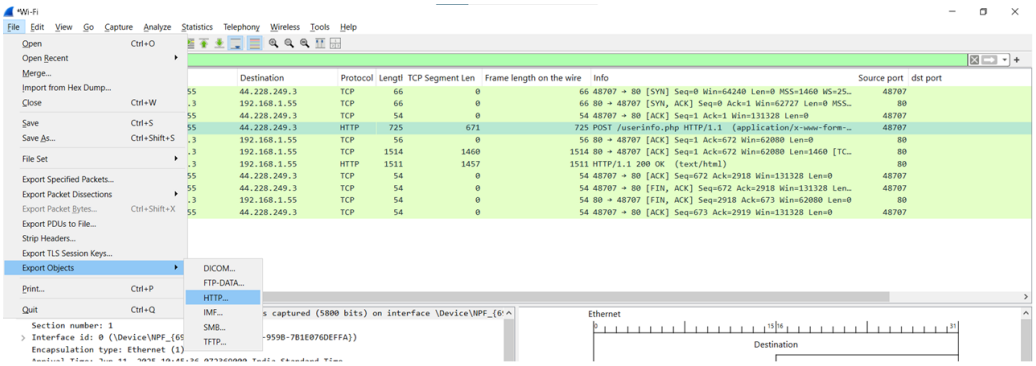

Figure 13: Exporting HTTP Objects

7.1 Steps of the 3-Way Handshake

1. **SYN (Synchronize Sequence Numbers):** The client initiates the connection by sending a SYN segment with its initial sequence number (ISN).

2. **SYN-ACK (Synchronize-Acknowledgment):** The server responds with a SYN-ACK segment, acknowledging the client's SYN and sending its own ISN.

3. **ACK (Acknowledgment):** The client completes the handshake by sending an ACK segment, acknowledging the server's SYN.

The diagram below provides a clear visual representation of the three steps involved in establishing a TCP connection.

7.2 Analyzing in Wireshark

A TCP stream is uniquely identified by a **4-tuple**: (Source IP, Source Port, Destination IP, Destination Port). This identification is crucial for Wireshark to group related packets into a single conversation.

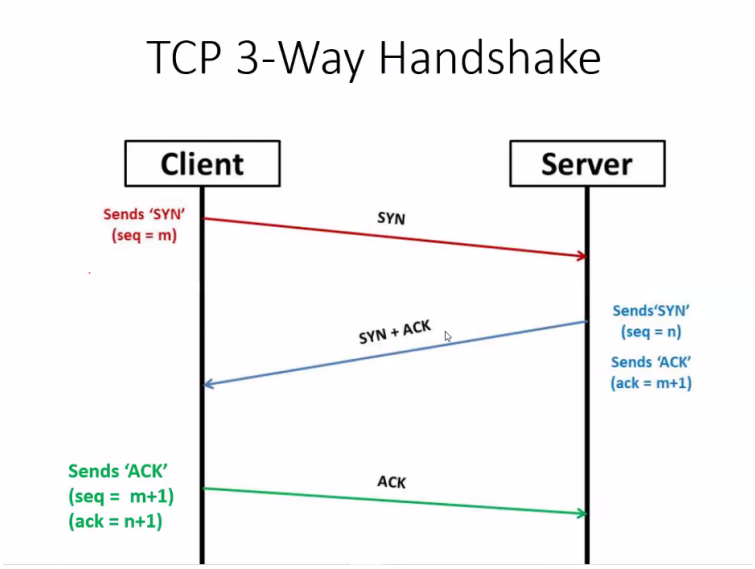

Figure 14: TCP 3-Way Handshake Diagram

In Wireshark, you can observe the flags and sequence/acknowledgment numbers in the TCP header:

For a **SYN Packet** (Client to Server), only the SYN bit is set. The first screenshot details the TCP flags, showing that the 'Syn' bit is set in a SYN packet (0x002).

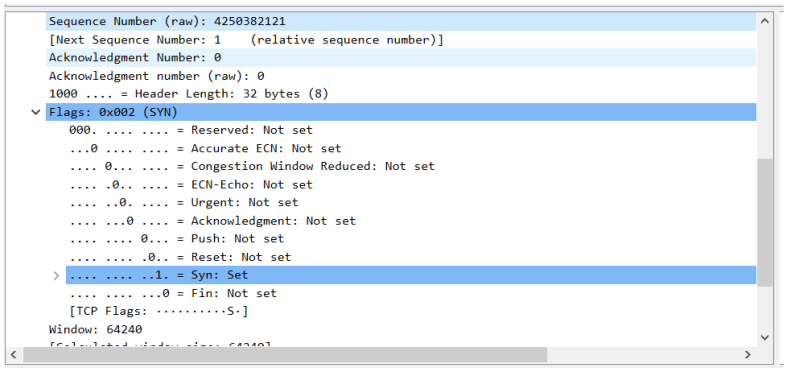

Figure 15: SYN Flag in Wireshark

In a **SYN-ACK Packet** (Server to Client), both SYN and ACK bits are set. The Acknowledgment Number will be the client's sequence number incremented by 1. The second image (Figure 16) shows that both 'Syn' and 'Acknowledgment' flags are set in a SYN-ACK packet (0x012), and the acknowledgment number reflects the client's next expected sequence number.

Finally, in an **ACK Packet** (Client to Server), only the ACK bit is set. The Acknowledgment Number will be the server's sequence number incremented by 1. The third screenshot (Figure 17) highlights the ACK flag being set in a response packet (0x010), demonstrating the client's acknowledgment of the server's SYN.

8 Cryptography Fundamentals

Cryptography is the practice and study of techniques for secure communication in the presence of adversarial behavior. It underpins much of modern cybersecurity. The following high-level diagram illustrates the overall flow of a standard TLS handshake and the preceding TCP connection establishment, providing a foundational understanding before diving into specific cryptographic concepts.

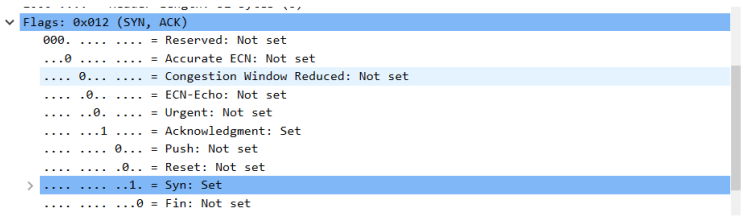

Figure 16: SYN, ACK Flags in Wireshark (Client Sequence Number Incremented)

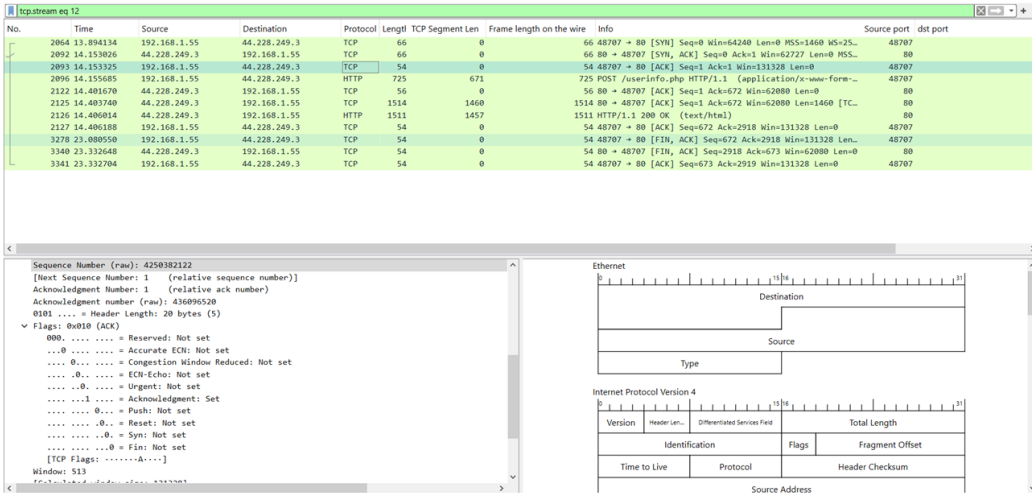

Figure 17: ACK Flag and Acknowledgment Number

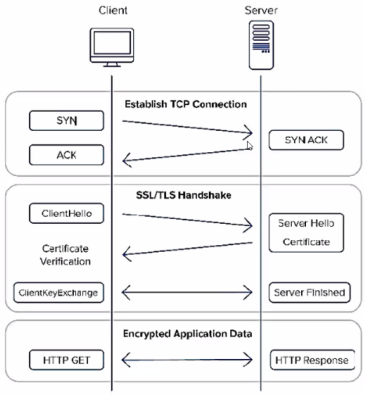

Figure 18: TCP Connection and SSL/TLS Handshake Flow

8.1 Symmetric Key Cryptography

In symmetric key cryptography, the same secret key is used for both encryption and decryption. This means both the sender (Alice) and the receiver (Bob) must possess the identical key. If an unauthorized party (Eve) obtains this key, they can decrypt all communications.

The basic concept of symmetric key cryptography is visualized below, where a single key is shared between Alice and Bob for secure communication, and Eve's ability to intercept without the key is limited.

Figure 19: Symmetric Key Cryptography Basic Concept

A simple example often used to illustrate symmetric encryption is the XOR operation, where a message 'm' is encrypted with a key 'k' to produce ciphertext 'c', and 'c' XOR 'k' returns 'm'. This example demonstrates how the XOR operation functions in symmetric cryptography for both encryption and decryption with the same key.

Plain Text	1	1	0	1	0	0	1	1
Key (Apply)	0	1	0	1	0	1	0	1
XOR (Cipher Text)	1	0	0	0	0	1	1	0
Key (Re-Apply)	0	1	0	1	0	1	0	1
XOR (Plain Text)	1	1	0	1	0	0	1	1

Figure 20: XOR Example in Symmetric Cryptography

8.2 Public Key Cryptography (Asymmetric Cryptography)

Public key cryptography, also known as asymmetric cryptography, uses a pair of keys: a public key and a private key.

- **Public Key ($\mathbf{K_{pub}}$):** This key can be freely distributed and is used to encrypt data or verify digital signatures.
- **Private Key ($\mathbf{K_{priv}}$):** This key must be kept secret and is used to decrypt data encrypted with the corresponding public key, or to create digital signatures.

The private key is the inverse of the public key in a cryptographic sense.

8.2.1 Encryption

To encrypt a message for Bob, Alice uses Bob's public key. Only Bob, with his corresponding private key, can decrypt the message. The diagram below illustrates the encryption process in public key cryptography, showing how Alice uses Bob's public key to encrypt a message that only Bob can decrypt with his private key.

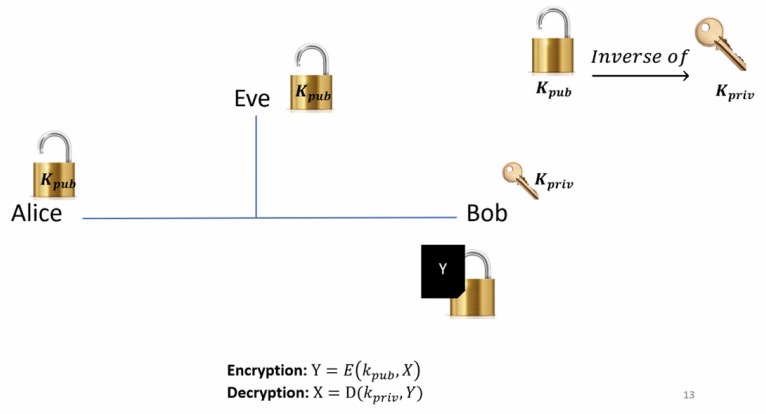

Figure 21: Public Key Cryptography - Encryption

8.2.2 Digital Signatures

Digital signatures ensure authenticity and integrity. Alice signs a message with her private key. Anyone can use Alice's public key to verify that the message indeed came from Alice and hasn't been tampered with. This visual explains how digital signatures work: Alice signs a message with her private key, and anyone can verify its authenticity using her public key.

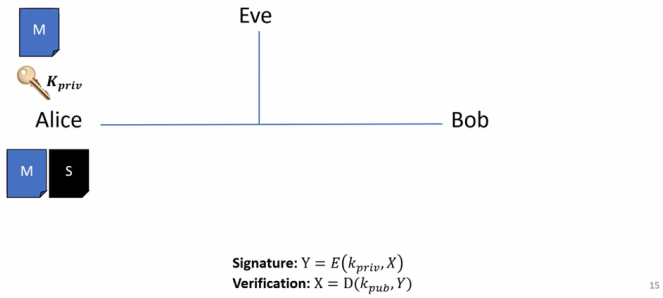

Figure 22: Public Key Cryptography - Digital Signature

8.3 Key Establishment

Key establishment refers to the process by which two or more parties agree on a shared secret key. This is critical for secure communication, especially when transitioning to symmetric encryption after an initial handshake. The conceptual overview below categorizes key establishment into Key Transport and Key Agreement.

8.3.1 Key Transport

In key transport, one party (e.g., Alice) generates a symmetric key (e.g., K_{AES}) and then encrypts it using the other party's (Bob's) public key. This encrypted symmetric key is then sent to Bob, who decrypts it with

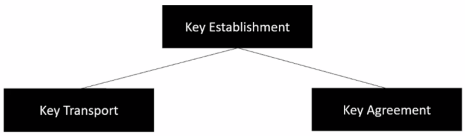

Figure 23: Introduction to Key Establishment

his private key. Public key cryptography is primarily used here to securely transport the symmetric key. This diagram shows the steps of key transport, where Alice securely sends a symmetric key to Bob using Bob's public key for encryption.

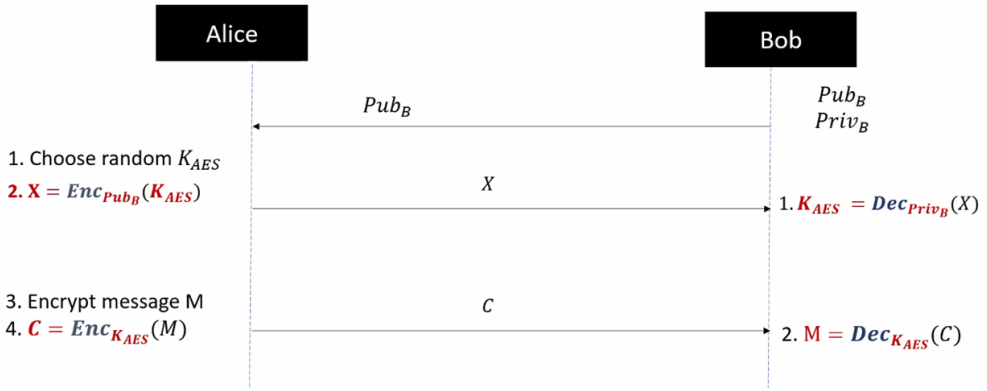

Figure 24: Key Transport using Public Key Cryptography

8.4 Diffie-Hellman Algorithm (Key Agreement)

The Diffie-Hellman algorithm is a method for two parties to establish a shared secret key over an insecure communication channel without directly exchanging the key. This is a "key agreement" rather than "key transport" because neither party creates the key and sends it; instead, they compute it together. The detailed steps of the Diffie-Hellman algorithm are shown, illustrating how Alice and Bob can arrive at a shared secret key over an insecure channel.

8.5 Hashing

Hashing involves transforming data of any size into a fixed-length string of characters, called a hash digest or message digest. This process is one-way (irreversible).

- **Key Properties:**
  - Works with arbitrary input lengths.
  - Produces a short, fixed-length output every time.
  - Even a small change in the input dramatically alters the digest (avalanche effect).

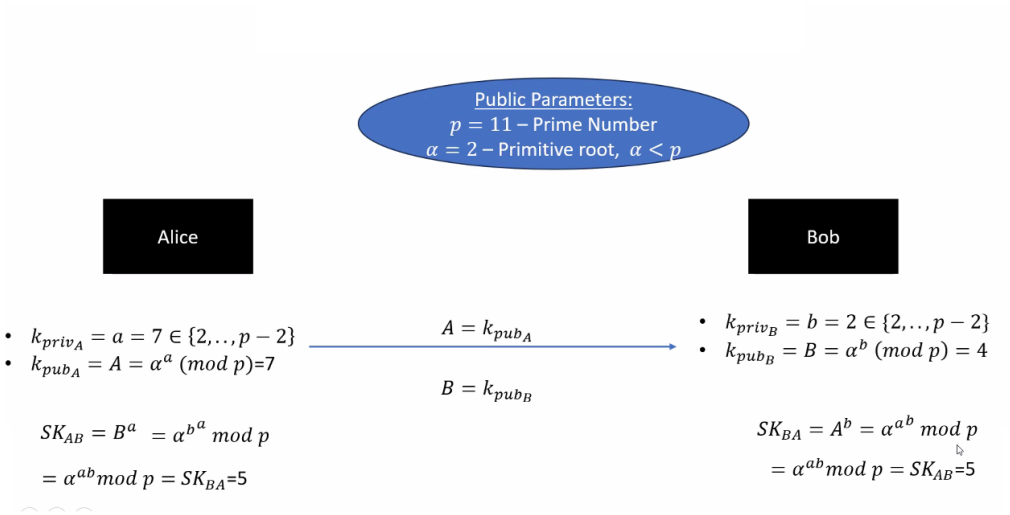

Figure 25: Diffie-Hellman Algorithm

- **Ensures Integrity:** Hashing primarily ensures data integrity. If the hash of a received message matches the hash of the original, it confirms the message hasn't been altered.

The visual demonstrates how different messages produce unique fixed-length hash digests, even with minor changes in the input, a key property of hash functions.

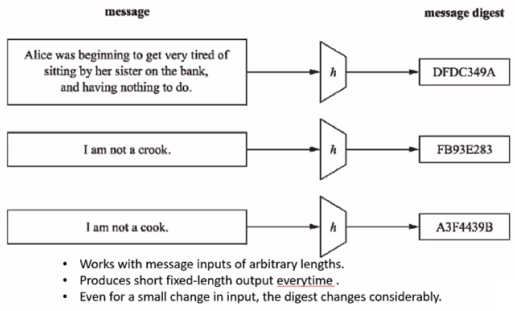

Figure 26: Principal Input-Output Behavior of Hash Functions

Hashing is primarily used to ensure data integrity, allowing recipients to verify that data has not been tampered with. This simple diagram illustrates that hashing guarantees data integrity, detecting any alteration in transit.

8.6 Hashed Message Authentication Code (HMAC)

HMAC extends hashing to provide both **integrity and authenticity**. It involves a cryptographic hash function and a secret cryptographic key. The sender computes an HMAC over the message using the shared secret key, and sends both the message and the HMAC. The receiver, using the same secret key, recomputes the HMAC and compares it with the received HMAC.

This image introduces HMAC, showing that a shared secret key ('K') is used with the message ('M') to produce a tag, ensuring both integrity and authenticity.

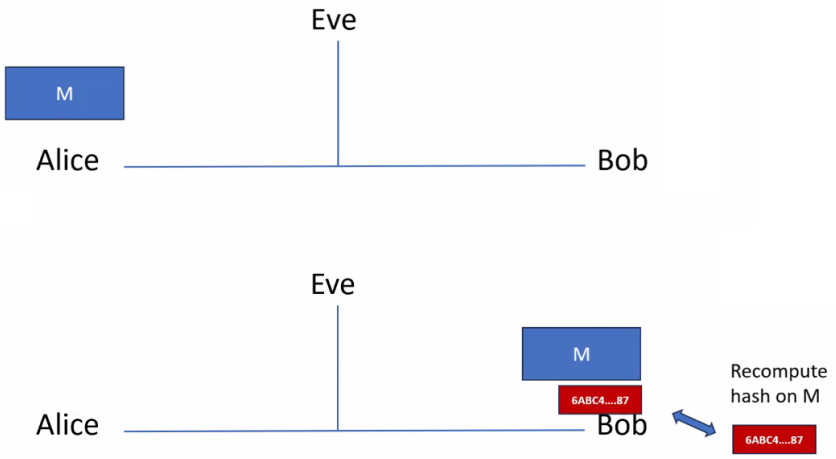

Figure 27: Hashing Ensures Only Integrity

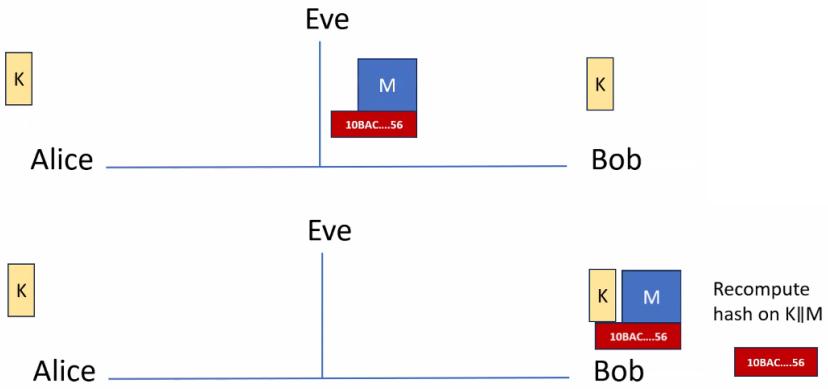

Figure 28: Hashed MAC (HMAC)

For verification, Bob (the receiver) recomputes the HMAC using the same secret key and the received message. If the recomputed HMAC matches the received HMAC, it confirms both data integrity and sender authenticity. The illustration below explains how the receiver (Bob) recomputes the hash on the message combined with the shared key to verify the received HMAC.

9 TLS/SSL Handshake and Decryption in Wireshark

TLS (Transport Layer Security), formerly SSL (Secure Sockets Layer), is a cryptographic protocol designed to provide secure communication over a computer network. The TLS handshake establishes a secure session.

9.1 TLS Preferences in Wireshark

To decrypt TLS traffic in Wireshark, you often need to provide a ‘(Pre)-Master-Secret log filename‘. This file, typically generated by browsers, contains the session keys required to decrypt the encrypted traffic. The Wireshark preferences window for ”Transport Layer Security” is displayed, showing where you can specify the ‘(Pre)-Master-Secret log filename‘ to enable TLS decryption.

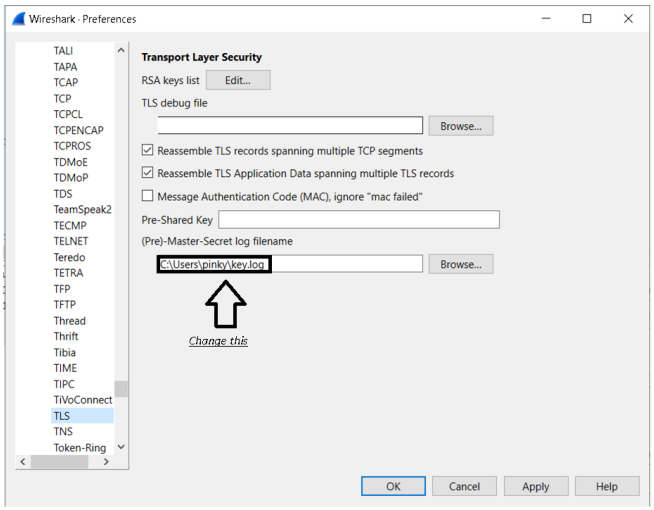

Figure 29: Wireshark TLS Preferences

9.2 DH Key Exchange with Digital Certificates

Digital certificates are crucial for authenticating identities in TLS. A Certificate Authority (CA) issues and signs these certificates, vouching for the identity of a server (or client). During the TLS handshake, especially with Diffie-Hellman (DH) key exchange, certificates help ensure that the public keys being exchanged are legitimate. The diagram below illustrates the process of DH key exchange augmented with digital certificates, showing how a Certificate Authority (CA) vouches for the public keys of Alice and Bob.

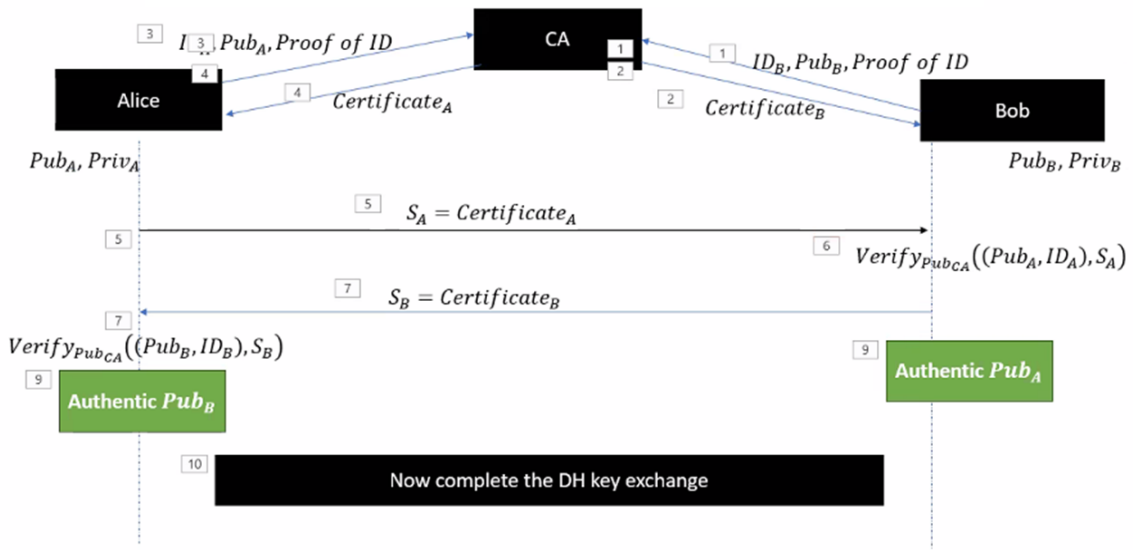

Figure 30: DH Key Exchange with Digital Certificates

Browsers allow you to view certificate details, including who issued them, their validity period, and fingerprints (like SHA-256). The first screenshot provides a general overview of a digital certificate's details, including common name, issuer, and validity period.

The second image shows the hierarchical structure of a certificate, illustrating the chain of trust from the root CA down to the specific certificate.

Browsers maintain a "root store" of trusted CAs, which are pre-installed certificates that your browser trusts implicitly.

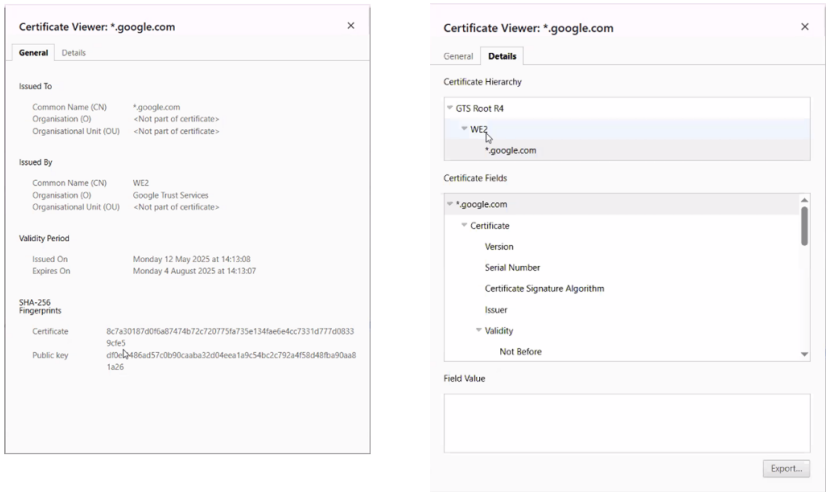

Figure 31: Certificate Viewer - General Details and Hierarchy

9.3 TLS Handshake Deep Dive

The TLS handshake is a series of messages exchanged between the client and server to establish the secure communication parameters.

The client initiates the handshake by sending a 'ClientHello' message. This message includes the highest TLS version it supports, a random number for session key generation, a session ID (for resuming sessions), and a list of proposed cryptographic cipher suites. The image illustrates the 'Client Hello' message and its various components like version, random number, session ID, ciphers, and extensions.

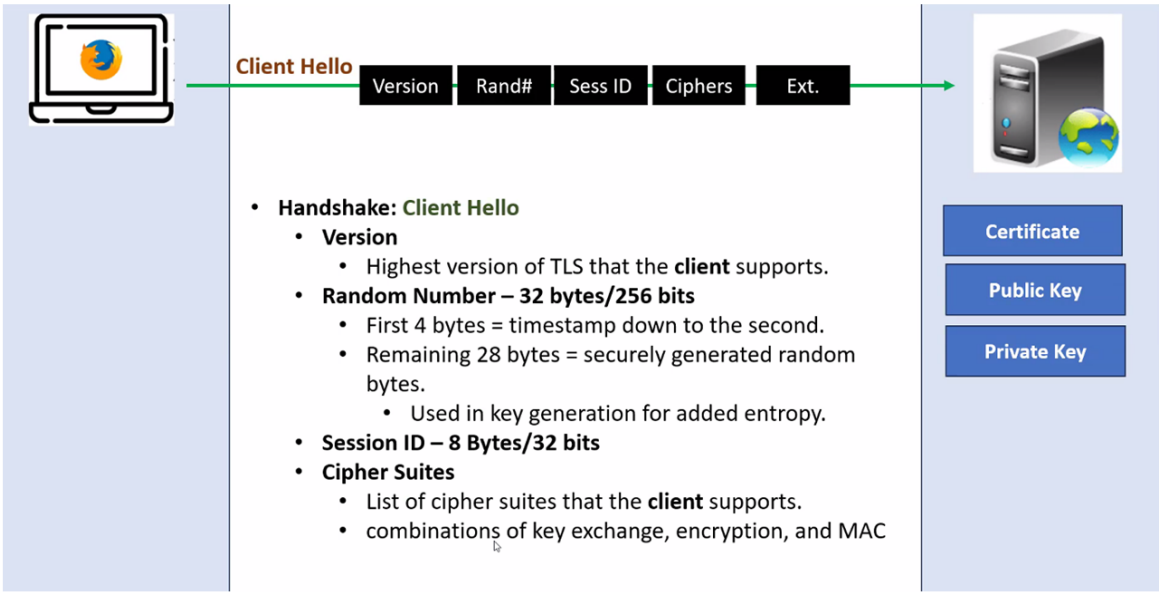

Figure 32: TLS Handshake: ClientHello

The server responds with a 'ServerHello' message. In this message, the server selects the highest mutually supported TLS version, provides its own random number, chooses a session ID, and picks a cipher suite from the client's proposals. This initial ServerHello response is depicted below, showing the server's selection of TLS version, random number, and preferred cipher suite.

A more detailed view of the 'ServerHello' confirms the selected parameters. The expanded view of the 'ServerHello' message explicitly lists the selected TLS version, random number details, session ID, and chosen cipher suite.

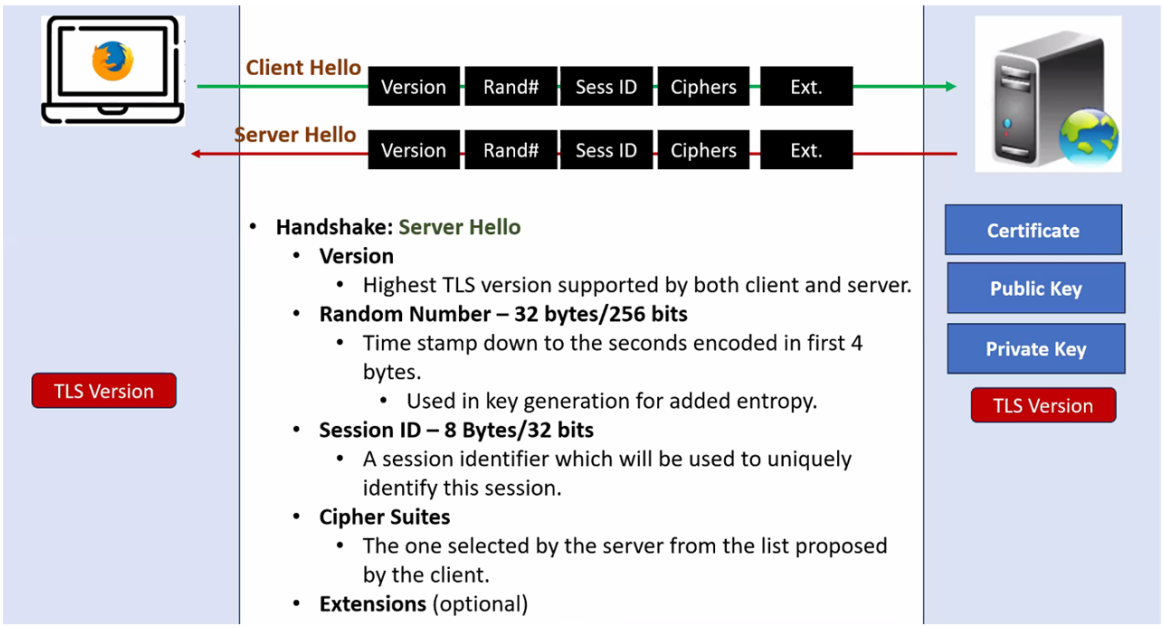

Figure 33: TLS Handshake: ServerHello (Initial)

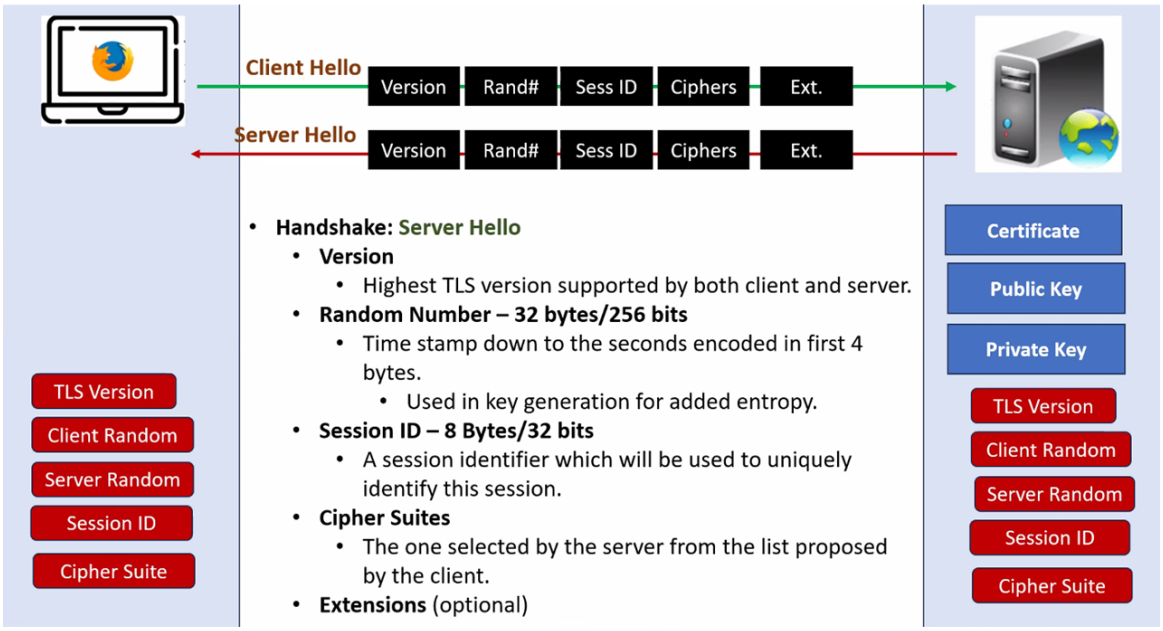

Figure 34: TLS Handshake: ServerHello (Detailed Parameters)

After ‘ServerHello‘, the server typically sends its **Certificate** (and optionally a Certificate Chain), followed by ‘Server Hello Done‘, indicating the server’s portion of the handshake is complete. This illustration shows the Server’s Certificate being sent, followed by the ”Server Hello Done” message, indicating the server has completed its initial handshake responses.

1. **Client Key Exchange:** The client sends the ‘Client Key Exchange‘ message.
   - If RSA key transport (TLS 1.2 and earlier): The client generates a ‘Pre-Master Secret‘ (48 bytes) and encrypts it with the server’s public key from the certificate. Only the server can decrypt this with its private key.
   - If Diffie-Hellman (TLS 1.3 uses DH exclusively): Both client and server use their ephemeral keys and the shared DH parameters to compute the ‘Pre-Master Secret‘ independently.
   - This ‘Pre-Master Secret‘ acts as the seed for all subsequent session keys.

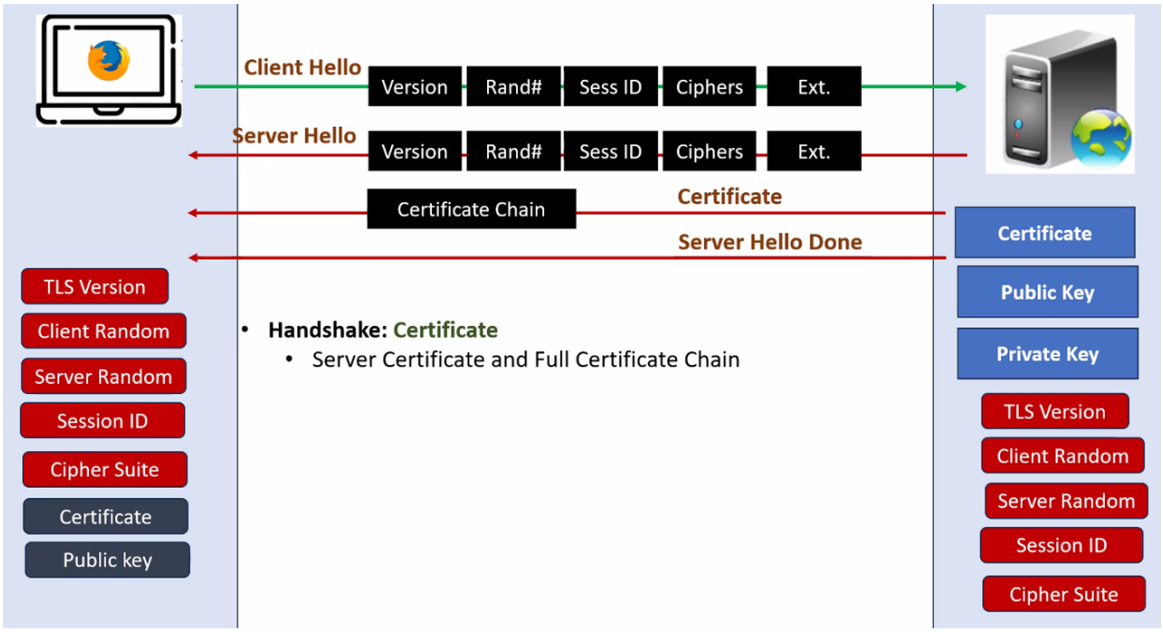

Figure 35: TLS Handshake: Certificate and Server Hello Done

The diagram below focuses on the ‘Client Key Exchange‘ message, showing the transmission of the (encrypted) Pre-master Secret, which is critical for generating session keys.

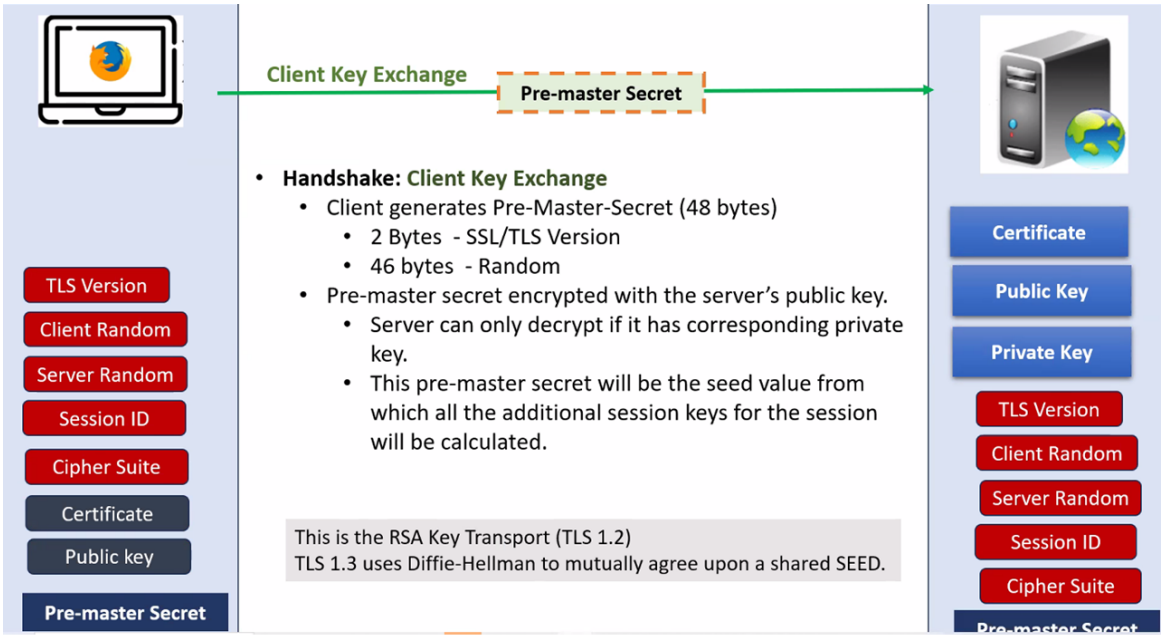

Figure 36: TLS Handshake: Client Key Exchange (Pre-master Secret)

9.3.1 Key Expansion

From the ‘Pre-Master Secret‘, ‘Client Random‘, and ‘Server Random‘, both client and server independently derive the **Master Secret**. The ‘Master Secret‘ is then used in a ”Key Expansion” process to generate all necessary session keys: ‘Client Encryption Key‘, ‘Client HMAC Key‘, ‘Server Encryption Key‘, and ‘Server HMAC Key‘. Initialization Vectors (IVs) are also generated. This flowchart visualizes the ”Key Expansion” process, where the Master Secret and random values are transformed into various session keys required for encryption and authentication.

1. **Change Cipher Spec:** Sent by both client and server, indicating that subsequent messages will be encrypted using the newly negotiated keys.

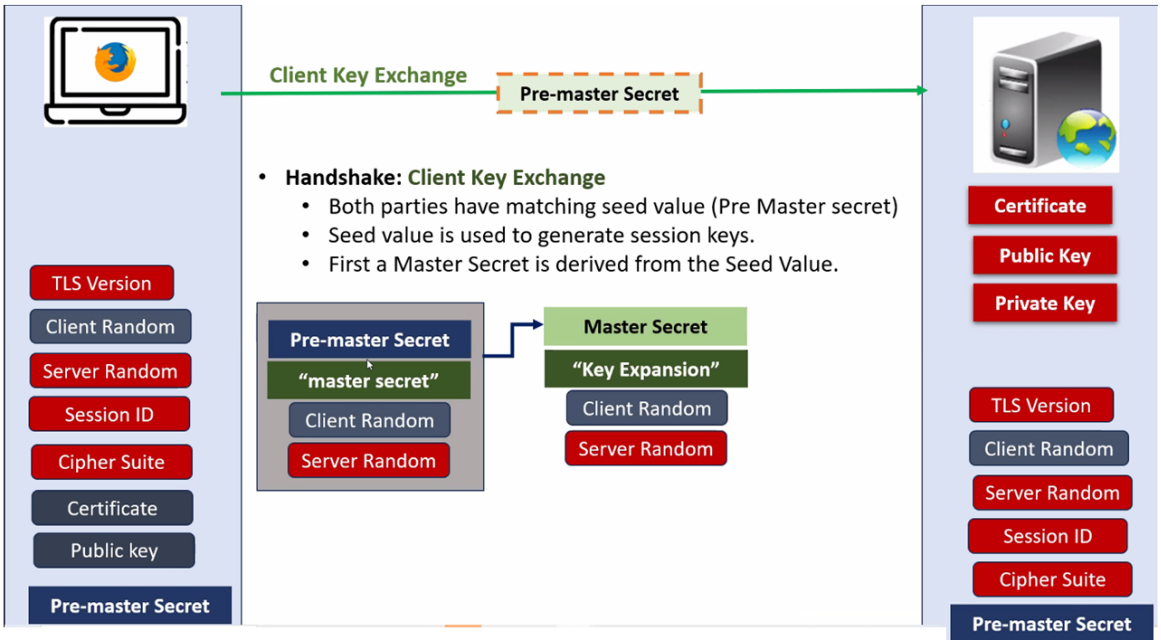

Figure 37: TLS Handshake: Key Expansion to Session Keys

2. **Finished:** The first message encrypted with the new session keys. It contains a hash of all previous handshake messages, serving as a verification that the handshake was successful and not tampered with.

The first diagram illustrates the 'Change Cipher Spec' message, a signal that all subsequent communication will be encrypted.

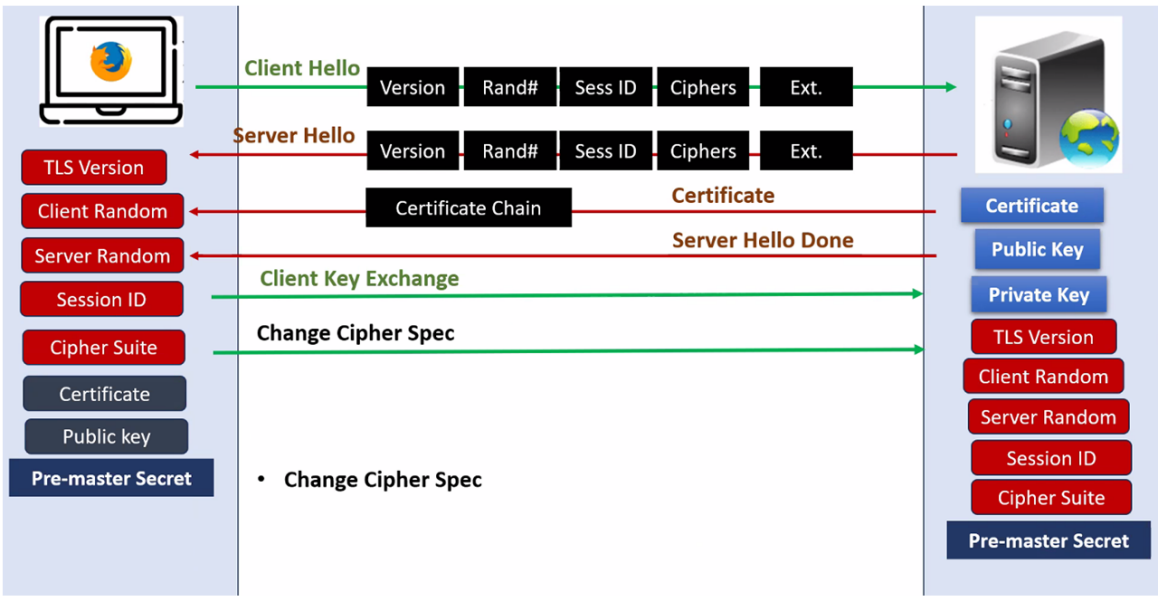

Figure 38: TLS Handshake: Change Cipher Spec

The second diagram shows the 'Finished' message, which is the first encrypted message, confirming the successful completion of the handshake.

9.4 RSA Pre-master Secret Transport Example

In Wireshark, when examining a TLS handshake that uses RSA key transport, you can see the 'RSA Encrypted PreMaster Secret' within the 'Client Key Exchange' message. This encrypted blob contains the crucial pre-master secret that will be used to derive the session keys. The screenshot below from Wireshark captures a

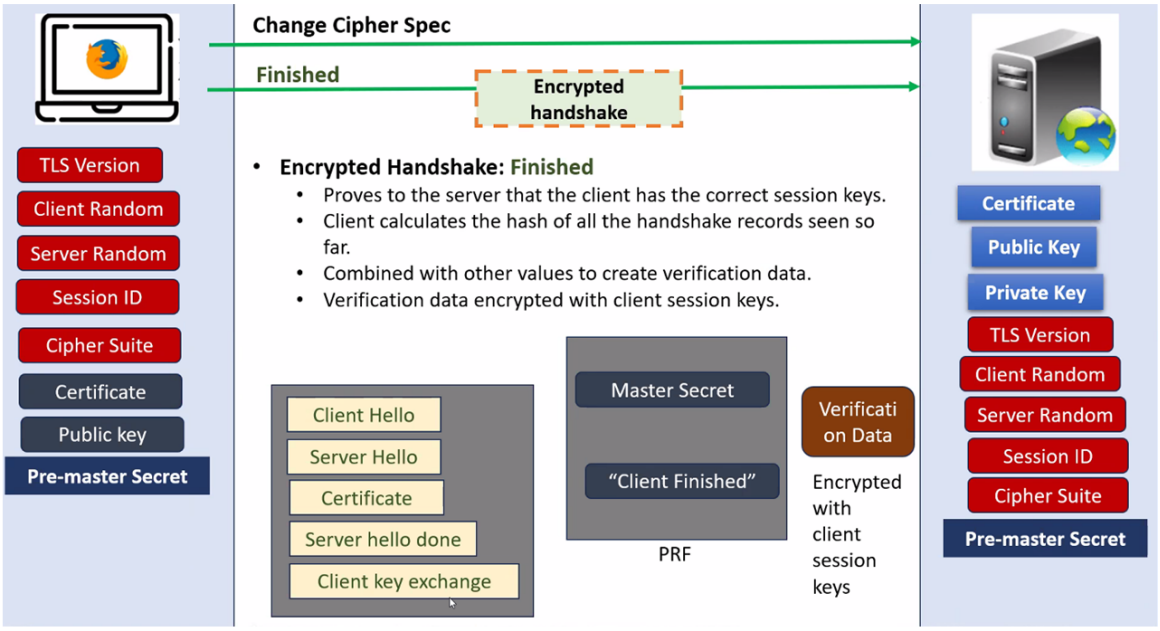

Figure 39: TLS Handshake: Finished (Encrypted Handshake)

'Client Key Exchange' packet, specifically highlighting the "RSA Encrypted PreMaster Secret" field, showing the raw encrypted value.

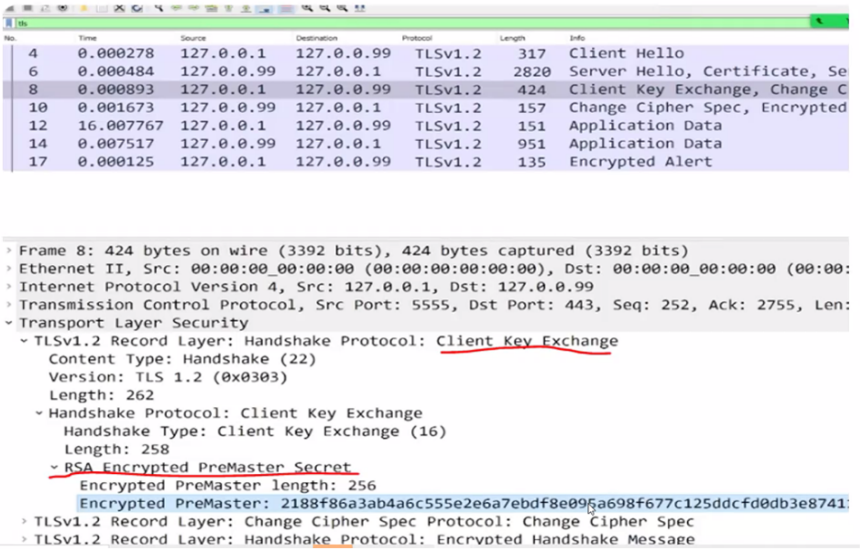

Figure 40: RSA Pre-master Secret Transport in Wireshark

10 Common Attacks: TCP SYN Flood

A TCP SYN Flood is a type of Denial-of-Service (DoS) attack where an attacker sends a high volume of SYN packets to a target server, but never completes the 3-way handshake. This leaves the server waiting for acknowledgments (ACKs) that never arrive, exhausting its resources (e.g., connection table entries) and making it unable to respond to legitimate connection requests.

In Wireshark, you can often identify a SYN flood by:

- A large number of 'SYN' packets without corresponding 'SYN-ACK' or 'ACK' responses from the expected client.

- Using the display filter 'tcp.flags.syn==1' to easily identify SYN packets.
- Observing multiple SYN packets originating from various (potentially spoofed) source IPs, all targeting the same destination.

11 Lab Exercises

These exercises are designed to help you apply the Wireshark and cryptography concepts discussed.

11.1 Exercise1.pcap Questions

1. How many packets were sent to 192.168.1.3?

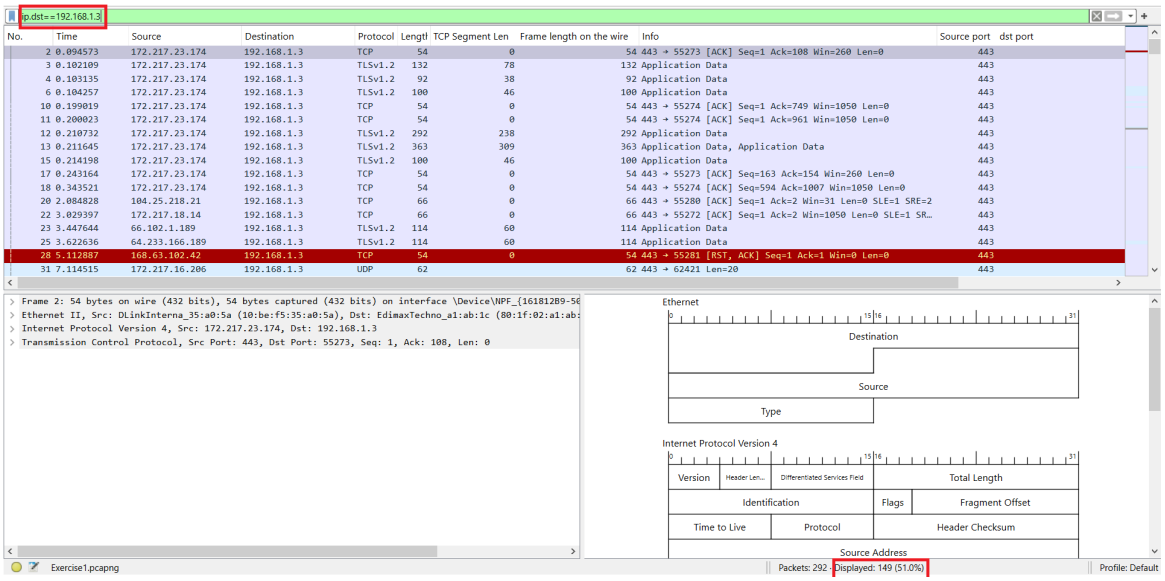

Figure 41: Q 1.1

2. What frames are ARP frames? Provide the frame numbers.

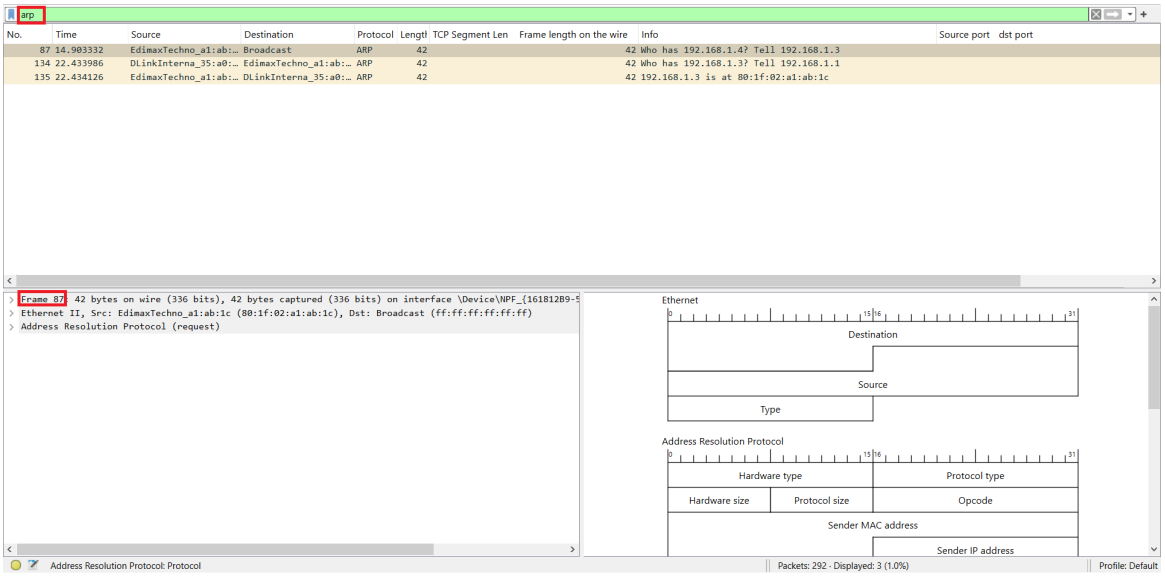

Figure 42: Q 1.2

3. How many of these packets are TCP (Transmission Control Protocol) packets?
4. How many packets have been captured overall?

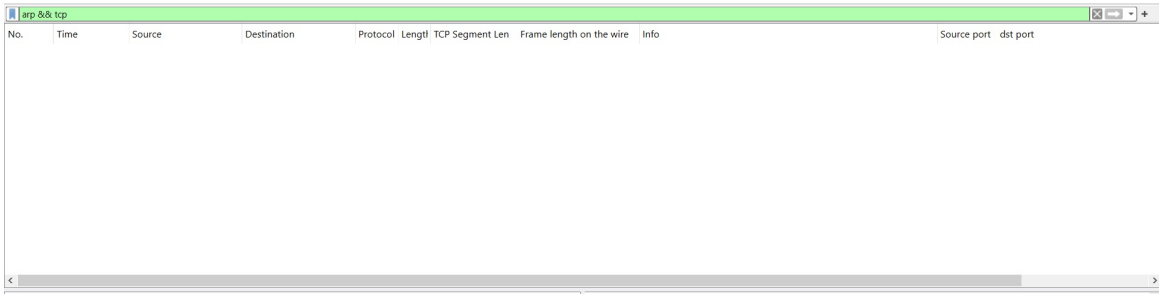

Figure 43: Q 1.3

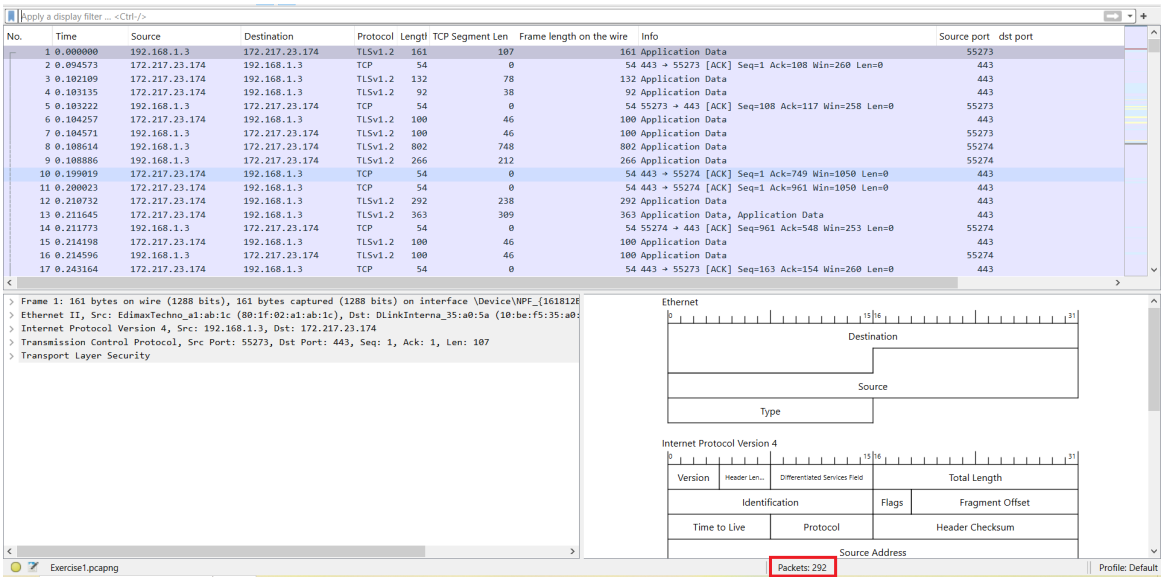

Figure 44: Q 1.4

11.2 Exercise2.pcap Questions

These questions guide you through applying various display filters and using Wireshark statistics to analyze the ‘Exercise2.pcap‘ file.

Set a filter for the following traffic types. How many packets do you get?

1. DNS packets (only dns, no ICMP)
2. DNS packets including the word ”foundation” (regardless of case)
3. DNS packets (only dns, no ICMP)
4. DNS requests for domains that do not have the word ”foundation”
5. ICMP packets (Extra credit, what port is not reachable?)
6. TCP Port 443
7. How many packets are in the top IP conversation? (Hint: Use ‘Statistics ¿ Conversations ¿ IPv4‘ to find the top conversation.) The ‘Statistics ¿ Conversations‘ window is useful for identifying the top talkers or most active conversations in a capture.

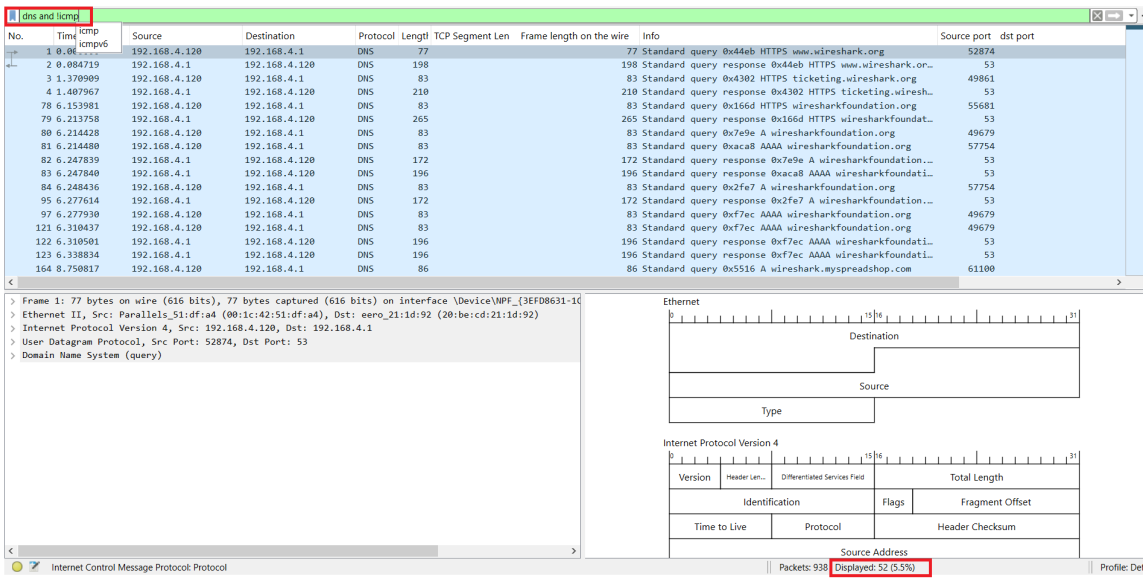

Figure 45: Q 2.1

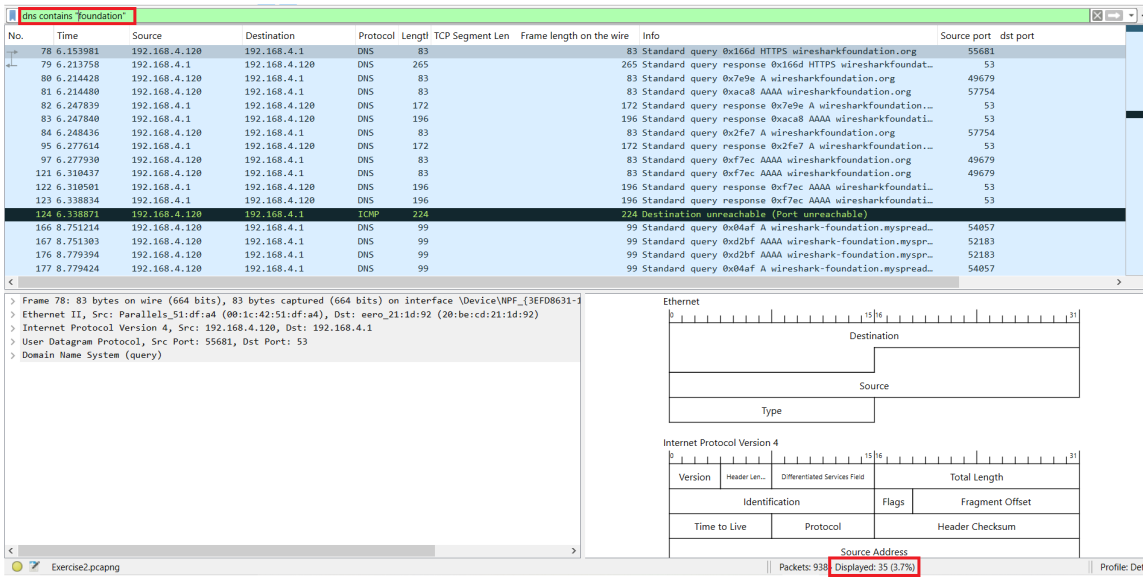

Figure 46: Q 2.2

8. What is the stream ID? Build a filter for this stream and apply it.
9. In this stream, how many packets are larger than 100 bytes?
10. Remove the previous filter, how many packets in the pcap have the TCP SYN bit set to 1?
11. How many packets have the TCP Reset bit set to 1?
12. How many TCP SYN/ACKs are in the PCAP?

11.3 Excercise3.pcap Questions

These questions prompt you to analyze the ‘Excercise3.pcap‘ file to identify potential attacks, compromised systems, and exfiltrated data.

1. What is the protocol using which the highest number of conversations have happened?
2. Do you think any system was attacked? What kind of machine was it? Provide the IP.
3. Find the IP of the attacker machine.
4. Identify the compromised user account.

5. What kind of attacks were launched by the attacker?
6. Do you think the attacker was successful in stealing some files from the victim? If yes, retrieve the file which was compromised.

11.4 Activity: Encrypted Login Walkthrough

This activity will demonstrate how to observe an encrypted login (HTTPS) using Wireshark and understand the underlying cryptographic mechanisms. Unlike the unencrypted HTTP login, you will not see plaintext credentials.

11.4.1 Objective

Understand how TLS/SSL encryption protects login credentials by performing a login on an HTTPS website and observing the traffic in Wireshark.

11.4.2 Prerequisites

- Wireshark installed and configured.
- Access to a web browser (e.g., Chrome, Firefox).
- Access to an HTTPS website where you can perform a login (e.g., your email provider, an online banking site, or search for "test https login page" if allowed by your lab environment). Ensure the URL starts with 'https://'.

11.4.3 Steps

1. **Prepare Wireshark for Capture:**
   - Open Wireshark.
   - Select the correct network interface that connects to the internet (e.g., Wi-Fi, Ethernet).
   - **Start a new capture** (blue fin icon).
2. **Browse to an HTTPS Login Page:**
   - Open your web browser.
   - Navigate to any website with a login page that uses HTTPS. (e.g., your email provider, an online banking site). Ensure the URL starts with 'https://'.
   - Do NOT log in yet.
3. **Initiate TLS Handshake and Observe (Pre-Login):**
   - Observe the packets being captured in Wireshark. You should immediately see packets on TCP port 443 (for HTTPS) labeled 'Client Hello', 'Server Hello', 'Certificate', 'Client Key Exchange', 'Change Cipher Spec', and 'Finished'. These indicate the TLS handshake.
   - Filter the display to 'tls' or 'ssl' to focus on these handshake packets.
   - **Question:** Can you identify the Certificate message and extract its details? (Hint: Review the 'Certificate Viewer' section).
4. **Perform the Encrypted Login:**
   - In your web browser, enter dummy (or your actual, if comfortable) username and password into the login fields of the HTTPS website.
   - Click the "Login" button.
   - Return to Wireshark immediately.
5. **Analyze Encrypted Application Data:**
   - Stop the Wireshark capture (red square icon).
   - Apply a display filter for 'http.request.method == POST' or just 'http' (though for HTTPS, this might not show content directly).
   - More effectively, filter for 'tls' or 'ssl'.

- **Question:** Can you locate the ‘Application Data‘ packets exchanged after the TLS handshake?
- **Question:** Attempt to right-click on one of the ‘Application Data‘ packets and select ‘Follow ¿ TLS Stream‘. Does this show you your plaintext username and password? Why or why not? (Hint: Recall the purpose of the ‘Pre-Master-Secret log filename‘ setting in Wireshark’s TLS preferences).

6. **Conclusion:**

- Document your observations. Explain why you could or could not see the plaintext login credentials.
- Discuss the importance of HTTPS/TLS for protecting sensitive information during transmission.