University of Science and Technology of Hanoi

WEB SECURITY

REPORT

STUDY OF SSL/TLS SECURITY PROTOCOL: ARCHITECTURE, OPERATION, AND APPLICATIONS

Cyber Security

Submitted by

Group 9

Tran Hong Nhat	**BA12-143**
Tran Duc Trung	**BA12-179**
Ha Dinh Tuan	**BA12-183**

Lectures: **Prof. Hoang Xuan Dau**

Hanoi, February 2025

TABLE OF CONTENT

I. Introduction

1.1 Overview of HTTP

The **Hypertext Transfer Protocol (HTTP)** is a fundamental communication protocol used for transferring data on the internet. It follows a **client-server model**, where the client (browser) sends an HTTP request to the web server, which then responds with the requested content.

HTTP is a stateless protocol, meaning it does not retain session information between requests. This design makes HTTP efficient but also introduces security risks, as data is transmitted in plaintext without encryption.

1.2 Security Challenges in HTTP

Since HTTP does not implement encryption mechanisms, it is vulnerable to several security threats, including:

- **Spying**: Attackers can intercept and read sensitive information.
- **Man-in-the-Middle (MITM) Attacks**: Malicious actors can manipulate HTTP traffic.
- **Data Integrity Issues**: Unauthorized modifications to transmitted data.

To address these vulnerabilities, the secure version of HTTP, known as **HTTPS (Hypertext Transfer Protocol Secure)**, was introduced.

1.3 Introduction to HTTPS and SSL/TLS

HTTPS is an enhanced version of HTTP that integrates **SSL/TLS** (Secure Sockets Layer/Transport Layer Security) encryption to protect transmitted data. Instead of sending information in plaintext, HTTPS ensures confidentiality, authentication, and integrity through cryptographic techniques.

SSL/TLS uses a combination of asymmetric encryption (public and private keys) and symmetric encryption (session keys) to establish a secure channel between the client and server. Modern implementations rely on **TLS 1.2 and TLS 1.3**, as older SSL versions are considered insecure.

1.4 The Importance of HTTPS in Web Security

With the rise of cyber threats, HTTPS has become a mandatory security standard for websites handling sensitive user data. Key benefits of HTTPS include:

- **Confidentiality**: Encrypts data to prevent unauthorized access.
- **Authentication**: Verifies the identity of the web server.
- **Data Integrity**: Ensures that transmitted data remains unaltered.

Furthermore, web browsers such as **Google Chrome, Mozilla Firefox,** and **Microsoft Edge** mark HTTP sites as **"Not Secure",** pushing organizations to adopt HTTPS. Compliance with regulations like **GDPR(**General Data Protection Regulation**)** and **PCI DSS(**Payment Card Industry Data Security Standard**)** also mandates the use of HTTPS for secure data handling.

1.5 Understanding SSL/TLS and Its Role in HTTPS

SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are cryptographic protocols that enable secure communication over the internet. While SSL was initially developed by Netscape in the 1990s, it has been deprecated in favor of TLS due to security vulnerabilities.

TLS operates by using public key infrastructure (PKI) and digital certificates to establish a secure session. The protocol consists of several phases:

1. **TLS Handshake**: Establishes a secure session between the client and server.
2. **Key Exchange**: Negotiates encryption keys using asymmetric cryptography.
3. **Data Transmission**: Uses symmetric encryption to protect information.

1.6 Objectives of the Report

This report aims to provide a comprehensive analysis of SSL/TLS, focusing on:

- The architecture and functionality of SSL/TLS in securing online communications.
- The advantages and limitations of implementing SSL/TLS.
- A practical guide to setting up a self-signed SSL certificate with Apache2.
- Methods for verifying and testing SSL/TLS configurations.

II. Architecture of SSL/TLS

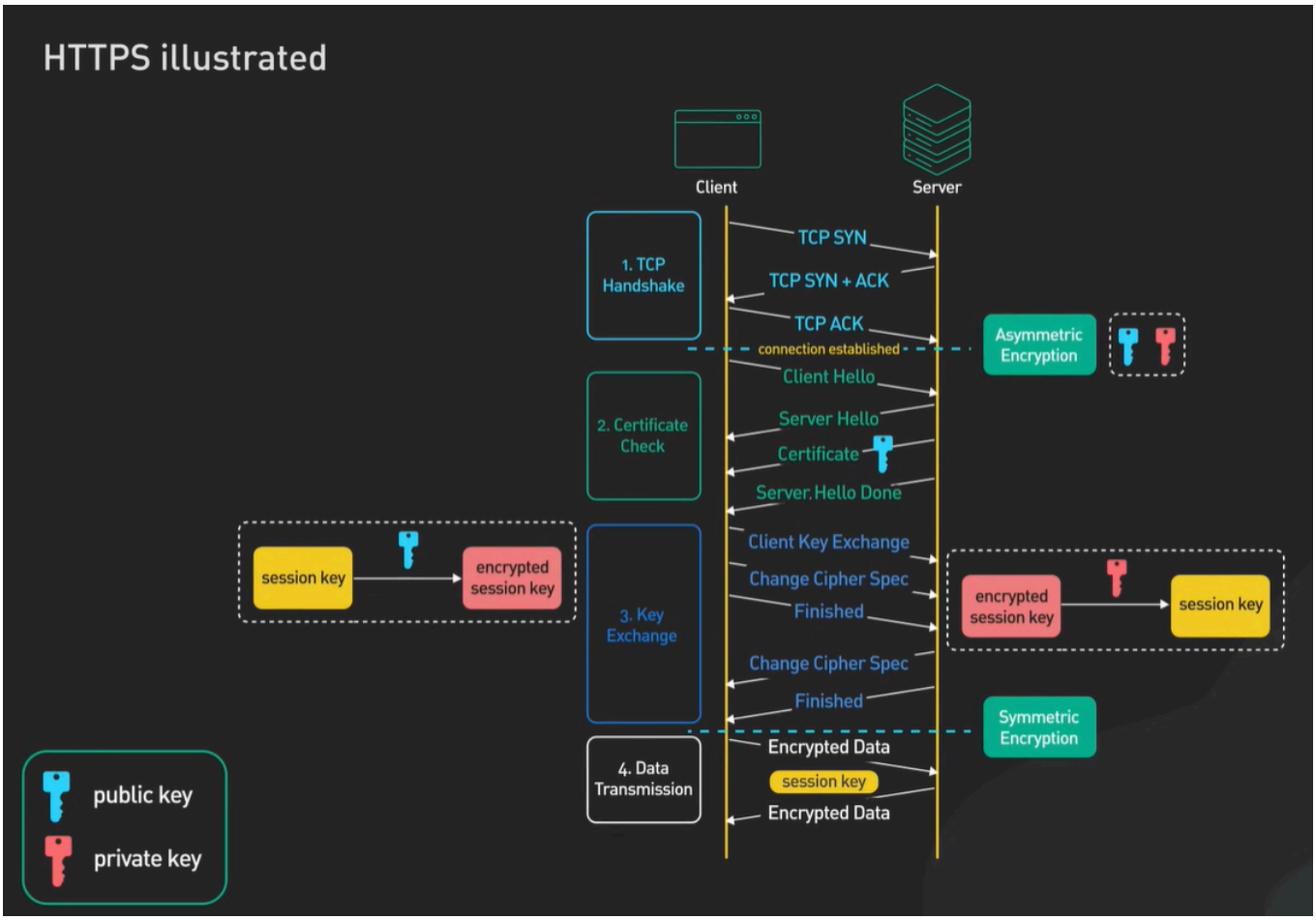
2.1 Overview of SSL/TLS

SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are cryptographic protocols designed to secure communication over networks. They provide encryption, authentication, and data integrity, ensuring that information exchanged between clients (e.g., web browsers) and servers is protected against interception and tampering.

SSL was first developed by Netscape in the 1990s, but due to security weaknesses, it was replaced by **TLS**, which is now the industry standard. The latest version, TLS 1.3, offers enhanced security and faster performance compared to its predecessors.

SSL/TLS operates between the application layer and transport layer of the OSI model, making it crucial for securing web traffic, emails, VoIP, and other network communications.

2.2 SSL/TLS Process



This diagram illustrates the SSL/TLS handshake process, including asymmetric encryption for key exchange and symmetric encryption for data transmission. The public and private key pair is used for securely establishing a shared session key, which then encrypts all further communication.

Step 1: TCP Handshake

Before initiating the SSL/TLS handshake, the client and server establish a **TCP connection** using the **three-way handshake**:

1. The client sends a **TCP SYN** (synchronize) request to the server.
2. The server responds with **TCP SYN + ACK** (synchronize and acknowledge).
3. The client sends a **TCP ACK** (acknowledge) to confirm the connection.

Once the TCP connection is established, the SSL/TLS handshake begins.

Step 2: Certificate Check

After the TCP connection is established, the SSL/TLS handshake process starts:

1. The client sends a **"Client Hello"** message:
   - Includes the **supported TLS versions** (e.g., TLS 1.2, TLS 1.3).
   - Lists **supported cipher suites** (encryption algorithms).
   - Sends a **random number** for encryption.

2. The server responds with a **"Server Hello"** message:
    - Chooses the **TLS version** and **cipher suite** from the client's list.
    - Sends **its SSL/TLS certificate** to verify its identity.
    - Sends another **random number** for encryption.
3. The **client verifies the certificate**:
    - It checks if the certificate was issued by a **trusted Certificate Authority (CA)**.
    - It verifies that the certificate is not expired.
    - It ensures that the domain name in the certificate matches the website.
    - If the certificate is valid, the process continues. Otherwise, the browser will show a security warning.

Step 3: Key Exchange

Once the certificate is verified, the client and server **establish a session key** using **asymmetric encryption**:

1. The client **generates a session key** (a random symmetric key).
2. The session key is **encrypted using the server's public key** (from the certificate).
3. The encrypted session key is sent to the server.
4. The server **decrypts the session key** using its **private key**.

At this stage, asymmetric encryption (public/private key cryptography) is used only for the session key exchange.

Step 4: Cipher Spec Change and Finalization

1. Both the client and server **send "Change Cipher Spec"** messages to indicate that they will now use the shared session key for encryption.
2. They send a **"Finished"** message, confirming that all handshake steps were completed successfully.

At this point, the **secure session is established**, and both parties can communicate using **symmetric encryption**.

Step 5: Secure Data Transmission

- Once the handshake is complete, all communication between the client and server is **encrypted using the session key**.
- The server and client **encrypt and decrypt data** using **symmetric encryption**, which is much **faster and more efficient** than asymmetric encryption.

Before handshake: Data is sent in plaintext (if using HTTP).
After handshake: All data is encrypted and secure (using HTTPS).

Asymmetric encryption is used only during the handshake, while symmetric encryption is used for actual data transmission.

2.3 Key Components of SSL/TLS

2.3.1 Public Key Infrastructure (PKI)

PKI is the system responsible for managing **digital certificates and encryption keys**. It includes:

- **Certificate Authorities (CA):** Trusted entities that issue SSL/TLS certificates.
- **Public and Private Keys:** Used in asymmetric encryption for authentication and key exchange.
- **Digital Certificates:** Verify the identity of websites and organizations.

2.3.2 Encryption Mechanisms

SSL/TLS uses two encryption techniques:

A. Asymmetric Encryption (Public Key Cryptography)

- Uses a **pair of keys**:
    + **Public Key:** Shared with anyone to encrypt data.
    + **Private Key:** Kept secret and used to decrypt data.
- Typically used during the **TLS handshake** for key exchange.
- Slower but provides strong security for authentication.

B. Symmetric Encryption (Session Key Encryption)

- Uses a **single shared key** for encryption and decryption.
- Once a secure session is established, **symmetric encryption** is used because it is **faster and more efficient**.
- Common symmetric encryption algorithms: **AES, ChaCha20**.

2.3.3 SSL/TLS Digital Certificates

SSL/TLS certificates confirm the identity of a website and encrypt communications. They are issued by **Certificate Authorities (CAs)** and contain:

- The website's public key.
- Information about the certificate owner.
- The CA's digital signature.
- Expiration date and validity period.

Self-signed certificates, often used for **testing environments**, are not trusted by browsers and result in **security warnings**.

2.4 Comparison: SSL versus TLS

Feature	SSL	TLS
Versions	SSL 2.0, SSL 3.0	TLS 1.0, TLS 1.1, TLS 1.2, TLS 1.3
Security	Weak, vulnerable	Stronger encryption, secure algorithms
Speed	Slower handshake	Faster handshake with session resumption
Cipher support	Limited, outdated	Supports AES, ChaCha20,...
Usage today	Outdated	TLS 1.2 and TLS 1.3 widely used

Due to the security weaknesses of SSL, SSL should not be used in any system today. Instead, **TLS 1.2 or TLS 1.3** should be implemented to ensure data confidentiality, integrity, and authentication. Major web browsers and organizations have already deprecated SSL in favor of TLS, making it the industry standard.

2.5 TLS 1.2 versus TLS 1.3:

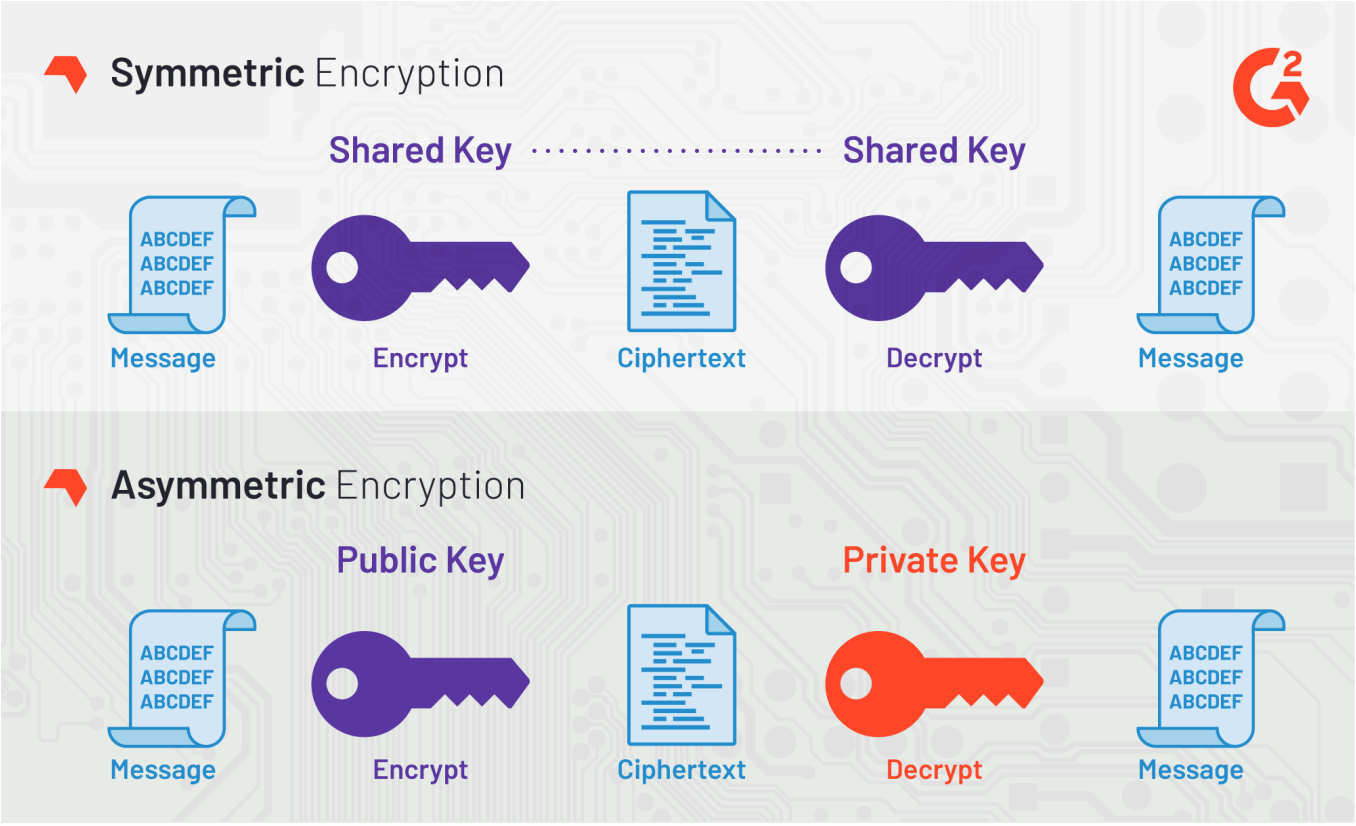
Feature	TLS 1.2	TLS 1.3
Handshake time	Slower, multiple round trips	Faster, fewer round trips
Cipher suites	Many options, some insecure	Simplified, only secure algorithms
Key exchange	RSA, Diffie-Hellman	Ephemeral Diffie-HellMman(Perfect Forward Secrecy)
Security	Allow outdated cryptography	Removes legacy vulnerabilities
Performance	Moderate	Optimized for speed

TLS 1.3 is the recommended standard for secure communication due to its improved efficiency and stronger security mechanisms. However, some legacy systems still rely on TLS 1.2, which remains widely supported. Organizations should transition to TLS 1.3 whenever possible to take advantage of its superior performance and enhanced security features.

III. Features of SSL/TLS

3.1 Encryption

Encryption is the core feature of SSL/TLS that ensures confidentiality by converting readable plaintext into ciphertext. Encryption prevents unauthorized users from accessing sensitive data during transmission.



3.1.1 The workflow of SSL/TLS Encryption:

- SSL/TLS uses a combination of asymmetric encryption and symmetric encryption.
- Asymmetric encryption (Public Key Cryptography) is used during the handshake to securely exchange a session key.
- Symmetric encryption is used for the actual data transmission because it is faster and more efficient.

3.1.2 Encryption Algorithms in SSL/TLS:

SSL/TLS supports multiple encryption algorithms, including:

- AES (Advanced Encryption Standard) - 128-bit, 256-bit (Widely used for secure encryption)
- ChaCha20 (Faster than AES on mobile devices)
- RSA (Rivest-Shamir-Adleman) - Used in older TLS versions for key exchange
- Elliptic Curve Cryptography (ECC) - Provides strong encryption with smaller key sizes

3.2 Authentication

Authentication ensures that the client is communicating with a legitimate server, preventing Man-in-the-Middle (MITM) attacks.

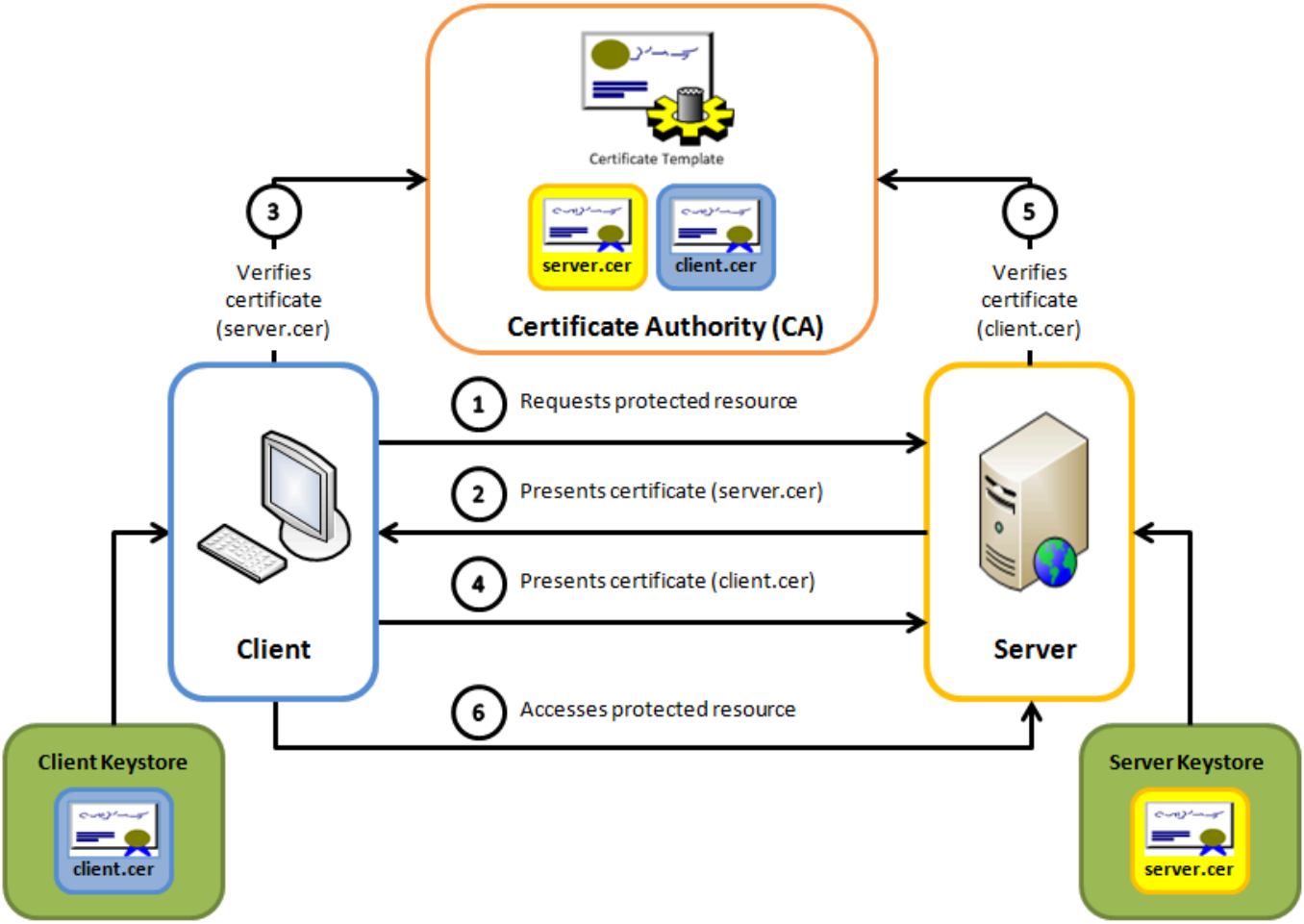
3.2.1 Workflow of SSL/TLS Authentication:

- SSL/TLS uses digital certificates issued by a Certificate Authority (CA) to verify the identity of the server.

- The certificate contains a public key, domain name, and issuer information.
- The client verifies the certificate before establishing a connection.

3.2.2 Types of SSL Certificates:

- Domain Validation (DV): Confirms domain ownership (low trust level).
- Organization Validation (OV): Confirms business legitimacy (medium trust level).
- Extended Validation (EV): Provides the highest level of trust and displays the company name in the browser.



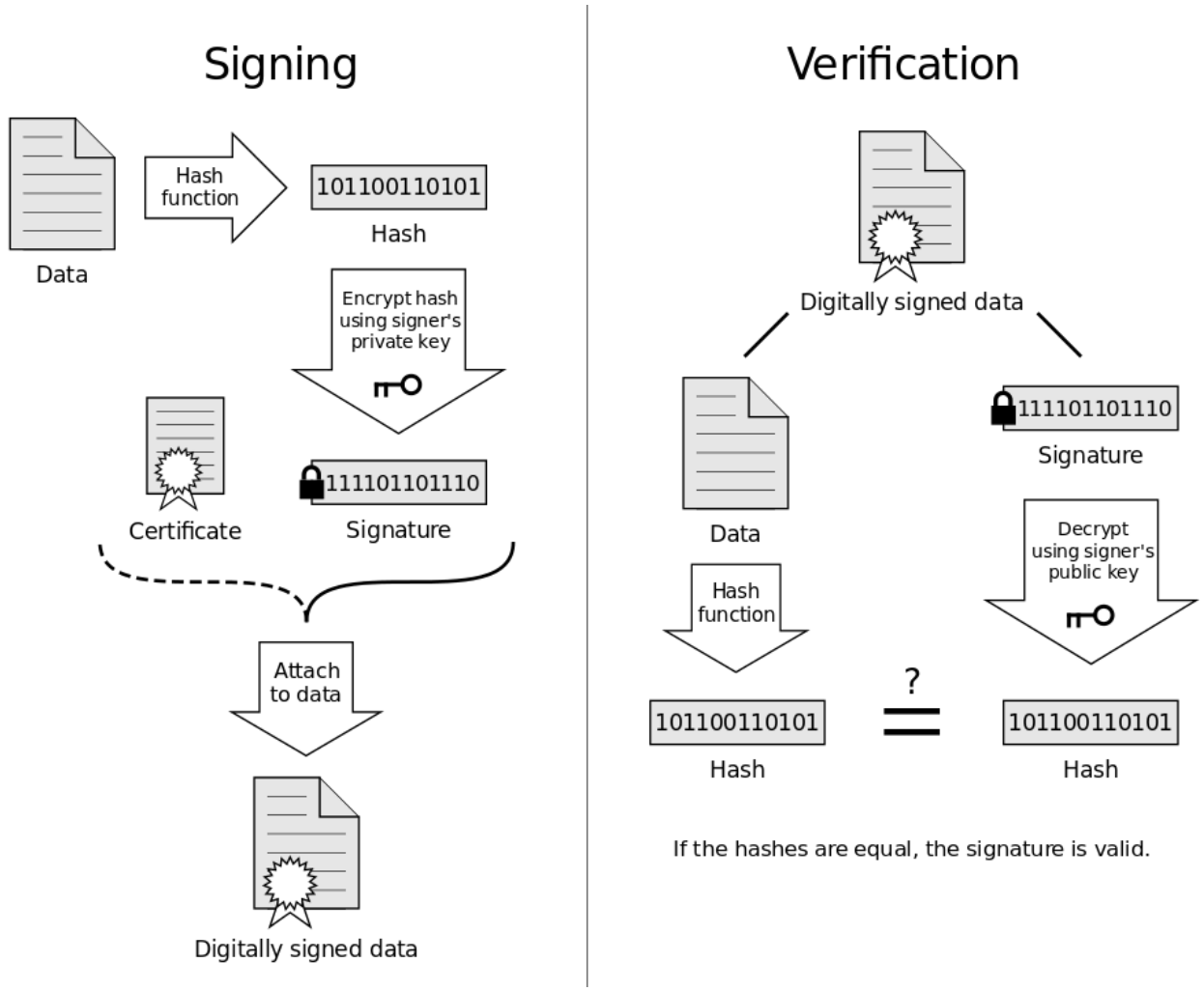
3.3 Data Integrity

Data integrity ensures that data is not altered or tampered with during transmission.

Ways SSL/TLS Ensures Data Integrity:

- SSL/TLS uses Message Authentication Codes (MACs) and cryptographic hashing to verify that transmitted data has not been modified.
- Each SSL/TLS message includes a hash of the original data, which is checked upon receipt.
- Common hashing algorithms used in SSL/TLS include:
- SHA-256 (Secure Hash Algorithm)
    + Standard hashing function

+ HMAC (Hash-Based Message Authentication Code) - Ensures message authenticity

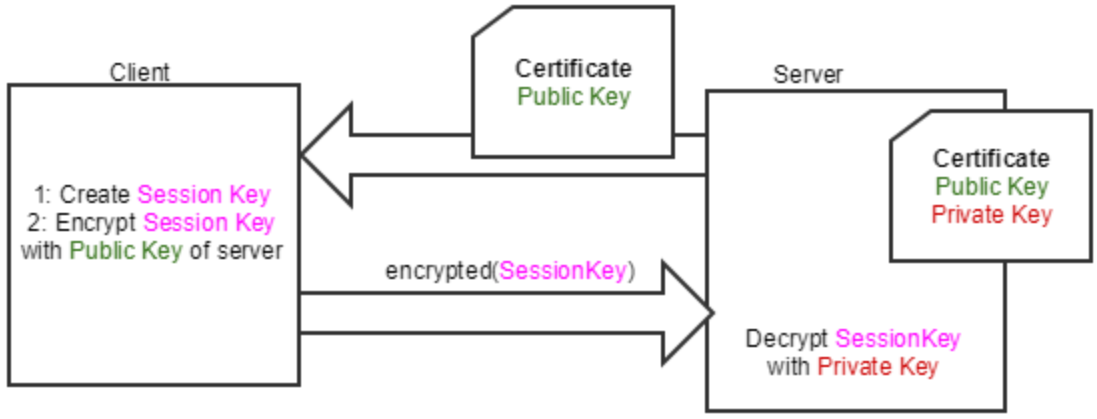


3.4 Forward Secrecy

Forward Secrecy (Perfect Forward Secrecy - PFS) ensures that past communications remain secure even if the private key is compromised.

3.4.1 Workflow of Forward Secrecy:

- Instead of using a static private key for all sessions, SSL/TLS generates a new session key for every new connection.
- This prevents attackers from decrypting previously captured communications, even if they later obtain the private key.



3.4.2 TLS 1.2 vs. TLS 1.3 in Forward Secrecy:

- TLS 1.2: Supports forward secrecy only if specific cipher suites are used (e.g., ECDHE).
- TLS 1.3: Enforces forward secrecy by default in all connections.

IV. Pros and Cons of SSL/TLS

SSL/TLS plays a crucial role in internet security, ensuring the confidentiality, integrity, and authentication of transmitted data. However, implementing SSL/TLS comes with both benefits and challenges. In this section, we will explore the advantages and disadvantages of SSL/TLS in detail, while also suggesting potential solutions to its limitations.

4.1 Advantages of SSL/TLS

4.1.1 Provides Confidentiality Through Encryption

Encryption is the fundamental security feature of SSL/TLS that ensures only authorized parties can access transmitted data. Without encryption, attackers can intercept and read sensitive information.

- SSL/TLS encrypts data before transmission, preventing eavesdropping and data theft.
- Supports both asymmetric encryption (RSA, ECC) for key exchange and symmetric encryption (AES, ChaCha20) for secure communication.

4.1.2 Protects Against Man-in-the-Middle (MITM) Attacks

MITM attacks occur when an attacker intercepts and manipulates communication between a client and a server. SSL/TLS prevents such attacks by encrypting all data exchanges and verifying the server's identity.

- Even if an attacker captures the transmitted data, they cannot decrypt it without the session key.
- Modern web browsers warn users when they visit non-HTTPS sites, reducing MITM attack risks.

4.1.3 Increases User Trust with HTTPS

Websites using SSL/TLS display the secure lock icon in the browser's address bar, indicating a trusted and encrypted connection. This improves user confidence and encourages safer browsing behavior.

- HTTPS reassures users that their data is protected, increasing engagement on secure websites.
- Google prioritizes HTTPS websites in search rankings, improving SEO and website visibility.

4.1.4 Ensures Data Integrity

SSL/TLS prevents unauthorized modifications to transmitted data by using cryptographic hash functions such as SHA-256. This ensures that data remains unaltered between the client and server.

- Cryptographic hashing verifies that data has not been tampered with during transmission.
- TLS uses Message Authentication Codes (MACs) to detect and reject altered data.

4.2 Disadvantages of SSL/TLS

4.2.1 Computational Overhead

SSL/TLS requires additional CPU and memory resources for encryption and decryption processes, which can lead to slower performance compared to plaintext HTTP.

Impact:

- SSL/TLS increases server workload, especially on high-traffic websites.
- Slower TLS handshake compared to unencrypted connections.

4.2.2 SSL/TLS Certificates Can Be Expensive

Obtaining a trusted SSL/TLS certificate from a Certificate Authority (CA) can be costly, especially for Organization Validation (OV) and Extended Validation (EV) certificates.

Impact:

- Small businesses or personal websites may find SSL certificate costs prohibitive.
- Large enterprises requiring multiple domain certificates face higher expenses.

4.2.3 Improper Implementation Can Lead to Security Vulnerabilities

SSL/TLS must be properly configured to ensure security; otherwise, it can introduce vulnerabilities.

Common Misconfigurations:

- Using outdated SSL/TLS versions (e.g., SSL 3.0, TLS 1.0, TLS 1.1).
- Allowing weak cipher suites (e.g., RC4, MD5, SHA-1).

- Not enabling Forward Secrecy, making past communications vulnerable if the private key is leaked.

V. Installation & Configuration of SSL/TLS with Apache2

SSL/TLS is a crucial security mechanism that provides encrypted communication between a client and a web server. In this section, we will install and configure **SSL/TLS on Apache2** using a **self-signed certificate** on **Ubuntu 20.04.6 LTS**. The goal is to enable **both HTTP (port 80) and HTTPS (port 443)** to allow comparison between secure and unsecure connections.

5.1 System Requirements

- **Operating System**: Ubuntu 20.04.6 LTS (installed on VMware Workstation)
- **Web Server**: Apache2
- **SSL Toolkit**: OpenSSL(used for generating self-signed certificates)

5.2 Steps to Set Up HTTPS with a Self-Signed Certificate

Step 1: Install Apache and OpenSSL

```
sudo apt update
sudo apt install apache2 openssl -y
```

Step 2: Enable SSL Module in Apache

To support HTTPS, we need to enable Apache's SSL module by command:

```
sudo a2enmod ssl
```

Then we restart apache2 service:

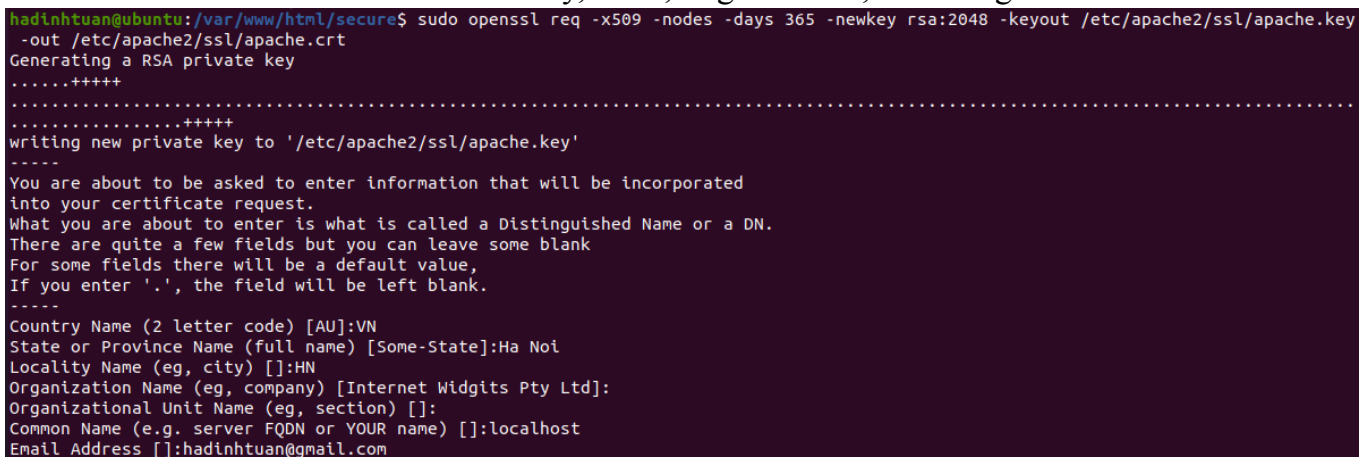
```
sudo systemctl restart apache2
```

Step 3: Generate a Self-Signed SSL Certificate

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
    /etc/apache2/ssl/apache.key -out /etc/apache2/ssl/apache.crt
```

- **-x509**: Generates a self-signed certificate.
- **-nodes**: No passphrase for the private key.
- **-days 365**: Certificate validity of 1 year.
- **-newkey rsa:2048**: Creates a 2048-bit RSA key.

Then we must to enter values for Country, State, Organization, etc. as figure below:



Step 4: Configure HTTPS Virtual Host

Edit the default SSL configuration file at /etc/apache2/sites-available/default-ssl.conf:

```
<IfModule mod_ssl.c>
   <VirtualHost *:443>
      ServerAdmin webmaster@localhost
      DocumentRoot /var/www/html/secure
      ServerName localhost

      SSLEngine on
      SSLCertificateFile /etc/apache2/ssl/apache.crt
      SSLCertificateKeyFile /etc/apache2/ssl/apache.key

      <Directory /var/www/html/secure>
         AllowOverride All
         Require all granted
      </Directory>

      ErrorLog ${APACHE_LOG_DIR}/error.log
      CustomLog ${APACHE_LOG_DIR}/access.log combined
   </VirtualHost>
</IfModule>
```

- SSLEngine on: Enables SSL/TLS.
- SSLCertificateFile & SSLCertificateKeyFile: Specify paths to the SSL certificate and private key.
- DocumentRoot /var/www/html/secure: Serve HTTPS content from the secure directory.

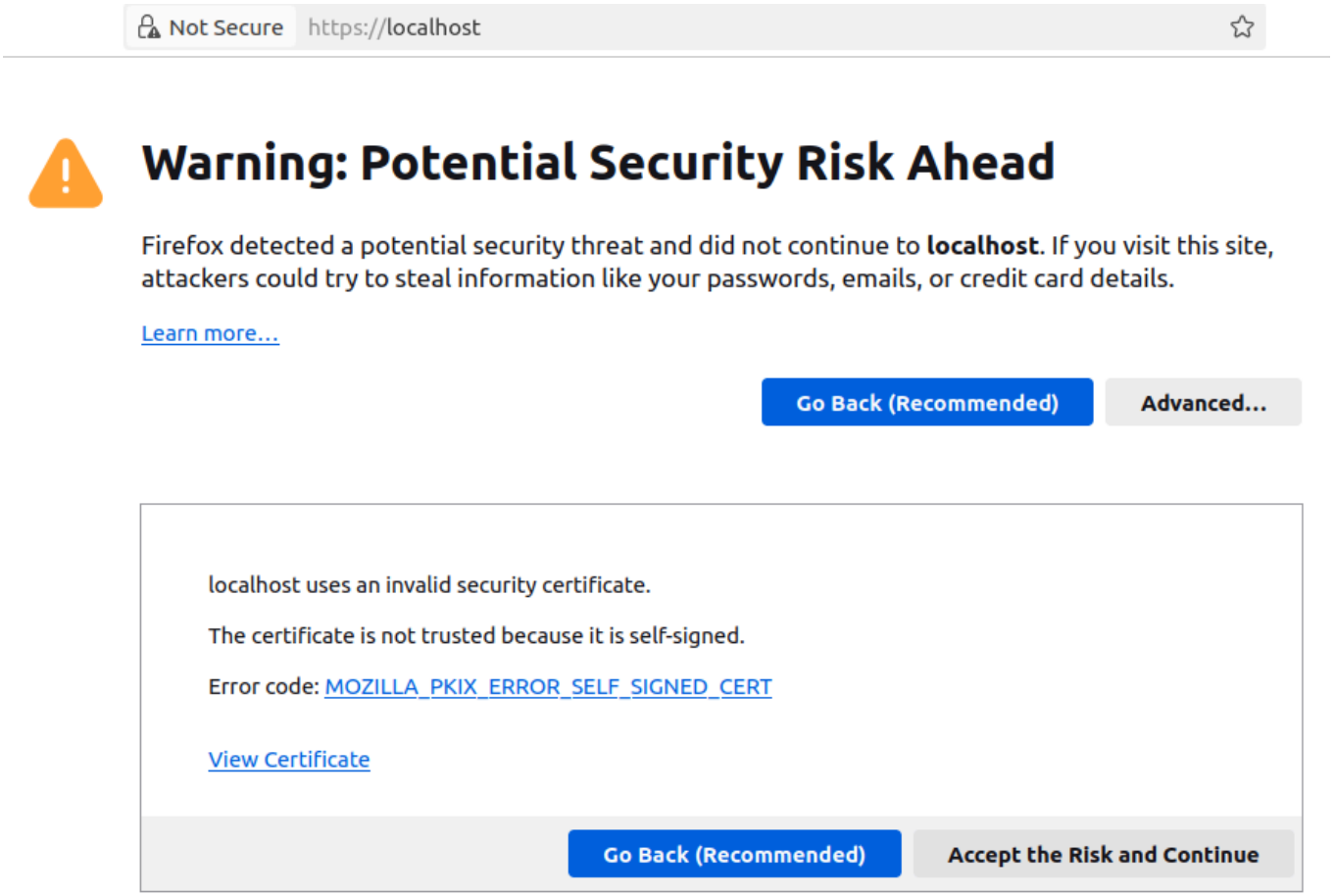
Then we need to enable SSL Virtualhost and Restart Apache2

```
sudo a2ensite default-ssl
```

```
sudo systemctl restart apache2
```

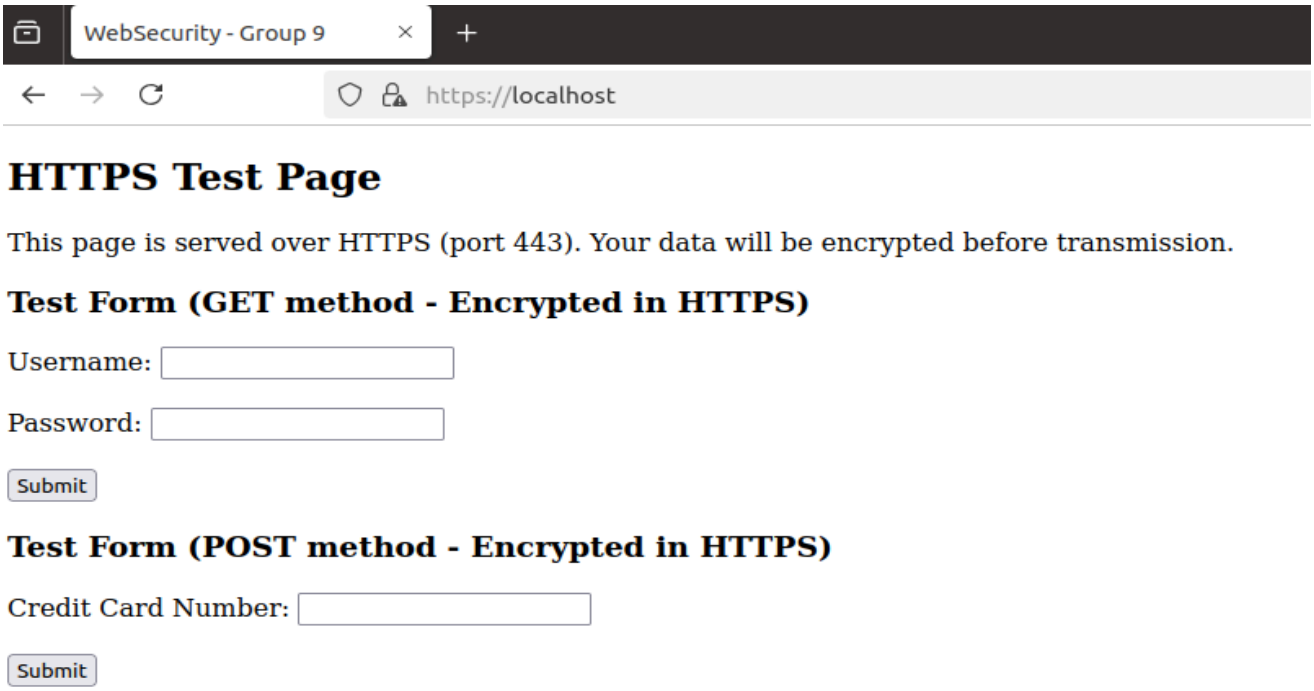
Step 5: Verify HTTPS Connection

- Open a web browser and go to: https://localhost
- Browser warning appears due to self-signed certificate.



- Click **Advanced** > **Accept the Risk and Continue** to access the website.

Result:



VI. Testing HTTPS Connection

The purpose of this section is to **demonstrate how HTTPS protects sensitive data** compared to HTTP.
Using **Wireshark**, we will capture network packets and analyze how data is transmitted in **plaintext over HTTP** but **encrypted in HTTPS**.

The analysis will focus on **two captured packets**:

- **HTTP Traffic** (unencrypted, data visible in plaintext).
- **HTTPS Traffic** (encrypted, preventing data interception).

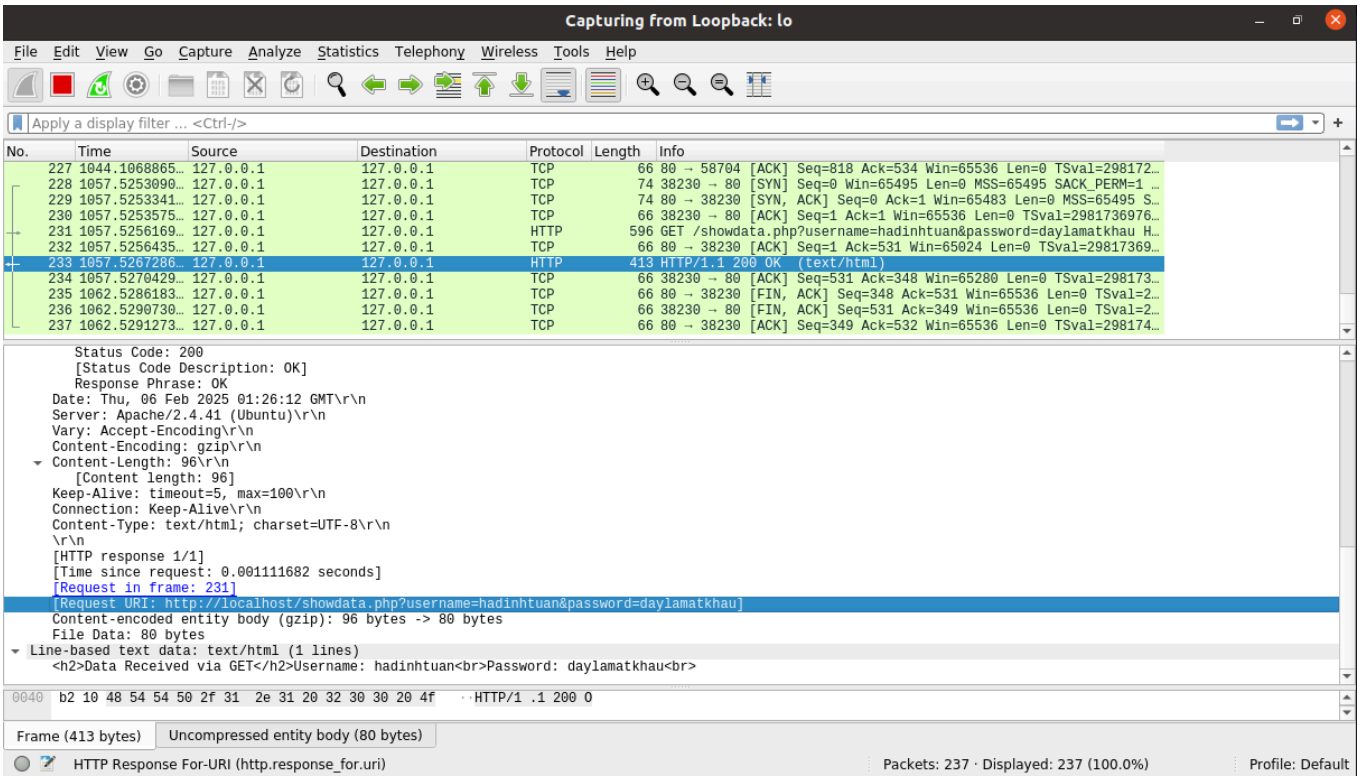
6.1 Capturing HTTP Traffic Using Wireshark

Steps to Capture HTTP Traffic

1. Start Wireshark and select the network interface (loopback).
2. Apply the **capture filter**: tcp.port==80 or http (This ensures that only **HTTP traffic** is recorded.)
3. Open a browser and visit: http://localhost
4. Fill in the form with:
   Username: **hadinhtuan**

   Password: **daylamatkhau**

5. Click Submit and capture from wireshark:



The packet contains an HTTP request and response.

Username and password appear in plaintext in the packet details:

GET /showdata.php?username=hadinhtuan&password=daylamatkhau HTTP/1.1

The entire URL, including credentials, is visible, making it vulnerable to Man-in-the-Middle (MITM) attacks.

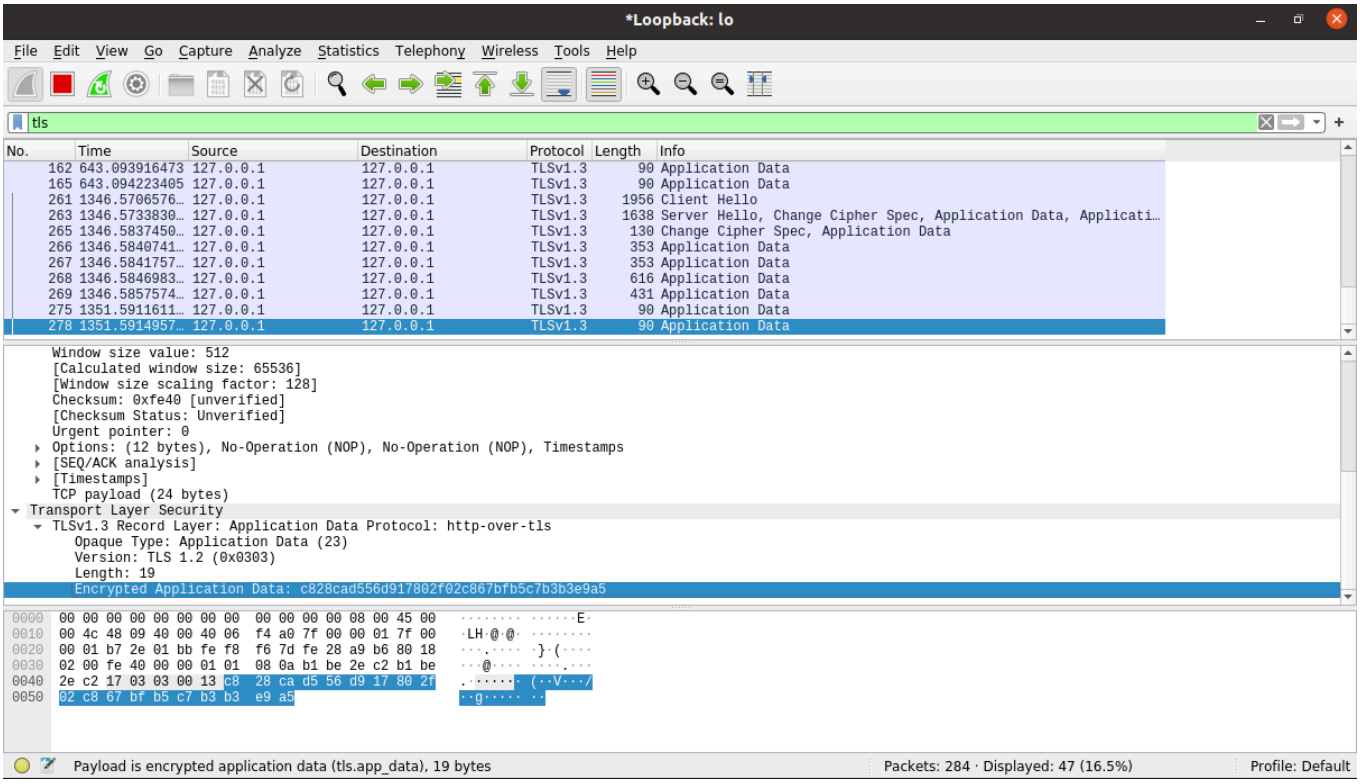
Security Risk:

Any attacker on the network can read sensitive data because HTTP does not encrypt requests or responses.

Credentials transmitted over HTTP can be intercepted and misused.

6.2 Capturing HTTPS Traffic Using Wireshark

We do the same as HTTP but change the filter in wireshark to tls or tcp.port=443



The **TLS 1.3 protocol** is used, meaning all data is encrypted.

Instead of a visible HTTP request, the captured packet only shows **Application Data (Encrypted)**:

TLSv1.3 Record Layer: Application Data Protocol: http-over-tls

The username and password are **NOT visible**, ensuring confidentiality and security.

Security Improvement:

HTTPS encrypts all transmitted data, preventing attackers from reading sensitive information.

Even if someone captures the packets, the encrypted payload cannot be decrypted without the correct TLS session keys.

VII. Conclusion

Through this study, I have gained a comprehensive understanding of SSL/TLS, its architecture, operation, and practical applications in securing web communications. By setting up an Apache2 server with both HTTP and HTTPS and analyzing network traffic using Wireshark, I was able to observe the impact of SSL/TLS encryption on data transmission.

Key Takeaways:

1. SSL/TLS is essential for web security, providing encryption, authentication, and data integrity to protect against cyber threats.
2. The SSL/TLS handshake establishes a secure communication channel between the client and the server by exchanging certificates and cryptographic keys.
3. TLS 1.3 improves security and performance compared to older versions by eliminating outdated cryptographic algorithms and reducing handshake latency.
4. Wireshark analysis confirms that HTTP transmits data in plaintext, while HTTPS encrypts all communication, preventing unauthorized access.
5. Self-signed certificates are useful for testing but should be replaced with CA-signed certificates in real-world deployments.

This research highlights why SSL/TLS is the foundation of secure web communication and why all modern web services must implement HTTPS.

VIII. References

[1] https://aws.amazon.com/vi/compare/the-difference-between-https-and-http/
[2] https://www.digicert.com/what-is-ssl-tls-and-https
[3] https://en.wikipedia.org/wiki/Transport_Layer_Security
[4] https://www.keyfactor.com/blog/http-vs-https-whats-the-difference
[5] https://viettelidc.com.vn/tin-tuc/quyen-rieng-tu-gdpr-la-gi-tat-tan-tat-cac-thong-tin
[6] https://letsencrypt.org/how-it-works/
[7] https://datatracker.ietf.org/doc/html/rfc5246
[8] https://datatracker.ietf.org/doc/html/rfc8446
[9] https://httpd.apache.org/docs/2.4/ssl/ssl_howto.html
[10] https://www.youtube.com/watch?v=j9QmMEWmcfo
[11] https://www.openssl.org/docs/manmaster/man1/openssl-req.html
[12]https://www.techtarget.com/searchsecurity/definition/PCI-DSS-Payment-Card-Industry-Data-security-Standard