Certificate Management and HTTPS for Apache2

1. Introduction

This documentation covers the basics and practical implementation of **Certificate Management and HTTPS Configuration** for the Apache2 web server on Linux. It is aimed at **students and teachers** who want to familiarize themselves with web security, encryption, and certificate infrastructure.

The goal is to create a private **Certificate Authority (CA)**, sign certificates for servers, and use these certificates to secure communication between client and server (**HTTPS**).

---

2. Fundamentals

Introduction:

Before we begin the practical implementation, it's important to understand **why certificates** and **HTTPS** are necessary. This chapter explains the theoretical foundations required for understanding the following steps.

2.1 What is HTTPS?

HTTPS (Hypertext Transfer Protocol Secure) is the encrypted variant of HTTP. It uses **TLS (Transport Layer Security)** to transfer data between the client (e.g., browser) and the server securely and with guaranteed integrity.

2.2 What are Certificates?

A **certificate** is a digital credential that confirms a public key belongs to a specific identity. Certificates contain:

- The public key
- Information about the owner (e.g., domain name)
- The issuing authority (CA)
- Validity period
- Signature of the CA

2.3 The Role of the Certificate Authority (CA)

A **CA** is a trusted instance that issues and signs certificates. There are:

- **Public CAs** (e.g., Let's Encrypt, DigiCert)
- **Private / self-owned CAs**, used for internal networks

In this guide, a **private CA** is used to fully understand the certificate creation and verification process.

---

3. Setting up a Private Certificate Authority (CA)

Introduction:

To issue our own certificates, we need a **private Certificate Authority (CA)**. This CA is the central point of the chain of trust—it signs server certificates and ensures a server's identity is verifiable.

Further details and examples for the complete CA setup can be found in the document [[2050 CA-sslmitSANZertifikat]].

3.1 CA Directory Structure

If you have already processed the [[2050 CA-sslmitSANZertifikat]] document, you have already completed this step and can skip steps 3 and 4.

A typical directory structure for a private CA might look like this:

```
/home/pdal/myCA/
├─ certs/           # issued certificates
├─ crl/             # certificate revocation lists
├─ newcerts/        # new certificates
├─ private/         # private keys (secret!)
├─ index.txt        # CA database
├─ serial           # serial number of the next certificates
```

3.2 Important Files

```
/home/pdal/myCA/private/ca.key.pem   # private key of the CA
/home/pdal/myCA/certs/ca.cert.pem    # Root certificate of the CA
```

3.3 Creating the Root CA Certificate

Example commands for creating the Root key and Root certificate:

```
# Private key of the Root CA (only accessible to root)
openssl genrsa -out /home/pdal/myCA/private/ca.key.pem 4096

# Create Root certificate (self-signed)
openssl req -x509 -new -nodes -key /home/pdal/myCA/private/ca.key.pem \
  -sha256 -days 3650 -out /home/pdal/myCA/certs/ca.cert.pem \
  -subj "/C=DE/ST=Niedersachsen/L=Wilhelmshaven/O=Hochschule/OU=IT/CN=MyRootCA"
```

Note:

- This structure and these commands allow the CA to be built directly without having to open another document.
- All private keys must be strictly protected to avoid compromising the security of the CA.

---

4. Creating the Server Certificate

Introduction:

In this chapter, we create a certificate for our Apache2 web server on our previously created CA LXC. This certificate will be signed by our CA and enables a secure HTTPS connection in our local network.

4.1 Generating the Key for the Apache2 Server

First, we need to switch to the `root` level to gain access to the `/home/pdal/myCA/private/` directory.

```
sudo -i
```

```
openssl genrsa -out /home/pdal/myCA/private/server.key.pem 2048
chmod 400 /home/pdal/myCA/private/server.key.pem
```

4.2 Creating the Certificate Signing Request (CSR)

We remain logged in as `root`:

```
openssl req -new -key /home/pdal/myCA/private/server.key.pem \
  -out /home/pdal/myCA/server.csr.pem \
  -subj
"/C=DE/ST=Niedersachsen/L=Wilhelmshaven/O=Hochschule/OU=IT/CN=apache.local"
```

4.3 Signing the Certificate with the Private CA

When signing the CSR, we use the `-extfile` option to specify the necessary certificate extensions. These define what the certificate can be used for (e.g., HTTPS) and which Subject Alternative Names (SANs) are valid. We also remain logged in as `root` here.

First, we create the file `server.ext` using an editor (e.g., `nano server.ext`) and the following content.

```
#################################################################################

# Allowed key usages (mandatory for TLS server certificates)
keyUsage=digitalSignature, keyEncipherment

# Extended key usages (serverAuth = HTTPS/TLS)
extendedKeyUsage=serverAuth

#################################################################################
# Subject Alternative Names (SANs)
# This is where the DNS and IP address through which the server
# is reachable are entered.
#################################################################################
```

```
subjectAltName=@alt_names

[alt_names]
DNS.1=apache.local
IP.1=192.168.137.110

##############################################################################
# End of file
##############################################################################
```

Now we can have the certificate signed by our CA.

```
openssl x509 -req \
  -in /home/pdal/myCA/server.csr.pem \
  -CA /home/pdal/myCA/certs/ca.cert.pem \
  -CAkey /home/pdal/myCA/private/ca.key.pem \
  -CAcreateserial \
  -out /home/pdal/myCA/certs/server.cert.pem \
  -days 825 -sha256 \
  -extfile /home/pdal/myCA/server.ext
```

Note:

- `-extfile` specifies the file containing the extensions.
- This file defines, among other things, which key usages are allowed (`keyUsage`), which extended key usages are active (`extendedKeyUsage`), and which SANs are valid.
- Since we are only creating a server certificate for Apache here, we only use one DNS and one IP entry.

4.4 Verifying the Server Certificate

```
openssl x509 -in /home/pdal/myCA/certs/server.cert.pem -text -noout
```

4.5 Copying the Certificates and the Server Key

On the CA container:

```
# Prepare the download folder (optional, if not already present)
sudo mkdir -p /home/pdal/download
sudo chown pdal:pdal /home/pdal/download

# Copy the required files to the download directory
# Caution: The private key of the CA must NOT be copied here!
sudo cp /home/pdal/myCA/certs/ca.cert.pem /home/pdal/download/
sudo cp /home/pdal/myCA/certs/server.cert.pem /home/pdal/download/
sudo cp /home/pdal/myCA/private/server.key.pem /home/pdal/download/
```

The certificate files and the server key are copied from the **CA** to our local client using `WinSCP` or another SFTP client, and then copied to the Apache2 container into the `/home/pdal/download` directory.

On the Apache2 container

We switch to the `root` user for the final copy operations:

```
sudo -i

# Copy the certificates to the public directory
mv /home/pdal/download/ca.cert.pem /etc/ssl/certs/
mv /home/pdal/download/server.cert.pem /etc/ssl/certs/

# Move the private key to the protected directory
mv /home/pdal/download/server.key.pem /etc/ssl/private/

# Set correct permissions for the private key (very important!)
chown root:root /etc/ssl/private/server.key.pem
chmod 600 /etc/ssl/private/server.key.pem
```

---

5. Configuring Apache2 for HTTPS

Introduction:

After the certificate has been created, we must configure the Apache2 web server to accept HTTPS connections and use the created certificates. We use the **global Apache configurations** for the standard ports **80** and **443**.

5.1 Activating the SSL Module

The SSL module is necessary for Apache2 to support HTTPS connections.

```
sudo a2enmod ssl
```

5.2 Global HTTP Configuration (Port 80)

Edit the `/etc/apache2/ports.conf` file and ensure the following entry is present:

```
Listen 80
```

Then edit the standard configuration `/etc/apache2/sites-available/000-default.conf` and ensure the HTTP settings are correctly set:

- Enter the Server Admin email.

- If necessary, adjust the DocumentRoot.

> Note: In an Apache configuration file, the <VirtualHost>-tags must be directly at the beginning of the line and must not contain any leading periods, spaces, or tabs.

```
<VirtualHost *:80>
    ServerAdmin admin@webserver.local
    DocumentRoot /var/www/html

    <Directory /var/www/html>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

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

This configuration ensures that Apache listens on port 80 and delivers the default web content.

5.3 Global HTTPS Configuration (Port 443)

Edit the `/etc/apache2/ports.conf` file and ensure the following entry is present:

```
Listen 443
```

Subsequently, the standard SSL configuration is adjusted. Edit `/etc/apache2/sites-available/default-ssl.conf`:

- Add or adjust the certificate paths.
- Enter the Server Admin email.
- If necessary, adjust the DocumentRoot.

> Note: In an Apache configuration file, the <VirtualHost>-tags must be directly at the beginning of the line and must not contain any leading periods, spaces, or tabs.

```
<VirtualHost *:443>
    ServerAdmin admin@webserver.local
    DocumentRoot /var/www/html

    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/server.cert.pem
    SSLCertificateKeyFile /etc/ssl/private/server.key.pem
    SSLCACertificateFile /etc/ssl/certs/ca.cert.pem

    <Directory /var/www/html>
```

```
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

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

This configuration activates HTTPS on port 443 and integrates the created certificates.

5.4 Activating Configuration and Reloading Apache

To ensure the changes take effect, we must activate the configuration and reload the Apache server.

```
sudo a2ensite default-ssl.conf
sudo systemctl reload apache2
```

This activates HTTPS operation.

5.5 Testing the HTTPS Connection

Call up the server's address in the browser:

```
https://<IP-des-Webservers>
```

If the browser shows a security warning, import the CA certificate of your private Certificate Authority into the browser or operating system.

Important Note on Production Environments

In production systems, you should **always work with VirtualHosts**. This allows multiple websites or services to be operated independently on the same server, each with its own configurations, domains, and certificates. Another important reason for using VirtualHosts is the **scarcity of IPv4 addresses**. Since not every website can receive its own IP address, the concept of **Name-Based VirtualHosts** was developed. Here, a large number of domains share the same IP address, while the hostname is transmitted by the client in the HTTP header. Apache can then deliver the appropriate configuration and website based on the hostname.

Thanks to modern technologies like **SNI (Server Name Indication)**, it is now also possible to operate **multiple HTTPS websites on one IP address**, as the desired hostname is transmitted during the TLS handshake. Without this function, operating multiple SSL certificates simultaneously on one IP address would not be practical.

Without VirtualHosts, it is only possible to use **one global standard configuration** for all requests on ports 80 and 443. This means that only **a single certificate** can be used for all domains, which is generally **not practical** for production environments.

With VirtualHosts, however, each website can use its **own SSL certificate**, which is the recommended procedure for both security and organizational reasons.

Furthermore, VirtualHosts offer a range of other advantages:

- **Better separation and organization:** Each domain or subdomain can have its own configuration file, error pages, logs, and access restrictions.
- **Flexibility in technologies:** Different VirtualHosts can use different PHP versions, proxy targets, or backends (e.g., for separate development and production environments).
- **Security advantages:** Permissions, directories, and modules can be specifically restricted through separate contexts.
- **Scalability:** New domains or services can be easily added without changing existing configurations.
- **Better maintainability:** Changes to one website do not affect other VirtualHosts, reducing misconfigurations.

In short: VirtualHosts are today a **Best Practice** for all web servers, as they enable order, security, flexibility, and efficient use of scarce IPv4 resources, while also supporting the secure use of HTTPS on a shared IP address via SNI.

---

6. Distributing the CA Certificate

Introduction:

For clients to trust the server certificate, they must also trust the issuing CA. This chapter shows how to import the CA certificate on different systems.

If you have already processed the [[2050 CA-sslmitSANZertifikat]] document, you have already completed this step.

6.1 On Linux

Replace `<user>` with the user ID to which you copied the certificates; e.g., `/home/pdal/downloads/...`.

```
# Copies the CA certificate from the download directory to the trust directory
sudo cp /home/<user>/download/certs/ca.cert.pem /usr/local/share/ca-certificates/ca.cert.pem
sudo update-ca-certificates
```

6.2 On Windows

- Press `Windows Key + R`
- Open `certmgr.msc`
- Import `ca.cert.pem` under **Trusted Root Certification Authorities**

6.3 On macOS

- Open "Keychain Access"
- Import `ca.cert.pem` and mark it as **Always Trust**

---

7. Troubleshooting and Tips

Introduction:

This section describes common errors and their solutions to fix typical problems in certificate management and Apache2 configuration.

Problem	Possible Cause	Solution
Browser reports "Connection not secure"	CA certificate not imported	Import the CA into the browser
Apache does not start	Error in SSL configuration	Run `sudo apachectl configtest`
Wrong certificate loaded	Incorrect path or name	Check the certificate file and key <paths of keys and certificates>, <filename of certificates>

---

8. Summary

Introduction:

The summary reiterates the most important points of the entire documentation.

This guide demonstrated:

- Creation and signing of server certificates
- Integration into Apache2 for HTTPS
- Distribution and trust of certificates on clients

This enables fully secured HTTPS communication in a controlled environment (e.g., school network, lab, or company).

---

9. Further Topics

Introduction:

For advanced users, there are many ways to extend or automate the concepts learned here.

- Automated certificate renewal (e.g., with scripts)
- Integration of client certificates for authentication
- Use of Intermediate CAs
- TLS hardening in Apache2 (Cipher Suites, Protocols, HSTS)

---

10. Alternative Methods of Certificate Management

Introduction:

In addition to a private CA, there are other methods for creating and managing certificates. These differ in effort, trust, cost, and automation.

10.1 Public Paid Certificate Authorities

Public CAs (e.g., DigiCert, Sectigo) issue certificates that are automatically trusted by all common browsers and operating systems. They are ideal for production websites but require payment. However, they are only suitable for public domains (not for internal networks without DNS resolution).

10.2 Let's Encrypt

Let's Encrypt is a free, automated, and public CA. It allows for easy setup and automatic renewal of certificates using tools like **Certbot**. However, it is only suitable for public domains (not for internal networks without DNS resolution).

10.3 Self-Signed Certificates without a CA

A **self-signed certificate** is generated directly by the server (tool `openssl`) without a CA. It provides encryption but no trust—browsers will therefore show a warning. This method is only suitable for testing or development environments.

Conclusion

A private CA offers maximum control and is excellently suited for **internal networks** or **teaching environments**. However, for **public websites**, **Let's Encrypt** or commercial CAs are the preferred solution, as they are automatically trusted and minimize maintenance effort.

Sources

- "Documentation". Accessed: October 10, 2025. [Online]. Available at: Let's Encrypt Doc
- "Getting Started". Accessed: October 10, 2025. [Online]. Available at: Let's Encrypt Getting Started
- "How to setup your own CA with OpenSSL", Gist. Accessed: October 10, 2025. [Online]. Available at: How to setup your own CA with OpenSSL
- "Install an SSL Certificate on Apache Mod_SSL", SSL.com. Accessed: October 10, 2025. [Online]. Available at: Install SSL Certificate
- "mod_ssl - Apache HTTP Server Version 2.4". Accessed: October 10, 2025. [Online]. Available at: SSL Modul Apache
- "openssl-ca - OpenSSL Documentation". Accessed: October 10, 2025. [Online]. Available at: OpenSSL Doc
- L. Rendek, "Setting Up a Secure Apache Server on Ubuntu 24.04", LinuxConfig. Accessed: October 10, 2025. [Online]. Available at: Secure Apache on Ubuntu
- "SSL/TLS Strong Encryption: How-To - Apache HTTP Server Version 2.4". Accessed: October 10, 2025. [Online]. Available at: SSL/TLS strong Encryption

---

License