Zertifikatsverwaltung und HTTPS für Apache2

1. Einleitung

Diese Dokumentation behandelt die Grundlagen und praktische Umsetzung der **Zertifikatsverwaltung und HTTPS-Konfiguration** für den Apache2-Webserver unter Linux. Sie richtet sich an **Studierende und Lehrende**, die sich mit Websicherheit, Verschlüsselung und Zertifikatsinfrastruktur vertraut machen möchten.

Ziel ist es, eine eigene **Certificate Authority (CA)** zu erstellen, Zertifikate für Server zu signieren und diese Zertifikate zur Absicherung der Kommunikation zwischen Client und Server (HTTPS) einzusetzen.

---

2. Grundlagen

Einleitung:

Bevor wir mit der praktischen Umsetzung beginnen, ist es wichtig zu verstehen, **warum Zertifikate** und **HTTPS** notwendig sind. In diesem Kapitel werden die theoretischen Grundlagen erklärt, die für das Verständnis der weiteren Schritte notwendig sind.

2.1 Was ist HTTPS?

HTTPS (Hypertext Transfer Protocol Secure) ist die verschlüsselte Variante von HTTP. Es verwendet **TLS (Transport Layer Security)**, um Daten zwischen Client (z. B. Browser) und Server vertraulich und integritätsgesichert zu übertragen.

2.2 Was sind Zertifikate?

Ein **Zertifikat** ist eine digitale Bescheinigung, die bestätigt, dass ein öffentlicher Schlüssel zu einer bestimmten Identität gehört. Zertifikate enthalten:

- Den öffentlichen Schlüssel
- Informationen über den Besitzer (z. B. Domainname)
- Die ausstellende Stelle (CA)
- Gültigkeitszeitraum
- Signatur der CA

2.3 Die Rolle der Certificate Authority (CA)

Eine **CA** ist eine vertrauenswürdige Instanz, die Zertifikate ausstellt und signiert. Es gibt:

- **Öffentliche CAs** (z. B. Let's Encrypt, DigiCert)
- **Private / eigene CAs**, die für interne Netzwerke verwendet werden

In dieser Anleitung wird eine **eigene CA** verwendet, um die Zertifikatserstellung und -prüfung vollständig zu verstehen.

---

3. Einrichtung einer eigenen Certificate Authority (CA)

Einleitung:

Damit wir unsere eigenen Zertifikate ausstellen können, benötigen wir eine **eigene Zertifizierungsstelle (CA)**. Diese CA ist der zentrale Punkt der Vertrauenskette – sie signiert Serverzertifikate und stellt sicher, dass die Identität eines Servers überprüfbar ist.

Weitere Details und Beispiele zur vollständigen Einrichtung der CA findest du im Dokument [[2050 CA-sslmitSANZertifikat]].

3.1 CA-Verzeichnisstruktur

Wer das [[2050 CA-sslmitSANZertifikat]] Dokument bereits bearbeitet hat, hat diesen Schritt bereits erledigt und kann die Schritte 3 und 4 überspringen.

Eine typische Verzeichnisstruktur für die eigene CA könnte wie folgt aussehen:

```
/home/pdal/myCA/
├─ certs/           # ausgestellte Zertifikate
├─ crl/             # Zertifikatsperrlisten
├─ newcerts/        # neue Zertifikate
├─ private/         # private Schlüssel (geheim!)
├─ index.txt        # CA-Datenbank
├─ serial           # Seriennummer der nächsten Zertifikate
```

3.2 Wichtige Dateien

```
/home/pdal/myCA/private/ca.key.pem   # privater Schlüssel der CA
/home/pdal/myCA/certs/ca.cert.pem    # Root-Zertifikat der CA
```

3.3 Root-CA-Zertifikat erstellen

Beispielbefehle zur Erstellung des Root-Schlüssels und des Root-Zertifikats:

```
# Privater Schlüssel der Root-CA (nur für root zugänglich)
openssl genrsa -out /home/pdal/myCA/private/ca.key.pem 4096

# Root-Zertifikat erstellen (selbstsigniert)
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"
```

Hinweis:

- Diese Struktur und Befehle ermöglichen es, die CA direkt nachzubauen, ohne dass ein anderes Dokument geöffnet werden muss.
- Alle privaten Schlüssel müssen streng geschützt werden, um die Sicherheit der CA nicht zu gefährden.

4. Serverzertifikat erstellen

Einleitung:

In diesem Kapitel erstellen wir ein Zertifikat für unseren Apache2-Webserver auf unserem bereits erstellten CA-LXC. Dieses Zertifikat wird von unserer CA signiert und ermöglicht eine sichere HTTPS-Verbindung in unserem lokalen Netzwerk.

4.1 Schlüssel für den Apache2-Server generieren

Zunächst müssen wir auf die `root`-Ebene wechseln, um Zugriff auf das Verzeichnis `/home/pdal/myCA/private/` zu erhalten.

```
sudo -i
```

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

4.2 Certificate Signing Request (CSR) erstellen

Wir bleiben als `root` angemeldet:

```
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 Zertifikat mit der eigenen CA signieren

Beim Signieren des CSR nutzen wir die Option `-extfile`, um die notwendigen Zertifikatserweiterungen (Extensions) anzugeben. Diese definieren, wofür das Zertifikat verwendet werden kann (z. B. HTTPS) und welche Subject Alternative Names (SANs) gültig sind. Auch hier bleiben wir weiterhin als `root` angemeldet.

Zuerst erstellen wir die Datei `server.ext` mit einem Editor (z. B. `nano server.ext`) und folgendem Inhalt.

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

# Erlaubte Schlüsselverwendungen (Pflicht für TLS-Serverzertifikate)
keyUsage=digitalSignature, keyEncipherment

# Erweiterte Schlüsselverwendungen (serverAuth = HTTPS/TLS)
extendedKeyUsage=serverAuth

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

```
# Alternative Namen (SANs = Subject Alternative Names)
# Hier werden DNS- und IP-Adresse eingetragen, über die der Server
# erreichbar ist.
#############################################################################

subjectAltName=@alt_names

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

#############################################################################
# Ende der Datei
#############################################################################
```

Nun können wir das Zertifikat mit unserer CA signieren lassen.

```
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
```

Hinweis:

- `-extfile` gibt die Datei an, die die Extensions enthält.
- In dieser Datei wird u. a. definiert, welche Schlüsselverwendungen erlaubt sind (`keyUsage`), welche erweiterten Schlüsselverwendungen aktiv sind (`extendedKeyUsage`) und welche SANs gültig sind.
- Da wir hier nur ein Serverzertifikat für Apache erstellen, verwenden wir nur einen DNS- und einen IP-Eintrag.

4.4 Prüfung des Serverzertifikats

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

4.5 Kopieren der Zertifikate und des Server-Schlüssels

Auf dem CA-Container:

```
# Vorbereitung des Download-Ordners (optional, falls nicht bereits vorhanden)
sudo mkdir -p /home/pdal/download
sudo chown pdal:pdal /home/pdal/download
```

```
# Kopieren der benötigten Dateien in das Download-Verzeichnis
# Achtung: Der private Schlüssel der CA darf hier NICHT mitkopiert werden!
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/
```

Die Zertifikatsdateien und der Serverschlüssel werden mit Hilfe von `WinSCP` oder einem anderen SFTP Client von der `CA` auf unseren lokalen Client kopiert und anschließend auf den Apache2 Container in das `/home/pdal/download` kopiert.

Auf dem Apache2-Container

Wir wechseln zum Benutzer `root` für die finalen Kopiervorgänge:

```
sudo -i

# Kopieren der Zertifikate in das öffentliche Verzeichnis
mv /home/pdal/download/ca.cert.pem /etc/ssl/certs/
mv /home/pdal/download/server.cert.pem /etc/ssl/certs/

# Verschieben des privaten Schlüssels in das geschützte Verzeichnis
mv /home/pdal/download/server.key.pem /etc/ssl/private/

# Korrekte Rechte für den privaten Schlüssel setzen (sehr wichtig!)
chown root:root /etc/ssl/private/server.key.pem
chmod 600 /etc/ssl/private/server.key.pem
```

---

5. Apache2 für HTTPS konfigurieren

Einleitung:

Nachdem das Zertifikat erstellt wurde, müssen wir den Apache2-Webserver so konfigurieren, dass er HTTPS-Verbindungen akzeptiert und die erstellten Zertifikate verwendet. Wir nutzen die **globalen Apache-Konfigurationen** für die Standardports **80** und **443**.

5.1 SSL-Modul aktivieren

Das SSL-Modul ist notwendig, damit Apache2 HTTPS-Verbindungen unterstützen kann.

```
sudo a2enmod ssl
```

5.2 Globale HTTP-Konfiguration (Port 80)

Bearbeite die Datei `/etc/apache2/ports.conf` und stelle sicher, dass folgender Eintrag vorhanden ist:

```
Listen 80
```

Bearbeite dann die Standardkonfiguration `/etc/apache2/sites-available/000-default.conf` und stelle sicher, dass die HTTP-Einstellungen korrekt gesetzt sind:

- Die Email des Serveradmins eintragen.
- ggf das DocumentRoot anpassen.

> Hinweis: In einer Apache-Konfigurationsdatei müssen die -Tags direkt am Zeilenanfang stehen und dürfen keine führenden Punkte, Leerzeichen oder Tabs enthalten.

```
<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>
```

Diese Konfiguration stellt sicher, dass Apache auf Port 80 lauscht und den Standard-Webinhalt ausliefert.

5.3 Globale HTTPS-Konfiguration (Port 443)

Bearbeite die Datei `/etc/apache2/ports.conf` und stelle sicher, dass folgender Eintrag vorhanden ist:

```
Listen 443
```

Anschließend wird die Standard-SSL-Konfiguration angepasst. Bearbeite dazu `/etc/apache2/sites-available/default-ssl.conf`:

- Die Zertifikatspfade hinzufügen bzw anpassen.
- Die E-mail des Serveradmins eintragen.
- ggf das DocumentRoot anpassen.

> Hinweis: In einer Apache-Konfigurationsdatei müssen die -Tags direkt am Zeilenanfang stehen und dürfen keine führenden Punkte, Leerzeichen oder Tabs enthalten.

```
<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>
```

Diese Konfiguration aktiviert HTTPS auf Port 443 und bindet die erstellten Zertifikate ein.

5.4 Konfiguration aktivieren und Apache neu laden

Damit die vorgenommenen Änderungen wirksam werden, müssen wir die Konfiguration aktivieren und den Apache Server neu laden.

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

Damit ist der HTTPS-Betrieb aktiv.

5.5 Test der HTTPS-Verbindung

Rufe im Browser die Adresse deines Servers auf:

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

Falls der Browser eine Sicherheitswarnung anzeigt, importiere das CA-Zertifikat deiner eigenen Zertifizierungsstelle in den Browser oder das Betriebssystem.

Wichtiger Hinweis zu produktiven Umgebungen

In produktiven Systemen sollte **immer mit VirtualHosts gearbeitet werden**. Dadurch lassen sich mehrere Webseiten oder Dienste auf demselben Server unabhängig voneinander betreiben, mit jeweils eigenen Konfigurationen, Domains und Zertifikaten. Ein weiterer wichtiger Grund für die Nutzung von VirtualHosts ist die **Verknappung von IPv4-Adressen**. Da nicht jede Website eine eigene IP-Adresse erhalten kann, wurde

das Konzept der **Name-Based VirtualHosts** entwickelt. Hierbei teilt sich eine Vielzahl von Domains dieselbe IP-Adresse, während der Hostname vom Client im HTTP-Header übermittelt wird. Apache kann so anhand des Hostnamens die passende Konfiguration und Website ausliefern.

Dank moderner Technologien wie **SNI (Server Name Indication)** ist es heute zudem möglich, auch **mehrere HTTPS-Websites auf einer IP-Adresse** zu betreiben, da der gewünschte Hostname bereits während des TLS-Handshakes übermittelt wird. Ohne diese Funktion wäre der gleichzeitige Betrieb mehrerer SSL-Zertifikate auf einer IP-Adresse nicht praktikabel.

Ohne VirtualHosts ist es nur möglich, **eine globale Standardkonfiguration** für alle Anfragen auf Port 80 und 443 zu verwenden. Das bedeutet, dass nur **ein einziges Zertifikat** für alle Domains genutzt werden kann, was für produktive Umgebungen in der Regel **nicht praktikabel** ist.

Mit VirtualHosts hingegen kann jede Website ihr **eigenes SSL-Zertifikat** verwenden, was sowohl aus Sicherheits- als auch aus organisatorischen Gründen die empfohlene Vorgehensweise ist.

Darüber hinaus bieten VirtualHosts eine Reihe weiterer Vorteile:

- **Bessere Trennung und Organisation:** Jede Domain oder Subdomain kann ihre eigene Konfigurationsdatei, Fehlerseiten, Logs und Zugriffsbeschränkungen besitzen.
- **Flexibilität bei Technologien:** Unterschiedliche VirtualHosts können verschiedene PHP-Versionen, Proxy-Ziele oder Backends nutzen (z. B. für getrennte Entwicklungs- und Produktionsumgebungen).
- **Sicherheitsvorteile:** Durch getrennte Kontexte lassen sich Berechtigungen, Verzeichnisse und Module gezielt einschränken.
- **Skalierbarkeit:** Neue Domains oder Dienste lassen sich einfach hinzufügen, ohne bestehende Konfigurationen zu verändern.
- **Bessere Wartbarkeit:** Änderungen an einer Website wirken sich nicht auf andere VirtualHosts aus, was Fehlkonfigurationen reduziert.

Kurz gesagt: VirtualHosts sind heute ein **Best Practice** für alle Webserver, da sie Ordnung, Sicherheit, Flexibilität und eine effiziente Nutzung der knappen IPv4-Ressourcen ermöglichen und gleichzeitig durch SNI die sichere Nutzung von HTTPS auf einer gemeinsamen IP-Adresse unterstützen.

---

6. CA-Zertifikat verteilen

Einleitung:

Damit Clients dem Serverzertifikat vertrauen, müssen sie auch der ausstellenden CA vertrauen. In diesem Kapitel zeigen wir, wie man das CA-Zertifikat auf verschiedenen Systemen importiert.

Wer das [[2050 CA-sslmitSANZertifikat]] Dokument bereits bearbeitet hat, hat diesen Schritt bereits erledigt.

6.1 Unter Linux

Ersetzen sie `<user>` durch das User-Kennung in das sie die Zertifikate kopiert haben; z. B. `/home/pdal/downloads/...`.

```
# Kopiert das CA-Zertifikat vom Downloadverzeichnis in das Vertrauensverzeichnis
sudo cp /home/<user>/download/certs/ca.cert.pem /usr/local/share/ca-
```

```
certificates/ca.cert.pem
sudo update-ca-certificates
```

6.2 Unter Windows

- drücke `Windows-Taste + R`
- Öffne `certmgr.msc`
- Importiere `ca.cert.pem` unter **Vertrauenswürdige Stammzertifizierungsstellen**

6.3 Unter macOS

- Öffne „Schlüsselbundverwaltung“
- Importiere `ca.cert.pem` und markiere es als **immer vertrauen**

---

7. Fehlersuche und Tipps

Einleitung:

In diesem Abschnitt werden häufige Fehler und deren Lösungen beschrieben, um typische Probleme bei der Zertifikatsverwaltung und Apache2-Konfiguration zu beheben.

Problem	Mögliche Ursache	Lösung
Browser meldet "Verbindung nicht sicher"	CA-Zertifikat nicht importiert	Importiere die CA in den Browser
Apache startet nicht	Fehler in der SSL-Konfiguration	`sudo apachectl configtest` ausführen
Falsches Zertifikat geladen	Pfad oder Name falsch	Überprüfe die Zertifikatsdatei und -schlüssel <Pfade der Schlüssel und Zertifikate>,

---

8. Zusammenfassung

Einleitung:

In der Zusammenfassung werden die wichtigsten Punkte der gesamten Dokumentation noch einmal wiederholt.

Diese Anleitung zeigte:

- Erstellung und Signierung von Serverzertifikaten
- Integration in Apache2 für HTTPS
- Verteilung und Vertrauen von Zertifikaten auf Clients

Damit ist eine vollständig abgesicherte HTTPS-Kommunikation in einer kontrollierten Umgebung (z. B. Schulnetz, Labor oder Unternehmen) möglich.

---

9. Weiterführende Themen

Einleitung:

Für fortgeschrittene Nutzer gibt es viele Möglichkeiten, die hier erlernten Konzepte zu erweitern oder zu automatisieren.

- Automatisierte Zertifikatserneuerung (z. B. mit Skripten)
- Integration von Client-Zertifikaten zur Authentifizierung
- Verwendung von Intermediate CAs
- TLS-Härtung in Apache2 (Cipher Suites, Protokolle, HSTS)

---

10. Alternative Wege der Zertifikatsverwaltung

Einleitung:

Neben einer eigenen CA gibt es auch andere Methoden, Zertifikate zu erstellen und zu verwalten. Diese unterscheiden sich in Aufwand, Vertrauen, Kosten und Automatisierung.

10.1 Öffentliche kostenpflichtige Zertifizierungsstellen

Öffentliche CAs (z. B. DigiCert, Sectigo) stellen Zertifikate aus, die automatisch von allen gängigen Browsern und Betriebssystemen vertraut werden. Sie sind ideal für produktive Websites, aber kostenpflichtig. Allerdings ist sie nur für öffentliche Domains geeignet (nicht für interne Netzwerke ohne DNS-Auflösung).

10.2 Let's Encrypt

Let's Encrypt ist eine kostenlose, automatisierte und öffentliche CA. Sie ermöglicht über Tools wie **Certbot** eine einfache Einrichtung und automatische Erneuerung von Zertifikaten. Allerdings ist sie nur für öffentliche Domains geeignet (nicht für interne Netzwerke ohne DNS-Auflösung).

10.3 Self-Signed Zertifikate ohne CA

Ein **selbstsigniertes Zertifikat** wird direkt vom Server erzeugt, (Tool `openssl`) ohne eine CA. Es bietet Verschlüsselung, aber kein Vertrauen – Browser zeigen daher eine Warnung an. Diese Methode eignet sich nur für Tests oder Entwicklungsumgebungen.

Fazit

Eine eigene CA bietet maximale Kontrolle und eignet sich hervorragend für **interne Netzwerke** oder **Lehrumgebungen**. Für **öffentliche Websites** sind jedoch **Let's Encrypt** oder kommerzielle CAs die bevorzugte Lösung, da sie automatisch vertraut werden und den Wartungsaufwand minimieren.

Quellen

- „Documentation“. Zugegriffen: 10. Oktober 2025. [Online]. Verfügbar unter: Let's Encrypt Doc
- „Getting Started“. Zugegriffen: 10. Oktober 2025. [Online]. Verfügbar unter: Let's Encrypt Getting Started
- „How to setup your own CA with OpenSSL“, Gist. Zugegriffen: 10. Oktober 2025. [Online]. Verfügbar unter: How to setup your own CA with OpenSSL
- „Install an SSL Certificate on Apache Mod_SSL“, SSL.com. Zugegriffen: 10. Oktober 2025. [Online]. Verfügbar unter: Install SSL Certificate

- „mod_ssl - Apache HTTP Server Version 2.4“. Zugegriffen: 10. Oktober 2025. [Online]. Verfügbar unter: SSL Modul Apache
- „openssl-ca - OpenSSL Documentation“. Zugegriffen: 10. Oktober 2025. [Online]. Verfügbar unter: OpenSSL Doc
- L. Rendek, „Setting Up a Secure Apache Server on Ubuntu 24.04“, LinuxConfig. Zugegriffen: 10. Oktober 2025. [Online]. Verfügbar unter: Secure Apache on Ubuntu
- „SSL/TLS Strong Encryption: How-To - Apache HTTP Server Version 2.4“. Zugegriffen: 10. Oktober 2025. [Online]. Verfügbar unter: SSL/TLS strong Encryption

---

Lizenz