**What is a PFX File?**

A **PFX** file (also known as **PKCS#12**, .pfx or .p12) is a **bundle** that contains:

* The **private key**
* The **public certificate**
* The **certificate chain** (intermediate and root CAs)

It is **password-protected** and used to **export and import SSL certificates securely**, especially across platforms like Windows, NGINX, Apache, Java (KeyStore), etc.

**Use Cases of PFX**

* Importing SSL certificates into **IIS (Windows Server)**
* Using certificates in **Java KeyStores (JKS)**
* Uploading to cloud services (Azure, AWS, etc.)
* Backing up and transferring certificate + private key securely

**How to Create a PFX from Your Files**

Assuming I have:

* myserver.crt (your server certificate)
* myserver.key (your private key)
* myCA.crt (your CA/root certificate)

**Step-by-Step PFX Practice with OpenSSL**

**Step 1: Create a PFX File**

openssl pkcs12 -export \

-inkey myserver.key \

-in myserver.crt \

-certfile myCA.crt \

-out mycert.pfx

This creates mycert.pfx, which combines:

* Server certificate
* Private key
* CA certificate

**Step 2: View Contents of the PFX**

openssl pkcs12 -info -in mycert.pfx

**Step 3: Extract Private Key from PFX**

openssl pkcs12 -in mycert.pfx -nocerts -out extracted.key

**Step 4: Extract Certificate from PFX**

openssl pkcs12 -in mycert.pfx -clcerts -nokeys -out extracted.crt

**Step 5: Extract CA Certificates from PFX**

openssl pkcs12 -in mycert.pfx -cacerts -nokeys -out extracted\_ca.crt

**Use PFX with Java KeyStore (JKS)**

**Convert .pfx to .jks using keytool (Java’s certificate tool):**

keytool -importkeystore \

-srckeystore mycert.pfx \

-srcstoretype PKCS12 \

-destkeystore mykeystore.jks \

-deststoretype JKS

**Use PFX in Docker Container**

**Dockerfile snippet (Nginx + OpenSSL tools)**

FROM nginx:alpine

COPY mycert.pfx /etc/ssl/private/mycert.pfx

RUN apk add --no-cache openssl \

&& openssl pkcs12 -in /etc/ssl/private/mycert.pfx -out /etc/ssl/private/server.crt -clcerts -nokeys -passin pass:YOURPASSWORD \

&& openssl pkcs12 -in /etc/ssl/private/mycert.pfx -out /etc/ssl/private/server.key -nocerts -nodes -passin pass:YOURPASSWORD

**Use PFX with NGINX or Apache**

openssl pkcs12 -in mycert.pfx -clcerts -nokeys -out /etc/ssl/certs/server.crt

openssl pkcs12 -in mycert.pfx -nocerts -nodes -out /etc/ssl/private/server.key

**Convert .pfx ➝ .crt and .key**

**Prepare Your Test Environment**

mkdir ~/ssl-practice && cd ~/ssl-practice

**Generate a Private Key & Self-Signed Certificate**

openssl req -x509 -newkey rsa:2048 -nodes \

-keyout private.key \

-out certificate.crt \

-days 365 \

-subj "/CN=localhost"

**Convert CRT + KEY ➝ PFX**

openssl pkcs12 -export \

-out mycert.pfx \

-inkey private.key \

-in certificate.crt

**Convert PFX ➝ CRT + KEY**

openssl pkcs12 -in mycert.pfx -clcerts -nokeys -out extracted.crt

openssl pkcs12 -in mycert.pfx -nocerts -nodes -out extracted.key

**What is a PEM File?**

**PEM** stands for **Privacy Enhanced Mail**, and it's a **Base64-encoded** format for storing:

* ✅ Certificates (.crt, .cert)
* ✅ Private keys (.key)
* ✅ Certificate bundles (full chains)

**Practice with PEM Files**

**Create a PEM Certificate and Key**

cd ~/ssl-practice

**Generate a new self-signed PEM certificate and key**

openssl req -x509 -newkey rsa:2048 -nodes \

-keyout pemkey.key \

-out pemcert.pem \

-days 365 \

-subj "/CN=localhost"

**Combine Both into One .pem File**

cat pemcert.pem pemkey.key > full.pem

**full.pem → one PEM file with both the cert and key**

cat full.pem

**Certificate Chains**

**What is a Certificate Chain?**

A certificate chain (or chain of trust) is a sequence of certificates where:

1. Each certificate is signed by the next authority in the chain,
2. Until we reach a trusted Root Certificate Authority (Root CA).

**Chain Order (Bottom to Top)**

1. Leaf Certificate (Your server’s certificate – e.g., yourdomain.com)
2. Intermediate Certificate(s) (Issued by Root or another intermediate)
3. Root Certificate (Trusted by operating systems and browsers)

**Why We Use a Certificate Chain (Certificate Hierarchy)**

A certificate chain builds trust between your SSL/TLS certificate and a trusted root authority.

**Breakdown:**

1. Your Certificate (Leaf Certificate)  
   Proves your domain identity (e.g., myserver.crt).
2. Intermediate Certificate(s)  
   Issued by a Root CA and used to issue end-entity (your) certificates.  
   Acts as a middle layer of trust — if compromised, the root CA is still safe.
3. Root Certificate  
   The ultimate trust anchor, pre-installed in browsers and OSs (e.g., myCA.crt).

**Why the Chain Matters:**

* Browsers don’t trust your certificate directly.
* They trust root CAs, and only if your cert can be linked (chained) to a known root via intermediate(s), it's accepted.
* Without a complete chain, users may see ❌ "Untrusted Certificate" warnings.

**How to Practice**

**Concatenate Full Chain**

cat myserver.crt intermediate.crt myCA.crt > fullchain.pem

**View Chain with OpenSSL**

openssl crl2pkcs7 -nocrl -certfile fullchain.pem | openssl pkcs7 -print\_certs -text -noout

**Or view individual certs**

openssl x509 -in myserver.crt -text -noout

openssl x509 -in intermediate.crt -text -noout

openssl x509 -in myCA.crt -text -noout

**Validate the Chain**

openssl verify -CAfile myCA.crt -untrusted intermediate.crt myserver.crt

| **Certificate Type** | **Purpose** |
| --- | --- |
| Leaf Certificate | Identifies your server |
| Intermediate CA | Bridges leaf and root, not self-signed |
| Root CA | Trusted by browsers, self-signed |

The goal is to **chain trust from the server to a trusted authority**.

**How to create a CSR (Certificate Signing Request)**

Creating a CSR (Certificate Signing Request) is a key step in getting your SSL certificate signed by a Certificate Authority (CA). Here's how to do it using OpenSSL

**Steps to Create a CSR using OpenSSL**

need:

* A private key (you can create one during the process).
* OpenSSL installed (already on most Linux distros).

**Generate a Private Key and CSR**

openssl req -new -newkey rsa:2048 -nodes -keyout myserver.key -out myserver.csr

**rsa:2048 = 2048-bit RSA key**

**-nodes = do not encrypt the private key with a passphrase (recommended for servers)**

**myserver.key = private key output**

**myserver.csr = your Certificate Signing Request**

**Prompted for Certificate Info**

**Asked-**

| **Field** | **Description** |
| --- | --- |
| **Country Name** | **2-letter country code (e.g., US, LK)** |
| **State or Province** | **Your state or province** |
| **Locality** | **City** |
| **Organization Name** | **Your company name or personal project** |
| **Organizational Unit** | **Division (can leave blank)** |
| **Common Name (CN)** | **Domain name (e.g., www.example.com)** |

**Optional fields**

* **A challenge password (leave blank)**
* **An optional company name**

**Output Files**

**myserver.key – your private key (keep this secure!)**

**myserver.csr – submit this to a CA or use it to sign your own cert**

**Wildcard TLS/SSL Certificate**

**What Is a Wildcard Certificate**

**A wildcard certificate allows you to secure one level of subdomains for a given domain using a single certificate. It uses an asterisk \* as a placeholder for subdomains.**

\*.example.com

**This will cover:**

* **www.example.com**
* **api.example.com**
* **mail.example.com**
* **anything.example.com**

**But not:**

* **example.com (you must request both \*.example.com and example.com)**
* **sub.mail.example.com (only one level of subdomain is covered)**

**How to Create a CSR for a Wildcard Cert with OpenSSL**

openssl req -new -newkey rsa:2048 -nodes \

-keyout wildcard.example.com.key \

-out wildcard.example.com.csr

**When to Use a Wildcard Certificate**

* Want to secure **many subdomains** on the **same level**.
* Want **simplified certificate management**.
* Don’t want to re-issue a cert for every new subdomain.

**Considerations**

| **Pros** | **Cons** |
| --- | --- |
| Easy management (one cert) | All subdomains share the same private key |
| Covers unlimited subdomains | Not valid for sub-subdomains |
| Cost-effective | Less secure than individual certs in large orgs |

**Wildcard vs SAN Certificates**

| **Wildcard Certificate** | **SAN (Subject Alternative Name) Certificate** |
| --- | --- |
| Covers all subdomains at one level | Explicitly lists all domains/subdomains |
| Easier if you have many subdomains | Better control over exactly what’s secured |
| Example: \*.example.com | Example: www.example.com, mail.example.net |