**SSL Command Practice**

**1. What is SSL/TLS?**

* **SSL** (Secure Sockets Layer) and its successor **TLS** (Transport Layer Security) are protocols that encrypt data sent over the internet.
* SSL ensures **data integrity**, **confidentiality**, and **authentication**.

**Difference between SSL and TLS**

* **SSL (Secure Sockets Layer)** is the original encryption protocol for securing data over the internet.
* **TLS (Transport Layer Security)** is the newer, more secure version of SSL.
* TLS has replaced SSL entirely — **SSL is now deprecated** (TLS 1.2 and TLS 1.3 are the current standards).

**Why HTTPS Uses SSL/TLS**

* **HTTPS = HTTP + SSL/TLS**
* HTTPS uses SSL/TLS to **encrypt the data** between your browser and the server, preventing attackers from reading or tampering with it.
* It also **authenticates the server**, proving that the user is communicating with the legitimate website (via certificates).

**How Encryption and Decryption Work (Basic Public/Private Key Concepts)**

* A **public key** is shared with anyone and used to **encrypt** data.
* A **private key** is kept secret and used to **decrypt** that data.
* When you connect to an HTTPS site:
  + Your browser uses the **server’s public key** to encrypt data.
  + Only the server’s **private key** can decrypt it — securing the communication.

**2. What is an SSL Certificate?**

* A digital certificate that authenticates a website's identity and enables encrypted connections.
* Issued by a **Certificate Authority (CA)**.
* Contains:
  + Domain name
  + Organization details
  + Public key
  + Expiry date
  + Digital signature from the CA

**How Browsers Validate SSL Certificates**

When you visit an HTTPS website, browser:

1. **Receives the website’s SSL certificate**.
2. **Checks if it’s issued by a trusted Certificate Authority (CA)**.
3. **Verifies the certificate’s validity** (expiration date, domain match, not revoked).
4. **Builds a chain of trust** from your site’s cert → intermediate CA(s) → root CA.
5. If all checks pass, it shows the **lock icon** 🔒 and allows a secure connection.

If any step fails, the browser shows a **security warning**.

**2. Types of SSL Certificates**

| **Type** | **Validated Info** | **Purpose** | **Display in Browser** |
| --- | --- | --- | --- |
| **DV (Domain Validated)** | Only verifies domain ownership | Basic SSL for blogs, personal sites | 🔒 Lock icon only |
| **OV (Organization Validated)** | Verifies domain + organization identity | Business websites | Lock + shows org info if clicked |
| **EV (Extended Validation)** | Strictest checks: domain, org, legal status | High-trust sites (banks, large companies) | Lock + org name in address bar (on some browsers) |

**What is PKI (Public Key Infrastructure)?**

**PKI** is a system of technologies, policies, and procedures used to **secure digital communication** using **public-key cryptography**.

**In Simple Terms:**

PKI provides the **framework** that allows you to:

* **Encrypt data**
* **Digitally sign information**
* **Authenticate identities** (like websites, users, or devices)

**Core Components of PKI:**

1. **Public and Private Keys**
   * A unique key pair used for encryption and decryption.
2. **Certificate Authority (CA)**
   * A trusted entity that **issues digital certificates** verifying identity.
3. **Digital Certificates**
   * Files that bind a public key to a verified identity (like a domain or organization).
4. **Registration Authority (RA)**
   * Verifies identity before a certificate is issued by the CA.
5. **Certificate Revocation List (CRL)** / OCSP
   * Used to **check if a certificate is no longer trusted** (e.g., if compromised).

**Step-by-step hands-on SSL/TLS practice guide using OpenSSL on Linux**

**Create a Private Key and Certificate (Self-signed)**

openssl req -x509 -newkey rsa:2048 -keyout privkey.pem -out cert.pem -days 365 -nodes -subj "/CN=localhost"

**privkey.pem: private key**

**cert.pem: self-signed SSL certificate**

**Build Your Own CA (Certificate Authority)**

**Step 1: Create a CA key and cert**

openssl genrsa -out myCA.key 2048

openssl req -x509 -new -key myCA.key -out myCA.crt -days 365 -subj "/CN=MyCustomCA"

**Step 2: Create a key for your server (domain)**

openssl genrsa -out myserver.key 2048

**Step 3: Create a Certificate Signing Request (CSR)**

openssl req -new -key myserver.key -out myserver.csr -subj "/CN=localhost"

**Step 4: Sign the CSR with your custom CA**

openssl x509 -req -in myserver.csr -CA myCA.crt -CAkey myCA.key -CAcreateserial -out myserver.crt -days 365

Now I have:

* myserver.key (private key)
* myserver.crt (SSL cert signed by your CA)
* myCA.crt (trusted root CA)

**3. Create a Full Chain (Bundle)**

cat myserver.crt myCA.crt > fullchain.pem

**4. Use It With NGINX**

**Create NGINX Config (minimal)**

**Edit /etc/nginx/sites-available/default or create a new file**

server {

listen 443 ssl;

server\_name localhost;

ssl\_certificate /path/to/fullchain.pem;

ssl\_certificate\_key /path/to/myserver.key;

location / {

return 200 "SSL is working!\n";

}

}

**Restart NGINX**

sudo nginx -t

sudo systemctl restart nginx

**Test it**

curl -k <https://localhost>

**View the Private Key – myserver.key**

openssl rsa -in myserver.key -check

**See-**

**The modulus, public exponent, and key parameters.**

**Confirms the key is valid.**

**View the SSL Certificate – myserver.crt**

openssl x509 -in myserver.crt -text -noout

**See-**

**Issuer: Should be your custom CA (MyCustomCA)**

**Subject: The domain name (e.g., CN=localhost)**

**Validity: Start and expiry dates**

**Public Key info**

**Signature algorithm**

**View the CA Certificate – myCA.crt**

openssl x509 -in myCA.crt -text -noout

**See-**

**Similar info as the server cert, but the Issuer and Subject are the same (since it’s self-signed).**

**Look for: CA:TRUE in the Basic Constraints section**

**Verify Certificate Chain**

openssl verify -CAfile myCA.crt myserver.crt