<https://sectigostore.com/page/wildcard-ssl-vs-standard-ssl-whats-the-difference/>

<https://www.sectigo.com/resource-library/different-types-of-ssl-certificates-explained>

Self-Signed Certificate vs. Wildcard Certificate

Both self-signed certificates and wildcard certificates are used for SSL/TLS encryption, but they serve different purposes and have different levels of trust and validation.

1. Self-Signed Certificate

A self-signed certificate is created and signed by the same entity that issues it, rather than a trusted Certificate Authority (CA).

🔹 Use Case:

Internal testing

Development environments

Local servers where security is not a primary concern

🔹 Pros:  
✅ Free to generate  
✅ Quick setup without a CA  
✅ Useful for internal purposes

🔹 Cons:  
❌ Not trusted by browsers (causes security warnings)  
❌ Must be manually installed on client systems  
❌ No third-party validation (can be vulnerable to MITM attacks)

🔹 Example:  
Generating a self-signed certificate with OpenSSL:

openssl req -x509 -newkey rsa:2048 -keyout selfsigned.key -out selfsigned.crt -days 365 -nodes

After generating, you must manually trust it in browsers or applications.

2. Wildcard Certificate

A wildcard certificate is issued by a Certificate Authority (CA) and is valid for a domain and all its subdomains.

🔹 Use Case:

Securing multiple subdomains (e.g., example.com, app.example.com, blog.example.com)

Public-facing websites and applications

🔹 Pros:  
✅ Trusted by browsers (no security warnings)  
✅ Covers all subdomains under a single certificate  
✅ Saves cost compared to buying individual certificates

🔹 Cons:  
❌ Requires payment (not free)  
❌ If compromised, all subdomains are at risk  
❌ More complex to issue and renew

🔹 Example:  
A wildcard certificate for \*.example.com would secure:

www.example.com

mail.example.com

shop.example.com

Key Differences

| Feature | Self-Signed Certificate | Wildcard Certificate |
| --- | --- | --- |
| Issuer | Self-signed (local) | Trusted CA (e.g., Let's Encrypt, DigiCert) |
| Trust Level | Not trusted (causes browser warnings) | Trusted by browsers |
| Use Case | Internal testing, local use | Public websites with multiple subdomains |
| Cost | Free | Paid (or free from Let's Encrypt for limited cases) |
| Security | Lower (no third-party validation) | Higher (CA validation ensures authenticity) |
| Covers | Single domain only | Main domain + all subdomains |

When to Use Which?

✅ Self-Signed: Internal projects, testing, local environments.  
✅ Wildcard: Public websites needing SSL across multiple subdomains.

Would you like help generating or installing one? 🚀

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### ****Configuring Self-Signed and Wildcard Certificates for an Application Server****

Below are step-by-step guides for configuring **self-signed** and **wildcard SSL certificates** on an application server (e.g., Nginx, Apache).

## ****1. Configuring a Self-Signed Certificate****

A **self-signed certificate** is useful for **internal** or **development environments**.

### ****Step 1: Generate a Self-Signed Certificate****

Run the following command to generate a self-signed certificate valid for example.com for **1 year (365 days)**:

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout selfsigned.key -out selfsigned.crt

* -x509: Creates a self-signed certificate.
* -nodes: Prevents encrypting the private key.
* -days 365: Valid for 1 year.
* -newkey rsa:2048: Generates a 2048-bit RSA key.

You'll be asked to enter details like Country, State, Organization, and Common Name (enter your domain, e.g., example.com).

### ****Step 2: Configure the Web Server****

#### ****For Nginx****

Edit the Nginx configuration file (e.g., /etc/nginx/sites-available/default):

server {

listen 443 ssl;

server\_name example.com;

ssl\_certificate /etc/ssl/selfsigned.crt;

ssl\_certificate\_key /etc/ssl/selfsigned.key;

location / {

root /var/www/html;

index index.html;

}

}

Then restart Nginx:

sudo systemctl restart nginx

#### ****For Apache****

Edit the Apache SSL configuration file (e.g., /etc/apache2/sites-available/default-ssl.conf):

<VirtualHost \*:443>

ServerName example.com

SSLEngine on

SSLCertificateFile /etc/ssl/selfsigned.crt

SSLCertificateKeyFile /etc/ssl/selfsigned.key

DocumentRoot /var/www/html

</VirtualHost>

Then restart Apache:

sudo a2enmod ssl

sudo a2ensite default-ssl

sudo systemctl restart apache2

### ****Step 3: Trust the Certificate (For Local Development)****

Since browsers **don't trust self-signed certificates**, you'll need to **manually trust it**:

* In **Chrome/Firefox**, go to chrome://settings/certificates or about:preferences#privacy and import the .crt file.
* On **Linux/macOS**, add it to the trusted CA store:

sudo cp selfsigned.crt /usr/local/share/ca-certificates/selfsigned.crt

sudo update-ca-certificates

## ****2. Configuring a Wildcard Certificate****

A **wildcard SSL certificate** is used when you need to secure multiple subdomains like:  
🔹 example.com, app.example.com, blog.example.com, etc.

### ****Step 1: Generate a Wildcard CSR (Certificate Signing Request)****

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

Enter \*.example.com as the **Common Name (CN)**.

### ****Step 2: Purchase & Obtain the Certificate****

* Submit wildcard.csr to a trusted CA like **Let's Encrypt, DigiCert, or GlobalSign**.
* The CA will issue wildcard.crt and wildcard.ca-bundle.crt.

### ****Step 3: Configure the Web Server****

#### ****For Nginx****

Edit /etc/nginx/sites-available/default:

server {

listen 443 ssl;

server\_name \*.example.com;

ssl\_certificate /etc/ssl/wildcard.crt;

ssl\_certificate\_key /etc/ssl/wildcard.key;

ssl\_trusted\_certificate /etc/ssl/wildcard.ca-bundle.crt;

location / {

root /var/www/html;

index index.html;

}

}

Restart Nginx:

sudo systemctl restart nginx

#### ****For Apache****

Edit /etc/apache2/sites-available/default-ssl.conf:

<VirtualHost \*:443>

ServerName example.com

ServerAlias \*.example.com

SSLEngine on

SSLCertificateFile /etc/ssl/wildcard.crt

SSLCertificateKeyFile /etc/ssl/wildcard.key

SSLCertificateChainFile /etc/ssl/wildcard.ca-bundle.crt

DocumentRoot /var/www/html

</VirtualHost>

Restart Apache:

sudo a2enmod ssl

sudo systemctl restart apache2

## ****Final Thoughts****

✅ **Use Self-Signed for:** Internal testing, development, local servers.  
✅ **Use Wildcard SSL for:** Public applications requiring secure subdomains.

Would you like help automating SSL renewal using Let's Encrypt? 🚀

### ****What is an Nginx Server?****

**Nginx** (pronounced **"Engine-X"**) is a **high-performance web server** that also functions as:

* **Reverse Proxy** (for load balancing)
* **Mail Proxy** (for SMTP, IMAP, and POP3)
* **API Gateway** (for microservices)

🔹 **Why Use Nginx?**  
✅ Faster than Apache (event-driven architecture)  
✅ Handles **high traffic loads** efficiently  
✅ Supports **static & dynamic content**  
✅ **Reverse proxy & load balancing** for microservices

## ****Example: Basic Nginx Web Server Setup****

### ****1. Install Nginx****

#### ****On Ubuntu/Debian:****

sudo apt update

sudo apt install nginx -y

#### ****On CentOS/RHEL:****

sudo yum install epel-release -y

sudo yum install nginx -y

#### ****Start Nginx & Enable on Boot:****

sudo systemctl start nginx

sudo systemctl enable nginx

#### ****Check if Nginx is Running:****

systemctl status nginx

### ****2. Configure a Basic Nginx Web Server****

#### ****Default Configuration File:**** /etc/nginx/sites-available/default (Ubuntu)

Or /etc/nginx/nginx.conf (CentOS)

Edit the file:

sudo nano /etc/nginx/sites-available/default

Add the following configuration:

server {

listen 80;

server\_name example.com;

root /var/www/html;

index index.html;

location / {

try\_files $uri $uri/ =404;

}

}

Save and restart Nginx:

sudo systemctl restart nginx

### ****3. Test the Nginx Web Server****

Create an **index.html** page:

echo "<h1>Welcome to Nginx!</h1>" | sudo tee /var/www/html/index.html

Now, open a browser and visit:  
👉 [http://your-server-ip](http://your-server-ip/)

You should see **"Welcome to Nginx!"**

## ****4. Example: Nginx as a Reverse Proxy for Backend Server****

If you have a backend running on localhost:5000, you can use Nginx to forward requests:

server {

listen 80;

server\_name example.com;

location / {

proxy\_pass http://localhost:5000;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

}

Restart Nginx:

sudo systemctl restart nginx

Now, all requests to http://example.com will be forwarded to http://localhost:5000.

### ****5. Example: Enabling SSL with Let's Encrypt****

Install **Certbot** and get a free SSL certificate:

sudo apt install certbot python3-certbot-nginx -y

sudo certbot --nginx -d example.com -d www.example.com

Auto-renew SSL:

sudo certbot renew --dry-run

## ****Final Thoughts****

✅ **Nginx is fast, scalable, and efficient**  
✅ **Great for serving static files and acting as a reverse proxy**  
✅ **Can be combined with SSL for secure connections**

Do you need help setting up Nginx with Docker, PHP, or other services? 🚀