HTTPS Certificate

Using Let's Encrypt/mkcert

What's the meaning of Certificate in HTTPS

In HTTPS, a certificate is like an official ID card for your website.

It proves two key things:

1. Identity — Confirms that the website really belongs to the domain owner (e.g., example.com), not an imposter.
2. Encryption — Contains the public key used to securely exchange data with visitors via SSL/TLS.

CA - Let's Encrypt

The certificate is issued by a Certificate Authority (CA) — a trusted organization that verifies domain ownership: In this example, we use Let’s Encrypt.

- It uses the **ACME** (Automatic Certificate Management Environment) protocol.
- The web server proves ownership through an HTTP or DNS challenge.
- Once verified, Let’s Encrypt issues a trusted certificate.
- **Certbot** automates the process of requesting, installing, and renewing certificates.

1. Goal: Prove Domain Ownership

Before issuing an HTTPS certificate, Let's Encrypt must confirm that you control the domain (e.g., `example.com`).

2. The Challenge–Response Model

Let's Encrypt uses an automated protocol called **ACME** (Automatic Certificate Management Environment).

ACME protocol overview:

1. **Request**: Your server (Certbot) asks Let's Encrypt for a certificate.
2. **Challenge**: Let's Encrypt replies:
   > "If you own example.com, place this secret token at a specific location."
3. **Response**: Your server places that token (file, DNS record, etc.)
4. **Verify**: Let's Encrypt visits your domain (HTTP or DNS) to check it exists.
5. **Issue**: If valid, the CA issues a signed certificate.

3. Types of Validation

Type	Method	Example
HTTP-01	Put token under `.well-known/acme-challenge`	For web servers
DNS-01	Add TXT record to DNS	For wildcard domains
TLS-ALPN-01	Use TLS handshake	For advanced setups

4. What the Certificate Contains

When verified, the CA issues a **digital certificate** that includes:

- Your domain name (CN = example.com)
- Public key (you generated a private/public key pair)
- Issuer info (Let’s Encrypt)
- Validity period (90 days)
- CA signature (proves authenticity)

5. The HTTPS Trust Chain

```
Browser
  ↓ trusts
Root CA (ISRG Root X1)
  ↓ signs
Intermediate CA (R3)
  ↓ signs
Your Domain Certificate
```

- ISRG (Internet Security Research Group) is the organization behind Let’s Encrypt.
- Let’s Encrypt issues certificates through its intermediate CA (R3), which is signed by the root CA (ISRG Root X1) trusted by browsers.

Install and run Certbot (Ubuntu)

> To install and run Certbot, you should own your domain name and VPS.
> In most cases, the web server is run on Linux, so I use Ubuntu Linux as an example.

1. Install Certbot

```
sudo apt update
sudo apt install certbot python3-certbot-nginx
```

2. Run Certboot

```
# Get certificate (requires public domain)
sudo certbot --nginx -d example.com -d www.example.com
```

Then certbot does the following (details in the next section):

1. Domain Verification (ACME Challenge)
2. Certificate Issuance
3. Nginx Configuration (optional, only with --nginx)

3. Renewal and Automation

- Let’s Encrypt certificates are valid for 90 days to reduce security risks.
- Certbot automatically handles renewals by repeating the same ACME verification process.
- Use the command above to test that automatic renewal works correctly before relying on it in production.

```
sudo certbot renew --dry-run
```

Detailed Certboot Actions

Domain Verification (ACME Challenge)

1. Certbot contacts Let's Encrypt:

> "I want a certificate for example.com"

2. Let's Encrypt replies:

> "Prove you control example.com"

3. Certbot creates challenge file:

- Certbot parses your existing Nginx configuration — it looks for the server_name that matches the domain (example.com) and reads its root or location directives.
- From this, Certbot learns that requests to `http://example.com/` are served from `/var/www/html` (for example).

```
/var/www/html/.well-known/acme-challenge/<token>
```

4. Let's Encrypt visits:

```
http://example.com/.well-known/acme-challenge/<token>
```

Certificate Issuance

- Let's Encrypt **signs** your public key.
- Certbot downloads:

File	Purpose
`/etc/letsencrypt/live/example.com/fullchain.pem`	Signed certificate
`/etc/letsencrypt/live/example.com/privkey.pem`	Private key

Nginx Configuration (optional, only with --nginx)

1. Certbot scans your `/etc/nginx/sites-enabled/` (or `/etc/nginx/conf.d/`) for server_name entries that match the domain(s) you specify.
2. It inserts SSL directives into the matching server block, such as:

```
ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
include /etc/letsencrypt/options-ssl-nginx.conf;
```

3. It also adds a redirect from HTTP → HTTPS by editing or creating a separate block:

```
server {
    listen 80;
    server_name example.com www.example.com;
    return 301 https://$host$request_uri;
}
```

4. Then Certbot reloads Nginx automatically so the new certificate takes effect.

Local HTTPS using mkcert

Browsers warn on self-signed certs.

- Use **mkcert** to create locally trusted certs for development.
- mkcert adds a local root CA to your OS trust store, then issues certs for `localhost`, custom dev domains, etc.
- For dev only — never use mkcert certs in production.

What mkcert is (and isn't)

- mkcert creates locally-trusted certificates by installing a local CA on *your* machine.
- Perfect for development (no browser warnings).
- Not for production => Use **Let's Encrypt** in production.

Key point:

- Browsers trust your mkcert certs **only on machines where the mkcert CA is installed**.

Quick Start (mkcert)

Install mkcert and the local CA (one-time)

```
# Mac
brew install mkcert nss         # macOS (add nss for Firefox)

# For Ubuntu (update for your CPU)

sudo apt install libnss3-tools -y
curl -JLO "<https://dl.filippo.io/mkcert/latest?for=linux/amd64>"
```

```
> mkcert -install
> mkcert localhost 127.0.0.1 ::1 myapp.local

Created a new certificate valid for the following names 📜
 - "localhost"
 - "127.0.0.1"
 - "::1"
 - "myapp.local"

The certificate is at "./localhost+3.pem" and the key at "./localhost+3-key.pem" ✅

It will expire on 10 February 2028 🗓️
```

Usage of mkcert

Use with Node.js (Express)

```
import fs from 'fs';
import https from 'https';
import express from 'express';

const app = express();
app.get('/', (_req, res) => res.send('Hello HTTPS (local)'));

https.createServer({
  key:  fs.readFileSync('./certs/localhost+3-key.pem'),
  cert: fs.readFileSync('./certs/localhost+3.pem'),
}, app).listen(8443, () => {
  console.log('✅ HTTPS server running at https://localhost:8443');
});
```

Example Directory Structure

You can put them in the `certs` directory.

```
dev-https/
├─ certs/
│  ├─ localhost+3.pem
│  └─ localhost+3-key.pem
└─ index.js
```

Copy this app inside dev-https/, you can start it directly from that folder (e.g., node index.js).

Use with Nginx (only for development purposes)

```
> mkcert -install
Created a new local CA 💥
Sudo password:
The local CA is now installed in the system trust store!
```

- Make sure mkcert is installed on your local computer.
- You can't reuse the certificate generated for other computers.

1. Create a subdirectory `certs` in the Nginx configuration directory.
2. Copy the `nginx_nodejs_secure.conf` file in the `servers` or `sites-enabled` directory.

It has the same `upstream` block.

```
upstream nodejs_app {
    server localhost:3000;
    keepalive 8;
}
```

It redirects the port 8080 to 8443.

```
# HTTP Server - Redirect to HTTPS
server {
    listen 8080;
    server_name localhost;

    # Redirect all HTTP to HTTPS
    return 301 https://$server_name:8443$request_uri;
}
```

It listens to 8443 port (HTTPS).

- Be sure to change the location of the certificates.

```
# HTTPS Server
server {
    listen 8443 ssl;
    server_name localhost;

    # SSL Certificate and Key
    ssl_certificate /opt/homebrew/etc/nginx/certs/localhost+3.pem;
    ssl_certificate_key /opt/homebrew/etc/nginx/certs/localhost+3-key.pem;
    ...
}
```