

# HTTPS

## Dag 2: HTTPS.

---

### Indhold:

1. Faglig mål .....	2
2. Øvelse (Case).....	3
Baggrund .....	3
Krav .....	3
Hvordan du bliver bedømmet .....	4

### Hjælpe materialer (teori) til gennemførsel af øvelserne:

1. HTTPS handshake process .....	5
2. Self-sign certifikat vs. godkendt CA .....	5
3. Erstatning af self-sign certifikat med en godkendt CA .....	6

## Faglige mål:

Dagens øvelse dækker følgende målpinde:

1. *Eleven kan udvikle serverside webapplikationer, der kan levere HTML-kode til browseren, samt Web API eller webservices, som kan udveksle data med en client-application, f.eks. en browser eller en mobil App.*
  2. *Eleven kan redegøre for forskellige arkitekturen for web Applikationer og web API (web Services), med fordele og ulemper.*
  3. *Eleven kan opbygge og konfigurere en web Application og web API (web service) vha. et framework.*
  4. *Eleven kan benytte validering af brugerinput i en web Applikation.*
  5. *Eleven kan implementere passende ViewModels eller DTO klasser.*
  6. *Eleven kan anvende Unit Test og mocking af objekter.*
  7. *Eleven kan konfigurere routing i en applikation.*
  8. ***Eleven kan udvide en applikation med en database, evt. med et ORM-framework.***
  9. *Eleven kan programmere services til brug for en applikation, f.eks. data- og logging-services. r*
  10. *Eleven kan benytte en hensigtsmæssig strategi for Exception handling.*
- 11. Eleven kan implementere sikkerhed og brugeradministration i en web applikation.**
12. *Eleven kan udrulle (deploy) en applikation, både On-Premises og Cloud baseret.*
  13. *Eleven kan udføre Parallel Programmering.*
- 14. Eleven kan redegøre for fordele/ulemper ved forskellige teknikker inden for Cryptography.**
15. *Eleven kan anvende Hashing, Symmetric og Asymmetric Encryption*

## Øvelse:

**Øvelsen er selvstændigt, hvor du via instruktionerne (under afsnit ”krav”) selv skal undersøg og afprøve dine løsninger. Brug gerne underviser som vejleder, også til afdækning af teori løbende efter jeres behov.**

### Baggrund:

HTTPS implementerer **asymetrisk kryptering** under dens ”handshake” proces når en brugers browser kontakter en server applikation, som ender med en **symetrisk kryptering** process hvor bruger og server deler brugerens premær nøgle (se afsnit HTTPS handshake process side 5).

Test for understøttelse af **HTTPS** kan implementeres for din serverside web app projekt skal ske med anvendelse af en **self-sign certifikat**. Alt programmering og konfigurering mod din self-sign certifikat i din web app vil når web app udgives, virke mod en godkendt CA. Dette skyldes at samme certifikat oprettelse mønstre gøre sig gældende for en self-sign certifikat som for en godkendt CA. (se afsnit self-sign certifikat vs godkendt CA, side 5).

Self-sign cert kan ses f.ex. som en local-db hvor du arbejder mod din local-db under test og udvikling og du ændre bare connection string fra din lokalt-db til produktion db når applikation udgives. Samme gælder for lokalt test certifikater (self-sign) som bare udskiftes med produktion certifikat ved udgivelse.

Når du bruger et self-signed certifikat, tester du i bund og grund, om din applikation kan håndtere HTTPS-forbindelser. Hvis din kode understøtter HTTPS korrekt med et self-signed certifikat, vil det samme setup fungere med et gyldigt certifikat fra en autoriseret udsteder (CA).

.NETs HttpClient, Kestrel, IIS, Docker og andre komponenter fungerer ens, uanset om certifikatet er self-signed eller fra en officiel CA. (se afsnit Erstatning af self-sign certifikat med et godkendt CA, side 5)

### Krav:

1. Sørge for, at din web app er configureret til at anvende HTTPS der altid auto navigeres til https url'en.
2. Med **dotnet CLI kommando**, opret en fysisk self-sign .pfx certifikat som **SKAL** placeres i din user folder (må også gerne være en .crt, .pem certifikat genereret af openssl).  
⚠️ Er dette ikke opfyldt, betragtes det som en væsenlig mangel.
3. Opret en folder: **Docker**, og lav en kopi af din web app fra forgående øvelse, og ”paste” det i folderen.

*Du har herved nu 2 kopier af din projekt, den originale som skal konfigureres til kørsel på en web server senere, og en i din Docker folder som skal konfigureres til at køre i en container.*

- Denne ”Docker kopi” af din web app skal du nu udgives til en **Docker Linux container**.  
*Det anbefales, at du afinstaller din nuværende Docker Desktop og download/Installerer det seneste version fra netet inden du prøver at udgive din web app i en Docker Linux container.*
  - Du skal ”deploy” din web app på Docker Linux container ved hjælp af **docker CLI** kommandoer (docker build/run) fra en command prompt (en cross-platform deployment, brug derfor **IKKE** Visual Studio).

- Hvis udgivelse af din web app på container med certifikat er en succes, vil du kunne navigere med browser til din web app med **https://.....**, prefix.
  - Opret en folder i din web projekt: "Docs" med en .txt fil: "readMe.txt" hvor du angiver, hvilket Docker CLI kommandoer du har brugt, til at udgive din web app på container'en.

Din underviser skal kunne anvende samme kommandoer fra din .txt fil til at udgive din web app projekt på sin Docker installation. Kommando'erne SKAL resulter i en successfuld udgivelse af din web projekt på din undervisers Docker installation, som så kan starte din web app op i sin container og ser, at den køre fejlfri på browseren med **https://.....**. **Er dette ikke opfyldt, betragtes det som en væsenlig mangel.**

- Upload denne kopi af din web app projekt til itslearning (eller fremvis til underviseren) til bedømmelse.
4. Konfigurer nu den original kopi af din web app til at køre på det indbygget cross-platform **Kestrel web server** som automatisk følger med i alle .NETs web projekt, netop til test kørsel af certifikater.
- Url til din certifikat fil og certifikat password (hvis du har opret en certifikat format som bruger password) skal gemmes i din web app, men de skal gemmes som en "secret" (hemlighed). **Er dette ikke opfyldt, betragtes det som en væsenlig mangel.**
  - Låst din certifikat til din cross-platform Kestrel web server, og test køre din web app på IIS. Din web app skal returner fejl når IIS køre den, med fejl som indiker, at din app ikke er konfigureret til kørsel på IIS. Men den skal kunne køre fejlfrit på Kestrel. **Er dette ikke opfyldt, betragtes det som en væsenlig mangel.**
    - Skolens PC kan lånes til test kørsel på IIS for dem som ikke bruger Windows PC, eller sammen i gruppe med en som bruger Windows PC til denne test.
  - Upload denne web app projekt til itslearning (eller fremvis til underviseren) til bedømmelse.

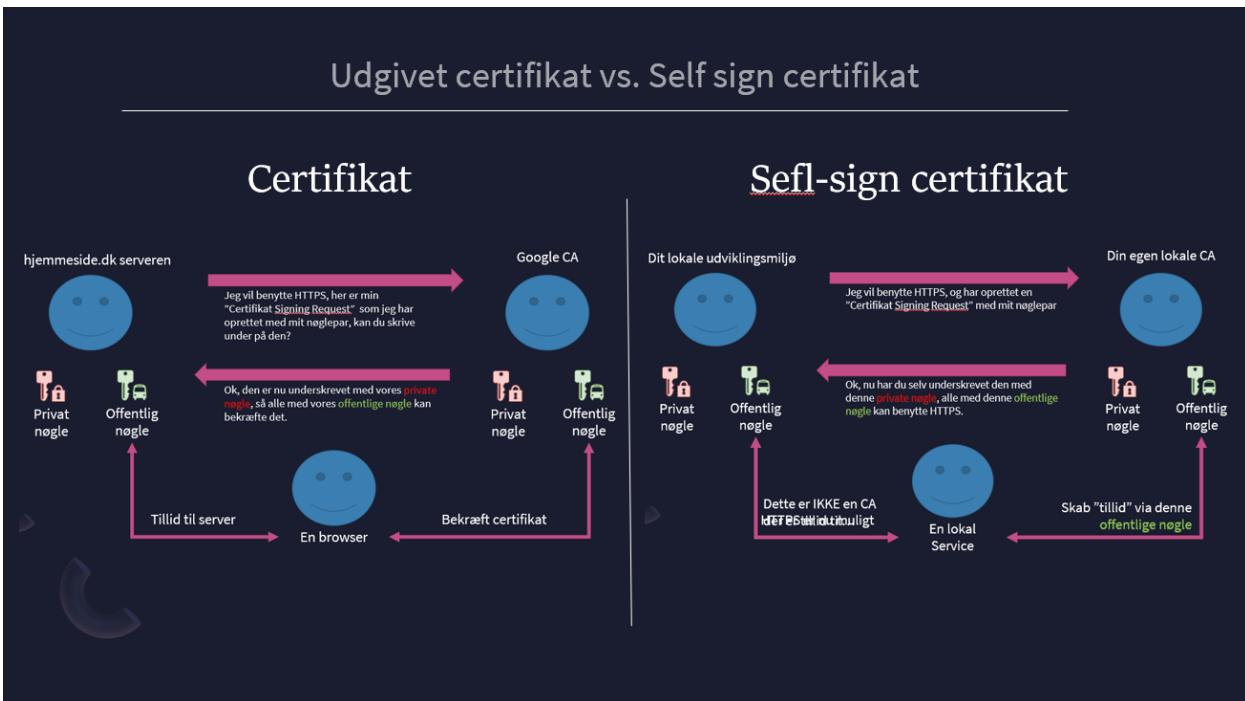
## Hvordan du bliver bedømmet

- Du skal kunne køre/fremvis din applikation fejlfrit på din browser med https:// prefix fra din container setup.
- Du skal kunne køre/fremvis din applikation fejlfrit på din browser med https:// prefix når den køre på det indbygget cross-platform Kestrel web server. Og den skal vise fejl når/hvis den køre på IIS, fordi din web apps certifikat skal være låst til Kestrel.
  - Du skal fremvis, at din certifikat url samt password er gemt som "secret" på din maskine.

# HTTPS handshake process



## Self-sign certifikat vs godkendt CA



# Erstatning af self-sign certifikat med en godkendt CA

Når du har oprettet et self-signed certifikat med f.ex. .pfx-filen ved hjælp af kommandoen:

```
dotnet dev-certs https -ep %USERPROFILE%\aspnet\https\CertifikateNavn.pfx -p MyPassword
```

og ønsker at udskifte det med et certifikat fra en godkendt CA, hvor adgangskoden findes i en fil, skal du gøre følgende:

## 1. Erstat self-signed certifikatet med et CA-udstedt certifikat

Du kan få et CA-udstedt certifikat i .pfx-format (som indeholder både den private nøgle og det offentlige certifikat). Dette gøres typisk via en **Certificate Signing Request (CSR)** og en betroet CA.

Hvis du allerede har et nyt CA-udstedt certifikat, kan du kopiere det til den samme placering, hvor det gamle self-signed certifikat lå:

```
copy MyNewCert.pfx %USERPROFILE%\aspnet\https\CertifikateNavn.pfx
```

## 2. Håndtering af adgangskode fra en fil

Hvis adgangskoden til .pfx-certifikatet er gemt i en fil (password.txt), kan du hente den og bruge den i din applikation eller importproces. I en PowerShell-terminal kan du f.eks. gøre følgende:

```
$certPassword = Get-Content "password.txt" | ConvertTo-SecureString -AsPlainText -Force  
Import-PfxCertificate -FilePath "$env:USERPROFILE\aspnet\https\CertifikateNavn.pfx" -Password $certPassword -CertStoreLocation Cert:\LocalMachine\My
```

Hvis du vil bruge certifikatet med **Kestrel i en .NET-applikation**, kan du læse adgangskoden fra filen og indlæse certifikatet i appsettings.json eller kode:

### Brug af adgangskode fra fil i C#

```
var certPassword = File.ReadAllText("password.txt").Trim();  
var cert = new X509Certificate2("CertifikateNavn.pfx", certPassword);
```

### Konfiguration i appsettings.json

```
"Kestrel": {  
    "Certificates": {  
        "Default": {  
            "Path": "%USERPROFILE%\\aspnet\\https\\CertifikateNavn.pfx",  
            "Password": "file:password.txt"  
        }  
    }  
}
```

## 3. Konfiguration af .NET til at bruge det nye certifikat

Hvis du bruger Kestrel-serveren i ASP.NET Core, kan du sikre, at den peger på det rigtige certifikat ved at tilføje følgende i Program.cs:

```
var builder = WebApplication.CreateBuilder(args);  
  
var certPassword = File.ReadAllText("password.txt").Trim();  
var cert = new X509Certificate2(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.UserProfile),  
    @"aspnet\https\CertifikateNavn.pfx"), certPassword);  
  
builder.WebHost.ConfigureKestrel(options =>  
{  
    options.ListenAnyIP(5001, listenOptions =>  
    {  
        listenOptions.UseHttps(cert);  
    });  
});  
  
var app = builder.Build();  
app.UseHttpsRedirection();  
app.Run();
```

## 4. Test at det nye certifikat fungerer

Start din .NET-applikation og test, at HTTPS fungerer korrekt uden advarsler.