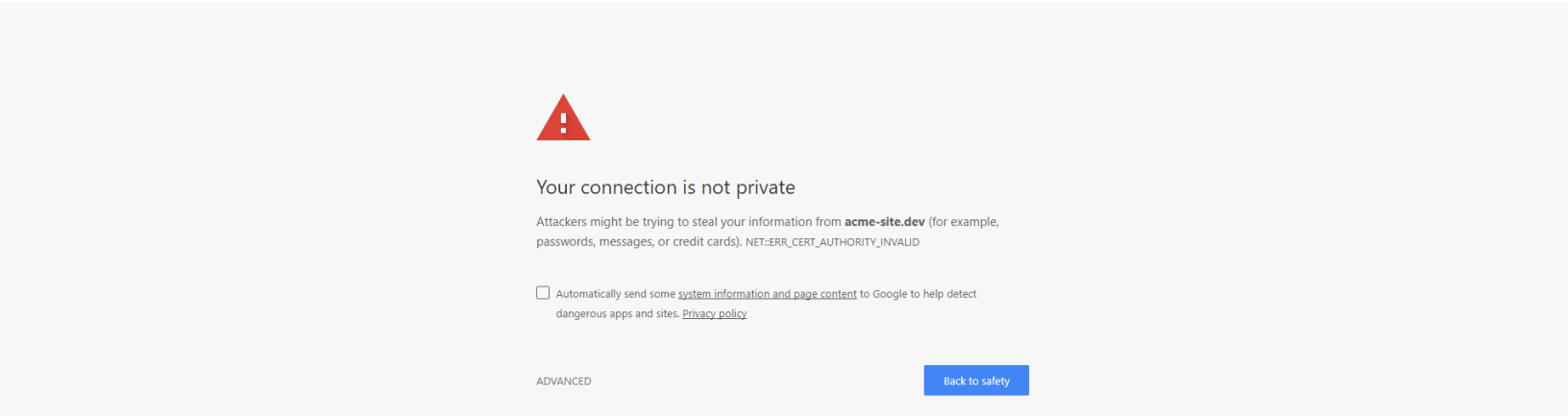
Thijs Busser

Thijs Busser Jun 23, 2017 · 11 min read



Chrome's error message when connecting to a site with an improper SSL certificate.

tl;dr a self signed SSL certificate to use for website development needs a root certificate and has to be an X.509 version 3 certificate. Creating one takes about 5 terminal commands, see at the bottom for the relevant commands.

Why another article on how to create a self signed certificate?

I went to the API special of CSSDay 2017 and got excited by all the new browser APIs demonstrated on stage. Once I got the time to sit on my laptop and experiment for myself, I got confronted with a hurdle: the new APIs I wanted to try require an HTTPS connection which I didn't have.

The production version of the website I wanted to experiment on has been available over HTTPS for a long time. The development environment never got an SSL certificate. I figured I would create a self signed certificate as it is not overly complicated. Soon I discovered creating a self signed certificate the browser trusts is a little more tricky.

There are various tutorials and examples on how to create such a certificate but it took several sources to pull everything together before I had a fully working setup. This is why I wrote it all down in a single place, perhaps it will help out others in a similar situation.

The goal

Before I go on it might be good to know a bit about the environment I am working in. The website I am developing is hosted on Microsoft IIS. My main development work I do in OSX. For development the website runs inside a Virtual Machine on my Mac.

Furthermore the website uses two separate domains. There is *acme-site.dev* which hosts the website and *acme-static.dev* which functions as a static domain for stylesheets, scripts, and images.

My goal is to create an SSL certificate that:

- I can use with IIS;
- Will be accepted by the browser without any warnings or errors**.**

Figuring out what I need

When I started I figured it was going to take separate SSL certificates to get the entire setup working. I knew a wildcard certificate would not serve my needs as I have two separate domains. I was going for one certificate for the website domain and one for the static domain.

I followed some tutorial on generating a self signed SSL certificate and not before long I had a certificate for acme-site.dev. Full of enthusiasm I loaded the site in Chrome only to be met with an error page that the connection was not secure and the error code **ERR_CERT_COMMON_NAME_INVALID**.

The error struck me as strange as I was sure I properly set the common name for my certificate to acme-site.dev. The DevTools Security tab was little extra help though it did mention something about my certificate missing something called Subject Alternative Names (SAN).

Ultimately this helped me find what it is I need. Though I didn't know the technical name at the time, I needed an **X.509 v3** certificate with SAN. The additional benefit of SAN is that I need only a single certificate for both of my domains, a nice extra bonus.

While writing this article I found out the cause of Chrome displaying the error message I got. Starting with Chrome v58 they no longer accept certificates without SAN information. If only the error page Chrome display would've been a little bit more informative it would've saved me quite some time.

Creating an X.509 v3 certificate

Okay, now that I finally know what I need, it is time to get to work. While reading tutorials on how to generate my self signed SSL certificate it soon became clear creating just an SSL certificate won't do. It has to do with the SSL certificate chain. Basically it needs to be issued by a party the browser knows it can trust so it knows it can trust your SSL certificate.

Creating a root certificate can be done in OSX, in the terminal. For this purpose you can use a tool called openssl. It was already on my machine, I probably needed it in the past for something, but YMMV.

Creating the root certificate

It takes two terminal commands to generate a root certificate. The first command is to create a private key. This can be accomplished by running the following command:

```
openssl genrsa -des3 -out rootCA.key 2048
```

This creates a key, 2048 bits long, The **-des3** parameter specifies to use the Tripple DES algorithm to encrypt the key and will require you to enter a password in order for the key file to be created. Be sure to remember the password you enter or you will have to generate a new key.

Now to generate the root certificate:

```
openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024  -
out rootCA.pem
```

I won't pretend to know exactly what all the parameters do, but in short I figure it does the following:

- -new: create a new request

- -nodes: don't encrypt the output key
- -x509: specifies the kind of certificate to make
- -key: the file with the private key to use
- -sha256: this is the hashing algorithm. When you omit this it will default to the SHA1 algorithm which will result in the browser generating a warning
- -days: the number of days the certificate should be valid for. Use as high a number as you feel comfortable with for your development environment
- -out: the name of the file to write the certificate to.

When you run the command you will be asked to provide some information. This will be included in the certificate and is public information. I used to the following to create the certificate:

```
> openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024
-out rootCA.pem

You are about to be asked to enter information that will be
incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name
or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:NL

State or Province Name (full name) [Some-State]:.

Locality Name (eg, city) []:ACME City

Organization Name (eg, company) [Internet Widgits Pty Ltd]:ACME
Websites

Organizational Unit Name (eg, section) []:ACME IT

Common Name (e.g. server FQDN or YOUR name) []:ACME ROOT CA
```

```
Email Address []:webmaster@acme.dev
```

Creating the SSL certificate

With the root certificate covered it is time to generate the SSL certificate. Because only an X.509 v3 certificate carries SAN information it requires a little more work than when creating an X.509 v1 certificate.

The extra work is the additional file needed with the v3 information. I've created a file named *v3.ext* and put the following information in there:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment,
dataEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = acme-site.dev
DNS.2 = acme-static.dev
```

It is the **alt_names** section which deserves most of the attention. This is where the domains can be specified for which the certificate is valid. I only need these two domains but you could add an extra entry for something like *www.acme-site.dev* or **.acme-site.dev.*

With this file created and stored in the same folder as the root CA key and certificate it is time to head back to the terminal. The first step is to create a private key for the SSL certificate and a certificate signing request. These two tasks can be combined into a single command:

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

It is quite similar to the command used to create the root certificate. The **-newkey** and **-keyout** are to specify the kind of private key to generate, and the file to store it in.

This command also triggers a set of questions. I used the following information to create my certificate:

```
You are about to be asked to enter information that will be
incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name
or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:NL

State or Province Name (full name) [Some-State]:.

Locality Name (eg, city) []:ACME City

Organization Name (eg, company) [Internet Widgits Pty Ltd]:ACME
Websites

Organizational Unit Name (eg, section) []:ACME IT

Common Name (e.g. server FQDN or YOUR name) []:ACME DEV CERTIFICATE

Email Address []:webmaster@acme.dev

Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:ACME

An optional company name []:.
```

Now that a private key and certificate signing request have been created it is possible to issue the certificate with the previously generated root certificate.

```
openssl x509 -req -in server.csr -CA rootCA.pem -CAkey rootCA.key -
CAcreateserial -out server.crt -days 500 -sha256 -extfile v3.ext
```

Preparing the certificate for IIS

This is the part I understand the least but it seems IIS needs the SSL certificate along with the private key in order to be able to use the certificate. Right now I've created a *server.key* and a *server.crt* file and these need to be combined into a single file. This can be accomplished with the following terminal command:

```
openssl pkcs12 -inkey server.key -in server.crt -export -out
server.pfx
```

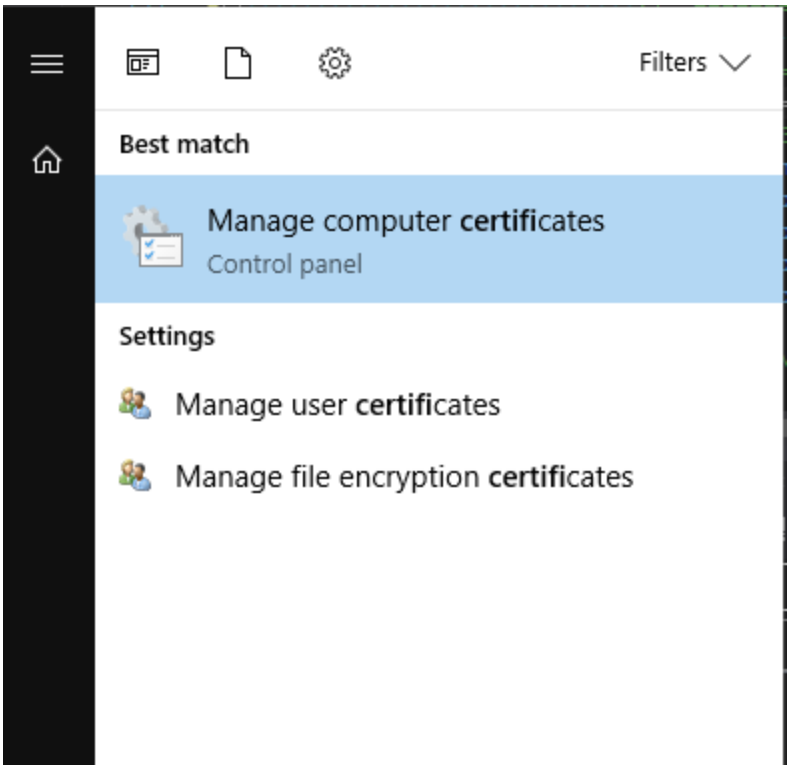
When the command is executed it will ask for an export password, this will be needed again when importing the resulting *server.pfx* into the windows certificate store.

With this command executed all the keys and certificates to get a fully functioning SSL certificate are generated. All that is left to do is importing the certificates and configuring IIS.

Configuring the Windows certificate store

In order to be able to use the certificate for the website, the certificates need to be imported into the Windows certificate store. My virtual machine runs Windows 10, it may work a little different on other versions.

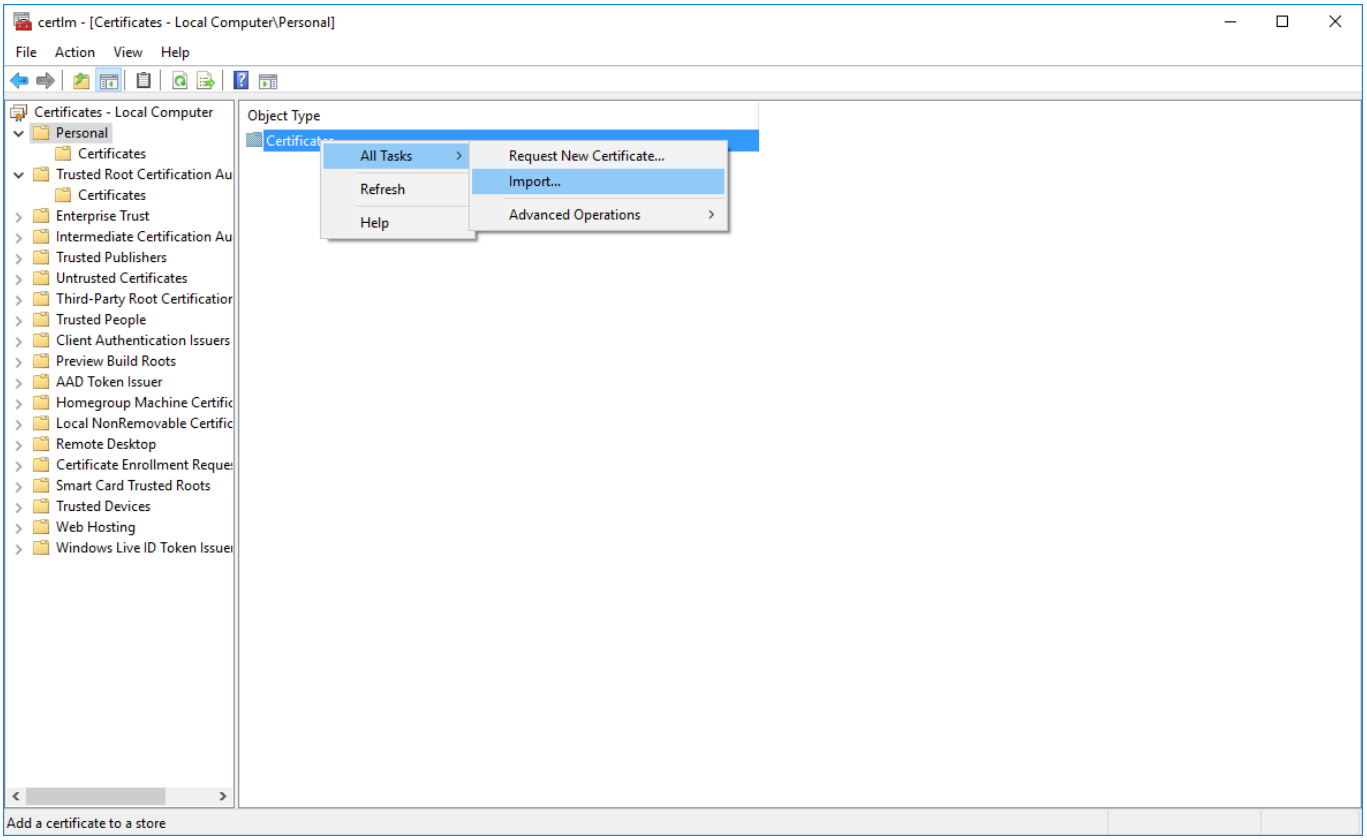
When you open the start menu in Windows 10 and you type "certificates", Windows comes up with two relevant suggestions: "Manage computer certificates" and "Manage user certificates". Both will be needed to install the SSL certificate.



Windows 10 start menu showing the suggestions for “certifi”

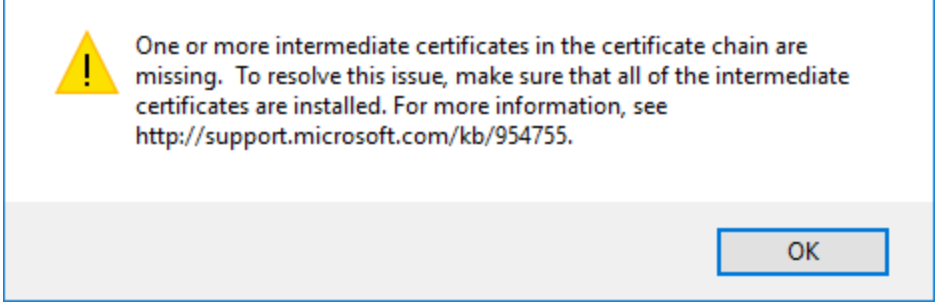
Computer certificates

The window for managing the computer certificates looks something like this:



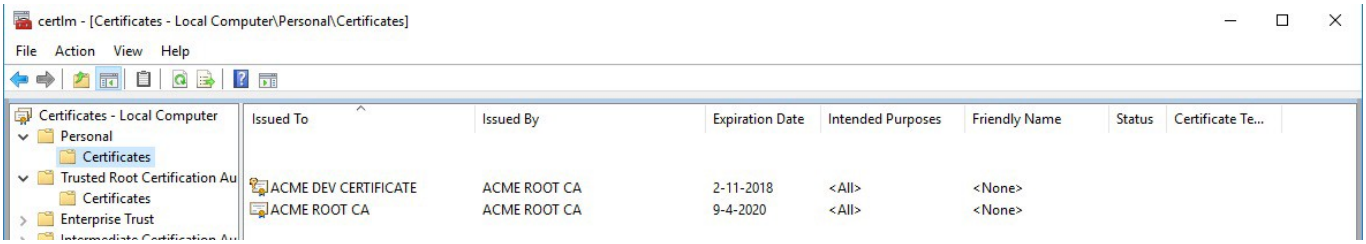
Windows 10 application for managing computer certificates with the context menu for Personal certificates opened.

When the context menu for *Personal* is accessed there is an option *Import…* under *All Tasks*. Selecting this item will start a wizard to select and import a certificate. In this certificate store both the *rootCA.pem* and *server.pfx* certificate need to be imported. By importing *server.pfx* the SSL certificate becomes selectable in IIS, importing *rootCA.pem* will stop IIS from generating warnings the certificate chain is not complete.



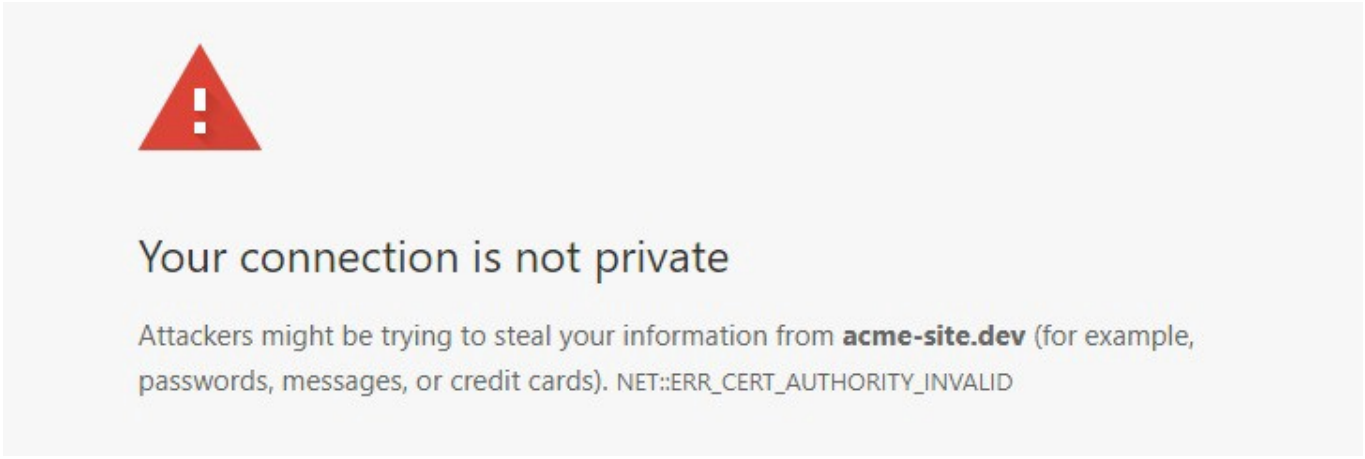
IIS warning message when the root certificate hasn't been added to credentials store.

With both certificates installed they will be listed in the application. More importantly, it is now possible to select them in IIS when creating an HTTPS binding and not get any warning messages from IIS.



Both the root certificate and SSL certificate imported in the computer credentials store.

When there is an HTTPS binding and you would try to visit *https://acme-site.dev* using Chrome in Windows, you would still see an warning page instead of the website itself. This is because Windows still needs to be told it can trust certificates signed with the self created root certificate.

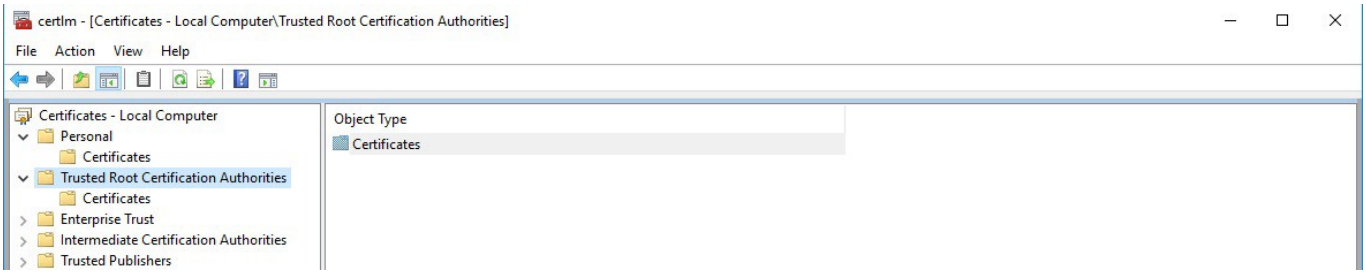


Chrome's error page for a certificate from an issuer unknown to Windows.

Personal certificates

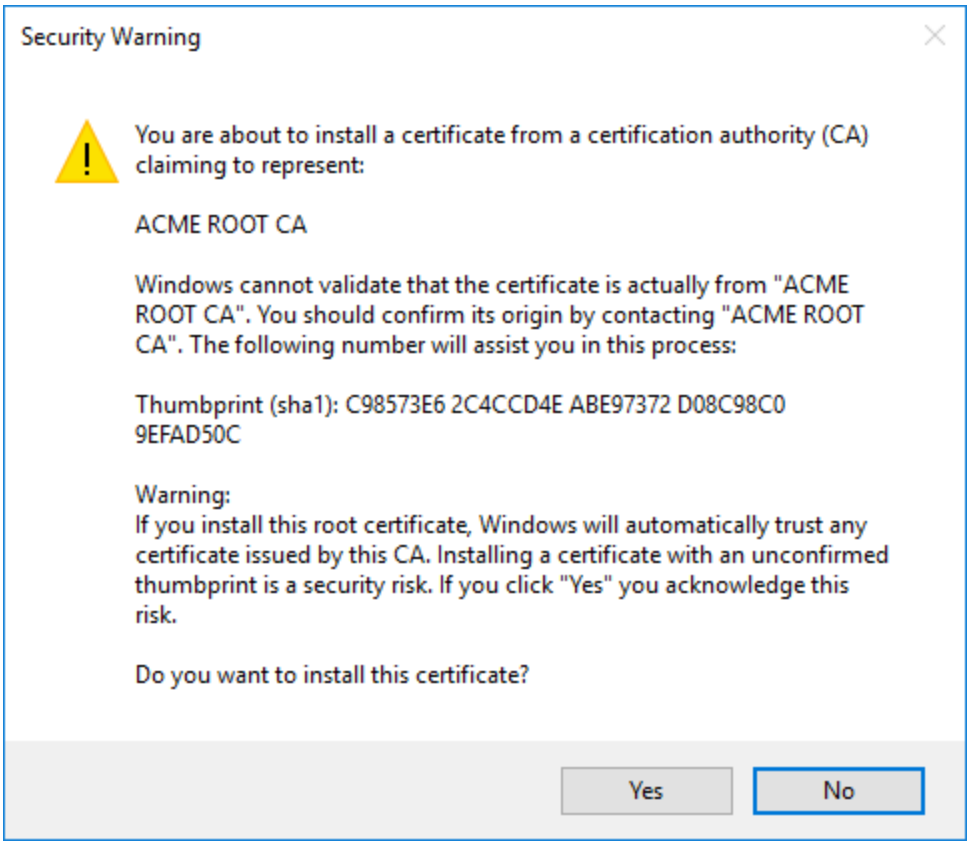
In order to inform Windows it can trust certificates issued with the self created root certificate, the root certificate should be imported under personal certificates. This application looks the same as the one for managing the computer certificates. The big

difference is the location where the root certificate should be imported into: *Trusted Root Certification Authorities*.



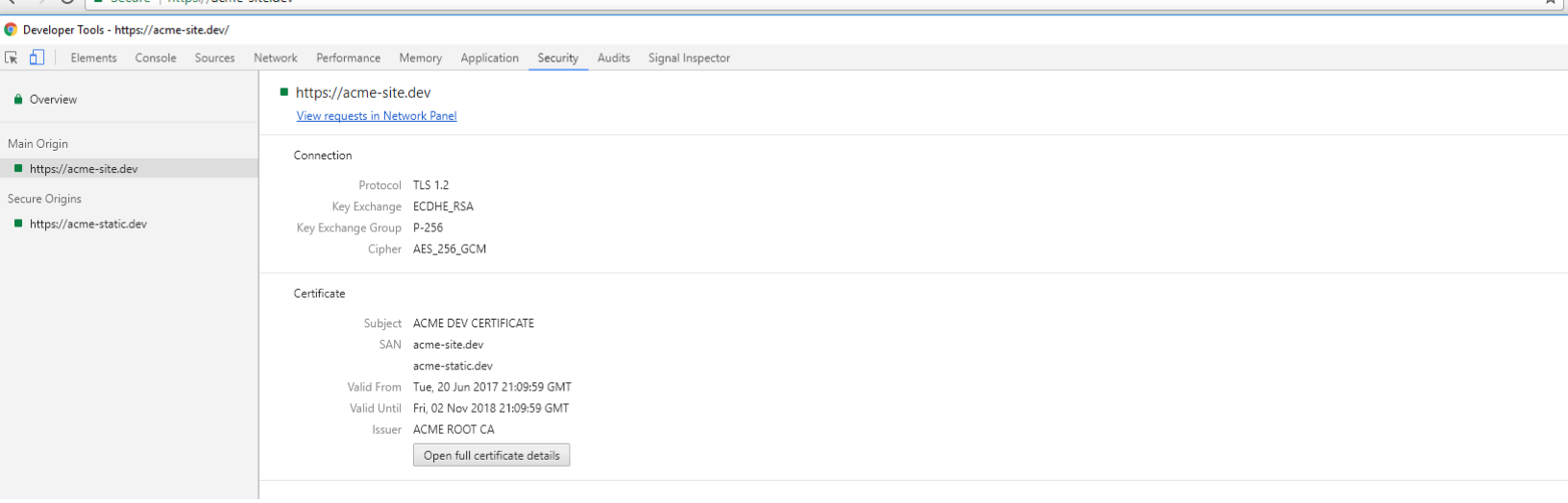
The Trusted Root Certification Authories item in the user credentials store

Importing the *rootCA.pem* certificate in this location will be met with a warning message. It informs that accepting an CA certificate from an unknown origin is dangerous and to make sure the certificate is actually legit.



The confirmation message displayed by Windows when adding an unknown root certifcate.

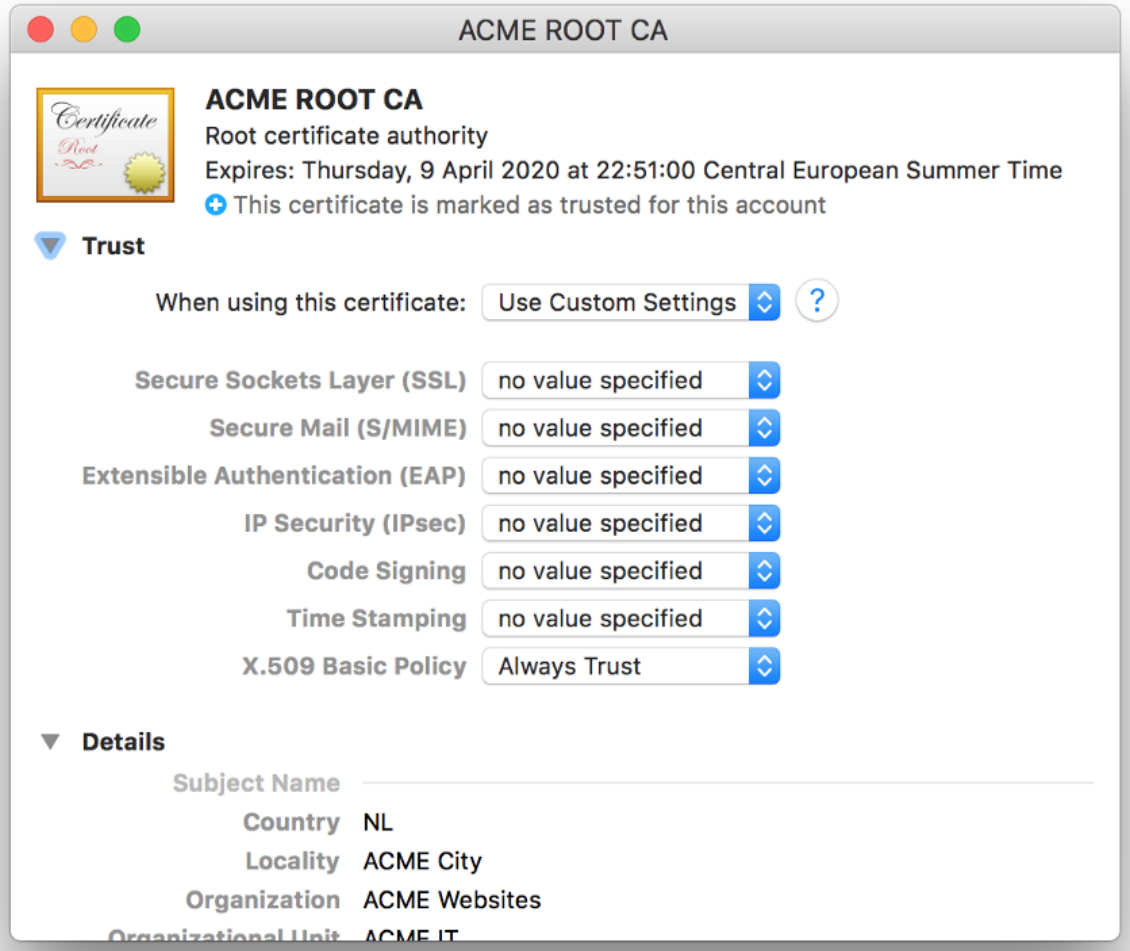
Since the certificate being added to the certificate store is the self signed certificate this dialog can safely be answered with *Yes*. With the root certificate added to the list of trusted root certification authorities all the steps are done. Opening https://acme-site.dev will no longer display any warnings, instead Chrome will display a nice “secure” status in the URL bar.



The final result, a secure website loading secure assets from a different domain. All using just a single certificate

Final Steps

On Windows the site is now accessible under HTTPS, the same is not true for OSX. This is because OSX doesn't yet know it can trust certificates signed with the self created root certificate. To accomplish this takes an action very similar to getting Windows to accept the certificate, the root certificate needs to be added to the keychain.



The Keychain Access dialog for specifying for which tasks the custom root certificate should be trusted.

To add the root certificate to the keychain open *Keychain Access* in OSX and drop the *rootCA.pem* in it from *Finder*. This will add the certificate to the store but is not yet enough to trust the SSL certificate. In order to trust the SSL certificate it is needed to tell OSX the root certificate is trusted for performing X.509 Basic Policy tasks. This dialog can be accessed by double clicking on the certificate in Keychain Access.

FireFox

Using the certificate in FireFox is a little different. FireFox doesn't use the operating system's credentials store but instead has its own managing interface. Google can help to find a document describing how to do this or try opening the site in FireFox and add the certificate through the warning page it will display.

The certificate will have to be added per domain. Just adding the exception for *acme-site.dev* will not automatically add the exception for *acme-static.dev*. This will have to be done manually by opening a valid URL for *acme-static.dev* and adding the exception.

Command Recap

The following commands are needed to create a root certificate:

```
openssl genrsa -des3 -out rootCA.key 2048

openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024  -
out rootCA.pem
```

The following commands are needed to create an SSL certificate issued by the self created root certificate:

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

openssl x509 -req -in server.csr -CA rootCA.pem -CAkey rootCA.key -
CAcreateserial -out server.crt -days 500 -sha256 -extfile v3.ext
```

The referenced *v3.ext* file should look something like this:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment,
dataEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = acme-site.dev
DNS.2 = acme-static.dev
```

In order to bundle the server certificate and private key into a single file the following command needs to be executed:

```
openssl pkcs12 -inkey server.key -in server.crt -export -out
server.pfx
```

Ssl Openssl Self Signed X509 Ssl Certificate