

AcuToWeb 10.2 & IIS

1. Contents

































1. Contents.....	1
2. Introduction	3
3. Pre Requisites.....	4
4. IIS Setup for AcuToWeb	5
4.1. Enabling WebSocket support on IIS.....	5
4.2. Download and install Application Request Routing (ARR 3.0).....	7
4.3. Download and install URL Rewrite.....	8
5. AcuToWeb and IIS on the same platform	9
5.1. Application Routing Cache (ARR) Setup.....	9
5.2. URL Rewrite Setup	10
5.3. Firewall setting.....	14
5.4. Tests	15
6. AcuToWeb and IIS on different platform.....	16
7. AcuToWeb and IIS on the same platform with SSL.....	17
7.1. Introduction	17
7.2. First test : With Self signed certificate generated by IIS	18
7.2.1. Introduction	18
7.2.2. URL Rewrite setup & AcuToWeb setup	18
7.2.3. Create a self-signed certificate with IIS.....	19
7.2.4. Setup HTTPS under IIS.....	20
7.2.5. SSL Settings	22
7.2.6. SSL Settings	24
7.3. Second Test : Create a certificate Request through IIS and use a free CA Web Site to sign this request	24
7.3.1. Create a certificate request with IIS.....	24
7.3.2. Signing the certificate request	27
7.3.3. Generate the Server Certificate	28
7.3.4. HTTPS setup	29
7.3.5. Importing the getacert CA certificate in the Browser.....	29
7.3.6. Tests	31
7.4. Third Test : Create a CA certificate and a server certificate with openssl commands	32

7.4.1.	Introduction	32
7.4.2.	Create the openssl configuration file use to create a CA Certificate and a server certificate	32
7.4.3.	Create the CA Certificate	33
7.4.4.	Import the CA certificate in your brower.....	35
7.4.5.	Create Server certificate signed by CA.....	36
7.4.6.	Prepare the deployment of the server certificate in IIS	37
7.4.7.	Deploy Server certificate in IIS	37
7.4.8.	Tests	38
8.	Conclusion.....	39
9.	Annexes.....	39
8.1.	openssl configuration file : ws2016acutoweb.cnf	39
8.2.	Script create_ca_certificate.sh to create a CA Root Certificate.....	45
8.1.	Script create_server_certificate.sh to create the server cerficate	47

2. Introduction

This document gives the details of various tests made with AcuToWeb 10.2 and IIS.

➔ Extend 10.2 documentation about the setup of IIS needed for AcuToWeb.

- [-]  AcuToWeb Version 10.2.0 User's Guide
 -  Copyright Statement
 -  AcuToWeb Introduction
- [-]  AcuToWeb Installation and Licensing
 -  AcuToWeb Gateway to AcuConnect prerequisites
 -  AcuToWeb install directories and files
 -  Licensing
 -  Preparing Your Application
 -  Getting Started (Windows)
- [-]  AcuToWeb Control Panel
 - +  Access Tab
 - +  Config Tab
 - +  Alias Tab
 - +  Info Tab
 - +  Services Tab
 - +  Gateway Services Tab
- [-]  AcuToWeb Gateway
 -  Example Gateway Configuration File
 - [-]  Working with IIS or Apache
 - +  Configuring Apache
 -  **Configuring IIS**
 -  Configuring SSL Authentication
 - [-]  Connecting to the Gateway
 - [-]  Connection URL Syntax Explained
 -  Including Special Characters
 - +  Cascading Style Sheets
 -  Using AcuToWeb on a Mobile Device
 - +  AcuToWeb Desktop
 -  AcuToWeb Security and Authentication
- [-]  Troubleshooting Your Application
 -  Using the Logging Console
 -  Limitations and Known Issues

In this documentation, you will find also information to setup https with IIS and to generate CA Certificate and Server Certificate in order to test SSL.

3. Pre Requisites

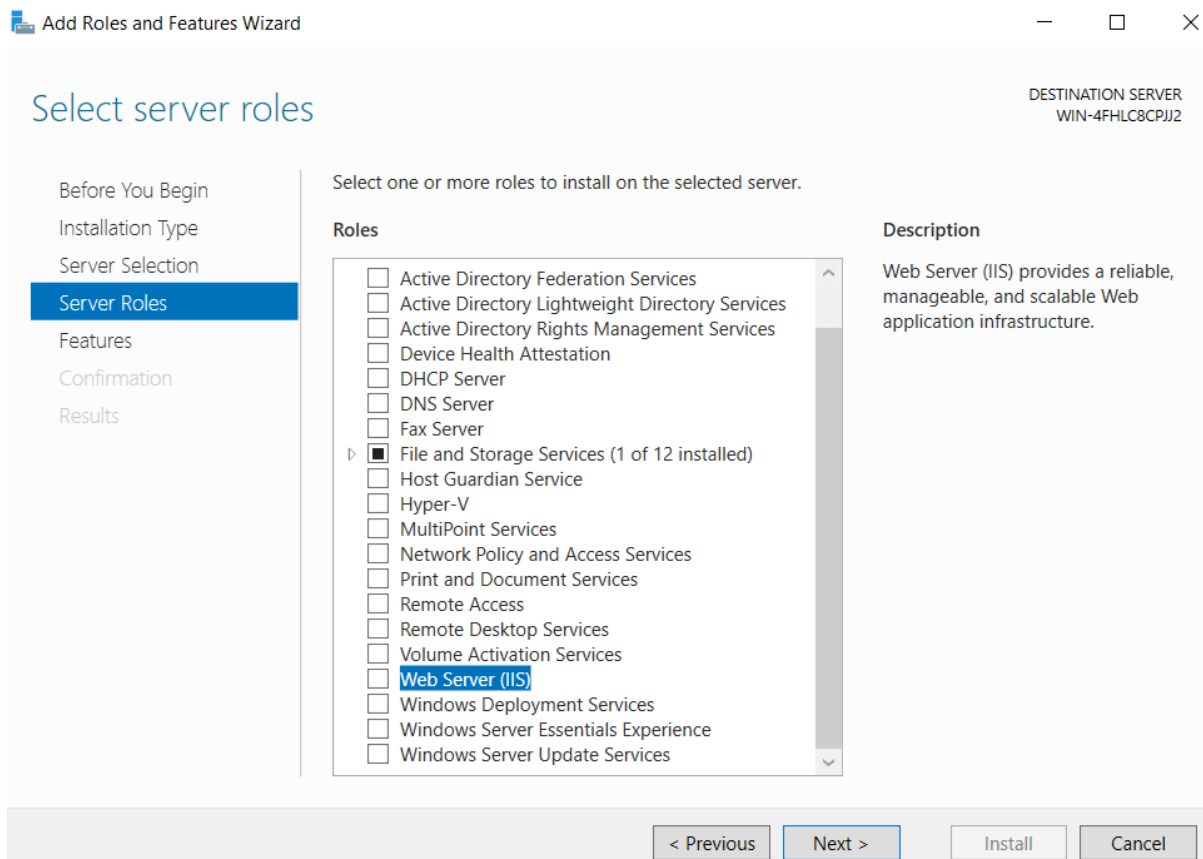
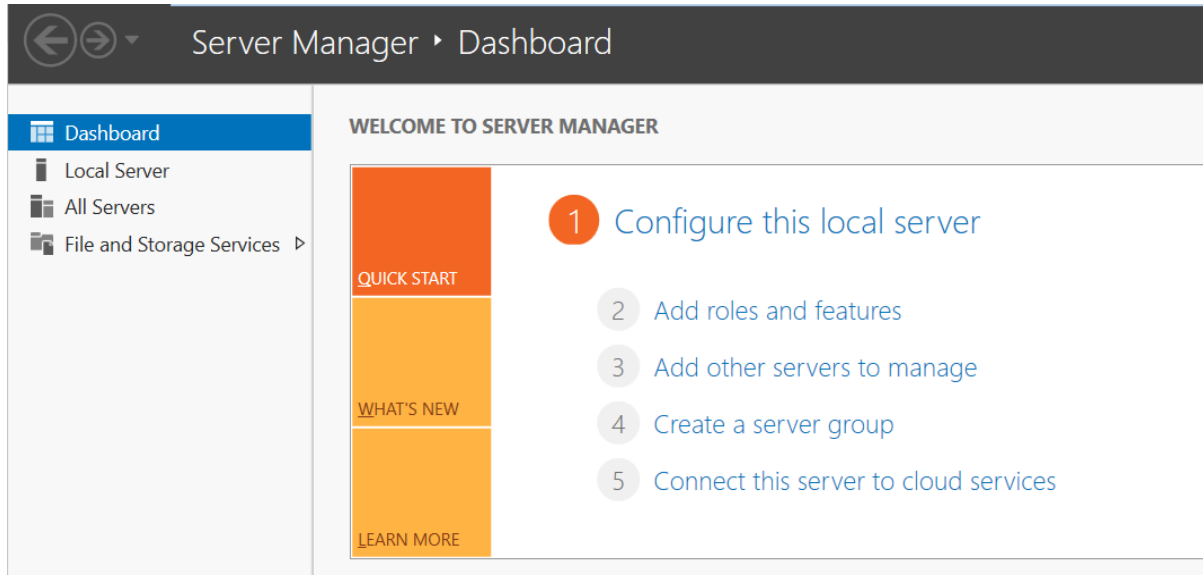
Note: You must use IIS version 8 or later, as this provides WebSocket support; IIS version 8 is supported on Windows Server 2012 and later. For more information, visit

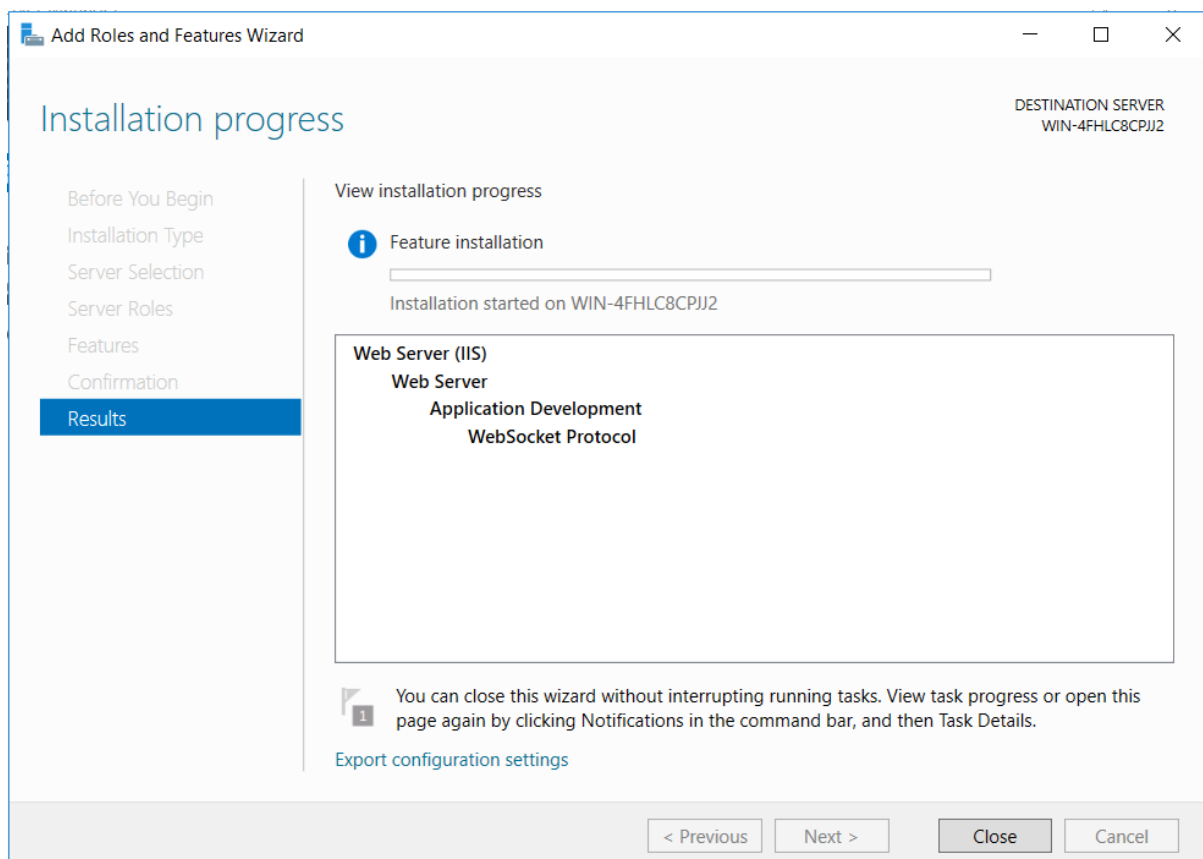
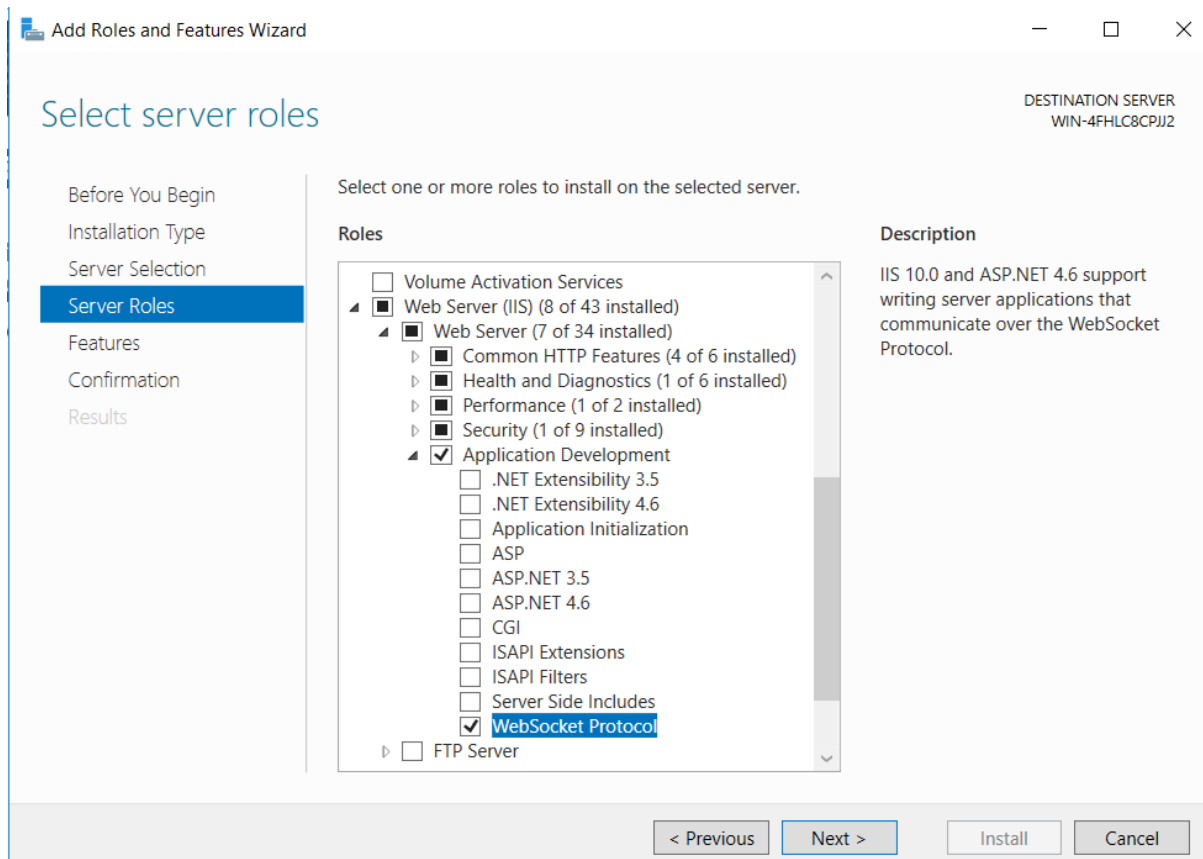
<http://www.iis.net/learn/get-started/whats-new-in-iis-8/iis-80-websocket-protocol-support>.

4. IIS Setup for AcuToWeb

4.1. Enabling WebSocket support on IIS

Enabling **WebSocket support** on IIS was done below on a Windows Server 2016 platform.





4.2. Download and install Application Request Routing (ARR 3.0)

Using IIS ARR extension with IIS URL Rewrite extension allows to route requests to AcuToWeb Gateway.

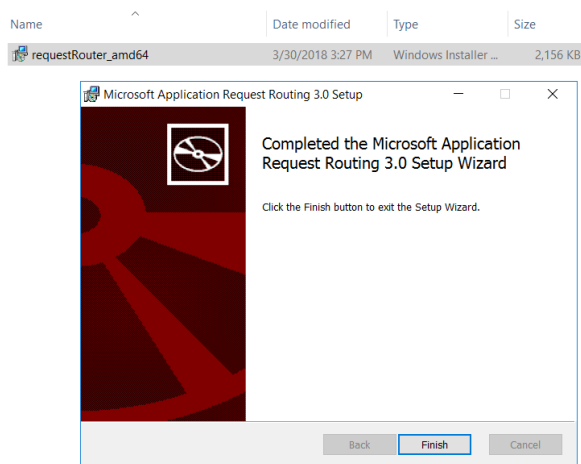
Application Request Routing (ARR) is an extension to [Internet Information Server \(IIS\)](#), which enables an IIS server to function as a load balancer. With ARR, an IIS server can be configured to route incoming requests to one of multiple web servers using one of several routing algorithms. By load balancing requests, high availability of web servers can be achieved without incurring the typically-high costs of dedicated load balancing products.^[1]

ARR is currently available in version 3.0, released on July 26, 2013. The current version is supported in [x86](#) and [x64](#), and can be installed on IIS 7.0 or later (Windows 2008 or later). ARR is available as a download from Microsoft's download center, or via Microsoft's Web Platform Installer (WebPI).

ARR requires the URL Rewrite extension to function, and uses it for routing requests. ARR can be configured to redirect traffic based on server variables, URLs, cookies and more, and performs full layer 7 load balancing. ARR's functionality can be described as a [load balancing](#) and [reverse proxy](#), although it is not as advanced as some dedicated reverse proxy products such as Microsoft [UAG](#) and dedicated load balancing solutions.

Below the URL to downloading ARR

<http://www.iis.net/downloads/microsoft/application-request-routing>

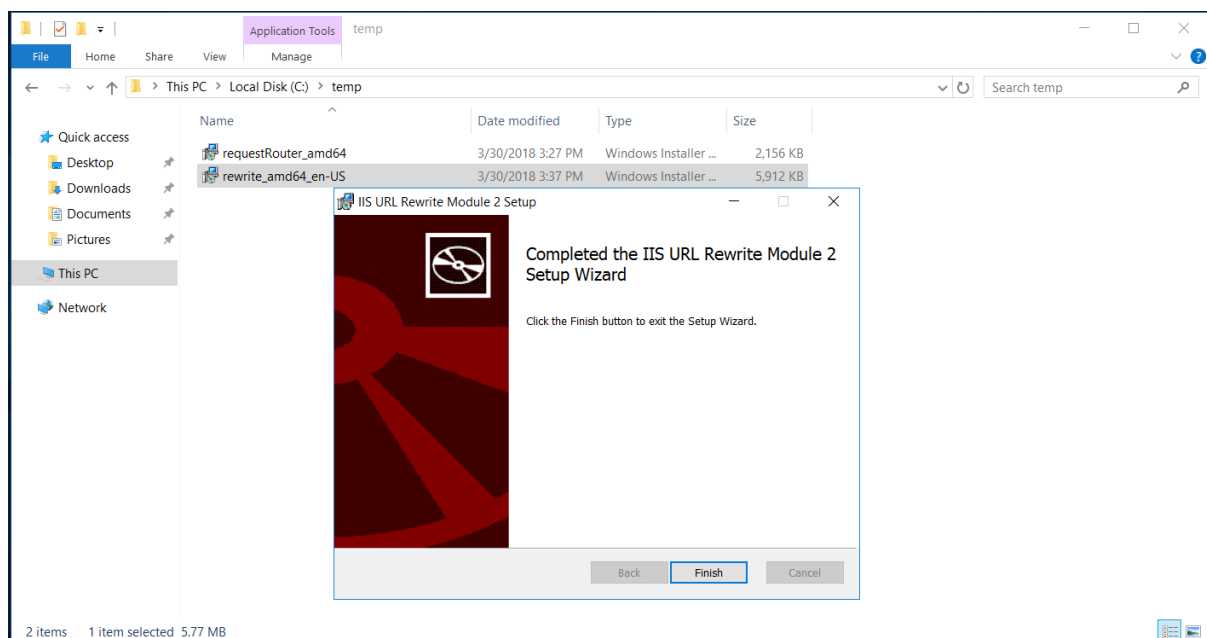


4.3. Download and install URL Rewrite

Using **IIS ARR extension** with **IIS URL Rewrite extension** allows to route requests to **AcuToWeb Gateway**.

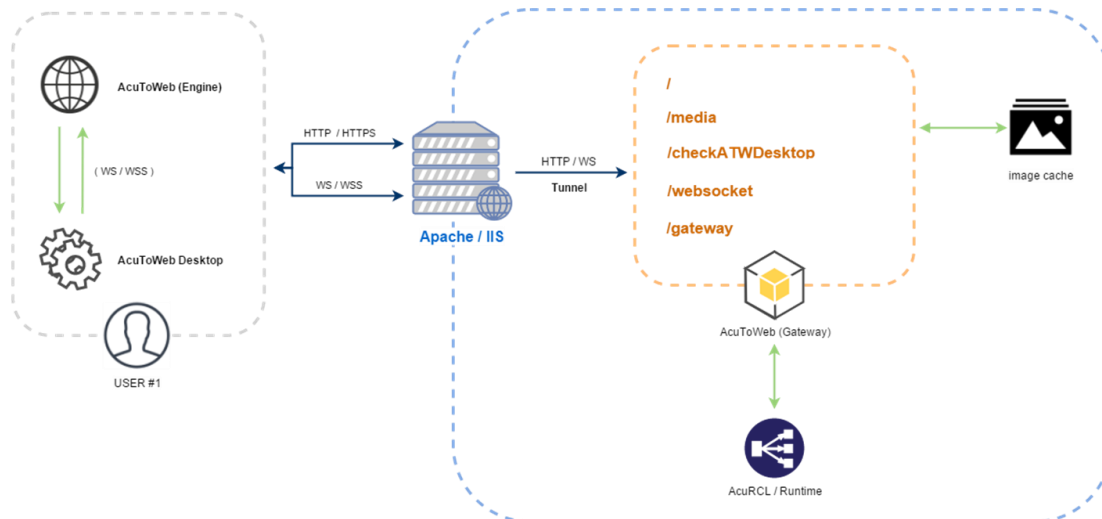
<http://www.iis.net/downloads/microsoft/url-rewrite>

IIS URL Rewrite 2.1 enables Web administrators to create powerful rules to implement URLs that are easier for users to remember and easier for search engines to find. By using rule templates, rewrite maps, .NET providers, and other functionality integrated into IIS Manager, Web administrators can easily set up rules to define URL rewriting behavior based on HTTP headers, HTTP response or request headers, IIS server variables, and even complex programmatic rules. In addition, Web administrators can perform redirects, send custom responses, or stop HTTP requests based on the logic expressed in the rewrite rules.



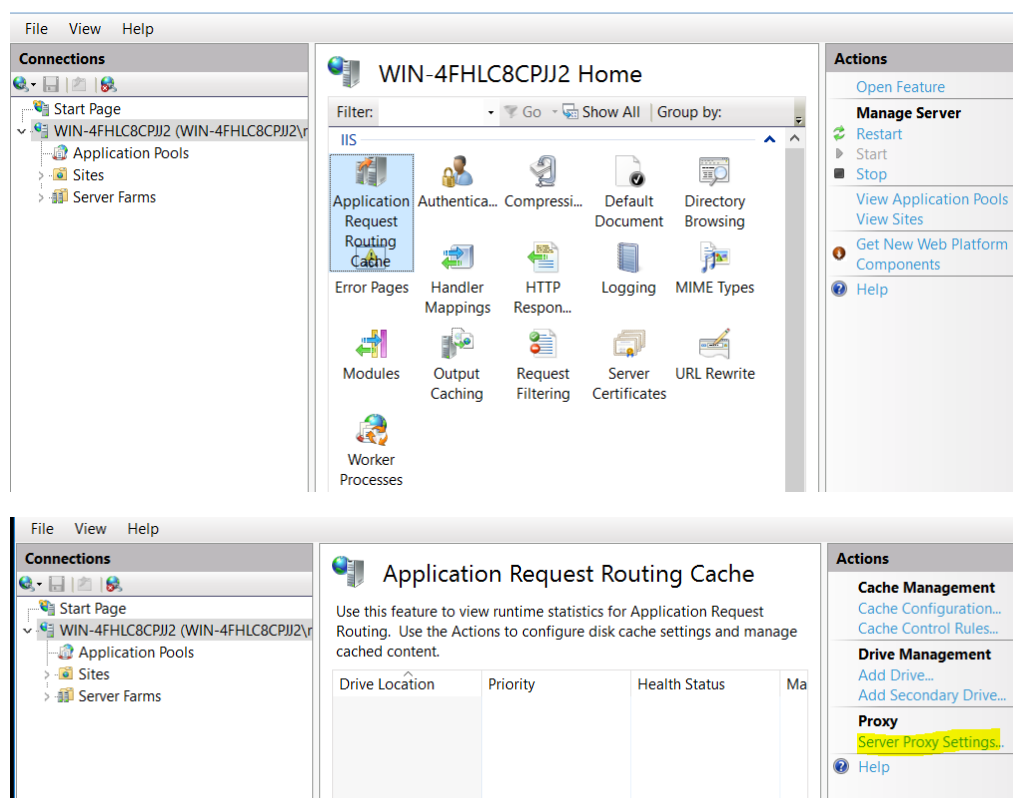
5. AcuToWeb and IIS on the same platform

Architectures avec Serveur Web Standard (Apache/IIS)

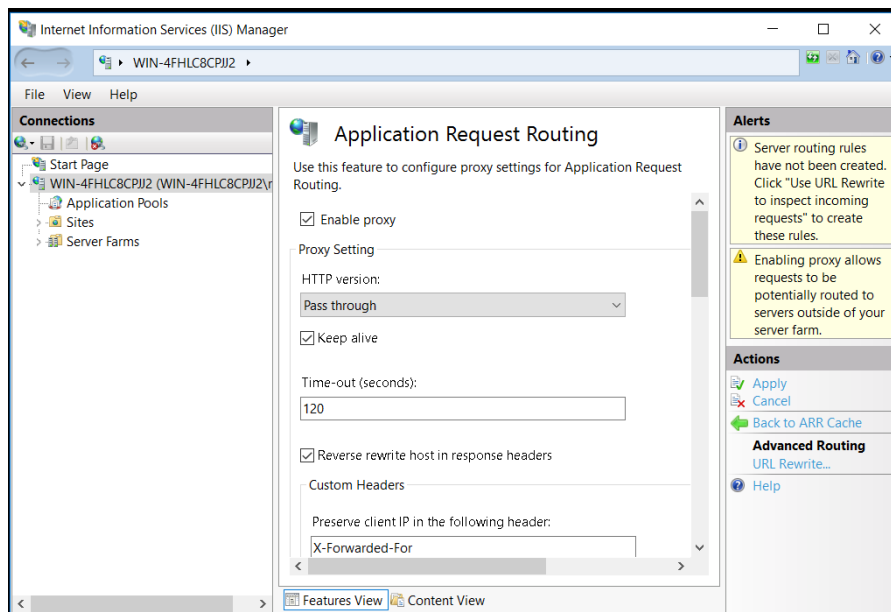


5.1. Application Routing Cache (ARR) Setup

Enabling proxy allows requests to be potentially routed to servers outside of your server farm. Servers routing rules have to be created with URL Rewrite.



Application Request Routing → Select **Enable Proxy**, then click Apply.



5.2. URL Rewrite Setup

Three rules will be created.

When all three rules are properly configured, the IIS-hosted site will be able to route all the communications towards the AcuToWeb Gateway.

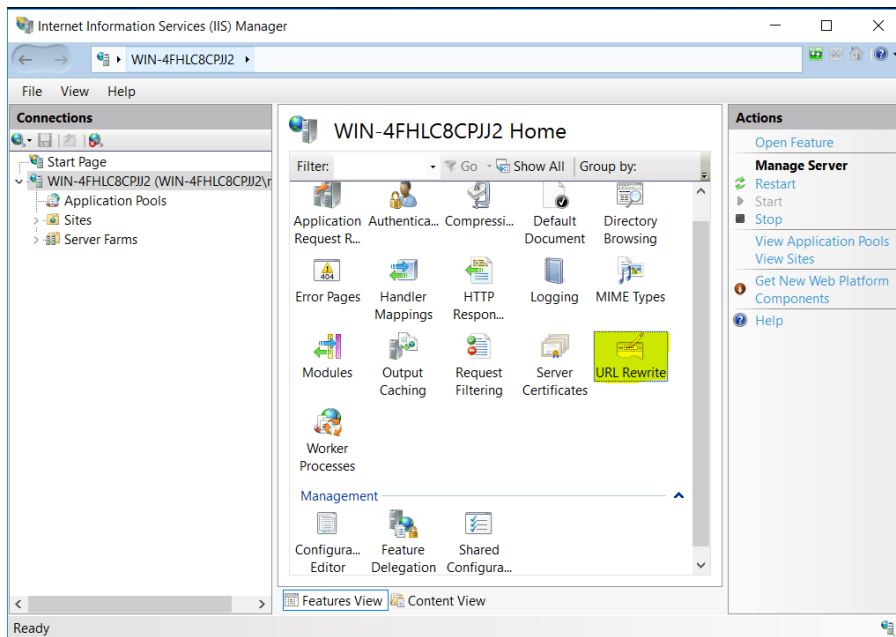
Those rules are triggered using filters on the URL path, so now only port 80 (the http port) will be open to the public network.

Public URL	Routed to
http://<public ip>/*	http://127.0.0.1:3000/*
ws://<public ip>/websocket	http://127.0.0.1:8009/websocket
http://<public ip>/gateway	http://127.0.0.1:8009/gateway

Important:

With this configuration, the AcuToWeb Gateway is hidden behind IIS and it is only reachable by using the HTTP port; therefore, it is mandatory to use the portgw parameter to override the gateway configuration; for example:

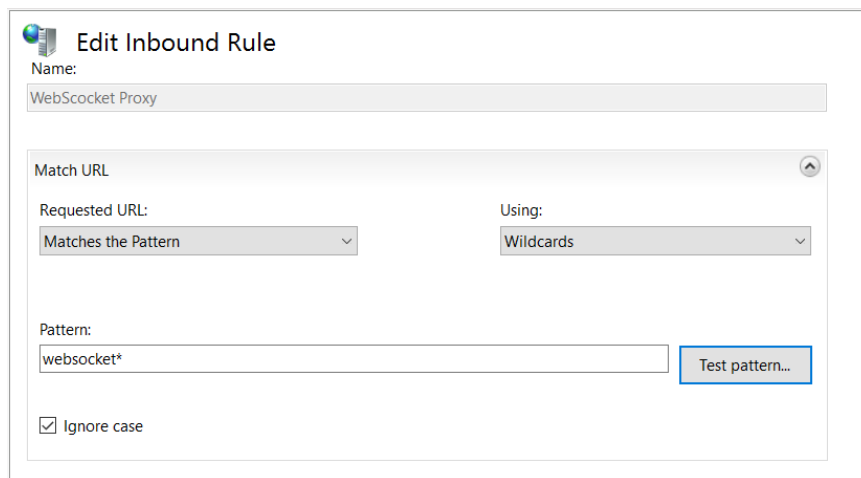
http://12.345.678.912?hostgw=12.345.678.912&portgw=80&alias=tour



Documentation about URL Rewrite

<https://docs.microsoft.com/en-us/iis/extensions/url-rewrite-module/url-rewrite-module-configuration-reference>

A rewrite **rule pattern** is used to specify a pattern to which the current URL path is compared. Current, in this context, means the value of the URL path when the rule is applied.



Edit Inbound Rule

Name: WebSocket Proxy

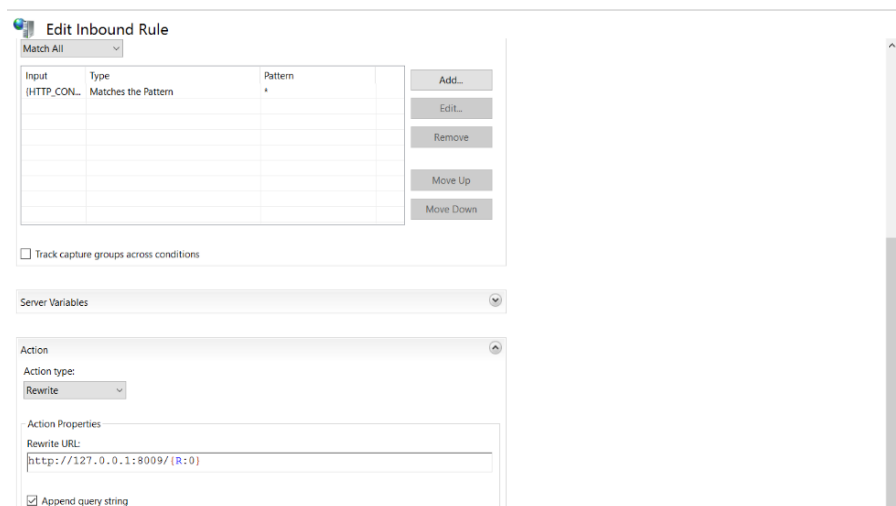
Match URL

Requested URL: Matches the Pattern Using: Wildcards

Pattern: websocket* [Test pattern...](#)

☒ Ignore case

Rule conditions allow defining additional logic for rule evaluation, which can be based on inputs other than just a current URL string.



Edit Inbound Rule

Match All

Input	Type	Pattern
(HTTP_CON...	Matches the Pattern	*

☐ Track capture groups across conditions

Server Variables

Action

Action type: Rewrite

Action Properties

Rewrite URL: http://127.0.0.1:8009/{R:0}

☒ Append query string

A **Rewrite** action replaces the current URL string with a substitution string.

A substitution string must always specify the URL path.

If you want you can add a condition stating this rule can be only executed when the hostname in the URL is contoso.com by adding a condition for **{HTTP_HOST}** matching contoso.com.

If a browser is configured to use an explicit proxy server then it will first issue the **HTTP CONNECT** method to that proxy server while establishing the WebSocket connection.

For example, to connect to the server example.com using the ws:// scheme (typically over port 80), the browser client sends the HTTP CONNECT method to the proxy server.

When the explicit proxy server allows the CONNECT method, the WebSocket connection upgrade handshake can be made and when that handshake succeeds, WebSocket traffic can start flowing unimpeded through the proxy server.

Below my URL Rewrite configuration

URL Rewrite							
Provides rewriting capabilities based on rules for the requested URL address and the content of an HTTP response. Inbound rules that are applied to the requested URL address:							
Name	Input	Match	Pattern	Action Type	Action URL	Stop Pro...	Entry Type
WebSocket Proxy	URL Path	Matches	websocket*	Rewrite	http://127.0.0.1:8009/{R:0}	True	Local
SockJS Proxy	{HTTP_CONNECT}	Matches the Pattern	*	Rewrite	http://127.0.0.1:8009/{R:0}	True	Local
	URL Path	Matches the Pattern	gateway*				
HTTP Proxy	{HTTP_HOST}	Matches the Pattern	*	Rewrite	http://127.0.0.1:3000/{R:0}	True	Local
	URL Path	Matches the Pattern	*				

See below the AcuToWeb Gateway configuration :

Gateway Service Properties

Gateway.config file

C:\ACUTOWEB\config\gateway.conf

Browse ...

AcuConnect host name

127.0.0.1

AcuConnect port

5632

Max Package Size

64000

TCP Port

8009

Handshake timeout

3

Web Server Port

3000

Public Root Directory

.\Web

Browse ...

File Cache Directory

C:\tmp\Web\cache

Browse ...

Logging

Log Level

7

Log filename

C:\ACUTOWEB\config\gateway.conf

Browse ...

☐ Request Username/Password Always

☐ Use WebSock compression

SSL

☐ Use SSL

SSL Key File

Browse ...

SSL Certification File

Browse ...

Dependencies

Add ...

Delete

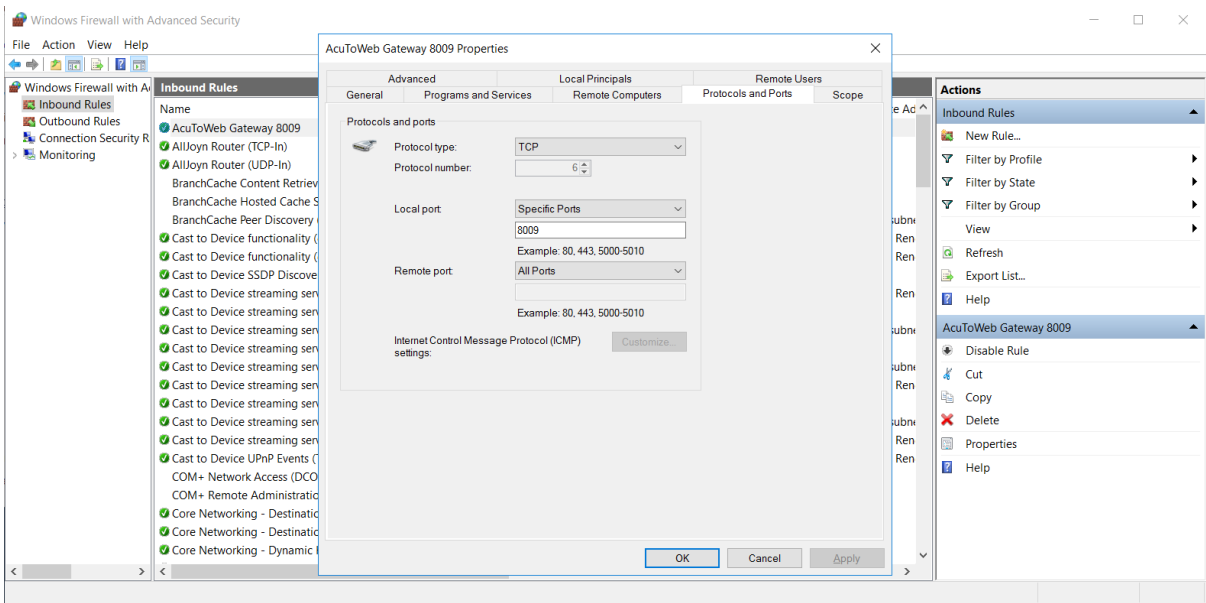
Themes ...

OK

Cancel

Important: **Due a limitation of ARR 3.0, this component does not support compressed WebSocket frames; therefore, IIS is not capable of routing them.** To make the WebSocket Proxy rule work, the WebSocket protocol must be enabled on the root node, but compression must not be enabled on the AcuToWeb node; add "ws_compression" : 0, to your gateway configuration file or clear the Use WebSock compression option on the Gateway Service Properties dialog box.

5.3. Firewall setting



5.4. Tests

My Windows Server 2016 in on a VM.
The AcuToWeb tests works fine.

With my VM in NAT configuration,

<http://192.168.119.151/?hostgw=192.168.119.151&portgw=80&alias=NewBOOKSF>

With my VM in Bridged configuration

<http://10.135.11.20/?hostgw=10.135.11.20&portgw=80&alias=NewBOOKSF>

In both case, IP and port number reference the IP address the Windows Server 2016 where IIS and AcuToWeb are installed and portgw reference the standard http port number 80 handle by IIS.

If I specify the gateway port of AcuToWeb, there is no Web Socket tunneling through IIS.

6. AcuToWeb and IIS on different platform

In my first setup, the AcuToWeb platform was on AZURE (See configuration below).

In both case, IP and port number reference the IP address of IIS platform and port number of IIS platform.

If I specify the gateway port of AcuToWeb and hostgw to IP of AcuToWeb platform, there is no Web Socket tunneling through IIS.

I had an Issue running the App on Google Chrome on Firefox but works fine in IEE.

With Google Chrome, nothing appears in the browser.

In the sample below, 10.135.11.20 is the IP address of the platform where IIS is setup and 52.232.1.93 is the IP address of the AZURE platform where AcuToWeb is setup.

<http://10.135.11.20/?hostgw=10.135.11.20&portgw=80&alias=NewBOOKSF>

URL Rewrite setup.

The screenshot shows the IIS URL Rewrite module configuration interface. It includes a table of inbound rules and a browser window below it.

Name	Input	Match	Pattern	Action Type	Action URL	Stop Pro...	Entry Type
WebSoccket Proxy	URL Path	Matches	websocket*	Rewrite	http://52.232.1.93:9400/{R:0}	True	Local
SockS Proxy	URL Path	Matches	gateway*	Rewrite	http://52.232.1.93:9400/{R:0}	True	Local
HTTP Proxy	URL Path	Matches	*	Rewrite	http://52.232.1.93:3005/{R:0}	True	Local

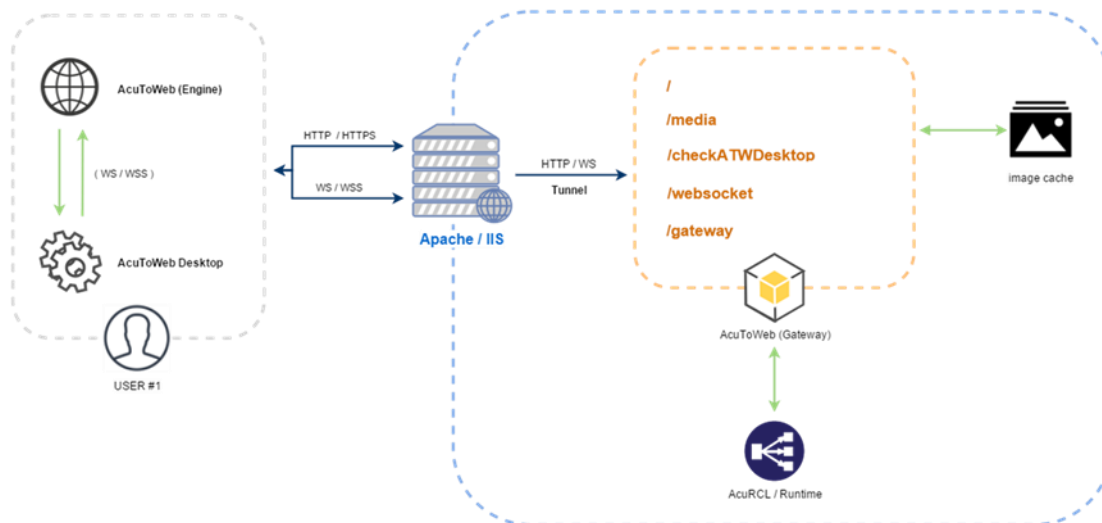
Below the table, a browser window is shown with the address bar displaying the URL: `http://10.135.11.20/?hostgw=10.135.11.20&portgw=80&alias=NewBOOKSF`. The browser tabs include 'AcuToWeb' and 'Word heading number blacke...'. The browser interface also shows 'Clip and Edit', 'Subscribe to Feed', and 'Annotate' options.



I make other tests with AcuToWeb on another platform which is not in the Cloud and it works fine with IEE and Google Chrome. I need to find time to check what is going wrong with the AZURE configuration setting.

7. AcuToWeb and IIS on the same platform with SSL

Architectures avec Serveur Web Standard (Apache/IIS)



7.1. Introduction

I made different tests.

For the first test, I created a self signed certificate.

Unfortunately browsers like Google Chrome shows warnings using self signed certificate.

For the second test, I use IIS to create a certificate request and I use the website

<http://www.getacert.com/> to sign the certificate request.

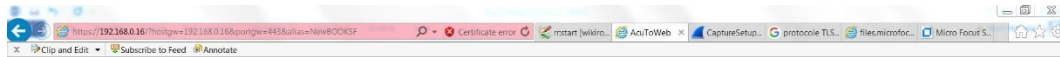
Unfortunately IIS doesn't allow to specify the Subject Alternative Name (SAN) information requested by latest version of Google Chrome.

For the third test, I created my own CA certificates and server certificates using openssl commands. It works fine. My advice is to use this method.

7.2. First test : With Self signed certificate generated by IIS

7.2.1. Introduction

Just keep in mind that visitors will see a warning in their browsers (like the one below) when connecting to an IIS site that uses a self-signed certificate until it is permanently stored in their certificate store (See sample below).



7.2.2. URL Rewrite setup & AcuToWeb setup

URL Rewrite							
Provides rewriting capabilities based on rules for the requested URL address and the content of an HTTP response.							
Inbound rules that are applied to the requested URL address:							
Name	Input	Match	Pattern	Action Type	Action URL	Stop Pro...	Entry Type
WebSocket Proxy	URL Path	Matches	websocket*	Rewrite	http://127.0.0.1:8009/(R:0)	True	Local
SockJS Proxy	URL Path	Matches	gateway*	Rewrite	http://127.0.0.1:8009/(R:0)	True	Local
HTTP Proxy	URL Path	Matches	*	Rewrite	http://127.0.0.1:3000/(R:0)	True	Local

Gateway Service Properties

Gateway.conf file

C:\ACUTOWEB\config\gateway.conf

Browse ...

AcuConnect host name

127.0.0.1

AcuConnect port

5632

Max Package Size

64000

TCP Port

8009

Handshake timeout

3

Web Server Port

3000

Public Root Directory

.\Web

Browse ...

File Cache Directory

C:\tmp\Web\cache

Browse ...

Logging

Log Level

7

Log filename

C:\ACUTOWEB\config\gateway.log

Browse ...

☐ Request Username/Password Always

☐ Use WebSocket compression

SSL

☐ Use SSL

SSL Key File

Browse ...

SSL Certification File

Browse ...

Dependencies

Add ...

Delete

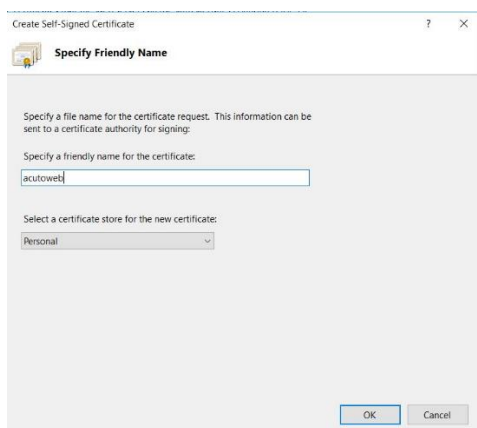
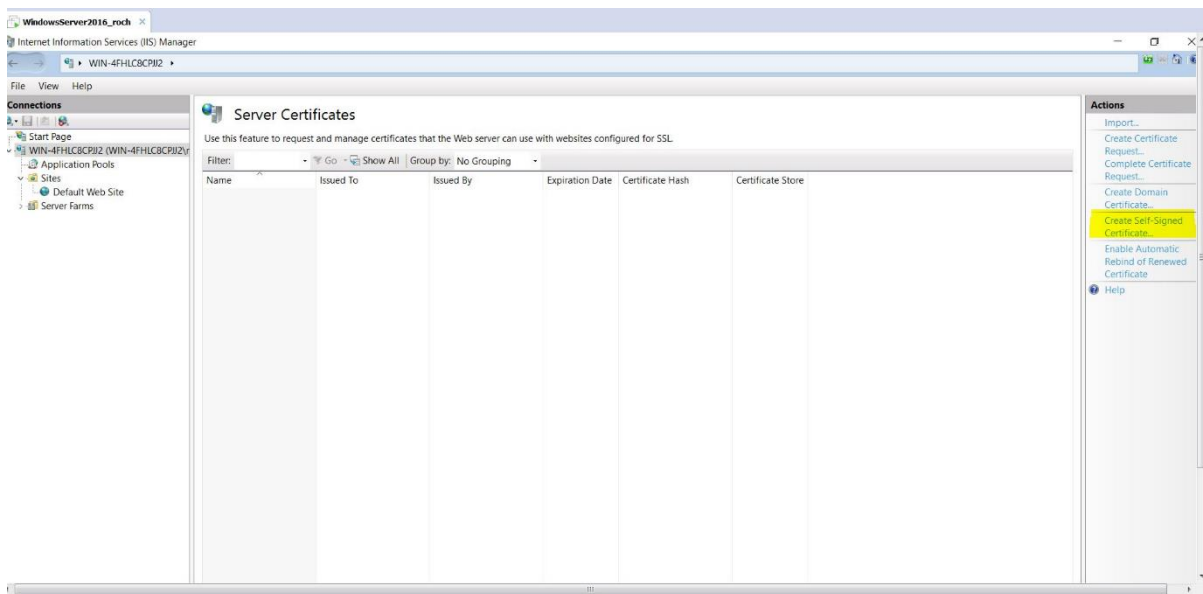
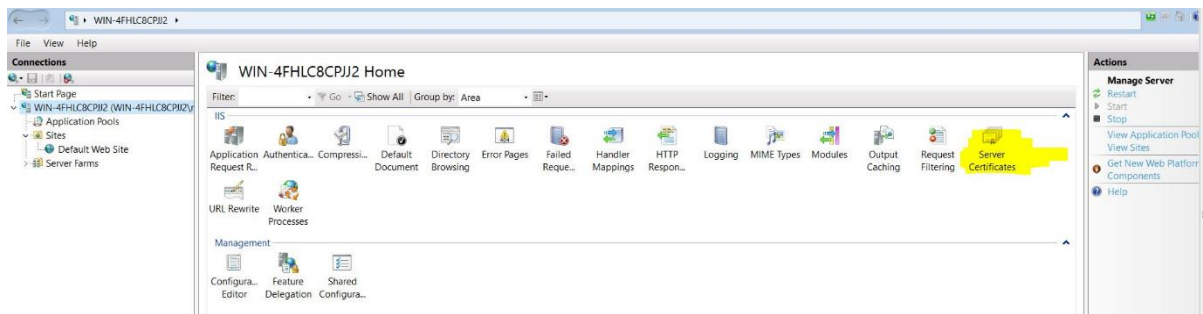
Themes ...

OK

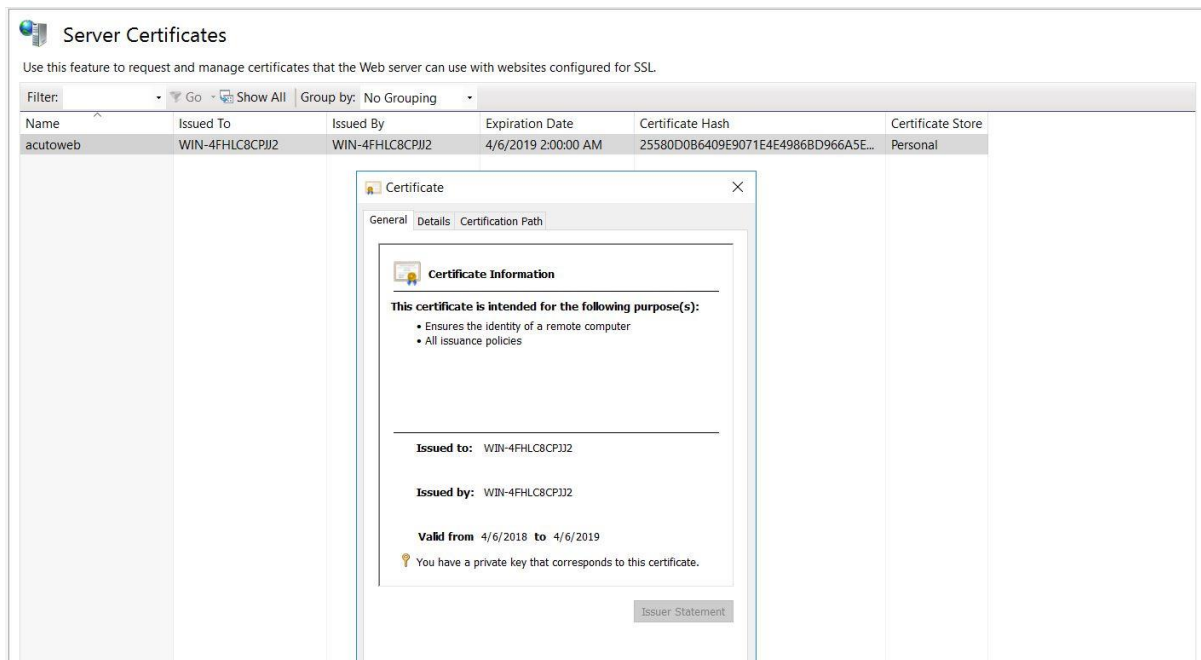
Cancel

7.2.3. Create a self-signed certificate with IIS

The pictures below show how to create a self-signed certificate with IIS.

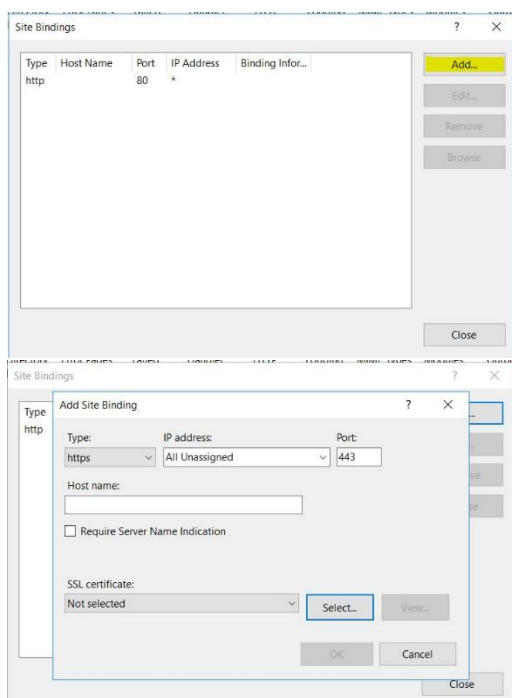
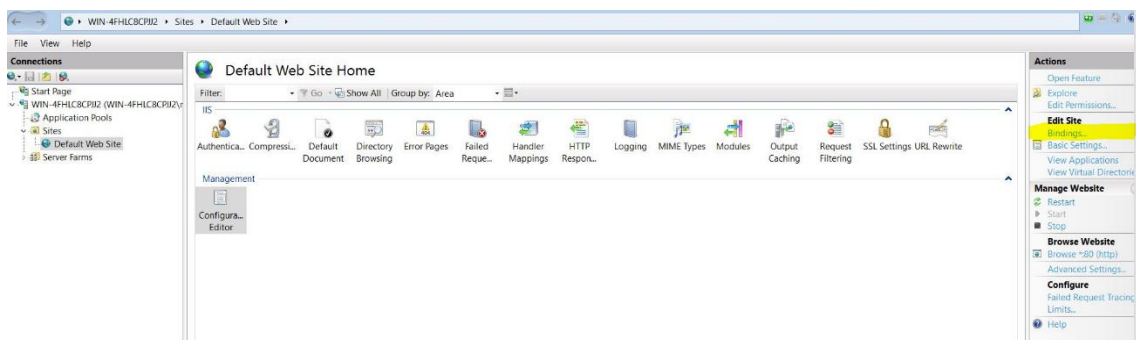


Click OK and the self-signed certificate is created.

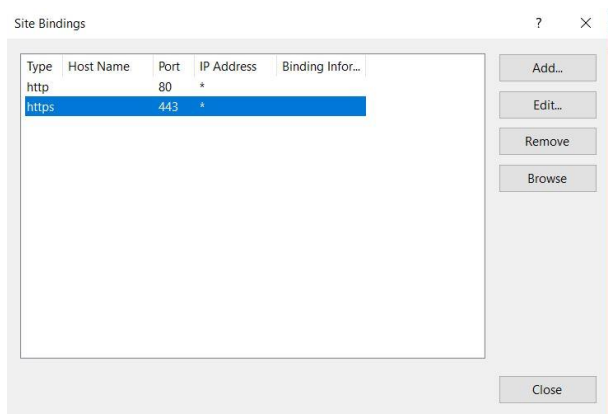
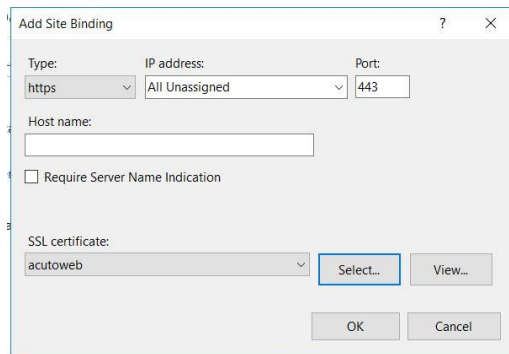
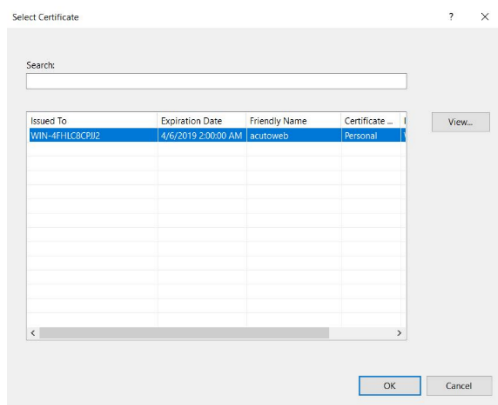


7.2.4. Setup HTTPS under IIS

To setup https under IIS, you need to use the **Bindings** link.



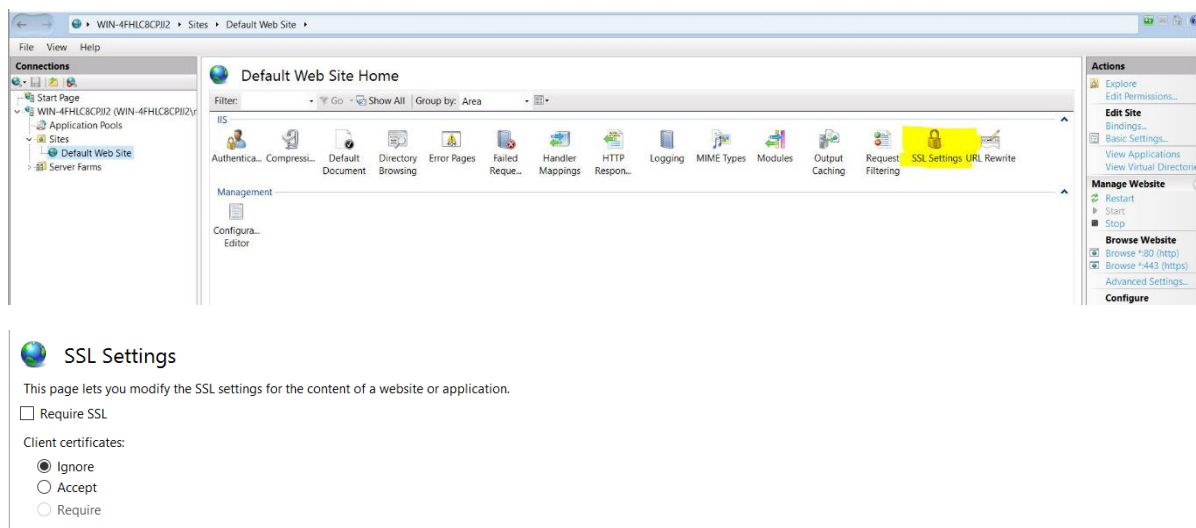
You need to select the certificate added in last step.



7.2.5. SSL Settings

This configuration is used for Client-certificate authentication setup.

In this demo, Client-certificate authentication is not used.



Client-certificate authentication can be optional or mandatory, or not used at all.

Ignore is when it's not used at all.

Accept will take a certificate if it's presented, but will also continue with connections where the client doesn't present one.

Require only continues with connections that have a client certificate.

Client-certificate authentication is something that can only be initiated by the server in SSL/TLS, so this terminology isn't quite correct, but that's what's used in IIS.

Information about client certificate:

Some of the misconceptions that we see on a regular basis are:

- Client certificates are needed to make SSL work properly.
- When using client certificates, you don't need a certificate on the server.
- A client certificate that is issued by any certification authority will work with any server.
- If you issue a client certificate, your Web server will automatically accept it.

The first and perhaps most confusing point is the difference between client certificates and Secure Sockets Layer (SSL) server certificates. Although client certificates and SSL server certificates both use certificates, they are not directly related to each other. **SSL server certificates provide encryption and security functionality. Client certificates provide user authentication functionality.** If this makes sense, the rest should be easy.

Client certificates are issued to a user by a certification authority. They consist of the public key

portion of the certificate and a private key that is held only by the entity to which the certificate is issued. The certification authority may be a well-known public organization that provides certificate services as part of its business, or it could be an internal server that only your company uses. In either case, the client certificate will have certain information that identifies the user either individually or as part of a group.

In IIS, you have the option of ignoring, accepting, or requiring client certificates when a user accesses resources on your server. Ignoring certificates simply means that you are not using them, will not ask the client for one, and will discard one if it is sent to your server. If you choose to accept certificates, your server will prompt for a certificate but will not necessarily deny access if a certificate is not provided. If you require client certificates, the user must supply a valid certificate or the user will receive an error message.

For a certificate to work properly, certain requirements must be met on both the server and the client. Each side has a list of root certification authorities that they trust. When the server prompts for a certificate, the request includes a list of the certification authorities that the server trusts. The client then compares this list to the list of certification authorities that the client trusts and creates a list of the ones that match. Then, the client compares that list to the client certificates it has and determines which, if any, certificates have been issued by certification authorities that both the client and the server trust. Depending on the client, you may see a list of certificates to choose from if there is more than one certification authority that both sides trust. The client then sends the public portion of the certificate to the server. At this point, the server generally checks to make sure that the certificate is valid and, if no mapping is performed, the communications between the client and the server can continue.

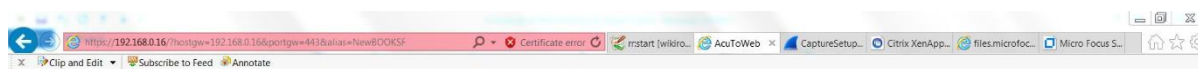
This is the most basic functionality of client certificates. At this point, the server knows only that the client has a valid certificate.

Here is where things get interesting. The server can be configured to do a mapping of the certificate to a user account. This can be either a one-to-one mapping, where the specific certificate is mapped to a single user account, or a many-to-one mapping, where the server uses certain fields in the certificate information to map any matching certificate to a designated user account. When a mapping is used, the certificate allows the user to be granted or denied access to resources as a particular user. When using client certificates in this manner, you do not have to use any other authentication method.

7.2.6. SSL Settings

<https://192.168.0.16/?hostgw=192.168.0.16&portgw=443&alias=NewBOOKSF>

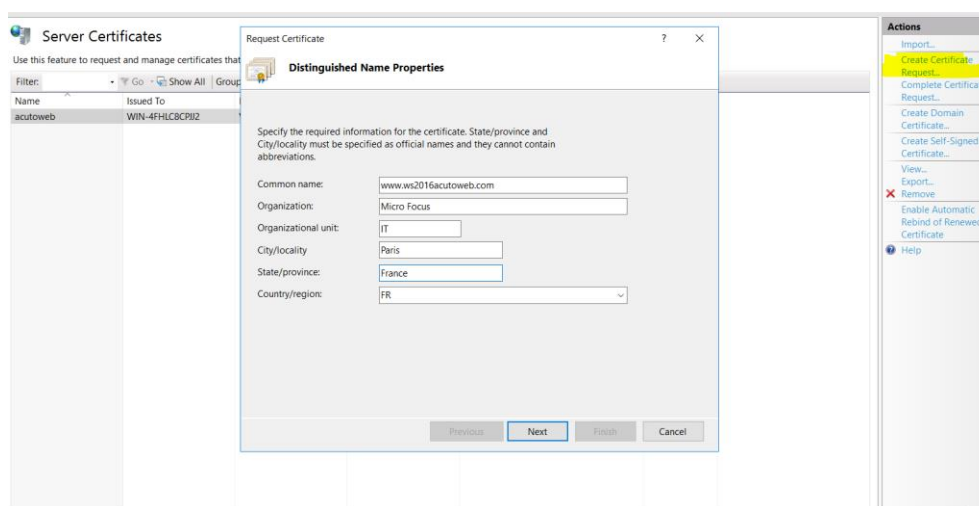
You see below the warning about using self-signed certificate.



7.3. Second Test : Create a certificate Request through IIS and use a free CA Web Site to sign this request

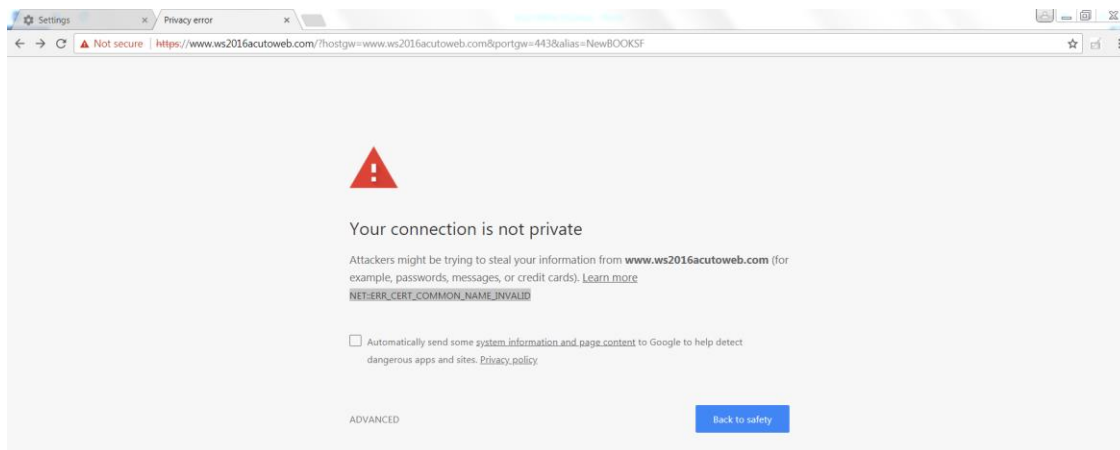
7.3.1. Create a certificate request with IIS

When I click on the Create Certificate Request... link, the following window appears. In the common name, I put the hostname of my server platform.

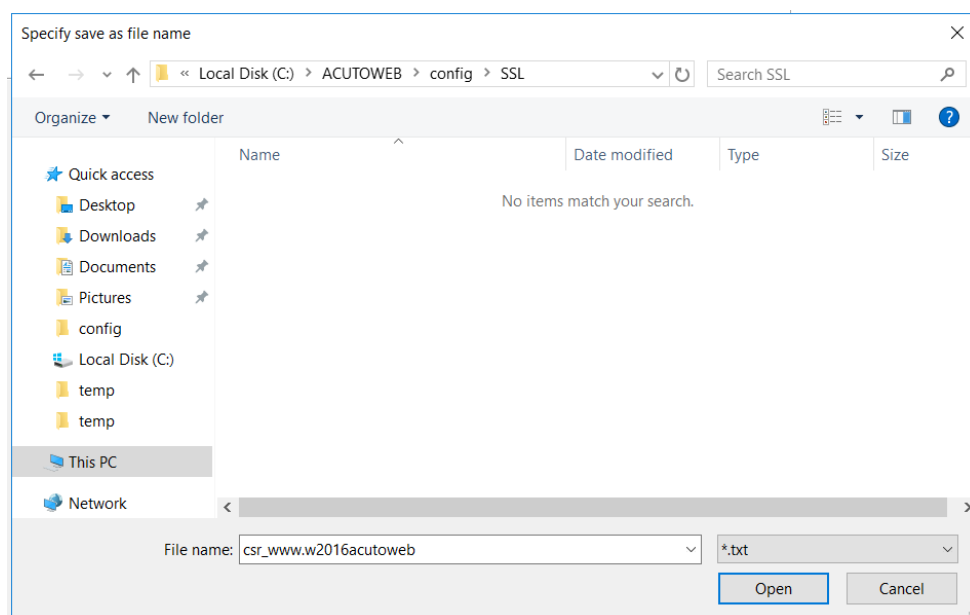
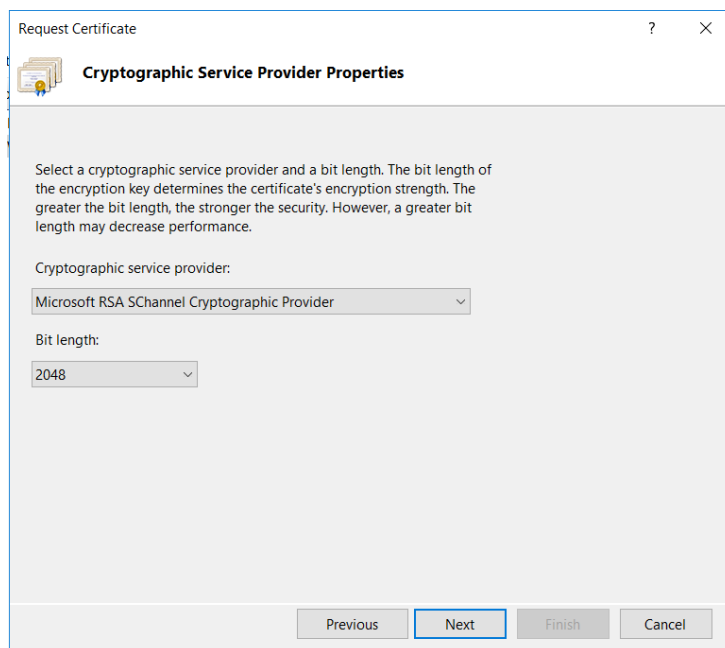


Please Note: If you are issuing a CSR for a Multi-Domain certificate and are wondering how to add SAN's, you will not be prompted in the wizard and will manually need to enter them on Entrust.net when placing your request for the certificate.


The fact that IIS doesn't allow to set SAN information can lead to future problems with browsers as Google Chrome (See picture next page).



After clicking on the Next button, the following window appears.



Request Certificate
?
X


File Name

Specify the file name for the certificate request. This information can be sent to a certification authority for signing.

Specify a file name for the certificate request:

Previous
Next
Finish
Cancel

Below the content of the certificate request file.

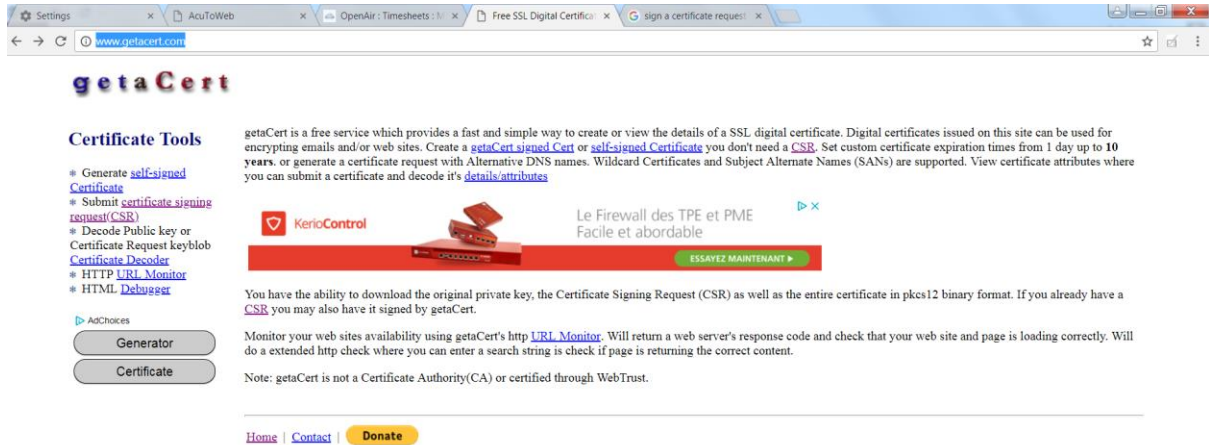
```

1  -----BEGIN NEW CERTIFICATE REQUEST-----
2  MIIEZTCCA00CAQAwcjELMAkGA1UEBhMCRLIx DzANBgNVBAGtMBkZyYW5jZTEOMAwG
3  A1UEBwFUGFyaXNkFDASBgNVBAoMC01pY3JvIEZvY3ZvMQswCQYDVQQLDAJJVDEf
4  MB0GA1UEAwWd3d3LndzMjAxNmFjdXRvd2ViLmNvbTCCASIwDQYJKoZIhvcNAQEB
5  BQADggEPADCCAQoCggEBBjWoi/HuLOAewWPh81SxYy78sgWbGQ18ekji7SIiwPXO
6  hykXP+WQeGu4w47JxRY4GvYk+6FNUiu6y02j5OgIf t0In4URyRDXPiJ5DZ7aZy+n
7  mij6U4BJ2Y7AtkGtaMv6tT4LtoCKHM7xjFhsQ0VEYjHHn+8oS8mQs10yQxx/Rk8L
8  W3Boa+m3w2T6Qxtbg6DbE6kdH4/5ACSQFIYB00EtG1HpbX6T0UjTUBv1JjKhIXx
9  4hL91dx+jzuFIFSoXipg1i1rel8x+XGWedGCZsKlaAj6zmhjEDVkp6jW8P5u9TGU
10 RgWgUNMmUwa6FR5Le3EyGkQKyzPT0T2qKBW4ezvY8QcCAwEAAaCAawwHAYKKwYB
11 BAGCNw0CAzEOFGwxMC4wLjE0MzkzLjIwRgYJKwYBAGCNxUUMTkwNwIBBQwPV01o
12 LTRGSExDOENQSkoyDBRXSU4tNEZITEM4Q1BKsjJccm9jaAwLSW51dE1nci5leGuw
13 cgYKKwYBAGCNw0CAjFkMGICAQEEwGBNAGkAYwByAG8AcwBvAGYAdAAgAFIAUwBB
14 ACAAUwBDAGgAYQBAG4AZQBsACAAQwByAHkACAB0AG8AZwByAGEACABoAGkAYwAg
15 AFAAcgBvAHYAaQBkAGUAcgMBADCBzwYJKoZIhvcNAQkOMYHBMIG+MA4GA1UdDwEB
16 /wQEAWIE8DATBgNVHSUEDDAKBggrBgEFBQcDATB4BgkqhkiG9w0BCQ8EazBpMA4G
17 CCgGSIb3DQMCAGIAgDAOBggqhkiG9w0DBAICAIAwCwYJYIZIAWUDBAEQMAoGCCqG
18 SAFlAwQBLTBglghkgBZQMEEAIwCwYJYIZIAWUDBAEFMAcGBSSoAwIHMAoGCCqG
19 SIb3DQMHMB0GA1UdDgQWBRECY3K2z4CS4pksB+g1o3V4/UusDANBgkqhkiG9w0B
20 AUUFAAOCAQEAEad+pPtZSGFbaIp3/oUy48Y0xa6d9YkMcshAafxhyO/osf0SGWWHO
21 U7/XhUR50QqpIQZi7b/ovKMct935QT+nYib+9aQTKeadIgtF93PEv8Wdz3ZngMh4
22 bndsblam2sZLvRz2+v/mEkEHS6g0BzvH/I31aNQ30rjP5QK9P2nubdsjnC9tyxv
23 Sng8VQbpkJwx8K0BuRcnqWt/Sye2LzdN8sgWgWyPa/xXe146odzKGcX0F6RUKP5
24 Z6qEcwWY/tDKtpXGauQ09d+a4w8aOWjRqwyHudH5ifi3DK/Esresfx30erAo5VM
25 /oJW2AofnM5Tx1ME+OSH15vvsj6EFmuGcw==
26  -----END NEW CERTIFICATE REQUEST-----
27

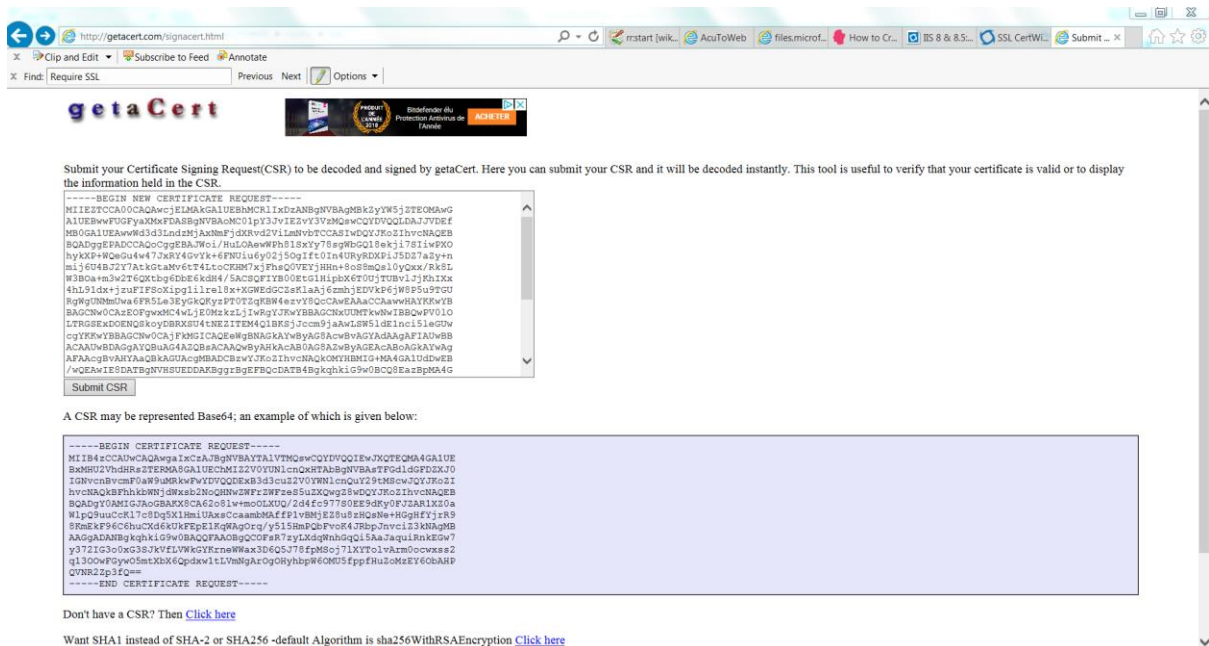
```

7.3.2. Signing the certificate request

I use the site www.getacert.com to sign the certificate.
I click on the link **submit certificate signing request (CSR)**.



I put the content of the certificate request file generated in the previous step in the text box and I click on the **Submit CSR** button.



Decoded CSR - Summary

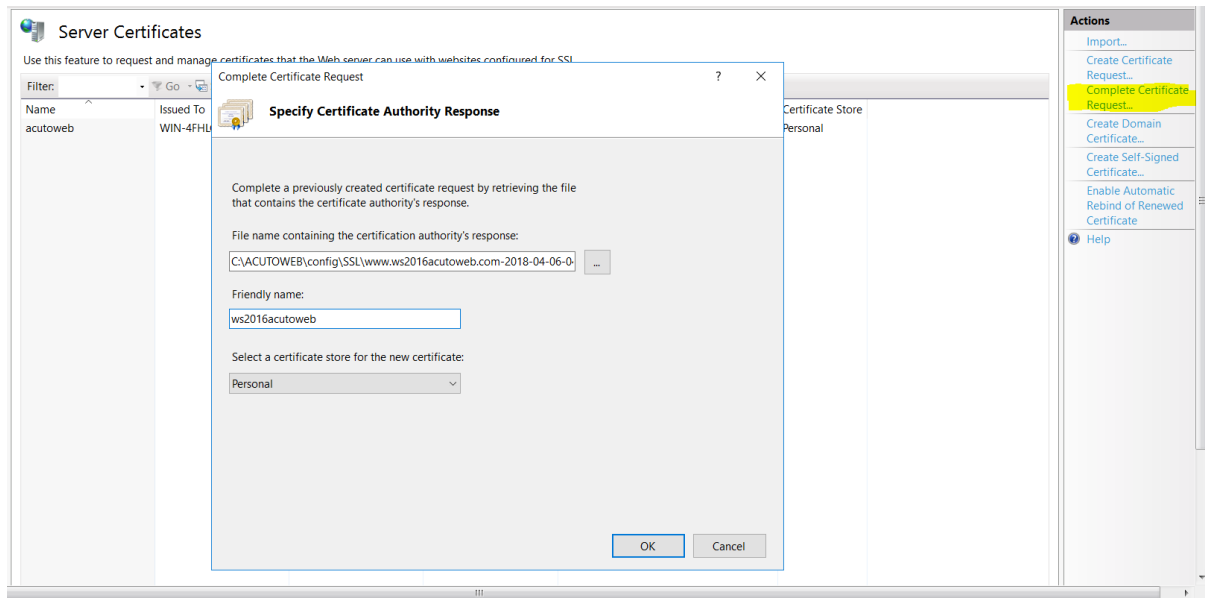
Attribute	Value
C	FR
ST	France
L	Paris
O	Micro Focus
OU	IT
CN	www.ws2016acutoweb.com

- Your signed public certificate/key(.cer) : www.ws2016acutoweb.com-2018-04-06-045527.cer
- In pem format(.pem) : www.ws2016acutoweb.com-2018-04-06-045527.pem
- Getacert's public certificate/key(.cer) : getacert.cer

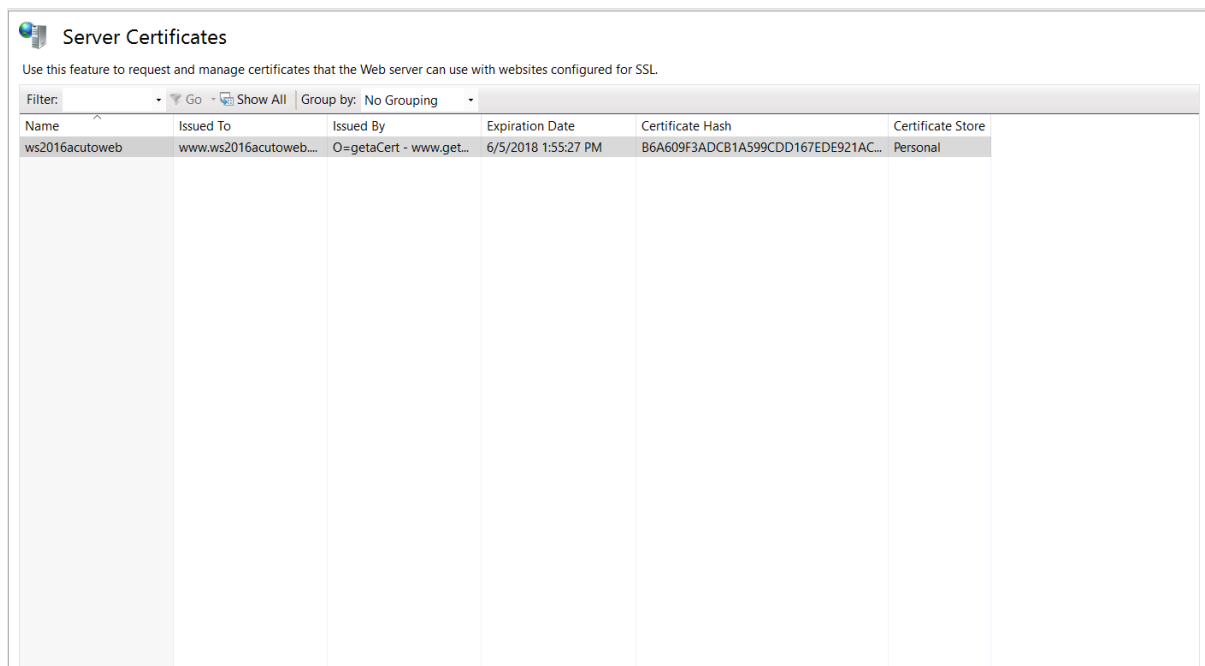
This certificate is signed for 60 days. If you would like a 10 year certificate please Donate and we will send you a 10 year CSR signing page

7.3.3. Generate the Server Certificate

I copied the signed public certificate generated on my Windows Server 2016 platform and I click on the complete Certificate Request button.

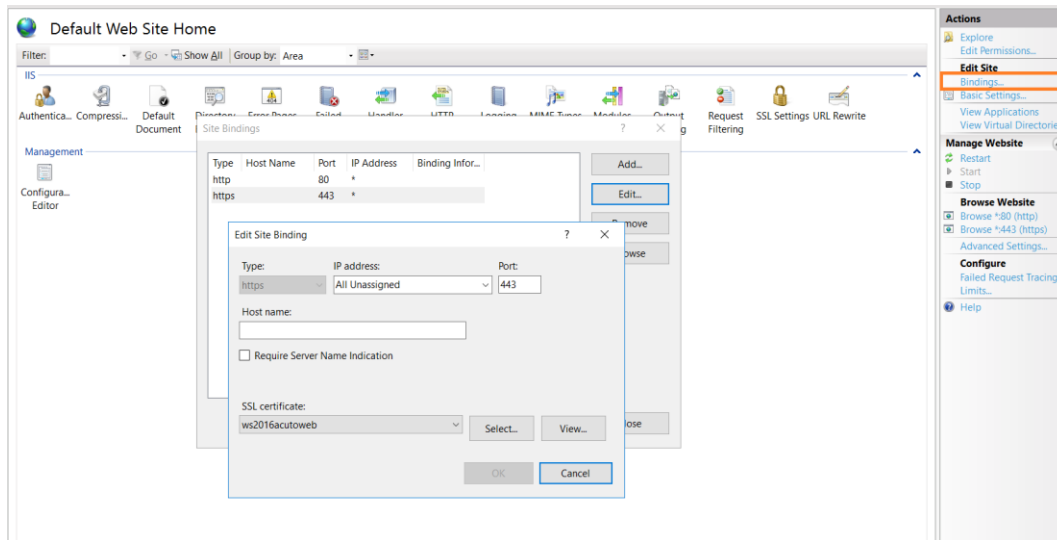


This generates a Server Certificates.



7.3.4. HTTPS setup

The certificate server should be associated with the HTTP configuration.



7.3.5. Importing the getacert CA certificate in the Browser

On the www.getacert.com website, there is a link to get the Getacert's public certificate. This certificate needs to be imported in your browser.



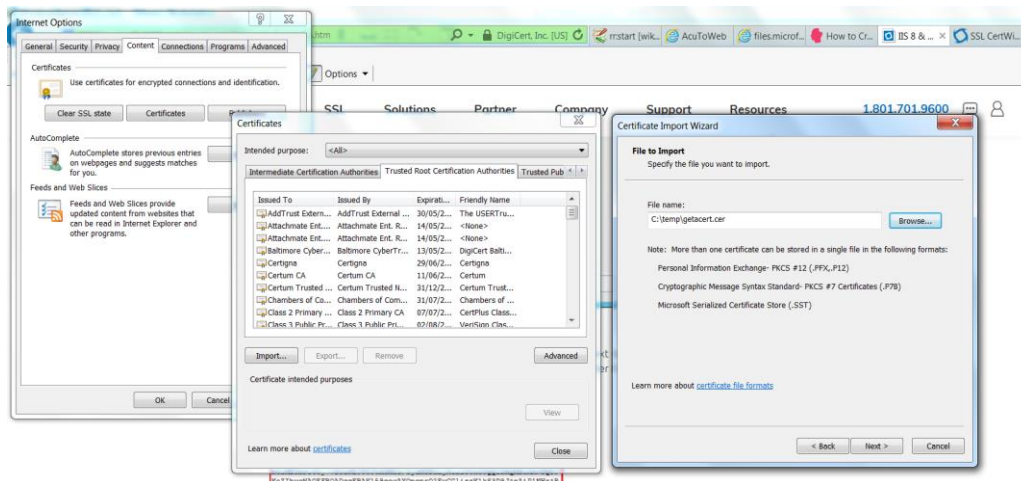
Decoded CSR - Summary

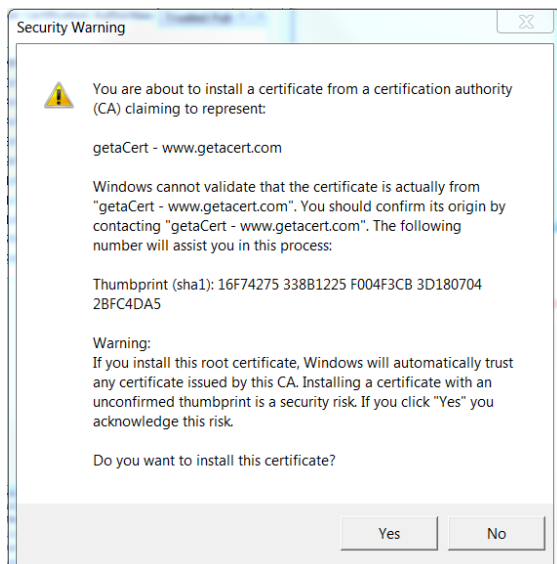
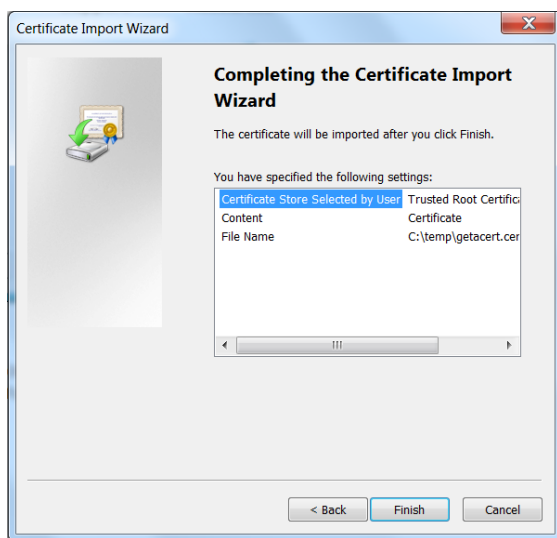
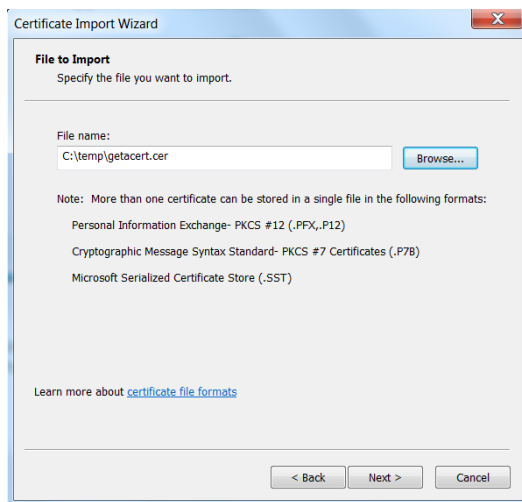
Attribute	Value
C	FR
ST	France
L	Paris
O	Micro Focus
OU	IT
CN	www.ws2016acutoweb.com

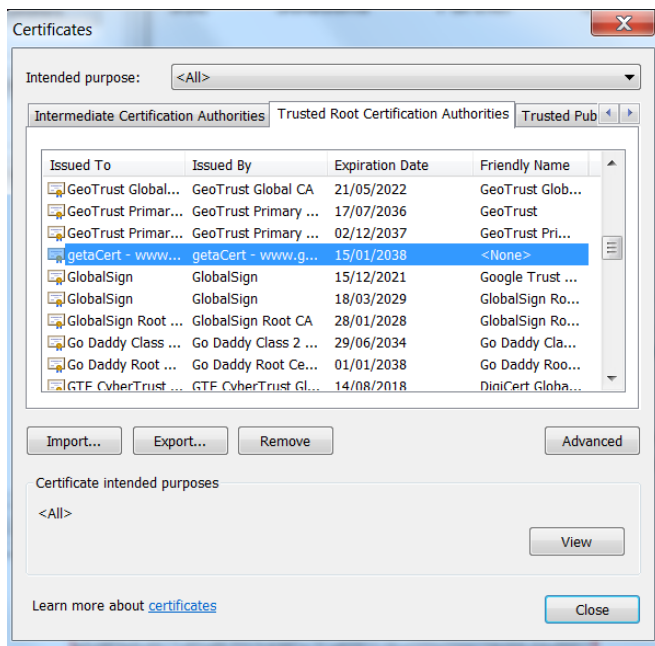
- Your signed public certificate/key(.cer) : www.ws2016acutoweb.com-2018-04-06-045527.cer
- In pem format(.pem) : www.ws2016acutoweb.com-2018-04-06-045527.pem
- Getacert's public certificate/key(.cer) : getacert.cer

This certificate is signed for 60 days. If you would like a 10 year certificate please Donate and we will send you a 10 year CSR signing page

Below the process to import this getacert certificate in your browser.



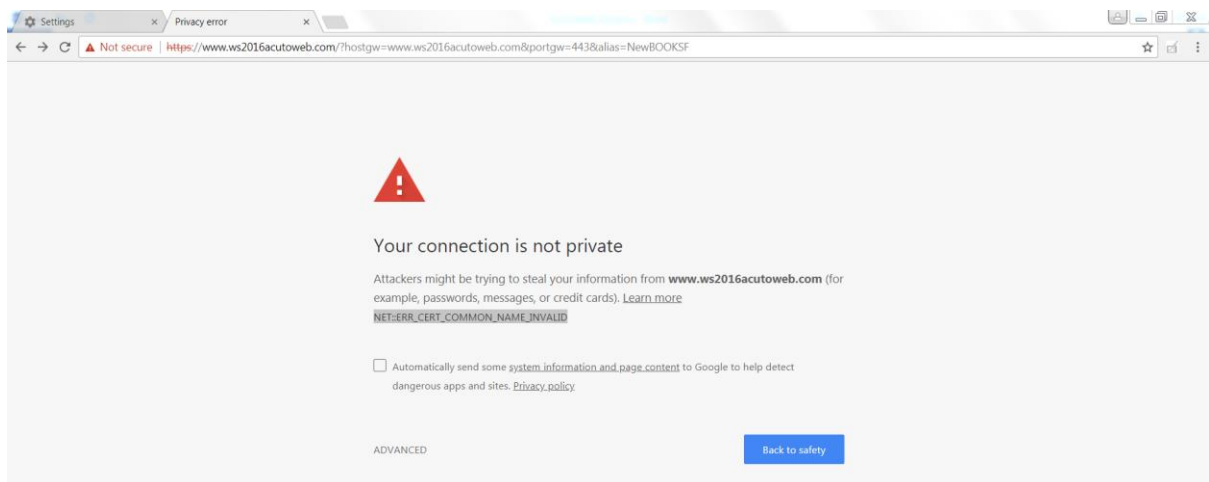




7.3.6. Tests

With Google Chrome

<https://www.ws2016acutoweb.com/?hostgw=www.ws2016acutoweb.com&portgw=443&alias=NewBOOKSF>



In Chrome 58 if you get the error `NET::ERR_CERT_COMMON_NAME_INVALID` you need to update and re-issue your certificates (eg. for the Web Filtering via Forcepoint) using a resolveable Fully Qualified Domain Name (FDQN) in the certificate Subject Alternative Name (SAN).

This is because Chrome 58 no longer matches the Common Name (CN) in certs.

<https://bugs.chromium.org/p/chromium/issues/detail?id=700595> - Chrome no longer accepts certificates that fallback to common name (`ERR_CERT_COMMON_NAME_INVALID`)

Chromium removed support for matching common name in certificates in M58:

* Issue 308330

* <https://www.chromestatus.com/features/4981025180483584>

7.4. Third Test : Create a CA certificate and a server certificate with openssl commands

7.4.1. Introduction

I use openssl commands on redhat to create a CA certificate and a server certificate. openssl configuration file and scripts are described in the chapter 9 : Annexes.

All information I used are from the following article.

<https://fabianlee.org/2018/02/17/ubuntu-creating-a-trusted-ca-and-san-certificate-using-openssl-on-ubuntu/>

7.4.2. Create the openssl configuration file use to create a CA Certificate and a server certificate

I work on a redhat platform to generate both CA and server certificates.

```
$ cp /etc/pki/tls/openssl.cnf .
```

```
$ mv openssl.cnf ws2016acutoweb.cnf
```

I modified the openssl configuration file `ws2016acutoweb.cnf` according to the Web Article. Following information is extracted from the Web Article.

Under the `[v3_ca]` section, add the following values. For the CA, this signifies we are creating a CA that will be used for key signing.

```
[ v3_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer
basicConstraints = critical, CA:TRUE, pathlen:3
keyUsage = critical, cRLSign, keyCertSign
nsCertType = sslCA, emailCA
```


Then under the “[v3_req]” section, set the following along with all the valid alternative names for this certificate.

```
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
#extendedKeyUsage=serverAuth
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = mydomain.com
DNS.2 = *.dydomain.com
```

Also uncomment the following line under the “[req]” section so that certificate requests are created with v3 extensions.

```
req_extensions = v3_req
```

When we generate each type of key, we specify which extension section we want to use, which is why we can share \$prefix.cnf for creating both the CA as well as the SAN certificate.

It's important to set in the openssl configuration file, **alt_names** section correctly. Below, extract from my openssl file configuration (**ws2016acutoweb.cnf**). www.ws2016acutoweb.com is the hostname of my Windows 2016 Server platform where IIS is installed.

```
[ alt_names ]
DNS.1 = www.ws2016acutoweb.com
DNS.2 = *.ws2016acutoweb.com
```

7.4.3. Create the CA Certificate

Now we will start using OpenSSL to create the necessary keys and certificates. First generate the private/public RSA key pair. This encodes the key file using a passphrase based on AES256.

```
$ openssl genrsa -aes256 -out ca.key.pem 2048
Enter pass phrase for ca.key.pem: toto
$ chmod 400 ca.key.pem
```

Then we need to create the self-signed root CA certificate.

```
$ openssl req -new -x509 -subj "/CN=rochCA" -extensions v3_ca -days 3650 -key ca.key.pem -sha256  
-out ca.pem -config ws2016acutoweb.cnf  
Enter pass phrase for ca.key.pem: toto
```

You can verify this root CA certificate using:

```
$ openssl x509 -in ca.pem -text -noout
```

This will show the root CA certificate, and the 'Issuer' and 'Subject' will be the same since this is self-signed. This is flagged as "CA:TRUE" meaning it will be recognized as a root CA certificate; meaning browsers and OS will allow it to be imported into their trusted root certificate store.

```
$ openssl x509 -in ca.pem -text -noout
```

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 10028972270554491369 (0x8b2e1124f0a4a9e9)

Signature Algorithm: sha256WithRSAEncryption

Issuer: CN=rochCA

Validity

Not Before: Apr 6 13:45:09 2018 GMT

Not After : Apr 3 13:45:09 2028 GMT

Subject: CN=rochCA

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (2048 bit)

Modulus:

00:b1:d6:37:06:98:74:b6:30:f0:33:4a:7d:f5:0d:
ae:7d:cf:e6:29:2d:2a:87:54:14:38:7d:d1:cb:56:
61:9e:14:1a:11:65:f8:78:7d:9c:b2:27:57:14:ac:
2e:d4:df:32:93:12:46:09:4d:1b:d5:d4:03:54:a2:
4d:a7:33:d1:62:d2:ce:f0:13:7a:35:fe:5e:06:f7:
08:61:27:1f:2a:0e:4c:18:f4:5a:b8:6e:d3:ad:98:
a6:e9:18:a5:10:2f:17:62:d8:5b:fe:2e:e3:cf:0c:
82:ee:0f:ab:0c:a1:81:f1:5b:b4:6f:07:ae:c9:2c:
77:44:6a:8b:91:f9:90:95:d0:e6:ea:91:4e:2d:5f:
64:10:97:86:40:de:16:39:62:38:b3:28:80:40:ac:
2f:7f:29:af:ac:29:f1:74:f6:ff:c2:2b:1a:bd:66:
a4:85:78:61:29:f8:ae:eb:2b:71:98:ee:33:54:d1:
ee:55:f8:4b:9d:b8:5d:3b:d9:b3:71:f0:b3:98:94:
58:b3:70:ec:1b:68:b0:bf:ab:17:95:58:c0:22:7a:
0e:1a:71:eb:e6:fc:5c:56:d5:73:3a:1f:aa:e3:53:
66:88:37:55:d1:a6:e9:42:8a:d3:70:62:55:87:7b:
4f:04:9e:59:c5:83:49:42:4a:74:27:44:fe:1a:a9:
a9:7b

Exponent: 65537 (0x10001)

X509v3 extensions:

X509v3 Subject Key Identifier:

C2:6A:39:7D:6D:2A:5C:6B:99:98:D8:72:BF:0A:F0:DA:D6:E1:F3

X509v3 Authority Key Identifier:

keyid:C2:6A:39:7D:6D:2A:5C:6B:99:98:D8:72:BF:0A:F0:DA:D6

X509v3 Basic Constraints: critical

CA:TRUE, pathlen:3

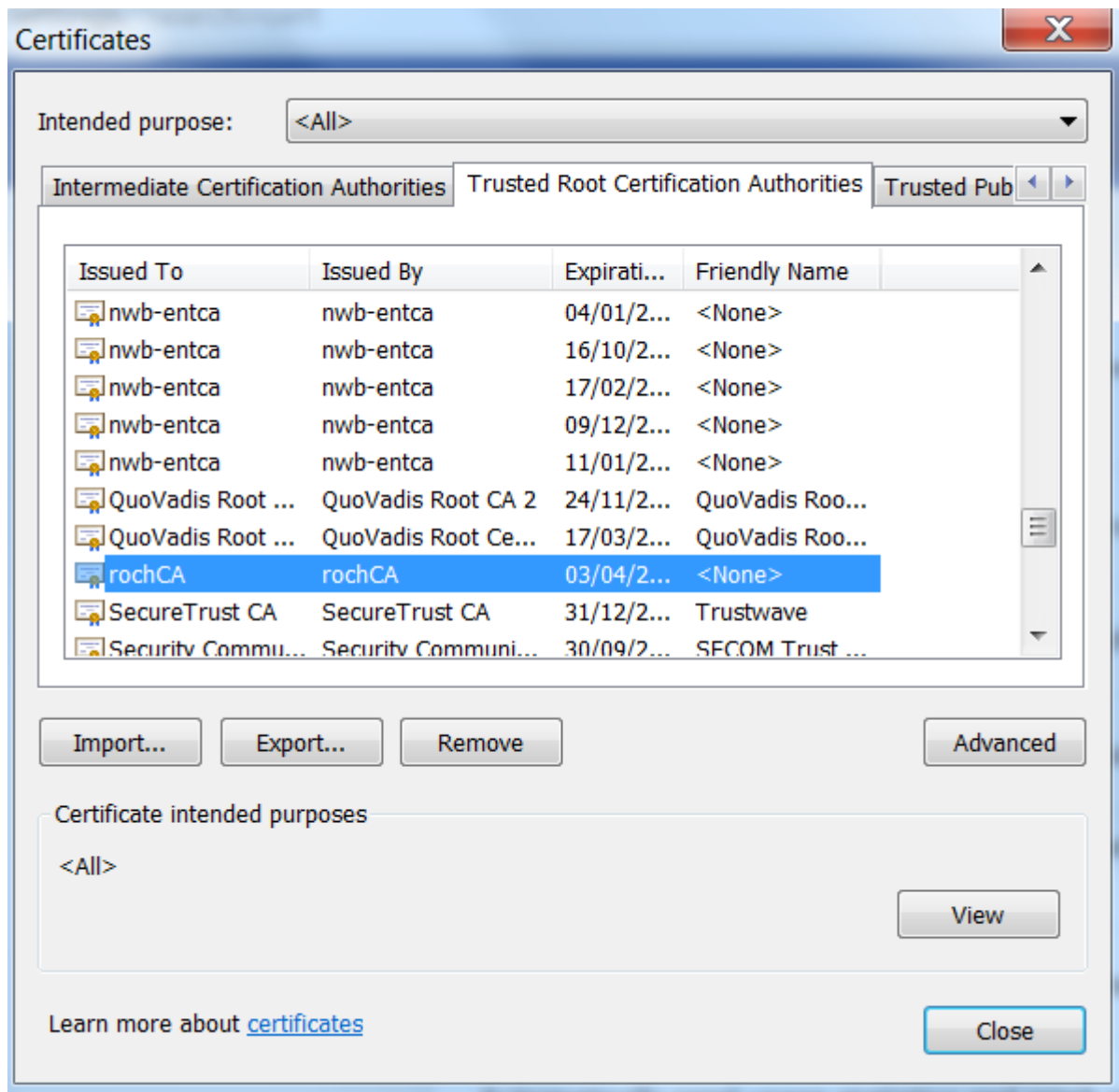
X509v3 Key Usage: critical

Certificate Sign, CRL Sign

Signature Algorithm: sha256WithRSAEncryption

7.4.4. Import the CA certificate in your browser

The process to import a CA certificate was described before in this document.



7.4.5. Create Server certificate signed by CA

With the root CA now created, we switch over to the server certificate.

First generate the private/public RSA key pair:

```
$ openssl genrsa -out ws2016acutoweb.key.pem 2048
```

We didn't put a passphrase on this key simply because the CA is more valuable target and we can always regenerate the server cert, but feel free to take this extra precaution.

Then create the server cert signing request:

```
openssl req -subj "/CN=www.ws2016acutoweb.com" -extensions v3_req -sha256 -new -  
key ws2016acutoweb.key.pem -out ws2016acutoweb.csr
```

Then generate the server certificate using the: server signing request, the CA signing key, and CA cert.

```
$ openssl x509 -req -extensions v3_req -days 3650 -sha256 -in ws2016acutoweb.csr -CA ca.pem -  
CAkey ca.key.pem -CAcreateserial -out ws2016acutoweb.crt -extfile ws2016acutoweb.cnf
```

Signature ok

Verify the certificate:

```
$ openssl x509 -in ws2016acutoweb.crt -text -noout
```

This will show the certificate, and the 'Issuer' will be the CA name, while the Subject is the prefix. This is not set to be a CA, and the 'Subject Alternative Name' field contains the URLs that will be considered valid by browsers.

Issuer:

CN=rochCA

...

Subject:

CN=www.ws2016acutoweb.com

...

X509v3 Basic Constraints:

CA:FALSE

X509v3 Key Usage:

Digital Signature, Non Repudiation, Key Encipherment

X509v3 Subject Alternative Name:

DNS:www.ws2016acutoweb.com, DNS:*.ws2016acutoweb.com

7.4.6. Prepare the deployment of the server certificate in IIS

Servers like HAProxy want the full chain of certs along with private key (server certificate+CA cert+server private key).

While Windows IIS wants a .pfx file.

Here is how you would generate those files.

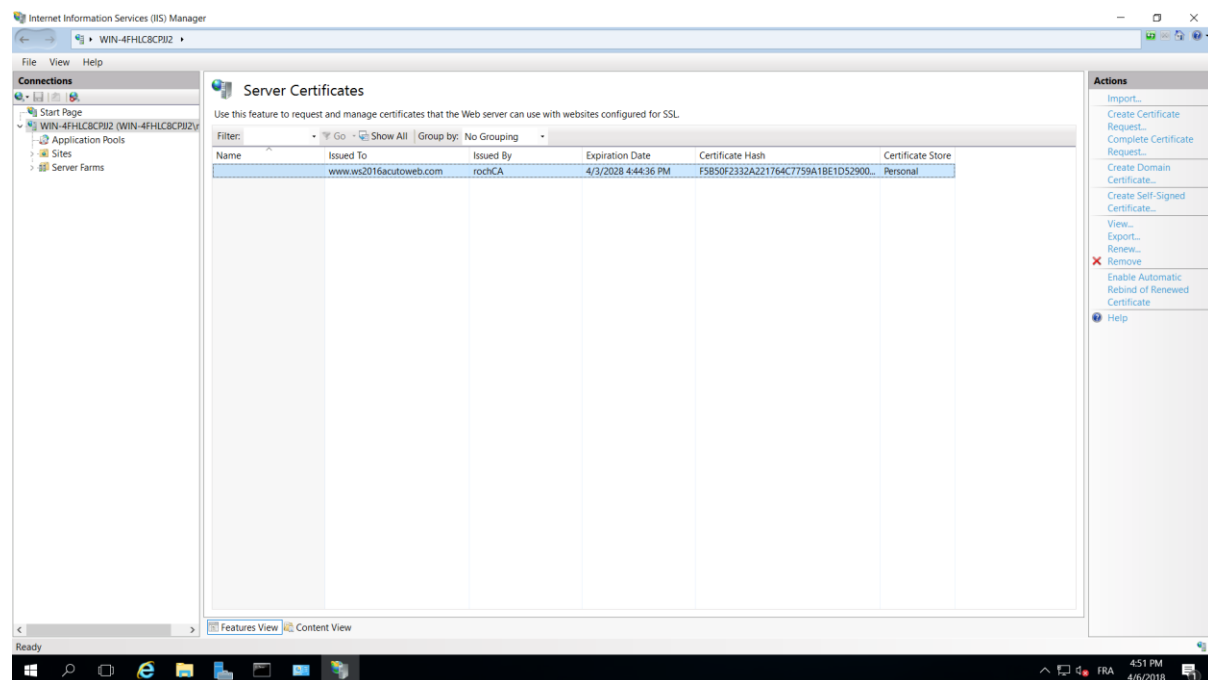
```
[roch@redhat6164 OPENSSL]$ openssl pkcs12 -export -out ws2016acutoweb.pfx -inkey  
ws2016acutoweb.key.pem -in ws2016acutoweb.crt -certfile ca.pem  
Enter Export Password:  
Verifying - Enter Export Password:
```

The export password will be used on IIS to import the server certificate.

7.4.7. Deploy Server certificate in IIS

The pfx file generated will be imported through the import link.

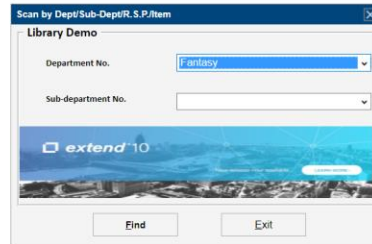
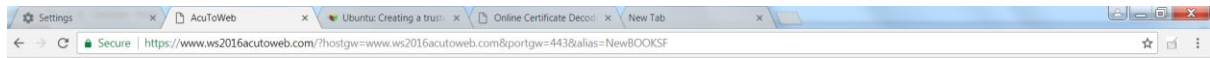
The user has to enter the password used during the generation of the .pfx file.



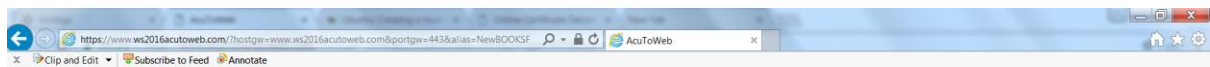
After the new certificate should be associated with the https configuration.
The process was already described in this document.

7.4.8. Tests

With Google chrome (No Warnings, No Issues).



With IE (No warnings, No Issues)



8. Conclusion

This is a work document. This document is not ready yet to be deliver as this to a customer.

9. Annexes

8.1. openssl configuration file : ws2016acutoweb.cnf

The file ws2016acutoweb.cnf was created on RedHat as a copy of /etc/pki/tls/openssl.cnf. I checked and modified **some lines** according to the documentation found in the following document:

<https://fabianlee.org/2018/02/17/ubuntu-creating-a-trusted-ca-and-san-certificate-using-openssl-on-ubuntu/>

All lines checked and modified are with a yellow background.

During my test, I forget to uncomment this:

nsCertType = sslCA, emailCA

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME               = .
RANDFILE           = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:
#oid_file           = $ENV::HOME/.oid
oid_section         = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions        =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca', 'req' and 'ts'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

# Policies used by the TSA examples.
tsa_policy1 = 1.2.3.4.1
tsa_policy2 = 1.2.3.4.5.6
tsa_policy3 = 1.2.3.4.5.7
```

```
#####
[ ca ]
default_ca      = CA_default          # The default ca section

#####
[ CA_default ]

dir              = /etc/pki/CA          # Where everything is kept
certs            = $dir/certs           # Where the issued certs are kept
crl_dir          = $dir/crl             # Where the issued crl are kept
database         = $dir/index.txt      # database index file.
#unique_subject  = no                  # Set to 'no' to allow creation of
                                      # several ctificates with same subject.
new_certs_dir    = $dir/newcerts        # default place for new certs.

certificate      = $dir/cacert.pem      # The CA certificate
serial           = $dir/serial          # The current serial number
crlnumber        = $dir/crlnumber       # the current crl number
                                      # must be commented out to leave a V1 CRL
crl              = $dir/crl.pem         # The current CRL
private_key      = $dir/private/cakey.pem # The private key
RANDFILE         = $dir/private/.rand   # private random number file

x509_extensions = usr_cert              # The extensions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default            # Certificate field options

# Extension copying option: use with caution.
# copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
# crl_extensions = crl_ext

default_days     = 365                  # how long to certify for
default_crl_days = 30                  # how long before next CRL
default_md       = default              # use public key default MD
preserve = no     # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy           = policy_match

# For the CA policy
[ policy_match ]
countryName      = match
stateOrProvinceName = match
organizationName = match
organizationalUnitName = optional
commonName       = supplied
emailAddress      = optional
```



```

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName          = optional
stateOrProvinceName  = optional
localityName         = optional
organizationName      = optional
organizationalUnitName = optional
commonName           = supplied
emailAddress          = optional

#####

[ req ]
default_bits          = 2048
default_md             = sha1
default_keyfile       = privkey.pem
distinguished_name     = req_distinguished_name
attributes            = req_attributes
x509_extensions       = v3_ca # The extensions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix    : PrintableString, BMPString (PKIX recommendation before 2004)
# utf8only: only UTF8Strings (PKIX recommendation after 2004).
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: ancient versions of Netscape crash on BMPStrings or UTF8Strings.
string_mask = utf8only

req_extensions = v3_req # The extensions to add to a certificate request

[ req_distinguished_name ]
countryName          = Country Name (2 letter code)
countryName_default  = XX
countryName_min      = 2
countryName_max      = 2

stateOrProvinceName  = State or Province Name (full name)
#stateOrProvinceName_default = Default Province

localityName         = Locality Name (eg, city)
localityName_default = Default City

0.organizationName    = Organization Name (eg, company)
0.organizationName_default = Default Company Ltd

# we can do this but it is not needed normally :-)
#1.organizationName    = Second Organization Name (eg, company)
#1.organizationName_default = World Wide Web Pty Ltd

organizationalUnitName = Organizational Unit Name (eg, section)
#organizationalUnitName_default =

```

commonName = Common Name (eg, your name or your server\'s hostname)
commonName_max = 64

emailAddress = Email Address
emailAddress_max = 64

SET-ex3 = SET extension number 3

[*req_attributes*]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20

unstructuredName = An optional company name

[*usr_cert*]

These extensions are added when 'ca' signs a request.

*# This goes against PKIX guidelines but some CAs do it and some software
requires this to avoid interpreting an end user certificate as a CA.*

basicConstraints=CA:FALSE

*# Here are some examples of the usage of nsCertType. If it is omitted
the certificate can be used for anything *except* object signing.*

This is OK for an SSL server.

nsCertType = server

For an object signing certificate this would be used.

nsCertType = objsign

For normal client use this is typical

nsCertType = client, email

and for everything including object signing:

nsCertType = client, email, objsign

This is typical in keyUsage for a client certificate.

keyUsage = nonRepudiation, digitalSignature, keyEncipherment

This will be displayed in Netscape's comment listbox.

nsComment = "OpenSSL Generated Certificate"

PKIX recommendations harmless if included in all certificates.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid,issuer

This stuff is for subjectAltName and issuerAltname.

Import the email address.

subjectAltName=email:copy

An alternative to produce certificates that aren't

deprecated according to PKIX.

subjectAltName=email:move

```

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl           = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This is required for TSA certificates.
# extendedKeyUsage = critical,timeStamping

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = www.ws2016acutoweb.com
DNS.2 = *.ws2016acutoweb.com

[ v3_ca ]

# Extensions for a typical CA

# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = critical, CA:true, pathlen:3

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
keyUsage = critical, cRLSign, keyCertSign

# Some might want this also
nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!

```

```

# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always

[ proxy_cert_ext ]
# These extensions should be added when creating a proxy certificate

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl        = http://www.domain.dom/ca-crl.pem

```

```

#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

#####
[ tsa ]

default_tsa = tsa_config1 # the default TSA section

[ tsa_config1 ]

# These are used by the TSA reply generation only.
dir           = ./demoCA           # TSA root directory
serial        = $dir/tsaserial     # The current serial number (mandatory)
crypto_device = builtin            # OpenSSL engine to use for signing
signer_cert   = $dir/tsacert.pem    # The TSA signing certificate
                                     # (optional)
certs         = $dir/cacert.pem     # Certificate chain to include in reply
                                     # (optional)
signer_key    = $dir/private/tsakey.pem # The TSA private key (optional)

default_policy = tsa_policy1        # Policy if request did not specify it
                                     # (optional)
other_policies = tsa_policy2, tsa_policy3 # acceptable policies (optional)
digests       = md5, sha1           # Acceptable message digests (mandatory)
accuracy= secs:1, millisecs:500, microsecs:100 # (optional)
clock_precision_digits = 0 # number of digits after dot. (optional)
ordering      = yes                # Is ordering defined for timestamps?
                                     # (optional, default: no)
tsa_name      = yes                # Must the TSA name be included in the reply?
                                     # (optional, default: no)
ess_cert_id_chain = no            # Must the ESS cert id chain be included?
                                     # (optional, default: no)

```

8.2. Script `create_ca_certificate.sh` to create a CA Root Certificate

```

$ cat create_ca_certificate.sh
# First generate the private/public RSA key pair
# This encodes the key file using an passphrase based on AES256.
rm -rf ca.key.pem ca.pem ca.srl
openssl genrsa -aes256 -out ca.key.pem 2048
chmod 400 ca.key.pem
# create the self-signed root CA certificate
openssl req -new -x509 -subj "/" -CN=rochCA -extensions v3_ca -days 3650 -key ca.k
ey.pem -sha256 -out ca.pem -config ws2016acutoweb.cnf
# verify this root CA certificate
openssl x509 -in ca.pem -text -noout

```


8.1. Script `create_server_certificate.sh` to create the server certificate

```
$ cat create_server_certificate.sh
# First generate the private/public RSA key pair
openssl genrsa -out ws2016acutoweb.key.pem 2048

# Create the server cert signing request
openssl req -subj "/CN=www.ws2016acutoweb.com" -extensions v3_req -sha256 -new -
key ws2016acutoweb.key.pem -out ws2016acutoweb.csr

# Generate the server certificate using the: server signing request,
# the CA signing key, and CA cert.
openssl x509 -req -extensions v3_req -days 3650 -sha256 -in ws2016acutoweb.csr -
CA ca.pem -CAkey ca.key.pem -CAcreateserial -out ws2016acutoweb.crt -extfile ws2
016acutoweb.cnf
openssl x509 -in ws2016acutoweb.crt -text -noout

# Generate a .pfx file for Windows IIS
openssl pkcs12 -export -out ws2016acutoweb.pfx -inkey ws2016acutoweb.key.pem -in
ws2016acutoweb.crt -certfile ca.pem
```