Vanilla CAs How To

Shirley Crompton, UKRI-STFC

Purpose

The vanilla CAs web application provides basic certification services to support the Year 2 demonstration of the mF2C project. It should be noted that the application is not intended for production use.

The application is written in JAVA (v. 1.8.0_162-b12) using the Jersey library (v. 2.27) and is deployed as an REST application on a Tomcat 8 (v 8.5.35) server running in a Docker (v. 18.03.0-ce) container.

The first part of the document provides information on how to build and deploy the application on the Engineering VM (running CENTOS 7.3). The second part gives examples on how to use the application. Please note that I use a Windows machine, please adjust the commands and path elements as appropriate to suit your environment.

Installation Notes

The user is assumed to have the appropriate privileges to access and deploy artifacts on the Engineering machine (213.205.14.13). The installation steps are as follows:

1 Create the docker build folder and assemble the artefacts:

1.1 Create the folder 'certauths', a subfolder each for 'tomcat' and 'credentials'. (The 'certauths' folder is currently located in /usr/local/libexec/catShirley/)

1.2 Copy the credential files (*.key, *.pem, tomcat*) from ownclouds (mF2C\Working Folders\WP5 PoC integration\CA\CA credentials\) to the local credentials folder. If you need a new tomcat server certificate, see the section on 'Generating a Tomcat Server Certificate' lower down.

1.3 Copy the tomcat server.xml and catalina.sh files from ownclouds (mF2C\Working Folders\WP5 PoC integration\CA\tomcat\) to the local tomcat folder

1.4 Clone the certauth application from the mf2c git (https://github.com/mF2C/certauth.git):
    [certauths] clone https://github.com/mF2C/certauth.git ./certauth

1.5 Build the Docker image, provide the correct tag version (e.g. mf2c/certauths:v1) as required:
    [certauths] docker build -f ./certauth/Dockerfile -–rm –t mf2c/certauths:v1.02 .

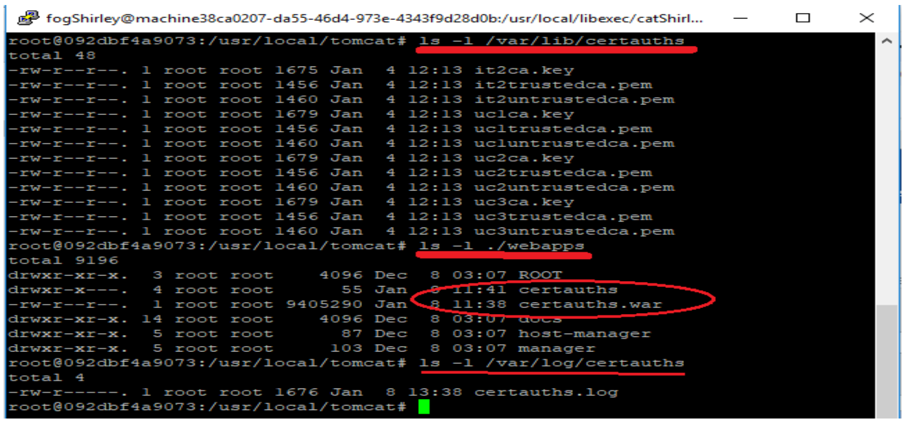
1.6 Run the image and create a container with the specific containerName, we want to map the container's port 8443 to the host's port 54443
    ~~[certauths] docker run –t –p 8080:8080 –-name <containerName> mf2c/certauths:v1~~
    [certauths] docker run –t –p 54443:8443 –-name <containerName> mf2c/certauths:v1.02

The Tomcat console output should indicate that the certauths.war has been deployed.

1.7 Cntr c to exit the running container. You could check the service endpoints using the methods listed in the Usage Notes section.

1.8 Verify the installation in the container via an interactive session. Start the session:
    [certauths] docker exec –it <container name> /bin/bash

1.8.1 Check that the credential files have been copied etc. Note that the command can only be used against a running container and that the log file will only be created after a call has been made to the endpoint.



```
fogShirley@machine38ca0207-da55-46d4-973e-4343f9d28d0b:/usr/local/libexec/catShirl...
root@092dbf4a9073:/usr/local/tomcat# ls -l /var/lib/certauths
total 48
-rw-r--r--. 1 root root 1675 Jan  4 12:13 it2ca.key
-rw-r--r--. 1 root root 1456 Jan  4 12:13 it2trustedca.pem
-rw-r--r--. 1 root root 1460 Jan  4 12:13 it2untrustedca.pem
-rw-r--r--. 1 root root 1679 Jan  4 12:13 uc1ca.key
-rw-r--r--. 1 root root 1456 Jan  4 12:13 uc1trustedca.pem
-rw-r--r--. 1 root root 1460 Jan  4 12:13 uc1untrustedca.pem
-rw-r--r--. 1 root root 1679 Jan  4 12:13 uc2ca.key
-rw-r--r--. 1 root root 1456 Jan  4 12:13 uc2trustedca.pem
-rw-r--r--. 1 root root 1460 Jan  4 12:13 uc2untrustedca.pem
-rw-r--r--. 1 root root 1679 Jan  4 12:13 uc3ca.key
-rw-r--r--. 1 root root 1456 Jan  4 12:13 uc3trustedca.pem
-rw-r--r--. 1 root root 1460 Jan  4 12:13 uc3untrustedca.pem
root@092dbf4a9073:/usr/local/tomcat# ls -l ./webapps
total 9196
drwxr-xr-x.  3 root root    4096 Dec  8 03:07 ROOT
drwxr-x---.  4 root root      55 Jan  8 11:41 certauths
-rw-r--r--.  1 root root 9405290 Jan  8 11:38 certauths.war
drwxr-xr-x. 14 root root    4096 Dec  8 03:07 docs
drwxr-xr-x.  5 root root      87 Dec  8 03:07 host-manager
drwxr-xr-x.  5 root root     103 Dec  8 03:07 manager
root@092dbf4a9073:/usr/local/tomcat# ls -l /var/log/certauths
total 4
-rw-r-----. 1 root root 1676 Jan  8 13:38 certauths.log
root@092dbf4a9073:/usr/local/tomcat#
```

You could also use curl commands against the localhost name to test the service (see examples in the Usage notes section).

1.8.2 Exit the interactive session by typing 'exit'

1.9 Next, we want to configure the firewall, we need to obtain the IP address of the container. First, get the container ID by listing the currently running containers, use the container name to locate the correct container:

[certauths] docker ps

1.10 Check the local IP for the container

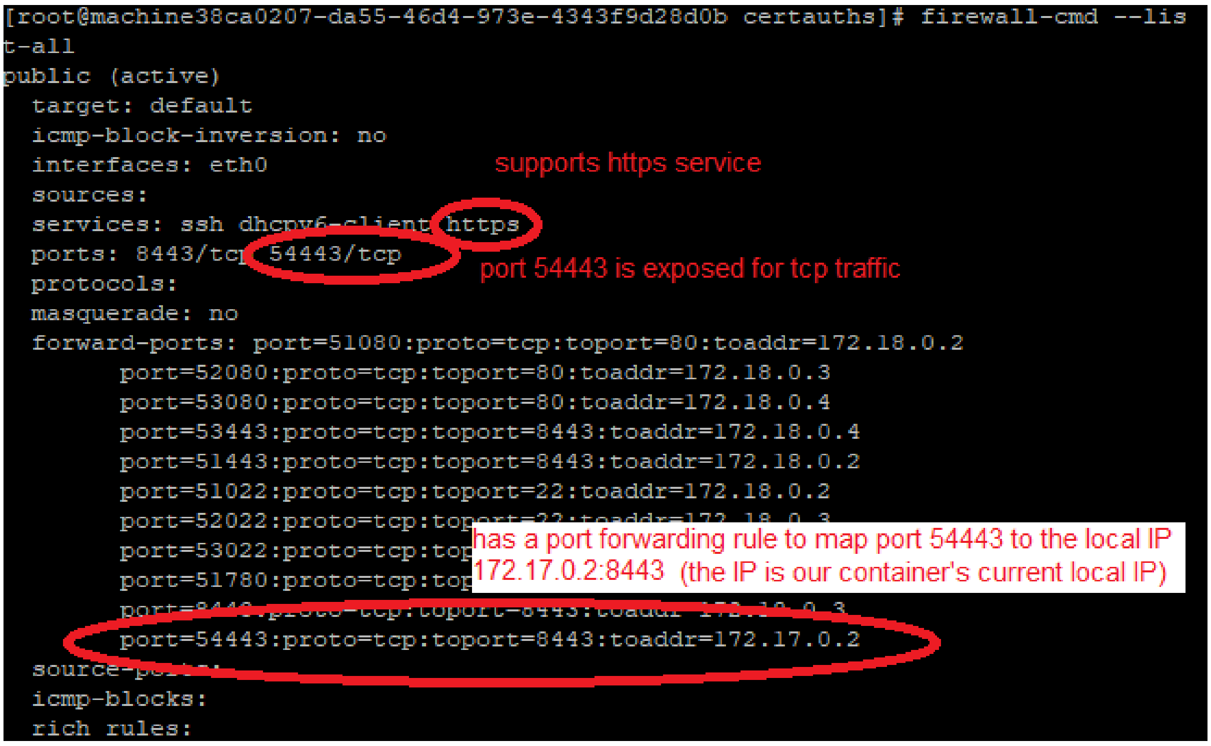
[certauths] docker inspect -f ip='{{.NetworkSettings.IPAddress}}' <containerID>

1.11 You may also like to verify the port mapping:

[certauths] docker inspect –f ports='{{.NetworkSettings.Ports}}' <containerID>

1.12 Check the firewall using firewall-cmd to see if it has the appropriate configuration:

[certauths] firewall-cmd –list-all



```
[root@machine38ca0207-da55-46d4-973e-4343f9d28d0b certauths]# firewall-cmd --lis
t-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: eth0
  sources:
  services: ssh dhcpv6-client https
  ports: 8443/tcp 54443/tcp
  protocols:
  masquerade: no
  forward-ports: port=51080:proto=tcp:toport=80:toaddr=172.18.0.2
        port=52080:proto=tcp:toport=80:toaddr=172.18.0.3
        port=53080:proto=tcp:toport=80:toaddr=172.18.0.4
        port=53443:proto=tcp:toport=8443:toaddr=172.18.0.4
        port=51443:proto=tcp:toport=8443:toaddr=172.18.0.2
        port=51022:proto=tcp:toport=22:toaddr=172.18.0.2
        port=52022:proto=tcp:toport=22:toaddr=172.18.0.3
        port=53022:proto=tcp:top
        port=51780:proto=tcp:top
        port=8443:proto=tcp:toport=8443:toaddr=172.18.0.3
        port=54443:proto=tcp:toport=8443:toaddr=172.17.0.2
  source-ports:
  icmp-blocks:
  rich rules:
```

supports https service
port 54443 is exposed for tcp traffic
has a port forwarding rule to map port 54443 to the local IP 172.17.0.2:8443 (the IP is our container's current local IP)

Check that it has the above settings. If not, update the configuration to match. See https://www.thegeekdiary.com/5-useful-examples-of-firewall-cmd-command/

1.13 Upload the image to Docker hub:

1.13.1 Login to Docker hub and follow the on-screen instructions:

```
[certauths] docker login
```

1.13.2 Stop the running container

```
[certauths] docker stop <container id/container name>
```

1.13.3 Commit

```
[certauths] docker commit –m “my commit message” –a “Author Name” <containername>
<imagename>
```

1.13.4 Push to Docker hub

```
[certauths] docker push <image-name/version>
```

1.13.5 Restart the container

```
[certauths] docker start <container id/container name>
```

Generating a Tomcat Server Certificate

Use the IT2trustedca service to issue a certificate for your server, provide the IP of your server as input (see the Usage Notes on how to do this). Then we need to use OpenSSL to generate a self-contained certificate in PKCS12 format:

```
<OpenSSL> pkcs12 -export -in <x509.pem> -inkey <RSAprivate.key> -out tomcat.p12 -name tomcat -CAfile <path\it2TrustedCA.pem> -caname root –chain
```

Then we use JAVA keytool to convert the tomcat.p12 file into a JAVA keystore. Enter a passphase when prompted, using the same one throughout (this passphrase is used in the server.xml connector configuration):

```
"%JAVA_HOME%"\bin\keytool -importkeystore -v -srckeystore <path\tomcat.p12> -srcstoretype PKCS12 -destkeystore <\path\tomcat.keystore> -deststoretype pkcs12
```

Next, check the keystore, enter the passphrase you used before as store pass:

```
"%JAVA_HOME%"\bin\keytool -list -keystore <path\tomcat.keystore>
```

Usage Notes

The webapp context root of the Tomcat server on the Engineering machine is:

~~http://213.205.14.13:8080/ or http://it1demo.mf2c-project.eu:8080/~~
https://213.205.14.13:54443/ or https://it1demo.mf2c-project.eu:54443/

The server has been updated to accept https connection only. Please use https protocol and the updated context root for the usage examples below.

The vanilla CA web application provides four sets of 2 endpoints with each set supporting a specific IT-2 demo scenarios:

1 ‘Hello World’ IT-2 demo:
1.1 https://213.205.14.13:54443/certauths/rest/it2trustedca
1.2 https://213.205.14.13:54443/certauths/rest/it2untrustedca
2 UC1 Smart City demo:
2.1 https://213.205.14.13:54443/certauths/rest/uc1trustedca
2.2 https://213.205.14.13:54443/certauths/rest/uc1untrustedca

3 UC2 Smart Boat demo:

3.1 https://213.205.14.13:54443/certauths/rest/uc2trustedca

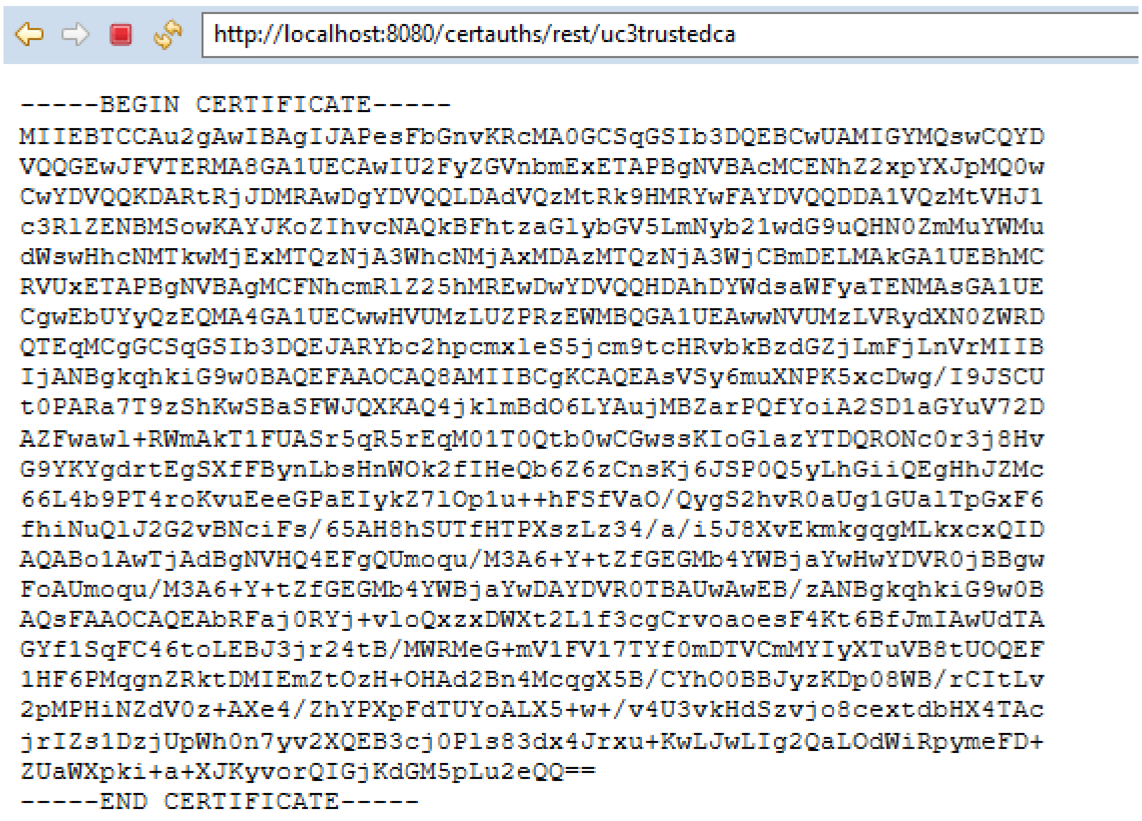
3.2 https://213.205.14.13:54443/certauths/rest/uc2untrustedca

4 UC3 Smart Airport demo:

4.1 https://213.205.14.13:54443/certauths/rest/uc3trustedca

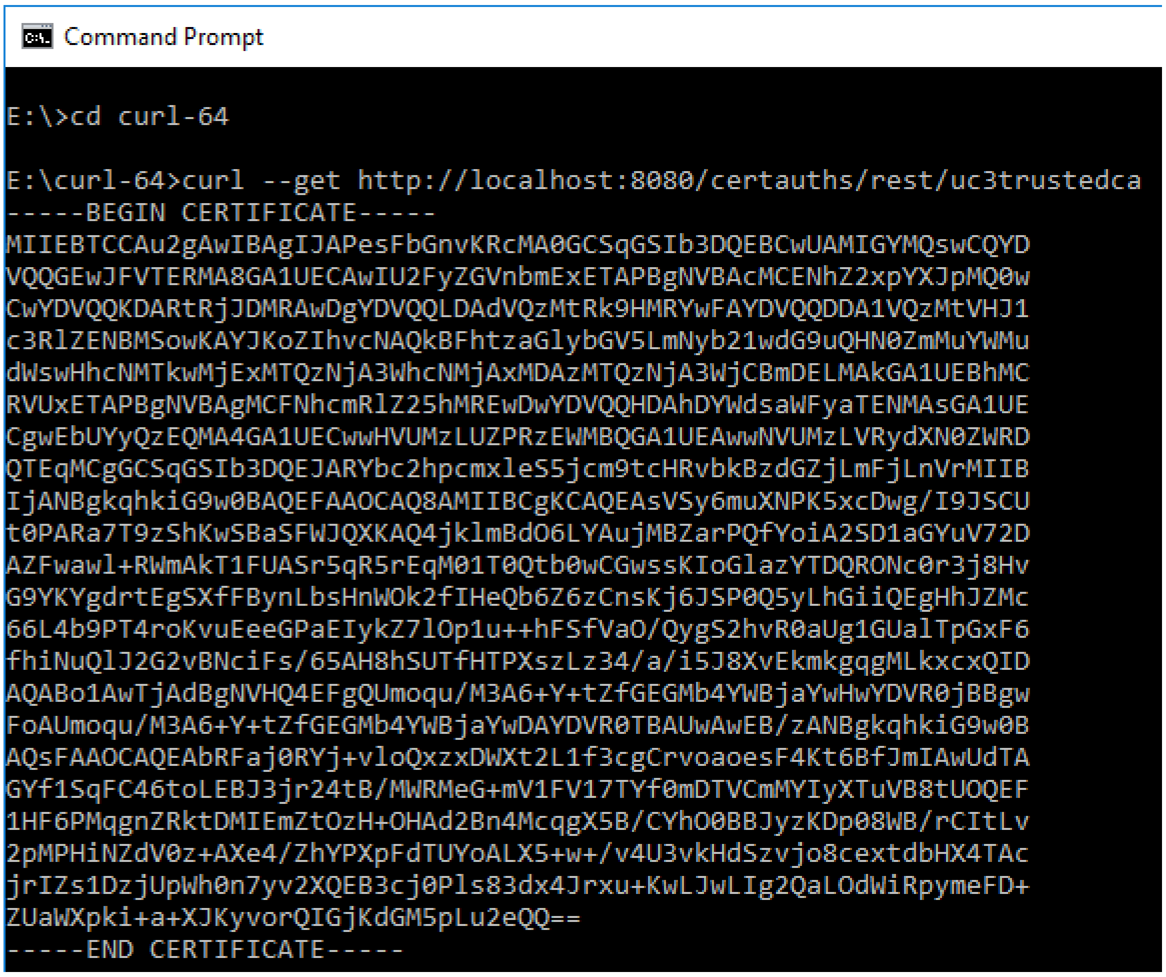
4.2 https://213.205.14.13:54443/certauths/rest/uc3untrustedca

The endpoints accepts both HTTP GET and POST methods. The GET method returns the CA's X509 certificate:



```
-----BEGIN CERTIFICATE-----
MIIEBTCCAu2gAwIBAgIJAPesFbGnvKRcMA0GCSqGSIb3DQEBCwUAMIGYMQswCQYD
VQQGEwJFVTERMA8GA1UECAwIU2FyZGVnbmExETAPBgNVBAcMCENhZ2xpYXJpMQ0w
CwYDVQQKDARtRjJDMRAwDgYDVQQLDAdVQzMtRk9HMRYwFAYDVQQDDA1VQzMtVHJ1
c3R1ZENBMSowKAYJKoZIhvcNAQkBFhtzaGlybGV5LmNyb21wdG9uQHN0ZmMuYWMu
dWswHhcNMTkwMjExMTQzNjA3WhcNMjAxMDAzMTQzNjA3WjCBmDELMAkGA1UEBhMC
RVUxETAPBgNVBAgMCFNhcmR1Z25hMREwDwYDVQQHDAhDYWdsaWFyaTENMAsGA1UE
CgwEbUYyQzEQMA4GA1UECwwHVUMzLUZPRzEWMBQGA1UEAwwNVUMzLVRydXN0ZWRD
QTEqMCgGCSqGSIb3DQEJARYbc2hpcmxleS5jcm9tcHRvbkBzdGZjLmFjLnVrMIIB
IjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsVSy6muXNPK5xcDwg/I9JSCU
t0PARa7T9zShKwSBaSFWJQXKAQ4jklmBdO6LYAujMBZarPQfYoiA2SD1aGYuV72D
AZFwawl+RWmAkT1FUASr5qR5rEqM01T0Qtb0wCGwssKIoGlazYTDQRONc0r3j8Hv
G9YKYgdrtEgSXfFBynLbsHnWOk2fIHeQb6Z6zCnsKj6JSP0Q5yLhGiiQEgHhJZMc
66L4b9PT4roKvuEeeGPaEIykZ71Op1u++hFSfVaO/QygS2hvR0aUg1GUalTpGxF6
fhiNuQ1J2G2vBNciFs/65AH8hSUTfHTPXszLz34/a/i5J8XvEkmkgqgMLkxcxQID
AQABo1AwTjAdBgNVHQ4EFgQUmoqu/M3A6+Y+tZfGEGMb4YWBjaYwHwYDVR0jBBgw
FoAUmoqu/M3A6+Y+tZfGEGMb4YWBjaYwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0B
AQsFAAOCAQEAbRFaj0RYj+vloQxzxDWXt2L1f3cgCrvoaoesF4Kt6BfJmIAwUdTA
GYf1SqFC46toLEBJ3jr24tB/MWRMeG+mV1FV17TYf0mDTVCmMYIyXTuVB8tUOQEF
1HF6PMqgnZRktDMIEmZtOzH+OHAd2Bn4McqgX5B/CYhO0BBJyzKDp08WB/rCItLv
2pMPHiNZdV0z+AXe4/ZhYPXpFdTUYoALX5+w+/v4U3vkHdSzvjo8cextdbHX4TAc
jrIZs1DzjUpWh0n7yv2XQEB3cj0Pls83dx4Jrxu+KwLJwLIg2QaLOdWiRpymeFD+
ZUaWXpki+a+XJKyvorQIGjKdGM5pLu2eQQ==
-----END CERTIFICATE-----
```

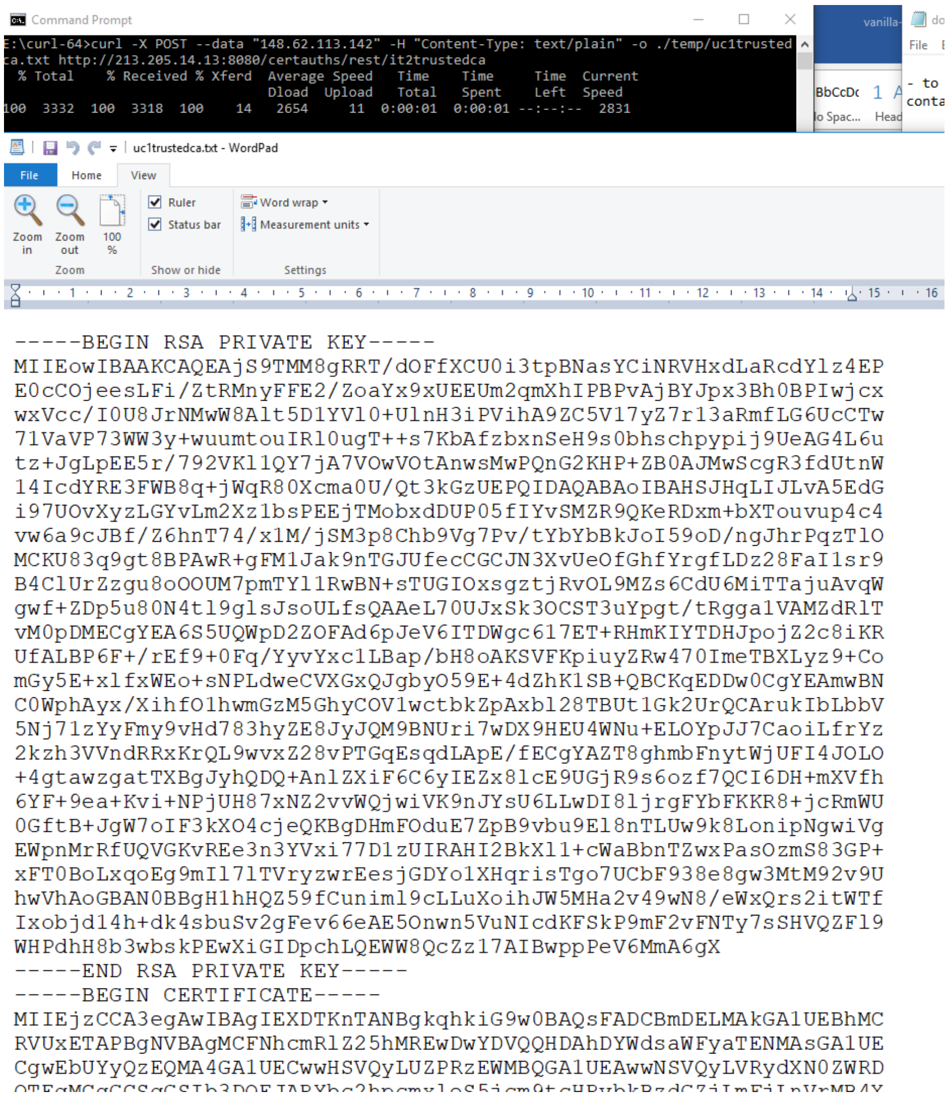
Alternatively, we could use curl to 'GET' the endpoint:


Command Prompt

```
E:\>cd curl-64

E:\curl-64>curl --get http://localhost:8080/certauths/rest/uc3trustedca
-----BEGIN CERTIFICATE-----
MIIEBTCCAu2gAwIBAgIJAPesFbGnvKRcMA0GCSqGSIb3DQEBCwUAMIGYMQswCQYD
VQQGEwJFVTERMA8GA1UECAwIU2FyZGVnbmExETAPBgNVBAcMCENhZ2xpYXJpMQ0w
CwYDVQQKDARtRjJDMRAwDgYDVQQLDAdVQzMtRk9HMRYwFAYDVQQDDA1VQzMtVHJ1
c3RlZENBMSowKAYJKoZIhvcNAQkBFhtzaGlybGV5LmNyb21wdG9uQHN0ZmMuYWMu
dWswHhcNMTkwMjExMTQzNjA3WhcNMjAxMDAzMTQzNjA3WjCBmDELMAkGA1UEBhMC
RVUxETAPBgNVBAgMCFNhcmRlZ25hMREwDwYDVQQHDAhDYWdsaWFyaTENMAsGA1UE
CgwEbUYyQzEQMA4GA1UECwwHVUMzLUZPRzEWMBQGA1UEAwwNVUMzLVRydXN0ZWRD
QTEqMCgGCSqGSIb3DQEJARYbc2hpcmxleS5jcm9tcHRvbkBzdGZjLmFjLnVrMIIB
IjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsVSy6muXNPK5xcDwg/I9JSCU
t0PARa7T9zShKwSBaSFWJQXKAQ4jklmBdO6LYAujMBZarPQfYoiA2SD1aGYuV72D
AZFwawl+RWmAkT1FUASr5qR5rEqM01T0Qtb0wCGwssKIoGlazYTDQRONc0r3j8Hv
G9YKYgdrtEgSXfFBynLbsHnWOk2fIHeQb6Z6zCnsKj6JSP0Q5yLhGiiQEgHhJZMc
66L4b9PT4roKvuEeeGPaEIykZ7lOp1u++hFSfVaO/QygS2hvR0aUg1GUalTpGxF6
fhiNuQlJ2G2vBNciFs/65AH8hSUTfHTPXszLz34/a/i5J8XvEkmkgqgMLkxcxQID
AQABo1AwTjAdBgNVHQ4EFgQUmoqu/M3A6+Y+tZfGEGMb4YWBjaYwHwYDVR0jBBgw
FoAUmoqu/M3A6+Y+tZfGEGMb4YWBjaYwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0B
AQsFAAOCAQEAbRFaj0RYj+vloQxzxDWXt2L1f3cgCrvoaoesF4Kt6BfJmIAwUdTA
GYf1SqFC46toLEBJ3jr24tB/MWRMeG+mV1FV17TYf0mDTVCmMYIyXTuVB8tUOQEF
1HF6PMqgnZRktDMIEmZtOzH+OHAd2Bn4McqgX5B/CYhO0BBJyzKDp08WB/rCItLv
2pMPHiNZdV0z+AXe4/ZhYPXpFdTUYoALX5+w+/v4U3vkHdSzvjo8cextdbHX4TAc
jrIZs1DzjUpWh0n7yv2XQEB3cj0Pls83dx4Jrxu+KwLJwLIg2QaLOdWiRpymeFD+
ZUaWXpki+a+XJKyvorQIGjKdGM5pLu2eQQ==
-----END CERTIFICATE-----
```

Use the HTTP POST method to generate the required credential/s. The trusted CAs issue X.509 certificates for trusted mF2C infrastructure components, e.g. CAU. The client should provide a max 64 chars length common-name in plain text to the appropriate trusted CA endpoint which should return the X.509 certificate and the associate private key in pem format. Users should obtain the credentials out-of-band as they are expected to be distributed with the components.
Figure 1 gives an example on obtaining the credentials using curl with the endpoint http://213.205.14.13:8080/certauths/rest/it2trustedca and the common-name 148.62.113.142. Note that the private key and X.509 certificates are concatenated together and users should use a text editor to break them into two files.

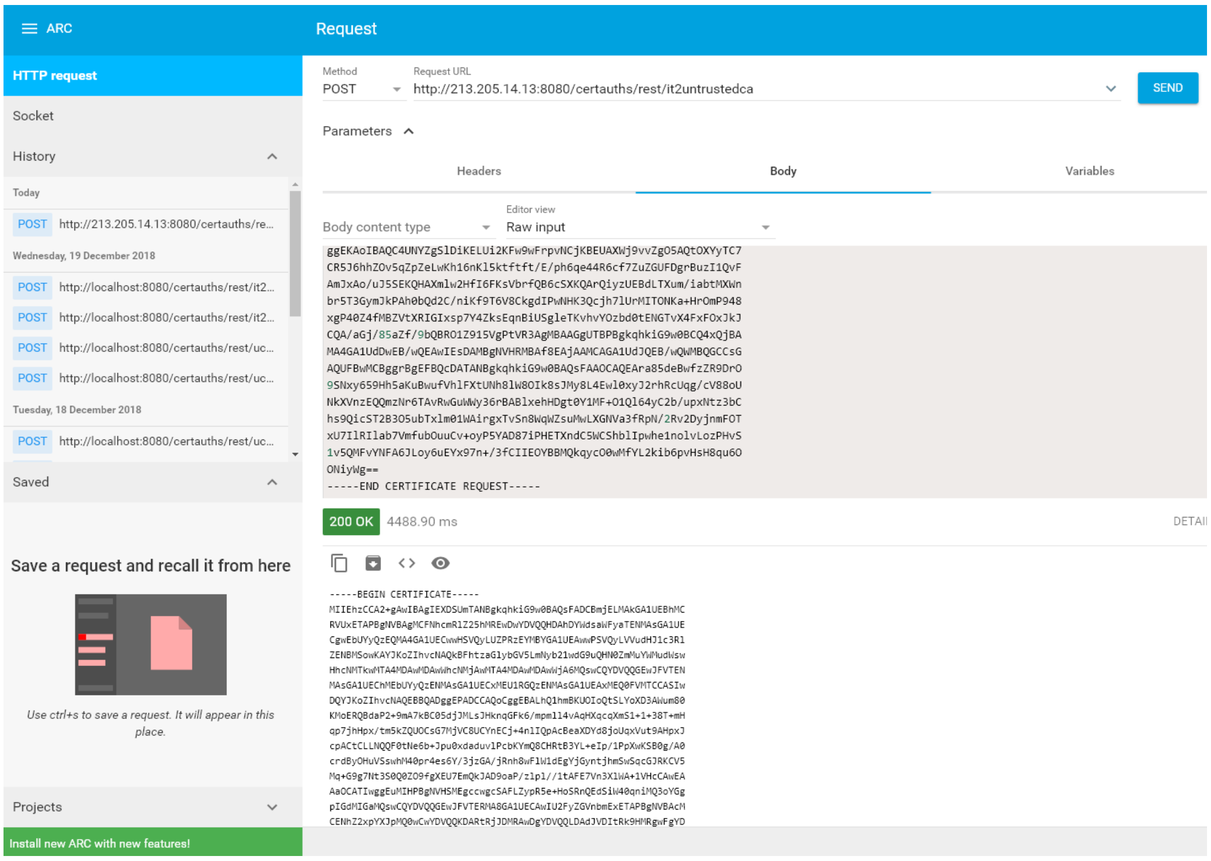


```
E:\curl-64>curl -X POST --data "148.62.113.142" -H "Content-Type: text/plain" -o ./temp/uc1trusted
ca.txt http://213.205.14.13:8080/certauths/rest/it2trustedca
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3332  100  3318  100    14   2654     11  0:00:01  0:00:01 --:--:--  2831
```

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAjS9TMM8gRRT/dOFfXCU0i3tpBNasYCiNRVHxdLaRcdYlz4EP
E0cCOjeesLFi/ZtRMnyFFE2/ZoaYx9xUEEUm2qmXhIPBPvAjBYJpx3Bh0BPIwjcx
wxVcc/I0U8JrNMwW8Alt5D1YVl0+UlnH3iPVihA9ZC5V17yZ7r13aRmfLG6UcCTw
71VaVP73WW3y+wuumtouIRl0ugT++s7KbAfzbxnSeH9s0bhschpypij9UeAG4L6u
tz+JgLpEE5r/792VKl1QY7jA7VOwVOtAnwsMwPQnG2KHP+ZB0AJMwScgR3fdUtnW
14IcdYRE3FWB8q+jWqR80Xcma0U/Qt3kGzUEPQIDAQABAoIBAHSJHqLIJLvA5EdG
i97UOvXyzLGYvLm2Xz1bsPEEjTMobxdDUP05fIYvSMZR9QKeRDxm+bXTouvup4c4
vw6a9cJBf/Z6hnT74/x1M/jSM3p8Chb9Vg7Pv/tYbYbBkJoI59oD/ngJhrPqzTlO
MCKU83q9gt8BPAwR+gFM1Jak9nTGJUfecCGCJN3XvUeOfGhfYrgfLDz28FaIlsr9
B4ClUrZzgu8oOOUM7pmTYl1RwBN+sTUGIOxsgztjRvOL9MZs6CdU6MiTTajuAvqW
gwf+ZDp5u80N4tl9glsJsoULfsQAAeL70UJxSk3OCST3uYpgt/tRgga1VAMZdRlT
vM0pDMECgYEA6S5UQWpD2ZOFAd6pJeV6ITDWgc617ET+RHmKIYTDHJpojZ2c8iKR
UfALBP6F+/rEf9+0Fq/YyvYxc1LBap/bH8oAKSVFKpiuyZRw470ImeTBXLyz9+Co
mGy5E+xlfxWEo+sNPLdweCVXGxQJgbyO59E+4dZhK1SB+QBCKqEDDw0CgYEAmwBN
C0WphAyx/XihfO1hwmGzM5GhyCOV1wctbkZpAxbl28TBUt1Gk2UrQCArukIbLbbV
5Nj71zYyFmy9vHd783hyZE8JyJQM9BNUri7wDX9HEU4WNu+ELOYpJJ7CaoiLfrYz
2kzh3VVndRRxKrQL9wvxZ28vPTGqEsqdLApE/fECgYAZT8ghmbFnytWjUFI4JOLO
+4gtawzgatTXBgJyhQDQ+AnlZXiF6C6yIEZx8lcE9UGjR9s6ozf7QCI6DH+mXVfh
6YF+9ea+Kvi+NPjUH87xNZ2vvWQjwiVK9nJYsU6LLwDI8ljrgFYbFKKR8+jcRmWU
0GftB+JgW7oIF3kXO4cjeQKBgDHmFOduE7ZpB9vbu9El8nTLUw9k8LonipNgwiVg
EWpnMrRfUQVGKvREe3n3YVxi77D1zUIRAHI2BkXl1+cWaBbnTZwxPasOzmS83GP+
xFT0BoLxqoEg9mIl7lTVryzwrEesjGDYo1XHqrisTgo7UCbF938e8gw3MtM92v9U
hwVhAoGBAN0BBgH1hHQZ59fCuniml9cLLuXoihJW5MHa2v49wN8/eWxQrs2itWTf
Ixobjdl4h+dk4sbuSv2gFev66eAE5Onwn5VuNIcdKFSkP9mF2vFNTy7sSHVQZFl9
WHPdhH8b3wbskPEwXiGIDpchLQEWW8QcZz17AIBwppPeV6MmA6gX
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIEjzCCA3egAwIBAgIEXDTKnTANBgkqhkiG9w0BAQsFADCBmDELMAkGA1UEBhMC
RVUxETAPBgNVBAgMCFNhcmRlZ25hMREwDwYDVQQHDAhDYWdsaWFyaTENMAsGA1UE
CgwEbUYyQzEQMA4GA1UECwwHSVQyLUZPRzEWMBQGA1UEAwwNSVQyLVRydXN0ZWRD
```

Figure 1 Using curl to obtain trusted credentials

Figure 2 shows an example using Google Advanced Rest Client (ARC) tool to obtain an X.509 certificate from an untrusted CA. The client is expected to send an PKCS10 Certification Request String in plain text. Untrusted CAs issue certificates to mF2C agents and the process takes place programmatically during the Discovery, Authentication (Security) and Categorization process flow (see D5.1).
The application provides a Junit (CertAuthRestServiceTest.java) which gives programmatic examples on calling the trusted and untrusted CAs. A Python client example (CAClientExample.py) is provided in the src/main/resources/ folder.


ARC
Request
HTTP request
Socket
History
Today
Wednesday, 19 December 2018
Tuesday, 18 December 2018
Saved
Save a request and recall it from here
Use ctrl+s to save a request. It will appear in this place.
Projects
Install new ARC with new features!
Parameters
Headers
Body
Variables
SEND
DETAI

POST http://213.205.14.13:8080/certauths/re...
POST http://localhost:8080/certauths/rest/it2...
POST http://localhost:8080/certauths/rest/it2...
POST http://localhost:8080/certauths/rest/uc...
POST http://localhost:8080/certauths/rest/uc...
POST http://localhost:8080/certauths/rest/uc...

Method: POST
Request URL: http://213.205.14.13:8080/certauths/rest/it2untrustedca

Body content type
Editor view: Raw input

```
ggEKAoIBAQC4UNYZgSlDiKELUi2KFw9wFrpvNCjKBEUAXWj9vvZgO5AQtOXYyTC7
CR5J6hhZOv5qZpZeLwKh16nKl5ktftft/E/ph6qe44R6cf7ZuZGUFDgrBuzI1QvF
AmJxAo/uJ5SEKQHAXmlw2HfI6FKsVbrfQB6cSXKQArQiyzUEBdLTXum/iabtMXWn
br5T3GymJkPAh0bQd2C/niKf9T6V8CkgdIPwNHK3Qcjh7lUrMITONKa+HrOmP948
xgP40Z4fMBZVtXRIGIxsp7Y4ZksEqnBiUSgleTKvhvYOzbd0tENGTvX4FxFOxJkJ
CQA/aGj/85aZf/9bQBRO1Z915VgPtVR3AgMBAAGgUTBPBgkqhkiG9w0BCQ4xQjBA
MA4GA1UdDwEB/wQEAwIEsDAMBgNVHRMBAf8EAjAAMCAGA1UdJQEB/wQWMBQGCCsG
AQUFBwMCBggrBgEFBQcDATANBgkqhkiG9w0BAQsFAAOCAQEAra85deBwfzZR9DrO
9SNxy659Hh5aKuBwufVhlFXtUNh8lW8OIk8sJMy8L4Ewl0xyJ2rhRcUqg/cV88oU
NkXVnzEQQmzNr6TAvRwGuWWy36rBABlxehHDgt0Y1MF+O1Ql64yC2b/upxNtz3bC
hs9QicST2B3O5ubTxlm01WAirgxTvSn8WqWZsuMwLXGNVa3fRpN/2Rv2DyjnmFOT
xU7IlRIlab7VmfubOuuCv+oyP5YAD87iPHETXndC5WCShblIpwhe1nolvLozPHvS
1v5QMFvYNFA6JLoy6uEYx97n+/3fCIIEOYBBMQkqycO0wMfYL2kib6pvHsH8qu6O
ONiyWg==
-----END CERTIFICATE REQUEST-----
```

200 OK 4488.90 ms

```
-----BEGIN CERTIFICATE-----
MIIEhzCCA2+gAwIBAgIEXDSUmTANBgkqhkiG9w0BAQsFADCBmjELMAkGA1UEBhMC
RVUxETAPBgNVBAgMCFNhcmRlZ25hMREwDwYDVQQHDAhDYWdsaWFyaTENMAsGA1UE
CgwEbUYyQzEQMA4GA1UECwwHSVQyLUZPRzEYMBYGA1UEAwwPSVQyLVVudHJ1c3Rl
ZENBMSowKAYJKoZIhvcNAQkBFhtzaGlybGV5LmNyb21wdG9uQHN0ZmMuYWMudWsw
HhcNMTkwMTA4MDAwMDAwWhcNMjAwMTA4MDAwMDAwWjA6MQswCQYDVQQGEwJFVTEN
MAsGA1UEChMEbUYyQzENMAsGA1UECxMEU1RGQzENMAsGA1UEAxMEQ0FVMTCCASIw
DQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALhQ1hmBKUOIoQtSLYoXD3AWum80
KMoERQBdaP2+9mA7kBC05djJMLsJHknqGFk6/mpml14vAqHXqcqXmS1+1+38T+mH
qp7jhHpx/tm5kZQUOCsG7MjVC8UCYnECj+4nlIQpAcBeaXDYd8joUqxVut9AHpxJ
cpACtCLLNQQF0tNe6b+Jpu0xdaduvlPcbKYmQ8CHRtB3YL+eIp/1PpXwKSB0g/A0
crdByOHuVSswhM40pr4es6Y/3jzGA/jRnh8wFlW1dEgYjGyntjhmSwSqcGJRKCV5
Mq+G9g7Nt3S0Q0ZO9fgXEU7EmQkJAD9oaP/zlpl//1tAFE7Vn3XlWA+1VHcCAwEA
AaOCATIwggEuMIHPBgNVHSMEgccwgcSAFLZypR5e+HoSRnQEdSiW40qniMQ3oYGg
pIGdMIGaMQswCQYDVQQGEwJFVTERMA8GA1UECAwIU2FyZGVnbmExETAPBgNVBAcM
CENhZ2xpYXJpMQ0wCwYDVQQKDARtRjJDMRAwDgYDVQQLDAdJVDItRk9HMRgwFgYD
```

Figure 2 ARC example on getting a certificate from the trusted CA