**WSFS CERTIFICATE CREATION GUIDE**

This document explains the process to create the following certificates which will be uploaded to the Dedicated Load Balancer in Anypoint platform

* Steps to generate CA signed certificates
  + Two-way SSL communication
  + One-way SSL communication
* Steps to generate a Self signed certificate

*Steps to generate CA Certificate for multi domain SAN*

**Prerequisites**

* OpenSSL must be available in the system before executing the below steps
* Details on OpenSSL can be found here <https://www.openssl.org/>

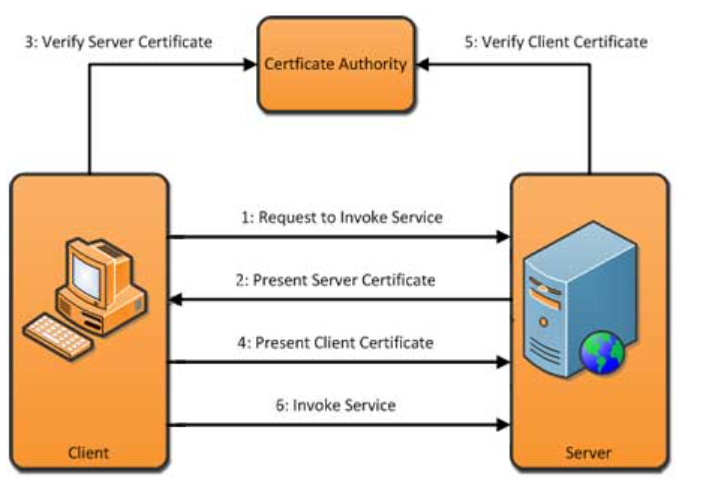
**How to install OpenSSL in Windows OS**

* Install GIT and run openssl command from Git Bash
* Git Installer can be found here: <https://git-scm.com/download/win>
* Git Bash icon looks like below



**CA Certificate Generation Steps (TWO-WAY SSL)**

* What is two-way SSL?
  + In two way SSL both the client and server have to present a certificate before the connection is established between the two. With two way SSL authentication, the Server not only authenticates itself to the client (which is the minimum requirement for certificate authentication), it also requires authentication from the requesting client. Clients are required to submit digital certificates issued by a trusted certificate authority.



**Step 1:**

* Create a file named openssl.conf
* Add below entries in the file

[req]

default\_bits = 2048

default\_md = sha256

prompt = no

distinguished\_name = dn

req\_extensions = req\_ext

[dn]

CN=api.wsfsbank.com

C=US

ST=Delaware

L=Wilmington

O=WSFS Bank

[req\_ext]

subjectAltName=@alt\_names

[alt\_names]

DNS.1= api.wsfsbank.com

DNS.2= www.api.wsfsbank.com

DNS.3= api-uat.wsfsbank.com

DNS.4= api-sandbox.wsfsbank.com

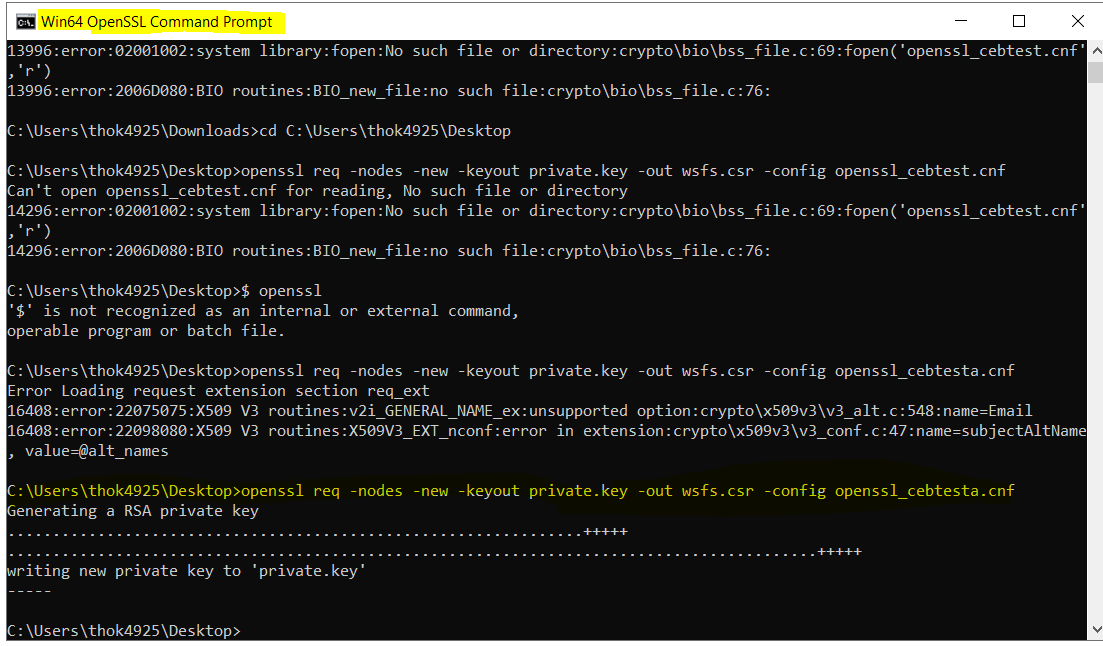
DNS.5= api-dev.wsfsbank.com

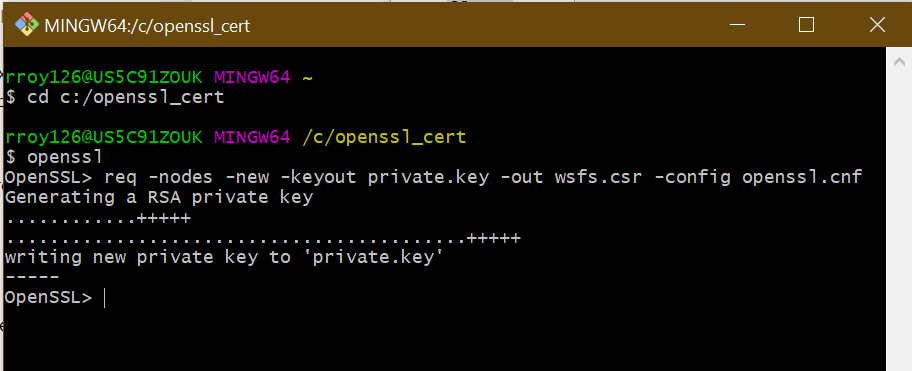
DNS.6= api-sit.wsfsbank.com

**Step2:**

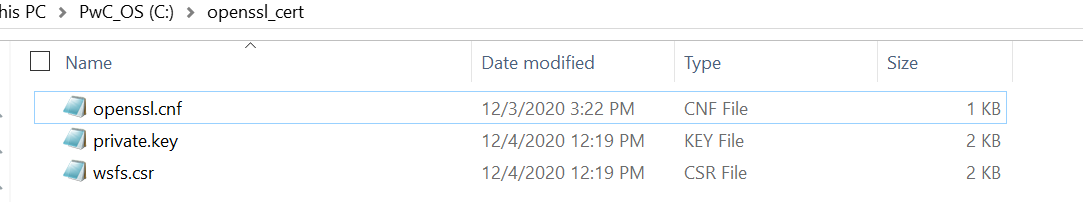
* Run below openssl command which will take the above **openssl.conf** configuration file and generate a private key named **private.key** and a csr named **wsfs.csr**

req -nodes -new -keyout private.key -out wsfs.csr -config openssl.cnf





* Verify that private.key and wsfs.csr file generated successfully.



**Step 3:**

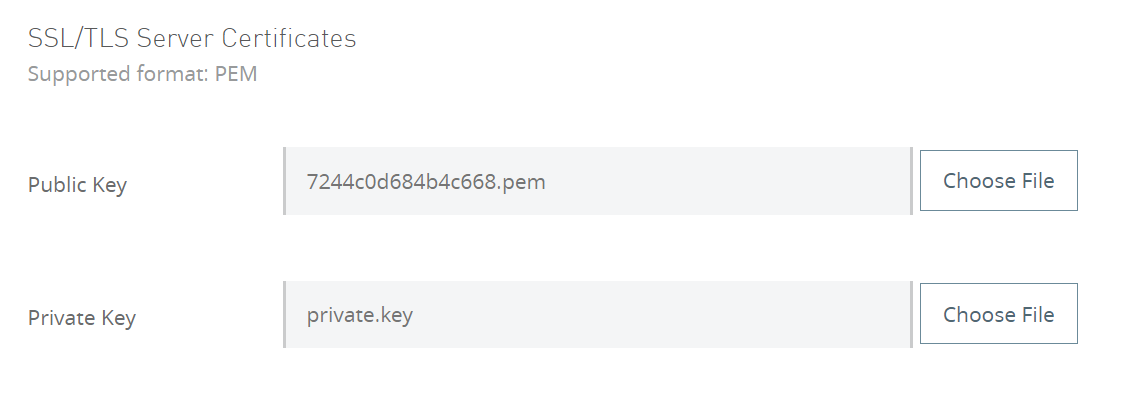
* Using this CSR (wsfs.csr), generate CA certified certificates.
* Please reach out to John Bailey from NTS team for CA certification and share this wsfs.csr file
  + You will receive the certificate in two formats from John B
    - .crt
    - .pem

These above 3 Steps will generate final private key and public key to be used in MuleSoft Cloudhub DLB

* + Please upload .pem format CA certificate received from John B / NTS team

as Public key in DLB

* + Please upload private.key as Private Key in DLB



**CA Certificate Generation Steps (ONE-WAY SSL)**

* What is one way SSL?
  + In one way SSL server presents the certificate to the client, but the client is not required to present a certificate to the server. The client must authenticate the server, but the server will accept any client into the  connection.

**Step 1:**

Create a file named openssl\_core.conf

Add below entries in the file.

[req]

default\_bits = 2048

default\_md = sha256

prompt = no

distinguished\_name = dn

req\_extensions = req\_ext

[dn]

CN=api-core.wsfsbank.com

C=US

ST=Delaware

L=Wilmington

O=WSFS Bank

[req\_ext]

subjectAltName=@alt\_names

[alt\_names]

DNS.1= api-core.wsfsbank.com

DNS.2= www.api-core.wsfsbank.com

DNS.3= api-core-uat.wsfsbank.com

DNS.4= api-core-sandbox.wsfsbank.com

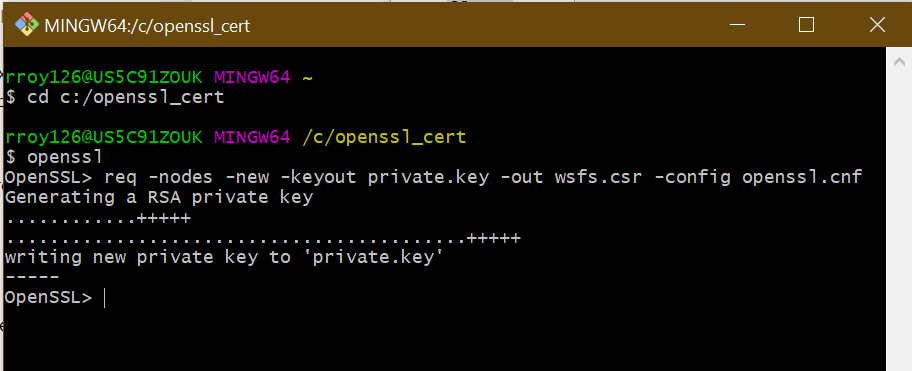
DNS.5= api-core-dev.wsfsbank.com

DNS.6= api-core-sit.wsfsbank.com

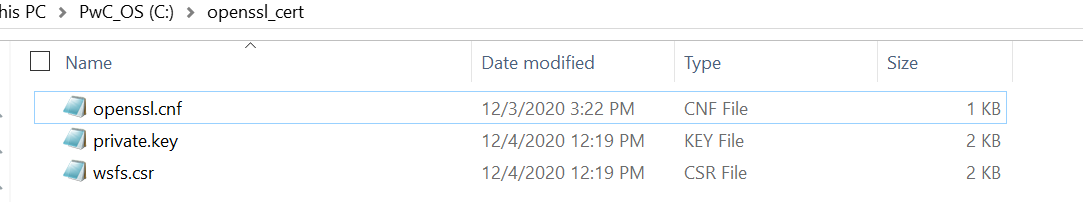
**Step2:**

Run below openssl command which will take the above **openssl\_core.conf** configuration file and generate a private key named **private.key** and a csr named **wsfs.csr**

req -nodes -new -keyout private.key -out wsfs.csr -config openssl\_core.cnf



Verify that private.key and wsfs.csr file generated successfully.



**Step 3:**

Using this CSR (wsfs.csr), generate CA certified certificates.

Please reach out to John Bailey from NTS team for CA certification and share this wsfs.csr file

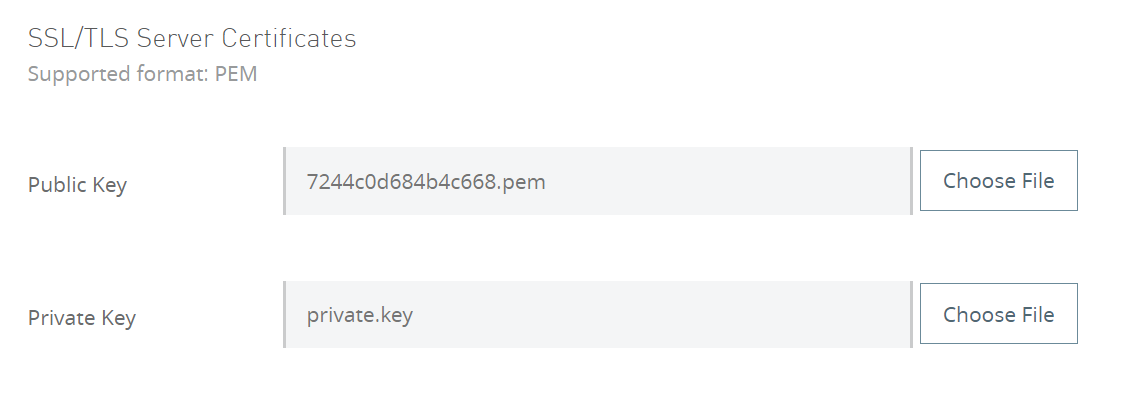
* + You will receive the certificate in two formats from John B
    - .crt
    - .pem

These above 3 Steps will generate final private key and public key to be used in MuleSoft Cloudhub DLB

* + Please upload .pem format CA certificate received from John B / NTS team

as Public key in DLB

* + Please upload private.key as Private Key in DLB



*Steps to generate KeyStore/Trust Store for Mule internal Https communication using Self Signed Certificate*

**Prerequisite**

* OpenSSL must be available in the system before executing the openssl commands

Details on OpenSSL can be found here <https://www.openssl.org/>

* Keytool must be available in system to run Java KeyStore/Trust Store (JKS) command

Keytool is normally available in bin directory of Java-JDK installation.

e.g keytool.exe is available here

C:\Program Files\AdoptOpenJDK\jdk-8.0.252.09-hotspot\bin

**KeyStore/TrustStore Generation Steps**

**Step 1:**

Create a file named openssl\_internal.conf

Add below entries in the file.

[req]

default\_bits = 2048

default\_md = sha256

prompt = no

distinguished\_name = dn

req\_extensions = req\_ext

[dn]

CN=\*.lb.anypointdns.net

C=US

ST=Delaware

L=Wilmington

O=WSFS Bank

[req\_ext]

subjectAltName=@alt\_names

[alt\_names]

DNS.1= \*.lb.anypointdns.net

*Note:*

Wild card \*.lb.anypointdns.net is used in Common Name (CN) and subjectAltName (SAN) as internal MuleSoft API layers (experience to process to system) will communicate using DLB internal URL.

Wild card is used to accommodate both prod and non-prod DLB names and any future update on DLB names.

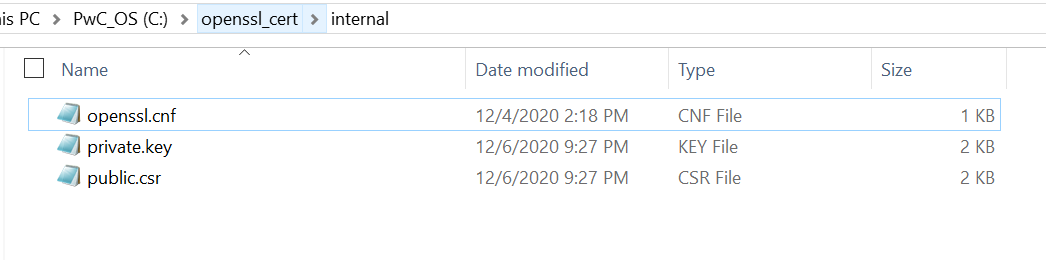
**Step2:**

Run below openssl command which will take the above **openssl\_internal.conf** configuration file and generate a private key named **private.key** and a csr named **public.csr**

**req -nodes -new -keyout private.key -out public.csr -config openssl\_internal.cnf**



Verify that private.key and public.csr file generated successfully.



**Step3:**

Run below openssl command which will take the above CSR (public.csr) and private key

(private.key) and generate a self-signed certificate with 10 years (3650 days) validity.

Certificates will be generated in PEM format.

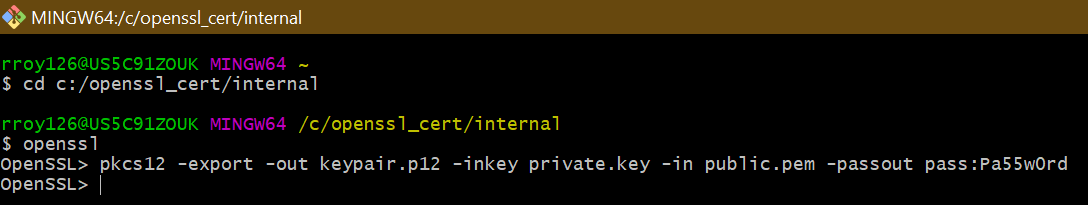
PEM format is enough in our case as PEM formatted certificate can be used in DLB as well as in TrustStore

**x509 -req -days 3650 -in public.csr -signkey private.key -sha256 -out public.pem -outform PEM**

**Step 4:**

combine public and private key, this concatenated file will be used in KeyStore later , provide a password for the combined file

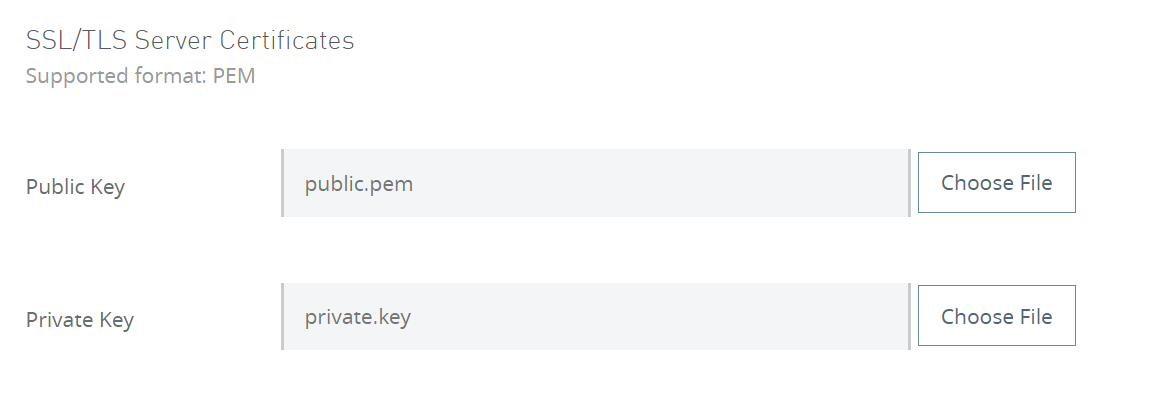
**pkcs12 -export -out keypair.p12 -inkey private.key -in public.pem -passout pass:Pa55w0rd**



**Step 5:**

Add another certificate entry in Cloudhub DLB

Use private.key as Private Key and public.pem as Public Key



**Step 5**

Now create a KeyStore and TrustStore to be used in MuleSoft application (HTTP Listener

HTTP Requester)

<https://help.mulesoft.com/s/article/Configure-HTTPS-Listener-secured-by-TLS-1-2>

*KeyStore Creation*

Create a key store named key\_store.jks and add/import previously generated public&private key pair keypair.p12 in that store . The same public and private key also added in DLB

**keytool -importkeystore -srckeystore keypair.p12 -srcstoretype pkcs12 -srcstorepass Pa55w0rd -destkeystore key\_store.jks -deststoretype JKS -deststorepass Pa66w0rd**

Note:

2-way SSL before DLB and 1-way SSL after DLB

CloudHub’s load balancer does NOT validate the worker’s certificate.

Still you must put key store in HTTPS Listener to avoid compilation error.

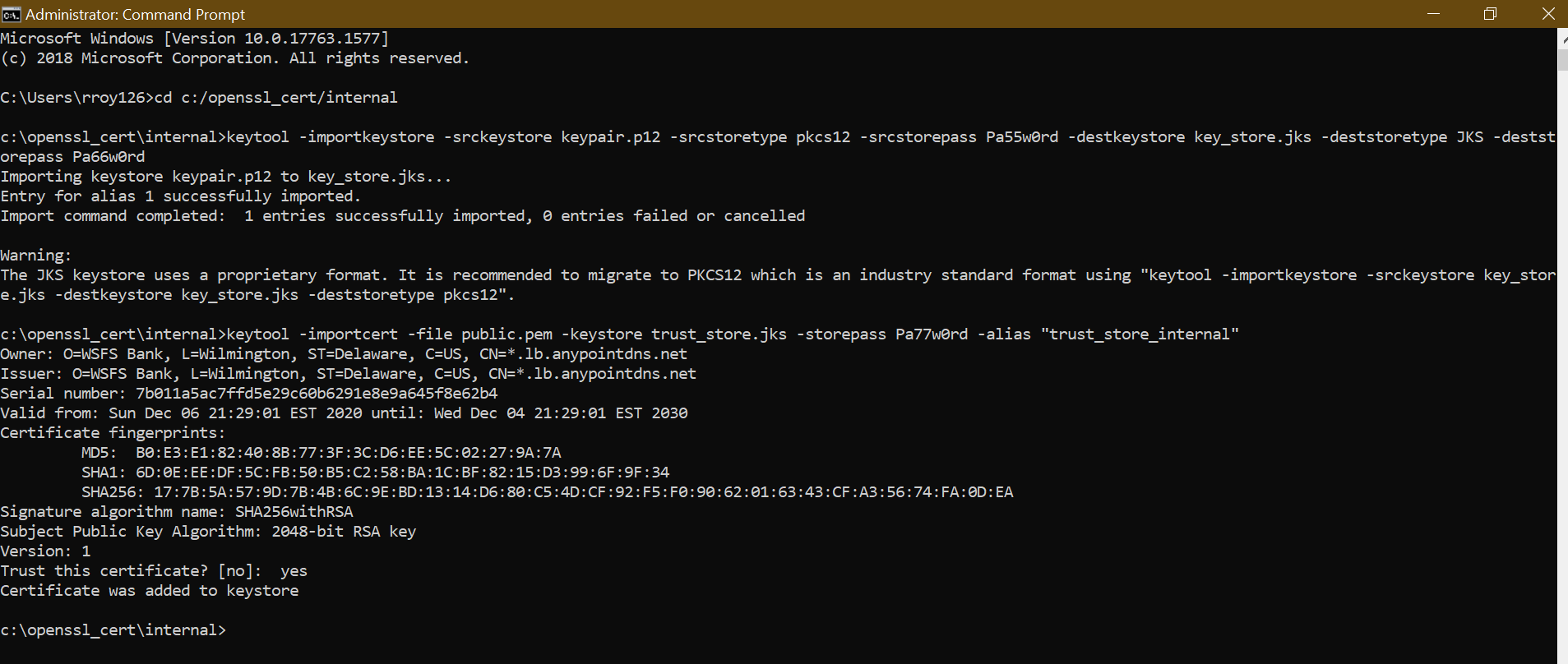
*TrustStore Creation*

Create a trust store named trust\_store.jks and add/import DLB public key public.pem

In that store

**keytool -importcert -file public.pem -keystore trust\_store.jks -storepass Pa77w0rd -alias "trust store\_internal"**

java.security.cert.CertificateException: No subject alternative DNS name matching internal-api-wsfs-non-prod-dlb.lb.anypointdns.net found. at sun.security.util.HostnameChecker.matchDNS(HostnameChecker.java:215)



**Step 6:**

Verify all files generated successfully.

