Reference
https://priyalwalpita.medium.com/create-your-own-certificate-authority-47f49d0ba086
https://www.wowza.com/docs/how-to-import-an-existing-ssl-certificate-and-private-key#:~:text=You%20can't%20directly%20import,12%20file%20into%20your%20keystore.

Creating your own CA – Root CA is always created via OPEN SSL in linux

-Step 1 (Create the private key)
--Create a private key using openssl
--Command(openssl genrsa -des3 -out CAPrivate.key 2048)
--This would output the CAPrivate.key file. Please note that you will be promoting to give a passphrase when executing this command. It is recommended to enter a password and secure your private key.
-Step 2(Generate the root certificate)
--As the next step, you need to create your root certificate using the generated private key. Ideally, this root certificate should be provided by another CA on top of your CA (this is called as the certificate chain). But in this instance , we are trying to act as the root CA and hence we need to create the root certificate file as follows
--Command(openssl req -x509 -new -nodes -key CAPrivate.key -sha256 -days 365 -out CAPrivate.pem)
--We are creating the certificate file as x-509 certificate standard and using the sha256 as the hashing algorithm. Our certificate would be valid for 365 days. You will be prompting to enter lots of certificate related data in this step followed by validation of your private key's password. You can give any value into these
--Congratz ! Not you are technically as CA. Let's test the operation of this CA.

Creating a self signed certificate using above CA – This steps we need to refer the one which uses KEYTOOL

-Step 1 (This step we should do it using keytool)
--First of all, we need to generate a private key.
--Command(openssl genrsa -out MyPrivate.key 2048)
--We are using the RSA asymmetric algorithm to generate this private key. As the next step we need to generate the CSR (Certificate request) using this private key.
-Step 2 (Creating CSR of private key generated in above step) (This step we should do it using keytool)
--Command(openssl req -new -key MyPrivate.key -out MyRequest.csr) a certificate signing request is created(CSR)
--you need to send this CSR to the CA to obtain the certificate file
-Step 3(Generate the Digital Certificate using the CSR)
--We are going to create a X509 certificate using the CSR. We are setting the certificate's validity period for 1 year (-days 365). Note that we need to use our CA's root certificate and the private key in this operation. We need to enter the CA private key's password as well when prompted
--openssl x509 -req -in MyRequest.csr -CA CAPrivate.pem -CAkey CAPrivate.key -CAcreateserial -out X509Certificate.crt -days 365 -sha256

We can test the generated certificate
--First let's sign some text file using the user's private key.
--Command(openssl dgst -sha256 -sign MyPrivate.key -out signature.txt sign.txt)
--The signature.txt would hold the signature of the content of the sign.txt file
--We can verify this signature by using user's certificate as follows. First of all , load the X509 certificate into the openssl tool and then perform the verification.
--openssl x509 -in X509Certificate.crt

--Verification step : openssl dgst -sha256 -verify <(openssl x509 -in X509Certificate.crt -pubkey -noout) -signature signature.txt sign.txt (this will display verified ok)
--Note : You have to use a secure vault or a HSM (Hardware Security Module) to persist all secret files in this process if you are going to use this in a production environment.

View the certificate
--openssl x509 -in X509Certificate.crt -text

How to combine (method 2) – *This actually works !!!*

- Reference https://help.highbond.com/helpdocs/analytics-exchange/5/Content/common/t_creating_a_keystore_and_importing_your_certificate.html
- Generate private-public key pair using keytool
    - keytool -genkey -alias <alias> -keyalg RSA -keystore <keystore_filename>
        - keytool -genkey -alias tomcat -keyalg RSA -keystore selfsignednew.jks (Actual command)
        - keytool -genkey -alias tomcat -keyalg RSA -keystore selfsignednew.jks -storetype JKS (here we have defined store type as JKS)
- We need to generate the CSR
    - keytool -certreq -alias <alias> -keyalg RSA -file <csr_output_file> -keystore <keystore_filename>
        - keytool -certreq -alias tomcat -keyalg RSA -file tomcat.csr -keystore selfsignednew.jks
- Once CSR is generated ,we are creating the digital certificate (self signed) by the root CA created in earlier steps
    - openssl x509 -req -in tomcat.csr -CA CAPrivate.pem -CAkey CAPrivate.key -CAcreateserial -out X509Certificate.crt -days 365 -sha256
- Now we need to combine below files into selfsignednew.jks
    - CAPrivate.pem (this is the root CA) ==> keytool -import -alias rootca -keystore selfsignednew.jks -trustcacerts -file CAPrivate.pem(this we have to move from linux to windows ,as all openssl artifacts were generated in linux)
    - X509Certificate.crt (this is the self signed certificate) ==> keytool -import -alias tomcat -keystore selfsignednew.jks -trustcacerts -file X509Certificate.crt
        - The alias specified must be the same value specified in step 2 when you generated the keystore. The imported certificate will replace the default self-signed certificate created in the keystore
    - The private key is already there in the keystore selfsignednew.jks

Debugging Keystore file type Issues – *Very Very Important!!!!*

- https://www.herongyang.com/JDK/keytool-keystore-File-Type-PKCS12-and-JKS.html
  If you create a new keystore file, herong.jks, with the JDK 17 "keytool" command and run the "-list" command option, you will see "PKCS12" is listed as the file type
- If you create a new keystore file, herong.jks, with the JDK 7 "keytool" command and run the "-list" command option, you will see "JKS" is listed as the file type
- For backward compatibility, JDK 12 "keytool" command still spports "JKS" file type with the "-storetype JKS" option
- Where we can change the keystore type supported in tomcat
    - "C:\Pega_8_8\PRPCPersonalEdition\jre1.8.0_121\lib\security\java.security"
    - keystore.type=jks (REFER Screen shot below)
- One of the time consuming issue was the change of default key store type on JDK17

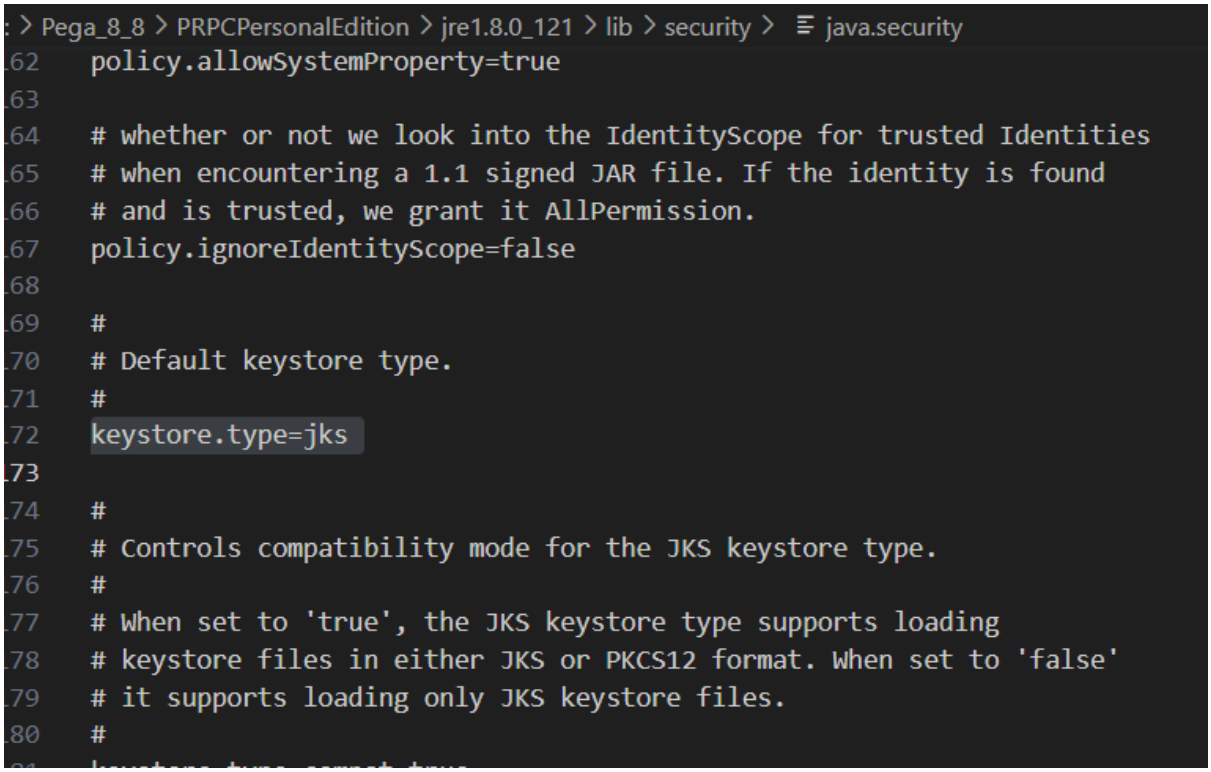
- The PCKS12 format was not getting accepted by TOMCAT by changing at different places.
- Finally the keystore was re-created using below command
    - keytool -genkey -alias tomcat -keyalg RSA -keystore selfsignednew.jks -storetype JKS

Importing CA Root

```
C:\George\Technical\cert_study\create_certs_new>keytool -import -alias rootca -keystore selfsignednew.jks -trustcacerts -file CAPrivate.pem
Enter keystore password:
Owner: EMAILADDRESS=georgeputhen@yahoo.com, CN=localhost, OU=gjpegaunit, O=gjpega, L=Hyderabad, ST=Telangana, C=IN
Issuer: EMAILADDRESS=georgeputhen@yahoo.com, CN=localhost, OU=gjpegaunit, O=gjpega, L=Hyderabad, ST=Telangana, C=IN
Serial number: 1f92eba59387371744791bf3eb5b7654b8d590c9
Valid from: Sat Jun 03 16:48:20 IST 2023 until: Sun Jun 02 16:48:20 IST 2024
Certificate fingerprints:
         SHA1: DD:57:5F:0F:84:97:E4:3F:1E:3D:4C:83:5E:52:F9:09:3F:99:51:0F
         SHA256: 72:3B:85:69:85:00:6C:54:56:73:72:A0:FA:D3:7E:7A:38:75:04:2C:E6:4B:4D:8A:4C:32:95:C3:BE:9D:D3:7E
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 82 98 03 76 95 12 D7 BF   FC 5D C4 84 19 79 59 E1  ...v.....]...yY.
0010: 0D C8 52 DA                                        ..R.
]
]

#2: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:true
  PathLen: no limit
]

#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 82 98 03 76 95 12 D7 BF   FC 5D C4 84 19 79 59 E1  ...v.....]...yY.
0010: 0D C8 52 DA                                        ..R.
]
]

Trust this certificate? [no]:  YES
Certificate was added to keystore
```

Importing Selfsigned digital certificate

```
C:\George\Technical\cert_study\create_certs_new>keytool -import -alias tomcat -keystore selfsignednew.jks -trustcacerts
-file X509Certificate.crt
Enter keystore password:
Certificate reply was installed in keystore
```


: > Pega_8_8 > PRPCPersonalEdition > jre1.8.0_121 > lib > security > java.security
162 policy.allowSystemProperty=true
163
164 # whether or not we look into the IdentityScope for trusted Identities
165 # when encountering a 1.1 signed JAR file. If the identity is found
166 # and is trusted, we grant it AllPermission.
167 policy.ignoreIdentityScope=false
168
169 #
170 # Default keystore type.
171 #
172 keystore.type=jks
173
174 #
175 # Controls compatibility mode for the JKS keystore type.
176 #
177 # When set to 'true', the JKS keystore type supports loading
178 # keystore files in either JKS or PKCS12 format. When set to 'false'
179 # it supports loading only JKS keystore files.
180 #