Convert PFX certificate to JKS, P12, CRT

I recently had to use a PFX certificate for client authentication (maybe another post will be coming) and for that reason I had to convert it to a Java keystore (JKS).

We will create BOTH a truststore and a keystore, because based on your needs you might need one or the other.
The difference between truststore and keystore if you are not aware is(quote from the JSSE ref guide:

TrustManager: Determines whether the remote authentication credentials (and thus the connection) should be trusted.
KeyManager: Determines which authentication credentials to send to the remote host.

Ok that's enough what you will need is openssl and Java 7+ ;) !

First let's generate a key from the pfx file, this key is later used for p12 keystore.

```
openssl pkcs12 -in example.pfx -nocerts -out example.key
Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

As shown here you will be asked for the password of the pfx file, later you will be asked to enter a PEM passphase lets for example use 123456 for everything here.
The second commands is almost the same but it is about nokey and a crt this time

```
openssl pkcs12 -in example.pfx -clcerts -nokeys -out example.crt
Enter Import Password:
MAC verified OK
```

Now we have a key and and a crt file
Next step is to create a truststore.

```
keytool -import -file example.crt -alias exampleCA -keystore
truststore.jks
Enter keystore password:
```

```
Re-enter new password:
Owner: CN=.....
.......
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

As you can see here you just import this crt file into a jks truststore and set some password. For the question do you trust this certificate you say yes, so it is added in the truststore.

We are done if you only need a truststore.
The last step(s) is to create a keystore

```
openssl pkcs12 -export -in example.crt -inkey example.key -
certfile example.crt -name "examplecert" -out keystore.p12
Enter pass phrase for example.key:
Enter Export Password:
Verifying - Enter Export Password:
```

This p12 keystore is enough in many cases, still if you need a JKS keystore you need one additional command

```
keytool -importkeystore -srckeystore keystore.p12 -srcstoretype
pkcs12 -destkeystore keystore.jks -deststoretype JKS
Importing keystore keystore.p12 to keystore.jks...
Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
Entry for alias examplecert successfully imported.
Import command completed:  1 entries successfully imported, 0
entries failed or cancelled

Warning:
The JKS keystore uses a proprietary format. It is recommended to
migrate to PKCS12 which is an industry standard format using
"keytool -importkeystore -srckeystore keystore.jks -destkeystore
keystore.jks -deststoretype pkcs12".
```

That is all folks ! I hope this helps someone :)

```
ls
example.pfx  example.key            keystore.p12
example.crt  keystore.jks           truststore.jks
```

Use Client Certificate Authentication with Java and RestTemplate

we now have a keystore and a truststore (if anyone needs) and we will use this keystore to send client side authentication using Spring's RestTemplate .

First copy your keystore.jks and truststore.jks in your classpath, no one wants absolute paths right?

Again a reminder The difference between truststore and keystore if you are not aware is(quote from the JSSE ref guide):

TrustManager: Determines whether the remote authentication credentials (and thus the connection) should be trusted.

KeyManager: Determines which authentication credentials to send to the remote host.

The magic happens in the creation of SSLContext. Keep in mind the Spring Boot have a nice RestTemplateBuilder but I will not gonna use it, because someone of you might have an older version or like me, might just use a plain old amazing Spring.

If you **just** want to use the keystore:

```
final String allPassword = "123456";
SSLContext sslContext = SSLContextBuilder
        .create()
        .loadKeyMaterial(ResourceUtils.getFile("classpath:keystore.jks"),
                    allPassword.toCharArray(), allPassword.toCharArray())
        .build();
```

if you just want to use the truststore

```
final String allPassword = "123456";
SSLContext sslContext = SSLContextBuilder
        .create()
        .loadTrustMaterial(ResourceUtils.getFile("classpath:truststore.jks"),
allPassword.toCharArray())
        .build();
```

I guess you know how to use both ;), if you want to IGNORE the truststore certificate checking and trust ALL certificates (might be handy for testing purposes and localhost)

```
final String allPassword = "123456";
TrustStrategy acceptingTrustStrategy = (X509Certificate[] chain, String
authType) -> true;
SSLContext sslContext = SSLContextBuilder
        .create()
        .loadTrustMaterial(ResourceUtils.getFile("classpath:truststore.jks"),
allPassword.toCharArray())
        .loadTrustMaterial(null, acceptingTrustStrategy) //accept all
        .build();
```

Ones you have the sslContext you simply do :

```
HttpClient client = HttpClients.custom()
                .setSSLContext(sslContext)
                .build();

HttpComponentsClientHttpRequestFactory requestFactory =
        new HttpComponentsClientHttpRequestFactory();

requestFactory.setHttpClient(client);

RestTemplate restTemplate = new RestTemplate(requestFactory);

return restTemplate;
```

And Voala, now each time you make a get/post or exchange with your restTemplate you will send the client side certificate.

Full example (the "tests" version) that sends client side certificate and ignores the SSL certificate

```
private RestTemplate getRestTemplateClientAuthentication()
        throws IOException, UnrecoverableKeyException, CertificateException,
NoSuchAlgorithmException,
        KeyStoreException, KeyManagementException {

  final String allPassword = "123456";
  TrustStrategy acceptingTrustStrategy = (X509Certificate[] chain, String
authType) -> true;
  SSLContext sslContext = SSLContextBuilder
          .create()
          .loadKeyMaterial(ResourceUtils.getFile("classpath:keystore.jks"),
                   allPassword.toCharArray(), allPassword.toCharArray())
//.loadTrustMaterial(ResourceUtils.getFile("classpath:truststore.jks"),
allPassword.toCharArray())
          .loadTrustMaterial(null, acceptingTrustStrategy)
          .build();

  HttpClient client = HttpClients.custom()
                .setSSLContext(sslContext)
                .build();

  HttpComponentsClientHttpRequestFactory requestFactory =
          new HttpComponentsClientHttpRequestFactory();

  requestFactory.setHttpClient(client);

  RestTemplate restTemplate = new RestTemplate(requestFactory);
  return restTemplate;
}
```