JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 10, November-December 2004

The Initium X.509 Certificate Wizard

Douglas Lyon, Fairfield University, Fairfield CT, U.S.A.

barf [ba:rf]
2. "He suggested using FORTRAN,
and everybody barfed."
From The Shogakukan
- DICTIONARY OF NEW ENGLISH
(Second edition)

Abstract

This paper describes the use of the Thawte's "Web of Trust" X.509 certificates for signing and distributing executable Jar resources. A *keytool* wizard (called the *Initium X.509 Certificate Wizard*) was developed in order to help with the importation and management of certificates.

A signed Jar file generally indicates that the signer authorizes the contents. Signing is accomplished using a certificate that has been issued by a Certificate Authority (CA). Several CA's are available for this task; however, few of them are free, like the Thawte CA. Once a free certificate is obtained, Jar files may be distributed and named as verified from the signer. Trusted Jar files can be run outside of the "sandbox" and thus be given improved access to the target system.

The impact of having a trusted Jar file is that Jar distribution systems (like Java Web Start, or Browser-based Applets) can run the program in the trusted manner. Therefore, such trusted programs can have access to the files, or be able to open connections to hosts other than the web host.

The *keytool* wizard addresses a subproblem of the *Initium* project, a joint, on-going project between the Fairfield University and DocJava, Inc. Initium is a Latin word that means: "at the start".

1 THE THAWTE WEB OF TRUST

In order for Java Web Start to give unrestricted permission for a Java program to execute, it must use a "signed" Jar file. A signed Jar file is designed to prove that the originator is the author of the code. This does NOT prevent the author from writing harmful code. On the other hand, if you trust the author to write non-harmful code, you may feel safer about running the authors' programs.

In order to sign a Jar file, you need a digital certificate. Certificates are issued after a proper background check, by a Certification Authority (CA). This is not generally a free service. For example, Verisign asks for $200 or $400 per year, in order to issue a certificate.

Applications that are signed by an untrusted signature (i.e., a signature that is not verified by a known CA) cause a dialog to be displayed saying:

"It is highly recommended not to install and run this code".

The software that I write is typically given away. As a result, I am disinclined to pay $200-$400 per year. Therefore, a free (or at least very cheap) approach to obtaining a certificate appeals to my sense of thrift. The Thawte personal e-mail certificate can be used indefinitely, and at no cost. *Thawte* is a CA that can issue a digital certificate to an organization or an individual. It is the role of the CA to verify that the company ordering the certificate is a registered organization that controls its domain and that the person in the company, who ordered the certificate, is authorized to do so.

Authentication helps to prevent spoofing. It is harder to substitute illegitimate programs for programs that come from established organizations when the programs must be properly signed. Con artists could make use of such untrusted programs to steal credit card numbers or create software that destroys or distorts data. Thus having an authentication procedure in place helps to establish trust and adds value to the Java programs that you distribute.

From the point of view of grid computing, a grid operator will want some assurance that the program being submitted for execution is safe. At the very least, the program can be attributed to its signer before being deployed on the grid. In the event the grid application contained damaging code, the compute servers on the grid would become infected with the code faster than normally propagated computer contagions. Such a program places the entire grid as risk.

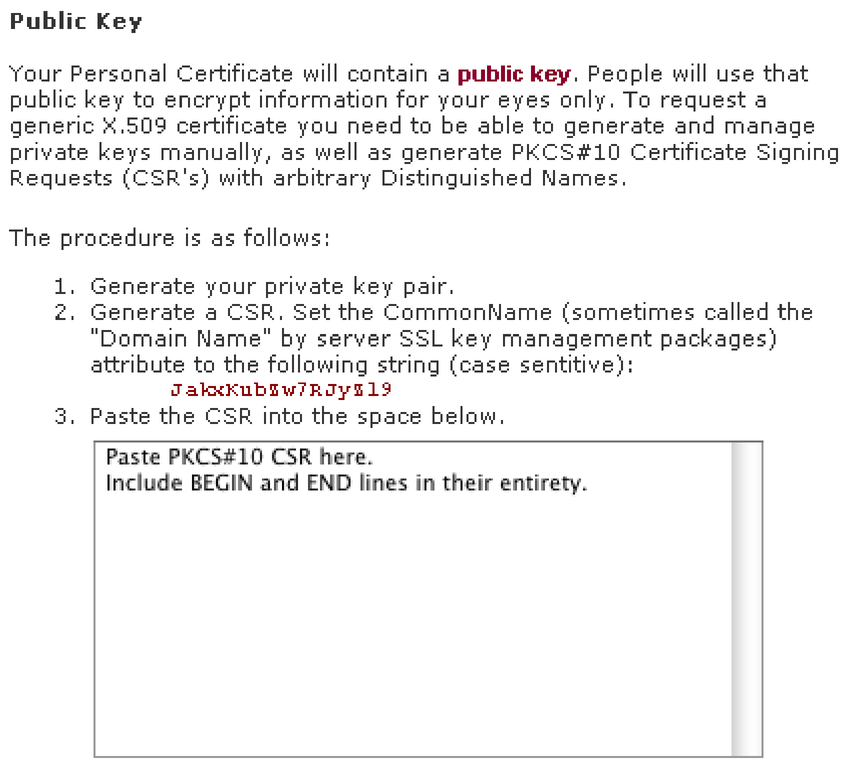
2 HOW DO I GET A CERTIFICATE?

This section presents the steps that are needed in order to obtain a free personal e-mail certificate from Thawte. First you must visit https://www.thawte.com/email/index.html#, run by Thawte. You then click on "join" and fill out the application. Information needed includes, name, data of birth, and a "national identification number". This can consist of a diver's license number, social security number or passport number. You must also enter an e-mail address. Other information needed includes: your phone number, your mother's maiden name, your father's middle name, what is the make of your fridge, etc. Your free certificate will not show your name when you sign your jar files. Java Web Start, for example, will show your name as: "Thawte Freemail Member".

In order to obtain a certificate for signing your jar, you will have to wait for a confirmation e-mail from Thawte. This will contain the codes that you need in order to confirm your e-mail address. Scroll down to “Developers of New Security Applications ONLY”. Then select:

```
“Paste-in CSR Certificate Enrollment”. Since my identity had
not been verified by Thawte’s web of trust, my name appears as
"Thawte Freemail Member".
```

There are several screens that you must follow. Eventually you will reach the screen shown in Figure 1, which contains a text box that enables the entry of a public key.



Public Key

Your Personal Certificate will contain a **public key**. People will use that public key to encrypt information for your eyes only. To request a generic X.509 certificate you need to be able to generate and manage private keys manually, as well as generate PKCS#10 Certificate Signing Requests (CSR's) with arbitrary Distinguished Names.

The procedure is as follows:

1. Generate your private key pair.
2. Generate a CSR. Set the CommonName (sometimes called the "Domain Name" by server SSL key management packages) attribute to the following string (case sentitive):
   `JakxKubSw7RJyS19`
3. Paste the CSR into the space below.

Paste PKCS#10 CSR here.
Include BEGIN and END lines in their entirety.

Figure 1. The Key Dialog Box

At this point you must generate a public key using *keytool*. An RSA key can be created using the following command:

```
keytool -genkey -keyalg RSA -keystore keystore -alias docjava
```

A public and private key pair is created when the “-genkey” parameter is specified. The “*–keyalg*” specifies the algorithm that will be used to encode the certificate, in this case *RSA*. The -*keystore* option specifies the name and location of the persistent keystore file for the keystore managed by the keytool. You will want to alter the “docjava” alias to reflect your own domain. You will be prompted for a password. As implemented, the password will be echoed on the command line, so don’t type it in front of anyone (a *keytool* man page details additional options). A sample session follows:

```
keytool -genkey -keyalg RSA -keystore keystore -alias docjava
```

<password prompt appears here, the password is echoed on the console, so do not let anyone look over your shoulder>

```
What is your first and last name?
  [Unknown]:  QeUxVvne83ETFF02
What is the name of your organizational unit?
  [Unknown]:  DocJava
What is the name of your organization?
  [Unknown]:  DocJava, Inc.
What is the name of your City or Locality?
  [Unknown]:  Milford
What is the name of your State or Province?
  [Unknown]:  CT
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=QeUxVvne83ETFF02, OU=DocJava, O="DocJava, Inc.",
L=Milford, ST=CT, C=US correct?
  [no]:  yes
cube.local{lyon}11: ls -al keystore
-rw-rw-r--  1 lyon  staff  1374 22 Apr 07:30 keystore
```

The next step is to export the key as a text file. This is done with the following command:

```
keytool -certreq -keystore keystore -file keystore.txt -alias
docjava
```

Once the *keytool.txt* file is available, you must list it out to the console and paste it into the key dialog box, shown in Figure 1. A confirmation certificate request dialog will appear, unless you make an error in the above steps. The confirmation informs you that:

```
"The certificate will have a distinguished name that looks
like this: Common: Thawte Freemail Member
Email:lyon@docjava.com
If you need a certificate with your full name in it, then you
need to join the Freemail  Web of Trust."
```

That is, you now have a certificate that you can use to sign files. However, the certificate identifies you as a generic "Thawte Freemail Member". If you want the certificate to identify you, by name, then you need join the "Freemail Web of Trust" membership, described in the following section.

3 FREEMAIL WEB OF TRUST

This section describes how you can get a certificate that proves you are who you say you are (by using the best efforts of the CA). This enables you to sign Jar files and have them executed by Java Web Start identifying you as the author and signer of the Jar file. Thus, your Jar files will be signed by a certificate issued by the CA, after the CA conducts a background check.

There are two ways in which to join the WOT, the free way and the non-free way. The free way requires that you meet with "Web of Trust" notaries. They will require two forms of identification and will vouch that you are who you say you are. WOT notaries may (as in my case) be spread out over some geographic distance. As a result, this may not be easy to do. Notaries can assign between 10 and 35 points. Once you have accumulated 50 Trust Points you become trusted, and you can put your name in your personal certificate. Once you accumulate 100 points you can apply to become a WOT notary.

The second (non-free) way to join the WOT is to obtain a “Remote Authentication” (RA). RA requires you to meet with two trusted third parties. Examples include, a bank manager, a practicing attorney or a CPA.

A bank manager must be the manager of a branch of a registered banking institution. The branch must be listed in the local telephone directory. A practicing attorney must be registered with the state bar association or your countries equivalent, and currently practice as an attorney. The attorney's law offices must be listed in the local telephone directory. A CPA must be licensed as a certified public accountant, with the relevant state or country authorities, and must be practicing as a CPA from an office that is listed as such in the local telephone directory.

There is a $25 fee for using RA and, once authorized, you become a notary yourself. I elected to pay the fee and went to two different notaries at two different local banks. As I had accounts there, they did not charge for the service of notarizing the Thawte forms.

The first time I made out the paperwork there was some sort of clerical error, which I still do not understand. As a result, I sent over 16 e-mails over a 4-week period. It was claimed that the notarized forms were not signed (which is impossible, since they were notarized). In response, I went and had the forms filled out and notarized again. The second attempt worked, and soon I had the coveted e-mail confirming that I had been notarized through the Thawte Remote Authentication System.

4 GETTING YOUR NEW CERTIFICATE

In order to get a new certificate (one that verifies that you are who you say you are) proceed to the Thawte web site https://www.thawte.com/cgi/personal/cert/enroll.exe and login. Select the link labeled “Developers of New Security Applications ONLY”. Then select:

```
 “Paste-in CSR Certificate Enrollment”.
```

To begin the process, I list the certificates loaded into my default keystore:

```
keytool -list
Enter keystore password:

Keystore type: jks
Keystore provider: SUN

Your keystore contains 2 entries

docjava, Apr 14, 2004, keyEntry,
Certificate fingerprint (MD5):
C0:67:B8:72:2D:DA:A3:7F:4B:A1:F0:D2:A6:24:2D:DD
mycert, Apr 20, 2004, keyEntry,
Certificate fingerprint (MD5):
CD:C1:A7:AB:3E:CA:5B:4D:B0:CA:BB:DF:66:61:73:83
```

The default keystore type is "jks" (the proprietary type of the keystore implementation provided by Sun). I then proceed to print information about my new Thawte x.509 certificate:

```
keytool -printcert -file deliver.exe
```

This outputs a chain of certificate data that, upon inspection, looks correct and trustworthy. Now I shall generate a public key using *keytool*. This is the same procedure as outlined in Section 1, except that now I am a confirmed Thawte notary:

```
keytool -genkey -keyalg RSA -alias docjavaInc
Enter keystore password:
What is your first and last name?
  [Unknown]:  82BWxXJf4LesA909
What is the name of your organizational unit?
  [Unknown]:  DocJava
What is the name of your organization?
  [Unknown]:  DocJava, Inc.
What is the name of your City or Locality?
  [Unknown]:  Milford
What is the name of your State or Province?
  [Unknown]:  CT
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=82BWxXJf4LesA909, OU=DocJava, O="DocJava, Inc.",
L=Milford, ST=CT, C=US correct?
  [no]:  yes
```

Now to get my certificate:

```
keytool -certreq -file keystore.txt -alias docjavaInc
```

Here is the output from the certificate request:

```
ls
deliver.exe     keystore.txt
cube.local{lyon}76: more *.txt
-----BEGIN NEW CERTIFICATE REQUEST-----
JLMEF (Junky Looking MIME Encoded Flotsam)
-----END NEW CERTIFICATE REQUEST-----
```

I then paste in the key tool output into the key dialog box, as shown in Figure 1. A confirmation dialog appears that says:

```
The certificate will have a distinguished name that looks like
this:
```

Surname:lyon

Given Names: Douglas

Common: Douglas lyon

Email:lyon@docjava.com

Compare the above output with that from Section 2:

```
"The certificate will have a distinguished name that looks
like this: Common: Thawte Freemail Member
Email:lyon@docjava.com
If you need a certificate with your full name in it, then you
need to join the Freemail  Web of Trust."
```

So far, so good. Now I proceed to my Thawte accounts' certificate manager page, at: https://www.thawte.com/cgi/personal/cert/status.exe and check the status of my X509 certificate request. After a few minutes, it changes status from *pending* to *issued* and I fetch my new X.509 certificate. The delivered certificate contents follows:

```
more deliver.exe

We deliver your developer cert as a chain, and we deliver it
in two formats. The first is the Netscape Cert Sequence
format, documented on the Netscape Security Site at
http://www.netscape.com/eng/security/. The next is a
degenerate PKCS7 SignedDate, with certificates and possibly
CRL's but no content.

-----BEGIN NETSCAPE CERTIFICATE CHAIN-----
MIII6wYJYIZIAYb4QgIFoIII3DCCCNgwggJgMIIByaADAgECAgMMzA0wDQYJKoZI
hvcNAQEEBQAw
YjELMAkGA1UEBhMCWkExJTAjBgNVBAoTHFRoYXd0ZSBDb25zdWx
0aW5nIChQdHkpIEx0ZC4xLDAq
```

In order to import the certificate into the key store, the PKCS7 part of the certificate must be saved into a file and then altered so that it looks like the following:

```
-----BEGIN PKCS #7 SIGNED DATA-----
MIII7AYJKoZIhvcNAQcCoIII3TCCCNkCAQExADALBgkqhkiG9w0BBwGgggjBMIICS
     TCCAbKgAwIB
AgIDDB2VMA0GCSqGSIb3DQEBBAUAMGIxCzAJBgNVBAYTAlpBMSUwIwYDVQQKExxUa
     GF3dGUgQ29u
```

That is, there must be 76 characters, per line. Thus, the carriage returns were inserted improperly. To extract the PKCS7 key, you can hand-edit the file, then fix the returns. As an alternative, I have written a small Java procedure called *cleanThawtes*. The *cleanThawtes* procedure automates the processing of certificates returned by Thawte. It is based on the work of Richard Dallaway [Dallaway]. As a part of the Initium keywizard, *cleanThawtes* was wrappered with a GUI and placed on the web under *Web Start Applications*, available at: http://www.docjava.com/. I used the program to create a file called *pk8.cert*. This was imported into the default location for the keystore file using:

```
keytool -import -file pk8.cert -alias docjavainc -trustcacerts
```

Errors in the parameters passed to the *keytool* command result in cryptic exceptions. For example:

```
keytool -import  -file pk8.cert -trustcacerts
Enter keystore password:
keytool error: java.lang.Exception: Input not an X.509
certificate
```

5 WHERE HAVE ALL THE KEYS GONE?

The default location for the keystore file is *.keystore* and is located in the users' home directory. In order to obtain this file, a public method was created in a new class called the *KeyUtils*:

```
/**
 * @return the keystore file
 */
    public static File getKeystoreFile() {
        File file = new File(System.getProperty(
                "user.home") +
                             File.separatorChar +
                             ".keystore");
        if (file.exists()) return file;

        return futils.Futil.getReadFile(
                "file:" + file +
                " does not exist; select a keystore");
    }
```

To import a PKCS7 certificate, after it has been properly formatted, I have added the following code to *KeyUtils*:

```
public static void importCertificate()
            throws FileNotFoundException,
                   KeyStoreException,
                   CertificateException {
        final String password = getPassword();
        KeyStore ks = getKeyStore(password);
        final String alias = getAlias(password);
        Certificate cert = getCertificate(
                getCertificateFile());

        if (ks.containsAlias(alias))
            ks.deleteEntry(alias);
        ks.setCertificateEntry(alias, cert);
    }

    public static Certificate getCertificate(
            File certF)
            throws CertificateException,
                   FileNotFoundException {
        CertificateFactory cf =
                CertificateFactory.getInstance(
                        "X509");
        Collection col = cf.generateCertificates(
                new FileInputStream(certF));
        Iterator i = col.iterator();
        X509Certificate c = null;
        while (i.hasNext()) {
            Certificate cert = (Certificate) i.next();
            if (cert instanceof X509Certificate)
                c = (X509Certificate) cert;
            System.out.println(cert);
        }
        return c;
    }


    public static File getCertificateFile() {
        return
                futils.Futil.getReadFile(
                        "select a cert in pkcs7 format");
    }
```

In order to deal with the case of an improperly formatted certificate, I have created a more sophisticated way to get certificates, one that invokes the Thawte cleaner:

```
public static Collection importCertificates(
            CertificateFactory cf, File certF)
            throws CertificateException,
                   FileNotFoundException,
                   IOException {
        final FileInputStream fis = new FileInputStream(
                certF);
        Collection col = null;
        try {
            col = cf.generateCertificates(fis);
        } catch (CertificateException e) {
            boolean b = futils.In.promptYesNo(
                    "Cert exception, would you like me to try
to translate it?");
            if (!b) return importCertificates(cf);
            CertUtils.cleanThawtes();
            return importCertificates(cf);
        }
        return col;
    }
```

After adding more error correction prompts, we are able to run the Initium key wizard under a series of different situations. For example, when no *.keystore* file appears in the users' home, the program will prompt you for a key store file. If you don't have one, the program will offer to make one for you, adding a self-signed certificate. The program will also offer to create a certificate request file, which can be used to obtain a trusted certificate. This simplifies key management.

6 RELATED WORK

The creation of key tool API's is not new [BouncyCastle]. Verisign has an API, but it is no longer supported, as far as I can tell [Verisign]. The interesting thing about the Verisign API is that it enables users to revoke public keys at the CA (which is an excellent idea). This is something that would enable the automation of Thawte interactions and is a topic of future work.

There have been articles on the signing of Java Web Start applications [Dallaway]. There have also been articles on the signing of Applets [Gallant].

The creation of a key wizard seems like a logical evolution of the key tool. It is surprising, therefore, that keytool wizards are so few and far between. A reason for this is that Sun has not open-sourced the JKS (Java Key Store) API. On the other hand, there are open-source versions of a JKS system [Marshall]. I have yet to test this system.

One notable keytool wizard is the BEA systems wizard for their Weblogic product [BEA]. This systems is closed-source, expensive and only works on Weblogic.

7 CONCLUSION

Basic elements of key management need to be automated in support of the automation of Java Web Start application deployment. In fact, it is the *Initium* project that provided the primary motivation for creating the key wizard.

The automation of key tool functions is not hard, give proper error handling and GUI implementation effort. One of the surprising elements is that the API upon which the key tool is based is neither public, nor open-sourced. What this means to the key wizard author is that a substantial implementation library has to be created, tested and documented in order to provide the key tool function points.

Even more disconcerting is the general lack of programmatic support for the retrieval of trusted certificates. After a certificate request is created, the user must still interact with a CA in order to go through a process of getting a certificate. This limitation is less a technical one than a security one.

For those in organizations where there may be multiple employees (or students) who want to sign applications, obtaining certificates using remote authentication is cumbersome, at best. In such cases, several notaries could be established so that students/employees can be issued certificates quickly and at little additional cost.

REFERENCES

[Bea] “The Bea Systems Keytool Wizard” http://edocs.bea.com/wli/docs70/b2bsecur/keystore.htm

[BouncyCastle] http://www.bouncycastle.org/specifications.html, An open-source API that provides JCE features.

[Dallaway] Richard Dallaway, “Java Web Start and Code Signing”, May 2002 http://www.dallaway.com/acad/webstart/

[Gallant] Michel I. Gallant, “Thawte Code Signing Certs with Authenticode, Netscape and Sun Signing”, http://www.jensign.com/JavaScience/Thawte/

[Marshall] Casey Marshall, “An implementation of the JKS key store” http://metastatic.org/source/JKS.html

[Versign] “Trust Services Integration Kit 1.7” http://home.postech.ac.kr/~chokee/cs499/tsik-1.7/tsik-1.7/api/com/verisign/xkms/tools/XKMSKeyStore.html, and http://www.verisign.com/static/005314.pdf

APPENDIX - The KeyUtils API

The general lack of high-level tools for performing key tool functions has led to the creation of the following API for key and certificate manipulations.

```
public static void runImportCertificate()
        - a gui for importing certificates

public static void importCertificate()
        - Open the .keystore file. Prompt the user for an
alias. Open the new certificate from the CA. Modify the
.keystore file with the new certificate. If files are missing,
recover from the error with grace.

public static boolean isSelfSigned(
        X509Certificate cert)
Returns - true if the subjectDN and issuer are the same.

public static void verifyCert(
        X509Certificate cert)
        - Display exception if certificate cannot be verified.

public static java.security.cert.Certificate getCertificate(
        File certF)
        - Use a certificate file to make a certificate instances.

public static void printProviders()
List the security providers in their order of preference.

public static void printKey(Key key)
        - print out a nicely formatted version of a given key.

public static java.security.cert.Certificate getCertificate()
        - get a certificate based on a GUI prompt to the user
for a password and an alias.
```

```
Returns - Certificate

public static java.lang.String getPassword()
        - Prompt the user for a password. Do not echo it on
the screen
Returns - a string containing the password.

public static java.security.cert.Certificate getCertificate(
        String alias,
        String password)
        - Given an alias and password, open the default
keystore and return the certificate.
Returns - a Certificate instance.

public static java.security.KeyStore getKeystore(
        String password)
        - Return a KeyStore assuming that one already exists.
If the .keystore file does not exist, then offer to create one
or look for one.
Parameters - password to the keystore
Returns - KeyStore instance

public static void testGenerateKeyPair()

public static java.security.KeyStore generateKeyPair()
        - Creates a keystore, then generates a keypair for it.
Returns - KeyStore with keypair in it.

public static java.security.KeyStore generateKeyStore()
        - Generate a keystore without reading it from a file.
Excellent for when no keystore is found.
Returns - a default type keystore instance

public static void generateKeyPair(KeyStore ks)
        - Prompts the user for X.509 certificate information.
Generates a private and public keypair, then add it to the
keystore. Uses RSA algorithm with a 1024 bit key size.

public static java.security.KeyPair getKeyPair(
        KeyStore keystore,
        String alias,
        String password)
        - get a public and private key, given that a KeyStore
exists and a certificate exists that corresponds to the given
alias.

public static java.security.KeyStore getKeyStore()
        - Prompts the user for the password and gets the
.keystore file in the users' home directory.

public static java.security.KeyStore getKeyStore(String
password)
```

```
        - Selects the .keystore file in the users home
directory.
Returns - KeyStore

public static java.security.KeyStore getKeyStore(
        File keyStoreFile,
        String password)

public static java.lang.String[] getAliasArray(
        KeyStore keystore)
        - Given a key store, list all the alias elements
there. Certificates are located via the alias.
Returns - An array of alias members.

public static void save(
        File ksFile,
        KeyStore ks,
        String password)
        - Write our key store instance out to the given file.
A GUI prints out exceptions, should they be thrown.
Parameters
        ksFile - a file to be created or overwritten.
        ks - the key store to be saved.
        password - verifies the file.

public static java.io.File getKeystoreFile()
        - Look for the .keystore file in the home directory.
If it is not there, offer to look for it. If the user does not
have it, offer to create one. If you have to create a
keystore, offer to create a certificate request, as well.
Returns - the keystore file

public static void writeCertReq(
        String alias,
        String keyPass,
        KeyStore ks)
        - Output a file based on user prompts, that contains
the text for a Certificate Request. This is used with a CA to
obtain a signed certificate.

public static java.io.File makeKeyStoreFile()
        - Creates the default .keystore file, assuming that it
does not already exist. Prompts the user to create a key pair.

public static java.io.File getDefaultKeyStoreFile()
        -Check to make sure this file exists.
Returns .keystore file in users home.

public static java.lang.String getAlias(
        KeyStore keyStore) - Given a keystore instance,
provide a multiple choice GUI that enables the user to select
a certificate alias.
```

```
Returns A string version of the certificate alias

public static void printX509Cert(
        X509Certificate cert,
        PrintStream out) - Given an X509 Certificate, print
out all the relevant detail.

public static sun.security.x509.X500Name getX500Name()
        - Prompt the user for all the details needed to
generate a self-signed x500 certificate. This is stored in a
datastructure called the X500Name
Returns X500Name
```

About the author

After receiving his Ph.D. from Rensselaer Polytechnic Institute, **Dr. Lyon** worked at AT&T Bell Laboratories. He has also worked for the Jet Propulsion Laboratory at the California Institute of Technology. He is currently the Chairman of the Computer Engineering Department at Fairfield University, a senior member of the IEEE and President of DocJava, Inc., a consulting firm in Connecticut. E-mail Dr. Lyon at Lyon@DocJava.com. His website is http://www.DocJava.com.