GlobalSign Enterprise Solutions

Server Application to Auto-sign Adobe PDFs from HSM

Eliminating the need to enter the HSM password for each signing

CONTENTS

INTRODUCTION

Companies using an HSM to host certificates used to digitally sign large numbers of PDF documents are faced with the need to enter the HSM password for each signing. This extra step makes applying digital signatures an almost impossible task after a certain volume of documents is reached.

The goal of this document is to outline a solution which will apply digital signatures from a server-mounted HSM to a PDF with the idea of using another form of user authentication instead of HSM passwords. This can be accomplished using either client authentication certificates or website password authentication, or even Active Directory user level security to drive a server-based application interacting with the HSM.

The solution outlined in this paper is based on a certificate loaded in a SafeNet iKey token running on a Windows server and using a Java-based application incorporating the iText® PDF library.

We will show the steps necessary for creating an application which will accept a PDF and, using iText®, sign it with a certificate stored on a Safenet iKey. Since the goal is to do this without entering a password for every signing, we will be incorporating the password into the program and moving the client authentication to the server/webserver level. For the webserver version, this will require a programming language that can run the PDF signing program in the background. For this example, we will use PHP as web language, but this is not mandatory.

REQUIREMENTS:

- SafeNet iKey with Adobe CDS Certificate installed
- Java JDK with Bouncy Castle crypto APIs
- SafeNet iKey SDK
- iText® library
- Eclipse IDE (http://www.eclipse.org/downloads/)
- Windows server

INSTALL REQUIRED SOFTWARE

INSTALL JAVA JDK

Download the Java JDK from http://www.oracle.com/technetwork/java/javase/downloads/index.html. Double click on the file and follow the instructions to install it.

INSTALL BOUNCY CASTLE PROVIDER

Download the Bouncy Castle provider and PKIX files from http://www.bouncycastle.org/latest_releases.html, unzip them, and move the two .jar files to you Java ext folder eg $JAVA_HOME/jre/lib/ext/

Now you need to link to the Bouncy Castle provider from within your Java implementation. To do this, open your java.security file in a text editor:

```
$JAVA_HOME/jre/lib/security/java.security
```

And add the following under your current list of security providers (where N is the next number above the highest number on your current list of providers):

```
security.provider.N=org.bouncycastle.jce.provider.BouncyCastleProvider
```

INSTALL THE SAFENET IKEY SDK

Contact your GlobalSign Account Manager for details on how to obtain the SafeNet iKey SDK. Once you have the file, double click on the SafenetSDK.exe file to install the SDK. This will install all drivers needed to extract the PKCS11 from the SafeNet iKey.

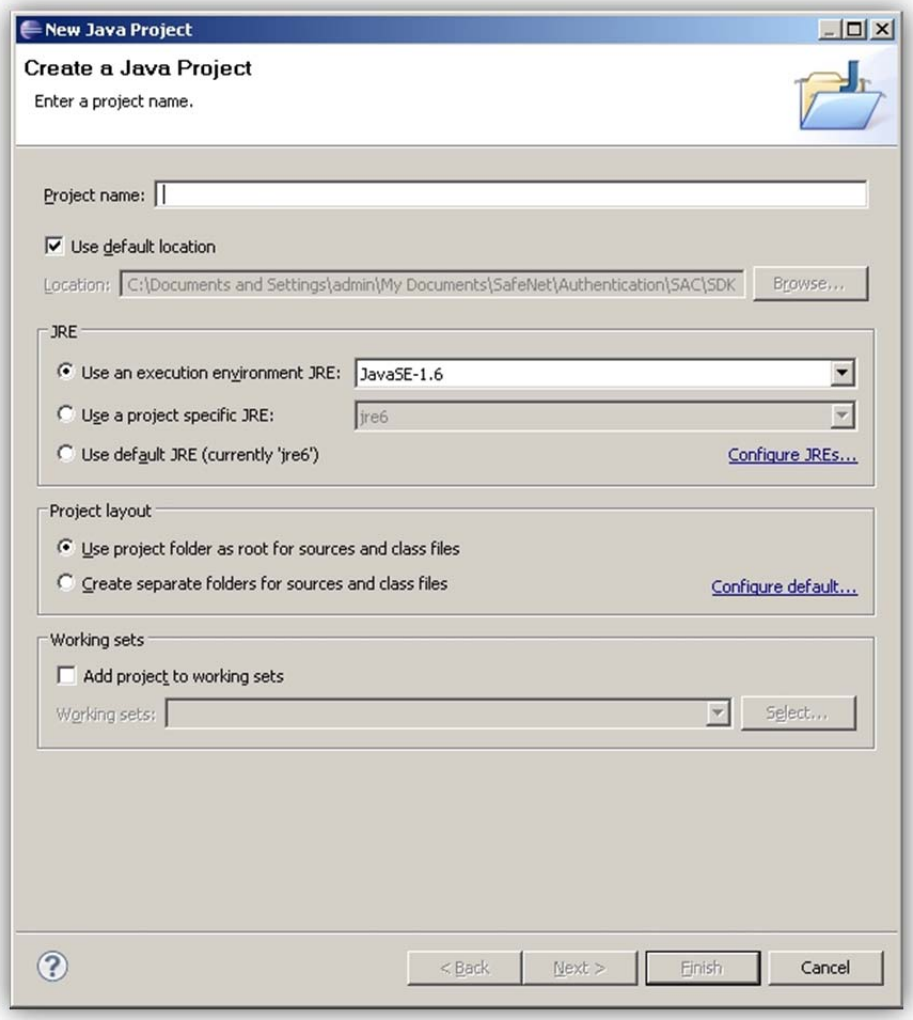
INSTALL THE ECLIPSE IDE FOR JAVA DEVELOPERS

Download the latest Eclipse IDE from http://www.eclipse.org/downloads/ and extract the zip on your system.

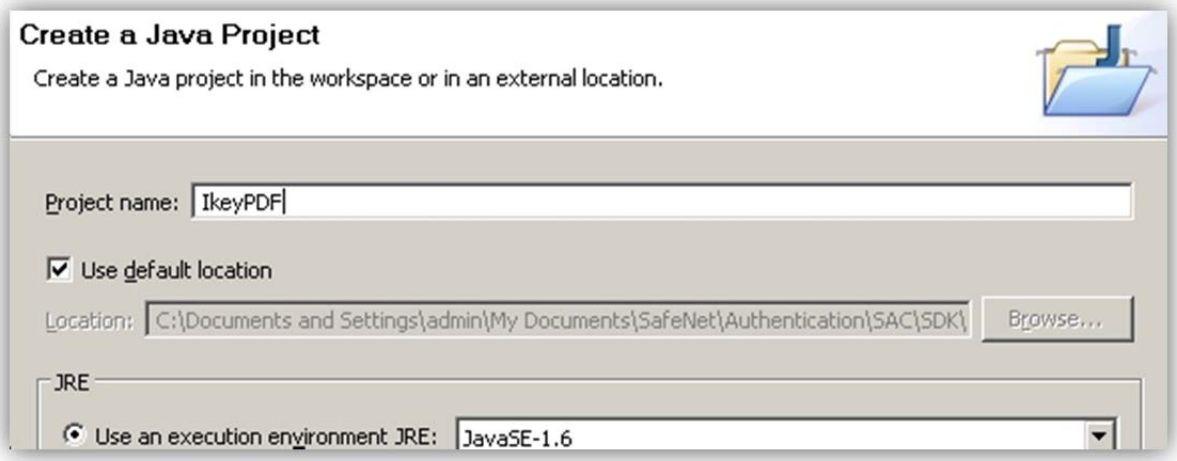
STARTING ON YOUR PROJECT

STEP 1 – CREATE A NEW JAVA PROJECT IN ECLIPSE

From the main Eclipse interface go to *File > New and choose Java Project*.



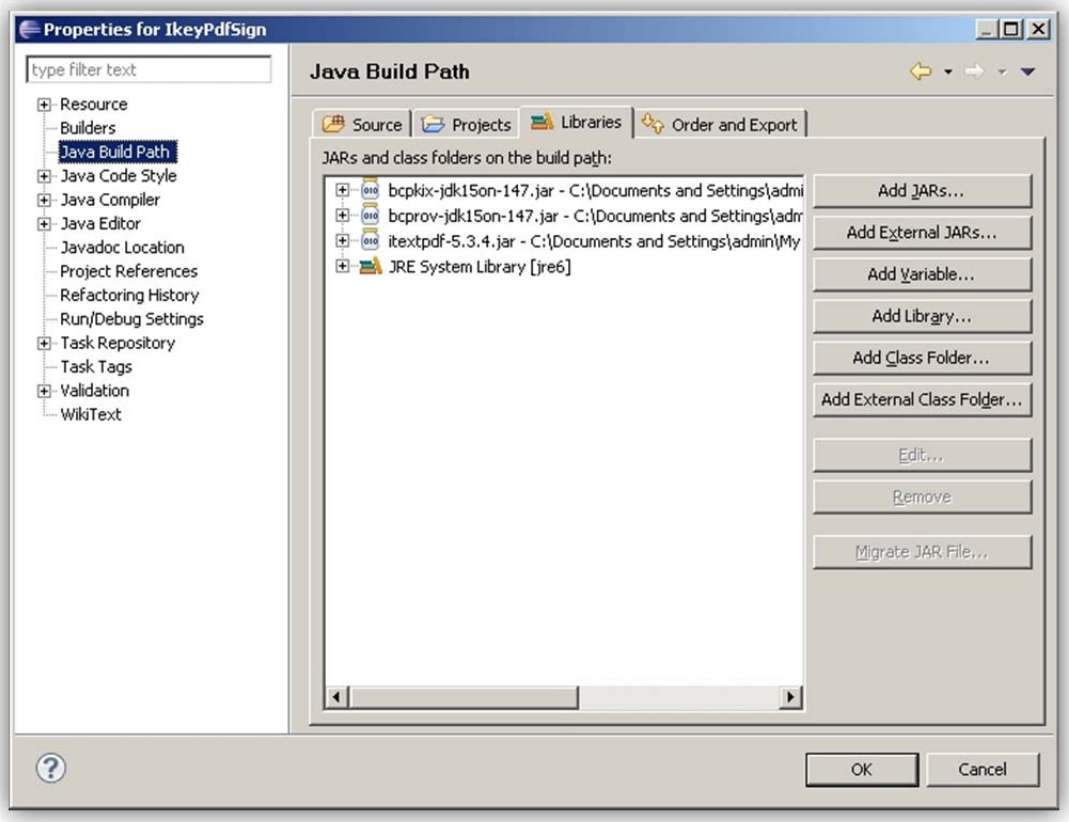
Give the project a name and choose a folder (or use the default location).



STEP 2 – ADD ITEXT AND BOUNCY CASTLE LIBRARIES

In order to add the iText library to the project, you need to download the iText library and extract it to our project folder.

To add the libraries to the project in Eclipse, right click on the project name in the left hand column and select properties. Now, in the window that pops up, select Java Build Path on the left and then Libraries on the right.



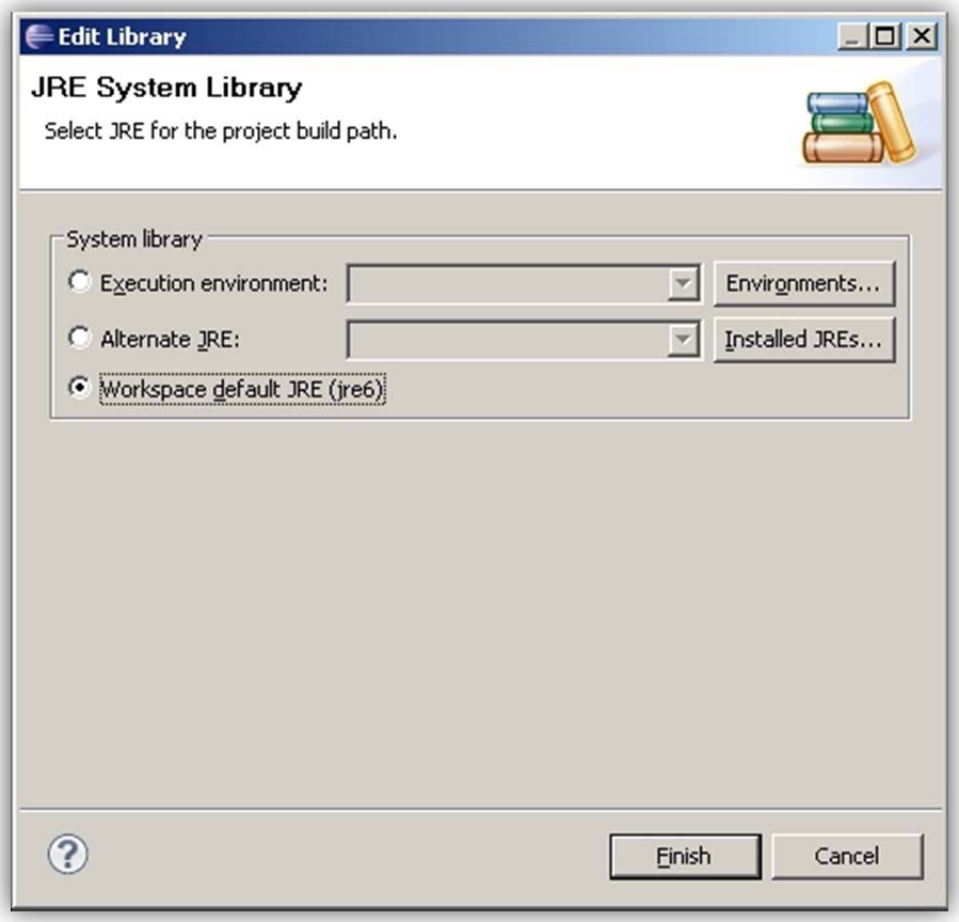
Click on **Add External JARs**.

Repeat this procedure for the two Bouncy Castle and one iText libraries.

Note that by default there will be a JRE System Library in the Libraries tab.

Click on this and select **Edit.**

On the window that pops up, make sure the Workspace default JRE is selected.

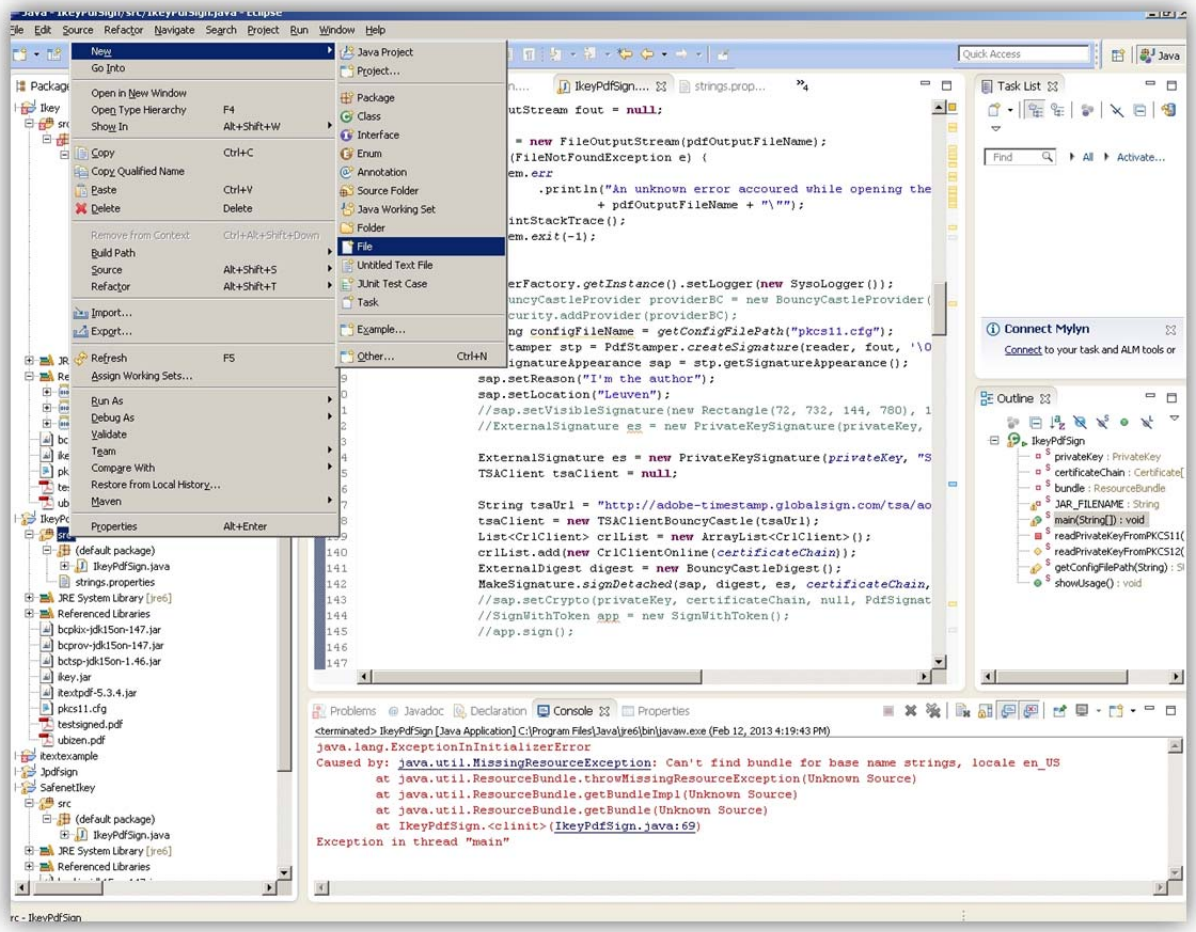


Close the Properties window.

STEP 3 – CREATE APPLICATION SOURCE FILE

In the main Eclipse window, on the left hand column, open the project and go to the **src** folder under it.

Here you need to create a new file. Right click on **src** folder then > New > File. This will now be the application source file.



WRITING YOUR APPLICATION

In this newly created file you will write the application which will initiate all the signing for you. The code has to read the PDF, extract a PKCS11 version of your certificate, sign the PDF, and save that signed version.

READING THE CERTIFICATES FROM THE TOKEN

STEP 1 – CREATE A CONFIG FILE TO SET UP THE CONNECTION TO THE TOKEN

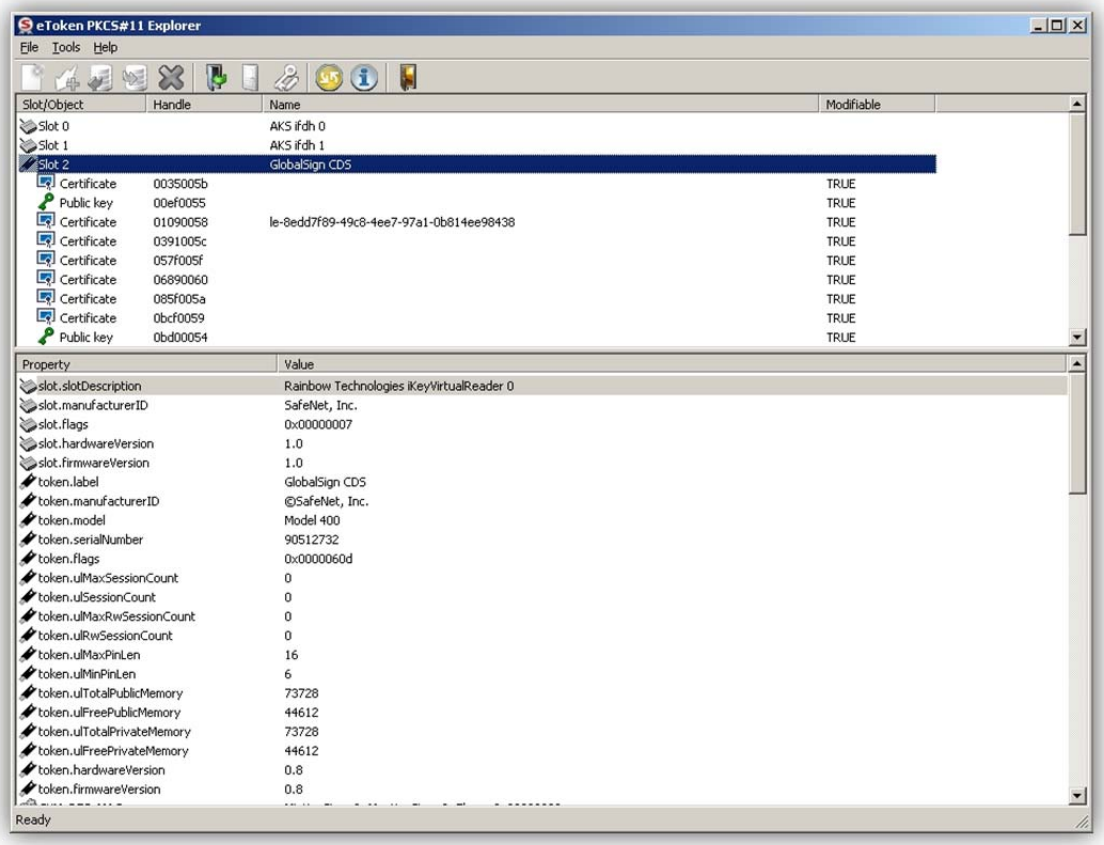
Create a file called pkcs11.cfg in your main directory and add the following:

```
name = PKCS11
library = C:\WINDOWS\system32\eToken.dll
slot=2
```

The top line, name, is concatenated with the prefix SunPKCS11to produce this provider instance's name (that is, the string returned by its `Provider.getName()` method). You will see it referenced below as SunPKCS11-PKCS11 during the signing process.

The second line is a link to eToken.dll library which was installed by the Safenet iKey SDK. This is the full pathname (including extension) of the PKCS#11 implementation.

The last line is the slot where your certificates are stored on your token. You can find the slot number by running the eToken PKCS#11 Explorer from the Safenet iKey SDK (etExplorer under Safenet – SDK from on the Start menu).



STEP 2 – LINK TO THE CONFIG FILE

You can then link to this file from your code for creating a “provider” to use for extracting a PKCS#11 from your token using:

```
String configFileName = getConfigFilePath("pkcs11.cfg");
p = new SunPKCS11(configFileName);
Security.addProvider(p);
```

STEP 3 – LOAD THE KEYSTORE FROM YOUR TOKEN

For this you will need the password, the slot and the provider:

```
String pkcs11PIN = "********";
ks = KeyStore.getInstance("pkcs11", p);
ks.load(null, pkcs11PIN.toCharArray());
```

STEP 4 – FIND THE REQUIRED PRIVATE KEY

Each user certificate on the token can be identified by its alias. You can find the alias for each of the certificates using the aforementioned eToken PKCS#11 Explorer. The alias can be found in the Name column and will generally be a large hexadecimal number eg: le-8edd7f89-49c8-4ee7-97a1-0b814ee98438

```
String alias = "le-8edd7f89-49c8-4ee7-97a1-0b814ee98438";
privateKey = (PrivateKey) ks.getKey(alias, pkcs11PIN.toCharArray());
```

STEP 5 – EXTRACT THE CERTIFICATE AND CERTIFICATE CHAIN

Extract the certificate and chain from the keystore on the token:

```
certificateChain = ks.getCertificateChain(alias);
```

STEP 6 – EMBED REVOCATION INFORMATION

Steps 1 – 5 provide everything required for creating the signature on the PDF. However, for PDF signing it also needs to embed revocation information and a timestamp for proving your certificate was valid at time of signing.

For revocation, you need to embed the CRLs into the PDF together with the signature before you time stamp it. You can read the CRL locations from each of the certificates in the chain:

```
List<CrlClient> crlList = new ArrayList<CrlClient>();
crlList.add(new CrlClientOnline(certificateChain));
```

You can also read the Timestamp services location from the certificates; however, you only need the one from the signing certificate, so you can loop through all the certificates and ignore any which are empty:

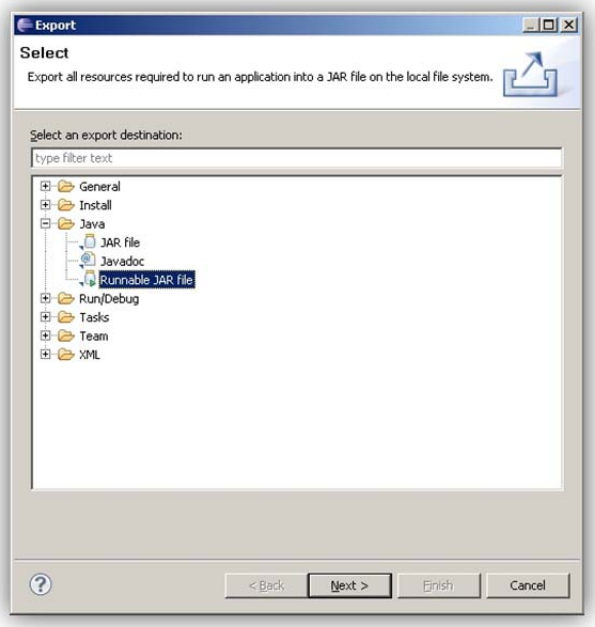
```
if (CertificateUtil.getTSAURL(cert) != null){
        tsaUrl = CertificateUtil.getTSAURL(cert);
}
```

STEP 7 – INITIATE PDF SIGNING

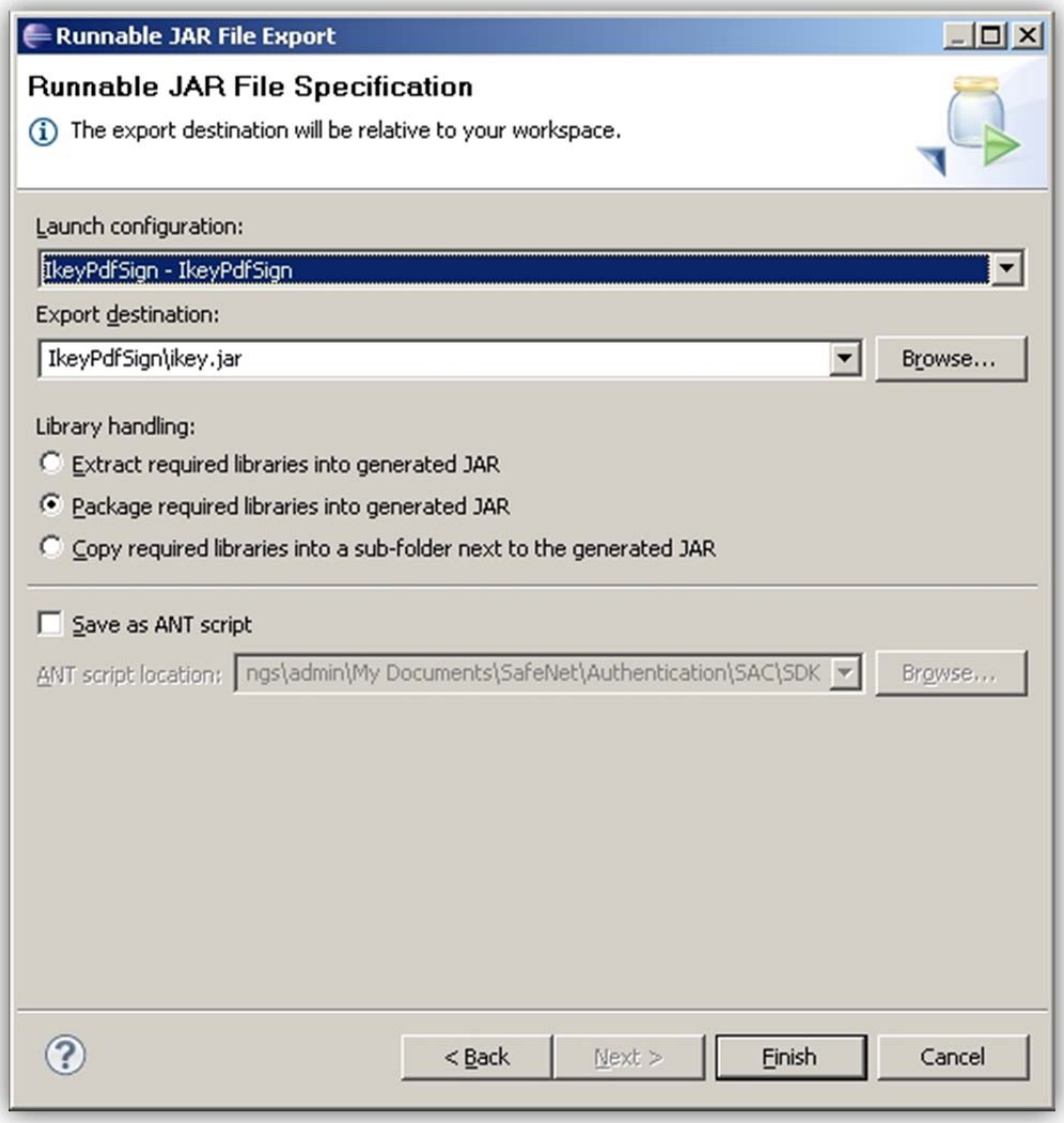
```
LoggerFactory.getInstance().setLogger(new SysoLogger());
PdfStamper stp = PdfStamper.createSignature(reader, fout, '\0');
PdfSignatureAppearance sap = stp.getSignatureAppearance();
sap.setReason("I'm the author");
sap.setLocation("Leuven");
//sap.setVisibleSignature(new Rectangle(72, 732, 144, 780), 1, null);
ExternalSignature es = new PrivateKeySignature(privateKey, "SHA-1", "SunPKCS11-PKCS11");
TSAClient tsaClient = null;
List<CrlClient> crlList = new ArrayList<CrlClient>();
crlList.add(new CrlClientOnline(certificateChain));
tsaClient = new TSAClientBouncyCastle(tsaUrl);
ExternalDigest digest = new BouncyCastleDigest();
MakeSignature.signDetached(sap, digest, es, certificateChain, crlList, null, tsaClient, 0, CryptoStandard.CMS);
```

EXPORTING YOUR APPLICATION AS A RUNNABLE JAR FILE

Right click on your package in the Package Explorer and select **Export**. In the **Export** popup window, open the Java folder and select Runnable JAR file. Click Next.



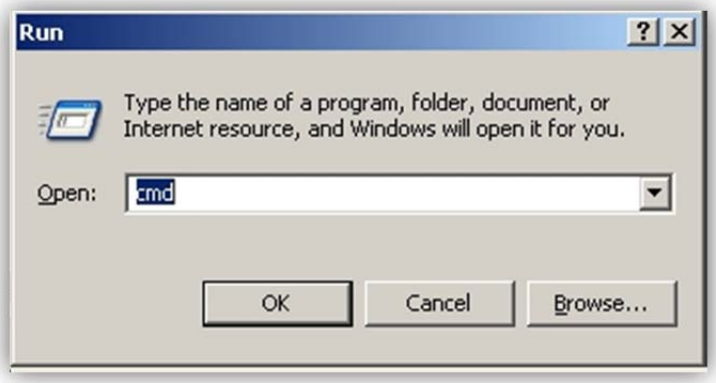
Select the **Launch Configuration** you require and enter your **Export Destination**. Also select the radio button for **Package required libraries into generated JAR**. Now click on **Finish**.

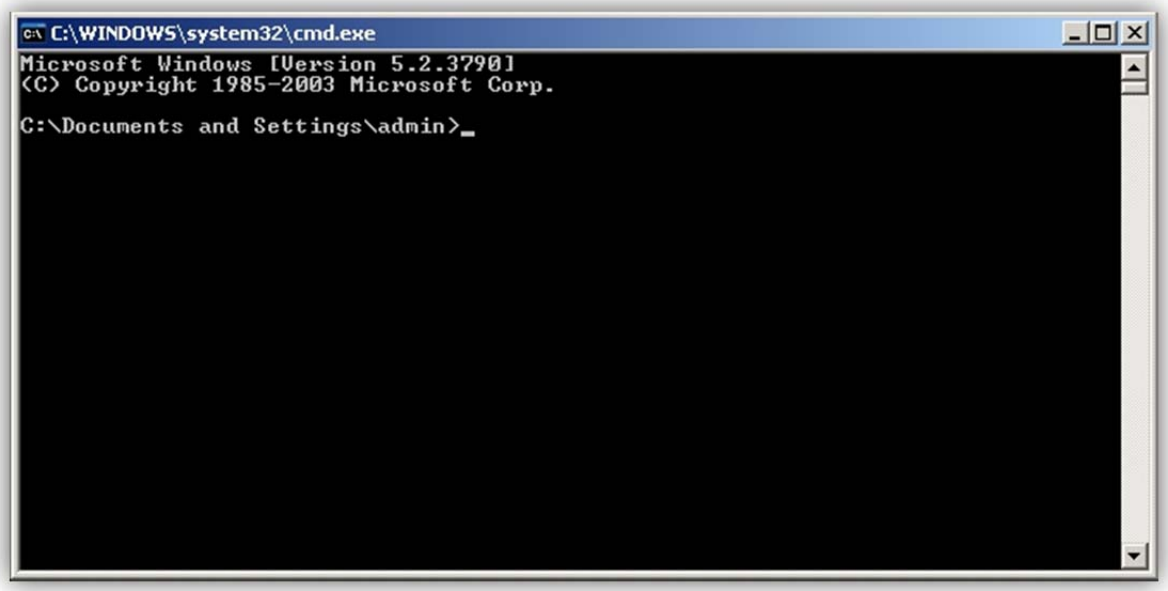


You are now ready to run your application.

TESTING YOUR APPLICATION TO SIGN A PDF DOCUMENT

Open a CMD line interface window by going to the main **Start** button then **Run**. Here enter “cmd” and click on **OK**.





In this window enter the following:

java –jar “\path to\{applicaton-name}.java” “\path to\original.pdf” “\path to\signed.pdf”

NOTE: If there are any spaces in the path or filename, you should put these in quotations, i.e. C:\My Documents\Safenet\IkeyPdfSign.java should be written as C:\”My Documents”\Safenet\IkeyPdfSign.java

Once you have confirmed your application is working you can either set up a webpage to collect the PDF for signing or implement a folder-based script for watching a folder for new PDFs to be signed.

SIGNING VIA A NETWORK SHARE WITH DIRECTORY MONITOR

You may want to implement a network running service where you can drag/drop files for signing. For this all you need is a piece of folder monitoring software (here we are using Directory Monitor as an example) and a simple batch script which will run when anything is added to the particular folder.

First, create a folder on your server (e.g. d:\pdfsigner) and set up a share allowing only those people with signing rights authority to access the folder. NOTE: they will require modify/create rights on this folder.

Inside this folder add another folder called "**result**".

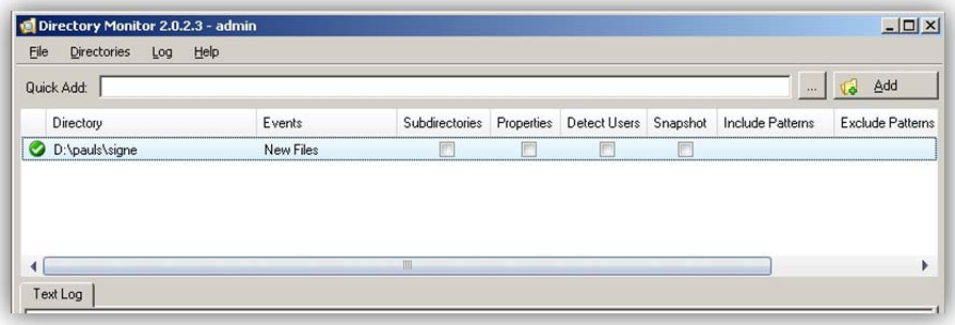
Now open notepad and type the following all on one line:

```
java -jar "\path to\{applicaton-name}\JPdfSign.jar" -PKCS11 %1 "D:\pdfsigner\result\%~n1-signed%~x1"
```

Next save the file somewhere outside this share and name it sign.bat

Now install and open **Directory Monitor**.

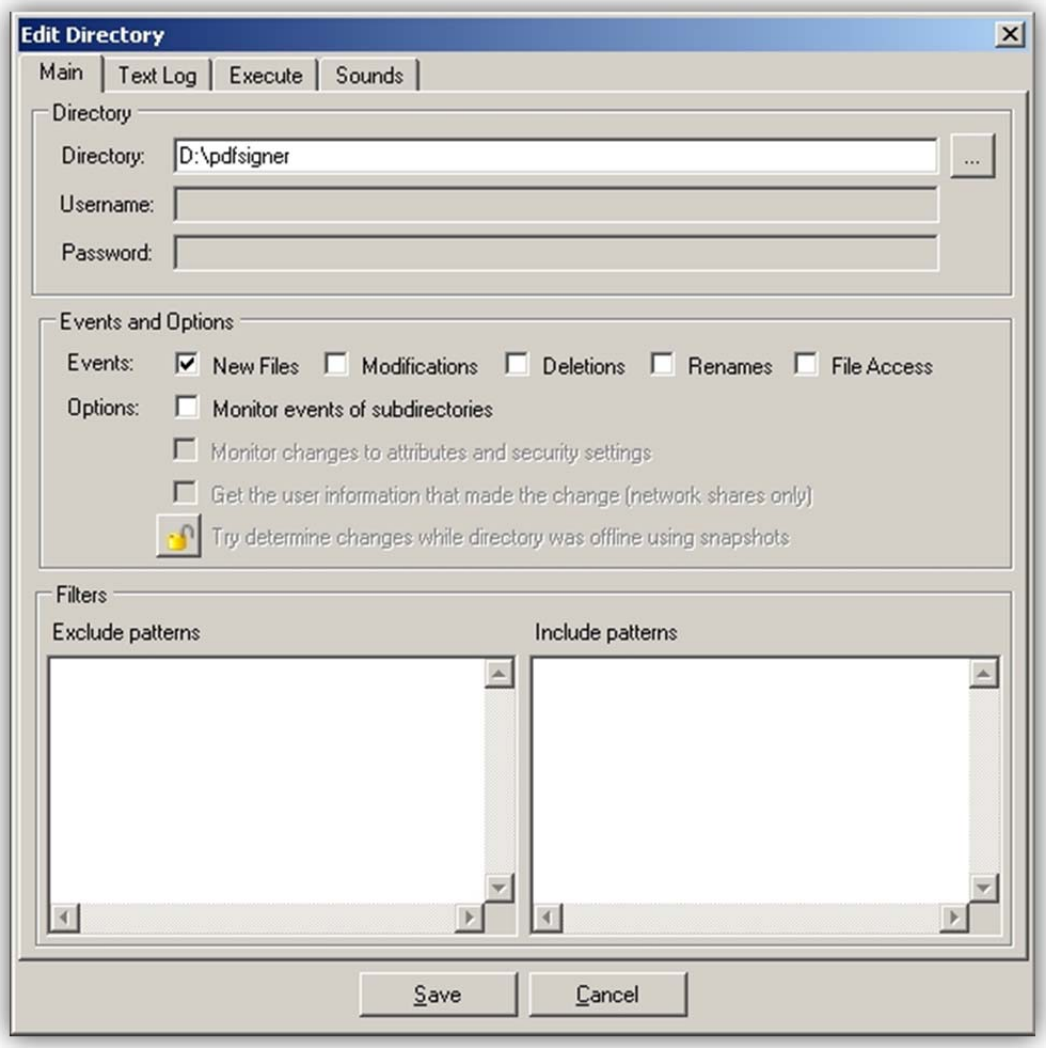
On the main screen click the **Add** button.



On the **Edit Directory** window, type D:\pdfsigner.

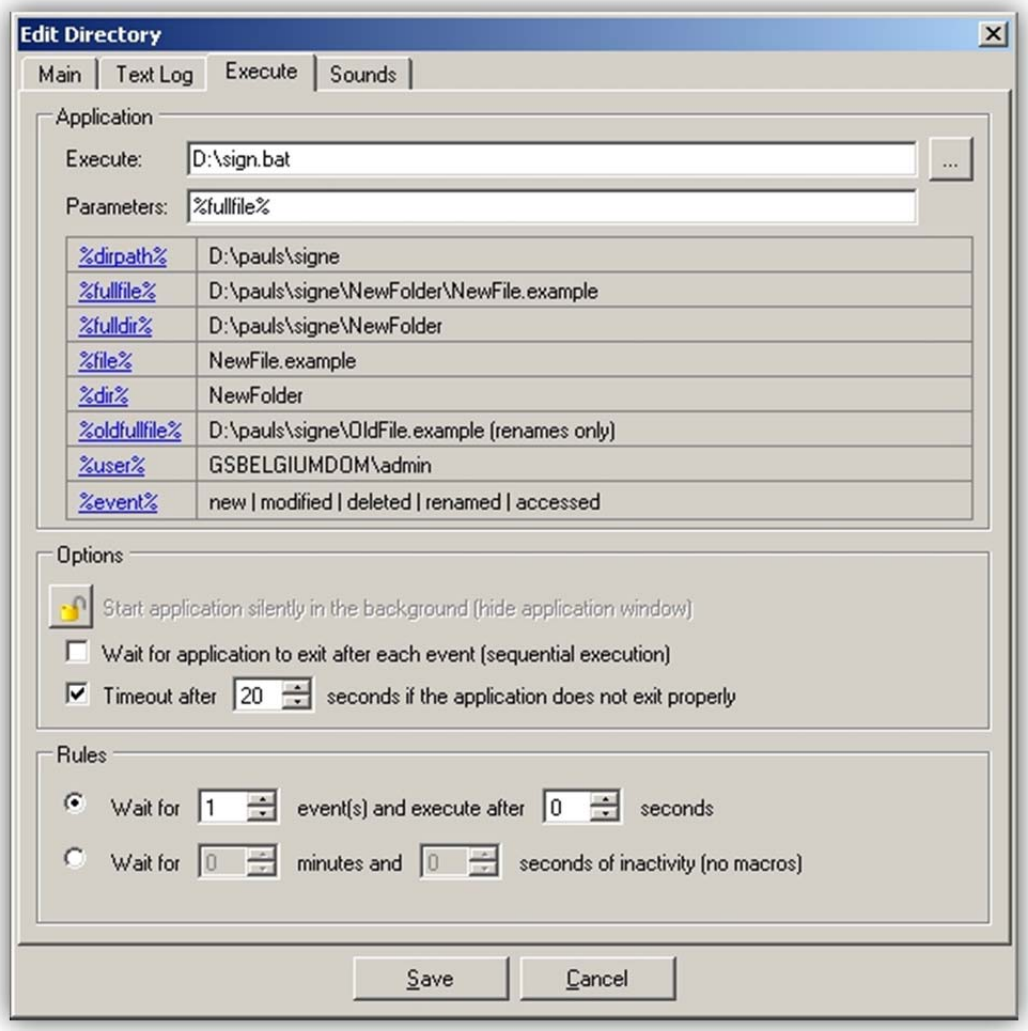
Make sure **New Files** is selected and **Monitor events of subdirectories** is not selected.

On the next tab you can enable a text log if you require logging.



On the **Execute** tab, for the **Execute: field**, enter the full path and name of the sign.bat file you created earlier.

Make sure that the Parameters text box only contains %fullfile%.



Your user should now have the ability to drop files into this folder to be signed (Note that you should be sure that your user is empowered to access this folder). First drop a file in the folder then a couple of seconds later a signed version of the file will be created in the "**result**" folder.

SIGNING VIA WEB-BASED APPLICATION

Signing via a web based application is similar to the folder-monitored solution. The only difference being that you will use a form to upload the PDF files and return the signed PDFs directly back to the user.

First create a form for uploading the PDF file. All that's required here is a form with a file select field.

The contents of this form have to be sent to a file which will run the Java file you created earlier and return the output as a PDF stream to the end user. Using PHP this can be simply achieved with, for example, the following code:

```
<?php

$fname = $_FILES["file"]["name"];

move_uploaded_file($_FILES["file"]["tmp_name"], "upped.pdf");

exec('java -jar "C:\Documents and Settings\user\My Documents\SafeNet\Authentication\SAC\SDK\Jpdfsign\JPdfSign.jar" -
PKCS11 upped.pdf "signed'.$fname.'"');

$fname = 'signed' .str_replace(" "," ",$_FILES["file"]["name"]);

$fname_enc = urlencode($_FILES["file"]["name"]);

header('Content-type: application/pdf');

header('Content-Disposition: attachment; filename="'.$fname.'";');

readfile($fname);

unlink("upped.pdf");

unlink($fname);

?>
```

Note here that you should implement some form of user authentication on this server.

GLOBALSIGN CONTACT INFORMATION

GlobalSign Americas Tel: 1-877-775-4562 www.globalsign.com sales-us@globalsign.com	**GlobalSign EU** Tel: +32 16 891900 www.globalsign.eu sales@globalsign.com	**GlobalSign UK** Tel: +44 1622 766766 www.globalsign.co.uk sales@globalsign.com
GlobalSign FR Tel: +33 1 82 88 01 24 www.globalsign.fr ventes@globalsign.com	**GlobalSign DE** Tel: +49 30 8878 9310 www.globalsign.de verkauf@globalsign.com	**GlobalSign NL** Tel: +31 20 8908021 www.globalsign.nl verkoop@globalsign.com

APPENDIX 1: EXAMPLE CODE LISTING

Combining all the code samples above we should end up with something similar to the following, this will be the finished code which carries out the signing and saves the signed file onto the server filesystem.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URL;
import java.net.URLDecoder;
import java.security.GeneralSecurityException;
import java.security.CodeSource;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.Provider;
import java.security.ProviderException;
import java.security.Security;
import java.security.UnrecoverableKeyException;
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.security.cert.CertificateParsingException;
import java.security.cert.CertificateException;
import java.security.MessageDigest;
import java.util.List;
import java.util.*;

import sun.security.pkcs11.SunPKCS11;

import com.itextpdf.text.*;
import com.itextpdf.text.log.*;
import com.itextpdf.text.pdf.*;
import com.itextpdf.text.pdf.security.*;
import com.itextpdf.text.pdf.PdfReader;
import com.itextpdf.text.pdf.PdfSignatureAppearance;
import com.itextpdf.text.pdf.PdfStamper;
import com.itextpdf.text.pdf.security.CrlClient;
import com.itextpdf.text.pdf.security.DigestAlgorithms;
import com.itextpdf.text.pdf.security.LtvTimestamp;
import com.itextpdf.text.pdf.security.LtvVerification;
import com.itextpdf.text.pdf.security.TSAClientBouncyCastle;
import com.itextpdf.text.pdf.security.TSAInfoBouncyCastle;
import com.itextpdf.text.pdf.PdfSignature;
import com.itextpdf.text.pdf.security.MakeSignature.CryptoStandard;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.tsp.*;
import org.bouncycastle.tsp.cms.*;
import org.bouncycastle.tsp.TimeStampToken;
```

```
import org.bouncycastle.tsp.TimeStampTokenInfo;
import org.bouncycastle.asn1.tsp.*;

public class IkeyPdfSign {
	private static PrivateKey privateKey;
	private static Certificate[] certificateChain;
	private static String tsaUrl;

	public static void main(String[] args) {
		if (args.length < 2)
			showUsage();
		try {
			Security.addProvider(new BouncyCastleProvider());
			String pdfInputFileName = args[0];
			String pdfOutputFileName = args[1];
			System.out.println("");
			System.out.println("pdfInputFileName : " + pdfInputFileName);
			System.out.println("pdfOutputFileName: " + pdfOutputFileName);

			readPrivateKeyFromPKCS11();

			PdfReader reader = null;
			try {

				reader = new PdfReader(pdfInputFileName);
			} catch (IOException e) {
				System.err
					println("An unknown error occurred while opening the input PDF file: \""
					+ pdfInputFileName + "\"");
				e.printStackTrace();
				System.exit(-1);
			}
			FileOutputStream fout = null;
			try {
				fout = new FileOutputStream(pdfOutputFileName);
			} catch (FileNotFoundException e) {
				System.err
					.println("An unknown error occurred while opening the output PDF file: \""
					+ pdfOutputFileName + "\"");
				e.printStackTrace();
				System.exit(-1);
			}
			try {
				LoggerFactory.getInstance().setLogger(new SysoLogger());
				PdfStamper stp = PdfStamper.createSignature(reader, fout, '\0');
				PdfSignatureAppearance sap = stp.getSignatureAppearance();
				sap.setReason("I'm the author");
				sap.setLocation("Leuven");
				//sap.setVisibleSignature(new Rectangle(70, 700, 150, 800), 1, null);
				ExternalSignature es = new PrivateKeySignature(privateKey, "SHA-1", "SunPKCS11-PKCS11");
				TSAClient tsaClient = null;
				List<CrlClient> crlList = new ArrayList<CrlClient>();
				crlList.add(new CrlClientOnline(certificateChain));
				tsaClient = new TSAClientBouncyCastle(tsaUrl);
				ExternalDigest digest = new BouncyCastleDigest();
		MakeSignature.signDetached(sap, digest, es, certificateChain, crlList, null, tsaClient, 0, CryptoStandard.CMS);
			} catch (Exception e) {
```

```
                                System.err.println("An unknown error occurred while signing the PDF file:");
                                e.printStackTrace();
                                System.exit(-1);
                        }
                } catch (KeyStoreException kse) {
                        System.err.println("An unknown error occurred while initializing the KeyStore instance:");
                        kse.printStackTrace();
                        System.exit(-1);
                }
}

private static void readPrivateKeyFromPKCS11() throws KeyStoreException {
                // Initialize PKCS#11 provider from config file
                String configFileName = getConfigFilePath("pkcs11.cfg");

                Provider p = null;
                try {
                        p = new SunPKCS11(configFileName);
                        Security.addProvider(p);
                } catch (ProviderException e) {
                        System.err.println("Unable to load PKCS#11 provider with config file: " + configFileName);
                        e.printStackTrace();
                        System.exit(-1);
                }
                System.out.println(p);
                String pkcs11PIN = "********";
                KeyStore ks = null;
                try {
                        ks = KeyStore.getInstance("pkcs11", p);
                        ks.load(null, pkcs11PIN.toCharArray());
                } catch (NoSuchAlgorithmException e) {
                        System.err.println("An unknown error occurred while reading the PKCS#11 smartcard:");
                        e.printStackTrace();
                        System.exit(-1);
                } catch (CertificateException e) {
                        System.err.println("An unknown error occurred while reading the PKCS#11 smartcard:");
                        e.printStackTrace();
                        System.exit(-1);
                } catch (IOException e) {
                        System.err.println("An unknown error occurred while reading the PKCS#11 smartcard:");
                        e.printStackTrace();
                        System.exit(-1);
                }

                String alias = "le-8edd7f89-49c8-4ee7-97a1-0b814ee98438";
                try {
                        privateKey = (PrivateKey) ks.getKey(alias, pkcs11PIN.toCharArray());
                } catch (NoSuchElementException e) {
                        System.err.println("An unknown error occurred while retrieving the private key:");
                        System.err.println("The selected token does not contain any private keys.");
                        e.printStackTrace();
                        System.exit(-1);
                } catch (NoSuchAlgorithmException e) {
                        System.err.println("An unknown error occurred while retrieving the private key:");
                        e.printStackTrace();
                        System.exit(-1);
                } catch (UnrecoverableKeyException e) {
```

```
                        System.err.println("An unknown error occurred while retrieving the private key:");
                        e.printStackTrace();
                        System.exit(-1);
                }
                certificateChain = ks.getCertificateChain(alias);
                Certificate[] chain = ks.getCertificateChain(alias);
                for (int i =0; i < chain.length; i++){
                        X509Certificate cert = (X509Certificate)chain[i];
                        try {
                                System.out.println(CertificateUtil.getCRLURL(cert));
                                System.out.println(CertificateUtil.getTSAURL(cert));
                                if (CertificateUtil.getTSAURL(cert) != null){
                                        tsaUrl = CertificateUtil.getTSAURL(cert);
                                }
                        } catch (CertificateParsingException e) {
                                e.printStackTrace();
                        }
                }
        }

        protected static String getConfigFilePath(String configFilename) {
                CodeSource source = IkeyPdfSign.class.getProtectionDomain().getCodeSource();
                URL url = source.getLocation();

                String jarPath = URLDecoder.decode(url.getFile());
                File f = new File(jarPath);
                try {
                        jarPath = f.getCanonicalPath();
                } catch (IOException e) {
                }
                if (!f.isDirectory()) {
                        f = new File(jarPath);
                        jarPath = f.getParent();
                }
                System.out.println(jarPath);
                if (jarPath.length() > 0) {
                        return jarPath + File.separator + configFilename;
                } else
                        return configFilename;
        }

        public static void showUsage() {
                System.out.println("Usage: java -jar ikey.jar pdfInputFileName pdfOutputFileName");
                System.exit(0);
        }
}
```