Document Control

Title Block	
Title	Code Signing Certificate Generation Technical Instructions
Description	Tactical Code Signing Instructions for Generating Certificate and Installing Certificate into JRE Trust Store
Version	0.2
Issue Status	Draft
Author	David Wozny
Customer Organisation	CBS

Change Record

Issue No.	Date	Issued By	Reason for Issue
0.1	14/05/2013	David Wozny	First draft
0.2	20/05/2013	David Wozny	Updated introduction, including alternative approaches considered

Table of Contents

1. Introduction

1.1. Background

Unipel (Sovereign) generate and maintain Java code in the form of JAR files for the CHS and CAD Maps applications – this code is developed *in-house*.

The requirement for signing JAR files is to enable strong JRE security policy to be enforced such that only signed (and trusted) JAR files can be executed, thus limiting the vulnerability to malicious code.

As well as signing code in the production Foundation environment, it is also necessary to sign code in reference and pre-production environments.

There is no requirement for signed JAR files to be trusted outside of the Foundation estate; it is recognised that CBS are not a software development house and the code signing solution implemented should not be over-engineered and relative to the requirement.

1.2. Alternatives Considered

1.2.1. Introduction

There are three principal options available to satisfy code signing requirements:

- Utilise ePKI (e.g. SubCA1, SubCAx)
- Purchase from external trust service (e.g. VeriSign)
- Use self-signed certificates

1.2.2. Utilise ePKI

The ePKI does not support any certificate profiles which are suitable for issuing code signing certificates. Whilst technically, it would be relatively simple to setup a registration policy to issue code signing certificates, from a policy perspective there would be a difficult choice of whether to utilise a lightweight or a comprehensive approach.

A lightweight approach would be a sop, and not in the spirit of ePKI. A comprehensive approach would require consideration of matters such as time-stamping and therefore likely to be both expensive and complex to implement - perhaps even requiring an entire new SubCA so that existing certificate policy on the SubCA1 / SubCA2 isn't compromised.

1.2.3. Purchase from VeriSign

Purchasing a code signing certificate from a third party such as VeriSign would enable CBS to acquire a fully legitimate and comprehensive code signing capability, which would also incorporate a time-stamping utility allowing signed code to be valid indefinitely. There would be a cost of approximately £2,500 (each) to purchase relevant certificates from VeriSign and it is likely that both the Oracle CBSHR team and Unipel Sovereign teams would require separate certificates given some of the technical environment constraints involved in their signing processes.

Perhaps the biggest issue with VeriSign code signing capability is the fact that CBS would need to be registered as the owner of the certificate and exposure / leakage of the certificate[1] would reflect extremely badly on CBS if it was used maliciously. Given that the code signing certificate would need to be in possession of the relevant support team there would need to be extremely severe Acceptable Usage Policy implemented as it is hard to imagine suitable technical controls could be implemented to meet the balance of security and operational availability.

[1] Specifically the private key – but the term certificate is used for simplicity of understanding

An alternative may be for Capgemini to register the CBSHR code signing certificate and Unipel the Sovereign certificate, therefore limiting the potential exposure of CBS. However, this may simply open up a can of worms that would take an inordinate amount of time to come to any satisfactory conclusion.

1.2.4. Self-Signed Certificates

This approach would require both Unipel and Sovereign creating self-signed code signing certificates following a precise, auditable process. The certificates would have very long lifetimes (perhaps ten years) so that signature expiry does not become a problem.

The self-signed certificates would be published into the relevant Foundation active directory (live, pre-prod or ref) which would then propagate the certificates down to workstations, thus enabling any code signed by these certificates to be trusted. In the event that there was a new requirement to not trust any code signed by these certificates, removal of the certificates from Active Directory would trigger the removal of the certificates from workstations and thus terminating the trust.

Whilst this sounds an unorthodox approach, it has significant merits:

- It would be very inexpensive and quick to implement
- It is low-tech and simple to maintain
- It entirely removes any risk of exposure of code signing certificates externally
- It satisfies the immediate requirement of meeting JRE code signing enforcement

The downside is that the certificate is explicitly trusted in the CBS estate, this is something that would usually only be done for a PKI that had comprehensive security policy and technical controls implemented. As a minimum, it would be necessary to ensure that the certificate had extremely limited (precise) key usage and could not be used to sign server certificates, etc. - beyond that, the risk is rather limited.

1.3. PMA Decision

The CBS PMA made the decision to use the third approach (self-signed certificates) with a ten-year lifetime – a suitable certificate profile was created. The use of self-signed certificates for code signing will continue indefinitely until such time as a new requirement arises.

1.4. Document Scope

1.4.1. In Scope

- Meeting code signing requirements for Unipel Sovereign
- Creation of the code signing certificate (and corresponding java key store) in a controlled environment
- Distribution of trust of the code signing certificate in the Foundation estate

1.4.2. Out of Scope

- Distribution of the code signing certificate (and corresponding trust stores) to Sovereign
- Use of the code signing certificate for signing java code
- Control over storage of the code signing certificate (and passwords)

2. Workstation Preparation

2.1. Operating System

Windows 7 Professional Enterprise (32-bit) with Service Pack 1 applied and current with all security hotfixes and patches.

Set a strong password for the Administrator account and record it suitably

2.2. Preparation

Create the following folders:

- `C:\Work`
- `C:\Software`

Insert the *Code Signing Engineering CD* into the workstation

Copy software from *engineering CD* into `C:\Software`

Copy scripts from *engineering CD* into `C:\Work`

Remove the Code Signing Engineering CD from the workstation

Include the following in the system path:

- `C:\Program Files\Java\jre7\bin`
- `C:\OpenSSL-Win32\bin`

2.3. Install Software

2.3.1. .Net Framework v4

Run the following installer: `C:\Software\dotNetFx40_Full_x86_x64.exe`

2.3.2. PowerShell v3.0

Run the following installer: `C:\Software\Windows6.1-KB2506143-x86.msu`

2.3.3. Sun Java Runtime Environment (JRE) v1.4.2 Update 19

Run the following installer: `C:\Software\j2re-1_4_2_19-windows-i586-p.exe`

2.3.4. Sun Java Development Kit (JDK) v7 Update 17

Run the following installer: `C:\Software\jdk-7u17-windows-i586.exe`

Note: The distribution includes Sun Java Runtime Environment (JRE)

2.3.5. Visual C++ Redistributables

Run the following installer: `C:\Software\vcredist_x86.exe`

Note: Required for installation of OpenSSL for Windows

2.3.6. OpenSSL for Windows

Run the following installer: `C:\Software\Win32OpenSSL_Light-1_0_1e.exe`

2.4. Set PowerShell Execution Policy

Run Windows PowerShell as administrator

Run the following command: **`set-executionpolicy remotesigned`**

Type [`Y`] to confirm the change

Run the following command: **`get-executionpolicy`**

Observe the executionpolicy is set to: `remotesigned`

3. Create Certificates in Crypto API (CAPI)

3.1. Create PFX File Using PowerShell

Run Windows PowerShell and change directory to `C:\Work`

Run the following command:

```
.\New-SelfSignedCertificateEx.ps1 -subject
"CN=Unipel_Sovereign,OU=CBS,O=CBS_Police,C=GB" -AlgorithmName RSA -
KeyLength 2048 -SignatureAlgorithm sha1 -EKU "Code Signing" -KeySpec
"Signature" -KeyUsage "DigitalSignature" -FriendlyName
"Unipel_Sovereign" -SerialNumber 1 -Path C:\work\Unipel_sovereign.pfx
-Exportable -NotAfter ([datetime]::now.addyears(10))
```

When prompted, enter a password for the PFX file once, and again to confirm it

Record the password as **Unipel_Sovereign Code Signing PFX File Password (e.g. *red*)**

Observe that the file `C:\work\Unipel_sovereign.pfx` is created

3.2. Export Certificate from CAPI User Store

Open an instance of certificate management MMC snap-in, focussed on the user certificate store by running `certmgr.msc`.

Browse to the personal store and select the Unipel_Sovereign certificate, export it to the following location:

`C:\work\Unipel_sovereign.cer`

Note: Choose to not export the private key; choose base-64 encoded format

Once exported, rename the file as: `C:\work\Unipel_sovereign.crt`

4. Convert PFX File to PKCS#12 (via PEM)

4.1. Use Open SSL to Convert PFX File to PEM

Open a DOS command prompt and change directory to `C:\Work`, then run the following command:

```
openssl pkcs12 -in Unipel_sovereign.pfx -out Unipel_sovereign.pem
```

Enter the import password (Unipel_Sovereign Code Signing PFX File Password, e.g. red)

Specify and confirm a PEM pass phrase

Record the password as Unipel_Sovereign Code Signing PEM File Password (e.g. blue)

4.2. Use Open SSL to Convert PEM File to PKCS#12

Run the following command:

```
openssl pkcs12 -export -in Unipel_sovereign.pem -out
Unipel_sovereign.p12 -name CBS
```

Note: Here "CBS" is the name of the alias to be used in the Java Key Store

Enter the PEM pass phrase (Unipel_Sovereign Code Signing PEM Password, e.g. blue)

Specify and confirm an Export Password

Note: Ignore message "unable to write random state"

Record the password as Unipel_Sovereign Code Signing PKCS#12 File Password (e.g. green)

Note: The "Export Password" is recognised as the Key passphrase when using JarSigner

5. Create Key Store using Java Keytool

5.1. Verify PKCS#12 File is OK

Open a DOS command prompt and change directory to `C:\Work`, then run the following command:

`keytool -v -list -keystore Unipel_sovereign.p12 -storetype pkcs12`

When prompted for keystore password, use Unipel_Sovereign Code Signing PKCS#12 File Password (e.g. green)

Truncated Output:

```
Your keystore contains 1 entry
Alias name: CBS
Creation date: 07-May-2013
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Unipel_Sovereign, OU=CBS, O=CBS_Police, C=GB
Issuer: CN=Unipel_Sovereign, OU=CBS, O=CBS_Police, C=GB
Serial number: 1
Valid from: Tue May 07 15:06:04 BST 2013 until: Sun May 07 15:06:03 BST 2023
```

5.2. Import PKCS#12 File into Java Key Store

Run the following command:

`keytool -importkeystore -srckeystore Unipel_sovereign.p12 -destkeystore Unipel_sovereign.jks -srcstoretype pkcs12`

When prompted for a destination keystore password create a new one and confirm

Record the password as Unipel Sovereign Code Signing Java Keystore Password (e.g. yellow)

When prompted for a source keystore password enter the Unipel Sovereign Code Signing PKCS#12 File (e.g. green)

Note: The "Java Keystore Password" is recognised as the Store passphrase when using JarSigner

5.3. Inspect Contents of Java Key Store

Run the following command:

`keytool -list -v -keystore Unipel_sovereign.jks`

When prompted, enter the Unipel Sovereign Code Signing Java Keystore Password (e.g. yellow)

Truncated Output:

```
Enter keystore password:
Alias name: CBS
Creation date: 07-May-2013
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Unipel_Sovereign, OU=CBS, O=CBS-Police, C=GB
Issuer: CN=Unipel_Sovereign, OU=CBS, O=CBS-Police, C=GB
Serial number: 1
```

6. Import New Code Signing Certificate into Existing Trusted Certificate Store

6.1. Retrieve Existing cacerts File

Extract the following file from a Foundation workstation:

C:\Program Files\Java\j2re1.4.2_19\lib\security\cacerts

Copy the file to the `C:\work` folder on the workstation

6.2. Verify Pass Phrase for Existing cacerts JKS

Note: Manipulation of the cacerts JKS must be done using the JRE 1.4.2_19 version of keytool

Open a DOS command prompt and change directory to `C:\Work`, then run the following command:

"C:\Program Files\Java\j2re1.4.2_19\bin\keytool" –list -keystore cacerts

Enter the keystore passphrase (**changeit**)

Note: This is an existing password that is universally recognised

6.3. Import Code Signing Certificates into JKS

Run the following command:

"C:\Program Files\Java\j2re1.4.2_19\bin\keytool" -keystore cacerts -import -alias CBS -file Unipel_sovereign.crt

Enter the keystore passphrase (**changeit**)

When prompted, type "`yes`" to trust the certificates

6.4. Verify Unipel_Sovereign Certificate is Imported

Run the following command:

"C:\Program Files\Java\j2re1.4.2_19\bin\keytool" -list -v -keystore cacerts

Enter the keystore passphrase (**changeit**)

Search the output for the alias "`CBS`", this should show the `Unipel_Sovereign` certificate

6.5. Distribute Updated cacerts JKS File

It will be necessary to distribute the updated cacerts JKS file to the following location on Foundation computers:

"C:\Program Files\Java\j2re1.4.2_19\lib\security"

Appendix 1

Passwords Registered

- Unipel Sovereign Code Signing PFX File Password
- Unipel Sovereign Code Signing PEM File Password
- Unipel Sovereign Code Signing PKCS#12 File Password
  Utilised as the Key passphrase when using JarSigner
- Unipel Sovereign Code Signing Java Keystore Password
  Utilised as the Store passphrase when using JarSigner
- JRE 1.4.2_19 cacerts file
  Existing password: changeit