How to create an app package signing certificate

Important MakeCert.exe is deprecated. For current guidance on creating a certificate, see **Create a certificate for package signing**.

Learn how to use **MakeCert.exe** and **Pvk2Pfx.exe** to create a test code signing certificate, so that you can sign your Windows Store app packages.

You must digitally sign your Windows Store apps before you deploy them. If you don't use Microsoft Visual Studio 2012 to create and sign your app packages, you need to create and manage your own code signing certificates. You can create certificates by using **MakeCert.exe** and **Pvk2Pfx.exe** from the Windows Driver Kit (WDK). Then you can use the certificates to sign the app packages, so they can be deployed locally for testing.

What you need to know

Technologies

- Introduction to Code Signing
- App packages and deployment
- Tools for Signing Drivers

Prerequisites

- **MakeCert.exe** and **Pvk2Pfx.exe** tools from the WDK

Instructions

Step 1: Determine the publisher name of the package

To make the signing certificate that you create usable with the app package that you want to sign, the subject name of the signing certificate must match the **Publisher** attribute of the **Identity** element in the AppxManifest.xml for that app. For example, suppose the AppxManifest.xml contains:

```
<Identity Name="Contoso.AssetTracker"
  Version="1.0.0.0"
  Publisher="CN=Contoso Software, O=Contoso Corporation, C=US"/>
```

For the *publisherName* parameter that you specify with the **MakeCert** utility in the next step, use "CN=Contoso Software, O=Contoso Corporation, C=US".

Note This parameter string is specified in quotes and is both case and whitespace sensitive.

The **Publisher** attribute string that is defined for the **Identity** element in the AppxManifest.xml must be identical to the string that you specify with the **MakeCert** /n parameter for the certificate subject name. Copy and paste the string where possible.

Step 2: Create a private key using MakeCert.exe

Use the **MakeCert** utility to create a self-signed test certificate and private key:

```
MakeCert /n publisherName /r /h 0 /eku "1.3.6.1.5.5.7.3.3,1.3.6.1.4.1.311.10.3.13" /e
expirationDate /sv MyKey.pvk MyKey.cer
```

This command prompts you to provide a password for the .pvk file. We recommend that you choose a **strong password** and keep your private key in a secure location.

We recommend that you use the suggested parameters in the preceding example for these reasons:

/r

Creates a self-signed root certificate. This simplifies management for your test certificate.

/h 0

Marks the basic constraint for the certificate as an end-entity. This prevents the certificate from being used as a Certification Authority (CA) that can issue other certificates.

/eku

Sets the Enhanced Key Usage (EKU) values for the certificate.

> **Note** Don't put a space between the two comma-delimited values.

- 1.3.6.1.5.5.7.3.3 indicates that the certificate is valid for code signing. Always specify this value to limit the intended use for the certificate.
- 1.3.6.1.4.1.311.10.3.13 indicates that the certificate respects lifetime signing. Typically, if a signature is time stamped, as long as the certificate was valid at the point when it was time stamped, the signature remains valid even if the certificate expires. This EKU forces the signature to expire regardless of whether the signature is time stamped.

/e

Sets the expiration date of the certificate. Provide a value for the *expirationDate* parameter in the mm/dd/yyyy format. We recommend that you choose an expiration date only as long as necessary for your testing purposes, typically less than a year. This expiration date in conjunction with the lifetime signing EKU can help to limit the window in which the certificate can be compromised and misused.

For more info about other options, see **MakeCert**.

Step 3: Create a Personal Information Exchange (.pfx) file using Pvk2Pfx.exe

Use the **Pvk2Pfx** utility to convert the .pvk and .cer files that **MakeCert** created to a .pfx file that you can use with **SignTool** to sign an app package:

```
Pvk2Pfx /pvk MyKey.pvk /pi pvkPassword /spc MyKey.cer /pfx MyKey.pfx [/po pfxPassword]
```

The *MyKey.pvk* and *MyKey.cer* files are the same files that **MakeCert.exe** created in the previous step. By using the optional /po parameter, you can specify a different password for the resulting .pfx; otherwise, the .pfx has the same password as *MyKey.pvk*.

For more info about other options, see **Pvk2Pfx**.

Remarks

After you create the .pfx file, you can use the file with **SignTool** to sign an app package. For more info, see **How to sign an app package using SignTool**. But the certificate is still not trusted by the local computer for deployment of app packages until you install it into the trusted certificates store of the local computer. You can use **Certutil.exe**, which comes with Windows.

To install certificates with WindowsCertutil.exe

1. Run **Cmd.exe** as administrator.
2. Run this command:

```
Certutil -addStore TrustedPeople MyKey.cer
```

We recommend that you remove the certificates if they are no longer in use. From the same administrator command prompt, run this command:

```
Certutil -delStore TrustedPeople certID
```

The **certID** is the serial number of the certificate. Run this command to determine the certificate serial number:

```
Certutil -store TrustedPeople
```

Security Considerations

By adding a certificate to **local machine certificate stores**, you affect the certificate trust of all users on the computer. We recommend that you install any code signing certificates that you want for testing app packages to the Trusted People certificate store. Promptly remove those certificates when they are no longer necessary, to prevent them from being used to compromise system trust.

Related topics

Samples
Create app package sample

Concepts

Code-Signing Best Practices

How to sign an app package using SignTool

Signing an app package