DIGITAL SUNDHED

SAMMENHÆNGENDE DIGITAL SUNDHED I DANMARK

CAPI PKI Plugin Guide

Revision History

Version	**Date**	**By**	**Comments**
1	06/01/10	Brian Graversen	Initial release of document
2	18/03/10	Brian Graversen	Added requirement for the DanID CSP to the documentation

Introduction

The CAPI PKI Plugin is an authentication plugin for the SignOn Library framework, that exposes the PKI interface through the application context to other plugins. Read the document “Sign-On projektet: A national architecture for SignOn” for details on entire SignOn Library framework.

Purpose of the plugin

This plugin, together with the SignOn Library framework, allows client applications to access the users certificates stored in the local machines CAPI store – the CAPI store is the Windows OS keystore, where certificates and keys are securely stored, and can be accessed through the Microsoft Cryptographic API. This plugin enables this access through the SignOn Library Framework, so no direct access to CAPI is needed by the client application.

Other authentication plugins exists for mobile signature solutions, and this plugin should only be used when access to locally installed certificates is needed.

Quick-guide: Using the plugin

We assume that the reader has read the document “Using the SignOn library”, and knows how to configure and call the SignOn method. This section will describe only the settings that are relevant for the CAPI PKI Plugin, and will show sample code in C#. If the SignOn Library is used from other languages, the language-specific changes should be made to the code below before using it.

```
// Create a SignOnRequest instance (additional configuration of the request is left out)
ISignOnRequest request = new SignOnRequest();

// Add the CAPI PKI Plugin - if the plugin is located in a subfolder, then the
// relative path from the client binary to the plugin dll should be included in
// the first argument. The second argument is the timeout value (see the Timeout
// section for details) and the second is the continue-on-error setting, which
// can be set to true/false depending on the vendors choice.
request.AddPlugin(@"CAPIPKIPlugin.dll", 2000, false);

// The only credential that the CAPI PKI Plugin requires is the users password.
// This password should reflect the password used to access the certificate, not
// the password used to logon to the application or Windows (though they could possible
// be the same).
request.Password = "Test1234";
```

This is the only plugin specific configuration that is needed to use the CAPI PKI Plugin. Any additional configuration should then be applied to the SignOnRequest instance, and the SignOn method can then safely be called on the SignOnFacade instance.

Flow Handling

The SignOn Library allows authentication plugins to pass errors regarding the authentication step, and the CAPI PKI Plugin makes use of this feature. The following "errors" can occur in this plugin, and should be handled by the calling application.

SignOnResponse StatusCode	Reason
NoCertificatesError	The user has no certificates in CAPI, or all the certificates in CAPI available to the current user are expired, revoked or otherwise invalid.
MultipleCertificatesError	The user has more than one valid certificate in CAPI. See the section on handling multiple certificates below.

InvalidCertificateError	The SignOnRequest has the CertificateChoice property set, but the value does not match any certificates available in CAPI. Check your code to ensure that you set the property correctly.
InvalidCredentialsError	The Password field in the SignOnRequest does not match the password required by the certificate stored in CAPI. Ask the user to retype his or her password, and try again.

The only status code that requires special handling is the MultipleCertificatesError, and the following full C# sample code should show how this flow could be handled – note that user-interaction is required as part of this flow, where the user needs to decide which of the certificates he wishes to use.

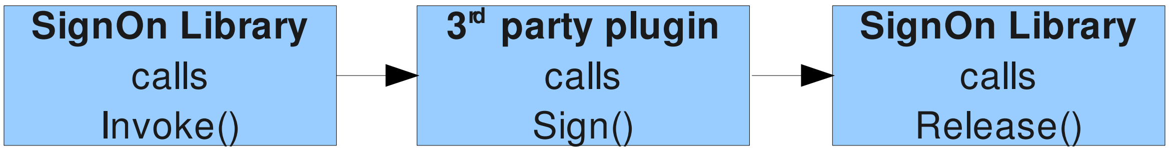
Handling multiple certificates

When the returned status code is MultipleCertificatesError, then the response field CertificateChoices contains an array of possible choices. The CertificateChoice class contains two fields, a pretty-name to be displayed to the user, and an internal identifier, which is not used by the client application at all.

```
if (response.Status == StatusCodes.MultipleCertificatesError)
{
        // Get the list of possible choices
        CertificateChoice[] choices = response.CertificateChoices;

        // Ask the user to pick one of the certificates, using the
        // PrettyName property of the CertificateChoice class to guide the user.
        // the code below really depends on the client
        foreach (CertificateChoice choice in choices)
        {
                // display "choice.PrettyName" to the user or display
                // them all in a dropdown-list, or whatever makes sense
        }

        // Set the choice
        CertificateChoic choice = ...;

        // Set the choice of certificate on the request, and call the SignOn
        // method again...
        request.SetCertificateChoice(choice);
        response = facade.SignOn(request);

        // TODO: parse response.Status again
}
```

Detailed description of the plugin

Internally, the plugin exposes two methods to the SignOn Library framework, namely Invoke() and Release(). It also exposes a Sign() method to the other plugins, and the order in which they are called is


SignOn Library calls Invoke()
3rd party plugin calls Sign()
SignOn Library calls Release()

When the SignOn Library calls Invoke, all the verification of certificates and the password occur, and the mentioned errors will all happen here. After this point, any 3rd party plugin can call the Sign() method (any number of times), until the SignOn Library calls Release(), after which calling the Sign() method will result in an exception.

The plugin accesses the “MY” keystore in CAPI, which is the default location for user-certificates and keys. To avoid the user being prompted for a password, the plugin will pass the password programatically into CAPI, and should the CSP (cryptographic service provider) respect this, the user will not see any password dialog.

The CSP provided by DanID, which is the default CSP for use with OCES in Denmark respects that the password is passed programatically, so for all practical purposes, the user will never see the password dialog from DanID.

It should be noted, that if no password is given when configuring the plugin, that is, the following line is left out

```
        request.Password = "Test1234";
```

then the CSP from DanID will prompt the user for his password.

Requirements

The CAPI plugin can only set the password on a CSP that supports this operation – otherwise the CSP will open a password dialog and prompt the user for the password. To ensure that OCES plugins are handled by a CSP that supports programatically-set passwords, ensure that the DanID CSP has been installed on the client machine.

The DanID CSP can be downloaded from this address:

https://danid.dk/export/sites/dk.danid.oc/da/support/vejledninger/windows/vedligeholdelse/hent_signatursoftware/

A direct link to the MSI installer is:

https://danid.dk/export/sites/dk.danid.oc/csp