Guidelines for Developing Cryptographic Service Providers (CSPs) for Acrobat on Windows

CONTENTS

This document provides guidelines for those developers who need to develop a Cryptographic Service Provider (CSP) for use with Acrobat® on the Windows® platform.

A CSP is a software module that uses cryptography algorithms for authentication, encoding, and encryption. The CSP makes it possible for MSCAPI-compliant applications to access digital IDs through an API rather than requiring the developer to provide a custom interface for each application to access a stored digital ID.

The guidelines are discussed in the following sections:

- "Requirements for Minimal Functionality" on page 1
- "Recommendations for Maximum Functionality" on page 2
- "For a Better User Experience Using CSPs in Acrobat" on page 3

NOTE: These guidelines have changed since Acrobat 6.0, where MSCAPI support was first introduced. These guidelines apply to the PPKLite.api plugin used with Acrobat 6.0 and above; guidelines for specific versions are noted where necessary. Except where otherwise noted, all guidelines apply to both Adobe Acrobat and Adobe Reader.

Requirements for Minimal Functionality

The following recommendations must be met for a CSP to have at least minimal functionality within Acrobat.

Provider types supported

The CSP must have one of the following provider types:

- PROV_RSA_FULL (recommended)
- PROV_RSA_SIG
- PROV_DSS

If the CSP is not one of these types, it may still work but it is not recommended or supported.

If the CSP is not PROV_RSA_FULL, decryption will not work even if the certificate usage allows the operation.

Adobe

Allow multiple operations with a single private key acquisition

To assure a good user experience, the number of times a user has to authenticate for a particular digital ID should be minimized. Acrobat assumes that once a private key handle is acquired, it will remain good until released.

If the CSP invalidates the handle after any operation other than CryptReleaseContext, Acrobat will simply fail on subsequent calls since it has no way to know it needs to reacquire the private key handle.

Triple-DES encryption support required (Acrobat 6.x only)

In order to use a certificate for encryption with Acrobat 6.x, the CSP must support Triple-DES. This is because the CMS (PKCS#7) object stored in the PDF file is being encrypted by CAPI. In Acrobat 7.0 and higher, this is no longer an issue as the CMS object is being encrypted by Acrobat itself and only the session key is encrypted via CAPI.

If a CSP does not support Triple-DES, then encryption/decryption under Acrobat 6.0 will not work. The user will receive the error message: *Unsupported algorithm*.

Recommendations for Maximum Functionality

The following recommendations should be met to allow use of all the CAPI features Acrobat can support. Each recommendation includes a brief description of what functionality will be enabled or disabled by its presence.

To support deletion of credentials from Acrobat

Acrobat requires the CSP to be of type PROV_RSA_FULL and support the `CRYPT_DELETEKEYSET` flag.

If not supported, then deleting credentials via the Security Console in Acrobat will result in one of several possible error messages, the most common of which will be:

- *The account you are logged into is not permitted to do this.*
- *Removing a digital ID is only permitted if the ID was created by Adobe Reader or Acrobat version 8.0 or later.*

To support JavaScript login with password

Acrobat requires CSP support for CryptSetProvParam with one of the following parameter sets:

- `PP_KEYEXCHANGE_PIN, PP_SIGNATURE_PIN`
- `KP_KEYEXCHANGE_PIN, KP_SIGNATURE_PIN`

This parameter is passed along with the password in UTF8 encoding, to `CryptSetProvParam` when an attempt is made to acquire private key access without displaying a UI. This is also essential for headless operation (see below).

If this is not supported, logging in via JavaScript will not work for any CSPs requiring an authentication UI (e.g. Microsoft Strong Cryptographic Provider).

To support headless operation

Adobe recommends that the CSP fully support the `CRYPT_SILENT` flag. This will allow the CSP to be run in a server environment with LiveCycle server products.

If the CSP does not fully support this flag (that is, some calls result in a UI being displayed), it can cause servers based on Acrobat to halt. This can also halt processing via Acrobat's JavaScript or batch mechanisms.

To support more efficient decryption

Acrobat takes the data to be decrypted and reverses the byte ordering before calling CryptDecrypt to decrypt it. This is because the majority of CSPs that have been tested require this (e.g. Microsoft Strong Cryptographic Provider). It is recommended that CSPs conform to the behavior of the built-in CSP in this respect. Acrobat supports both byte orderings, but the majority case is handled first.

For a Better User Experience Using CSPs in Acrobat

Authentication dialogs that are presented by the CSP will interrupt the Acrobat signing and decryption workflows. The following are scenarios that can cause the authentication UI for the CSP to be requested.

In Acrobat 8.0 and later

To make sure that the CSP dialog window has focus as the topmost window:

1. Call `CryptAquireContext` to get a handle to Acrobat.
2. Set the window handle using `CryptSetProvParam`, passing the `PP_CLIENT_HWND` parameter, to make sure the CSP dialog is a child of Acrobat..

Support for removable tokens. For those CSPs that support removable tokens of some type (e.g. smart cards, USB tokens, etc.), the credentials will be transient. By this I mean that the usage of a particular credential depends on whether the token is physically present. For these types of credentials either the certificates should be installed and uninstalled into the Windows Personal Certificate Store as the token is added and removed, or, that appropriate UI is presented warning the user that a hardware token is required. An example of a reasonable warning message is: "Please insert your SampleCorp hardware token."

Simply failing to acquire the private key is not sufficient and will result in a bad user experience.

Display more Contextual Information. For those CSPs that do require some UI to prompt the user for a token or to authenticate, the more contextual information that can be provided, the better. This will let the user know why a particular piece of UI is showing up and avoid Acrobat having to show UI before or after warning the user what is about to happen.

Examples include:

- What is the operation being attempted?

- What is the certificate associated with the private key being accessed?

Obviously, this recommendation conflicts with some security needs as this might lead to an unacceptable information leakage. Also some of this information may not be available or unique enough to display meaningfully (e.g. which certificate). Use your best judgement here.

In Acrobat 7.0 and later

1. Login from JavaScript: This acquires the private key handle for a credential. Depending on the parameters passed, this may or may not allow an authentication dialog to be presented.
2. Signing: This acquires and uses the private key handle once during signing. Acrobat only signs the document digest.
3. Decryption: This acquires and uses the private key handle during decryption. Multiple calls will be made, first trying to acquire the private key without UI and then with allowing UI to be presented.

   It is recommend that a CSP present an authentication dialog on the first attempt to use the private key for signing or decryption. If this attempt is successful, the authentication dialog should not be presented again until the provider handler is closed.

   The provider handle will be closed when Acrobat is closed or when the user performs one of the following actions, and there is no document open that requires the credential for decryption:

   - The user releases the credential using the Security Console
   - The user deletes the credential using the Security Console
   - The credential is released via JavaScript

   If the CSP uses a different scheme for authentication (e.g. authenticate on every key access, authenticate after elapsed period, etc.), the workflow the user experiences may not be as smooth.

In Acrobat 6.x and later

1. Login from the menu

   This acquires the private key handle for a credential and uses it to sign some dummy data. This signature is discarded. This will allow a UI to be presented.

2. Login from JavaScript

   This acquires the private key handle for a credential and uses it to sign some dummy data. This signature is discarded. This may or may not allow a UI to be presented, depending on the parameters.

3. Signing

   This acquires the private key handle and uses it twice during signing. The first access is to sign some dummy data and get the size of the resulting signature. This signature is discarded. The second access is to sign the document digest. This will allow a UI to be presented.

4. Decryption

   This acquires and uses the private key handle during decryption. Acrobat first tries to acquire the private key without a UI. If it does not get the private key, it allows five calls per private key handle: UI is not allowed for the first call, but is allowed for the remaining four calls.

Other Recommendations

To support better credential validation. We recommend that the CSP support the `CRYPT_ACQUIRE_COMPARE_KEY_FLAG` when calling `CryptAcquireContext`. This is a useful test to ensure that the correct credential is being used.

To support more efficient signing. Since Acrobat already has a digest of the document data to be signed, it is inefficient to have the CSP digest the data again when signing. We recommend that the CSP support the `HP_HASHVAL` flag when calling `CryptSetHashParam`. This allows the digest to be signed directly, rather than hashed and then signed.

Adobe Systems Incorporated

345 Park Avenue, San Jose, CA 95110-2704 USA

http://www.adobe.com

28 August 2006