Remote Signature Server
User Manual

Introduction

The main function of **Remote Signature Server** is to digitally sign files in PDF or CadES/PKCS#7 cryptographic standard (.P7S or .P7M files) using X.509 certificates stored on remote server.

Remote Signature Server is useful when you have a single "company" certificate that it must be used by more people from your company.

The Remote Signature Server is a webservice that digitally sign the information that arrives with a single digital certificate.

How to test the solution:
- download Remote Signature Server http://www.signfiles.com/sdk/RemoteSignature.zip
- on your Microsoft IIS server you must install the Server itself available here: *RemoteSignature.zip\Remote Signature*
- Configure the Remote Signer Server to use a digital certificate
- Open Remote PDF Signer Client available here: *RemoteSignature.zip\Desktop Clients*
- Replace the Remote Signer Server URL with your Server URL (e.g. *https://your.internal.url/RemoteSigner/RemoteSignature.asmx*)
- Digitally sign a file.

Links

Remote Signature Server: http://www.signfiles.com/remote-signature/
Remote Signature Server Live Demo: https://ca.signfiles.com/RemoteSigner/

Warning and Disclaimer

Every effort has been made to make this manual as complete and accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this manual.

Trademarks

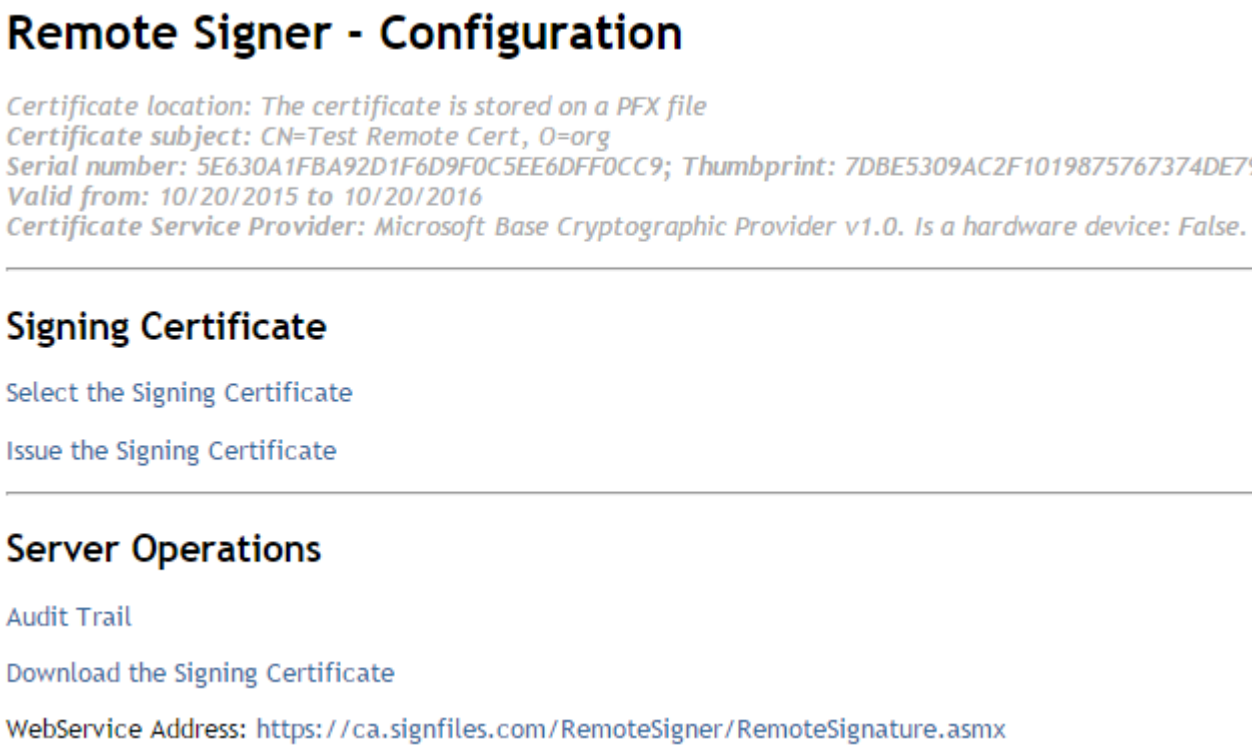
.NET, Visual Studio .NET are trademarks of Microsoft Inc.
Adobe, Adobe Reader are trademarks of Adobe Systems Inc.
All other trademarks are the property of their respective owners.

Remote Signature Server Installation

Remote Signature Server is useful when you have a single "company" certificate that it must be used by more people from your company.

The Remote Signature Server is a webservice that digitally sign the information that arrives with a single digital certificate.

How to test the solution:
- download Remote Signature Server http://www.signfiles.com/sdk/RemoteSignature.zip
- on your Microsoft IIS server you must install the Server itself available here: RemoteSignature.zip\Remote Signature\
- Configure the Remote Signer Server to use a digital certificate
- Open Remote PDF Signer Client available here: *RemoteSignature.zip\Desktop Clients*
- Replace the Remote Signer Server URL with your Server URL (e.g *https://your.internal.url/RemoteSigner/RemoteSignature.asmx*)
- Digitally sign a file.



PDF Digital Signatures

Validating Digital Signatures in Adobe

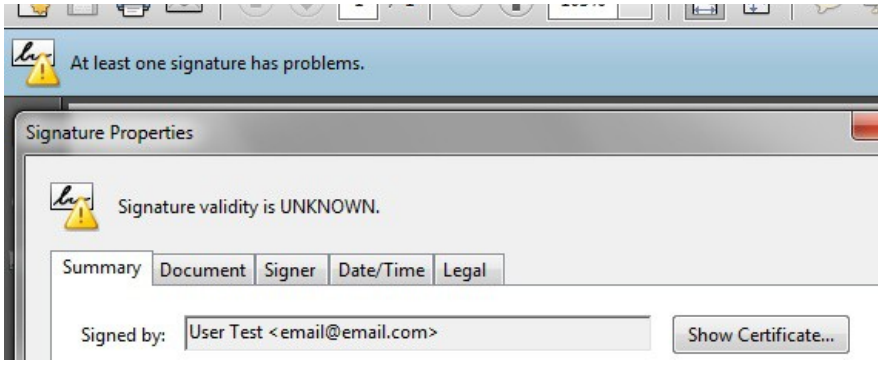
Every digital certificate is issued by a Root CA (Certification Authority). Some of the Root CA's are included by default in Windows Certificate Store (Trusted Root Certification Authorities) and only a few are included in Adobe Certificate Store. Microsoft and Adobe use different Certificate Stores different certificate validation procedures.

If the signing certificate (or the Root CA that issued the signing certificate) is not included in Adobe Store, the digital signature is considered "not trusted" when a user open a document with Adobe Reader (see example).

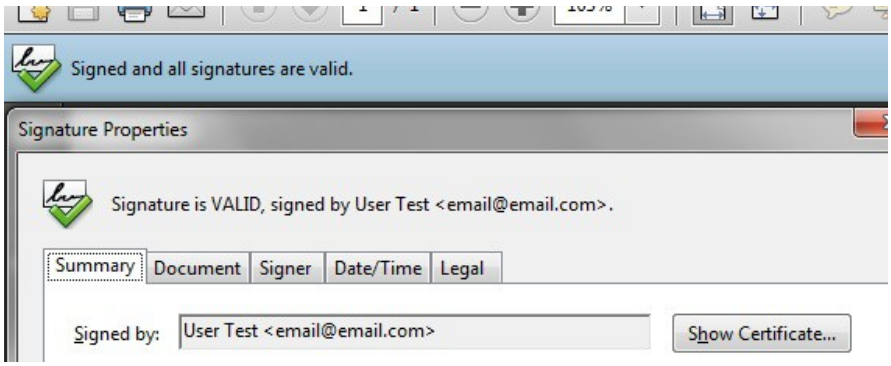
This behavior has nothing to do with the signing engine but with the Adobe certification validation procedure.

To trust a signature the user must add the signing certificate on the Adobe Certificate Store because only a few Root CA's are considered trusted by default by Adobe certificate validation engine (See this article: http://www.adobe.com/security/partners_cds.html)

To validate the signing certificate in Adobe use the methods described on this document: http://www.signfiles.com/manuals/ValidatingDigitalSignaturesInAdobe.pdf



Validity Unknown signature



Valid signature

Loading the PDF Document

The PDF can be loaded from a file, a byte array or from an URL like below:

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");

//Load the PDF from byte[] array
ps.LoadPdfDocument(File.ReadAllBytes("c:\\source.pdf"));

//Load the PDF from a file
ps.LoadPdfDocument("c:\\source.pdf");

//Load the PDF from an URL
ps.LoadPdfDocument(new Uri("http://www.signfiles.com/test.pdf"));
```

Obtaining the Document Information (Number of Pages, Page Size)

In some cases you will need some information about the opened document (is document already signed, number of pages, document page size).

DocumentPageSize property is useful when you want to place a custom digital signature rectangle on the PDF document.

DocumentProperties.NumberOfPages is useful when you want to place a signature on the last page of the document.

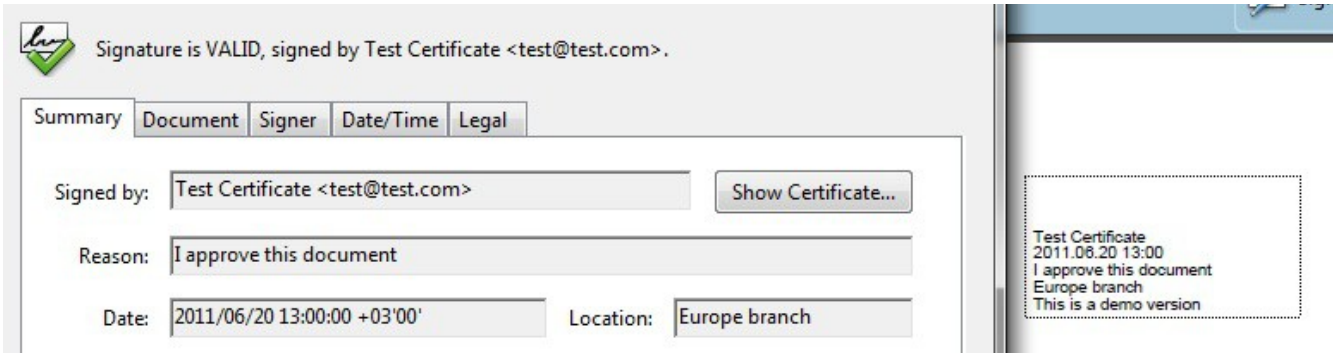
```
//Load the PDF file
ps.LoadPdfDocument(File.ReadAllBytes("c:\\source.pdf"));

//get the page size of the last page of the document
ps.DocumentPageSize(ps.DocumentProperties.NumberOfPages);

//get the number of digital signatures already attached to this document
int signatures = ps.DocumentProperties.NumberOfDigitalSignatures;
```

Set the Digital Signature Properties (Reason, Location)

Adobe digital signatures can be customized with SignLib SDK. In order to set the Reason or Location properties, use the code below.

Observation: Some digital signature properties (like “Signed by” in Adobe) will not appear with your custom value because of Adobe policy. If Time stamping is used, the signing date (*SignatureDate* property) is taken from the time stamping response.

```
ps.SigningReason = "I approve this document";
ps.SigningLocation = "Europe branch";
```



Signed by, Reason, Location and Date properties in Adobe

Signed by, Reason, Location, Date and Signer's contact properties in other PDF reader

Set the Digital Signature Rectangle Properties

The digital signature rectangle can appear on the PDF document on a standard location (like Top Right) or in a custom place based on the PDF page size.

Example: put the digital signature rectangle on the last page of the document on top middle position:

```
ps.SignaturePage = ps.DocumentProperties.NumberOfPages;
ps.SignaturePosition = SignaturePosition.TopMiddle;
```

Observation: In Adobe, the corner (0,0) is on the bottom left of the page.

Example: put the digital signature on a custom position (top right corner) on the first page of the document:

```
ps.SignaturePage = 1;
//get the pdf page size
System.Drawing.Point page = ps.DocumentProperties.DocumentPageSize(1);

//set the rectangle width and height
int width = 80;
int height = 40;
ps.SignatureAdvancedPosition = new System.Drawing.Rectangle(page.X - width, page.Y
- height, width, height);
```

Set a Custom Digital Signature Text

The default digital signature text contains information extracted from the signing certificate, signing date, signing reason and signing location.

The signature text can be set using *SignatureText* propery like below:

```
ps.SignatureText =
"Signed by:" + ps.DigitalSignatureCertificate.GetNameInfo(X509NameType.SimpleName,
false) + "\n Date:" + DateTime.Now.ToString("yyyy.MM.dd HH:mm") + "\n" +
"Reason:" + ps.SigningReason;
```

Set the Text Direction on the Signature Rectangle

The default text direction is left to right. To change the text direction to right to left use the following code (e.g. for Hebrew language):

```
ps.TextDirection = TextDirection.RightToLeft;
```

Set the Digital Signature Font

The default font file for the digital signature rectangle is Helvetica. It is possible that this font to not include all necessary UNICODE characters like **ä, à, â**. On this case you will need to use an external font.

The font size is calculated based on the signature rectangle size in order to fit on the signature rectangle (it not have a fixed size). To set the font size you can use *FontSize* propery like below:

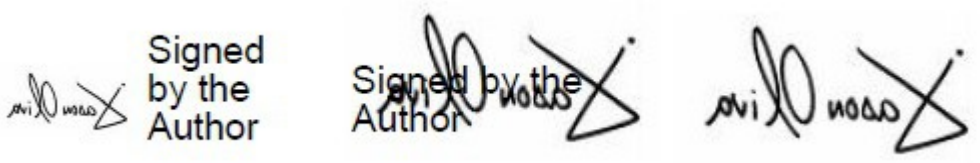
```
ps.FontFile = "c:\\windows\\fonts\\arial.ttf";
ps.FontSize = 10;
```

Set the Digital Signature Image

The digital signature rectangle can contains text, image or text with image. To add an image on the digital signature rectangle use the following code:

```
ps.SignatureText = "Signed by the Author";
ps.SignatureImage = System.IO.File.ReadAllBytes("c:\\graphic.jpg");

//text on the right and image on the left
ps.SignatureImageType = SignatureImageType.ImageAndText;
//image as bakground and text on above
ps.SignatureImageType = SignatureImageType.ImageAsBackground;
//only image
ps.SignatureImageType = SignatureImageType.ImageWithNoText;
```

These types of signatures are shown below:



1. Image and text, 2. Image as background, 3. Image with no text

Set a Visible or Hidden Signature

Sometimes the digital signature rectangle is not necessary to appear on the PDF document. The default value of *VisibleSignature* property is true.

To set an invisible digital signature use the code below:

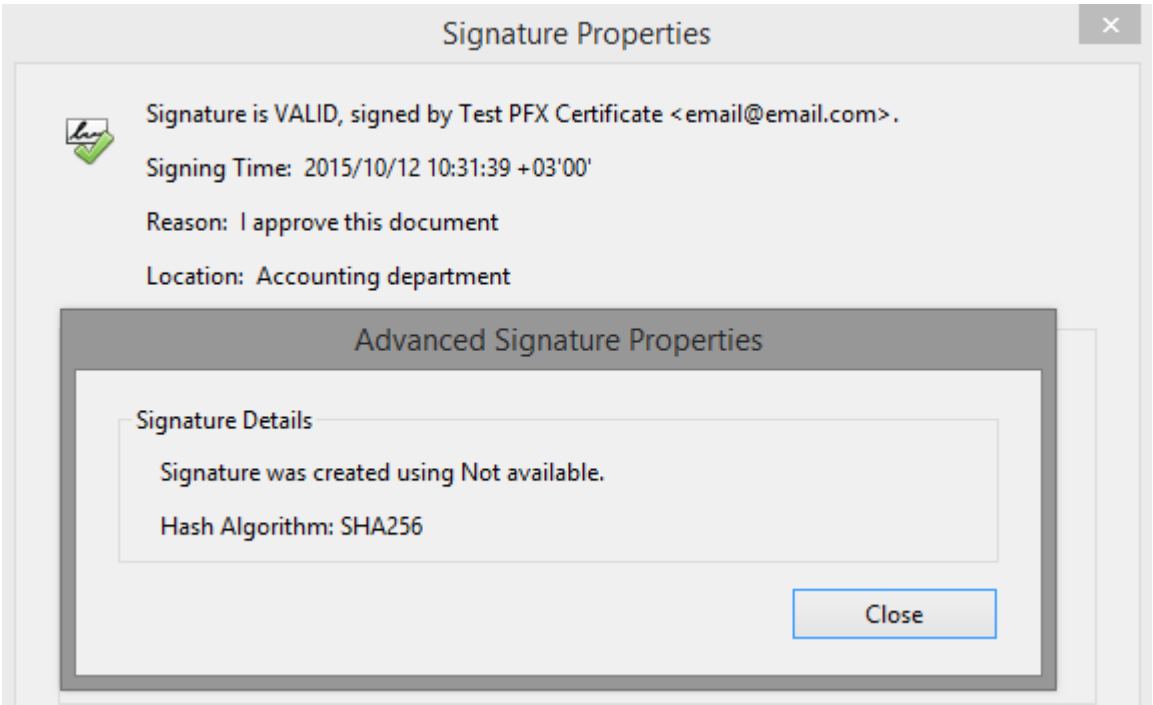
```
//invisible signature
ps.VisibleSignature = false;

//digitally sign and save the PDF file
File.WriteAllBytes("c:\\dest.pdf", ps.ApplyDigitalSignature());
```

Hash Algorithms

By default, the hash algorithm used to create the digital signatures is SHA-1. In order to use SHA-256 or SHA-512 hashing algorithm, check the property *HashAlgorithm*.

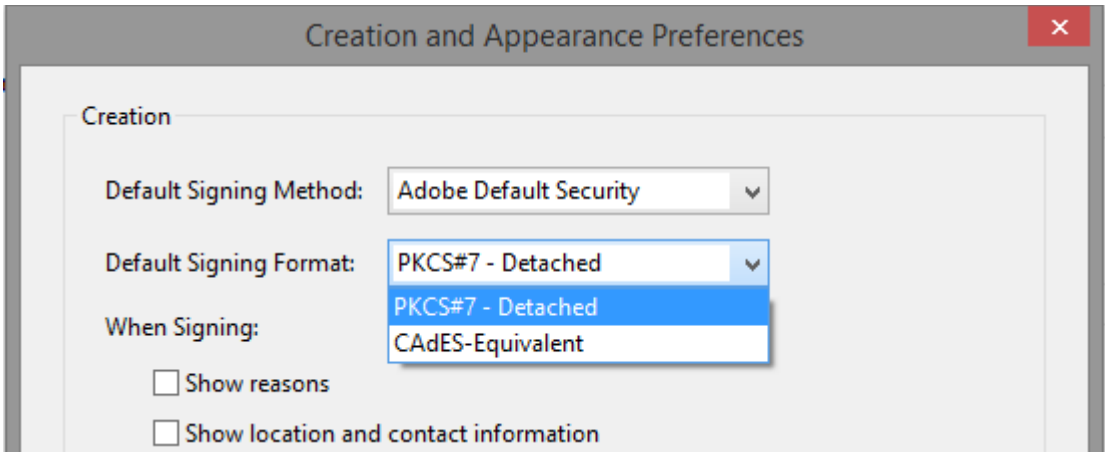
```
//hash algorithm used for creating the digntal signature
ps.HashAlgorithm = SignLib.HashAlgorithm.SHA256;

//hash algorithm used for creating the time stamp request
ps.TimeStamping.HashAlgorithm = SignLib.HashAlgorithm.SHA256;
```

Attention: SHA-256, SHA-384 and SHA-512 hash algorithms are not supported by Windows XP. Note that some smart cards and USB tokens not support SHA-256, SHA-384 and SHA-512 hash algorithms.



Advanced PDF Signatures (e.g. Required by Italian Law)

In order to be compatible with all Adobe Reader versions and with third party PDF readers, the default signature standard is PKCS#7 – Detached.



Some countries require the new PDF signature standard named *CAdES (PadES)*.
In order to use this new standard, use the code below (note that the signature must be SHA-256).

```
PdfSignature ps = new PdfSignature(serialNumber);

//load the PDF document
ps.LoadPdfDocument(unsignedDocument);

ps.HashAlgorithm = SignLib.HashAlgorithm.SHA256;
ps.SignatureStandard = SignLib.SignatureStandard.AdvancedSignature;

//optionally, the signature can be timestamped (SHA-256 algorithm must be used).
ps.TimeStamping.ServerUrl = new Uri("http://ca.signfiles.com/TSAServer.aspx");
ps.TimeStamping.HashAlgorithm = SignLib.HashAlgorithm.SHA256;

//write the signed file
File.WriteAllBytes(signedDocument, ps.ApplyDigitalSignature());
```

Attention: The old versions of Adobe Reader and some versions of digital signature verification software will not recognize this format.

Time Stamping

Time Stamp the PDF Digital Signature

Timestamping is an important mechanism for the long-term preservation of digital signatures, time sealing of data objects to prove when they were received, protecting copyright and intellectual property and for the provision of notarization services.

To add time stamping information to the PDF digital signature you will need access to a RFC 3161 time stamping server.

A fully functional version of our TSA Authority is available for testing purposes at this link: http://ca.signfiles.com/TSAServer.aspx (no credentials are needed).

Use the code below to digitally sign and timestamp your PDF file:

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");

//load the PDF document
ps.LoadPdfDocument("d:\\source.pdf");

ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");


//Time stamp the PDF digital signature
ps.TimeStamping.ServerUrl = new Uri("http://ca.signfiles.com/TSAServer.aspx");

//write the signed file
File.WriteAllBytes("d:\\dest.pdf", ps.ApplyDigitalSignature());
```

Authentication With Username and Password

If your **TSA server requires username and password**, use the following code:

```
ps.TimeStamping.ServerUrl  = new Uri("http://ca.signfiles.com/TSAServer.aspx");
ps.TimeStamping.UserName = "username";
ps.TimeStamping.Password = "password";
```

Authentication with a Digital Certificate

In some cases, the access to your **TSA server must be done using a digital certificate (authentication certificate)**. On this case use the following code:

```
//Time stamp the PDF digital signature
ps.TimeStamping.ServerUrl  = new Uri("http://ca.signfiles.com/TSAServer.aspx");

ps.TimeStamping.AuthenticationCertificate =
DigitalCertificate.LoadCertificate("d:\\time_stamping_certificate", "123456");
```

Nonce and Time Stamping Policy OID

The **nonce**, if included, allows the client to verify the timeliness of the response when no local clock is available. The nonce is a large random number with a high probability that the client generates it only once (e.g., a 64 bit integer).

To include (or exclude) a Nonce on the time stamping request use the following code. The default value of the *UseNonce* propery is true.:

```
ps.TimeStamping.UseNonce = true;
```

Some TSA servers require to set a **Policy OID** on the TSA requests. To set a TSA policy OID on the time stamping requests use the code below. By default, no TSA OID is included on the TSA request.

```
ps.TimeStamping.ServerUrl  = new Uri("http://ca.signfiles.com/TSAServer.aspx");

ps.TimeStamping.PolicyOid = new
System.Security.Cryptography.Oid("1.3.7.2.9.1.829.3");
```

Hash Algorithms

By default, the hash algorithm used to generate the Time Stamp Request is SHA-1.
In order to use SHA-256 or SHA-512 hashing algorithm, check the property *TimeStamping.HashAlgorithm*.

```
ps.TimeStamping.HashAlgorithm = SignLib.HashAlgorithm.SHA256;
```

Attention: SHA-256, SHA-384 and SHA-512 hashing algorithms are not supported by Windows XP. Note that some smart cards and USB tokens not support SHA-256, SHA-384 and SHA-512 hashing algorithms.

Validating the Time Stamping Response on Adobe

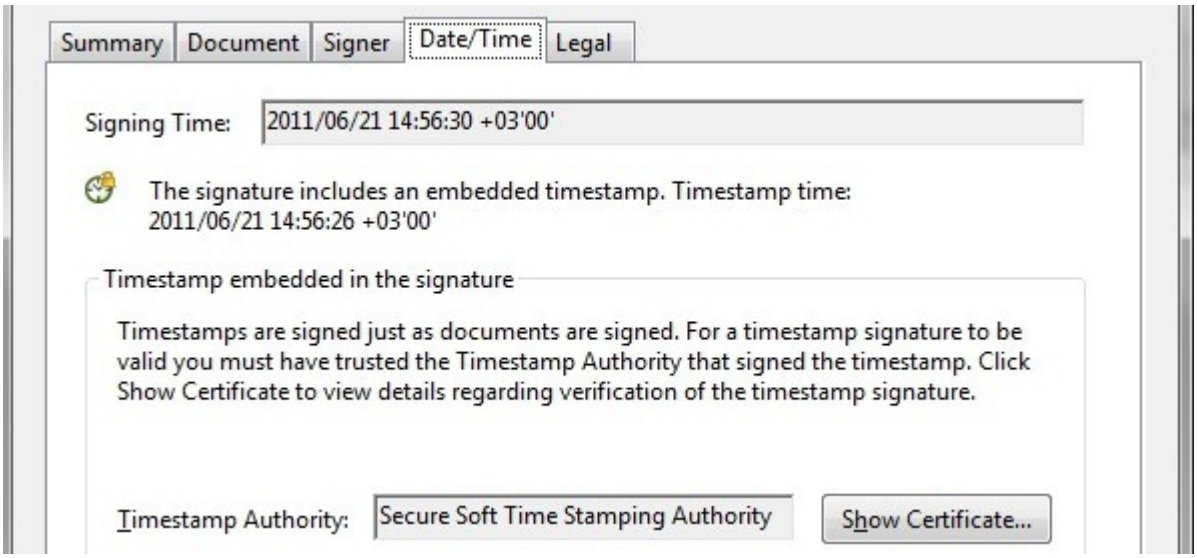
As digital signatures certificates, the time stamping responses are signed by a certificate issued by a Certification Authority.

If the time stamping certificate (or the Root CA that issued the time stamping certificate) is not included in Adobe Store, the time stamping response could not be verified when a user open a document with Adobe Reader (see example).

This behavior has nothing to do with the signing engine but with the Adobe certification validation procedure.

To validate the signing certificate in Adobe use the methods described on this document: http://www.signfiles.com/manuals/ValidatingDigitalSignaturesInAdobe.pdf.

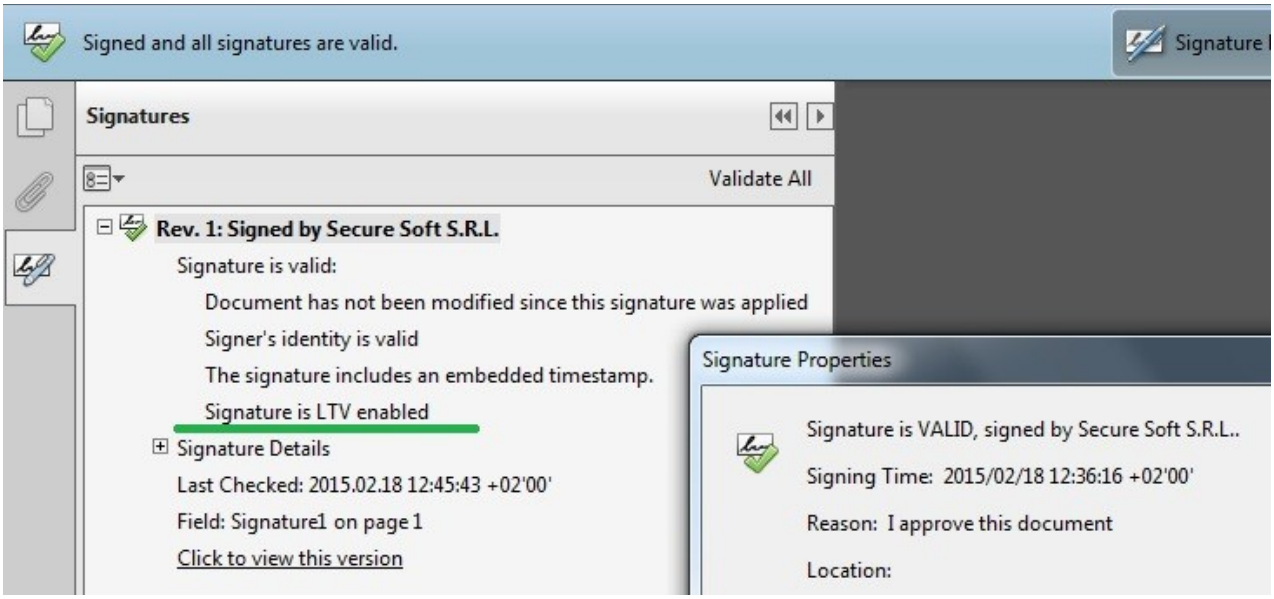
Not verified timestamp



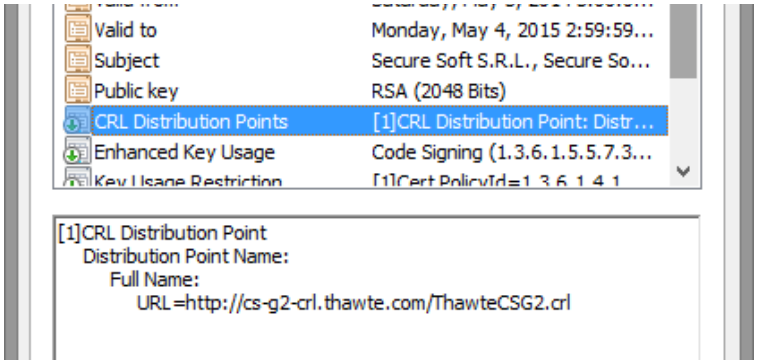
Trusted time stamping response

LTV Signatures (Long Term Validation)

PAdES recognizes that digitally-signed documents may be used or archived for many years – even many decades. At any time in the future, in spite of technological and other advances, it must be possible to validate the document to confirm that the signature was valid at the time it was signed – a concept known as Long-Term Validation (LTV).



In order to have a LTV signature, be sure that the certificate have a CRL and the revocation info is included on the signature. Including a timestamp is also recommended.



If the CRL revocation information will not be available online, the digital signature cannot be verified as Long Term Validation signature by the Adobe Reader engine.

```
ps.IncludeCrlRevocationInfo = true;
```

Attention: In some cases, the CRL file is very large (1 to 3 MB) so the signed PDF file size will increase with at least the size of the CRL file.

Certify a PDF Digital Signature

When you certify a PDF, you indicate that you approve of its contents. You also specify the types of changes that are permitted for the document to remain certified.

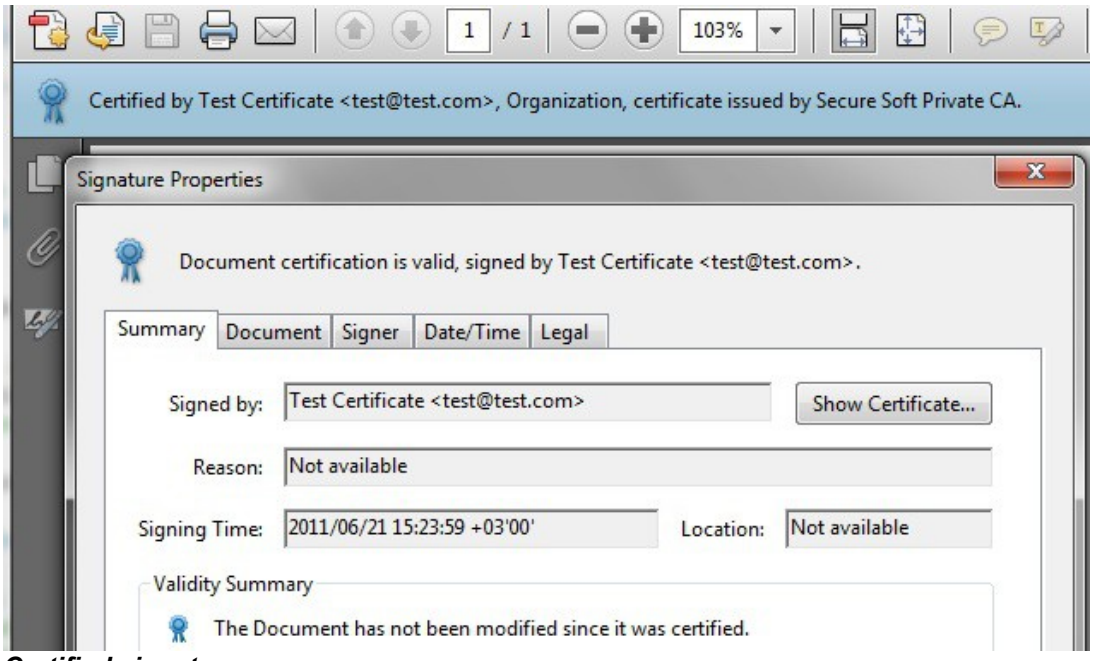
Attention: If the certification type is "*No changes allowed*", additional digital signatures cannot be added on the document.

You can apply a certifying signature only if the PDF doesn't already contain any other signatures. Certifying signatures can be visible or invisible. A blue ribbon icon in the Signatures panel indicates a valid certifying signature (see example).

More information about the certification process you can find here.

To certify a digital signature use the following code:

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");
//adding annotations and form filling are allowed
ps.CertifySignature = CertifyMethod.AnnotationsAndFormFilling;
//form filling is allowed
ps.CertifySignature = CertifyMethod.FormFilling;
//no changes allowed
ps.CertifySignature = CertifyMethod.NoChangesAllowed;
//digitally sign and save the PDF file
File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
```



Certified signature

PDF Digital Signatures and the PDF/A Standard

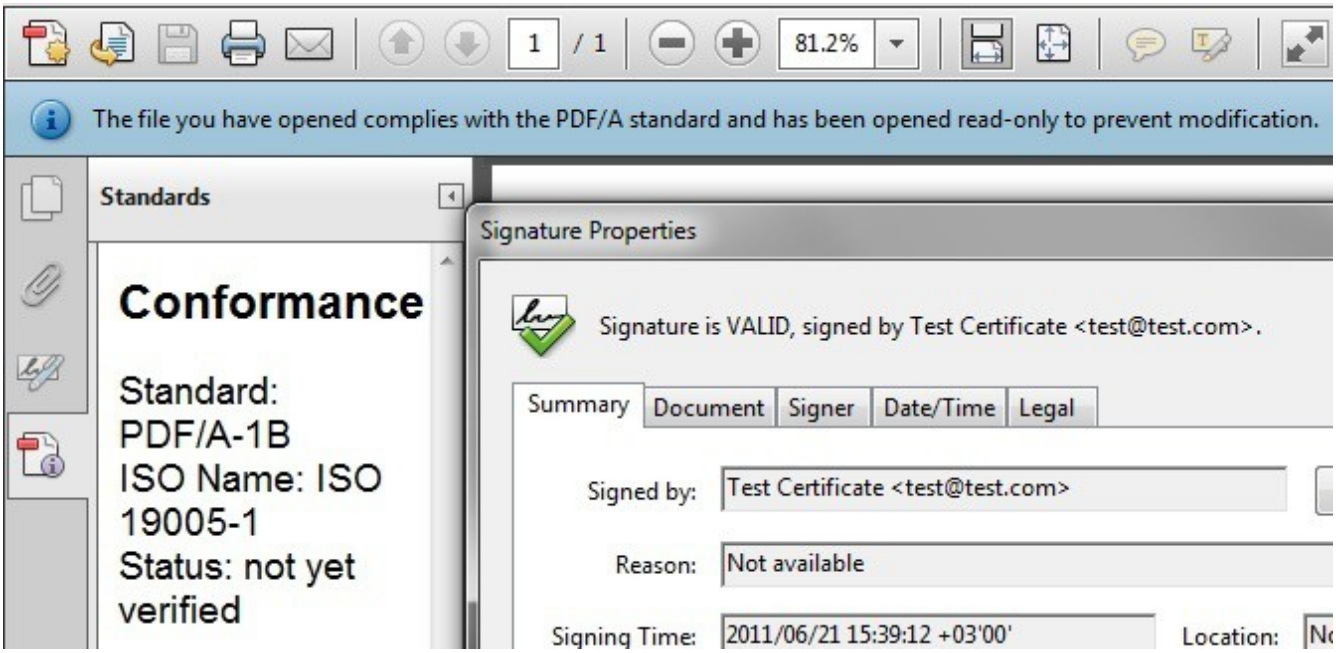
PDF/A is a file format for the long-term archiving of electronic documents. It is based on the PDF Reference Version 1.4 from Adobe Systems Inc. (implemented in Adobe Acrobat 5 and latest versions) and is defined by ISO 19005-1:2005.

SignLib library can save PDF file in PDF/A-1b - Level B compliance in Part 1 standard.

Observation: In order to save a PDF/A-1b file all fonts used on the PDF document must be embedded (including the font used on the digital signature rectangle).

To digitally sign a file in PDF/A-1b standard use the following code:

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");
//Load the PDF file
ps.LoadPdfDocument(File.ReadAllBytes("d:\\source.pdf"));
ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");
ps.SignaturePage = 1;
ps.SaveAsPdfA = true;
ps.FontFile = "c:\\windows\\fonts\\arial.ttf";
//digitally sign and save the PDF file
File.WriteAllBytes("d:\\dest.pdf", ps.ApplyDigitalSignature());
```



PDF/A-1b document with digital signature

Other Features of the PDF Signatures

Digitally Sign all Pages From a PDF Document

To add the digital signature rectangle to all pages from the PDF document use the following code (the default values is false):

```
ps.SignaturePage = 1;
ps.SignaturePosition = SignaturePosition.TopLeft;

ps.SignatureAppearsOnAllPages = true;
```

Adding Multiple Digital Signatures on the PDF Document

Digital signature is appended to the document in order to add multiple signatures to the document. In order to add only one digital signature set the *AppendSignature* propery to false (the default value is true). When you choose to encrypt and digitally sign a PDF file *AppendSignature* property will be automatically set to false.

Observation: This is an invisible property and will not appear on autocomplete.

```
ps.AppendSignature = false;
```

Set an Approximate Block Size for the Digital Signature

The default block size for the digital signature information is 16384 bytes. This space should be enough for the digital signature information and the time stamping response.

In some cases, the size of the document is an critical factor so the size of the signed file can be reduced by setting a lower value of the signature block size.

Observation: This value is approximative and cannot be set on the signed document to an exact value so the final size of the signed file is not equal with the original file size + *SignatureByteBlockSize*.

The digital signature block contains:

- public key of the signing certificate
- information like signing reason, signing location
- document signed digest in PKCS#7 format
- time stamping response

To set a custom space for the signature block size (this is an invisible property and will not appear on autocomplete) use the following code:

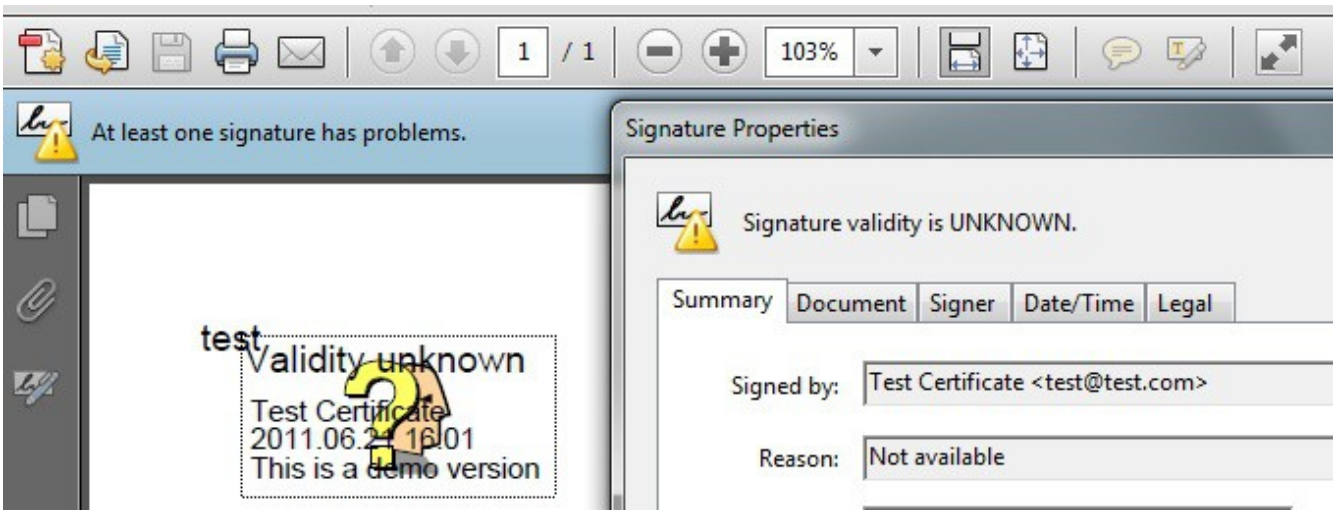
```
ps.SignatureByteBlockSize = 8192;
```

Old Style Adobe Digital Signature Appearance

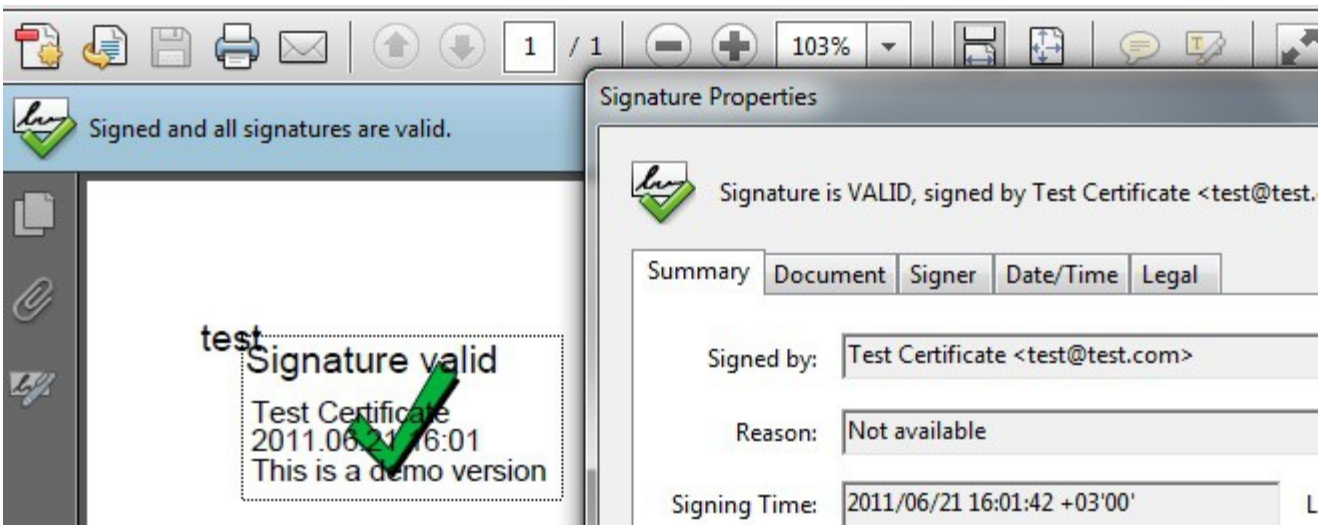
To use an old style appearance of the digital signature rectangle (see example) set the *OldStyleAdobeSignature* property to true. The default value is false.

Observation: This is an invisible property and will not appear on autocomplete.

```
ps.OldStyleAdobeSignature = true;
```



Validity unknown signature



Signature valid

Include the CRL Revocation Information on the PDF Signature

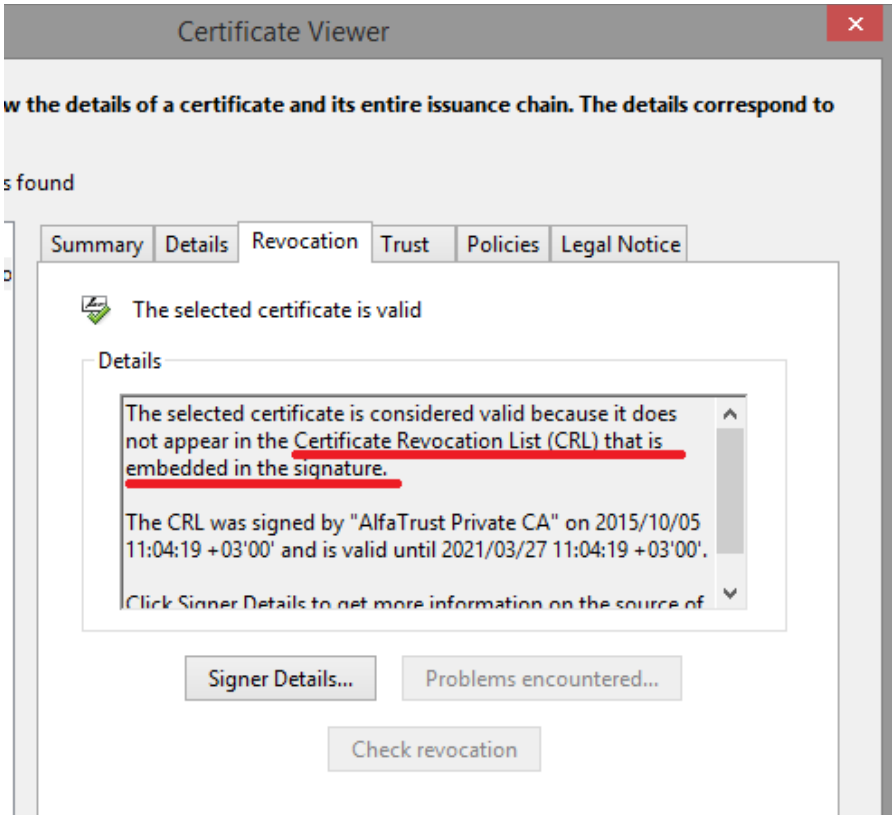
If the CRL revocation information will not be available online, the digital signature cannot be verified by the Adobe Reader engine so it is recommeded to include the CRL on the signature block. The defauld value of the *IncludeCRLRevocationInfo* property is false.

To include the revocation information, set the property to true.

```
ps.IncludeCrlRevocationInfo = true;
```

Attention: In some cases, the CRL file is very large (1 to 3 MB) so the signed PDF file size will increase with at least the size of the CRL file.

Read more about this on the section: LTV Signatures (Long Term Validation)



PDF Signatures and Encryption

If you want to protect the signed document by preventing actions like printing or content copying, it must be encrypted. The document can be encrypted using passwords or digital certificates.

Password Security

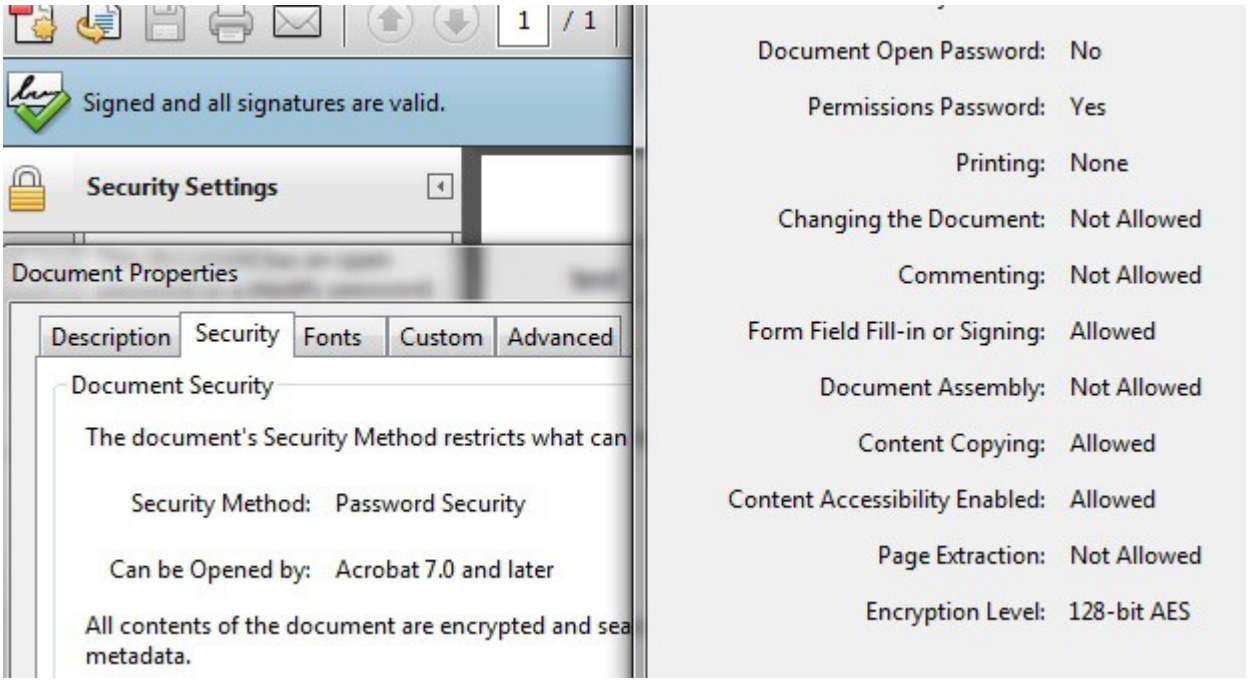
In order to encrypt the PDF document, the *AppendSignature* propery must be set to false. Also, the encryption algorithm must be specified using *EncryptionAlgorithm* property.

OwnerPassword property is used to set the password that protects the PDF document for printing or content copying.

To digitally sign and encrypt a PDF document using a password, use the following code:

```
PdfSignature ps = new PdfSignature("serial number");

//Load the PDF file
ps.LoadPdfDocument(File.ReadAllBytes("d:\\source.pdf"));

ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");

//append signature must be set to false in order to encrypt de document
ps.AppendSignature = false;

//set the document restrictions
ps.Encryption.DocumentRestrictions = PdfDocumentRestrictions.AllowContentCopying |
PdfDocumentRestrictions.AllowFillingOfFormFields;

//set the encryption algorithm
ps.Encryption.EncryptionAlgorithm =
PdfEncryptionAlgorithm.StandardEncryption128BitRC4;

//set the encryption method
ps.Encryption.EncryptionMethod = PdfEncryptionMethod.PasswordSecurity;

//set the owner password
ps.Encryption.OwnerPassword = "123456";

//digitally sign, encrypt and save the PDF file
File.WriteAllBytes("d:\\dest.pdf", ps.ApplyDigitalSignature());
```

When the signed and encrypted document is opened in a PDF reader, the security settings are shown like below.

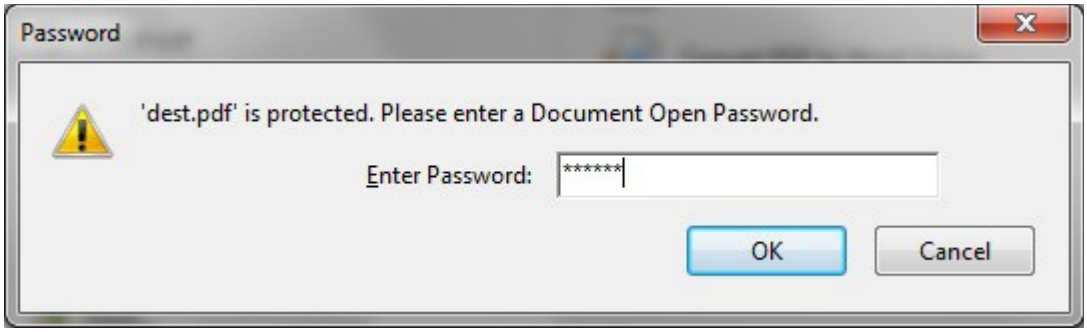


Security settings for a digitally sign and encrypted document

To digitally sign and protect the document with an opened password use the code below instead of the commented line:

```
//PDFSign.Encryption.OwnerPassword = "123456";
ps.Encryption.UserPassword = "123456";
```

When the document is opened in PDF reader, the passwor must be entered.



Password is required to open the document

PDF Signature Code Samples

Digitally Sign All Pages From a PDF File

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");
//load the pdf file
ps.LoadPdfDocument("d:\\source.pdf");

ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");
//put the signature to all pages
ps.SignatureAppearsOnAllPages = true;
//set the signature position
ps.SignaturePosition = SignaturePosition.TopLeft;
//digitally sign and save the PDF file
File.WriteAllBytes("d:\\dest.pdf", ps.ApplyDigitalSignature());
```

Set a Custom Signature Rectangle

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");
ps.LoadPdfDocument("d:\\source.pdf");
ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");
ps.SignaturePage = 1;
//set the signature position
System.Drawing.Point pageRectangle = ps.DocumentProperties.DocumentPageSize(1);
//put the signature on the middle of the page
ps.SignatureAdvancedPosition = new System.Drawing.Rectangle(pageRectangle.X / 2,
pageRectangle.Y / 2, 100, 50);
File.WriteAllBytes("d:\\dest.pdf", ps.ApplyDigitalSignature());
```

Digitally Sign a PDF Located on the Web Only if it is not Already Signed

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature PDFSign = new PdfSignature("serial number");
//load the pdf file from web
PDFSign.LoadPdfDocument(new Uri("http://www.signfiles.com/test.pdf"));

//sign the document only if it is not signed
if (PDFSign.DocumentProperties.DigitalSignatures.Count == 0)
{
      ps.RemoteSignatureServer = new
      Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");
      File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
}
```

Set a Custom Text and Font for the Digital Signature Rectangle

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");

ps.LoadPdfDocument("c:\\source.pdf");
ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");

ps.SignaturePage = 1;
ps.SignaturePosition = SignaturePosition.BottomLeft;

//set the font file
ps.FontFile = "c:\\windows\\fonts\\verdana.ttf";
//set the font size
ps.FontSize = 6;

//customize the text that appears on the signature rectangle
ps.SignatureText = "Signed by: " +
ps.ReadRemoteCertificate().GetNameInfo(X509NameType.SimpleName, false) +
"\nSigning time: " + DateTime.Now.ToShortDateString() +
"\nSigning reason: " + ps.SigningReason +
"\nLocation: " + ps.SigningLocation;

File.WriteAllBytes("c:\\dest.pdf", ps.ApplyDigitalSignature());
```

Add an Image on the Signature Rectangle and Save the File as PDF/A

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");

ps.LoadPdfDocument("d:\\source.pdf");
ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");

ps.SignaturePage = ps.DocumentProperties.NumberOfPages;
ps.SignaturePosition = SignaturePosition.BottomLeft;

ps.SignatureText = "Signed by the author";

//path to the signature image
ps.SignatureImage = File.ReadAllBytes("d:\\graphic.jpg");
ps.SignatureImageType = SignatureImageType.ImageAsBackground;

//the font must be embedded in orde to save the file as PDF/A
ps.FontFile = "c:\\windows\\fonts\\verdana.ttf";
ps.SaveAsPdfA = true;

File.WriteAllBytes("d:\\dest.pdf", ps.ApplyDigitalSignature());
```

Set an Invisible Signature and Certify the PDF File

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");

ps.LoadPdfDocument("d:\\source.pdf");
ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");

//certify the signature
ps.CertifySignature = CertifyMethod.NoChangesAllowed;

//set an invisible signature
ps.VisibleSignature = false;

File.WriteAllBytes("d:\\dest.pdf", ps.ApplyDigitalSignature());
```

Time Stamp a PDF File

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");

ps.LoadPdfDocument("d:\\source.pdf");
ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");

//Set the TSA Server URL
ps.TimeStamping.ServerUrl = new Uri("http://ca.signfiles.com/TSAServer.aspx");

File.WriteAllBytes("d:\\dest.pdf", ps.ApplyDigitalSignature());
```

Time Stamp a PDF file Using TSA Server Authentication

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");

ps.LoadPdfDocument("d:\\source.pdf");
ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");

//Set the TSA Server URL
ps.TimeStamping.ServerUrl = new Uri("http://ca.signfiles.com/TSAServer.aspx");
//set username and password
ps.TimeStamping.UserName = "username";
ps.TimeStamping.Password = "P@ssw0rD";

File.WriteAllBytes("d:\\dest.pdf", ps.ApplyDigitalSignature());
```

Digitally Sign and Time Stamp a Folder with PDF files

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature("serial number");

ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");

ps.TimeStamping.ServerUrl = new Uri("http://ca.signfiles.com/TSAServer.aspx");

System.IO.DirectoryInfo di;
System.IO.FileInfo[] rgFiles;

//get the pdf files from the folder
di = new System.IO.DirectoryInfo("d:\\source_dir");
rgFiles = di.GetFiles("*.pdf");

foreach (FileInfo fi in rgFiles)
{
      //for readonly files
      fi.Attributes = FileAttributes.Normal;
      //load the PDF document
      ps.LoadPdfDocument(di.FullName + "\\" + fi.Name);
      //digitally sign and save the PDF file
      File.WriteAllBytes("d:\\output_dir\\" + fi.Name,
ps.ApplyDigitalSignature());
}
```

Digitally Sign a PDF file in a ASP.NET Application (IIS)

```
using SignLib.Certificates;
using SignLib.Pdf;

protected void Page_Load(object sender, EventArgs e)
{
            PdfSignature ps = new PdfSignature("serial number");

ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");

            ps.LoadPdfDocument(Server.MapPath("source.pdf"));

            System.IO.File.WriteAllBytes(Server.MapPath("dest.pdf"),
ps.ApplyDigitalSignature());
}
```

Automatically Sign a Folder Using a Smart Card Certificate / USB Token

```
using SignLib.Certificates;
using SignLib.Pdf;

PdfSignature ps = new PdfSignature(serialNumber);

ps.SignaturePosition = SignaturePosition.TopLeft;
ps.SignaturePage = 1;

ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");


ps.TimeStamping.ServerUrl = new Uri("http://ca.signfiles.com/TSAServer.aspx");

System.IO.DirectoryInfo di;
System.IO.FileInfo[] rgFiles;

//get the pdf files from the folder
di = new System.IO.DirectoryInfo("d:\\source_dir");
rgFiles = di.GetFiles("*.pdf");

foreach (FileInfo fi in rgFiles)
{
      //for readonly files
      fi.Attributes = FileAttributes.Normal;

      //load the PDF document
      ps.LoadPdfDocument(di.FullName + "\\" + fi.Name);

      //digitally sign and save the PDF file
      File.WriteAllBytes("d:\\output_dir\\" + fi.Name,
ps.ApplyDigitalSignature());
}
```

Verifying a Digital Signature

In some cases is needed to verify the digital signatures attached to a PDF document.

To verify the digital signatures added to PDF document, use the following code:

```
using SignLib.Certificates;
using SignLib.Pdf;

void ExtractCertificateInformation(X509Certificate2 cert)
{
Console.WriteLine("Certificate subject:" + cert.Subject);
Console.WriteLine("Certificate issued by:" + cert.GetNameInfo(X509NameType.SimpleName,
true));
Console.WriteLine("Certificate will expire on: " + cert.NotAfter.ToString());
Console.WriteLine("Certificate is time valid: " +
DigitalCertificate.VerifyDigitalCertificate(cert, VerificationType.LocalTime).ToString());
}

void VerifyPDFSignature(string signedDocument)
{
      PdfSignature ps = new PdfSignature(serialNumber);
      ps.LoadPdfDocument(signedDocument);

      Console.WriteLine("Number of signatures: " +
      ps.DocumentProperties.DigitalSignatures.Count.ToString());

//verify every digital signature form the PDF document
foreach (PdfSignatureInfo csi in ps.DocumentProperties.DigitalSignatures)
{
      Console.WriteLine("Signature name: " + csi.SignatureName);
      Console.WriteLine("Hash Algorithm: " + csi.HashAlgorithm.ToString());
      Console.WriteLine("Signature Certificate Information");

      ExtractCertificateInformation(csi.SignatureCertificate);

      Console.WriteLine("Signature Is Valid: " + csi.SignatureIsValid.ToString());
      Console.WriteLine("Signature Time: " + csi.SignatureTime.ToLocalTime().ToString());
      Console.WriteLine("Is Timestamped: " + csi.SignatureIsTimestamped);
if (csi.SignatureIsTimestamped == true)
{
      Console.WriteLine("Hash Algorithm: " + csi.TimestampInfo.HashAlgorithm.FriendlyName);
      Console.WriteLine("Is TimestampAltered: " +
csi.TimestampInfo.IsTimestampAltered.ToString());
      Console.WriteLine("TimestampSerial Number: " + csi.TimestampInfo.SerialNumber);
      Console.WriteLine("TSA Certificate: " + csi.TimestampInfo.TsaCertificate.Subject);
}//if

      Console.WriteLine(Environment.NewLine);
} //foreach
} //method
```

Merge Multiple PDF Files into a Single PDF File

If you need to merge multiple PDF files into a single one, use the following code:

```
using SignLib.Pdf;

List<byte[]> sourceFiles = new List<byte[]>();

sourceFiles.Add(File.ReadAllBytes("d:\\1.pdf"));
sourceFiles.Add(File.ReadAllBytes("d:\\2.pdf"));
sourceFiles.Add(File.ReadAllBytes("d:\\3.pdf"));
sourceFiles.Add(File.ReadAllBytes("d:\\4.pdf"));

File.WriteAllBytes("d:\\merge.pdf", PdfMerge.MergePdfFiles(sourceFiles));
```

Insert Texts and Images in a PDF file

```
using SignLib.Pdf;

PdfInsertObject PdfInsertImage = new PdfInsertObject();

/**************************
Insert images on PDF document
**************************/
PdfInsertImage.LoadPdfDocument("c:\\source.pdf");

//adds an image on a specific rectangle location on the page 1. The image will be
placed over the PDF content of the page.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\watermark.png"), new
System.Drawing.Rectangle(10, 10, 100, 100), 1, ImagePosition.ImageOverContent);

//adds an image that will cover all the page 2. The image will be placed under the
PDF content (backgorund) of the page.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\watermark.png"), 2,
ImagePosition.ImageUnderContent);

//adds an image that will start on a specific starting position on the page 3. The
image will not be resized. The image will be placed over the PDF content of the
page.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\watermark.png"), new
System.Drawing.Point(200, 200), 3, ImagePosition.ImageOverContent);

//adds an image on the top right corner of the document.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\signature_image.jpg"), new
System.Drawing.Rectangle(PdfInsertImage.DocumentProperties.DocumentPageSize(4).X -
100, PdfInsertImage.DocumentProperties.DocumentPageSize(4).Y - 100, 100, 100), 4,
ImagePosition.ImageOverContent);

//adds an image on the top left corner of the document.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\signature_image.jpg"), new
System.Drawing.Rectangle(0,
PdfInsertImage.DocumentProperties.DocumentPageSize(5).Y - 100, 100, 100), 5,
ImagePosition.ImageOverContent);
```

```
//adds an image on all document pages over the text.
PdfInsertImage.AddImage(File.ReadAllBytes("c:\\certificate_graphic.png"), new
System.Drawing.Point(100, 100), 0, ImagePosition.ImageOverContent);

//adds an image on all document pages under the text in the middle.
PdfInsertImage.AddImage(File.ReadAllBytes("c\\watermark.png"), new
System.Drawing.Rectangle(PdfInsertImage.DocumentProperties.DocumentPageSize(3).X /
2, PdfInsertImage.DocumentProperties.DocumentPageSize(3).Y / 2, 100, 100), 0,
ImagePosition.ImageUnderContent);

/**************************
Insert texts on PDF document
***************************/

CustomText custText = new CustomText();
custText.Align = TextAlign.Left;
custText.FontFile = "c:\\arial.ttf";
custText.FontSize = 8;
custText.PageNumber = 1;
custText.StartingPointPosition = new System.Drawing.Point(100, 100);
custText.Text = "The first text inserted";
custText.TextColor = iTextSharp.text.Color.BLUE;

PdfInsertImage.AddText(custText); //add the first text

CustomText custText2 = new CustomText();
custText2.Align = TextAlign.Left;
custText2.FontFile = "c:\\arial.ttf";
custText2.FontSize = 6;
custText2.PageNumber = 1;
custText2.StartingPointPosition = new System.Drawing.Point(80, 150);
custText2.TextDirection = TextDirection.RightToLeft;
custText2.Text = "יהול קופה, ניהול מלאאי";
custText2.TextColor = iTextSharp.text.Color.BLACK;

PdfInsertImage.AddText(custText2); //add the second text

//insert objects and save the PDF file
File.WriteAllBytes("c:\\destination.pdf", PdfInsertImage.InsertObjects());
```

CAdES Digital Signatures

The library can be used to create and verify CAdES (or PKCS#7/CMS) digital signatures.

Creating CAdES Signatures

```
using SignLib.Certificates;
using SignLib.Cades;

CadesSignature cs = new CadesSignature(serialNumber);

ps.RemoteSignatureServer = new
Uri("http://ca.signfiles.com/RemoteSigner/RemoteSignature.asmx");

//optionally, the signature can be timestamped.
//cs.TimeStamping.ServerUrl = new Uri("http://ca.signfiles.com/TSAServer.aspx");

//write the signed file
//usually, the signed CAdES file should be saved with .p7s or .p7m extension

File.WriteAllBytes("d:\\test.txt.p7s", cs.ApplyDigitalSignature("d:\\test.txt"));

Console.WriteLine("The CAdES signature was created." + Environment.NewLine);
```

Verifying CAdES Signatures

```
using SignLib.Certificates;
using SignLib.Cades;

void ExtractCertificateInformation(X509Certificate2 cert)
{
            Console.WriteLine("Certificate subject:" + cert.Subject);
            Console.WriteLine("Certificate issued by:" +
cert.GetNameInfo(X509NameType.SimpleName, true));
            Console.WriteLine("Certificate will expire on: " +
cert.NotAfter.ToString());
            Console.WriteLine("Certificate is time valid: " +
DigitalCertificate.VerifyDigitalCertificate(cert,
VerificationType.LocalTime).ToString());
}

CadesVerify cv = new CadesVerify("d:\\test.txt.p7s", serialNumber);

Console.WriteLine("Number of signatures: " + cv.Signatures.Count.ToString());

//verify every digital signature from the signed document
foreach (CadesSignatureInfo csi in cv.Signatures)
{
Console.WriteLine("Hash Algorithm: " + csi.HashAlgorithm.FriendlyName);
Console.WriteLine("Signature Certificate Information");

ExtractCertificateInformation(csi.SignatureCertificate);

Console.WriteLine("Signature Is Valid: " + csi.SignatureIsValid.ToString());
Console.WriteLine("Signature Time: " +
csi.SignatureTime.ToLocalTime().ToString());
Console.WriteLine("Is Timestamped: " + csi.SignatureIsTimestamped);

if (csi.SignatureIsTimestamped == true)
{
Console.WriteLine("Hash Algorithm: " +
csi.TimestampInfo.HashAlgorithm.FriendlyName);
Console.WriteLine("Is TimestampAltered: " +
csi.TimestampInfo.IsTimestampAltered.ToString());
Console.WriteLine("TimestampSerial Number: " + csi.TimestampInfo.SerialNumber);
Console.WriteLine("TSA Certificate: " + csi.TimestampInfo.TsaCertificate.Subject);
}
Console.WriteLine(Environment.NewLine);
}
```

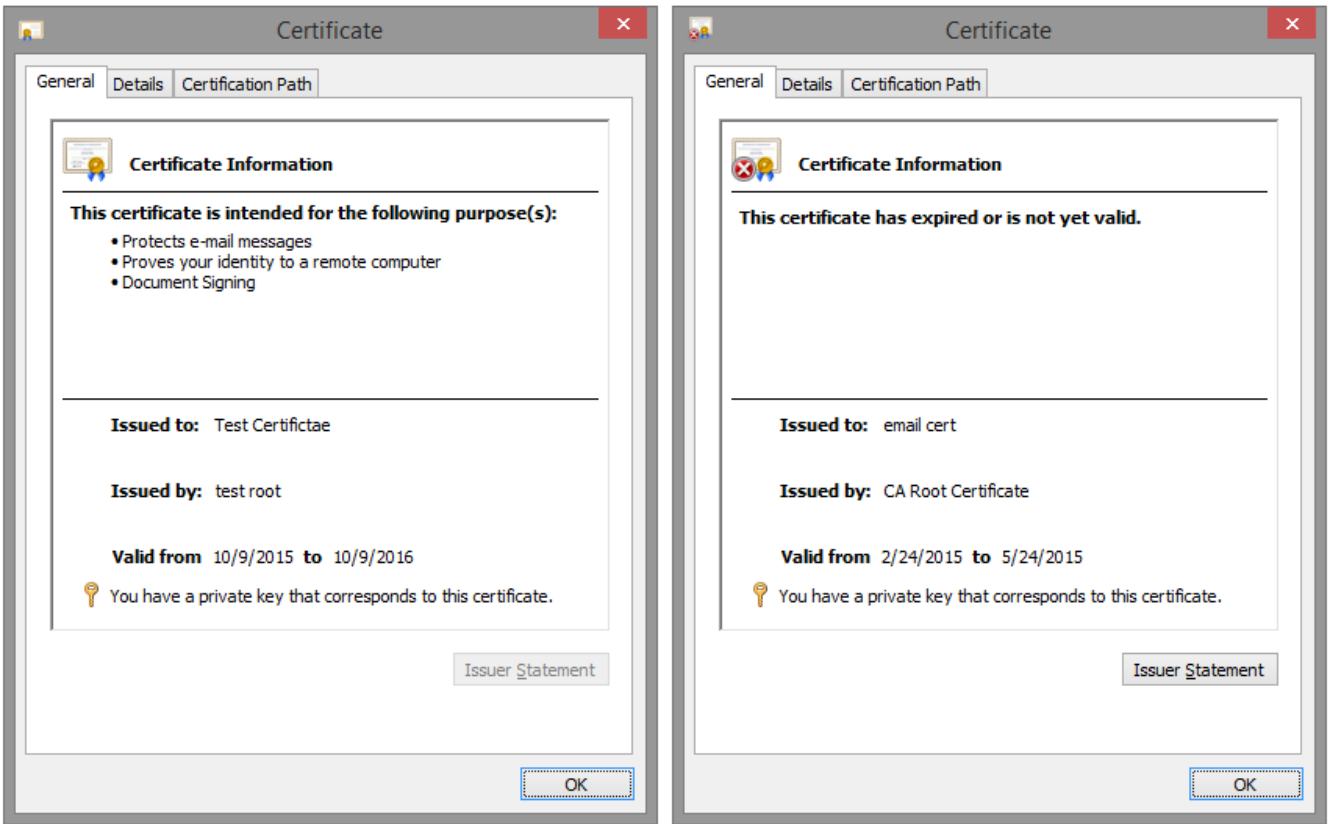
Validating Digital Certificates

A digital certificate can be validated agains three criteria: Local time, CRL (Certificate Revocation List) and OCSP (Online Certificate Status Protocol).

Observation: Not all certificates have CRL and OCSP.

Local Time Validation

Every certificate is valid for a limited period. If a certificate is expired, it should not be used to perform digital signtures.



Time valid certificate versus an expired certificate

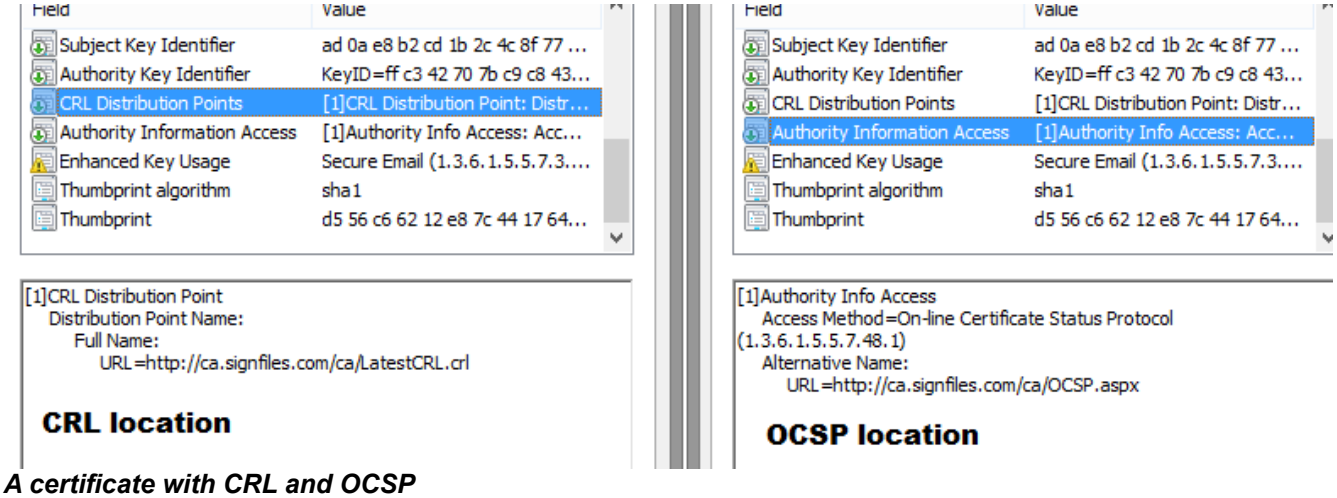
CRL and OCSP Validation

For some reasons, a digital certificate could be revoked before expiration date (e.g. a person leaves the company, the person lost the smart card, forgot the PIN, etc.).

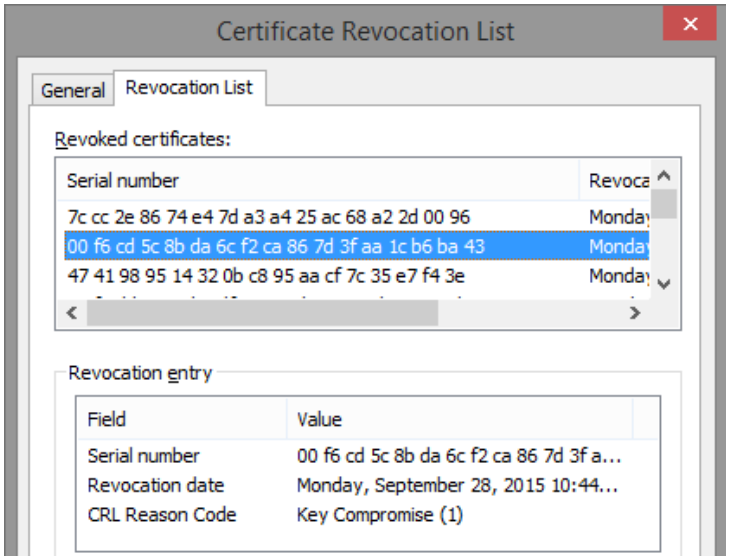
When a certificate is revoked, the certificate serial number is added on the CRL. To verify if a certificate is revoked, the CRL must be downloaded and check if the certificate serial number appears on the CRL.

If the certificate serial number appears on the CRL, it is considered revoked.

In some cases, the CRL is very large (more than 1MB). On this case, the OCSP protocol verifies only a specific serial number instead downloading the entire CRL file.

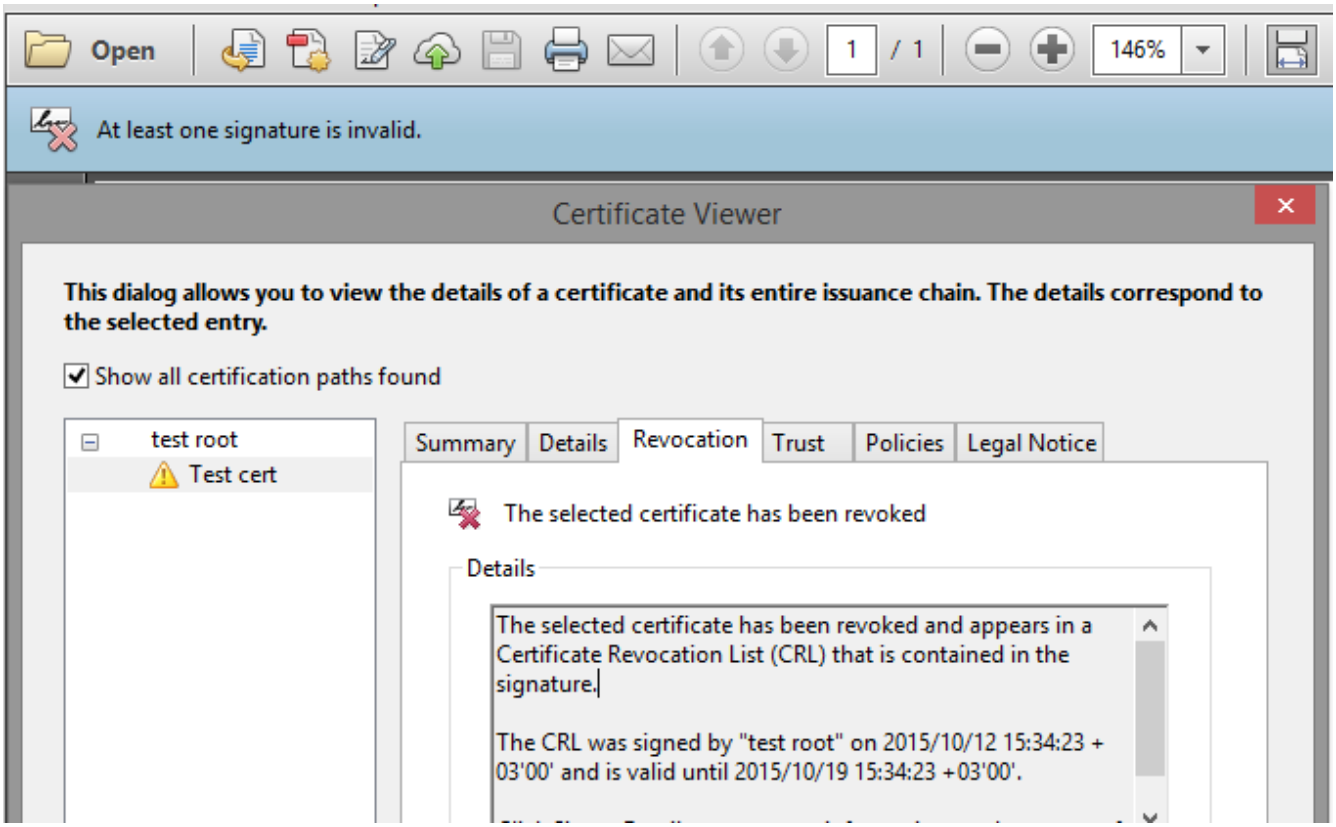


A certificate with CRL and OCSP

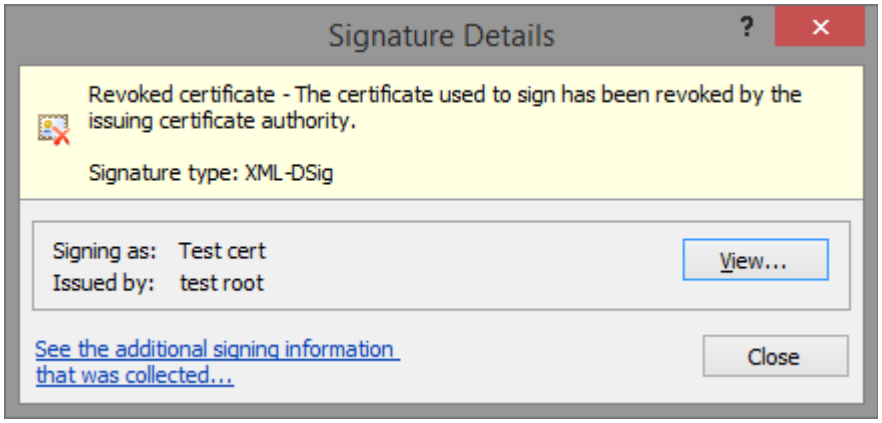


A CRL file contains revoked certificates

If a revoked certificate is used for digital signature, a proper message will appear.



A revoked certificate was used to digitally sign a PDF file



A revoked certificate was used to digitally sign an Office document

Validating Digital Certificates - Code Sample

```
//check if the certificate is time valid
X509Certificate2 certificate = DigitalCertificate.LoadCertificate("d:\\cert.pfx",
"123456");

if (certificate == null)
      throw new Exception("No certificate was found or selected.");

Console.WriteLine("Verify against the local time: " +
DigitalCertificate.VerifyDigitalCertificate(certificate,
VerificationType.LocalTime));

Console.WriteLine("Verify against the CRL: " +
DigitalCertificate.VerifyDigitalCertificate(certificate, VerificationType.CRL));

Console.WriteLine("Verify against the OCSP: " +
DigitalCertificate.VerifyDigitalCertificate(certificate, VerificationType.OCSP));

//CertificateStatus.Expired - the certificate is expired
//CertificateStatus.Revoked - the certificate is revoked
//CertificateStatus.Unknown - the CRL or the OCSP service is unavailable
//CertificateStatus.Valid - the certificate is OK
```