.NET Digital Signature Library SDK
User Manual

Introduction

The main function of **.NET Digital Signature Library SDK** is to digitally sign files in PDF or PKCS#7 cryptographic standard (.P7S or .P7M files) using X.509 certificates stored on PFX files, smart cards, crypto tokens, HSM's stored on Microsoft Certificate Store.

The positioning of the PDF signature appearance is configurable, plus on which pages of the document it should appear (first page, last page or all pages).

Also, using **.NET Digital Signature Library SDK** can digitally sign Office 2007, 2010, 2013 XPS and XML documents using X.509 certificates. Using this library you can quickly digitally sign .docx, .xlsx, .pptx, .xps and .xml files using a simple SDK.

.NET Digital Signature Library SDK can be used to create X.509 certificates in PFX format. Using this library you can quickly create PFX digital certificates and custom certificates with different Key usage or Enhanced key usage.

The main function of *X509CertificateGenerator* class is to issue X.509 Version 3 digital certificates in PFX format. Using this library you can quickly issue all kind of certificates (user, self signed, root, time stamping, digital signature).

Links

.NET Digital Signature Library SDK: http://www.signfiles.com/sdk/SignatureLibrary.zip
.NET Digital Signature Library main page: http://www.signfiles.com/file-sign-library/

Warning and Disclaimer

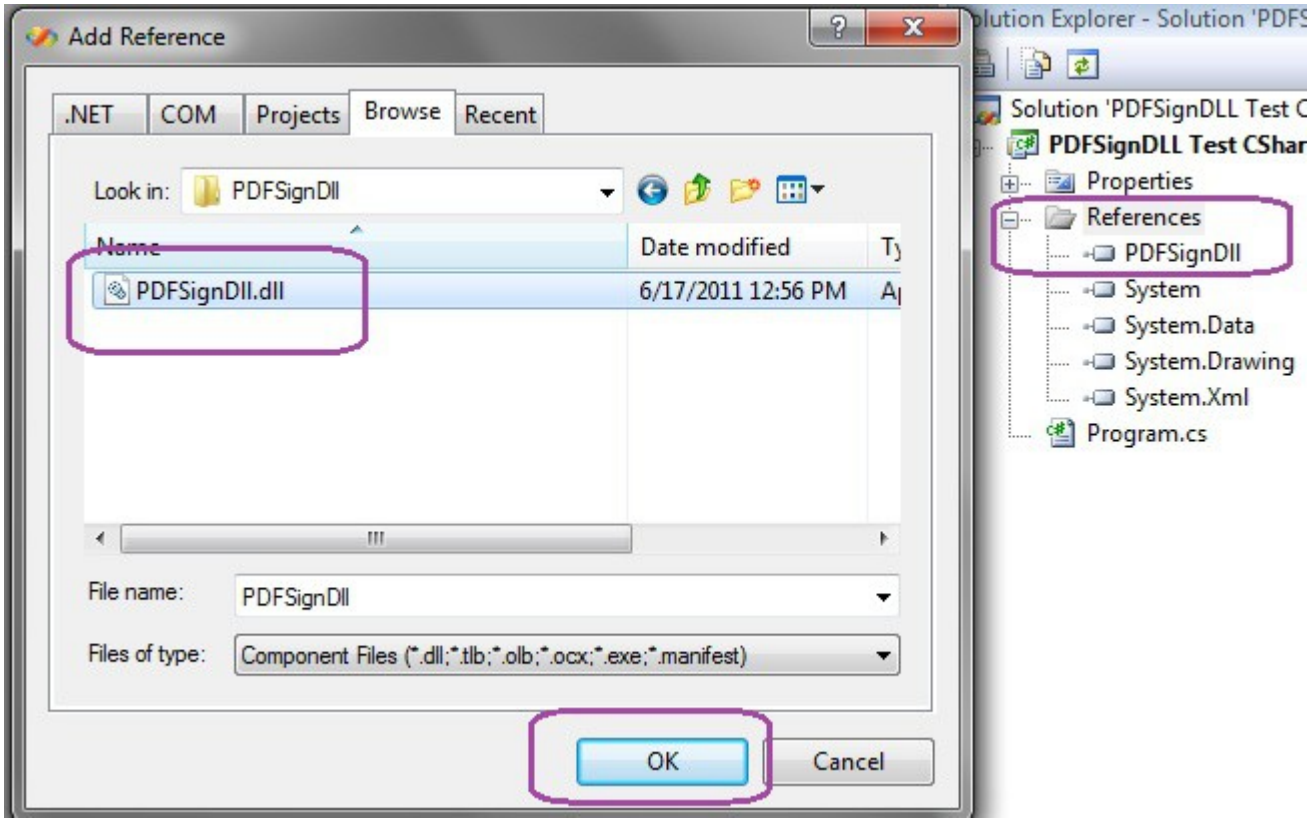
Every effort has been made to make this manual as complete and accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from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

How to use .NET Digital Signature Library SDK in Visual Studio

- Unzip the file and copy the *PDFSignDll.dll / FileSignDll.dll* and *PDFSignDll.xml/ FileSignDll.xml* on your project location.
- In your project, go to *References,* select *Add Reference..,*select the *PDFSignDll.dll/ FileSignDll.dll* as below.



Adding as reference PDFSignDll library

Notes:
PDFSignDll.dll can be used to create native PDF digital signatures.
FileSignDll.dll can be used to create PKCS#7, Office 2007-2013, XPS and XML digital signatures.

Digital Certificates

Digital Certificates Used for Digital Signatures

To create a digital signature, a digital certificate is needed. The digital certificates are stored in two places:

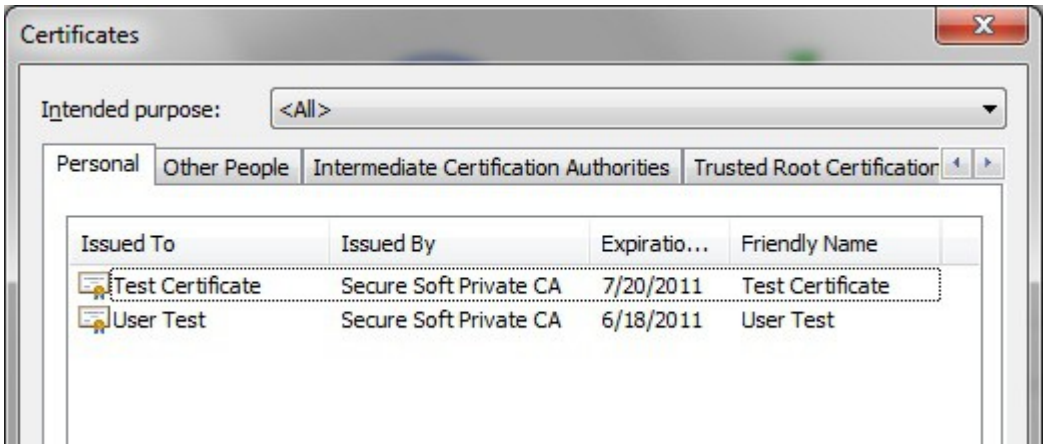
- in Microsoft Store
- in PFX on P12 files

The certificates stored on **Microsoft Store** are available by opening *Internet Explorer* – *Tools* menu – *Internet Options* – *Content* tab – *Certificates* button (see below).

For digital signatures the certificates stored on *Personal* tab are used. These certificates have a public and a private key.

The digital signature is created by using the private key of the certificate. The private key can be stored on the file system (imported PFX files), on an cryptographic smart card (like Aladdin eToken or SafeNet iKey) or on a HSM (Hardware Security Module).

For encryption, only the public key of the certificate is necessary (certificates stored on *Personal* or *Other People* tabs).



Signing certificates available on Microsoft Store

Another way to store a digital certificate is a **PFX (or P12) file**. This file contain the public and the private key of the certificate. This file is protected by a password in order to keep safe the key pair.

Note that a PFX file can be imported on Microsoft Store (just open the PFX file and follow the wizard).

Create a Digital Certificate Using *X509CertificateGenerator* Class

Every certificate must have a *Subject*. The *Subject* can contains Unicode characters like *ä,æ, £, Ñ*.

Every certificate has a validity period. A certificate becomes invalid after it expires.

The default value of *ValidFrom* property is *DateTime.Now* (curent date).
The default value of *ValidTo* property is *DateTime.Now.AddYears(1).*

Observation: On the demo version of the library, the certificate validity cannot exceed 30 days (this is the single limitation of the library on the demo version).

```
X509CertificateGenerator cert = new X509CertificateGenerator("serial number");
//set the certificate Subject
cert.Subject = "CN=Certificate name,E=name@email.com,O=Organization";

//the certificate becomes valid after 4th February 2012
cert.ValidFrom = new DateTime(2012, 2, 4);

//the certificate will expires on 25th February 2012
cert.ValidTo = new DateTime(2012, 2, 25);

//save the PFX certificate on a file
File.WriteAllBytes("c:\\cert.pfx", cert.GenerateCertificate("password", false));
```

More details about *X509CertificateGenerator* class can be found on the corresponding section below.

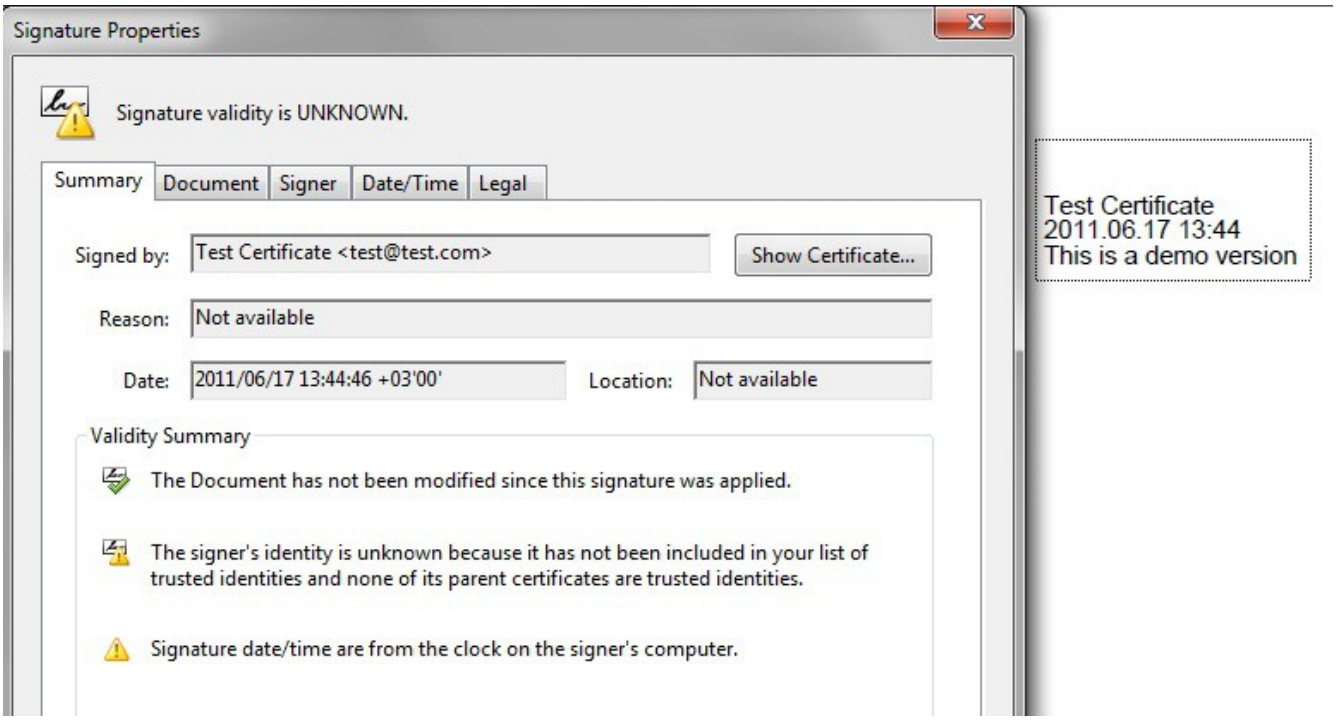
Digitally Sign a PDF File Using a Digital Certificate Stored on a PFX File

The code below demonstrates how to digitally sign a PDF file using a PFX certificate.

```
//Initializes a new instance of PDFSign class
PDFSign PDFSign = new PDFSign("serial number");

//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

//Load the certificate from .PFX
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "PFX certificate
password");

//Set the signature rectangle attributes
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopRight;

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

When the *dest.pdf* is opened in Adobe Reader, a signature rectangle appear on the top right corner.

When the signature rectangle is clicked, the digital signature information appears.



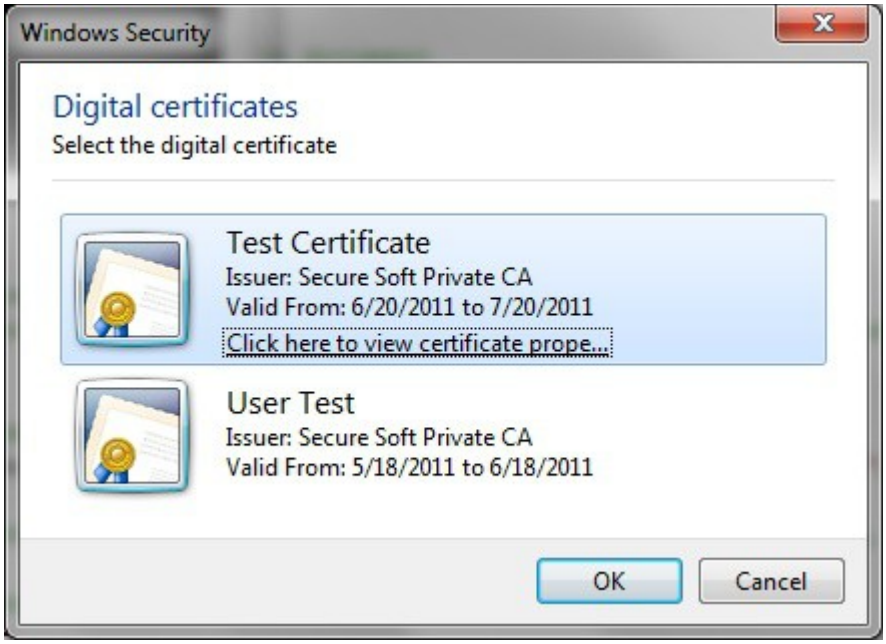
Digital signature properties on Adobe Reader

Digitally Sign a PDF File Using a Digital Certificate Stored on Microsoft Store

To digitally sign a PDF using a certificate stored on Microsoft store use this line of code:

```
PDFSign.DigitalSignatureCertificate = PDFSign.LoadCertificate(false, "", "Digital
certificates", "Select the digital certificate",
DigitalCertificateScope.ForDigitalSignature);
```

When the project is launched, the user must select the digital certificate from all certificates available in *Personal* tab.



Digital certificates selection window

Probably you want to digitally sign a PDF file without user intervention using a certificate available on Microsoft Store. For that you must identify that certificate by a criteria.

If your desired certificate has in the *Subject* filed the value *E = email@email.com,* you can use the following code to automatically use thie certificate for the signing operation.

```
PDFSign.DigitalSignatureCertificate = PDFSign.LoadCertificate(false,
DigitalCertificateSearchCriteria.EmailE, "test@test.com",
DigitalCertificateScope.ForDigitalSignature);
```

Note that there are a lot of criteria to automatically select your certificate (Common Name, Serial number, Thumbprint, etc.).

Validating Digital Signatures in Adobe

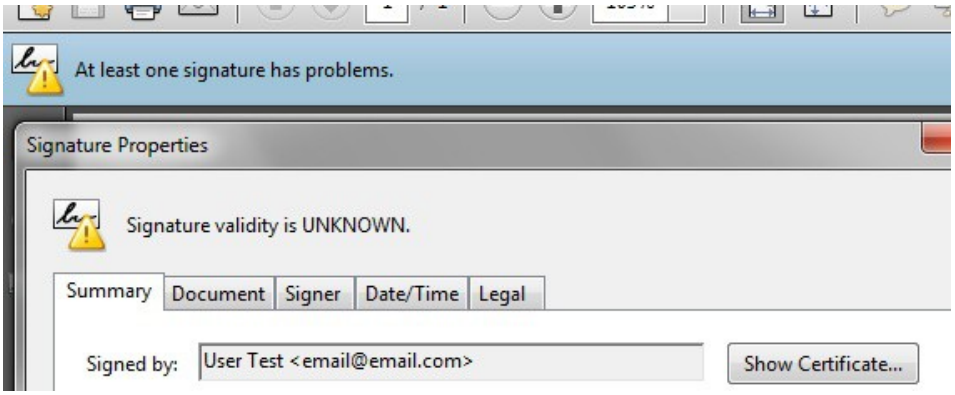
Every digital certificate is issued by a Root CA (Certification Authority). Some of the Root CA's are included by default in Windows Certificate Store (Trusted Root Certification Authorities) and only a few are included in Adobe Certificate Store. Microsoft and Adobe use different Certificate Stores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

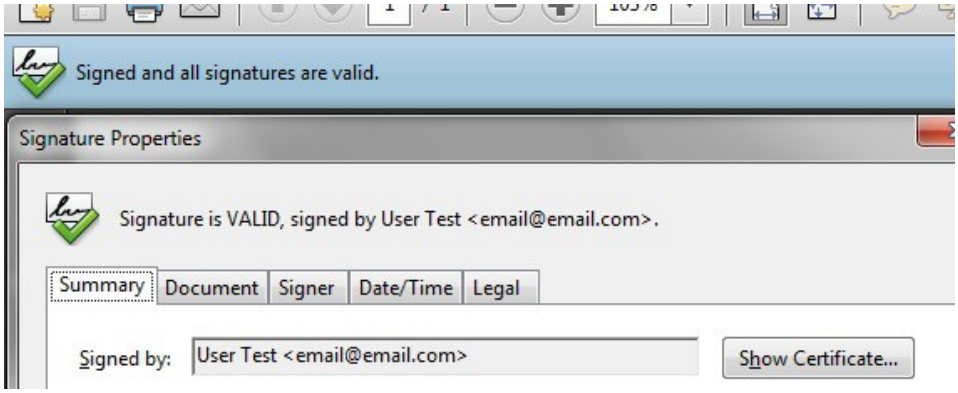
This behavior has nothing to do with the signing engine but with the Adobe certification validation procedure.

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

Digitally Sign a PDF Document with PDFSignDLL

Loading the PDF

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

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

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

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

Digitally Sign an Encrypted PDF File

To digitally sign an encrypted PDF file you must first provide the protection password like below:

```
//Initializes a new instance of PDFSign class
PDFSign PDFSign = new PDFSign("serial number");

//set the document password first
PDFSign.DocumentProperties.Password = "document password";

//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.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.

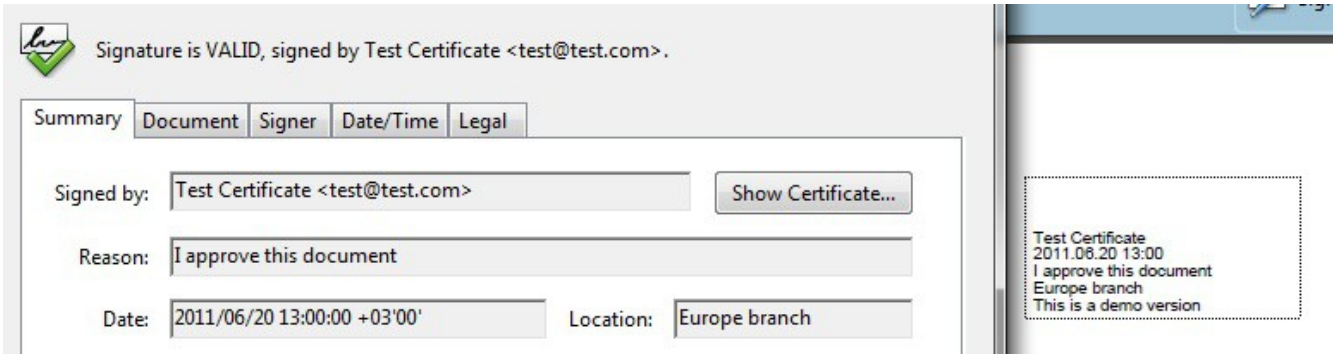
```
//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

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

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

Set the Digital Signature Properties (Reason, Location)

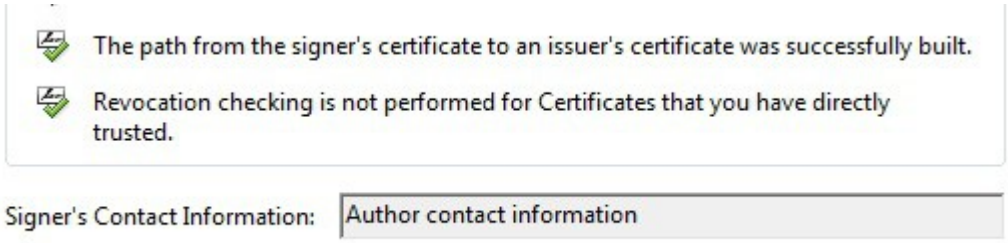
Adobe digital signatures can be fully customized by PDFSignDLL SDK. In order to set the Reason, Location, Date, “Signed by” or “Signer's Contact Information” property use the code below.

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.

```
PDFSign.SigningReason = "I approve this document";
PDFSign.SigningLocation = "Europe branch";
PDFSign.SignerContactInformation = "Author contact information";
PDFSign.SignedBy = "Signed by the Author";
PDFSign.SignatureDate = new DateTime(2011, 6, 20, 13, 00, 00);
```



Signed by, Reason, Location and Date properties in Adobe



Signer's Contact Information 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:

```
PDFSign.SignaturePage = PDFSign.DocumentProperties.NumberOfPages;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.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:

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

//set the rectangle width and height
int width = 80;
int height = 40;
PDFSign.SignaturePosition = 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:

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

```
PDFSign.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:

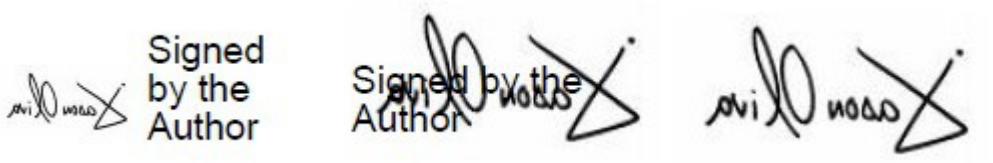
```
PDFSign.FontFile = "c:\\windows\\fonts\\arial.ttf";
PDFSign.FontSize = 10;
```

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:

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

//text on the right and image on the left
PDFSign.SignatureImagePosition = SignatureImageType.ImageAndText;
//image as bakground and text on above
PDFSign.SignatureImagePosition = SignatureImageType.ImageAsBackground;
//only image
PDFSign.SignatureImagePosition = 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
PDFSign.VisibleSignature = false;

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

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:

```
//Initializes a new instance of PDFSign class
PDFSign PDFSign = new PDFSign("serial number");

//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

//Load the certificate from .PFX
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");

//Set the signature rectangle attributes
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopRight;

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

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

Authentication With Username and Password

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

```
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");
PDFSign.TimeStamping.UserName = "username";
PDFSign.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
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");

PDFSign.TimeStamping.AuthenticationCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\time_stamping_certificate.pfx"),
"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.:

```
PDFSign.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.

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

PDFSign.TimeStamping.PolicyOID = new
System.Security.Cryptography.Oid("1.3.7.2.9.1.829.3");
```

Proxy Settings

If you are behind a Proxy server, a TSA request can be done using the following code:

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

//Code example:
//http://msdn.microsoft.com/en-us/library/system.net.httpwebrequest.proxy.aspx

//set the proxy settings
System.Net.WebProxy tsaProxy = new System.Net.WebProxy();
tsaProxy.Address = new Uri("http://myproxy.example.com:1024");
tsaProxy.Credentials = new System.Net.NetworkCredential("username", "password");

PDFSign.ProxySettings = tsaProxy;
```

Validating the Time Stamping Response on Adobe

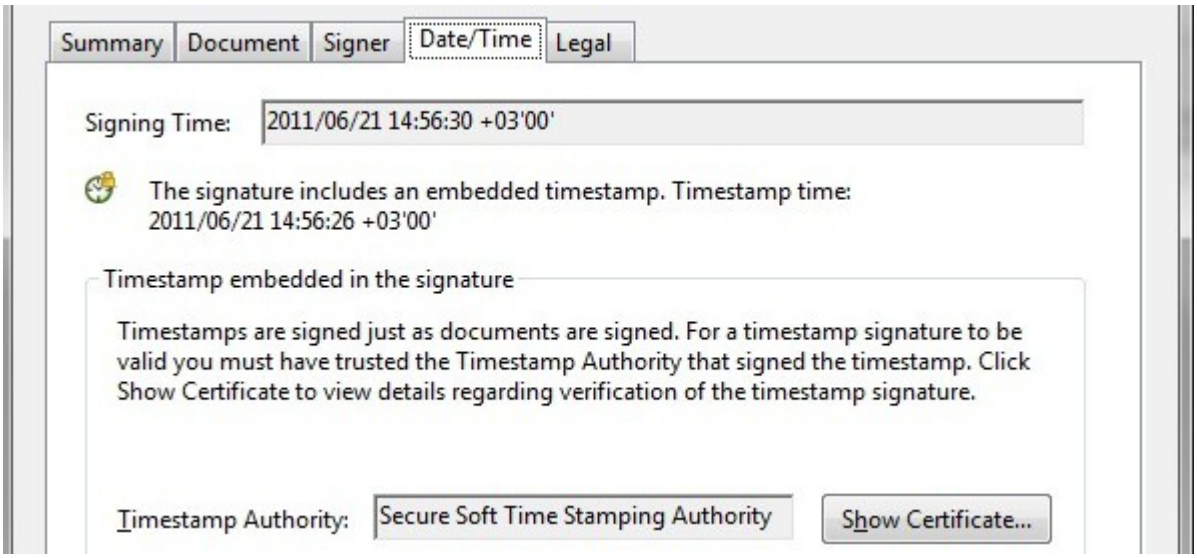
As digital signatures certificates, the time stamping responses are signed by a certificate issued by a Certification Authority.

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.

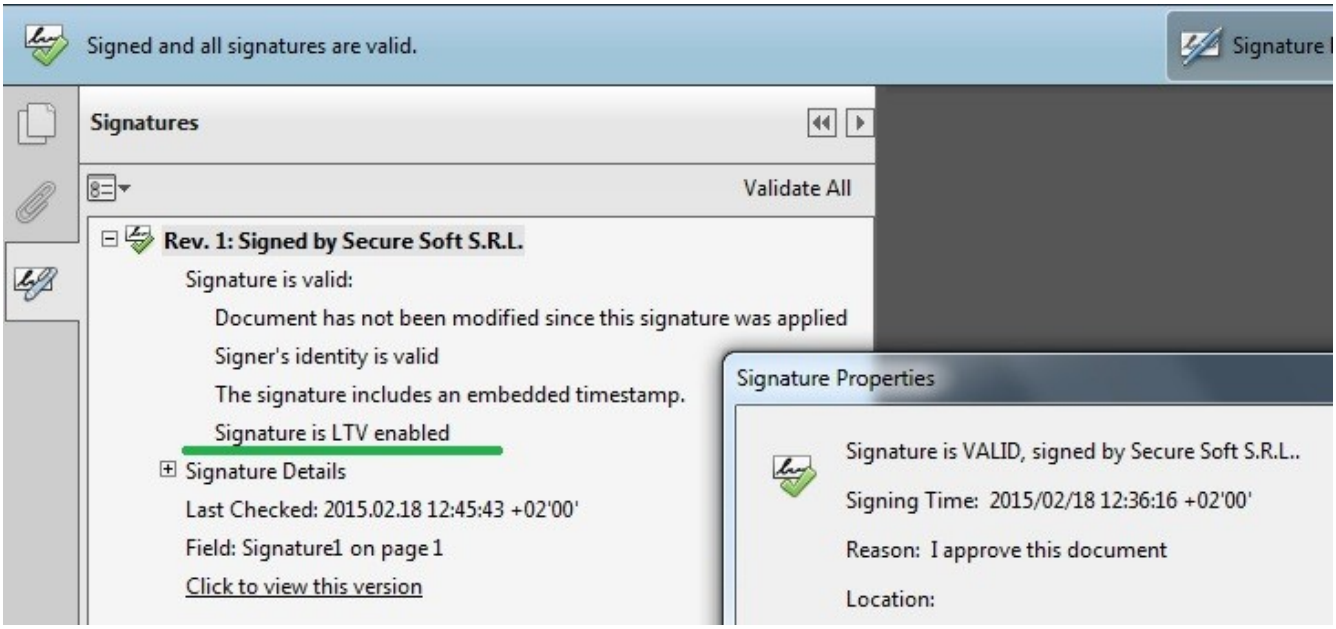
Not verified timestamp



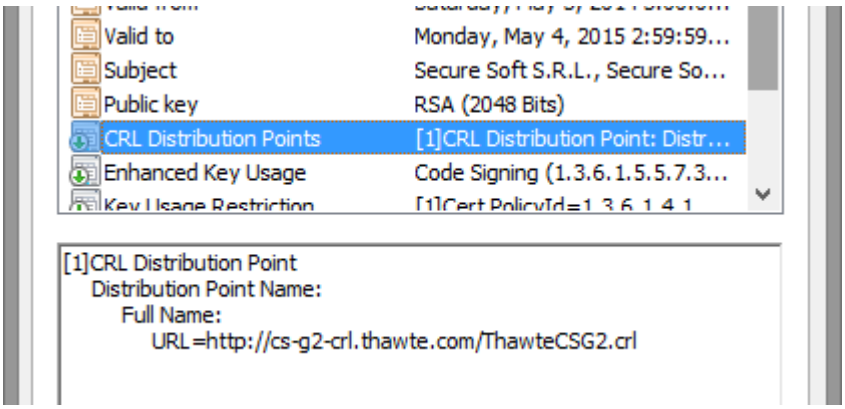
Trusted time stamping response

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.

```
PDFSign.IncludeCRLRevocationInfo = true;
```

Certify a PDF Digital Signature

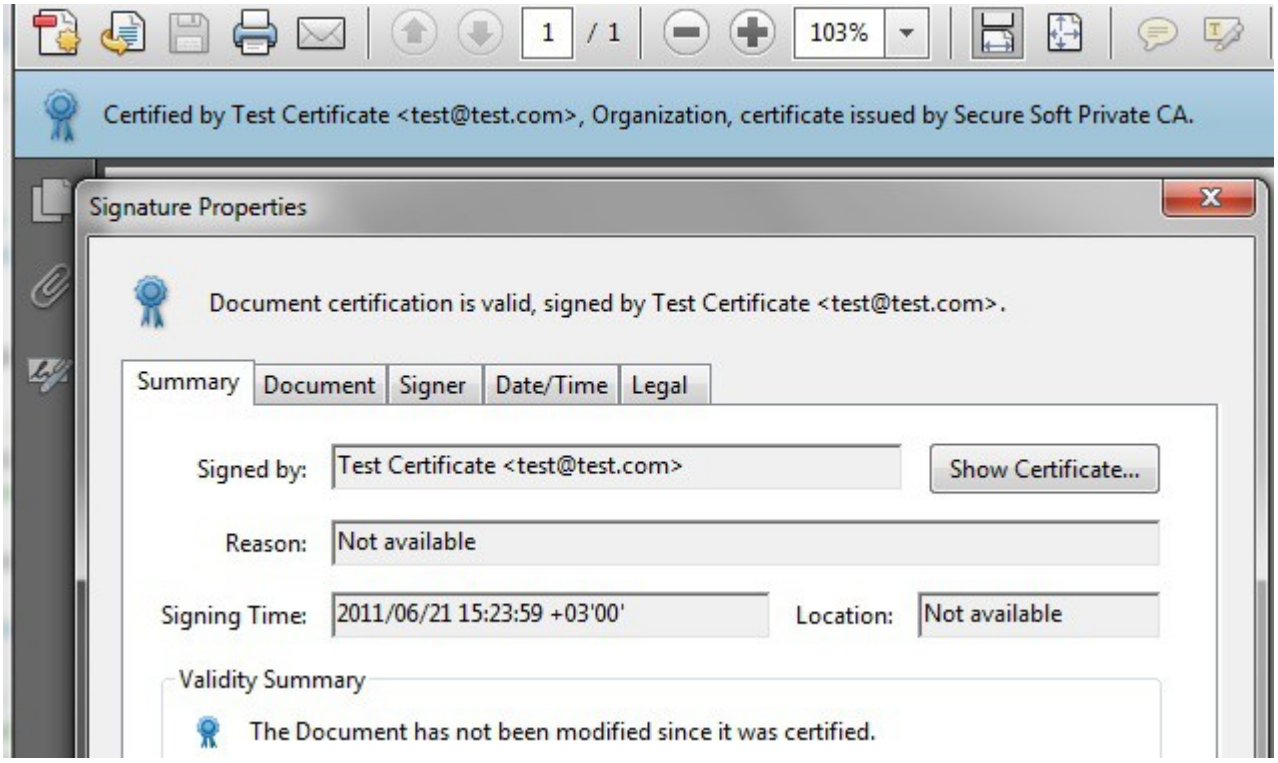
When you certify a PDF, you indicate that you approve of its contents. You also specify the types of changes that are permitted for the document to remain certified.

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:

```
//adding annotations and form filling are allowed
PDFSign.CertifySignature = CertifyMethod.AnnotationsAndFormFilling;
//form filling is allowed
PDFSign.CertifySignature = CertifyMethod.FormFilling;
//no changes allowed
PDFSign.CertifySignature = CertifyMethod.NoChangesAllowed;

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



Certified signature

Digital Signatures and the PDF/A Standard

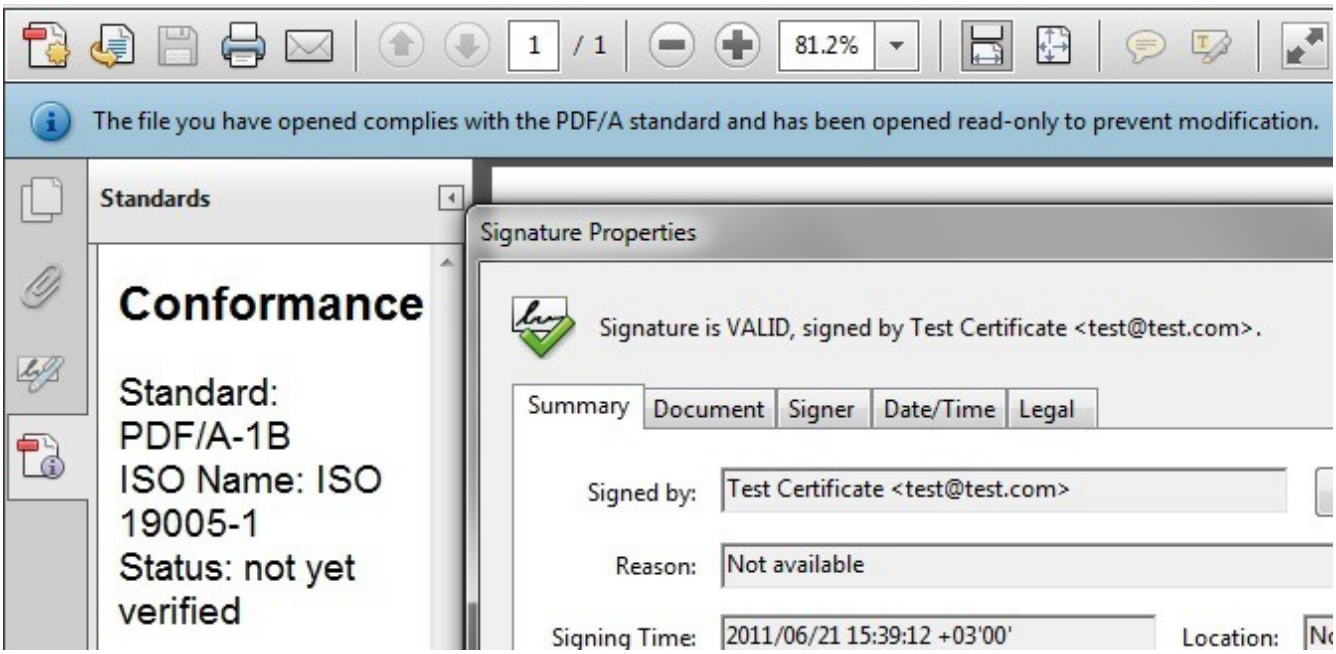
PDF/A is a file format for the long-term archiving of electronic documents. It is based on the PDF Reference Version 1.4 from Adobe Systems Inc. (implemented in Adobe Acrobat 5 and latest versions) and is defined by ISO 19005-1:2005.

PDFSignDLL 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:

```
//Initializes a new instance of PDFSign class
PDFSign PDFSign = new PDFSign("serial number");

//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

//Load the certificate from .PFX
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopRight;

PDFSign.SaveAsPDFA = true;
PDFSign.FontFile = "c:\\windows\\fonts\\arial.ttf";

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



PDF/A-1b document with digital signature

Other features of the PDFSignDll Library

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):

```
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopRight;

PDFSign.SignatureToAllPages = 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.

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

```
PDFSign.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:

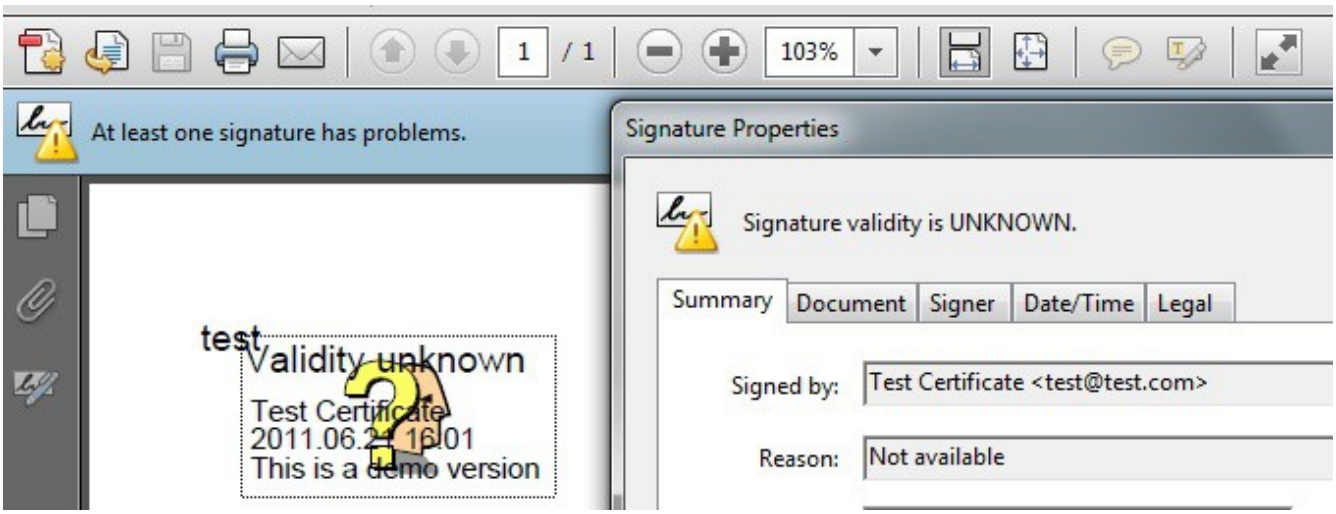
```
PDFSign.SignatureByteBlockSize = 8192;
```

Old Style Adobe Digital Signature Appearance

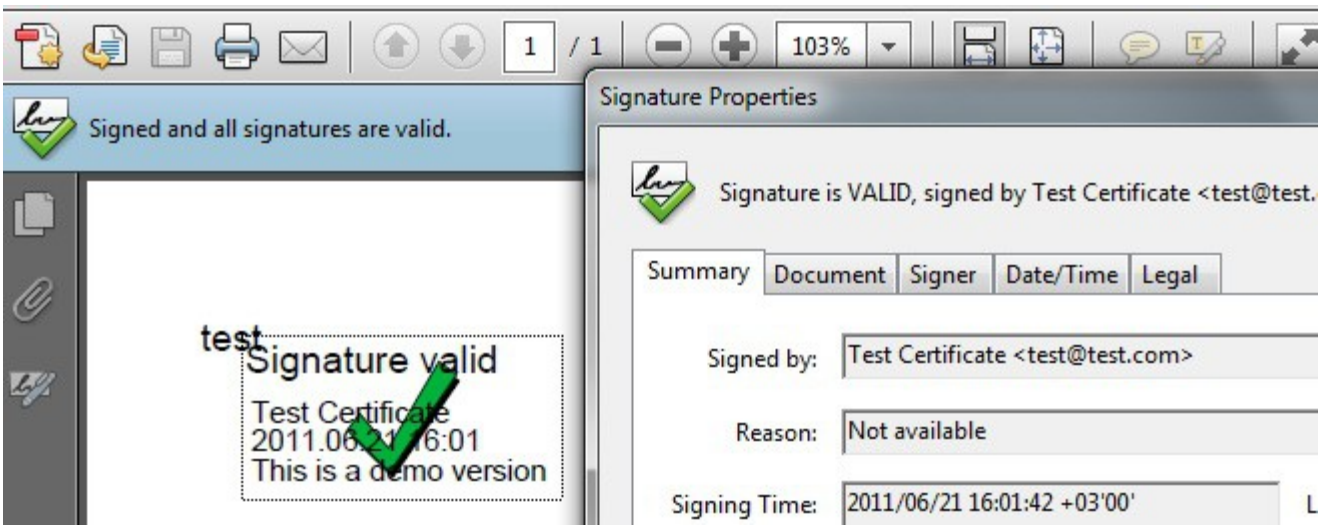
To use an old style appearance of the digital signature rectangle (see example) set the *OldStyleAdobeSignature* property to true. The default value is false.

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

```
PDFSign.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 true.

To NOT include the revocation information, set the property to false.

```
PDFSign.IncludeCRLRevocationInfo = false;
```

Hash Algorithms

The default (and recommended) hash algorithm used by the library is **SHA1** but in some cases, SHA256/384/512 must be used (note that not all PDF readers can validate a such of signature). To set the hash algorithm, use the following code:

```
PDFSign.HashAlgorithm = DigestAlgorithm.SHA256;
```

Also, the titme stamping requests/responses could be made using other hash algorithms, as follow. Note that the time stamping server must support this algorithms.

```
PDFSign.TimeStamping.HashAlgorithm = DigestAlgorithm.SHA512;
```

Verifying the Signature Certificate

The signing certificate can be verified against OCSP and CRL before the signing operation.

```
PDFSign.VerifyDigitalCertificate(PDFSign.DigitalSignatureCertificate);
```

Bypass the Smart Card PIN Dialog

The smart card password dialog can be bypassed by specifying the smart card PIN like below:

```
PDFSign.SmartCardPIN = "SmartCardPassword";
```

Because of some .NET Framework limitations, this option is not available for all the smart card types. This property is supported only if the smart card allows to bypass PIN.

PDF Signatures and Encryption

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

Password Security

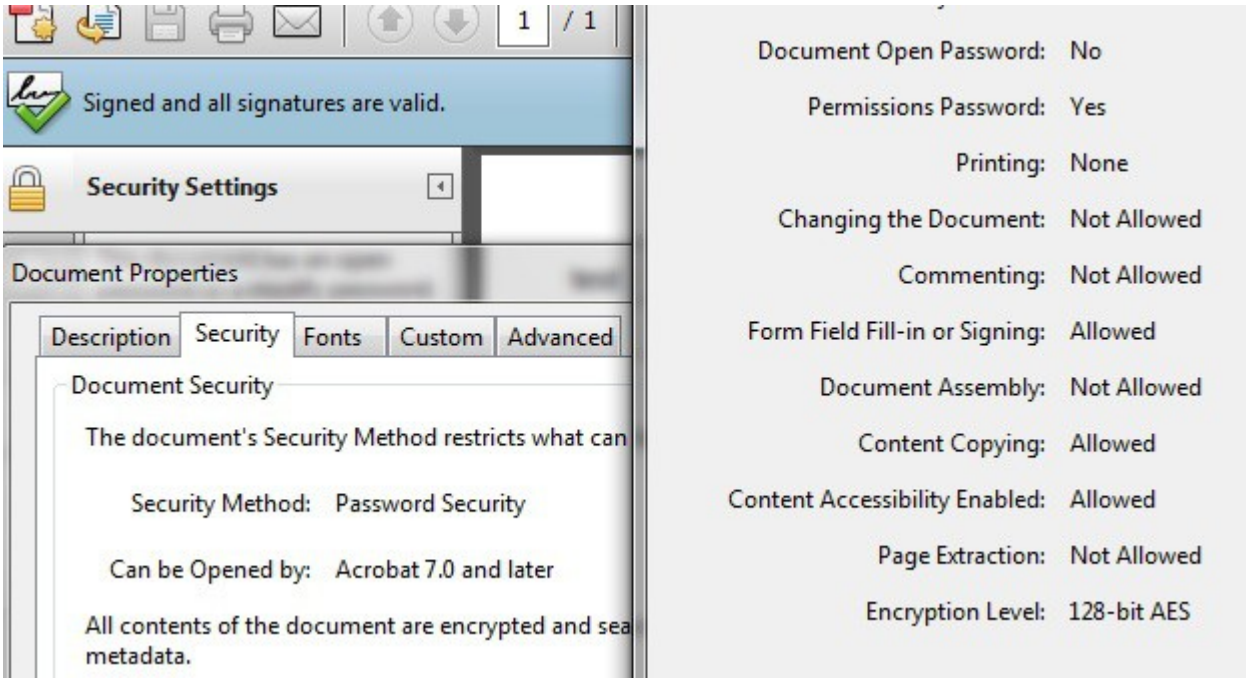
In order to encrypt the PDF document the *AppendSignature* propery must be set to false. Also, the encryption algorithm must be specified using *EncryptionAlgorithm* property.

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:

```
//Initializes a new instance of PDFSign class
PDFSign PDFSign = new PDFSign("serial number");

//Load the PDF file
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));

//Load the certificate from .PFX
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopLeft;

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

//set the document restrictions
PDFSign.Encryption.DocumentRestrictions =
PDFDocumentRestrictions.AllowContentCopying |
PDFDocumentRestrictions.AllowFillingOfFormFields;

//set the encryption algorithm
PDFSign.Encryption.EncryptionAlgorithm =
PDFEncryptionAlgorithm.EnhancedEncryption128BitAES;

//set the encryption method
PDFSign.Encryption.EncryptionMethod = PDFEncryptionMethod.PasswordSecurity;

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

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

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

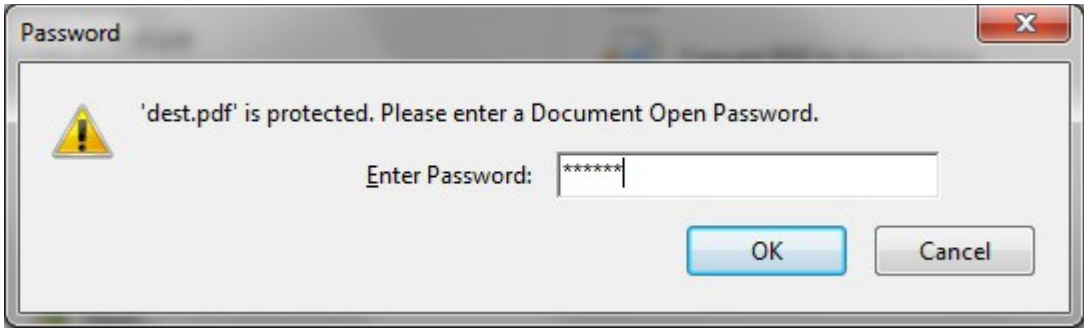


Security settings for a digitally sign and encrypted document

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

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

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



Password is required to open the document

Digital Certificate Security

The document can be also protected using a digital certificate. Remember that the digital signature is created using the private key of the certificate. For the encryption the public key of the certificate is necessary. The public key of the encryption certificates are stored on *Microsoft Store – Other People* tab or in .cer files.

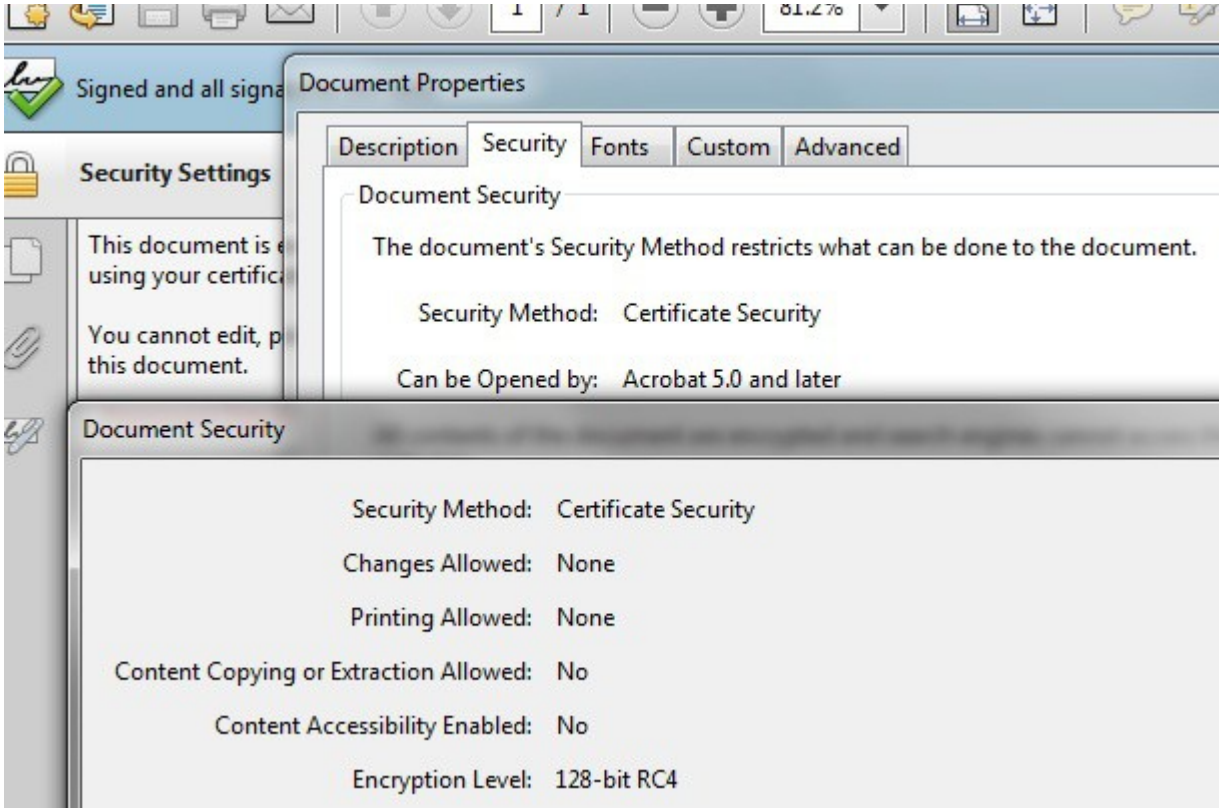
To encrypt a signed message using a digital certificate use the code below:

```
//append signature must be set to false in order to encrypt de document
PDFSign.AppendSignature = false;
//set the document restrictions
PDFSign.Encryption.DocumentRestrictions = PDFDocumentRestrictions.AllowNone;
//set the encryption algorithm
PDFSign.Encryption.EncryptionAlgorithm =
PDFEncryptionAlgorithm.StandardEncryption128BitRC4;
//set the encryption method
PDFSign.Encryption.EncryptionMethod = PDFEncryptionMethod.CertificateSecurity;

//use the digital certificates stored on Microsoft Store - Other People
PDFSign.Encryption.EncryptionCertificate = PDFSign.LoadCertificateFromStore(false,
"", "Encryption certificate", "Select the encryption certificate",
DigitalCertificateScope.ForEncryption);
```

If you want to encrypt the PDF file using a .CER file (public key) use the code below instead of the commented lines:

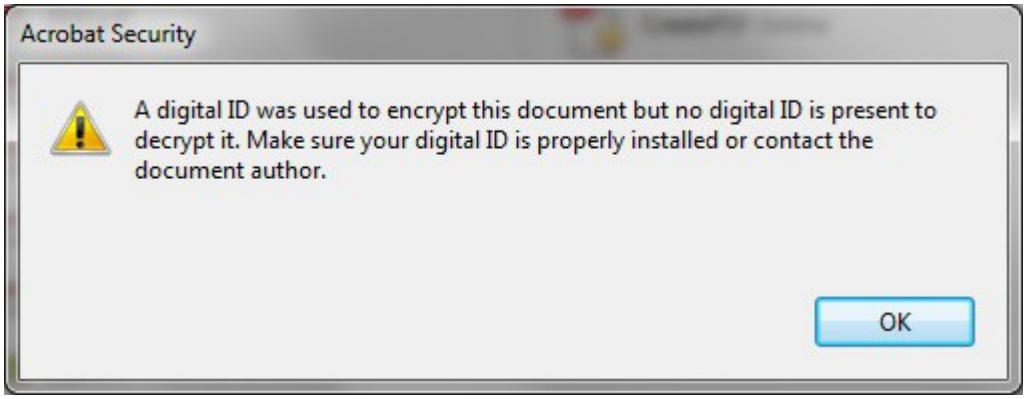
```
//append signature must be set to false in order to encrypt de document
PDFSign.AppendSignature = false;
//set the document restrictions
PDFSign.Encryption.DocumentRestrictions = PDFDocumentRestrictions.AllowNone;
//set the encryption algorithm
PDFSign.Encryption.EncryptionAlgorithm =
PDFEncryptionAlgorithm.StandardEncryption128BitRC4;
//set the encryption method
PDFSign.Encryption.EncryptionMethod = PDFEncryptionMethod.CertificateSecurity;

//PDFSign.Encryption.EncryptionCertificate =
PDFSign.LoadCertificateFromStore(false, "", "Encryption certificate", "Select the
encryption certificate", DigitalCertificateScope.ForEncryption);

//loads the certificate from a .CER file
PDFSign.Encryption.EncryptionCertificate = new
System.Security.Cryptography.X509Certificates.X509Certificate2
(File.ReadAllBytes("c:\\encryption_certificate.cer"));
```

If the private key corresponding to the public key used for encryption is available on the computer where the the encrypted file is opened, the security settings are shown like below:



Security settings for a digitally sign and encrypted document

Observation: A file encrypted with the public key can be opened only by the corresponding private key of that certificate. If you want to encrypt a file for a person, you will need the public key of the certificate issued for that person. If the file is encrypted with your certificate only you can open that file. If the private key of the encryption certificate is not present a warning message will be displayed like below:



Decryption certificate (private key) is not present

PDFSignDll Code Samples

Digitally Sign All Pages From a PDF File with a Certificate Stored on PFX File

```
PDFSign PDFSign = new PDFSign("serial number");
//load the pdf file
PDFSign.LoadPDFDocument("c:\\source.pdf");

//load the certificate
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
//put the signature to all pages
PDFSign.SignatureToAllPages = true;

//set the signature position
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopLeft;

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

Set a Custom Signature Rectangle and Sign Using a Certificate from Microsoft Store

```
PDFSign PDFSign = new PDFSign("serial number");
PDFSign.LoadPDFDocument("c:\\source.pdf");

//load the certificate from Microsoft Store
PDFSign.DigitalSignatureCertificate = PDFSign.LoadCertificate(false, "", "Digital
certificates", "Select the digital certificate",
DigitalCertificateScope.ForDigitalSignature);

PDFSign.SignaturePage = 1;
//set the signature position
System.Drawing.Point pageRectangle = PDFSign.DocumentPageSize(1);

//put the signature on the middle of the page
PDFSign.SignaturePosition = new System.Drawing.Rectangle(pageRectangle.X / 2,
pageRectangle.Y / 2, 100, 50);

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

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

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

//sign the document only if is not signed
if (PDFSign.DocumentProperties.NumberOfDigitalSignatures == 0)
{
      PDFSign.DigitalSignatureCertificate =
      PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
      PDFSign.SignaturePage = 1;
      PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopMiddle;
      File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
}
```

Digitally Sign a PDF file with a PFX Certificate Created on the Fly

```
string certificatePassword = "tempP@ssword";

//create the digital certificate used to digitally sign the PDF document
X509CertificateGenerator cert = new X509CertificateGenerator();

//set the validity of the certificate (2 years from now)
cert.ValidFrom = DateTime.Now;
cert.ValidTo = DateTime.Now.AddYears(2);

//set the signing algorithm and the key size
cert.KeySize = KeySize.KeySize2048Bit;
cert.SignatureAlgorithm = SignatureAlgorithm.SHA1WithRSA;

//set the certificate subject
cert.Subject = "CN=Your User, E=useremail@email.com, O=Organzation";

//add some simple extensions to the client certificate
cert.Extensions.AddKeyUsage(CertificateKeyUsage.DigitalSignature);
cert.Extensions.AddKeyUsage(CertificateKeyUsage.DataEncipherment);

//add some enhanced extensions to the client certificate marked as critical

cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.DocumentSigning);
cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.SecureEmail);



//create the certificate
byte[] digitalCertificate = cert.GenerateCertificate(certificatePassword);

//create the PDF signature
PDFSign PDFSign = new PDFSign("serial number");

//Load the PDF file
PDFSign.LoadPDFDocument("d:\\source.pdf");

//Create a new certificate and load it
PDFSign.DigitalSignatureCertificate = PDFSign.LoadCertificate(digitalCertificate,
certificatePassword);

//Signing reason & location
PDFSign.SigningReason = "I approve this document";
PDFSign.SigningLocation = "Europe branch";

//Set the signature rectangle attributes
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopRight;

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

Set a Custom Text and Font for the Digital Signature Rectangle

```
PDFSign PDFSign = new PDFSign("serial number");

PDFSign.LoadPDFDocument("c:\\source.pdf");
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");

PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.BottomLeft;

//set some signature properties
PDFSign.SigningReason = "Signing reason";
PDFSign.SigningLocation = "Signing location";
PDFSign.SignerContactInformation = "Contact information";

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

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

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

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

```
PDFSign PDFSign = new PDFSign("serial number");

PDFSign.LoadPDFDocument("c:\\source.pdf");
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");

PDFSign.SignaturePage = PDFSign.DocumentProperties.NumberOfPages;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.BottomRight;

PDFSign.SignatureText = "Signed by the author";

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

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

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

Set an Invisible Signature and Certify the PDF file

```
PDFSign PDFSign = new PDFSign("serial number");
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");

PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopMiddle;

PDFSign.SigningLocation = "My location";
PDFSign.SigningReason = "I am the author of this document";

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

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

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

Time Stamp a PDF file

```
PDFSign PDFSign = new PDFSign("serial number");
PDFSign.LoadPDFDocument(File.ReadAllBytes("c:\\source.pdf"));
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");

PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopMiddle;

//simply set the TSA server URL
PDFSign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");

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

Time Stamp a PDF file Using Authentication

```
PDFSign PDFSign = new PDFSign("serial number");
PDFSign.LoadPDFDocument("c:\\source.pdf");
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
PDFSign.SignaturePage = 1;
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopMiddle;
PDFSign.OldStyleAdobeSignature = true;

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

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

Digitally Sign and Time Stamp a folder with PDF files

```
PDFSign PDFSign = new PDFSign("serial number");
PDFSign.LoadPDFDocument("c:\\source.pdf");
PDFSign.DigitalSignatureCertificate =
PDFSign.LoadCertificate(File.ReadAllBytes("c:\\cert.pfx"), "123456");
PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopLeft;
PDFSign.SignaturePage = 1;

PDFSign.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("c:\\source_dir");
rgFiles = di.GetFiles("*.pdf");

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

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

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

      //set the signing certificate
      //the PFX certificate must use MachineKeySet
      PDFSign.DigitalSignatureCertificate = new
System.Security.Cryptography.X509Certificates.X509Certificate2(Server.MapPath("cer
t.pfx"), "123456",
System.Security.Cryptography.X509Certificates.X509KeyStorageFlags.MachineKeySet);

      PDFSign.LoadPDFDocument(Server.MapPath("source.pdf"));

      PDFSign.SignatureBasicPosition = BasicSignatureLocation.TopLeft;
      PDFSign.SignaturePage = 1;

      System.IO.File.WriteAllBytes(Server.MapPath("dest.pdf"),
PDFSign.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:

```
PDFSign PDFSign = new PDFSign("serial number");
PDFSign.LoadPDFDocument("c:\\source.pdf");

foreach (PDFSignatureInformation psi in
PDFSign.DocumentProperties.DigitalSignatures)
{
Console.WriteLine("Signature name: " + psi.SignatureName);
Console.WriteLine("Digest Algorithm: " + psi.DigestAlgorithm);
Console.WriteLine("Signing Reason: " + psi.SigningReason);
Console.WriteLine("Signing location: " + psi.SigningLocation);
Console.WriteLine("Signer: " + psi.DigitalSignatureCertificate.SubjectName.Name);
Console.WriteLine("Signing date: " + psi.SignatureDate.ToString());
Console.WriteLine("Signature is timestamped:" + psi.IsTimeStamped.ToString());
Console.WriteLine("Is signature valid: " + psi.IsSignatureValid.ToString());
}
```

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:

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

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

File.WriteAllBytes("c:\\merge.pdf", PdfMerge.MergeFiles(sourceFiles));
```

Insert Texts and Images in a PDF file

```
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.BaseColor.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.BaseColor.BLACK;

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

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

FileSignDll Library

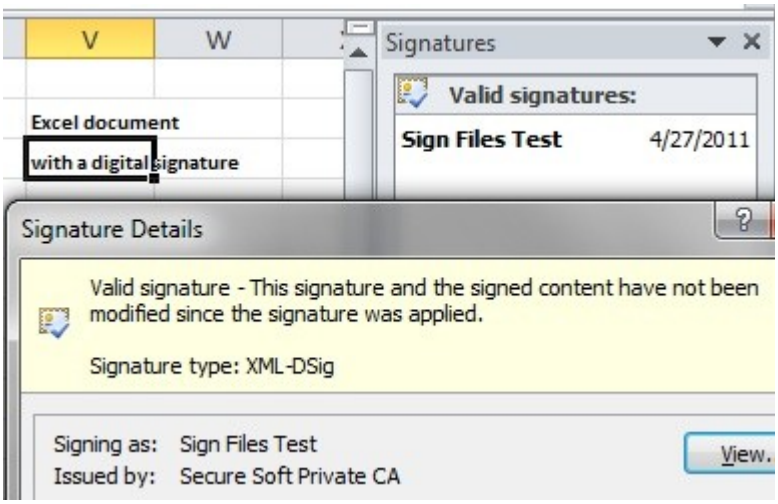
FileSignDll.dll can be used to create PKCS#7 (CMS/CadES), XML, XPS or Office 2007-2013 digital signatures.

Creating PKCS#7 (CMS/CAdES) signatures using FileSignDll

```
PKCS7Sign pkcs7Sign = new PKCS7Sign("serial number");
DigitalCertificates certificate = new DigitalCertificates();

//load the certificate from a PFX file
pkcs7Sign.DigitalSignatureCertificate =
certificate.LoadCertificate("d:\\cert.pfx", "123456");

//optionally, the file can be timestamped
//pkcs7Sign.TimeStamping.ServerURL = new
Uri("http://ca.signfiles.com/TSAServer.aspx");

//the file can be saved as .p7s or .p7m file
File.WriteAllBytes("d:\\test.txt.p7s", pkcs7Sign.SignFile("d:\\test.txt"));


PKCS7Verify pkcs7Verify = new PKCS7Verify("d:\\test.txt.p7s", "serial number");

Console.WriteLine("Number of signatures: " + pkcs7Verify.Signatures.Count);
Console.WriteLine("Original document name: " + pkcs7Verify.DocumentName);

//the signed document can contains multiple signatures
foreach (SignatureInfo si in pkcs7Verify.Signatures)
{
      Console.WriteLine("Signature is valid: " + si.SignatureIsValid);
      Console.WriteLine("Hash algorithm: " + si.HashAlgorithm.FriendlyName);
      Console.WriteLine("Signing Certificate: " +
si.SignatureCertificate.SubjectName.Name);
      Console.WriteLine("Signing Reason: " + si.SignatureReason);
      Console.WriteLine("Signing Date: " +
si.SignatureTime.ToLocalTime().ToString());
      Console.WriteLine("Is timestamped: " + si.Timestamping.IsTimeStamped);
      if (si.Timestamping.IsTimeStamped == true)
      {
            Console.WriteLine("Time stamp certificate: " +
si.Timestamping.Certificate.SubjectName.Name);
            Console.WriteLine("Time Stamp Signature date: " +
si.Timestamping.SignatureTime.ToLocalTime().ToString());
      }
}

//extract and save the unsigned document
File.WriteAllBytes("d:\\[unsigned]" + pkcs7Verify.DocumentName,
pkcs7Verify.UnsignedDocument);
```

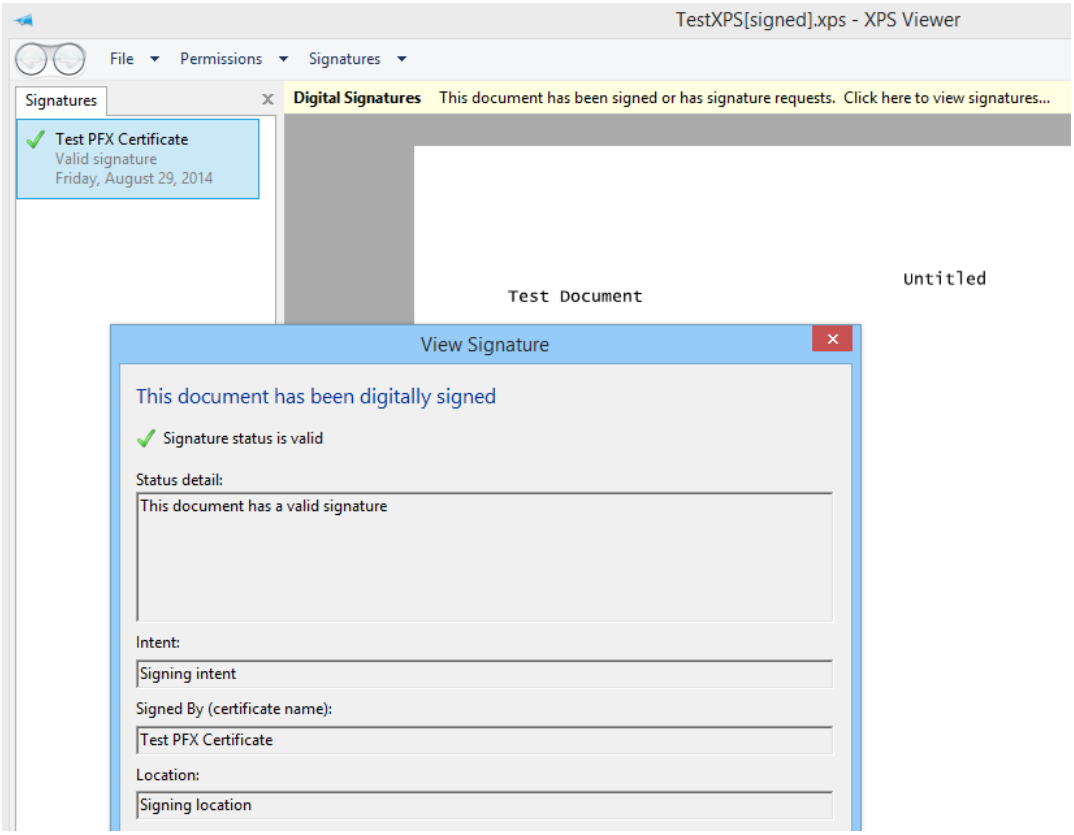
Digitally Sign an Office Document (.docx, .xlsx)

```
OfficeSign officeSign = new OfficeSign("serial number");
DigitalCertificates digitalCertificate = new DigitalCertificates();

officeSign.DigitalSignatureCertificate =
digitalCertificate.LoadCertificate("d:\\cert.pfx"), "123456");

//Load the certificate from Microsoft Store
//officeSign.DigitalSignatureCertificate =
digitalCertificate.LoadCertificate(false, "", "Digital certificates", "Select the
digital certificate", DigitalCertificateScope.ForDigitalSignature);

//sign DOCX document
officeSign.SignOfficeDocument("d:\\TestDOCX.docx", "d:\\TestDOCX[signed].docx");

//sign XLSX document
officeSign.SignOfficeDocument("d:\\TestXLSX.xlsx", "d:\\TestXLSX[signed].xlsx");

//verify signature
Console.WriteLine("Digital signature verification: " +
officeSign.VerifyDigitalSignature("d:\\TestDOCX[signed].docx", 1));
```


V
W
Excel document
with a digital signature
Signatures
Valid signatures:
Sign Files Test
4/27/2011
Signature Details
Valid signature - This signature and the signed content have not been modified since the signature was applied.
Signature type: XML-DSig
Signing as: Sign Files Test
Issued by: Secure Soft Private CA
View.

Digitally Sign an XPS Document

```
XPSSign xpsSign = new XPSSign("serial number");

DigitalCertificates digitalCertificate = new DigitalCertificates();

xpsSign.DigitalSignatureCertificate =
digitalCertificate.LoadCertificate("D:\\cert.pfx"), "123456");

//Load the certificate from Microsoft Store
//xpsSign.DigitalSignatureCertificate = digitalCertificate.LoadCertificate(false,
"", "Digital certificates", "Select the digital certificate",
DigitalCertificateScope.ForDigitalSignature);

xpsSign.AllowMultipleSignatures = false;
xpsSign.SigningIntent = "Signing intent";
xpsSign.SigningLocation = "Signing location";

//sign XPS document
xpsSign.SignXPSDocument("d:\\TestXPS.xps", "D:\\TestXPS[signed].xps");

//verify signature
Console.WriteLine("Digital signature verification: " +
xpsSign.VerifyDigitalSignature("d:\\TestXPS[signed].xps", 1));
```



Digitally Sign an XML Document (XMLDSig Format)

```
XMLSign xmlSign = new XMLSign(_SERIAL_NUMBER_);

DigitalCertificates digitalCertificate = new DigitalCertificates();

xmlSign.DigitalSignatureCertificate =
digitalCertificate.LoadCertificate("d:\\cert.pfx"), "123456");

//Load the certificate from Microsoft Store
//xmlSign.DigitalSignatureCertificate = digitalCertificate.LoadCertificate(false,
"", "Digital certificates", "Select the digital certificate",
DigitalCertificateScope.ForDigitalSignature);


//sign the XML document
xmlSign.SignXMLDocument("d:\\TestXML.xml", "d:\\TestXML[signed].xml");

//verify signature
Console.WriteLine("Digital signature is valid: " + xmlSign.VerifyDigitalSignature(
"d:\\TestXML[signed].xml", 1));
```

```
- <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  - <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    - <Reference URI="">
      - <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>tgHoW+7DwRNcYOCjugsJW4MFngc=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>MJInS69R5dSGAePSDS5banbEdfvW3U1fPU2424WLVWS9BQHCHRmMH1GM0hGUeeQlmg
  - <KeyInfo>
    - <KeyValue>
      - <RSAKeyValue>
          <Modulus>nvbwLDDphzQznQGPfh/kM/HUILza5VxHZKAb80zgaWNaj15+dDIIMFvMa8UULKKh1
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    - <X509Data>
        <X509Certificate>MIICjjCCAfegAwIBAgIRAKua7Iy+gKuw5efYjzXGq9UwDQYJKoZIhvcNAQEFBQAw
      </X509Data>
    </KeyInfo>
  </Signature>
</doc>
```

Creating Digital Certificates Using *X509CertificateGenerator* Class

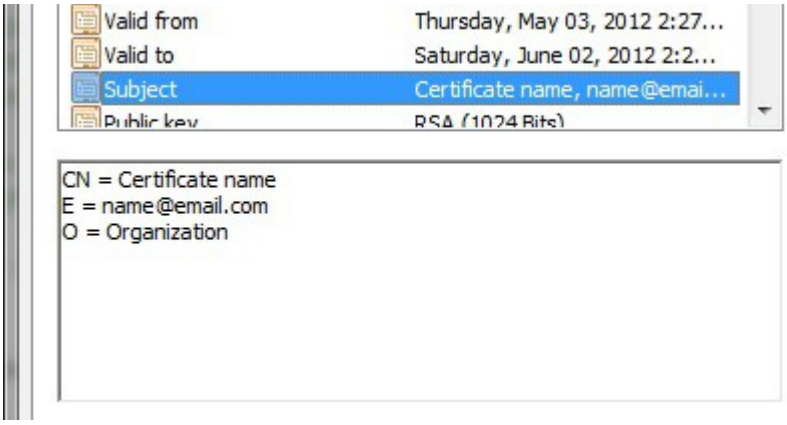
The main function of X509CertificateGenerator is to issue X.509 Version 3 digital certificates in PFX format. Using this library you can quickly issue all kind of certificates (user, self signed, root, time stamping, digital signature).

Certificate Subject

Every certificate must have a *Subject*. The *Subject* can contains Unicode characters like *ä,æ, £, Ñ*.

Set the *Subject* property:

```
X509CertificateGenerator cert = new X509CertificateGenerator();
//comma character is not permitted on the Subject name
cert.Subject = "CN=Certificate name,E=name@email.com,O=Organization";

//save the PFX certificate on a file
File.WriteAllBytes("c:\\cert.pfx", cert.GenerateCertificate("password", false));
```



Certificate Subject

Validity Period

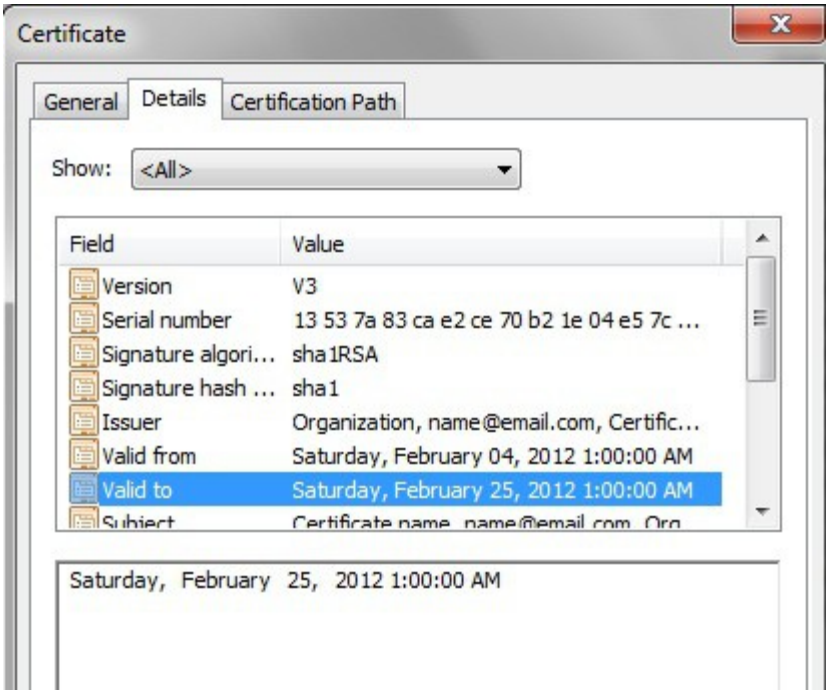
Every certificate has a validity period. A certificate becomes invalid after it expires. To set the validity period of the certificate use the following code:

```
X509CertificateGenerator cert = new X509CertificateGenerator();
//set the certificate Subject
cert.Subject = "CN=Certificate name,E=name@email.com,O=Organization";

//the certificate becomes valid after 4th February 2012
cert.ValidFrom = new DateTime(2012, 2, 4);

//the certificate will expires on 25th February 2012
cert.ValidTo = new DateTime(2012, 2, 25);

//save the PFX certificate on a file
File.WriteAllBytes("c:\\cert.pfx", cert.GenerateCertificate("password", false));
```

The default value of *ValidFrom* property is *DateTime.Now* (curent date).

The default value of *ValidTo* property is *DateTime.Now.AddYears(1).*

Observation: On the demo version of the library, the certificate validity cannot exceed 30 days (this is the single limitation of the library on the demo version).



Certificate validity period

Key Size and Signature Algorithm

The certificates issued by X509Dll use RSA algorithm (RSA is an algorithm for public-key cryptography that is based on the presumed difficulty of factoring large integers).

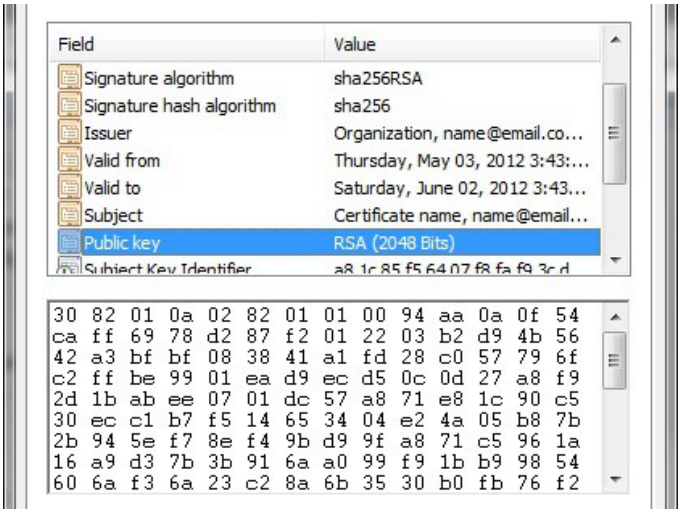
To set the key size and the signature algorithm of the certificate, use the following code:

```
X509CertificateGenerator cert = new X509CertificateGenerator();
//set the certificate Subject
cert.Subject = "CN=Certificate name,E=name@email.com,O=Organization";

//an RSA 2048 key will be used
cert.KeySize = KeySize.KeySize2048Bit;

//the certificate will use SHA256 hash algorithm
cert.SignatureAlgorithm = SignatureAlgorithm.SHA256WithRSA;

//save the PFX certificate on a file
File.WriteAllBytes("c:\\cert.pfx", cert.GenerateCertificate("password", false));
```

The default value of *KeySize* property is *KeySize.KeySize1024Bit* and should be enough for common certificates. For the Root certificates a 2048 key could be used.

The default value of *SignatureAlgorithm* property is *SignatureAlgorithm.SHA1WithRSA*.

Observation: The certificate will requires more time to be generated if a larger key size is used.



Certificate Key Size and Signature Algorithm

Serial Number

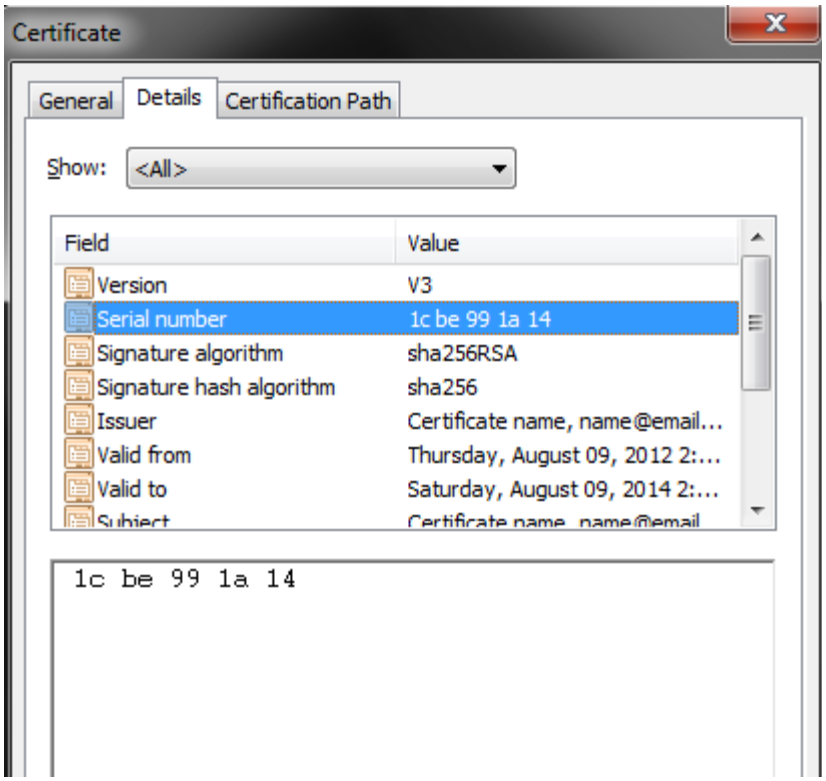
Every certificate must have a serial number. If the *SerialNumber* property is not set, a random value will be used.

To set the certificate serial number, use the code below:

```
 //set the certificate serial number
cert.SerialNumber = 123456789012;
```

The serial number can be lately used to identify a certificate but, according to X.509 standard, the certificate serial number appears on the digital certificate in hexadecimal notation. To set the serial number in hexadecimal format, use the code below:

```
//set the certificate serial number in hexadecimal format
cert.SerialNumber = long.Parse("1cbe991a14",
System.Globalization.NumberStyles.AllowHexSpecifier);
```



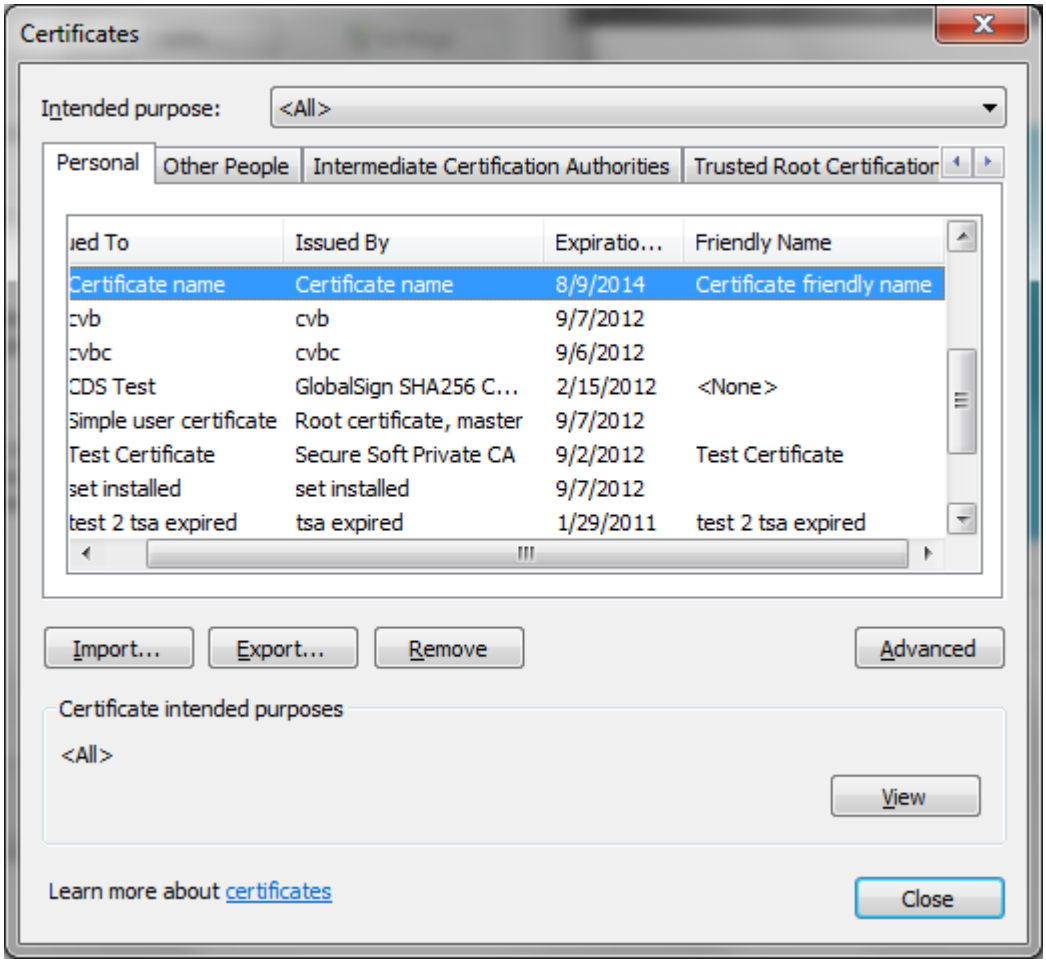
Certificate serial number

Friendly Name

When the certificate is imported to Microsoft Store, it will appear on the certificate list. If more certificates has the same subject, in order to identify a specific certificate, *FriendlyName* property can be set.

To set the certificate friendly name, use the code below:

```
cert.FriendlyName = "Certificate friendly name";
```



Certificate friendly name

Certificate Key Usage

Key Usage

A CA, user, computer, network device, or service can have more than one certificate. The Key Usage extension defines the security services for which a certificate can be used. The options can be used in any combination and can include the following:

DataEncipherment - The public key can be used to directly encrypt data, rather than exchanging a symmetric key for data encryption.

DigitalSignature - The certificate use the public key for verifying digital signatures that have purposes other than non-repudiation, certificate signature, and CRL signature.

KeyEncipherment - The certificate use the public key for key transport.

NonRepudiation - The certificate use the public key for verifying a signature on CRLs.

CRLSigning - The certificate use the public key for verifying a signature on certificates.

CertificateSigning - The certificate use the public key for key agreement.

KeyAgreement - The certificate public key may be used only for enciphering data while performing key agreement.

EncipherOnly - The certificate public key may be used only for enciphering data while performing key agreement.

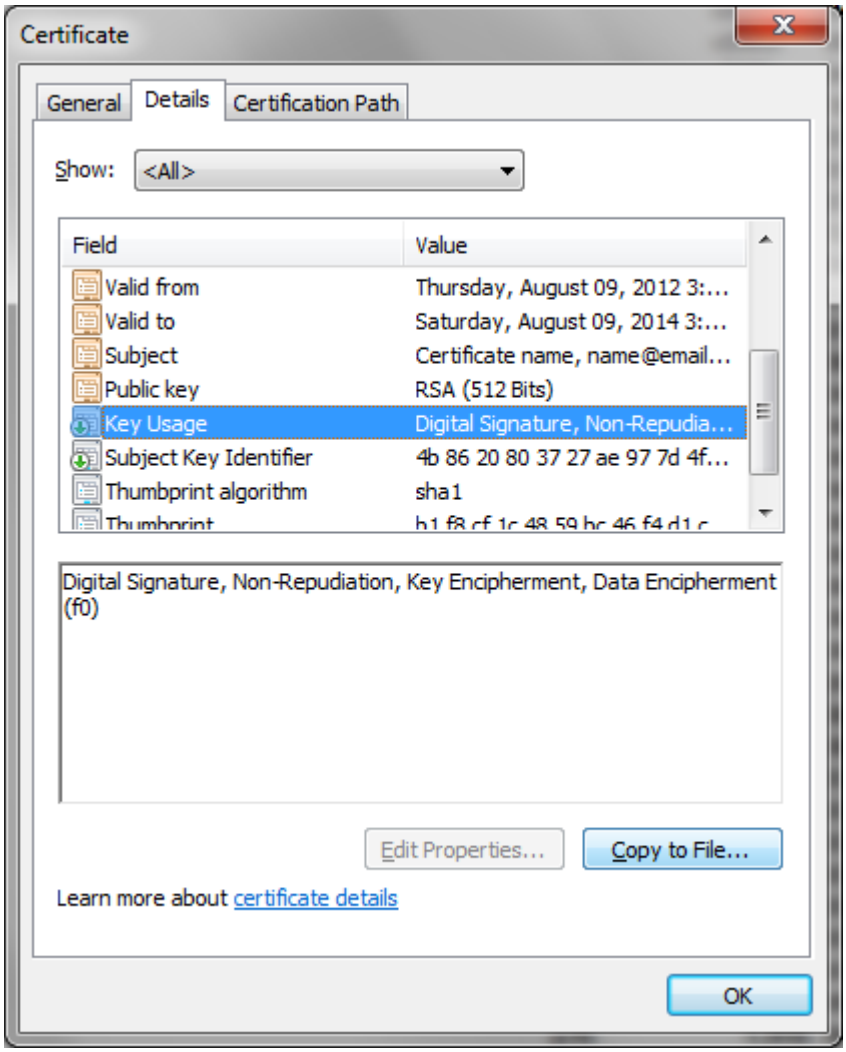
DecipherOnly - The certificate public key may be used only for enciphering data while performing key agreement.

For a simple certificate, the most used Key Usages are: *DigitalSignature, NonRepudiation, KeyEncipherment* and *DataEncipherment.*

For a Root Certificate (CA certificate), the most used Key Usages are: *CertificateSigning* and *CRLSigning.*

To add Key Usage to a digital certificate, use the following code:

```
cert.Extensions.AddKeyUsage(CertificateKeyUsage.DigitalSignature);
cert.Extensions.AddKeyUsage(CertificateKeyUsage.NonRepudiation);
cert.Extensions.AddKeyUsage(CertificateKeyUsage.KeyEncipherment);
cert.Extensions.AddKeyUsage(CertificateKeyUsage.DataEncipherment);
```



Certificate Key Usage

Enhanced Key Usage

This extension indicates how a certificate's public key can be used. The Enhanced Key Usage extension provides additional information beyond the general purposes defined in the Key Usage extension. For example, OIDs exist for Client Authentication (1.3.6.1.5.5.7.3.2), Server Authentication (1.3.6.1.5.5.7.3.1), and Secure E-mail (1.3.6.1.5.5.7.3.4).

When a certificate is presented to an application, an application can require the presence of an Enhanced Key Usage OID specific to that application.

The library supports a lot of well known Enhanced Key Usages but also support to specify a custom Enhanced Key Usage extension.

Some of Enhanced Key Usages available by default on the library are:

CodeSigning - The certificate can be used for signing code.

SmartcardLogon - The certificate enables an individual to log on to a computer by using a smart card.

DocumentSigning - The certificate can be used for signing documents.

TimeStamping - The certificate can be used for signing public key infrastructure timestamps according to RFC 3161.

To add *Enhanced Key Usage* to a digital certificate, use the following code:

```
cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.SmartcardLogon);
cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.TimeStamping);
cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.SecureEmail);
```

To add a custom *Enhanced Key Usage* extension, see below:

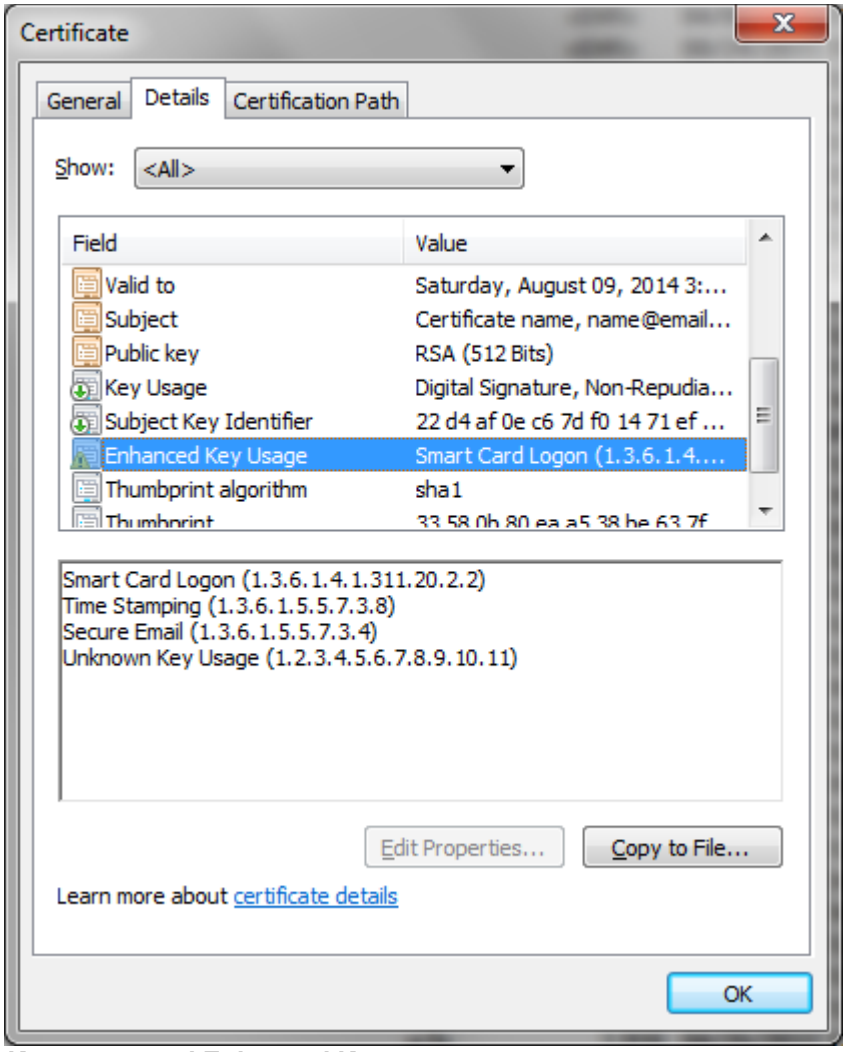
```
cert.Extensions.AddEnhancedKeyUsage(new
System.Security.Cryptography.Oid("1.2.3.4.5.6.7.8.9.10.11"));
```

Critical Key Usage

In some scenarios, *Key Usage* or *Enhanced Key Usage* must be set as *Critical extension*.

By default, these properties are considered non-critical but the behavior can be changed as below:

```
cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.TimeStamping);
cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.SecureEmail);
cert.Extensions.AddEnhancedKeyUsage(new
System.Security.Cryptography.Oid("1.2.3.4.5.6.7.8.9.10.11"));

//set Enhanced Key Usage as critical
cert.Extensions.EnhancedKeyUsageIsCritical = true;
cert.Extensions.KeyUsageIsCritical = false;
```



Key usage and Enhanced Key usage

Issuing Digital Certificates

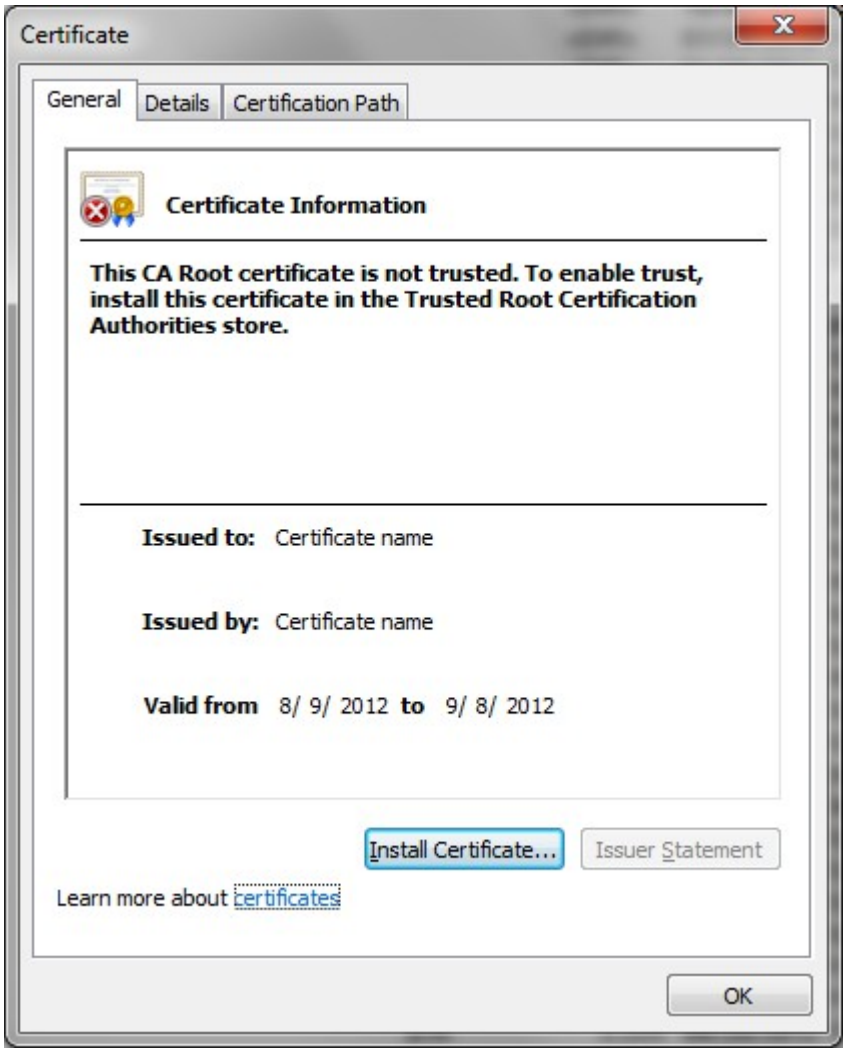
Issue a Self-signed Digital Certificate

A self-signed certificate is not issued by a Root CA so it cannot be verified as “trusted”.

To issue a self signed certificate, use the following code:

```
X509CertificateGenerator cert = new X509CertificateGenerator();

//set the validity of the certificate
cert.ValidFrom = DateTime.Now;
cert.ValidTo = DateTime.Now.AddYears(2);

//set the signing algorithm and the key size
cert.KeySize = KeySize.KeySize1024Bit;
cert.SignatureAlgorithm = SignatureAlgorithm.SHA256WithRSA;

//set the certificate subject
cert.Subject = "CN=Certificate name,E=name@email.com,O=Organization";

cert.Extensions.AddKeyUsage(CertificateKeyUsage.DigitalSignature);
cert.Extensions.AddKeyUsage(CertificateKeyUsage.NonRepudiation);

cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.DocumentSigning);
cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.SecureEmail);

//set Enhanced Key Usage as critical
cert.Extensions.EnhancedKeyUsageIsCritical = true;

//create the PFX certificate
File.WriteAllBytes("C:\\cert.pfx", cert.GenerateCertificate("P@ssword"));

//optionally, save the public part to see the certificate
File.WriteAllBytes("c:\\user.cer", new
System.Security.Cryptography.X509Certificates.X509Certificate2("c:\\cert.pfx",
"P@ssword").RawData);
```

Because the certificate is a self-signed certificate, when it is opened (e.g. *c:\user.cer*) or the PFX file is imported on Microsoft Store, it will appear as “untrusted”.



A self-signed certificate

Issue a Root Certificate

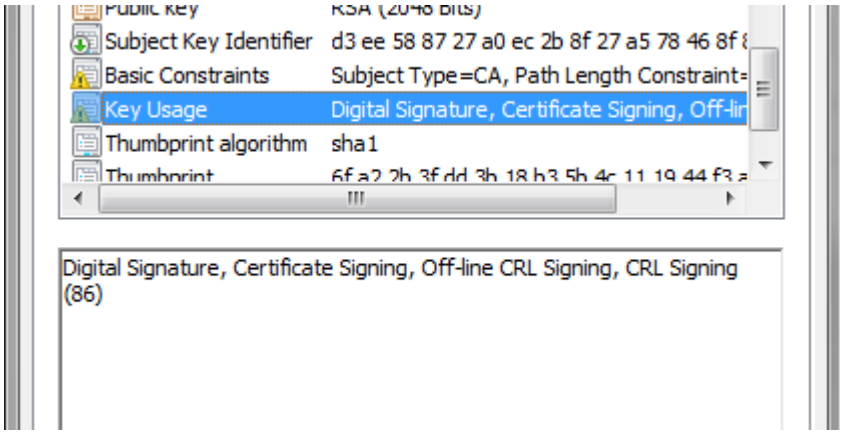
A Root Certificate (CA certificate) is a special type of certificate that can be used to digitally sign other certificates. Also, a Root Certificate can also sign other Root Certificates.

To issue a Root Certificate, use the code below:

```
//on the demo version the certificates will be valid 30 days only
//this is the single restriction of the library in demo mode
X509CertificateGenerator cert = new X509CertificateGenerator();

//set the validity of the Root certificate
cert.ValidFrom = DateTime.Now;
cert.ValidTo = DateTime.Now.AddYears(5);

//set the signing algorithm and key size
cert.KeySize = KeySize.KeySize2048Bit;
cert.SignatureAlgorithm = SignatureAlgorithm.SHA512WithRSA;

cert.Subject = "CN=Root Certificate,E=root@email.com,O=Organization Root";

//add some extensions to the certificate marked as critical
cert.Extensions.AddKeyUsage(CertificateKeyUsage.DigitalSignature);
cert.Extensions.KeyUsageIsCritical = true;

bool isRootCertificate = true;
File.WriteAllBytes("C:\\root.pfx",
cert.GenerateCertificate("Root_password", isRootCertificate));
```

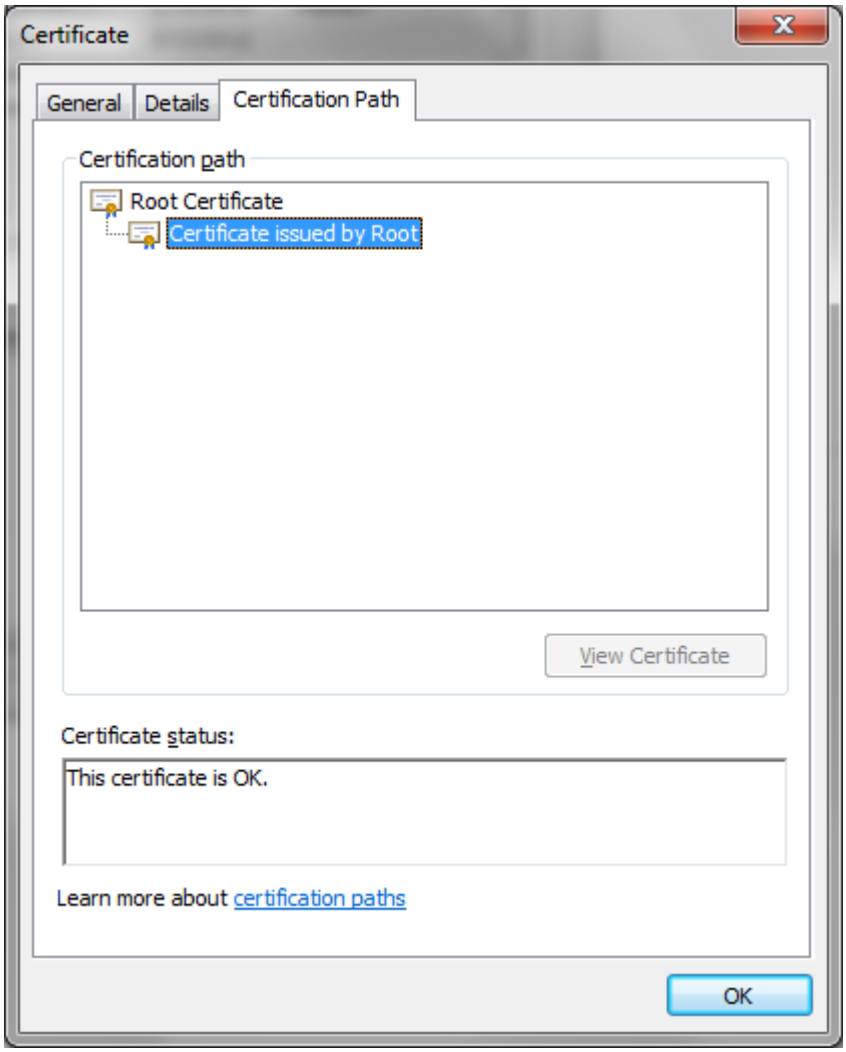
Note that creating a Root certificate is very similar with creating a self signed certificate. The only main difference is on the second parameter of *GenerateCertificate()* method that must be set to *true*.

Also, some Key Usage extension is automatically added for a Root Certificate as below:



Key usage for a Root Certificate

The Root Certificate is used for issue other certificates. When a Root Certificate issues a client certificate and this certificate is imported on Microsoft (including the Root Certificate), the entire hierarchy will look like this:



Root certificate issued other certificates

Issue a Digital Certificate Signed by a Root Certificate

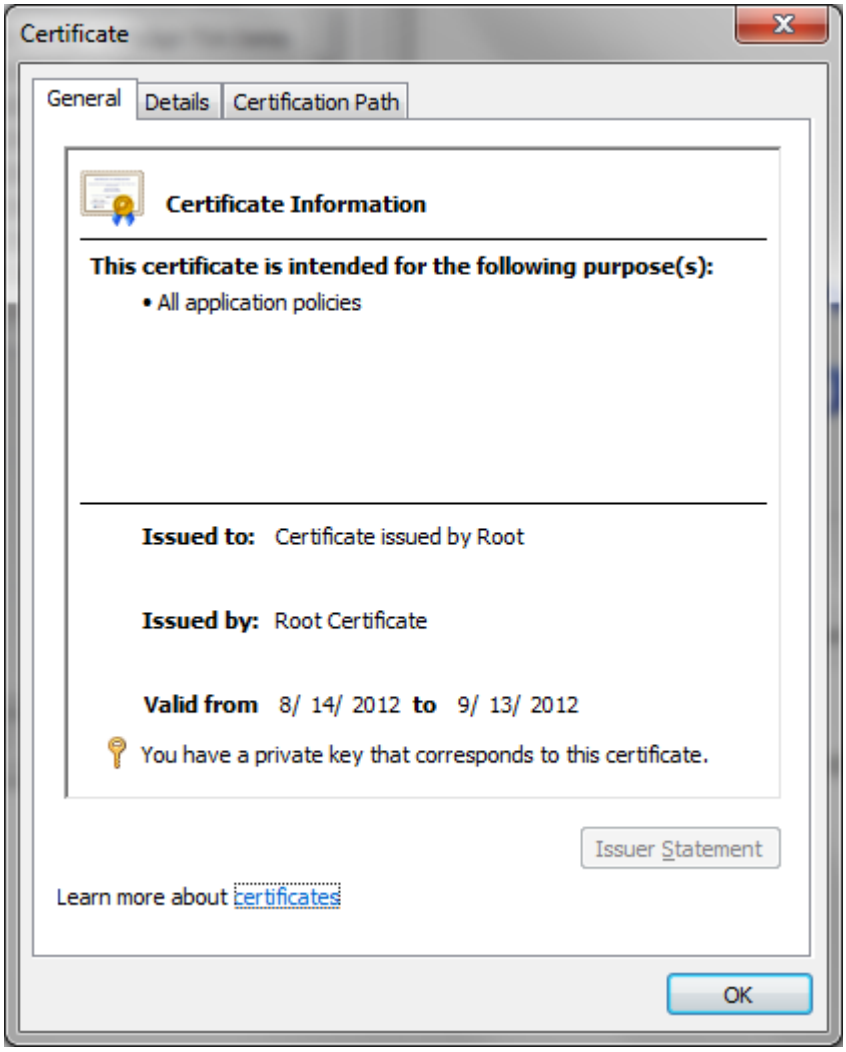
In some cases, is necessary to issue certificates for an entire organization. On this scenario you have two options:

- Issue a self signed certificates for every entity (see section *Creating a self-signed digital certificate*).
- Issue a Root Certificate and every certificate issued for an entity to be issued (signed) by this Root Certificate.

To issue a digital certificate signed by a Root Certificate, use the code below:

```
//Issue the Root certificate first

X509CertificateGenerator root = new X509CertificateGenerator();

//set the validity of the Root certificate
root.ValidFrom = DateTime.Now;
root.ValidTo = DateTime.Now.AddYears(5);

//set the signing algorithm and key size
root.KeySize = KeySize.KeySize2048Bit;
root.SignatureAlgorithm = SignatureAlgorithm.SHA512WithRSA;

root.Subject = "CN=Root Certificate,E=root@email.com,O=Organization Root";

bool isRootCertificate = true;
File.WriteAllBytes("C:\\root.pfx",
root.GenerateCertificate("Root_password", isRootCertificate));

//Issue the User Certificate
X509CertificateGenerator cert = new X509CertificateGenerator();

//load the root certificate to sign the intermediate certificate
cert.LoadRootCertificate(File.ReadAllBytes("c:\\root.pfx"), "Root_password");

cert.Subject = "CN=Certificate issued by Root,E=name@email.com,O=Organization";

//set the validity of the certificate
cert.ValidFrom = DateTime.Now;
cert.ValidTo = DateTime.Now.AddYears(1);

//set the signing algorithm and key size
cert.KeySize = KeySize.KeySize1024Bit;
cert.SignatureAlgorithm = SignatureAlgorithm.SHA1WithRSA;

File.WriteAllBytes("c:\\user.pfx", cert.GenerateCertificate("123456"));
```

After the client certificate is imported on Microsoft Store, the user certificate will look like this:



An User Certificate issued a Root Certificate

Importing Digital Certificates

Digital Certificates and Microsoft Store

Usually, the digital certificates are stored in two places:

- in Microsoft Store
- in PFX on P12 files

A PFX file can be imported on Microsoft Store as on the next section.

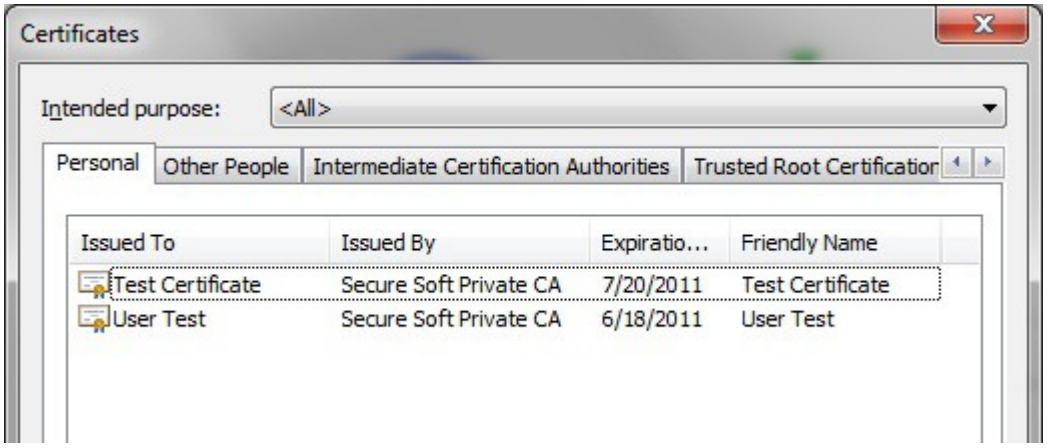
The certificates stored on **Microsoft Store** are available by opening *Internet Explorer* – *Tools* menu – *Internet Options* – *Content* tab – *Certificates* button (see below) or by entering ***certmgr.msc*** command on Run window.

For digital signatures, the certificates stored on *Personal* tab are used. These certificates have a public and a private key.

The Root Certificates are stored on *Trusted Root Certification Authorities* tab.

The digital signature is created by using the private key of the certificate. The private key can be stored on the file system (imported PFX files), on an cryptographic smart card (like Aladdin eToken or SafeNet iKey) or on a HSM (Hardware Security Module).

For encryption, only the public key of the certificate is necessary (certificates stored on *Personal* or *Other People* tabs).



Signing certificates available on Microsoft Store

Another way to store a digital certificate is a **PFX (or P12) file**. This file contain the public and the private key of the certificate. This file is protected by a password in order to keep safe the key pair.

Importing PFX Certificates on Microsoft Store

The PFX file can be imported on Microsoft Store (just open the PFX file and follow the wizard).

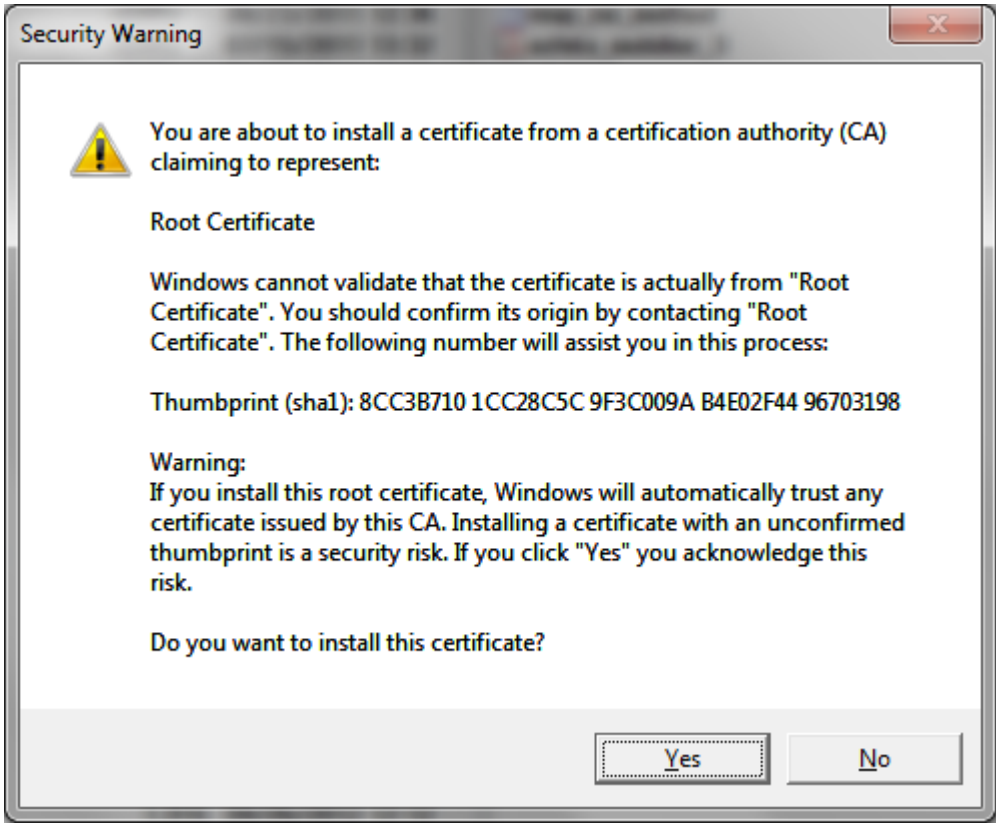
In order to install the certificate, follow this steps:

- double click on the PFX file (e.g. *c:\cert.pfx*)
- click Next
- click Next again (or browse for other PFX file)
- enter the PFX certificate password (e.g. *P@ssword*)
- click Next, Next
- click Finish.

Trusting Certificates

When a user certificate is issued by a Root Certificate, in order to trust the user certificate, the Root Certificate must be imported on *Microsoft Store – Trusted Root Certification Authorities.*

When the PFX user certificate is imported on Microsoft Store, the Root Certificate can be also imported as follow:



Importing the Root Certificate on Microsoft Store

At this step, the Root Certificate is imported and every certificate issued by this Root is

considered trusted.

Anyway, if a document or email message is digitally signed by the client certificate and the document/email is opened on other computer, the **digital signature might be considered untrusted** because the Root certificate is not imported on that computer so **the Root Certificate must be manually imported on every client machine that will be related with this certificate.**

Because the Root Certificate is not included by default in *Microsoft Store – Trusted Root Certification Authorities*, the Root Certificate that issues the User Certificate must be imported on that store when the PFX certificate is imported.

See more details at this link: Validating Digital Certificates in Windows

More advanced options to manually install certificates on the client machines are available by using Certmgr.exe (Certificate Manager Tool).

Other useful links:

- Adding digital signature and encryption in Outlook emails
- Adding digital signature on Mozilla Thunderbird emails
- Validating digital signatures in Adobe

Importing Certificates From Code

In order to add the Root Certificate on Microsoft Store, use the following code:

```
using System.Security.Cryptography.X509Certificates;

//open the Microsoft Root Store
var store = new X509Store(StoreName.Root, StoreLocation.CurrentUser);
store.Open(OpenFlags.ReadWrite);

try
{
      var cert = new X509Certificate2(File.ReadAllBytes("c:\\root.cer"));
     //use dirrectly the PFX
     //var cert = new X509Certificate2("c:\\root.pfx", "Root_password");

      store.Add(cert);
}
finally
{
      store.Close();
}
```

Issue Digital Signature Certificates

Digital certificates can be used for digitally sign PDF, Office, XPS documents or email messages.

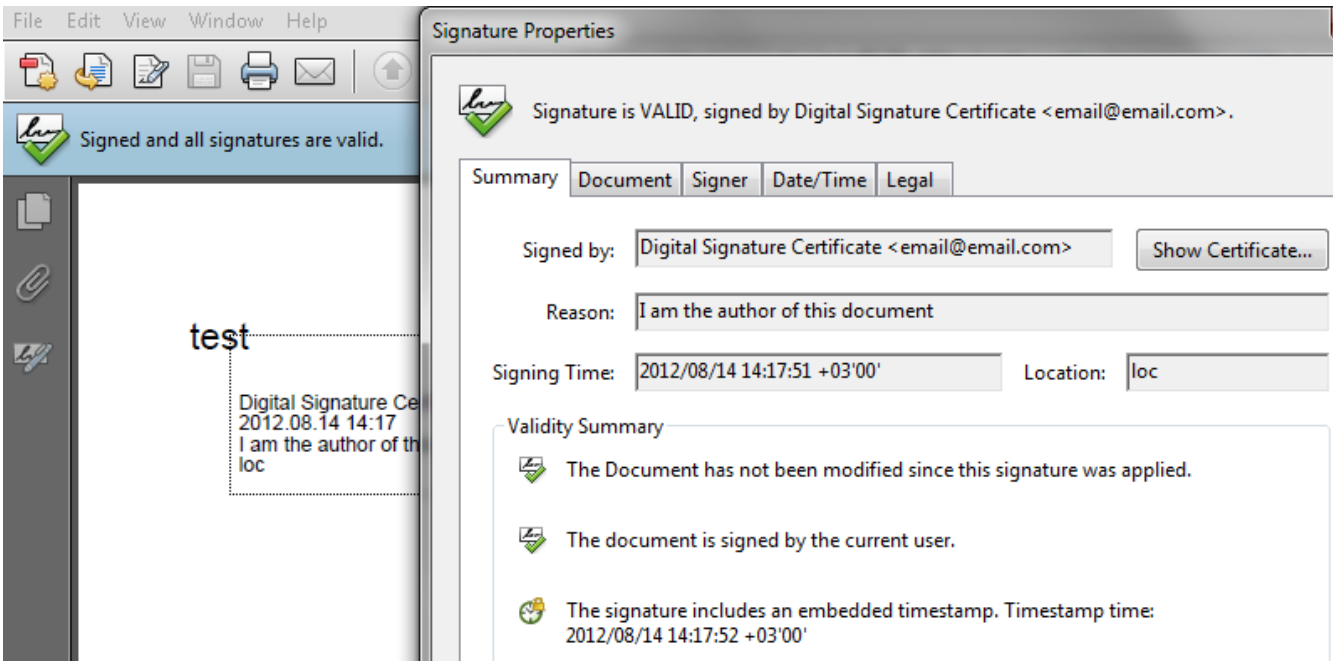
The time digital signature certificate profile will look like this:
- It is recommended to be issued by a Root Certificate (not self signed certificate).
- Use RSA1024 key size (or RSA 2048 for more security).
- Key Usage: Digital Signature.
- Extended Key Usage - add ONLY Time Stamping extension (OID: 1.3.6.1.5.5.7.3.8) marked as critical.
- Expiration date: 1 year or more.

In order to create a certificate for digital signature, use the code below:

```
//Issue the Root Certificate
//on the demo version the certificates will be valid 30 days only
//this is the single restriction of the library in demo mode
X509CertificateGenerator root = new X509CertificateGenerator();

//set the validity of the Root certificate
root.ValidFrom = DateTime.Now;
root.ValidTo = DateTime.Now.AddYears(10);

//set the signing algorithm and key size
root.KeySize = KeySize.KeySize2048Bit;
root.SignatureAlgorithm = SignatureAlgorithm.SHA512WithRSA;

root.Subject = "CN=Root Certificate,E=root@email.com,O=Organization Root";

File.WriteAllBytes("C:\\root.pfx", root.GenerateCertificate("Root_password",
true));

//Issue the digital signature certificate
X509CertificateGenerator cert = new X509CertificateGenerator();

//load the root certificate to sign the intermediate certificate
cert.LoadRootCertificate(File.ReadAllBytes("c:\\root.pfx"), "Root_password");

cert.Subject = "CN=Digital Signature Certificate,E=email@email.com,
O=Organization";

//set the validity of the certificate
cert.ValidFrom = DateTime.Now;
cert.ValidTo = DateTime.Now.AddYears(1);

//set the signing algorithm and key size
cert.KeySize = KeySize.KeySize2048Bit;
cert.SignatureAlgorithm = SignatureAlgorithm.SHA1WithRSA;
//add the certificate key usage
cert.Extensions.AddKeyUsage(CertificateKeyUsage.DigitalSignature);
cert.Extensions.AddKeyUsage(CertificateKeyUsage.NonRepudiation);
```

```
//for encryption - optionally
cert.Extensions.AddKeyUsage(CertificateKeyUsage.DataEncipherment);

//add the certificate enhanced key usage
cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.DocumentSigning);
cert.Extensions.AddEnhancedKeyUsage(CertificateEnhancedKeyUsage.SecureEmail);
cert.Extensions.EnhancedKeyUsageIsCritical = true;

File.WriteAllBytes("c:\\userCertificate.pfx",
cert.GenerateCertificate("user_password"));
```

After the certificate is created and imported, it can be used for digital signature.



Adding a digital signature on a PDF document

License Terms

PDFSignDll License Agreement (EULA)

Important! Read the following terms carefully before installing, copying and/or using the product. Installing, copying or using the product indicates your acceptance of these terms, as well the terms in the contract between Client and Secure Soft.

This End-User License Agreement ("EULA") is a legal agreement between Client and Secure Soft governing the use of the software (SOFTWARE) accompanying this EULA, including any and all associated media, printed materials, and "online" or electronic documentation protected by copyright laws. By installing, copying, or otherwise using the SOFTWARE, you agree to be bound by the terms of this EULA. If you do not agree with the terms of this EULA, do not use the SOFTWARE.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall Secure Soft or its suppliers be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use the SOFTWARE, even if Secure Soft has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability, for consequential or incidental damages, the above limitation may not apply to you. Liability of the Vendor will be limited to a maximum of the original purchase price of the Software. The Vendor will not be liable for any general, special, incidental or consequential damages including, but not limited to, loss of production, loss of profits, loss of revenue, loss of data, or any other business or economic disadvantage suffered by the Licensee arising out of the use or failure to use the Software. The Vendor makes no warranty expressed or implied regarding the fitness of the Software for a particular purpose or that the Software will be suitable or appropriate for the specific requirements of the Licensee. The Vendor does not warrant that use of the Software will be uninterrupted or error-free. The Licensee accepts that software in general is prone to bugs and flaws within an acceptable level as determined in the industry.

TERMINATION. Without prejudice to any other rights, Secure Soft may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such an event, you must destroy all copies of the SOFTWARE, all of its component parts and uninstall the SOFTWARE.

WARRANTIES. The SOFTWARE is supplied "as is", Secure Soft does not guarantee that the SOFTWARE will carry no errors, nor will it take on any liability for damages more than written here. Secure Soft does not warrant that the SOFTWARE will meet further requirements. However, Secure Soft states that every reasonable effort has been made to avoid presence of any possible malevolent code in the SOFTWARE (including, but not limited to, viruses, trojans, root kits and/or spyware) and declares that to the best knowledge of competent experts of Secure Soft the SOFTWARE does not contain any such malware. Supplemental enhances and updates are subject to other contracts.

NO OTHER WARRANTIES. Secure Soft disclaims all other warranties, either express or

implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the SOFTWARE and the accompanying written materials. This limited warranty gives you specific legal rights. You may have other which vary from state to state.

MISCELLANEOUS. This EULA comes into force when you accept all the conditions stated herein.

CONTROLLING LAW AND SEVERABILITY. This License shall be governed by the laws of the Romania. If for any reason a court of competent jurisdiction finds any provision, or portion thereof, to be unenforceable, the remainder of this License shall continue in full force and effect.

OTHER LICENSES

PDFSignDll library is compiled using Microsoft .NET Framework 2.0 and it is based on the MPL / LGPLv2 version of the iTextSharp 4.1.6 library. The original library and all changes are available here: http://www.signfiles.com/license/PDFSignDllSRC.zip

ITextSharp is subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/MPL/

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is 'iText, a free JAVA-PDF library'. The Initial Developer of the Original Code is Bruno Lowagie. Portions created by the Initial Developer are Copyright (C) 1999, 2000, 2001, 2002 by Bruno Lowagie.
All Rights Reserved.

Co-Developer of the code is Paulo Soares. Portions created by the Co-Developer are Copyright (C) 2000, 2001, 2002 by Paulo Soares. All Rights Reserved.

Contributor(s): all the names of the contributors are added in the source code where applicable.

Alternatively, the contents of this file may be used under the terms of the LGPL license (the "GNU LIBRARY GENERAL PUBLIC LICENSE"), in which case the provisions of LGPL are applicable instead of those above. If you wish to allow use of your version of this file only under the terms of the LGPL License and not to allow others to use your version of this file under the MPL, indicate your decision by deleting the provisions above and replace them with the notice and other provisions required by the LGPL. If you do not delete the provisions above, a recipient may use your version of this file under either the MPL or the GNU LIBRARY GENERAL PUBLIC LICENSE.

This library is free software; you can redistribute it and/or modify it under the terms of the MPL as stated above or under the terms of the GNU Library General Public License as published by the Free Software Foundation either version 2 of the License, or any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library general Public License for more details.

Copyright (c) 2000 - 2011 The Legion of the Bouncy Castle Inc. (http://www.bouncycastle.org)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

FileSignDll License Agreement (EULA)

Important! Read the following terms carefully before installing, copying and/or using the product. Installing, copying or using the product indicates your acceptance of these terms, as well the terms in the contract between Client and Secure Soft.

This End-User License Agreement ("EULA") is a legal agreement between Client and Secure Soft governing the use of the software (SOFTWARE) accompanying this EULA, including any and all associated media, printed materials, and "online" or electronic documentation protected by copyright laws. By installing, copying, or otherwise using the SOFTWARE, you agree to be bound by the terms of this EULA. If you do not agree with the terms of this EULA, do not use the SOFTWARE.

COPYRIGHT. The SOFTWARE is the proprietary property of or under license to Secure Soft SRL and is protected by Romanian copyright laws and international treaty provisions. Therefore, you must treat the software like any other copyrighted material.

RESTRICTIONS. You may not reverse engineer, decompile or disassemble the SOFTWARE. Your rights under this License will terminate automatically without notice from Secure Soft if you fail to comply with any term(s) of the License.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall Secure Soft or its suppliers be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use the SOFTWARE, even if Secure Soft has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability, for consequential or incidental damages, the above limitation may not apply to you. Liability of the Vendor will be limited to a maximum of the original purchase price of the Software. The Vendor will not be liable for any general, special, incidental or consequential damages including, but not limited to, loss of production, loss of profits, loss of revenue, loss of data, or any other business or economic disadvantage suffered by the Licensee arising out of the use or failure to use the Software. The Vendor makes no warranty expressed or implied regarding the fitness of the Software for a particular purpose or that the Software will be suitable or appropriate for the specific requirements of the Licensee. The Vendor does not warrant that use of the Software will be uninterrupted or error-free. The Licensee accepts that software in general is prone to bugs and flaws within an acceptable level as determined in the industry.

TERMINATION. Without prejudice to any other rights, Secure Soft may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such an event, you must destroy all copies of the SOFTWARE, all of its component parts and uninstall the SOFTWARE.

WARRANTIES. The SOFTWARE is supplied "as is", Secure Soft does not guarantee that the SOFTWARE will carry no errors, nor will it take on any liability for damages more than written here. Secure Soft does not warrant that the SOFTWARE will meet further requirements.

However, Secure Soft states that every reasonable effort has been made to avoid presence of any possible malevolent code in the SOFTWARE (including, but not limited to, viruses, trojans, root kits and/or spyware) and declares that to the best knowledge of competent experts of Secure Soft the SOFTWARE does not contain any such malware. Supplemental enhances and updates are subject to other contracts.

NO OTHER WARRANTIES. Secure Soft disclaims all other warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the SOFTWARE and the accompanying written materials. This limited warranty gives you specific legal rights. You may have other which vary from state to state.

MISCELLANEOUS. This EULA comes into force when you accept all the conditions stated herein.

LICENSE GRANT AND RESTRICTIONS. Secure Soft grants you a non-exclusive right to use the SOFTWARE under the terms of this EULA.

CONTROLLING LAW AND SEVERABILITY. This License shall be governed by the laws of the Romania. If for any reason a court of competent jurisdiction finds any provision, or portion thereof, to be unenforceable, the remainder of this License shall continue in full force and effect.

COPYRIGHT NOTES

Copyright (c) 2000 - 2011 The Legion Of The Bouncy Castle (http://www.bouncycastle.org)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

All other trademarks are property of their respective owners.