.NET Digital Signature Library
User Manual

Introduction

The main function of the .NET Digital Signature Library is to digitally sign PDF, .P7M, .P7S, and XML files using X.509 certificates stored in PFX files, on smart cards, USB tokens, or HSMs (Hardware Security Modules) available through the Microsoft Certificate Store or accessible via PKCS#11.

The .NET Digital Signature Library can also be used to create X.509 certificates in PFX format. The library allows you to generate PFX digital certificates as well as custom certificates with different Key Usage and Enhanced Key Usage extensions (Self-Signed, Root, Intermediate, Timestamping certificates).

General Library Features: Compiled for .NET Framework (3.5, 4.6.2, 4.8), .NET Standard 2.0, NET 8, NET 9, and NET 10. Support for Cloud/Remote/External signature and PKCS#11 to bypass smart card/HSM PIN protection, PFX/P12 digital certificate generator.

PDF Signature Features: configurable signature appearance (custom page, all pages, signature text, position, signature image), Long Term Validation (LTV), PAdES-LT, Long-Term Archival PAdES-LTA, Timestamping, Encryption, Remote/Cloud/External signatures, certified signatures, SHA-256, SHA-384, SHA-512, Elliptic Curves, DSS eIDAS Validation, Qualified Digital Certificates and Qualified Seals, PKCS#11 support to bypass smart card/HSM PIN protection.

CAdES Signature Features: CAdES-B (Baseline), C, XL, LT, and A profiles, timestamping, remote/cloud/external signatures, SHA-256, SHA-384, SHA-512, Elliptic Curves, DSS eIDAS validation, Qualified Digital Certificates and Seals, PKCS#11 support to bypass smart card/HSM PIN protection.

XML Features: XAdES Baseline profile, XMLDsig, SHA256, DSS eIDAS validation, Qualified Digital Certificates and Seals, PKCS#11 support to bypass smart card/HSM PIN protection.

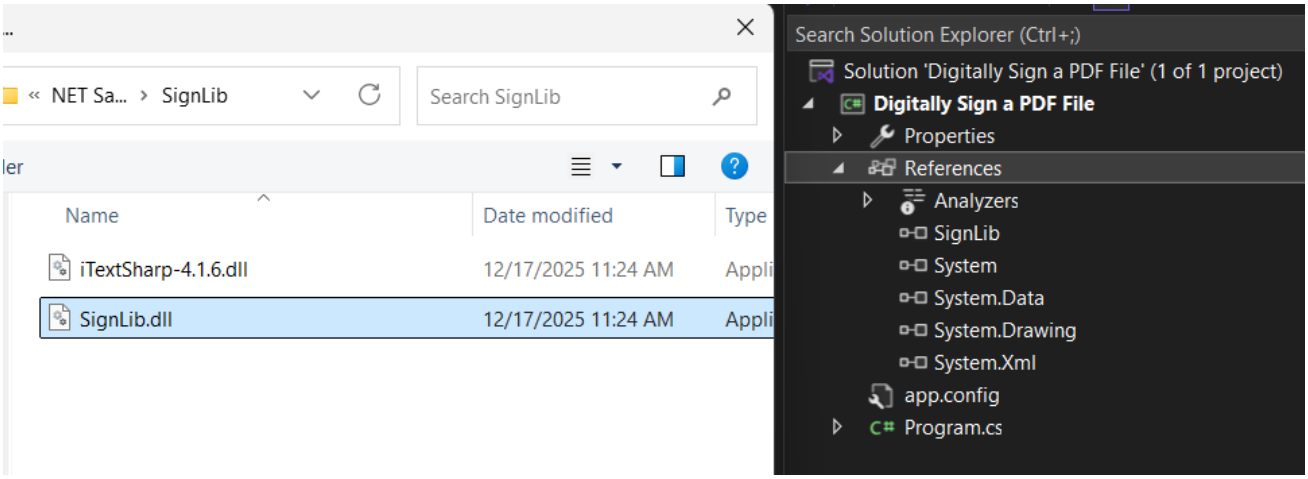
Links

.NET Digital Signature Library: https://www.signfiles.com/sdk/SignatureLibrary.zip
.NET Digital Signature Library main page: https://www.signfiles.com/signature-library/

This user manual: https://www.signfiles.com/manuals/SignatureLibraryUserManual.pdf

How to use .NET Digital Signature Library in Visual Studio

- Download the library from here: https://www.signfiles.com/sdk/SignatureLibrary.zip
- Unzip the file and copy the *SignLib.dll, iTextSharp-4.1.6.dll* and *SignLib.xml* on your project location.
- In your project, go to *References,* select *Add Reference..,*select the *SignLib.dll* as below.



Adding as reference SignLib library

Note:
SignLib.dll is compiled for .NET Framework (3.5, 4.6.2, 4.8), .NET Standard 2.0, and NET 8, NET 9 and NET 10.

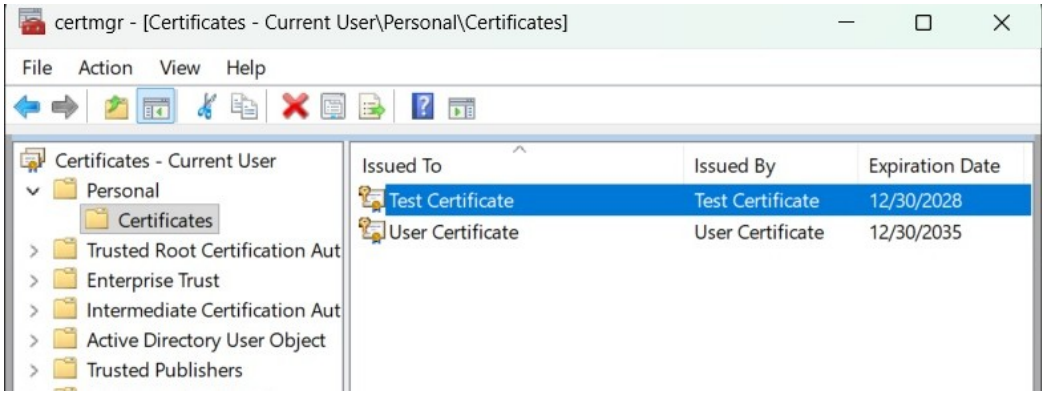
Digital Certificates

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 (smart card certificates and USB tokens certificates are stored here)
- in PFX on P12 files

The certificates stored on **Microsoft Store** can be accessed using the command: *Start – Run – certmgr.msc* (for Current User) *or certlm.msc* (for Local Machine).



Signing certificates available on Microsoft Store

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 a cryptographic smart card (like eToken or SafeNet iKey) or on a HSM (Hardware Security Module) like Luna HSM.

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.

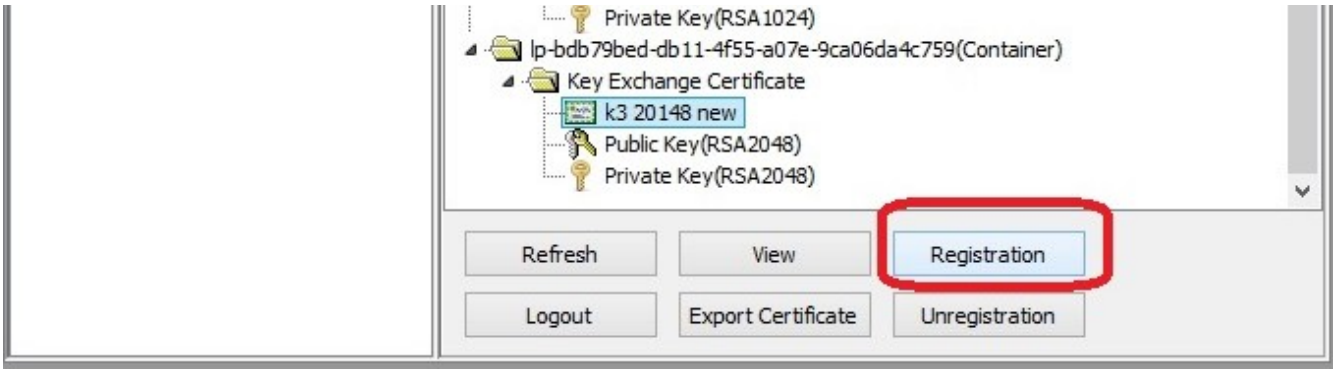
Note that a PFX/P12 file can be imported on Microsoft Store (just open the PFX/P12 file and follow the wizard).

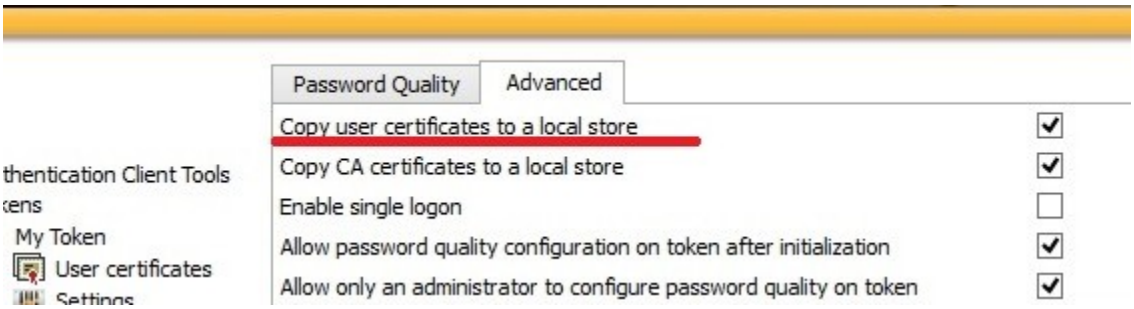
Certificates Stored on Smart Cards or USB Tokens

If your certificate is stored on a smart card or USB token (like SafeNet eToken), the certificate must appear on Microsoft Certificate Store in order to be used by the library.

If the certificate not appears on Microsoft Store, you must ask your vendor about how to import the certificate on the MS Store. Usually, the smart card driver or the middleware automatically install the certificate on Microsoft Certificate Store.

You should also look at the middleware options, like below:





Create a Digital Certificate Using *X509CertificateGenerator* Class

Every certificate must have a *Subject*. The *Subject* can contains Unicode characters like *ă,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).

```
using SignLib.Certificates;

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.

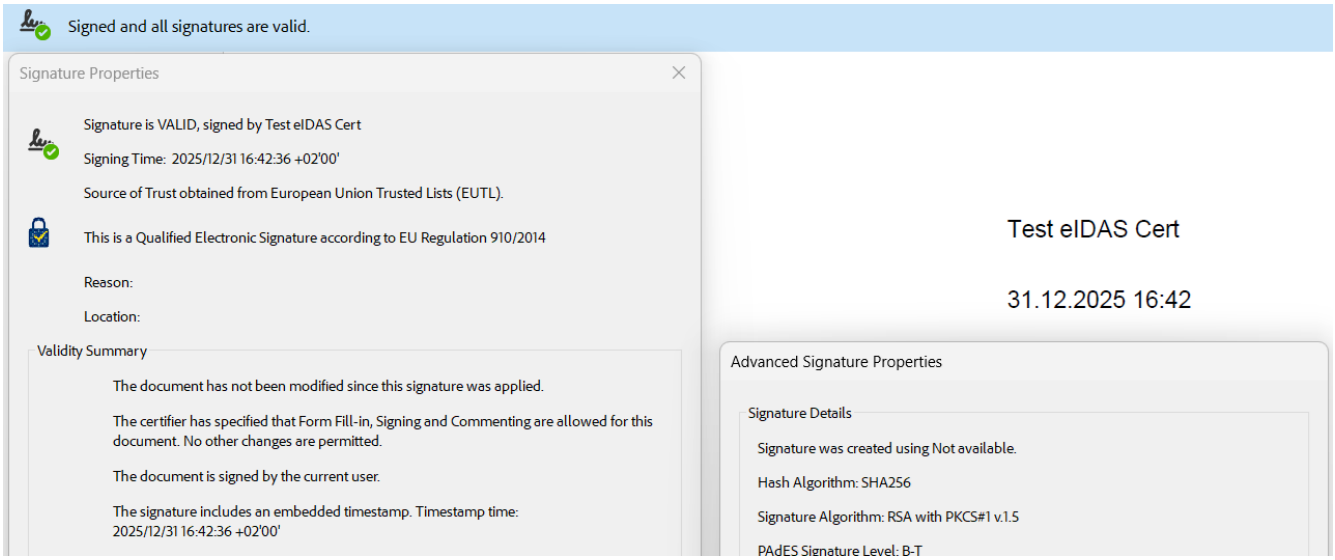
Digitally Sign a PDF File Using a Digital Certificate Stored on a PFX File

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

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

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

//load the PDF document
ps.LoadPdfDocument("d:\\source.pdf");
ps.SignaturePosition = SignaturePosition.TopRight;
ps.SigningReason = "I approve this document";

ps.SignaturePosition = SignaturePosition.TopRight;

//Load the signature certificate from a PFX or P12 file
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

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



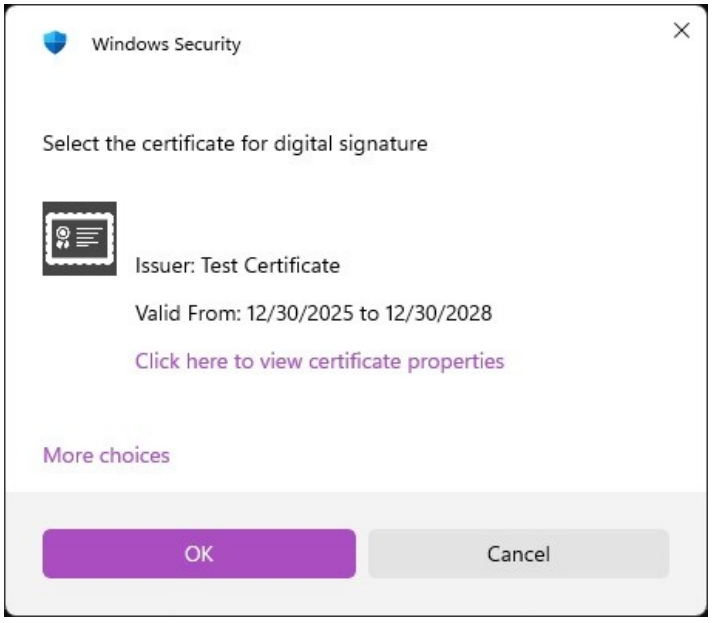
Digital signature properties on Adobe Reader

Perform a Digital Signature Using a Certificate stored on a Smart Card (USB Token)

To digitally sign a PDF using a certificate stored on the smart card, it must be first installed on Microsoft Certificate Store (see the section below: *Certificates Stored on Smart Cards or USB Tokens*)

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

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

//load the PDF document
ps.LoadPdfDocument("d:\\source.pdf");
ps.SignaturePosition = SignaturePosition.TopRight;
ps.SigningReason = "I approve this document";

ps.SignaturePosition = SignaturePosition.TopRight;

//Load the signature certificate from Microsoft Certificate Store
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate(false, "",
"Select the certificate", "");

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

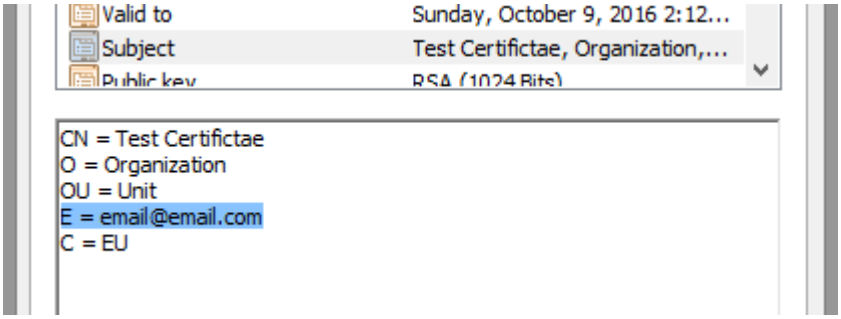
When the application is launched, the user must select the digital certificate from all certificates available in *Personal* tab.



Digital certificates selection window

Perform a Digital Signature Without User Intervention

In case the digital signature must be made without user intervention (automate the entire digital signature process), the certificate must be selected using a unique criteria.

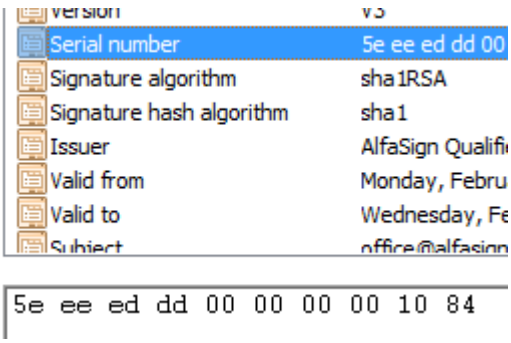


If the desired certificate has in the *Subject* field the value *E = email@email.com,* you can use the following code to automatically use the certificate for the signing operation.

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

PdfSignature ps = new PdfSignature("serial number");
//load the PDF document
ps.LoadPdfDocument("d:\\source.pdf");
ps.SignaturePosition = SignaturePosition.TopRight;
ps.SigningReason = "I approve this document";
ps.SignaturePosition = SignaturePosition.TopLeft;

//Load the certificate from Microsoft Certificate Store without user intervention
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate(false,
DigitalCertificateSearchCriteria.EmailE, "email@email.com");

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

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

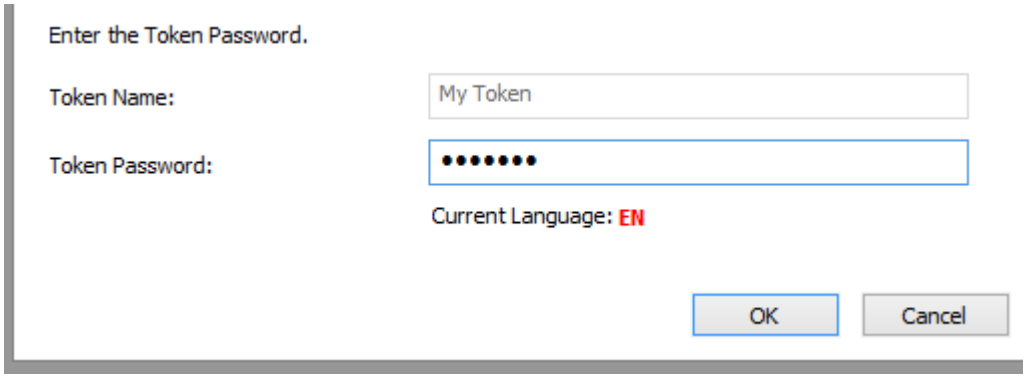
Note: Be careful if Serial Number criteria is used. On copy-paste operation, a non-printable character might be added.




```
//a non-printable character is added
‎5e ee ed dd 00 00 00 00 10 84
```

Bypassing the Smart Card PIN

In case the digital signature must be made without user intervention and the certificate is stored on a smart card or USB token, the PIN dialog might be automatically bypassed for some models.



PIN dialog can be bypassed

Attention: This feature will NOT work for all available smart card/USB tokens because of the drivers or other security measures. Use this property carefully.

In order to bypass the PIN dialog window, *DigitalCertificate.SmartCardPin* property must be set. The code below, bypass the PIN dialog and the file is automatically signed without any user intervention.

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

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

//load the PDF document
ps.LoadPdfDocument("d:\\source.pdf");
ps.SignaturePosition = SignaturePosition.TopRight;
ps.SigningReason = "I approve this document";
ps.SignaturePosition = SignaturePosition.TopLeft;



oad the certificate from Microsoft Certificate Store without user intervention
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate(false,
DigitalCertificateSearchCriteria.EmailE, "email@email.com");

//The PIN dialog is now bypassed
DigitalCertificate.SmartCardPin = "123456";

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

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.

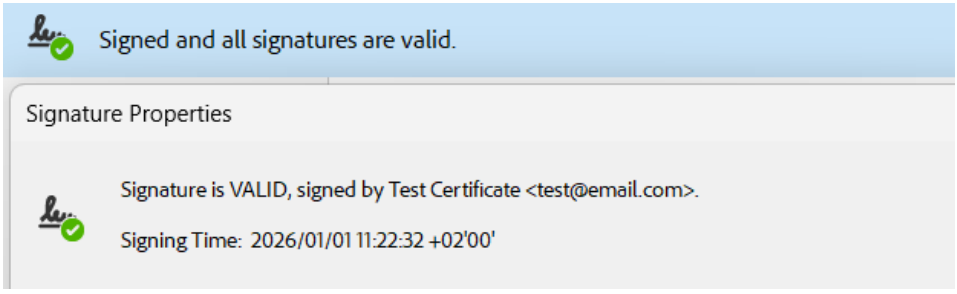
If the signing certificate (or the Root CA that issued the signing certificate) is not included in Adobe Store, the digital signature is considered "not trusted" when a user open a document with Adobe Reader (see example).

This behavior has nothing to do with the signing engine (e.g. .NET Digital Signature Library) 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: https://helpx.adobe.com/acrobat/kb/trust-services.html)

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

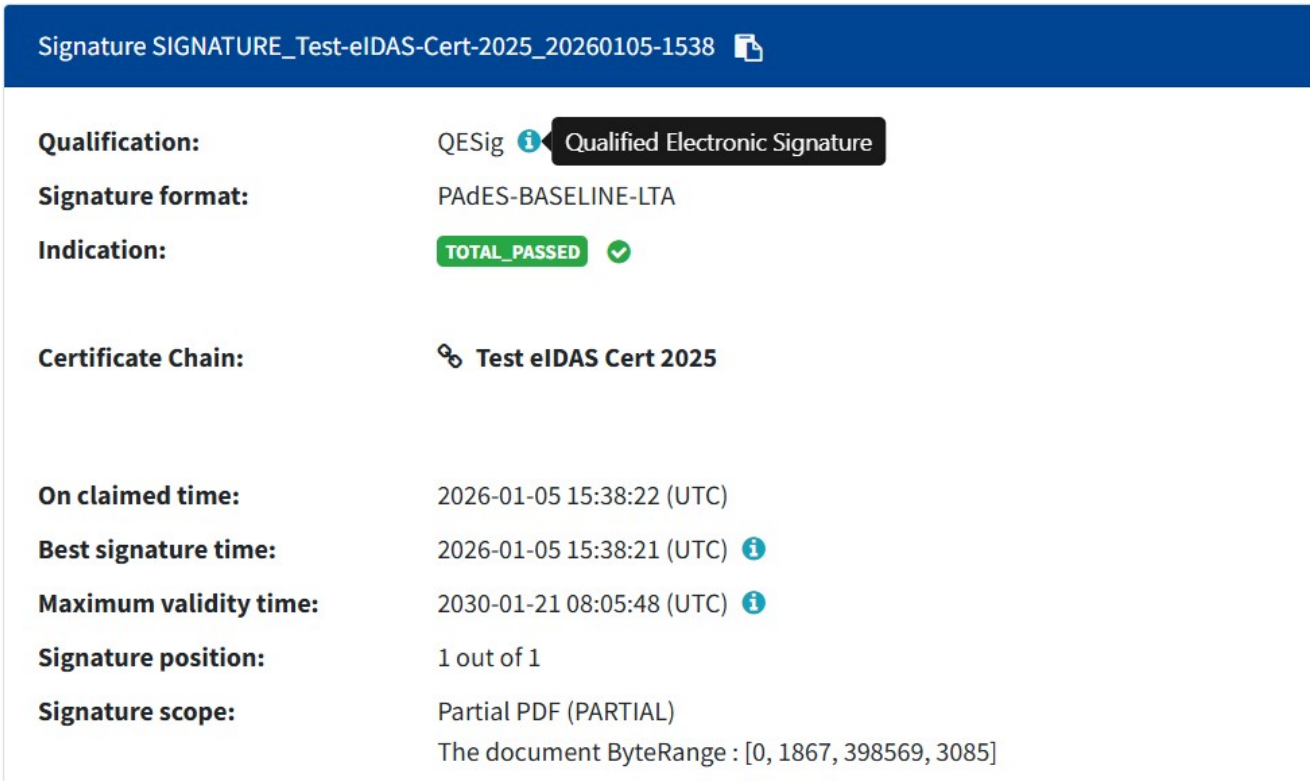
Validity Unknown signature



Valid signature

Validating Digital Signatures in eIDAS DSS

When an eIDAS Qualified Digital Certificate (or Seal) is used, the digital signatures generated by the library are successfully validated by the eIDAS DSS validator: https://ec.europa.eu/digital-building-blocks/DSS/webapp-demo/validation



PDF Digital Signatures

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("https://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:

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

//Load the PDF file
ps.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
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
```

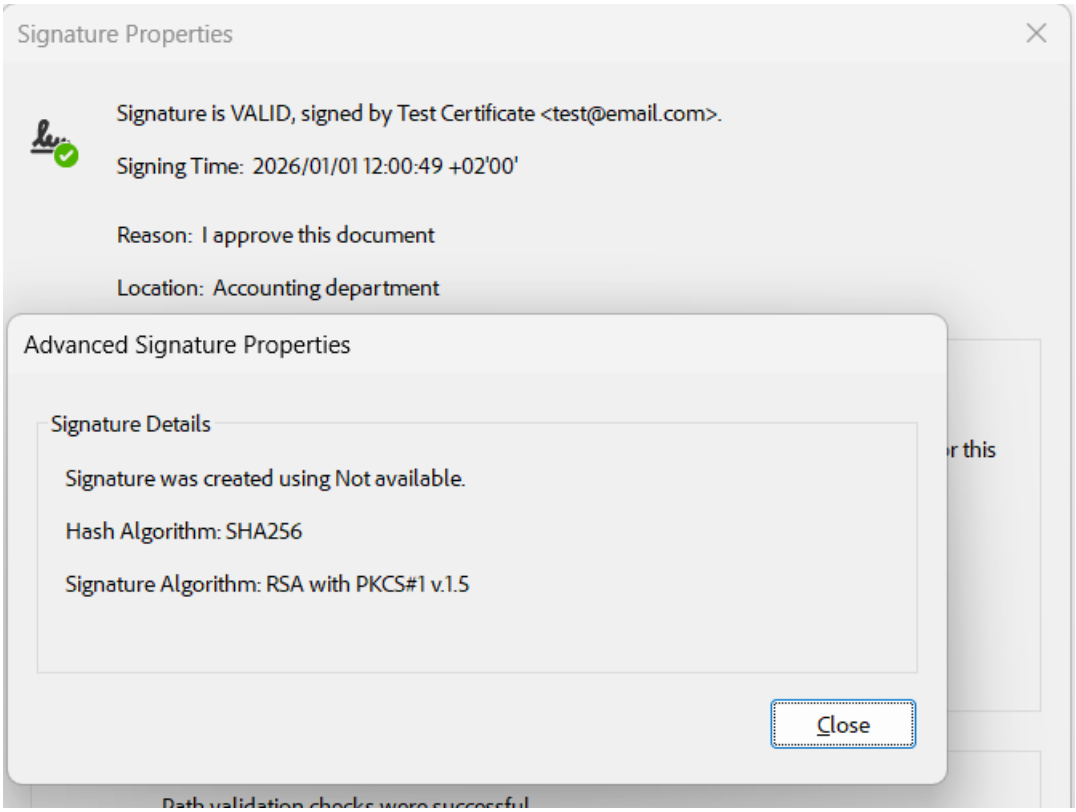
```
File.WriteAllBytes("c:\\dest.pdf", ps.ApplyDigitalSignature());
```

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*.

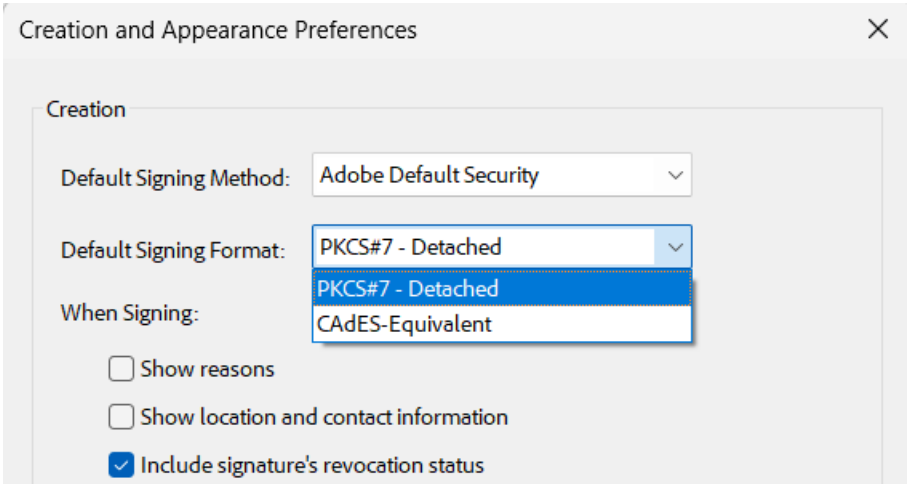
```
//hash algorithm used for creating the digntal signature
ps.HashAlgorithm = SignLib.HashAlgorithm.SHA256;

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

Attention: Some smart cards and USB tokens not support SHA-256, SHA-384 and SHA-512 hash algorithms.



Long Term Validation PAdES-LTV PDF Signatures

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 at least SHA-256).

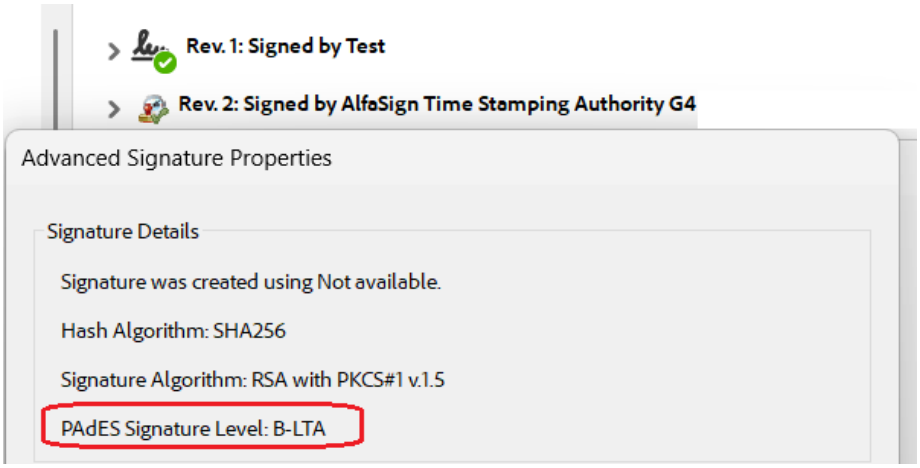
```
PdfSignature ps = new PdfSignature(serialNumber);

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

ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

ps.HashAlgorithm = SignLib.HashAlgorithm.SHA256;
ps.SignatureStandard = PdfSignatureStandard.Pades;

//include the revocation information (PAdES-LTV)
ps.PadesLtvLevel = PadesLtvLevel.IncludeCrlAndOcsp;

//optionally, the signature can be timestamped (SHA-256 algorithm must be used).
ps.TimeStamping.ServerUrl = new Uri("https://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.

Long-Term Archival PAdES-LTA PDF Signatures

In order to use PAdES-LTA, use the code below (note that the signature must be at least SHA-256).

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

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

ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

ps.HashAlgorithm = SignLib.HashAlgorithm.SHA256;
ps.SignatureStandard = PdfSignatureStandard.PadesLTA;

//include the revocation information (PAdES-LTA)
ps.PadesLtvLevel = PadesLtvLevel.IncludeCrlAndOcsp;

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

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



Long-Term Archival PAdES-LTA PDF Signature in Adobe

Validating Long-Term Archival PAdES-LTA PDF Signature in eIDAS DSS

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: https://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");

//Load the signature certificate from Microsoft Certificate Store
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("cert.pfx",
"123456");

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

ps.TimeStamping.AuthenticationCertificate = DigitalCertificate.LoadCertificate("d:
\\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.:

```
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("https://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-256.
In order to use SHA-384 or SHA-512 hashing algorithm, check the property *TimeStamping.HashAlgorithm*.

```
ps.TimeStamping.HashAlgorithm = SignLib.HashAlgorithm.SHA384;
```

Attention: Some smart cards and USB tokens not support SHA-256, SHA-384 and SHA-512 hashing algorithms.

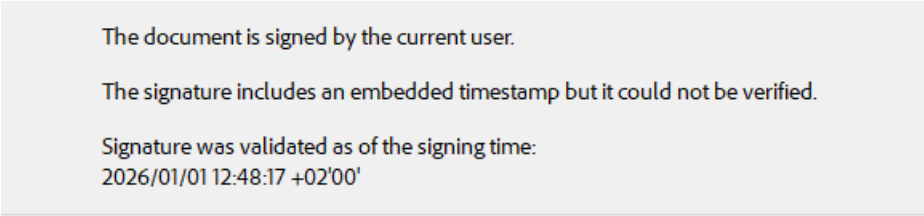
Validating the Time Stamping Response on Adobe

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

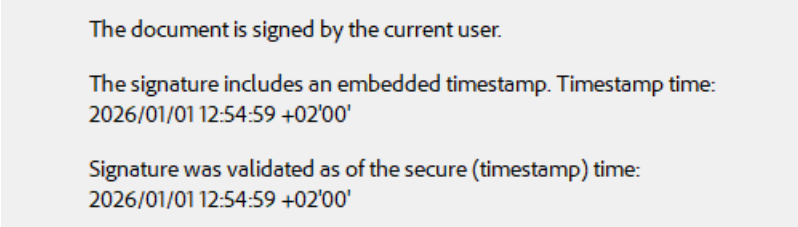
If the time stamping certificate (or the Root CA that issued the time stamping certificate) is not included in Adobe Store, the time stamping response could not be verified when a user open a document with Adobe Reader (see example).

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: https://www.signfiles.com/manuals/ValidatingDigitalSignaturesInAdobe.pdf.



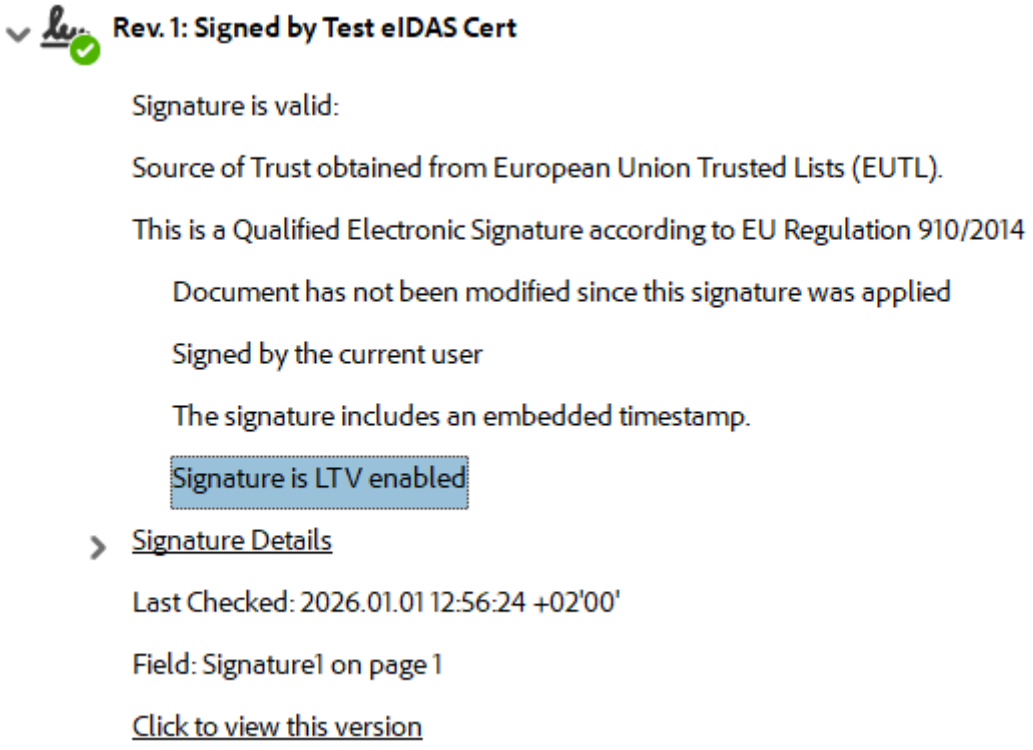
Not verified timestamp



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

```
ps.PadesLtvLevel = PadesLtvLevel.IncludeCrlAndOcsp; //include both CRL and OCSP
ps.MaxCrlSize = 2048 * 1024; //2 MB - very large CRL will also be added
ps.SignatureStandard = SignLib.Pdf.PdfSignatureStandard.Pades; //PAdES signature
```

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 a document is certified, 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

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 about 20.000 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:

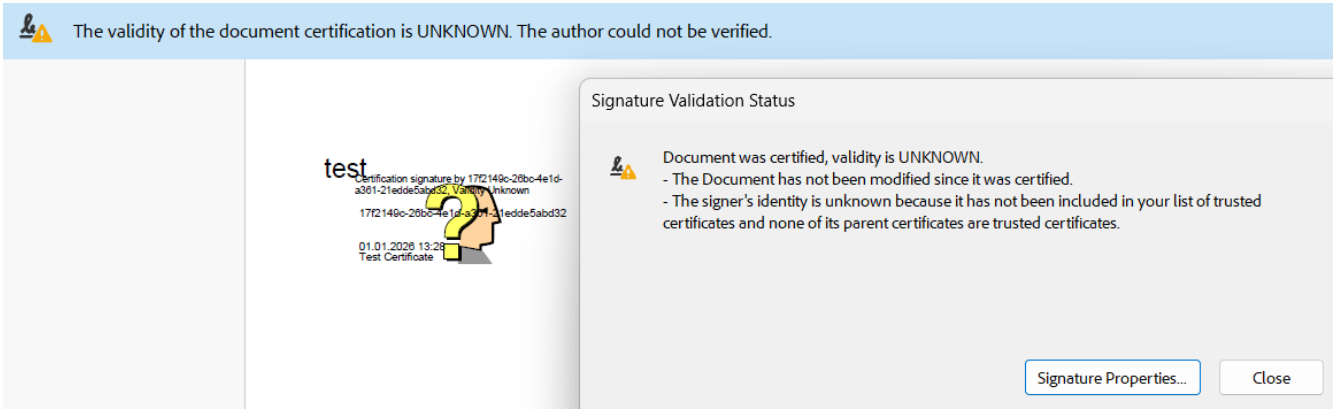
```
ps.SignatureByteBlockSize = 8192;
```

Old Style Adobe Digital Signature Appearance

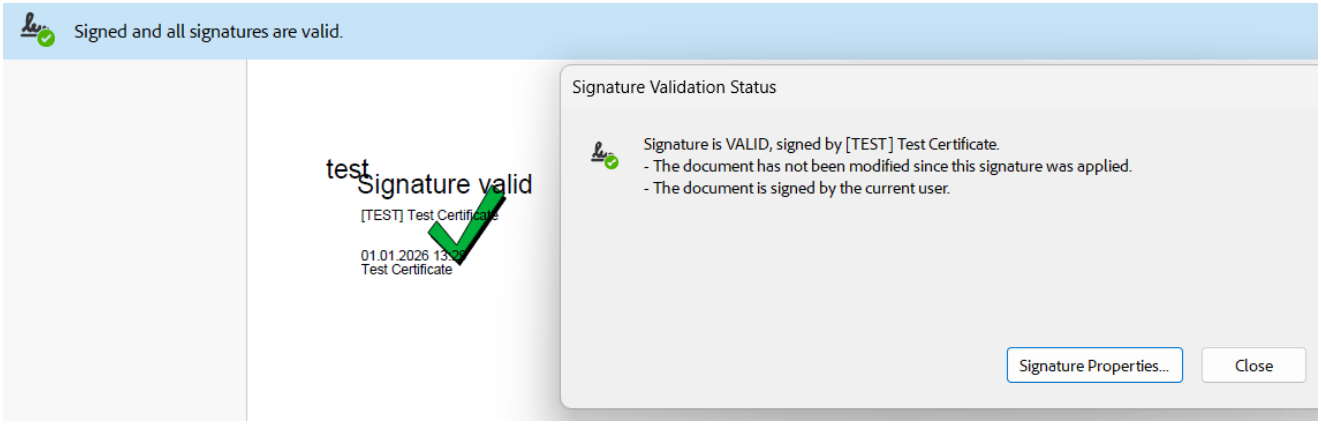
To use an old style appearance of the digital signature rectangle (see example) set the *OldStyleAdobeSignature* property to true. The default value is false.

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

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



Validity unknown signature



Signature valid

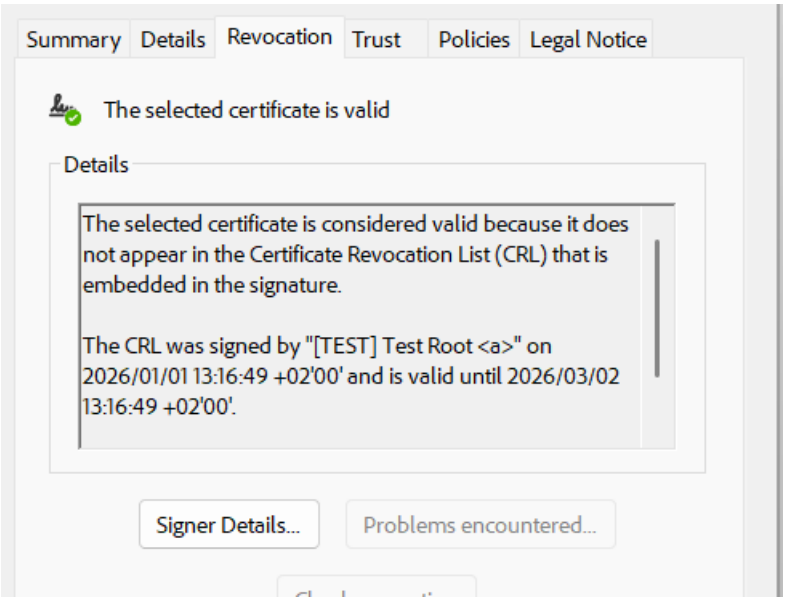
Include the Revocation Information on the PDF Signature

If the 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.

To include the revocation information, see the code below:

```
s.PadesLtvLevel = PadesLtvLevel.IncludeCrlAndOcsp;
//very large CRL will also be added
ps.MaxCrlSize = 2048 * 1024; //2 MB
ps.SignatureStandard = SignLib.Pdf.PdfSignatureStandard.Pades;
```

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.



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

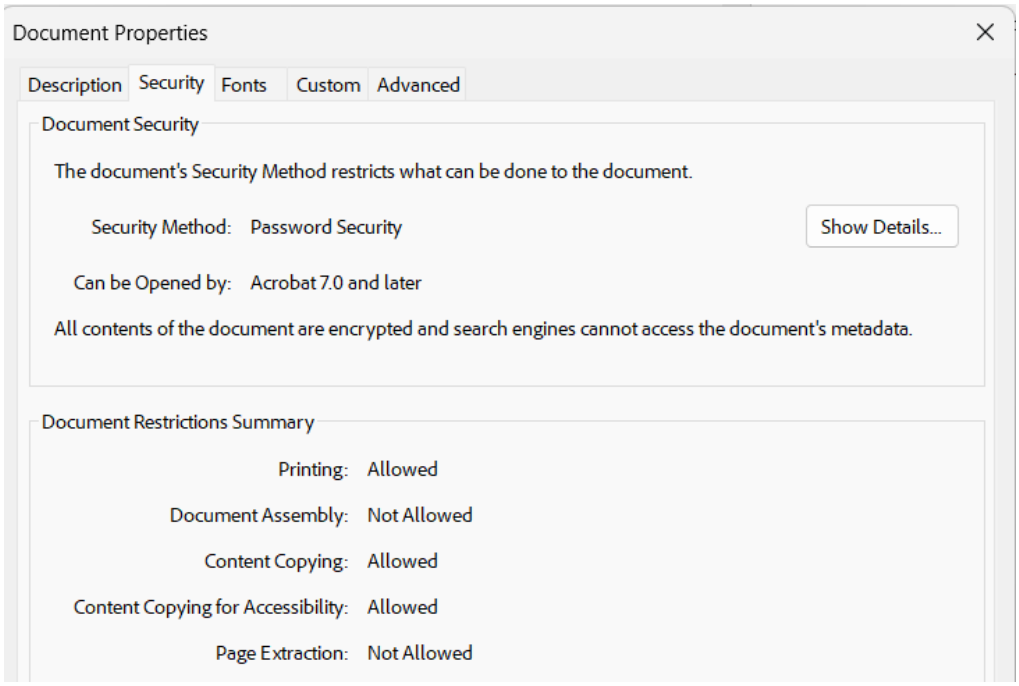
In order to encrypt the PDF document, the *AppendSignature* propery must be set to false. Also, the encryption algorithm must be specified using *EncryptionAlgorithm* property.

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"));

//Load the certificate from .PFX
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

//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.AllowPrinting;

//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.

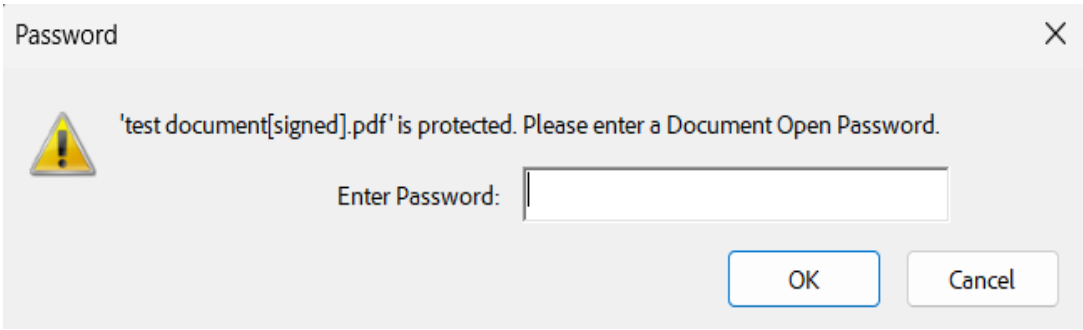


Security settings for a digitally sign and encrypted document

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

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:

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

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

//Load the signing  certificate from .PFX
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

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

//set the document restrictions
ps.Encryption.DocumentRestrictions = PdfDocumentRestrictions.AllowNone;

//set the encryption algorithm
ps.Encryption.EncryptionAlgorithm =
PdfEncryptionAlgorithm.EnhancedEncryption128BitAES;

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

//select the encryption certificate from Microsoft Store
ps.Encryption.EncryptionCertificate = DigitalCertificate.LoadCertificate(false,
string.Empty, "Select Certificate", "Select the certificate for encryption");

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

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

```
//ps.Encryption.EncryptionCertificate = DigitalCertificate.LoadCertificate(false,
string.Empty, "Select Certificate", "Select the certificate for encryption");

ps.Encryption.EncryptionCertificate = new
System.Security.Cryptography.X509Certificates.X509Certificate2(File.ReadAllBytes("
d:\\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:

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:

PDF Signature Code Samples

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

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

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

ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");
//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 and Sign Using a Smart Card Certificate

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

PdfSignature ps = new PdfSignature("serial number");
ps.LoadPdfDocument("d:\\source.pdf");
//load the certificate from Microsoft Store
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate(false, "",
"Select Certificate", "");
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("https://www.signfiles.com/test.pdf"));

//sign the document only if it is not signed
if (PDFSign.DocumentProperties.DigitalSignatures.Count == 0)
{
      PDFSign.DigitalSignatureCertificate =
      DigitalCertificate.LoadCertificate("d:\\cert.pfx", "123456");
      File.WriteAllBytes("c:\\dest.pdf", PDFSign.ApplyDigitalSignature());
}
```

Digitally Sign a PDF file with a Remote Signature Service (External Signature)

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

PdfSignature ps = new PdfSignature("serial number");
ps.LoadPdfDocument("d:\\source.pdf");
ExternalSignature exSignature = new ExternalSignature();

//set the certificate
ps.DigitalSignatureCertificate = exSignature.RemoteSignatureCertificate;

//bind the external signature with the library
SignLib.Certificates.DigitalCertificate.UseExternalSignatureProvider = exSignature;

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


public class ExternalSignature : SignLib.Certificates.IExternalSignature
{
  public ExternalSignature()
    {
      //obtaining the remote signature certificate
      remote.signature.RemoteSignature rs = new remote.signature.RemoteSignature();
      RemoteSignatureCertificate = new X509Certificate2(rs.GetSigningCertificate());
    }
  public X509Certificate2 RemoteSignatureCertificate { get; set; }
  public byte[] ApplySignature(byte[] dataToSign, Oid hashAlgorithm)
      {
          //only the document hash is sent to the Remote Signature Server
          remote.signature.RemoteSignature rs = new remote.signature.RemoteSignature();
          return rs.RemoteSignature(dataToSign, hashAlgorithm);
      }
 }
```

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

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

string certificatePassword = "tempP@ssword";

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

//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.SHA256WithRSA;

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

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

//Load the new certificate
ps.DigitalSignatureCertificate =
DigitalCertificate.LoadCertificate(digitalCertificate, certificatePassword);

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

//digitally sign the PDF file
File.WriteAllBytes("d:\\dest.pdf", ps.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.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

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.DigitalSignatureCertificate.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

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

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

ps.LoadPdfDocument("d:\\source.pdf");
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

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 order to keep the file as PDF/A
ps.FontFile = "c:\\windows\\fonts\\verdana.ttf";

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

//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.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

//Set the TSA Server URL
ps.TimeStamping.ServerUrl = new Uri("https://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.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

//Set the TSA Server URL
ps.TimeStamping.ServerUrl = new Uri("https://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.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

ps.TimeStamping.ServerUrl = new Uri("https://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 web Application (IIS)

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

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

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

            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;

//automaticall load the digital signature certificate using email criteria
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate(false,
DigitalCertificateSearchCriteria.EmailE, "user@test.com", false);

//bypass the smart card PIN
DigitalCertificate.SmartCardPin = "123456";

ps.TimeStamping.ServerUrl = new Uri("https://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 an Existing Adobe Signature Form Field

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

//load the PDF document
ps.LoadPdfDocument(unsignedDocument);
ps.SigningReason = "I approve this document";
ps.SigningLocation = "Accounting department";

//Digital signature certificate can be loaded from various sources

//Load the signature certificate from a PFX or P12 file
ps.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("cert.pfx",
"123456");


//digitally sign an existing signature field (the name of the field must be known)
File.WriteAllBytes(signedDocument, ps.ApplyDigitalSignature("Signature1"));

Console.WriteLine("The PDF signature field was signed." + Environment.NewLine);
```

Apply a PDF Timestamp Signature

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

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

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

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

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 and PKCS#7 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);

//Digital signature certificate can be loaded from various sources

//Load the signature certificate from a PFX or P12 file
cs.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate("d:\
\cert.pfx", "123456");

//Load the certificate from Microsoft Store.
//The smart card or USB token certificates are usually available on Microsoft
Certificate Store (start - run - certmgr.msc).
//If the smart card certificate not appears on Microsoft Certificate Store it
cannot be used by the library
//cs.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate(false,
string.Empty, "Select Certificate", "Select the certificate for digital
signature");

//The smart card PIN dialog can be bypassed for some smart cards/USB Tokens.
//ATTENTION: This feature will NOT work for all available smart card/USB Tokens
becauase of the drivers or other security measures.
//Use this property carefully.
//DigitalCertificate.SmartCardPin = "123456";

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

//SHA256 algorithm is recommended for CAdES signatures
cs.HashAlgorithm = SignLib.HashAlgorithm.SHA256;

cs.SignatureStandard = SignLib.SignatureStandard.CadesBes;

//write the signed file
//usually, the signed CAdES file should be saved with .p7s or .p7m extension
File.WriteAllBytes("d:\\test.txt.p7m", cs.ApplyDigitalSignature("d:\\test.txt"));

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

Creating PKCS#7 Signatures

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

CadesSignature cs = new CadesSignature(serialNumber);

//Load the signature certificate from a PFX or P12 file
cs.DigitalSignatureCertificate =
DigitalCertificate.LoadCertificate(Environment.CurrentDirectory + "\\cert.pfx",
"123456");

//Load the certificate from Microsoft Store.
//The smart card or USB token certificates are usually available on Microsoft
Certificate Store (start - run - certmgr.msc).
//If the smart card certificate not appears on Microsoft Certificate Store it
cannot be used by the library
//cs.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate(false,
string.Empty, "Select Certificate", "Select the certificate for digital
signature");

//The smart card PIN dialog can be bypassed for some smart cards/USB Tokens.
//ATTENTION: This feature will NOT work for all available smart card/USB Tokens
becauase of the drivers or other security measures.
//Use this property carefully.
//DigitalCertificate.SmartCardPin = "123456";

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

//The hash algorithm can be set
//cs.HashAlgorithm = SignLib.HashAlgorithm.SHA256;

cs.SignatureStandard = SignLib.SignatureStandard.CadesBes;

//write the signed file
//usually, the signed PKCS#7 file should be saved with .p7s or .p7m extension
File.WriteAllBytes(signedDocument, cs.ApplyDigitalSignature(unsignedDocument));

Console.WriteLine("The PKCS#7 signature was created." + Environment.NewLine);
```

Verifying CAdES/PKCS#7 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.p7m", 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);
}
```

CAdES Signature Types

CMS Signature

This is a basic signature compatible with the most of the signature validators. No special attributes are added.

```
cs.SignatureStandard = SignLib.Cades.CadesSignatureStandard.Cms;
```

This signature can be timestamped and the timestamp is saved on the attribute *id-aa-timeStampToken ("1.2.840.113549.1.9.16.2.14")*.

CAdES-BES Signature

```
cs.SignatureStandard = SignLib.Cades.CadesSignatureStandard.CadesBes;
```

This signature is a CMS Signature with two signed attributes: *EssCertID and EssCertIDv2.*

If a CAdES-BES signature is timestamped, a CadES-T signature is generated.

The CAdES timestamps are saved in two attributes: *id-aa-timeStampToken ("1.2.840.113549.1.9.16.2.14") and id-aa-ets-escTimeStamp ("1.2.840.113549.1.9.16.2.25").*

CAdES-EPES Signature

```
cs.SignatureStandard = SignLib.Cades.CadesSignatureStandard.CadesBes;

//create the Signature Policy Info (required by some national regulations
Org.BouncyCastle.Asn1.Esf.SignaturePolicyInfo spi = <your data here>;
//adding the Signature Policy Info to the signature
cs.SignaturePolicyInfo = spi;
```

If for a CAdES-BES signature, Signature Policy Info is set, results a CAdES-EPES signature.

CAdES-C Signature

```
cs.SignatureStandard = SignLib.Cades.CadesSignatureStandard.CadesC;
```

This signature is a CAdES-BES signature with two unsigned attributes: *IdAaEtsCertificateRefs ("1.2.840.113549.1.9.16.2.21") and IdAaEtsCertValues("1.2.840.113549.1.9.16.2.22").*

An internet connection is required in order to get the references to the CRLs and the OCSPs responses.

This signature is timestamped.

CAdES-XL Signature

```
cs.SignatureStandard = SignLib.Cades.CadesSignatureStandard.CadesXL;
```

This signature is a CAdES-C signature with two unsigned attributes: *IdAaEtsCertValues ("1.2.840.113549.1.9.16.2.23") and IdAaEtsCertValues("1.2.840.113549.1.9.16.2.24").*

An internet connection is required in order to get the references to the CRLs and the OCSPs responses. The CRL files will be included on the signed file so it can be very large in some cases.

This signature is timestamped.

CAdES-LT Signature

```
cs.SignatureStandard = SignLib.Cades.CadesSignatureStandard.CadesLT;
```

This signature is a CAdES-C signature with *IdAaEtsCertificateRefs, IdAaEtsCertValues, IdAaEtsRevocationRefs and IdAaEtsRevocationValues* attributes.

An internet connection is required in order to get the references to the CRLs and the OCSPs responses. The CRL files will be included on the signed file so it can be very large in some cases.

This signature is timestamped.

CAdES-A Signature (Long-Term Archival)

```
cs.SignatureStandard = SignLib.Cades.CadesSignatureStandard.CadesA;
```

This signature is a timestamped CAdES-XL signature with two unsigned attributes: *id-aa-ets-archiveTimestampV2 ("1.2.840.113549.1.9.16.2.48") and id-aa-ets-escTimeStamp ("1.2.840.113549.1.9.16.2.25").*

This signature is timestamped.

Attention! This profile is NOT fully compliant with all standards.

XML Digital Signatures (XMLDSig Standard)

The library is able to create XMLDSig signatures.

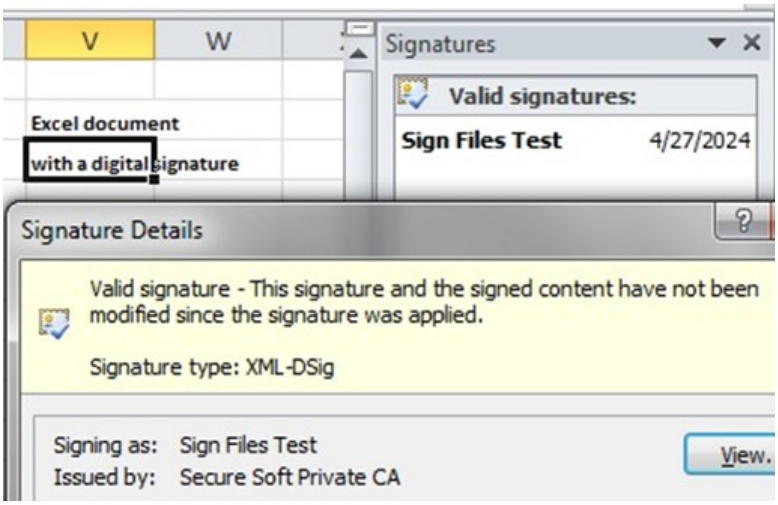
```
using SignLib;
using SignLib.Certificates;

XmlSignature cs = new XmlSignature(serialNumber);

cs.DigitalSignatureCertificate =
DigitalCertificate.LoadCertificate(Environment.CurrentDirectory + "\\cert.pfx",
"123456");

//apply the digital signature
cs.ApplyDigitalSignature("test.xml", "test[signed].xml");

XmlSignature cv = new XmlSignature(serialNumber);

//for SHA-256 signatures, use XmlSignatureSha256 class

Console.WriteLine("Signatures: " + cv.GetNumberOfSignatures("test[signed].xml"));
```

```
- <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>PW5FcXY11Kf1JGr5gA2OLmbIOUc=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>QmPKCXhGVy0+RRbh3wzqIpIkrHpDBTSqMBr+MDLnyw/I3DeXh8to743s+paLuvdp
  - <KeyInfo>
    - <KeyValue>
      - <RSAKeyValue>
          <Modulus>nvbwLDDphzQznQGPfh/kM/HUILza5VxHZKAb80zgaWNaj15+dDIIMFvMa8UUI
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    - <X509Data>
        <X509Certificate>MIICjjCCAfegAwIBAgIRAKua7Iy+gKuw5efYjzXGq9UwDQYJKoZIhvcNAQEFI
      </X509Data>
    </KeyInfo>
  </Signature>
```

Office and XPS Digital Signatures (.NET Framework Only)

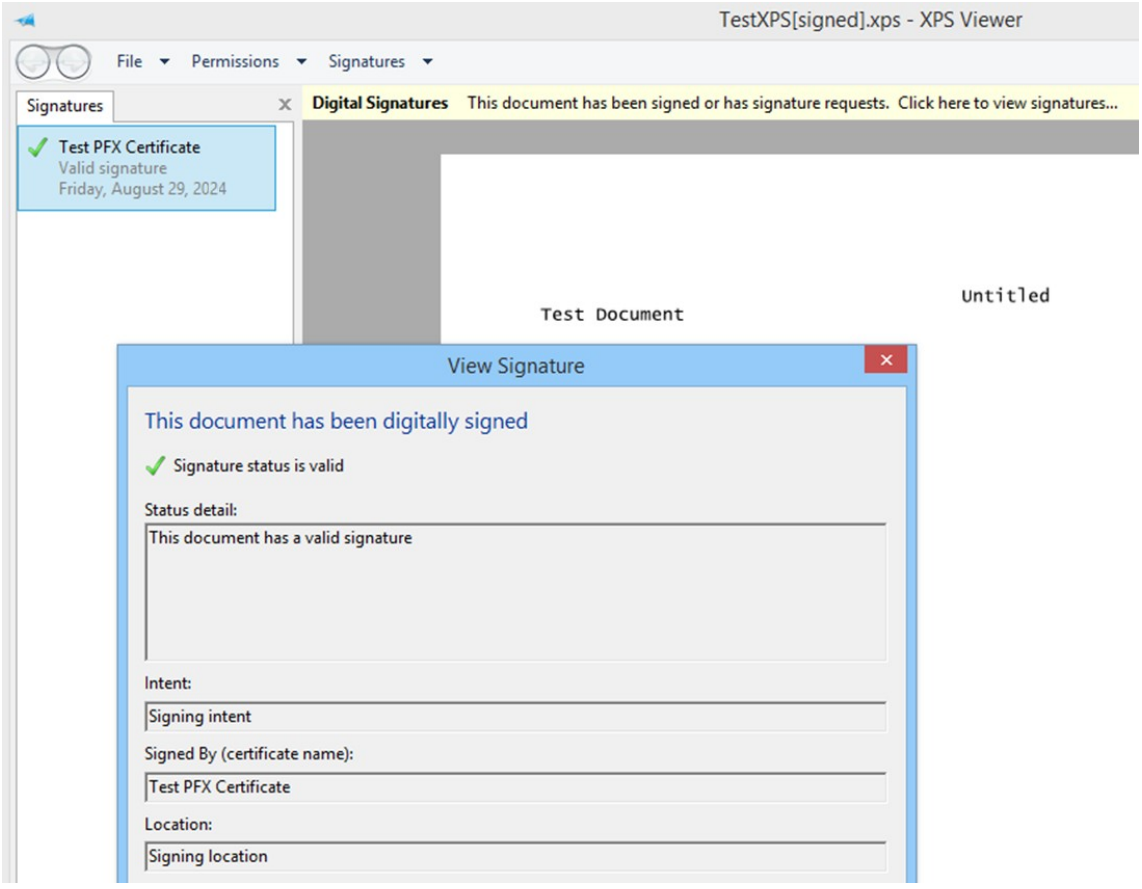
Digitally Sign and Verify an Office Document (.docx, .xlsx)

```
using SignLib;
using SignLib.Certificates;

OfficeSignature cs = new OfficeSignature("serial number");

//Digital signature certificate can be loaded from various sources

//Load the signature certificate from a PFX or P12 file
cs.DigitalSignatureCertificate =
DigitalCertificate.LoadCertificate(Environment.CurrentDirectory + "\\cert.pfx",
"123456");
//Load the certificate from Microsoft Store.
//The smart card or USB token certificates are usually available on Microsoft
Certificate Store (start - run - certmgr.msc).
//If the smart card certificate not appears on Microsoft Certificate Store it
cannot be used by the library
//cs.DigitalSignatureCertificate = DigitalCertificate.LoadCertificate(false,
string.Empty, "Select Certificate", "Select the certificate for digital
signature");
//The smart card PIN dialog can be bypassed for some smart cards/USB Tokens.
//ATTENTION: This feature will NOT work for all available smart card/USB Tokens
becauase of the drivers or other security measures.
//Use this property carefully.
//DigitalCertificate.SmartCardPin = "123456";
cs.ApplyDigitalSignature("test.docx", "test[signed].docx");

OfficeSignature cv = new OfficeSignature("serial number");
Console.WriteLine("Number of signatures: " +
cv.GetNumberOfSignatures("test[signed].docx"));
//verify the first signature
Console.WriteLine("Signature validity status: " +
cv.VerifyDigitalSignature("test[signed].docx", 1));
```



Digitally Sign an XPS Document

```
using SignLib;
using SignLib.Certificates;

XpsSignature cs = new XpsSignature(serialNumber);

cs.DigitalSignatureCertificate =
DigitalCertificate.LoadCertificate(Environment.CurrentDirectory + "\\cert.pfx",
"123456");

cs.SigningLocation = "My location";
cs.SigningIntent = "I attest the accuracy of this document";

//apply the digital signature
cs.ApplyDigitalSignature("test.xps", "test[signed].xps");

XpsSignature cv = new XpsSignature(serialNumber);

Console.WriteLine("Signatures: " + cv.GetNumberOfSignatures("test[signed].xps"));

///verify the first signature
Console.WriteLine("Status: " + cv.VerifyDigitalSignature("test[signed].xps", 1));
```



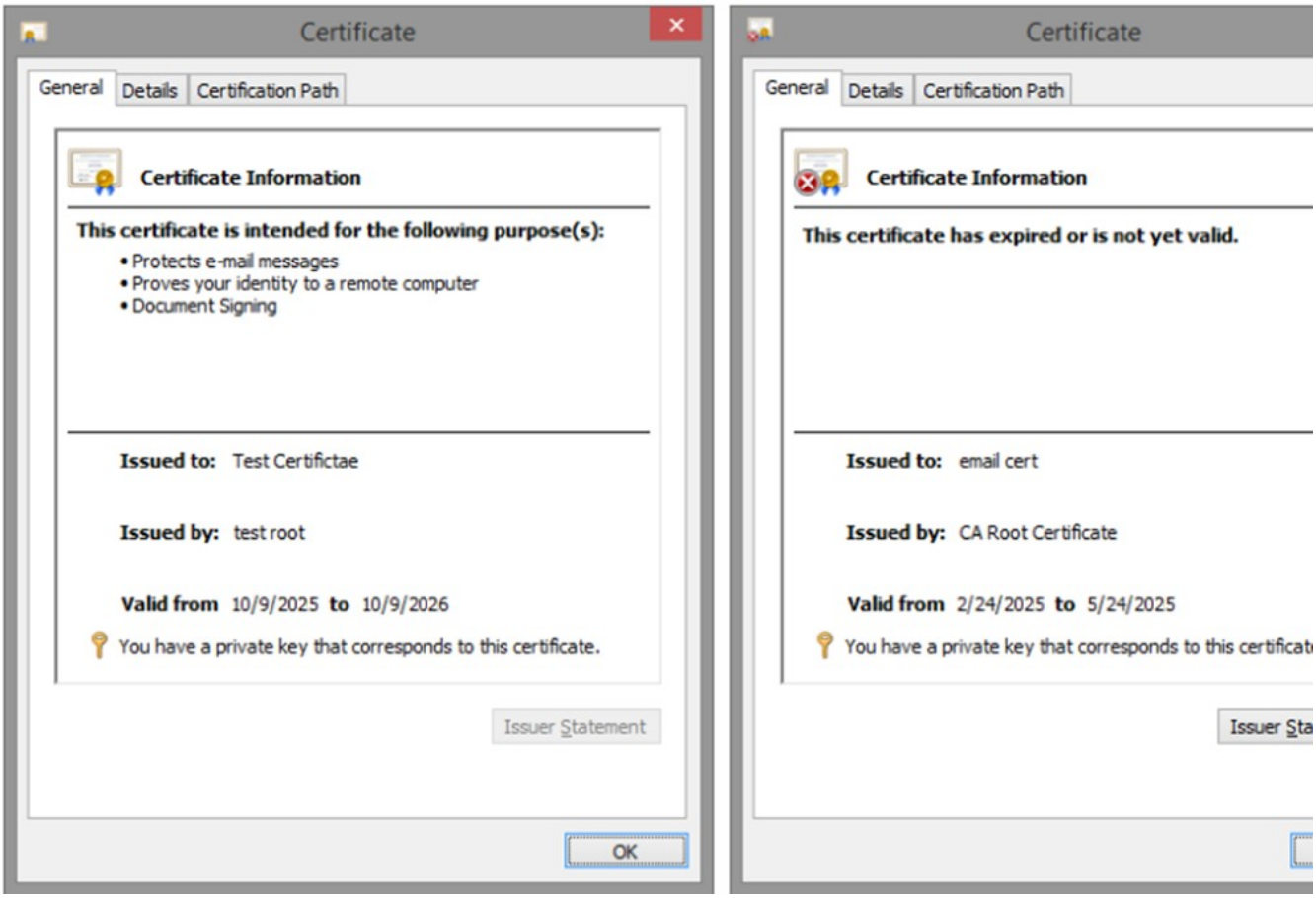
Validating Digital Certificates

A digital certificate can be validated against 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 signatures.



Time valid certificate versus an expired certificate

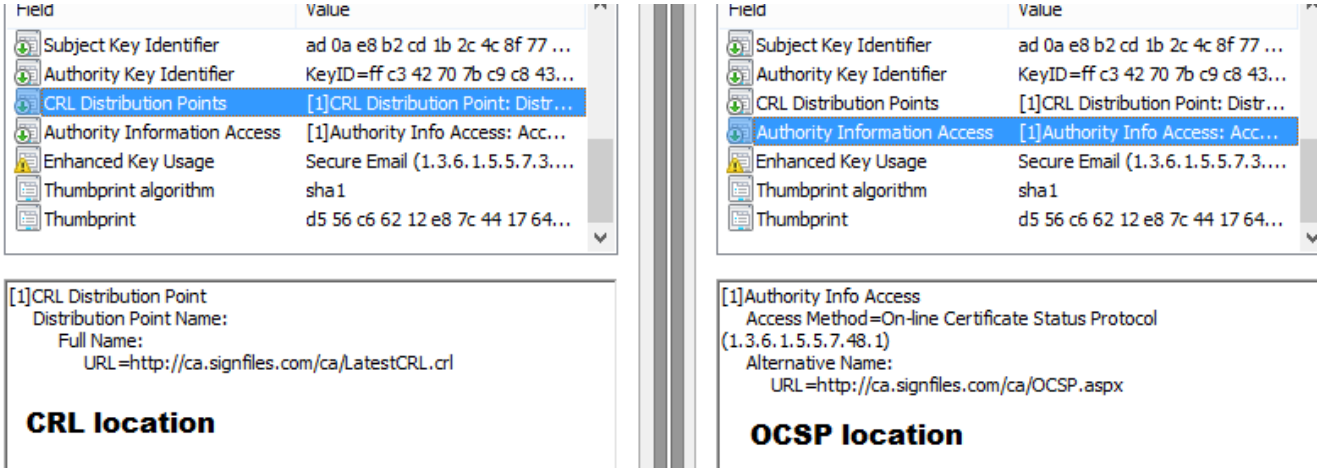
CRL and OCSP Validation

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.).

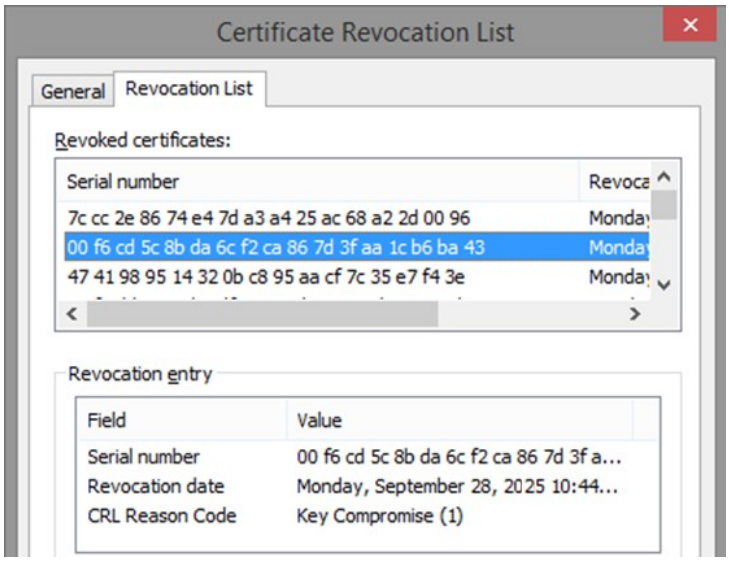
When a certificate is revoked, the certificate serial number is added on the CRL. To verify if a certificate is revoked, the CRL must be downloaded and check if the certificate serial number appears on the CRL.

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.

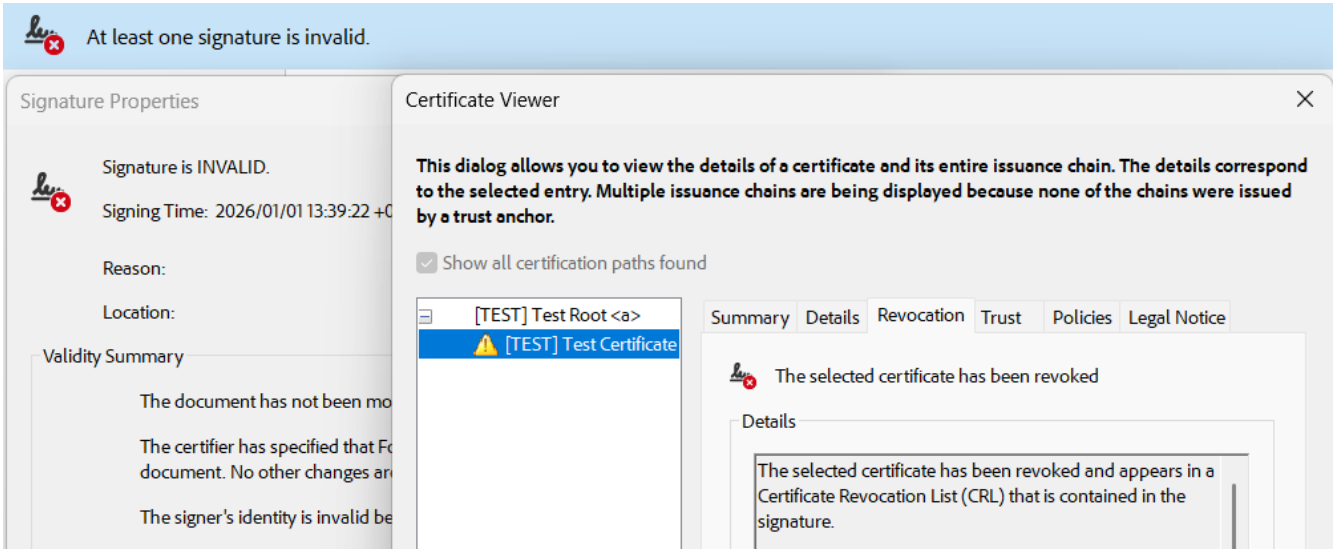


A certificate with CRL and OCSP



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
```

Creating Digital Certificates

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).

Certificate Subject

Every certificate must have a *Subject*. There are two methods to set the certificate subject.

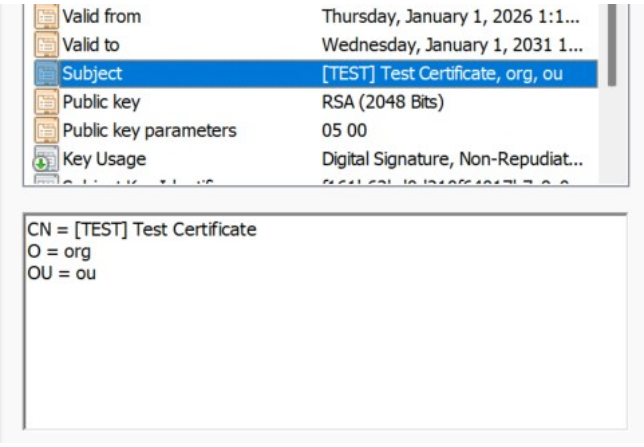
If the subject contains comma characters (“**,**” e.g. *My Company, Subsidiary 1*), the first method must be used. The *Subject* can contains Unicode characters like *ä,æ, £, Ñ*.

1. Manually set every *SubjectType* of the certificate using the following code:

```
using SignLib.Certificates;

X509CertificateGenerator cert = new X509CertificateGenerator("serial number");
cert.AddToSubject(SubjectType.CN, "Certificate name");
cert.AddToSubject(SubjectType.E, "name@email.com");

//comma character is permitted on the Subject name
cert.AddToSubject(SubjectType.O, "My Company, Subsidiary 1");

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

2. Set the *Subject* property:

```
using SignLib.Certificates;

X509CertificateGenerator cert = new X509CertificateGenerator("serial number");
//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("d:\\cert.pfx", cert.GenerateCertificate("password", false));
```



Certificate Subject

Validity Period

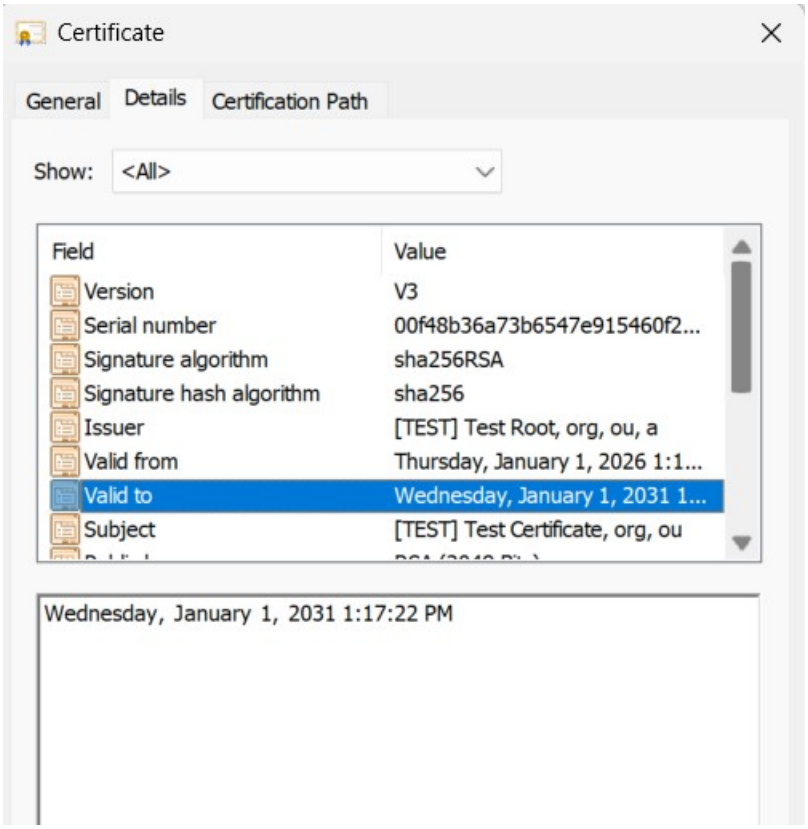
Every certificate has a validity period. A certificate becomes invalid after it expires. To set the validity period of the certificate use the following code:

```
using SignLib.Certificates;

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));
```

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.



Certificate validity period

Key Size and Signature Algorithm

The certificates issued by the library 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:

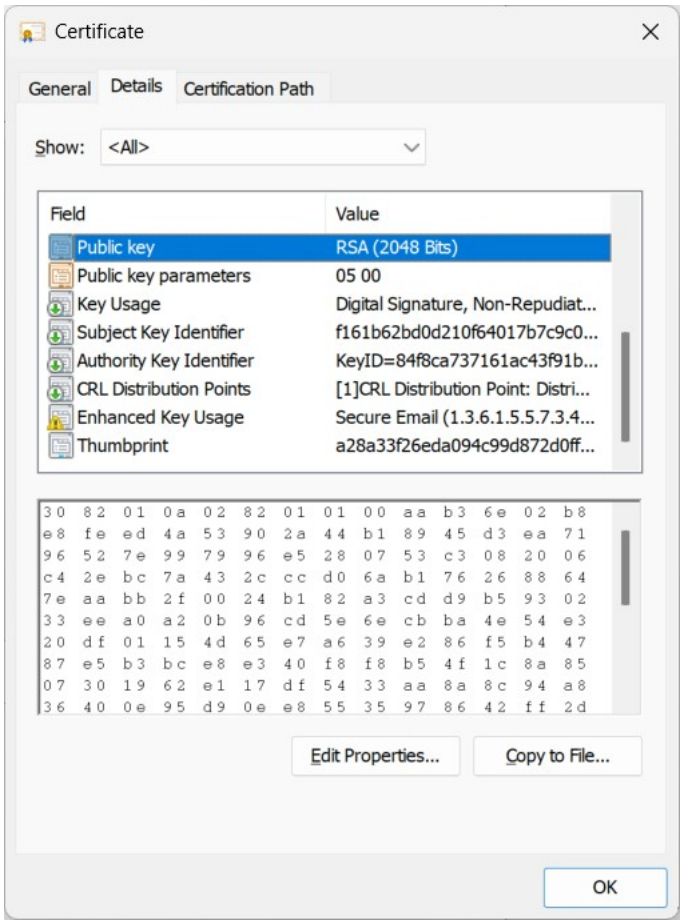
```
using SignLib.Certificates;

X509CertificateGenerator cert = new X509CertificateGenerator("serial number");
//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.KeySize2048Bit* and should be enough for

common certificates. For the Root certificates a 2048 key can be used.

The default value of *SignatureAlgorithm* property is *SignatureAlgorithm.SHA256WithRSA*.

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

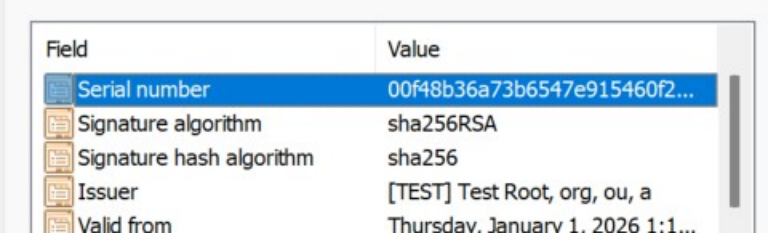
Every certificate must have a serial number. If the *SerialNumber* property is not set, a random value will be used.

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 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.

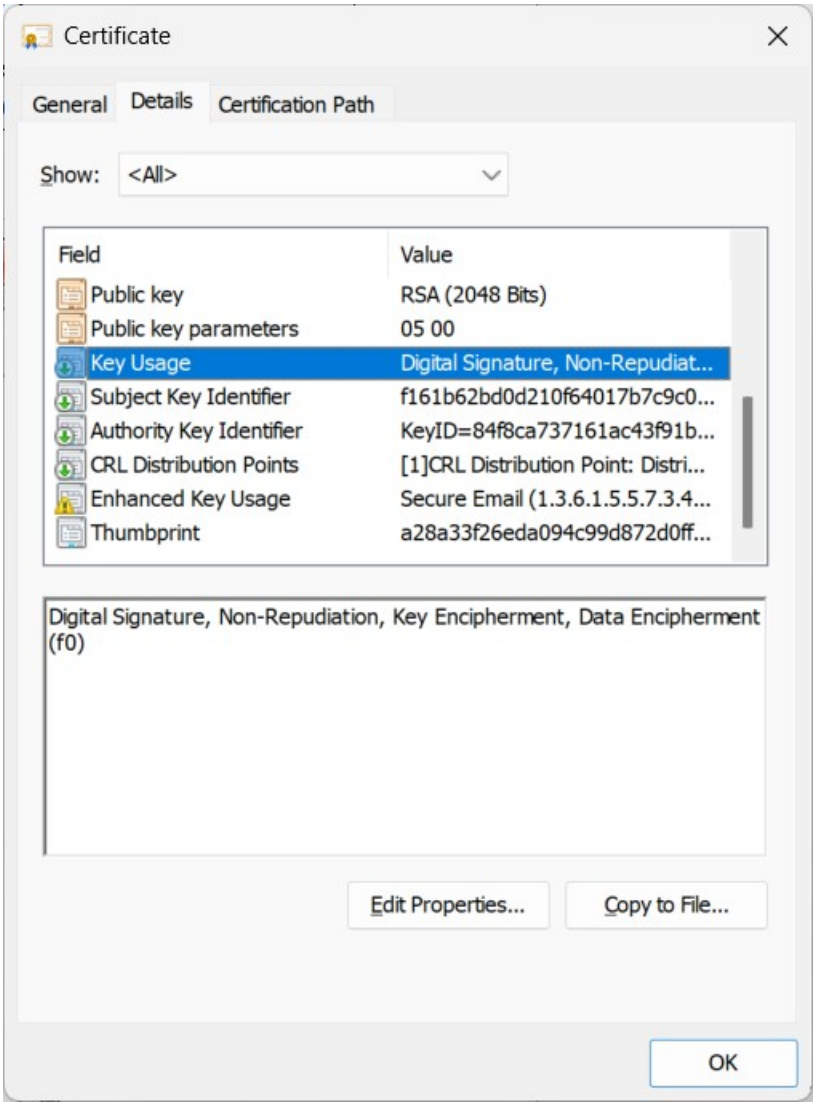
DecipherOnly - The certificate public key may be used only for enciphering data while performing key agreement.

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:

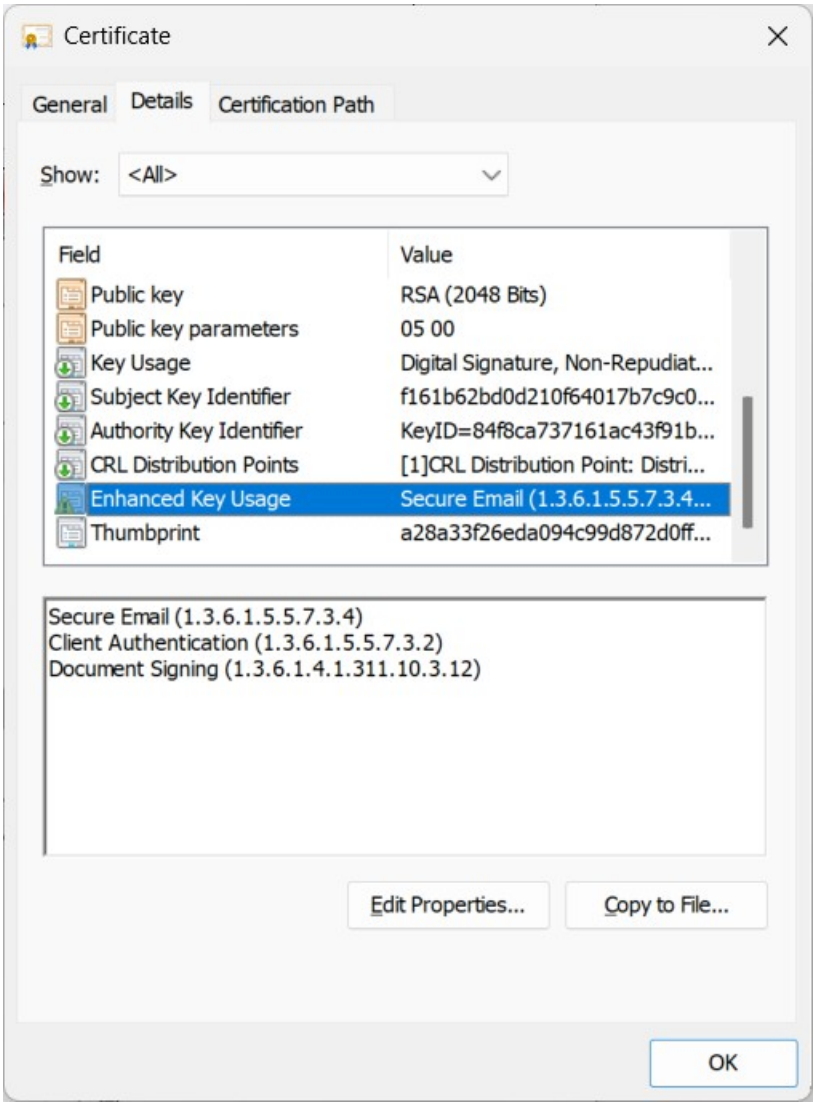
```
cert.Extensions.AddEnhancedKeyUsage(new
System.Security.Cryptography.Oid("1.2.3.4.5.6.7.8.9.10.11"));
```

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

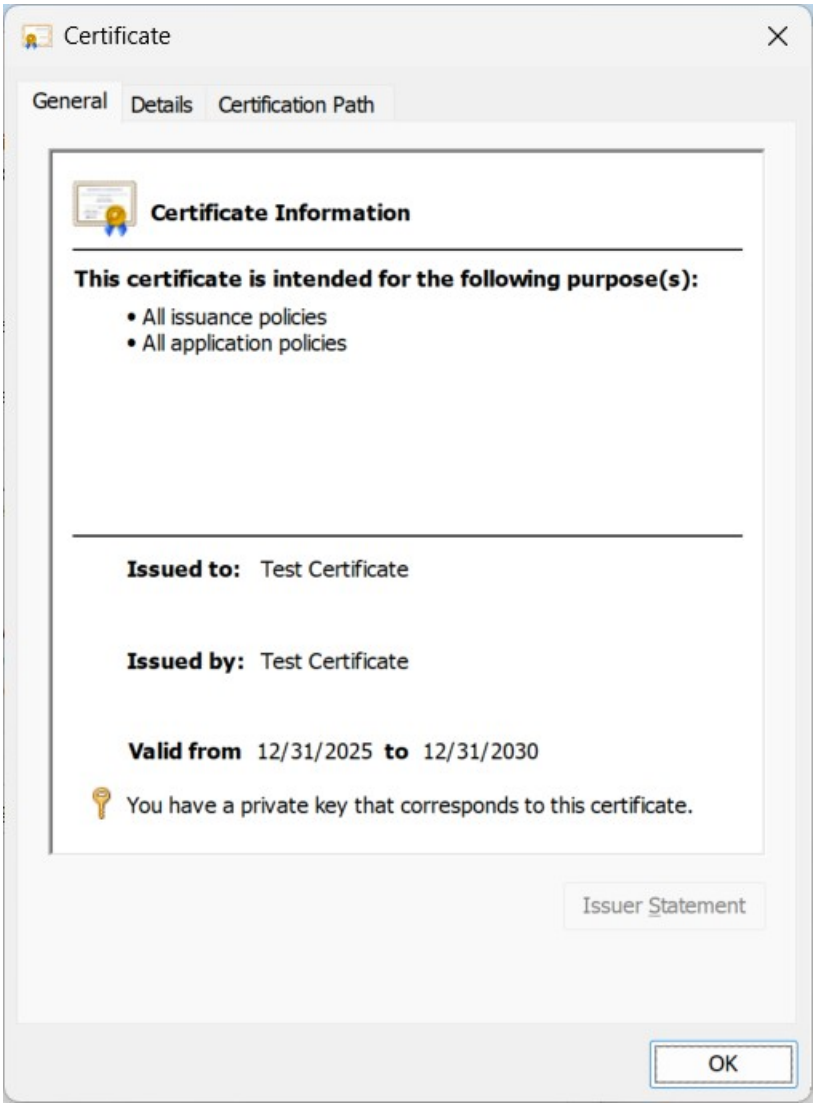
Issue a Self-signed Digital Certificate

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:

```
using SignLib.Certificates;

X509CertificateGenerator cert = new X509CertificateGenerator("serial number");

//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.KeySize2048Bit;
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

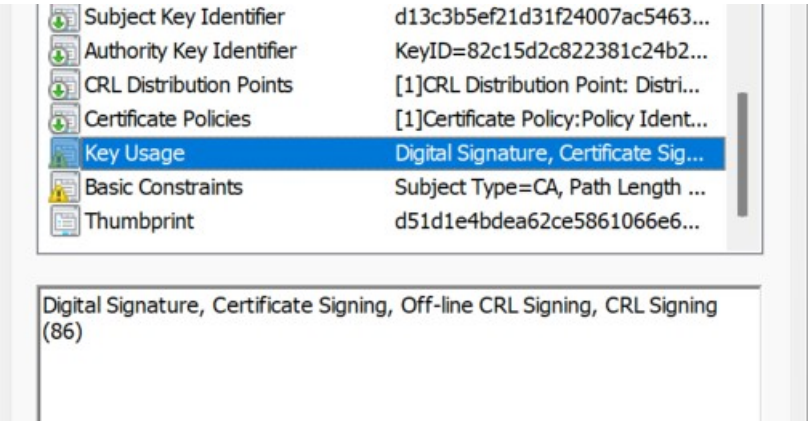
A Root Certificate (CA certificate) is a special type of certificate that can be used to digitally sign other certificates. Also, a Root Certificate can also sign other Root Certificates.

To issue a Root Certificate, use the code below:

```
using SignLib.Certificates;

X509CertificateGenerator cert = new X509CertificateGenerator("serial number");

//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));
```

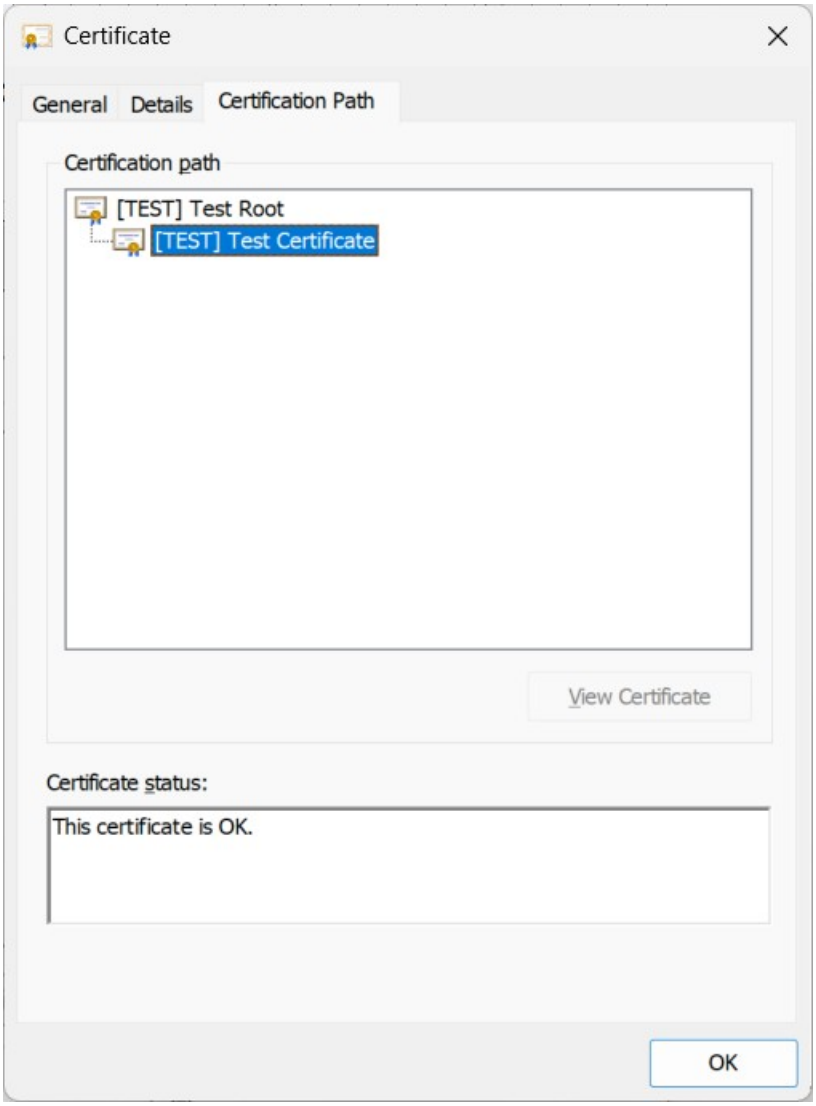
Note that creating a Root certificate is very similar with creating a self signed certificate. The only main difference is on the second parameter of *GenerateCertificate()* method that must be set to *true*.

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

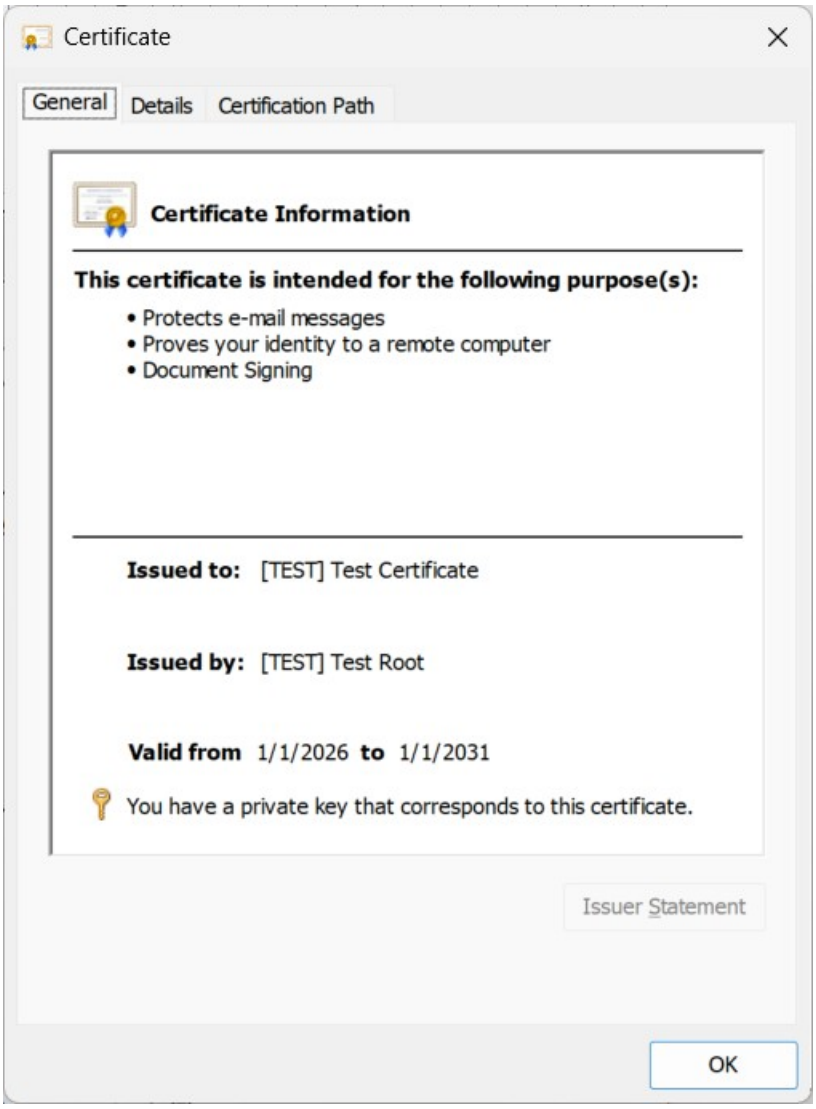
In some cases, is necessary to issue certificates for an entire organization. On this scenario you have two options:

- 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:

```
using SignLib.Certificates;

X509CertificateGenerator root = new X509CertificateGenerator("serial number");

//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("serial number");

//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.KeySize2048Bit;
cert.SignatureAlgorithm = SignatureAlgorithm.SHA256WithRSA;

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** can be accessed using the command: *Start – Run – certmgr.msc* (for Current User) *or certlm.msc* (for Local Machine).

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 a cryptographic smart card (like SafeNet eToken) or on a HSM (Hardware Security Module) like Luna.

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

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).

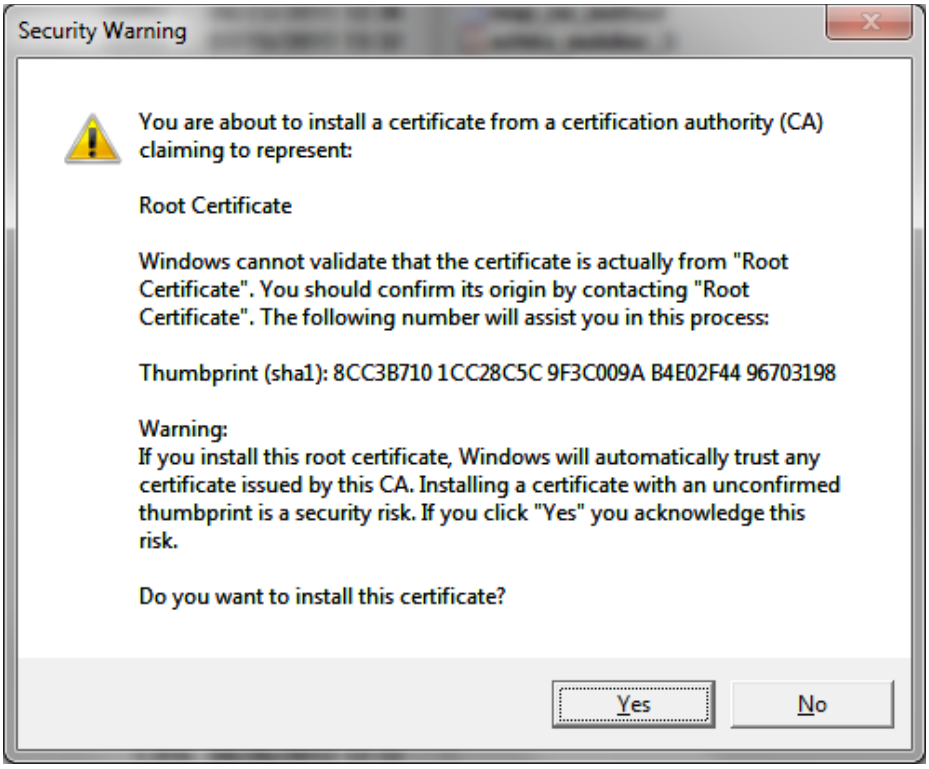
In order to install the certificate, follow this steps:

- 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.

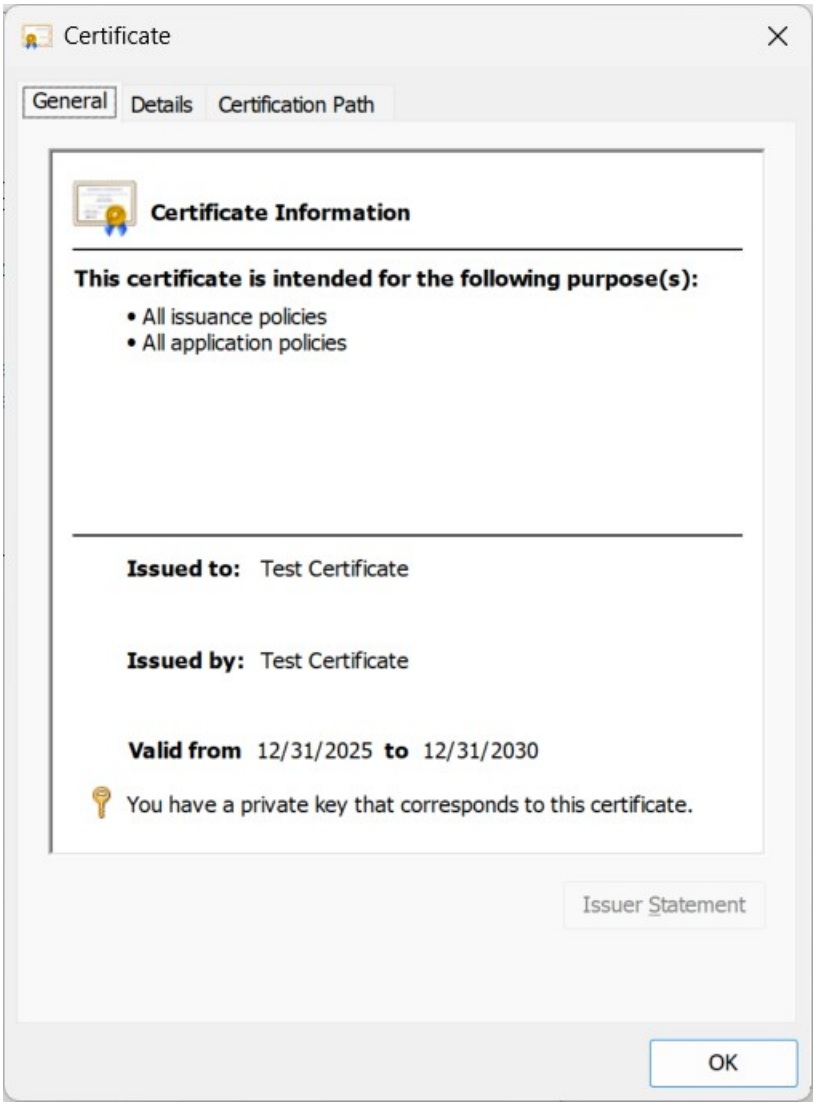
The time digital signature certificate profile will look like this:
- It is recommended to be issued by a Root Certificate (not self signed certificate).
- Use RSA 2048
- Key Usage: Digital Signature.
- Extended Key Usage - add necessary extensions
- Expiration date: 2 years or more.

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
using SignLib.Certificates;

X509CertificateGenerator root = new X509CertificateGenerator("serial number");

//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("serial number");

//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.SHA256WithRSA;
//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.

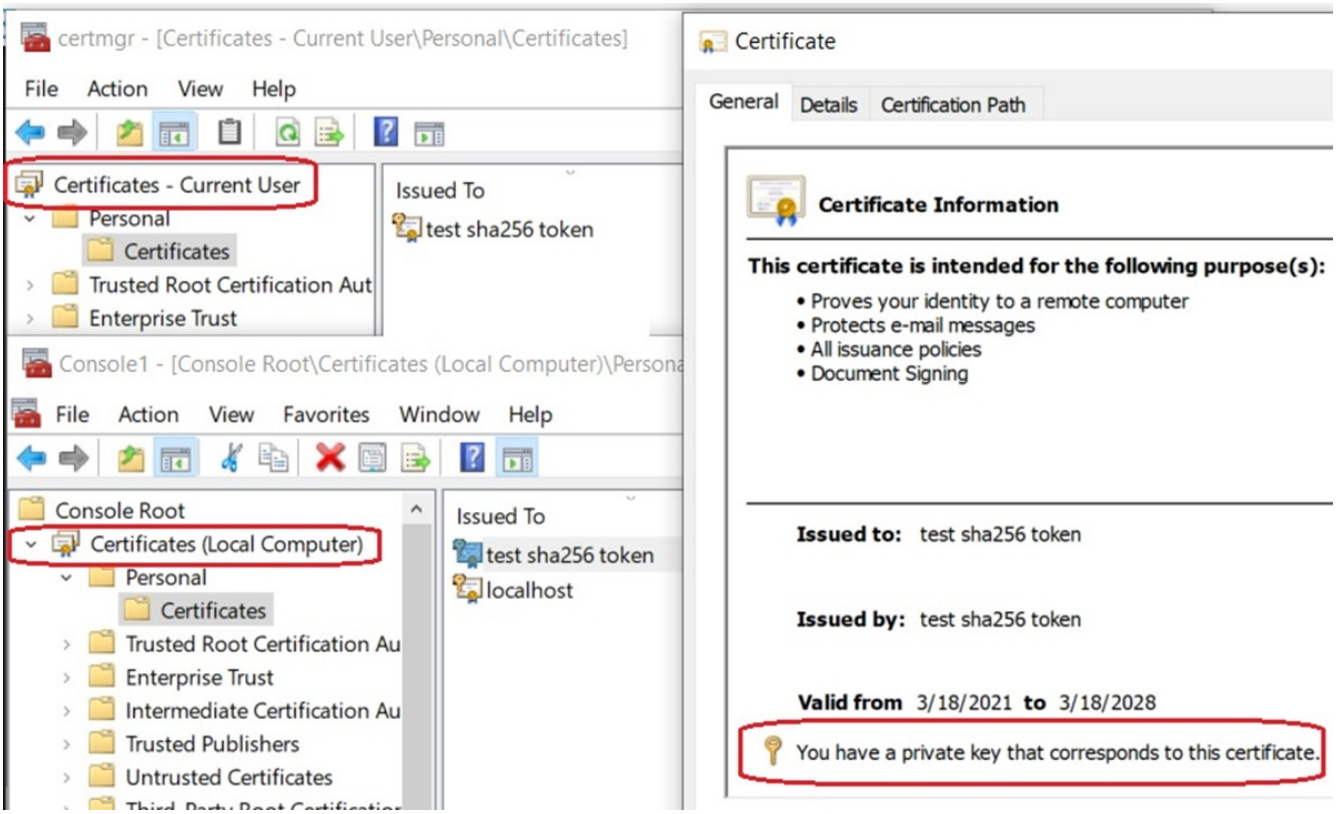


Copying certificates from Current User to Local Computer

Because IIS cannot access Current User Certificate Store, you can make available the same certificate on Local Computer Certificate Store, as below:

- Open the Current User Certificate Store (Start-Run-certmgr.msc) and export the public part of the certificate to d:\certificate.cer
- Open d:\certificate.cer and copy to Notepad (or other location) the certificate serial number (e.g. 4e54c00056bbbff410)
- Open Command Prompt with Administrator rights
- Run the command: certutil -addstore -f "My" "d:\certificate.cer" to import the certificate into Local Computer – Personal account. After this command, the certificate is imported without the reference to the private key
- Run the command certutil -repairstore my "4e54c00056bbbff410" and enter the smart card/HSM password if it is required. After this command the certificate is correctly imported and the reference to the private key is established.

- Now the certificate is imported to Local Computer Certificate Store and ready to be used.



Final Notes

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