NDigiDoc Programmer's Guide

Joosep Ilves

Contents

Version information

Date	Version	Changes
2012	1.0	Document published
05.03.2015	2.0	Updated with information about using direct encryption/decryption vs using intermediary DDOC containers. Using compression during encryption is deprecated. Added URL's to documentation and GitHub.

1. Overview

NDigiDoc is a .NET library for creating and decrypting CDoc-s – the subset of XML-ENC's standard defined by W3C.

CDoc format's specification document can be downloaded from http://id.ee/public/SK-CDOC-1.0-20120625_EN.pdf. Additional information about the format can be found from http://id.ee/?lang=en&id=35780#cdoc.

NDigiDocUtility is command line utility which provides an interface between the user and the NDigiDoc library.

The precompiled NDigiDoc libraries requires .NET 3.5 framework (client profile) or newer. The restriction on 3.5 .NET framework is enforced by the dependency on Microsoft's Cryptography Next Generation (CNG) API.

Currently NOT supported:

- Operations regarding digitally signed documents (DigiDocs)
- Crypto operations via hardware

Development of the NDigiDoc library can also be monitored in GitHub environment: https://github.com/open-eid/ndigidoc

2. Dependencies

- Required – NDigiDoc won't function without the listed dependency. The responsability to resolve the given reference lies on the user of this library.
- Optional – NDigiDoc uses the listed dependency over a proxy. If you don't intent to use the functionality provided by the .dll, then there is no need to reference it.
- Integrated – Parts used from the .dll are already integrated.

Required:

.NET framework 3.5 or above - …

Security.Cryptography.dll – Security.Cryptography.dll provides a new set of algorithm implementations to augment the built in .NET framework supported algorithms. It also provides some APIs to extend the existing framework cryptography APIs. All of the CNG APIs provided in this library require Windows Vista or greater to run. AuthenticatedAesCng additionally requires Windows Vista SP1 or greater. The library itself is built upon the .NET Framework version 3.5.

Optional:

log4net.dll – Port of the popular Java's logging library, log4j. Only two levels are used – DEBUG and ERROR. The former provides bits of data about specific operations, the latter marks exceptions before rethrow.

Integrated:

Ionic.Zlib.dll – More generally known as the **"DotNetZip Library"**. The library is a toolset for manipulating zip files and folders. NDigiDoc has an integrated subset of this library for lossless data compression. NDigiDoc uses the DotNetZib library for compressing and decompressing the payload during CDoc's encrypt and decrypt operations.

Since version 3.9 of the library, compression functionality during CDoc's encryption is deprecated. If the data has been compressed nevertheless, then decompression is used during decryption.

The algorithm used by the ZLIB format is DEFLATE . ZLIB is defined in RFC 1950. While the standard .NET library does implement a DeflateStream that produces a raw DEFLATE

bytestream, it does not provide anything that produces or consumes ZLIB. The same ZlibStream is used by JDigiDoc and CDigiDoc.

3. Compiling the source code

Assuming you use Visual Studio 2008 express and above, this is a matter of opening the solution and rebuilding the projects. You might be able to open your project using an earlier version of Visual Studio, but it's not tested for.

If you try to open the solution with Visual Studio 2008, but receive error messages regarding the solution being built by a newer version of Visual Studio, editing the following (Usually the first line) in .sln file will fix the problem.

Microsoft Visual Studio Solution File, Format Version 11.00

To

Microsoft Visual Studio Solution File, Format Version 10.00

Special attention is required on what .NET framework you target. Anything below .NET 3.5 client profile is excluded by the dependency on Microsofts CNG provider.

4. Using the NDigiDoc library

4.1 Encrypting a CDoc

4.1.1 Creating CDoc and adding recipients

```
var cdoc = new NDigiDoc.CDoc();

// You have to specify at least 1 recipient for your CDoc.
cdoc.Recipients.Add(new X509Certificate2("C:\recipient.pfx")); // Standard .NET class
cdoc.Recipients.Add(new X509Certificate2("C:\recipient2.pem"));
```

NB! When encrypting a document for an individual person then encryption should be done for the authentication certificates on all the recipient's valid identity tokens, i.e. if the recipient has a valid ID-card and Digi-ID card then encryption should be done for the certificates on both of the tokens. Mobile-ID authentication certificate should not be used for encryption as decryption with Mobile-ID is not possible.

4.1.2 Adding encryption properties

The encrypted CDOC document can contain a number of <EncryptionProperty> elements that can be used to store various meta-data.

The default encryption properties can be automatically generated by the library when the following variable is set to true:

```
cdoc.AutoGenerateEncryptionProperties = true;
```

It is possible to use temporary intermediary DDOC container for encapsulating the original files to be encrypted. In this case, the original files need to be placed inside an unsigned DDOC container (with a DigiDoc library that supports DDOC format) and then encrypted. **Note:** using intermediary DDOC containers is generally not recommended but needed for compatibility with DigiDoc3 Crypto 3.8 and earlier versions.

When using the intermediary DDOC container then the encryption property "orig_file" must be specified as follows:

- there must be one “orig_file” property for every data file (<DataFile> element) in the DDOC container
- each property must be in the following format: <file-name>|<size-in-bytes>|<mime-type>|<DataFile-ID-in-DDOC>

For example, do as follows:

cdoc.AutoGenerateEncryptionProperties = false;

cdoc.EncryptionProperties.Add(CDoc.ENC_PROP_ORIG_FILE, yourProperty);

cdoc.EncryptionProperties.Add(..

Note that also the <EncryptedData> element’s “Mimetype“ attribute value must be set to „http://www.sk.ee/DigiDoc/v1.3.0/digidoc.xsd“ but this is done automatically by the library when the data file to be encrypted has .ddoc extension.

4.1.3 Encrypting

// Calling Encrypt() updates the CDoc’s ‘Content’ property, which is of type XDocument.

// CDoc created.

cdoc.Encrypt(“Your-data-from-whatever-source, I intend to encrypt this string”);

Anything more specific/advanced is easily achievable by LINQ to XML via the ’Content’ property.

4.2 Decrypting a CDoc

CDoc cdoc = CDoc.LoadCDoc(.XDocument.Load("C:\\Cdoc.cdoc"));

string decrypted = cdoc.Decrypt(new X509Certificate2(“c:\\MyCert.p12”, “password”));

Note: when decrypting files then it should be taken into account sometimes the data file that has been encrypted is placed inside **a temporary DDOC container before encryption**. In this case, it is also necessary to extract the original data file(s) from DigiDoc container after decryption (with a DigiDoc library that supports DDOC format). Using intermediary DDOC

containers is generally not recommended but needed for compatibility with DigiDoc3 Crypto 3.8 and earlier versions.

It is possible to detect if a temporary DDOC container has been used in the following way:

- <EncryptedData> element's "Mimetype" attribute value is „http://www.sk.ee/DigiDoc/v1.3.0/digidoc.xsd".
- There are one or more EncryptionProperty Name="orig_file"> elements in the following format: <file-name>|<size-in-bytes>|<mime-type>|<DataFile-ID-in-DDOC>
- In other cases, it can be assumed that the data has been encrypted directly, without the temporary DDOC container.

5. NDigiDoc Utility

The NDigiDoc library ships with a command line utility, which acts as an interface between the user and the library. NDigiDoc command syntax is strictly based on JDigiDoc. If a command is NDigiDoc specific, it is marked correspondingly.

Run the executable without arguments for syntax info and examples eq.

.\NDigiDocUtility.exe

Standard syntax: (Comptable between different platform utilities)

Cryptography commands:

-cdoc-decrypt-pkcs12 <cert-uri> <cert-password> <cert-type> <decrypted-file-uri>

-cdoc-encrypt <file-input-uri> <file-output-uri>

-cdoc-in <cdoc-input-uri>

-cdoc-recipient <<recipient-cert-uri> <recipient-cert-uri> … >

NDigiDoc specific commands:

-cdoc-recipient <<cert-uri>?<password> < cert-uri>?<password> ...>

-log4net-xmlconfig-in <log4net-xml-config-uri>

See below for examples:

Example1: Decrypt CDoc and specify logger settings

-cdoc-in C:\toDecrypt.cdoc –cdoc-decrypt-pkcs12 C:\MyCert.p12 1234 PKCS12 decrypted.txt –log4net-xmlconfig-in log4n.xml

Example2: Encrypt CDoc

-cdoc-recipient RecipientCert.pem –cdoc-encrypt tocdoc.txt newcdoc.cdoc

Example3: Encrypt CDoc. The public key resides in a password protected cert store

-cdoc-recipient C:\CertStore.pfx?password123 –cdoc-encrypt tocdoc.txt newcdoc.cdoc