**NDigiDoc Programmers Guide**

Joosep Ilves

Contents

[1. Overview 3](#_Toc312922932)

[2. Dependencies 4](#_Toc312922933)

[3. Compiling the source code 6](#_Toc312922934)

[4. Using the NDigiDoc library 7](#_Toc312922935)

[4.1 Creating a CDoc 7](#_Toc312922936)

[4.2 Decrypting a CDoc 8](#_Toc312922937)

[5. NDigiDoc Utility 9](#_Toc312922938)

# Overview

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

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

# 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.

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.

# 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.

# Using the NDigiDoc library

## Creating a CDoc

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”));*

// 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”);*

You might not want to automatically generate EncryptionProperties.

*cdoc.AutoGenerateEncryptionProperties = false;*

*cdoc.EncryptionProperties.Add(CDoc.ENC\_PROP\_ORIG\_FILE, yourProperty);*

*cdoc.EncryptionProperties.Add( ..*

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”));

# 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**