|  |
| --- |
| Peter Curran |
| OSCA Documentation V1.0 |
| The Offline Simple CA |
|  |
| **Peter Curran** |
| **2/7/2013** |

|  |
| --- |
| An overview of OSCA, including explanations of use cases and documentation for the API. |

Contents

[Licenses 4](#_Toc355910225)

[OSCA 4](#_Toc355910226)

[Bouncy Castle 4](#_Toc355910227)

[Background 5](#_Toc355910228)

[Overview 6](#_Toc355910229)

[OSCA Class Library 7](#_Toc355910230)

[Basic Principles 7](#_Toc355910231)

[CA Configuration File 7](#_Toc355910232)

[Profiles 7](#_Toc355910233)

[FIPS Mode vs BC Mode 8](#_Toc355910234)

[Bouncy Castle Crypto 8](#_Toc355910235)

[System Crypto 8](#_Toc355910236)

[Database 9](#_Toc355910237)

[Logging 9](#_Toc355910238)

[API Description 10](#_Toc355910239)

[OSCA.Offline 10](#_Toc355910240)

[Classes 10](#_Toc355910241)

[Enumeration Type Documentation 10](#_Toc355910242)

[Class Documentation 12](#_Toc355910243)

[OSCA.Offline.CAConfig Struct Reference 12](#_Toc355910244)

[Public Attributes 12](#_Toc355910245)

[Detailed Description 12](#_Toc355910246)

[Member Data Documentation 13](#_Toc355910247)

[OSCA.Offline.fipsCA Class Reference 15](#_Toc355910248)

[Public Member Functions 15](#_Toc355910249)

[Protected Member Functions 15](#_Toc355910250)

[Additional Inherited Members 15](#_Toc355910251)

[Detailed Description 15](#_Toc355910252)

[Constructor & Destructor Documentation 15](#_Toc355910253)

[Member Function Documentation 15](#_Toc355910254)

[OSCA.Offline.simpleCA Class Reference 17](#_Toc355910255)

[Public Member Functions 17](#_Toc355910256)

[Protected Member Functions 17](#_Toc355910257)

[Protected Attributes 17](#_Toc355910258)

[Properties 18](#_Toc355910259)

[Detailed Description 18](#_Toc355910260)

[Constructor & Destructor Documentation 18](#_Toc355910261)

[Member Function Documentation 19](#_Toc355910262)

[Property Documentation 21](#_Toc355910263)

[Use Cases 1](#_Toc355910264)

[Use Case #1 – An Off-line CA 1](#_Toc355910265)

# Licenses

## OSCA

Copyright © 2010 - 2013 Peter Curran (peter@currans.eu). All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY PETER CURRAN "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL PETER CURRAN OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are those of the author alone.

## Bouncy Castle

Copyright © 2000 - 2012 The Legion Of The Bouncy Castle (http://www.bouncycastle.org)

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

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

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

# Background

I have worked in the field of PKI management for a number of years, over that time I have become increasingly frustrated by many of the PKI products that I have worked with. Two key areas have annoyed me enough to develop OSCA:

1. There is no easy CA system that truly supports an off-line Root CA – most require a complicated directory and/or database setup that is an administrative (and reliability) pain, with no obvious benefit. A Root CA will typically only ever issue a handful of certificates in its lifetime – mostly it is used to issue CRLs: I don’t need a complex database to hold this information. For security, it is preferable that a Root CA is offline: I don’t need a complex directory service that no one can ever access – I don’t want to make the CA online just so that I can publish CRLs to an enterprise or system directory.
2. There are many circumstances when I want to maintain a simple certificate-issuing service to support a closed community, for example SSL/TLS certificates for all the switches and routers in my network. Do I really need the overhead of a full blown PKI, or full blown CA products to do this? In many cases, the ‘cut-down’ versions of products like Microsoft’s ADCS (i.e. operating in stand-alone mode) do not give me the flexibility of defining the certificate profiles that I want, or simply do not support the class of certificates that I want to issue (e.g. IPsec authentication certificates).

The Offline Simple CA (OSCA) is my answer to these two requirements. It is made possible by the excellent Bouncy Castle cryptographic library and its comprehensive support for ASN.1, as well as numerous tools for creating X.509 certificates and CRLs – not to mention tools for working with PKCS standards. OSCA is written in C#, using the Microsoft Visual Studio 2010 IDE, and is targeted on Windows systems (but will probably work with Mono).

The OSCA system is delivered as a class library (DLL). You need to provide your own user interface, but the basic *OSCAcli.exe* provides a sample implementation of a command line system that uses OSCA. (You may also be interested in *OSCAmmc*, which provides a turn-key CA solution as an MMC v3.0 snap-in).

Peter Curran  
February 2013

# Overview

OSCA is a CLR-compatible DLL that provides a class library, which supports the creation and operation of a CA. The key features/limitations of the CA are:

* Supports self-signed CA creation.
* Supports subordinated CA creation (generates a PKCS#10 request).
  + Currently, there is limited support for imposing policies on subordinate CAs.
* Uses the Bouncy Castle crypto library for all cryptographic functions by default.
* Optionally, operation in a FIPS 140-2 compatible mode is provided by using the .Net (system) libraries for key storage and cryptographic functions.
  + Work is underway to add support for cryptographic hardware (e.g., HSMs and smartcards) that offer a PKCS#11 interface.
* Accepts certificate requests as PKCS#10 only.
* Optional profile function supports full definition for certificate profiles.
* CRLs are created on demand.
  + There is no automated CRL creation within the library.
* Database and CA configuration use signed XML files.

OSCA can be viewed as a core CA function with a number of support functions that can be used for a variety of purposes, either directly relating to CA operation, or to allow the creation of utilities (such as PKCS#10 request generators and parsers).

# OSCA Class Library

## Basic Principles

The first step in using OSCA is to create a CA: OSCA supports Root (self-signed) CA and Subordinate CA types. A Subordinate CA may have its certificate signed by another OSCA CA instance (this must exist within the same execution context, as the instance is passed into the factory). Alternatively, the factory will generate a PKCS #10 request that can be passed to any other CA for processing; subsequently the creation process is completed by supplying the CA certificate.

Creation of an OSCA CA results in the production of a CA configuration file and a CA database file. These are both XML files, and stored in a CA directory that is specified when the CA is created. These files are signed by the CA to protect them from accidental or deliberate modification or tampering.

Once a CA has been created, it may be instantiated (‘started’) by specifying the CA configuration file and a password to decrypt/access the key material.

The CA instance supports a variety of methods to retrieve status information (e.g. last certificate or CRL serial number) as well as methods that perform fundamental CA functions:

* Issue a certificate
* Revoke a certificate
* Issue a CRL
* Get current CRL

The CA may be ‘stopped’ by deleting the instance.

## CA Configuration File

OSCA configuration is stored in a CA configuration file (*CAConfig.xml*). This file is read when the CA object is instantiated and provides the CA with basic information about its operating environment, as well as operating parameters such as the serial number of the last certificate and CRL created.

To protect from tampering (e.g., interfering with serial numbers) the configuration file is signed using an XML signature.

## Profiles

When a certificate is issued, the caller has the option of using the certificate extensions specification that is contained in the request, or alternatively specifying the certificate format by use of an OSCA Profile. The purpose of an OSCA Profile is to provide a certificate template that defines the existence, and content, of X.509 extensions.

When an OSCA Profile is used, all of the content of the certificate request are ignored, apart from the public key and naming information:

* The *Subject* field from the request is used verbatim.
* If the *subjectAlternativeName* extension is present, then this will be used verbatim.

OSCA Profiles are stored in XML files, within the CA directory structure. They can be created manually (by use of a text editor) or programmatically via the API. In the latter case, they may be created from an existing extension (e.g. from an existing certificate), or alternatively by specifying the values using the API.

The profiles mechanism is designed for use in a variety of scenarios. For example, the API includes a generic PKCS#10 parser that can be used to interpret extensions contained in a certificate request, to allow analysis by an RA function. Alternatively, it is possible to create a PKCS#10 request file containing the fields (including extensions) contained in a profile: The resultant request could then be processed by any CA that handles PKCS#10 requests – including OSCA itself.

Whilst OSCA consumes profiles, and the library functions that support profiles can be used to create them programmatically, some form of user interface is required to create and edit profiles. The *OSCAmmc* GUI provides a simple example of this process.

## FIPS Mode vs BC Mode

When a CA is created the cryptographic library is specified. The options are to use the Bouncy Castle crypto library (*FIPS140 = false*) or the .Net (*System.Security.Cryptography*) crypto library (*FIPS140 = true*). There are pros and cons to both approaches:

### Bouncy Castle Crypto

The Bouncy Castle crypto library is implemented in C#, and it is relatively slow (particularly for public key operations) compared to the System crypto libraries. Whilst the BC library offers comprehensive functionality (far more so than the System library), there are security concerns over the protection of key material – all of which is held in the process memory and could be accessed readily with a debugger or via a memory dump.

From the viewpoint of OSCA, the main benefit gained from the BC library is that it provides true portability of OSCA CAs. The CA key (and certificate) is held in an encrypted PKCS#12 object that is stored within the CA configuration file – an XML file. Simply copying the CA directory structure, with its files, provides a complete backup of the CA that can be readily moved between computers.

### System Crypto

The System crypto libraries (*System.Security.Cryptography*) are managed code wrappers around the underlying operating system FIPS 140 validated cryptographic module. For Windows 7/2008 this validation is against FIPS 140-2. OSCA is designed to use the System crypto in a manner that is within the FIPS 140 security target – as such it can claim to be a ‘FIPS 140 System’. In the future it is intended to expand the CSP support to include HSMs and smartcards (either via their PKCS#11 *cryptoki* library, or using the native CAPI).

The main disadvantage of using the System crypto is that key material is stored in a container that belongs to the user. This means that backups, and general portability, is constrained as the key material has to be exported in an encrypted form (to comply with the FIPS 140 requirements) and imported if the system is moved to a different computer, or user. To enable this, the *fipsCA* implementation supports exporting/importing of PKCS#12 files – but the key has to be exported to memory to allow its incorporation into the PKCS#12 file (hence, there is a risk to the key material).

## Database

OSCA uses a simple XML-based database (*CADatabase.xml*) to hold information about certificates, and their revocation status. This provides a basic audit function (in that the details of each certificate creation – date/time, profile used, original request, etc. are captured) but is used by OSCA for the creation of CRLS. Each certificate issued has an entry in the database; the entry includes the ‘revocation status’ of the certificate – either current, expired or revoked. Revoked certificates have the date/time of revocation and the revocation reason recorded in the database. When creating a CRL, OSCA simply iterates through the database and creates a CRL entry for each revoked certificate (unless the original certificate has expired).

To protect from tampering (e.g., resetting the status of a revoked certificate) the database is signed using an XML signature.

## Logging

In addition to the audit information embedded in the database entries, OSCA also maintains a log (*CALog.xml*) of significant events (‘starting/stopping’ the CA, certificate issuance, CRL creation, etc.). A simple command line audit tool is provided (OSCAaudit) that creates a basic audit report – this can either provide the life history of a single certificate, or all certificates.

To protect from tampering (e.g., modifying or deleting log entries) the logfile is signed using an XML signature.

# API Description

## OSCA.Offline

### Classes

struct CAConfig

Object describing the setup parameters for a CA

class CaFactory

CA Factory

class Database

Class to support access to the **OSCA** database

class **fipsCA**

A basic implementation of a Certification Authority using FIPS 140 (System) crypto

class simpleCA

A basic implementation of a Certification Authority

class Utility

* Collection of utility functions Enumerations

enum **X509ver** { **V1** = 1, **V3** = 3 }

X.509 certificate version

enum CA\_Type { simpleCA, fipsCA }

Type of CA

enum CA\_Profile { RootCA, SubCA }

* Profile of CA

### Enumeration Type Documentation

enum OSCA.Offline.CA\_Profile

Profile of CA

**Enumerator**

***RootCA*** Root CA

***SubCA*** Subordinate CA

enum OSCA.Offline.CA\_Type

Type of CA

**Enumerator**

***simpleCA*** **OSCA** Simple CA (BC crypto)

***fipsCA*** **OSCA** FIPS CA (System crypto)

Definition at line 49 of file CAConfig.cs.

enum OSCA.Offline.X509ver

X.509 certificate version

**Enumerator**

***V1*** Version 1

***V3*** Version 3

# Class Documentation

## OSCA.Offline.CAConfig Struct Reference

Object describing the setup parameters for a CA

### Public Attributes

string **name**

*Friendly name for the CA*

X509Name **DN**

*X.501 distinguished name of the CA*

CA\_Profile profile

*Type of CA (rootCA|subCA)*

string profileFile

*Location of the profile XML file*

string **pkAlgo**

*Public key alogorithm used by the CA*

int pkSize

*Size if the public key (eg RSA modulus)*

string **sigAlgo**

*Signature algorithm used by the CA*

int keyUsage

*Key usage extension value*

X509ver version

*X509 version for the CA certificate*

int **life**

*Lifetime of the CA certificate*

string **units**

*Units used by the lifetime value*

double crlInterval

*Frequency of publishing CRLs*

CA\_Type caType

*Type of CA (simplCA|fipsCA)*

bool FIPS140

*FIPS 140 mode*

string location

*Root directory of CA*

string password

*Password for accessing key material*

### Detailed Description

Object describing the setup parameters for a CA

### Member Data Documentation

#### CA\_Type OSCA.Offline.CAConfig.caType

Type of CA (simplCA|fipsCA)

#### double OSCA.Offline.CAConfig.crlInterval

Frequency of publishing CRLs

#### X509Name OSCA.Offline.CAConfig.DN

X.501 distinguished name of the CA

#### bool OSCA.Offline.CAConfig.FIPS140

FIPS 140 mode

Definition at line 136 of file CAConfig.cs.

#### int OSCA.Offline.CAConfig.keyUsage

Key usage extension value

Definition at line 112 of file CAConfig.cs.

#### int OSCA.Offline.CAConfig.life

Lifetime of the CA certificate

Definition at line 120 of file CAConfig.cs.

#### string OSCA.Offline.CAConfig.location

Root directory of CA

Definition at line 140 of file CAConfig.cs.

#### string OSCA.Offline.CAConfig.name

Friendly name for the CA

Definition at line 84 of file CAConfig.cs.

#### string OSCA.Offline.CAConfig.password

Password for accessing key material

Definition at line 144 of file CAConfig.cs.

#### string OSCA.Offline.CAConfig.pkAlgo

Public key alogorithm used by the CA

Definition at line 100 of file CAConfig.cs.

#### int OSCA.Offline.CAConfig.pkSize

Size if the public key (eg RSA modulus)

Definition at line 104 of file CAConfig.cs.

#### CA\_Profile OSCA.Offline.CAConfig.profile

Type of CA (rootCA|subCA)

Definition at line 92 of file CAConfig.cs.

#### string OSCA.Offline.CAConfig.profileFile

Location of the profile XML file

Definition at line 96 of file CAConfig.cs.

#### string OSCA.Offline.CAConfig.sigAlgo

Signature algorithm used by the CA

Definition at line 108 of file CAConfig.cs.

#### string OSCA.Offline.CAConfig.units

Units used by the lifetime value

Definition at line 124 of file CAConfig.cs.

#### X509ver OSCA.Offline.CAConfig.version

X509 version for the CA certificate

Definition at line 116 of file CAConfig.cs.

#### The documentation for this struct was generated from the following file:

* C:/Users/Peter/Documents/Visual Studio 2010/Projects/OfflineSimpleCA/OSCA/Offline/CAConfig.cs

## OSCA.Offline.fipsCA Class Reference

A basic implementation of a Certification Authority using FIPS 140 (System) crypto

Inheritance diagram for OSCA.Offline.fipsCA:

### Public Member Functions

**fipsCA** (string ConfigFile)

*Construct a CA object*

### Protected Member Functions

override X509Certificate **generate** (ICertGen gen)

*Generate a certificate*

override X509Certificate **generate** (ICertGen gen, X509Extensions ext)

*Generate a certificate*

### Additional Inherited Members

### Detailed Description

A basic implementation of a Certification Authority using FIPS 140 (System) crypto

### Constructor & Destructor Documentation

#### OSCA.Offline.fipsCA.fipsCA (string ConfigFile)

Construct a CA object

##### Parameters:

|  |  |
| --- | --- |
| *ConfigFile* | Full pathname to config file |

### Member Function Documentation

#### override X509Certificate OSCA.Offline.fipsCA.generate (ICertGen gen)[protected], [virtual]

Generate a certificate

##### Parameters:

|  |  |
| --- | --- |
| *gen* |  |

##### Returns:

Reimplemented from **OSCA.Offline.simpleCA** (*p.pagenum*).

Definition at line 79 of file fipsCA.cs.

#### override X509Certificate OSCA.Offline.fipsCA.generate (ICertGen gen, X509Extensions ext)[protected], [virtual]

Generate a certificate

##### Parameters:

|  |  |
| --- | --- |
| *gen* |  |
| *ext* |  |

##### Returns:

Reimplemented from **OSCA.Offline.simpleCA** (*p.pagenum*).

Definition at line 90 of file fipsCA.cs.

#### The documentation for this class was generated from the following file:

* C:/Users/Peter/Documents/Visual Studio 2010/Projects/OfflineSimpleCA/OSCA/Offline/fipsCA.cs

## OSCA.Offline.simpleCA Class Reference

A basic implementation of a Certification Authority

Inheritance diagram for OSCA.Offline.simpleCA:

### Public Member Functions

**simpleCA** (string ConfigFile, string Password)

*Construct a CA object*

X509Certificate **IssueCertificate** (byte[] Request, string ProfileFile)

*Issue a certificate based on a PKCS#10 certificate request and* ***OSCA*** *profile*

X509Certificate **IssueCertificate** (byte[] Request, ValidityPeriod Validity)

*Issue a certificate based on a PKCS#10 certificate request and validity period*

X509Certificate **IssueCertificate** (Pkcs10CertificationRequest Request, string ProfileFile)

*Issue a certificate based on a PKCS#10 certificate request object and* ***OSCA*** *profile*

X509Certificate **IssueCertificate** (Pkcs10CertificationRequest Request, ValidityPeriod Validity)

*Issue a certificate based on a PKCS#10 certificate request object and validity period*

string **RevokeCertificate** (X509Certificate **Certificate**, CRLReason Reason)

*Revoke a certificate*

string **IssueCRL** ()

*Issue a CRL (containing all revoked certificates)*

X509Crl **GetCRL** ()

*Retrieve the current CRL*

### Protected Member Functions

simpleCA ()

*Stub constructor to support inheritance*

virtual X509Certificate **generate** (ICertGen gen)

virtual X509Certificate **generate** (ICertGen gen, X509Extensions ext)

### Protected Attributes

string configFile

char[] password

string **name**

string **type**

string dbFileLocation

string profilesLocation

string publicKeyAlgorithm

string publicKeySize

string signatureAlgorithm

bool fips140

string lastSerial

string crlFileLocation

string **lastCRL**

double crlInterval

X509Certificate **certificate**

### Properties

string **LastCRLNumber** [get]

*Serial number of the last CRL issued*

string LastSerialNumber [get]

*Serial number of the last certificate issued*

bool **FIPS140Mode** [get]

*True if this CA is in FIPS 140 mode*

string **CAType** [get]

*CA Type (Root or subCA)*

string **CAName** [get]

*Distinguished Name of the CA*

string PublicKeyAlgorithm [get]

*Algorithm for the CAs key-pair*

string SignatureAlgorithm [get]

*Algorithm for the CAs signature*

X509Certificate **Certificate** [get]

*CA certificate*

### Detailed Description

A basic implementation of a Certification Authority

### Constructor & Destructor Documentation

#### OSCA.Offline.simpleCA.simpleCA (string ConfigFile, string Password)

Construct a CA object

##### Parameters:

|  |  |
| --- | --- |
| *ConfigFile* | Full pathname to config file |
| *Password* | Password for key file |

Definition at line 76 of file simpleCA.cs.

#### OSCA.Offline.simpleCA.simpleCA ()[protected]

Stub constructor to support inheritance

Definition at line 107 of file simpleCA.cs.

### Member Function Documentation

#### X509Crl OSCA.Offline.simpleCA.GetCRL ()

Retrieve the current CRL

##### Returns:

The current CRL

Definition at line 423 of file simpleCA.cs.

#### X509Certificate OSCA.Offline.simpleCA.IssueCertificate (byte[] Request, string ProfileFile)

Issue a certificate based on a PKCS#10 certificate request and **OSCA** profile

##### Parameters:

|  |  |
| --- | --- |
| *Request* | Encoded PKCS#10 certificate request |
| *ProfileFile* | Full pathname of certificate profile |

##### Returns:

Requested certificate

##### Exceptions:

|  |  |
| --- | --- |
| *ArgumentException* | Invalid signature algorithm in request |

Definition at line 210 of file simpleCA.cs.

#### X509Certificate OSCA.Offline.simpleCA.IssueCertificate (byte[] Request, ValidityPeriod Validity)

Issue a certificate based on a PKCS#10 certificate request and validity period

##### Parameters:

|  |  |
| --- | --- |
| *Request* | DER encoded PKCS#10 certificate request |
| *Validity* | Validity period for certificate |

##### Returns:

Requested certificate

##### Exceptions:

|  |  |
| --- | --- |
| *ArgumentException* | Invalid signature algorithm in request |

Definition at line 223 of file simpleCA.cs.

#### X509Certificate OSCA.Offline.simpleCA.IssueCertificate (Pkcs10CertificationRequest Request, string ProfileFile)

Issue a certificate based on a PKCS#10 certificate request object and **OSCA** profile

##### Parameters:

|  |  |
| --- | --- |
| *Request* | PKCS#10 certificate request |
| *ProfileFle* | Full pathname of certificate profile |

##### Returns:

Requested certificate

##### Exceptions:

|  |  |
| --- | --- |
| *ArgumentException* | Invalid signature algorithm in request |

X509Certificate OSCA.Offline.simpleCA.IssueCertificate (Pkcs10CertificationRequest *Request*, ValidityPeriod *Validity*)

Issue a certificate based on a PKCS#10 certificate request object and validity period

##### Parameters:

|  |  |
| --- | --- |
| *Request* | PKCS#10 certificate request |
| *Validity* | Validity period for certificate |

##### Returns:

Requested certificate

##### Exceptions:

|  |  |
| --- | --- |
| *ArgumentException* | Invalid signature algorithm in request |

string OSCA.Offline.simpleCA.IssueCRL ()

Issue a CRL (containing all revoked certificates)

##### Returns:

string OSCA.Offline.simpleCA.RevokeCertificate (X509Certificate *Certificate*, CRLReason *Reason*)

Revoke a certificate

##### Parameters:

|  |  |
| --- | --- |
| *Certificate* | Certificate to revoke |
| *Reason* | Revocation reason |

##### Returns:

Status of the certificate

##### Exceptions:

|  |  |
| --- | --- |
| *ApplicationException* | Certificate not found |
| *ApplicationException* | Duplicate serial number |
| *ApplicationException* | Certificate is already revoked |

### Property Documentation

string OSCA.Offline.simpleCA.CAName[get]

Distinguished Name of the CA

string OSCA.Offline.simpleCA.CAType[get]

CA Type (Root or subCA)

X509Certificate OSCA.Offline.simpleCA.Certificate[get]

CA certificate

Definition at line 144 of file simpleCA.cs.

bool OSCA.Offline.simpleCA.FIPS140Mode[get]

True if this CA is in FIPS 140 mode

string OSCA.Offline.simpleCA.LastCRLNumber[get]

Serial number of the last CRL issued

string OSCA.Offline.simpleCA.LastSerialNumber[get]

Serial number of the last certificate issued

string OSCA.Offline.simpleCA.PublicKeyAlgorithm[get]

Algorithm for the CAs key-pair

string OSCA.Offline.simpleCA.SignatureAlgorithm[get]

Algorithm for the CAs signature

# Use Cases

## Use Case #1 – An Off-line CA