Create a DDS Secure demo while acting as your own CA

[1 Introduction 2](#_Toc37841321)

[1.1 Files You’ll Be Creating 2](#_Toc37841322)

[1.1.1 Per-Domain artifacts (shared among all applications on the same DDS domain) 2](#_Toc37841323)

[1.1.2 Per-participant artifacts 3](#_Toc37841324)

[1.2 Prerequisites 3](#_Toc37841325)

[1.3 Overview of Adding Security 3](#_Toc37841326)

[1.4 Best Practices for Adding Security 3](#_Toc37841327)

[1.5 Directory Structure Used in the Examples Below 3](#_Toc37841328)

[2 Setup for self-signing: Becoming your own Identity CA 5](#_Toc37841329)

[3 Setup for self-signing: Becoming your own Permissions CA (optional) 7](#_Toc37841330)

[4 Generate an identity for each DDS application 8](#_Toc37841331)

[5 Create and Sign a Domain Governance File 10](#_Toc37841332)

[5.1 Domain Governance File Overview 10](#_Toc37841333)

[5.2 Domain Governance File Sections 10](#_Toc37841334)

[5.3 Signing the Domain Governance File 11](#_Toc37841335)

[6 Create and Sign Permissions files 12](#_Toc37841336)

[7 Files to Deploy 13](#_Toc37841337)

[8 OpenSSL command summary 14](#_Toc37841338)

[8.1 Creating Self-Signed CAs 14](#_Toc37841339)

[8.2 Creating Signed Certificates for Each DDS Application 14](#_Toc37841340)

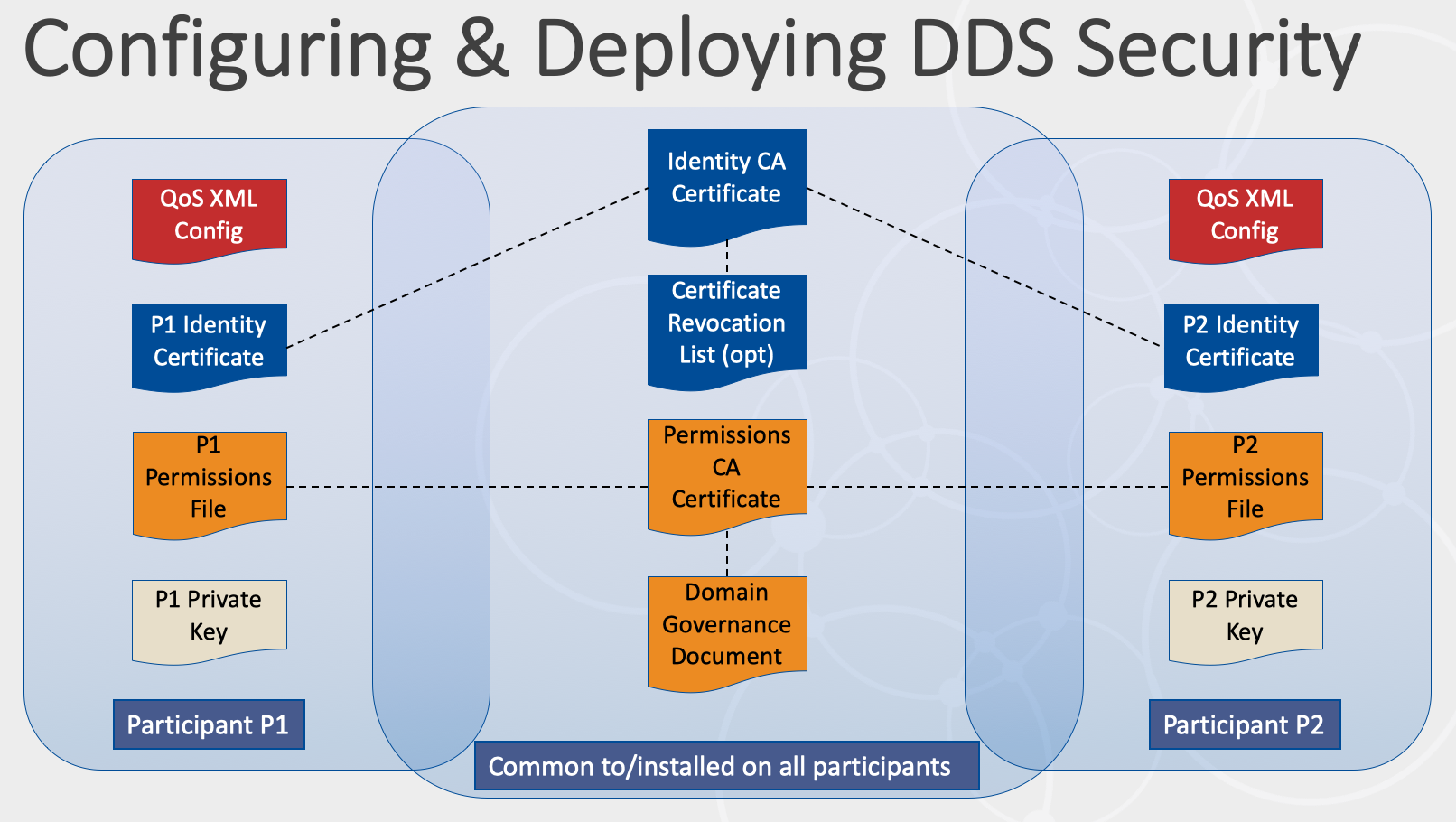
[8.3 Signing Documents with SMIME 14](#_Toc37841341)

# Introduction

Secure a DDS application with DDS Secure is relatively simple. There are no coding changes needed, no new APIs to learn. You simply rebuild your code linking in the DDS Secure libraries, and then provide each application a number of files (artifacts) that are needed at runtime. Some of these you will create, others like certificates and keys are created using a tool like OpenSSL. You’ll need to modify existing QoS configurations to point to these files and to configure new features like Secure logging. Finally, you’ll deploy these new files along with your application. Each of these steps are detailed in the following chapters.

## Files You’ll Be Creating

A DDS Secure application requires the following files to be deployed:



### Per-Domain artifacts (shared among all applications on the same DDS domain)

*You will create one set of the following documents and deploy them to all machines running DDS Secure.*

* Identity CA certificate
* Permissions CA certificate
  + May be the same as the identity CA certificate
* Domain Governance Document
  + XML file you create, signed by the Permissions CA
* (Optional) Certificate Revocation List (CRL)
  + Signed by Identity CA

### Per-participant artifacts

* Private key
* Identity certificate
  + Signed by the identity CA
* Permissions document
  + Signed by the permissions CA
  + Subject name must match participants identity certificate
* QoS Configuration File
  + Points to the other security artifacts

## Prerequisites

Connext DDS Secure and OpenSSL 1.1.1+ have been installed.

## Overview of Adding Security

First, you’ll create the cryptographic files: identity CA files, permission CA files, and individual participant identity files. Then you will create the domain governance file, and the per participant permissions files, and sign them. Finally the XML QoS settings for each application will be created that point to the other files you created.

## Best Practices for Adding Security

* Get your system working properly without security enabled.
* Create a Domain Governance file with permissive settings that allow unauthenticated participants with no access control for topics.
* Link in the Secure libraries, modify the QoS files to enable Security, and verify that your system works.
* Enable access control on the domain. Verify that every Secure application can join and participate in the network.
* Begin enabling read and write access control on individual topics. Ensure that access control settings are in effect by trying to create read and write violations. Check with tools.
* Enable crypto settings for topics. Verify in Wireshark
* Enable crypto settings for meta data and discovery. Check with tools.

## Directory Structure Used in the Examples Below

In the following examples, we will use the Shapes demo and create various security configurations for it. Each configuration will be placed in its own directory. In the same way, you likely create a separate directory for each unique application in your system to store that application’s Secure artifacts and configuration files.

* Below the root directory, there is an “apps” directory.
  + Within the apps directory is a folder for each secured application in your system;
  + The contents of “template” serve as the starting point.
* The “ca” directory houses two directories used in these examples to become your own identity and permissions Certificate Authority.
  + “identity” is for issuing and signing identity certificates.
  + “permissions” is for signing Permissions files, and the Domain Governance file
* Finally, the “governance” directory contains the source XML file for the common Domain Governance shared by all apps in the system.
* Files deployed to every application (like the CA certs and signed Domain Governance file) end up in “apps\deployall”.

# Setup for self-signing: Becoming your own Identity CA

In this section you’ll be working in “\ca\identity”. You will be creating the following files.

|  |  |  |
| --- | --- | --- |
| File | Location | Description |
| openssl.cnf | . | Configuration file for OpenSSL. A template is provided in ca\identity\resource\ |
| identcapriv.pem | .\private | The private key for the Identity CA |
| identca.csr | . | Certificate request used to create CA. Based on the CA private key. Can be deleted once the CA cert if created. |
| identcacert.pem | . | The signed public identity certificate of the CA |

There is a template “openssl.cnf” file included in the sample files for this workbook. For full details on all the fields in openssl.cnf, consult the OpenSSL documentation. For the simple examples in this workbook, change the HOME directory defined on line 4 to match your system’s directory structure (windows may require you to escape ‘\’ characters by using ‘\\’). Also you can edit the entries under the [req\_distinguished\_name] section as appropriate for your organization.

After the above directories and configuration files are created, you have an environment where you can create a CA certificate for your local CA.

**There is a windows batch file that performs all of the following identity CA commands for you called generate.ident.ca.bat but you can also run them individually.**

Before you can create your local CA certificate, you’ll need to create a private key for the CA. Use the following command (Note: all the examples in this document use elliptic curve encryption using the secp256k1 curve).

openssl ecparam -name secp256k1 -genkey -noout -out private\identcapriv.pem

Now you’ll create the actual CA certificate in a two-step process: generate a certificate request (.csr file) using the CA private key, then generate the actual CA certificate.

Generate the certificate request:

openssl req -new -sha256 -key private\identcapriv.pem -out identca.csr -config openssl.cnf

Generate the CA certificate (-text includes ASCII certificate information)

openssl x509 -req -sha256 -days 3650 -in identca.csr -text -signkey private\identcapriv.pem -out identcacert.pem

You can now delete the identca.csr file. Copy the CA certificate to a common location:

copy identcacert.pem ..\..\apps\deployall\

You are now your own Identity CA and can create identity certificates for use by DDS Secure applications.

# Setup for self-signing: Becoming your own Permissions CA (optional)

This step is optional. DDS Secure allows you to have separate CAs to create identity certificates (Identity CA) and to sign permissions and governance files (Permissions CA). You can also use a single CA for both purposes. In this example, we’ll create a separate Permissions CA.

You will be creating the following files:

|  |  |  |
| --- | --- | --- |
| File | Location | Description |
| openssl.cnf | . | Configuration file for OpenSSL. |
| permcapriv.pem | .\private | Make this directory to store your private key |
| permca.csr | . | Certificate request used to create CA. Based on the CA private key. Can be deleted once the CA cert if created. |
| permcacert.pem | . | The signed public permission certificate of the CA |

**There is a windows batch file that performs all of the following identity CA commands for you called generate.perm.ca.bat but you can also run them individually.**

Before you can create your local permissions CA certificate, you’ll need to create a private key for the CA. Use the following command (Note: all the examples in this document use elliptic curve encryption using the secp256k1 curve).

openssl ecparam -name secp256k1 -genkey -noout -out private\permcapriv.pem

Now you’ll create the actual CA certificate in a two-step process: generate a certificate request (.csr file) using the CA private key, then generate the actual CA certificate.

Generate the certificate request:

openssl req -new -sha256 -key private\permcapriv.pem -out permca.csr -config openssl.cnf

Generate the CA certificate (-text includes ASCII certificate information)

openssl x509 -req -sha256 -days 3650 -in permca.csr -text -signkey private\permcapriv.pem -out permcacert.pem

You can now delete the identca.csr file. Copy the CA certificate to a common location:

copy permcacert.pem ..\..\apps\deployall\

You are now your own Permissions CA and can sign permissions and domain governance files for use by DDS Secure applications.

# Generate an identity for each DDS application

Each DDS application in a secured system requires a valid identity to support authentication. An application’s identity is stored in an identity certificate file in X.509 format. Creation of the identity cert is a three-step process: generate a private key, generate a certificate request, and then use the certificate request (.csr file) to generate the identity certificate. You will need a configuration file for OpenSSL when generating the CSR because it points to the ca’s keys.

A template folder is provided with this workbook. It contains the following:

|  |  |
| --- | --- |
| File | Description |
| deploy\ | Folder where keys, certs, and signed files for this app get generated. |
| appname.cnf | Sample OpenSSL configuration file. This should be renamed for each application |
| app1limited\_permissions.xml  app2fullperm\_permissions.xml | Sample permissions files. This should be renamed for each application |
| USER\_QOS\_PROFILES.xml | Sample QoS configuration file. |

To begin creating files for a new application, copy the template folder to a new directory. I suggest creating two applications called “app1limited” and “app2fullperm”  
run from the apps\ dir:

xcopy /E template\\* <APPNAME>\

Within the new application directories, edit the “appname.cnf” file and change the following fields to match your application:

countryName=US

stateOrProvinceName=CA

localityName=Sunnyvale

organizationName=Real Time Innovations

emailAddress=appname@rti.com

commonName=appname

These fields will need to match those in your permissions file when you create those files later.

Now rename the “appname.cnf” file to your application’s name: <APPNAME>.cnf

**There is a windows batch file that performs all of the following application identity commands for you called generate.app.ident.bat but you can also run them individually. This bat file asks for the application name during execution.**

Execute the following command within each new application directory, substituting in the application name each time:

openssl ecparam -name secp256k1 -genkey -noout -out deploy\**<APPNAME>**priv.pem

You now have a private key for each application in your system. You’ll use these to create a unique identity certificate for each application. Repeat the following commands with each of the private keys you just created.

Generating the CSRs, run within each new application directory you create:

openssl req -new -sha256 -key deploy\**<APPNAME>**priv.pem -out **<APPNAME>**.csr -config **<APPNAME>**.cnf

Now use the CSRs to generate identity certificates for each application, signed by the identity CA. The .cnf file referenced in this command points to the identity CA and it’s private key:

openssl ca -config ..\..\ca\identity\openssl.cnf -days 3650 -in **<APPNAME>**.csr -out deploy\**<APPNAME>**identcert.pem

At the conclusion of this step you can delete the <APPNAME>.csr file, and you will have two files ready to be deployed with the application that uniquely identify it:

|  |  |  |
| --- | --- | --- |
| File | Location | Description |
| <app name>identcert.pem | deploy\ | Public identity certificate for this application, signed by the Identity CA |
| <app name>priv.pem | deploy\ | Private key file used by this application (keep private to this application!) |

You’ll notice in the ca\identity\ directory the CertificateDatabase files were automatically backed up into “.old” files. And in the ca\identity\participantsignedcerts\ contains each application certificate the identity CA signed (named with their sequential serial number).

# Create and Sign a Domain Governance File

All the DDS applications in your system share a common, signed copy of a Domain Governance file. Once you have created the Domain Governance file, you will sign it with your Identity CA (when using a single CA) or your Permissions CA (when using a dual CA).

## Domain Governance File Overview

The domain governance file specifies broad domain-level security settings such as discovery, participant access control, and topic access rules. It does not provide any per-participant controls like the permissions files do. Rules can be applied in a whitelist (allow) and blacklist (deny) manner.

## Domain Governance File Sections

The domain governance file defines two levels of configuration properties: **domain**, affecting participants in the domain; and **topic**, affecting endpoints in that domain.

The elements <domain\_rule> section control the security settings for all participants in the specified domains. The layout of this section is as follows:

<domain\_rule>

<domains>

<id>0</id>

</domains>

<allow\_unauthenticated\_participants>true</allow\_unauthenticated\_participants>

<enable\_join\_access\_control>false</enable\_join\_access\_control>

<discovery\_protection\_kind>NONE</discovery\_protection\_kind>

<liveliness\_protection\_kind>NONE</liveliness\_protection\_kind>

<rtps\_protection\_kind>NONE</rtps\_protection\_kind>

<topic\_access\_rules>

<topic\_rule>

<topic\_expression>\*</topic\_expression>

<enable\_discovery\_protection>false</enable\_discovery\_protection>

<enable\_read\_access\_control>false</enable\_read\_access\_control>

<enable\_write\_access\_control>false</enable\_write\_access\_control>

<metadata\_protection\_kind>NONE</metadata\_protection\_kind>

<data\_protection\_kind>NONE</data\_protection\_kind>

</topic\_rule>

</topic\_access\_rules>

</domain\_rule>

...

The top level <domain\_access\_rules> section can contain one or more <domain\_rule> sections. Within each <domain\_rule> section there is a list of domains to apply the rules to.

The <domains> tag specifies either a single domain ID enclosed by a single <id></id> pair as shown, or a range of domain IDs specified with <id\_range>, <min>, and <max> tags. All domain settings that follow apply to the specified domain(s).

<allow\_unauthenticated\_participants> determines if an authenticated participant is allowed to match and share data with an unauthenticated domain participant. This value can be “true” or “false”. Setting this value to “true” during development can be useful to enable apps that have not yet been converted to DDS Secure to share data.

<enable\_join\_access\_control> controls whether remote DomainParticipant permissions are checked when a remote DomainParticipant is discovered. Local DomainParticipant permissions are always checked using the local DomainParticipant’s Permissions Document. This value can be set to “true” or “false”.

<discovery\_protection\_kind> controls how the builtin datareader and datawriter entities used for discovery protect the information they exchange.

<liveliness\_protection\_kind> controls how the builtin datareader and datawriter entities used for liveliness protect the information they exchange.

<rtps\_protection\_kind> controls how outgoing messages are protected and the protection required for incoming messages. This is at the RTPS level, which can envelope sub-messages within a single packet. If <allow\_unauthenticated\_participants> is true, then this setting must be set to “NONE”. (you cant secure the data if you’re allowing unauthenticated participants)

Any setting other than “NONE” will cause the domain participant to reject incoming messages with an invalid GMAC/GCM.

Available protection kinds are:

NONE

SIGN – authentication only, no encryption

ENCRYPT – authentication and encryption

SIGN\_WITH\_ORIGIN\_AUTHENTICATION – unique auth keys per participant pair, no encryption

ENCRYPT\_WITH\_ORIGIN\_AUTHENTICATION – unique auth keys per participant pair, and encryption

Topic access rules

<enable\_discovery\_protection> controls if security should be enabled for the builtin endpoints for traffic related to this topic. This value can be “TRUE” or “FALSE”.

<enable\_read\_access\_control> controls whether local and remote datareader permissions are checked

<enable\_write\_access\_control> controls whether local and remote datawriter permissions are checked

<metadata\_protection\_kind> controls how to protect a datareader and datawriter’s messages including heartbeat, acknack, gap, and data submessages which can envelope around a serialized payload

<data\_protection\_kind> controls how to protect a datawriter’s serialized payload. Historic samples are stored after protection is applied, and not re-protected when resent. Receiver specific GMACs are not included so WITH\_ORIGIN\_AUTHENTICATION values are not allowed. This value can be “NONE”, “SIGN”, or “ENCRYPT”.

## Signing the Domain Governance File

In the example below, we sign the file “demo\_governance.xml” with the permissions CA (in a dual-CA installation). Run this command from the \governance\ folder:

**There is a windows batch file that performs the following domain governance signing command for you called sign.governance.bat but you can also run the following command manually.**

openssl smime -sign -in demo\_governance.xml -text -out ..\apps\deployall\demo\_governance\_signed.p7s -signer ..\ca\permissions\permcacert.pem -inkey ..\ca\permissions\private\permcapriv.pem

This creates a signed copy of the Domain Governance file (DemoGovernance\_signed.p7s in this case) which you then deploy to your system. This is a common file that should be deployed to and accessible by all the applications in your system.

Copy the signed domain governance file to the deployall directory:

copy demo\_governance\_signed.p7s ..\apps\deployall\

# Create and Sign Permissions files

Each application requires a signed permissions files that specifies the topics it can read and write. The permissions file is an XML file that is created by the developer and then signed by the Permissions CA. There are two sample permissions files located in the template directory. app1limited\_permissions.xml does not have permissions to publish triangles, where app2fullperm\_permissions.xml has full permissions to publish and subscribe to every topic. The files generated for this application can also be given to admin console to analyze the secure domain.

From each \apps\<APPNAME>\ directory rename the <APPNAME>\_permissions.xml file to your specific application name. Then modify the <APPNAME>\_permissions.xml file and verify the <subject\_name> attribute matches what is in the <APPNAME>.cnf file.

As noted in section “Setup for self-signing: Becoming your own Permissions CA (optional)” the use of a separate Permissions CA is optional – you can use your Identity CA to sign permissions files. In this example, we will use a separate Permissions CA. To use a single CA for both Identity and Permissions, change the -signer and -inkey arguments in the commands below.

**There is a windows batch file that performs the following application permissions signing command for you called sign.permissions.bat but you can also run the following command manually. This bat file asks for the application name during execution.**

Sign an apps permission file by the permissions CA, run from the \apps\<APPNAME>\ directory:

openssl smime -sign -in **<APPNAME>**\_permissions.xml -text -out deploy\**<APPNAME>**\_permissions\_signed.p7s -signer ..\..\ca\permissions\permcacert.pem -inkey ..\..\ca\permissions\private\permcapriv.pem

This creates a signed copy of the Permissions file (<APPNAME>\_permissions\_signed.p7s) in the deploy\ directory that you deploy with the application. Each time you change the permissions file, you will need to sign the permissions file using the command above.

# Files to Deploy

From the examples above, you will deploy the following files:

Files common to all applications located in \apps\deployall\:

|  |  |  |
| --- | --- | --- |
| Step created | Filename | Description |
| 2 | identcacert.pem | Identity CA certificate |
| 3 | permcacert.pem | Permissions CA certificate |
| 5 | demo\_governance\_signed.p7s | Signed Domain Governance file |

Files specific to each application located in each application directory \apps\<APPNAME>\deploy\:

|  |  |  |
| --- | --- | --- |
| Step created | Filename | Description |
| 4 | <APPNAME>identcert.pem | Identity certificate for this application |
| 4 | < APPNAME>priv.pem | Private key file for this application (best to keep in a protected directory separate from the application!) |
| 6 | <APPNAME>\_permissions\_signed.p7s | Signed permissions file for this application |

To test using the shapesdemo you can modify:

C:\Program Files\rti\_connext\_dds-6.0.1\resource\xml\RTI\_SHAPES\_DEMO\_QOS\_PROFILES.xml

Under the <qos\_library name=”Security”> paste the qos profiles from the apps\ USER\_QOS\_PROFILES.xml that point at the necessary files:

# OpenSSL command summary

## Creating Self-Signed CAs

Generate a self-signed CA:

openssl ecparam -name secp256k1 -genkey -noout -out private\identcapriv.pem

openssl req -new -sha256 -key private\identcapriv.pem -out identca.csr -config openssl.cnf

openssl x509 -req -sha256 -days 3650 -in identca.csr -text -signkey private\identcapriv.pem -out identcacert.pem

Keep the private key file (identcapriv.pem) private! You can distribute the CA certificate (identcacert.pem).

(optional) Generate a self-signed 2048-bit RSA Permissions CA:

openssl ecparam -name secp256k1 -genkey -noout -out private\permcapriv.pem

openssl req -new -sha256 -key private\permcapriv.pem -out permca.csr -config openssl.cnf

openssl x509 -req -sha256 -days 3650 -in permca.csr -text -signkey private\permcapriv.pem -out permcacert.pem

Keep the private key file (permcapriv.pem) private! You can distribute the Permissions CA certificate (permcacert.pem).

## Creating Signed Certificates for Each DDS Application

Change the <APPNAME> for each application:

openssl ecparam -name secp256k1 -genkey -noout -out deploy\**<APPNAME>**priv.pem

openssl req -new -sha256 -key deploy\**<APPNAME>**priv.pem -out **<APPNAME>**.csr -config **<APPNAME>**.cnf

openssl ca -config ..\..\ca\identity\openssl.cnf -days 3650 -in **<APPNAME>**.csr -out deploy\**<APPNAME>**identcert.pem

## Signing Documents with SMIME

Sign a document using an existing Permissions CA (dual CA setup):

openssl smime -sign -in %appname%\_permissions.xml -text -out deploy\**<APPNAME>**\_permissions\_signed.p7s -signer ..\..\ca\permissions\permcacert.pem -inkey ..\..\ca\permissions\private\permcapriv.pem