Create a DDS Secure demo while acting as your own CA

[1 Introduction 1](#_Toc46297092)

[1.1 Files You’ll Be Creating 1](#_Toc46297093)

[1.1.1 Per-Domain artifacts (shared among all applications on the same DDS domain) 1](#_Toc46297094)

[1.1.2 Per-participant artifacts 1](#_Toc46297095)

[1.2 Prerequisites 1](#_Toc46297096)

[1.3 Overview of Adding Security 1](#_Toc46297097)

[1.4 Best Practices for Adding Security 1](#_Toc46297098)

[1.5 Directory Structure Used in the Examples Below 1](#_Toc46297099)

[2 Setup for self-signing: Becoming your own Identity CA 1](#_Toc46297100)

[3 Setup for self-signing: Becoming your own Permissions CA (optional) 1](#_Toc46297101)

[4 Generate an identity for each DDS application 1](#_Toc46297102)

[5 Create and Sign a Domain Governance File 1](#_Toc46297103)

[5.1 Domain Governance File Overview 1](#_Toc46297104)

[5.2 Domain Governance File Sections 1](#_Toc46297105)

[5.3 Signing the Domain Governance File 1](#_Toc46297106)

[6 Create and Sign Permissions files 1](#_Toc46297107)

[7 Files to Deploy 1](#_Toc46297108)

[8 OpenSSL command summary 1](#_Toc46297109)

[8.1 Creating Self-Signed CAs 1](#_Toc46297110)

[8.2 Creating Signed Certificates for Each DDS Application 1](#_Toc46297111)

[8.3 Signing Documents with SMIME 1](#_Toc46297112)

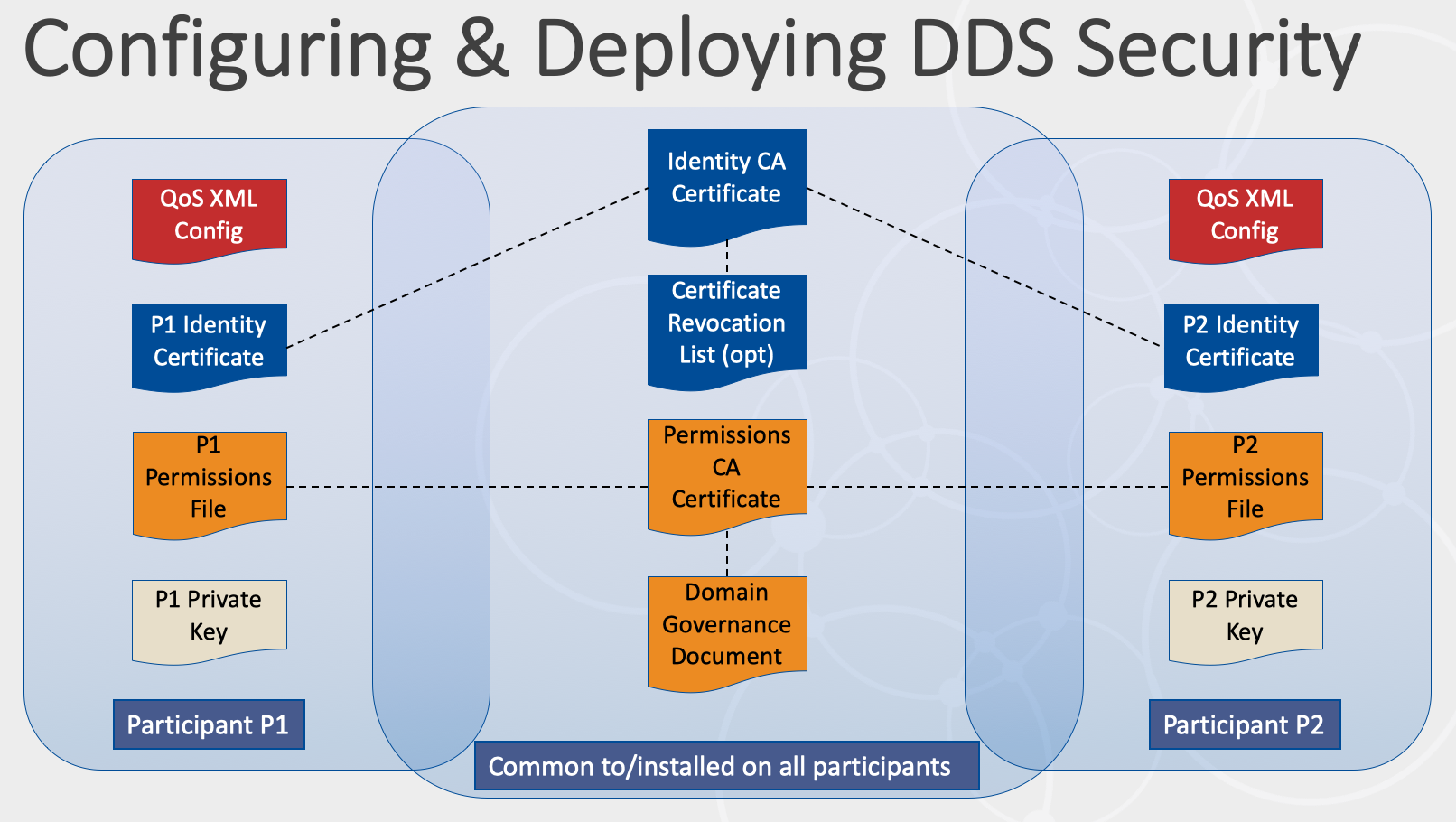
[9 Starting Shapes 1](#_Toc46297113)

# 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 (i.e. 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

* Identity certificate and a private key
  + Issued 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 have been installed.

## Overview of Adding Security

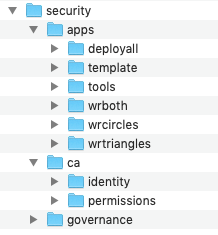
First, you’ll need to create the XML configuration files that define the security settings for your system. This includes a Domain Governance document, one or more Permissions files, and QoS settings for each application.

## 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 “security” 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 Certificate Authority for issuing and signing identity certificates (‘identity’), and for signing Permissions files and the Domain Governance file (“permissions”). 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 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 “security/ca/identity”. Copy the following files from the workbook. Clear out / zero out the DatabaseFiles [ps> for initial CertificateDatabaseIndex.txt put 00 <CRLF> in the file]

|  |  |  |
| --- | --- | --- |
| File | Location | Description |
| Openssl.cnf | . | Configuration file for OpenSSL. A template is provided. |
| CertificateDatabaseIndex | . | Certificates database index used by openssl |
| CertificateDatabase | . | Certificates database use by openssl |
|  | ./private | The private key for the Identity CA is stored here |
|  | ./certs | Signed identity certificates are stored here |

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. 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. 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 RSA encryption with 2048-bit key lengths).

openssl genrsa -out ./private/cakey.pem 2048

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 -key ./private/cakey.pem -out ca.csr -config openssl.cnf

Generate the CA certificate

openssl x509 -req -days 3650 -in ca.csr -signkey ./private/cakey.pem -out cacert.pem

Copy the CA certificate to a common location:

cp cacert.pem ../../apps/deployall

You are now your own Identity CA and can create identity certificates for use by DDS Secure applications. The following files have been added to your filesystem:

|  |  |  |
| --- | --- | --- |
| File | Location | Description |
| ca.csr | . | Certificate request used to create CA. Based on the CA private key. Can be deleted once the CA cert if created. |
| cacert.pem | . | Signed CA certificate. Copy this file to ../../apps/deployall |
| cakey.pem | ./private | Private key for your CA. Keep this private! Anyone that gets this key can impersonate your Identity CA and create identity certificates for DDS apps. |
| *nn*.pem | ./certs | Copies of issued identity certificates. The base name of file is a number that matches an index in the database. The first identity certificate you create is 01.pem, the next 02.pem, and so on. (Note: you won’t see any of these until you start creating identities for apps below in step 3. |

# 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.

In this section you’ll be working in “security/ca/permissions”. Copy the following files from the workbook. Clear out / zero out the DatabaseFiles

|  |  |  |
| --- | --- | --- |
| File | Location | Description |
| Openssl.cnf | . | Configuration file for OpenSSL. |
| PermissionsDatabaseIndex | . | Index of signed permissions files |
| PermissionsDatabase | . | Signed permissions file database |
|  | ./private | Make this directory to store your private key |

After the above configuration files are created, you have an environment where you can create a CA certificate for your local CA. 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 RSA encryption with 2048-bit key lengths).

openssl genrsa -out ./private/permissionscakey.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 -key ./private/permissionscakey.pem -out ca.csr -config openssl.cnf

Generate the CA certificate

openssl x509 -req -days 3650 -in ca.csr -signkey ./private/permissionscakey.pem -out permissionscacert.pem

Copy the CA certificate to a common location:

cp permissionscacert.pem ../../apps/deployall

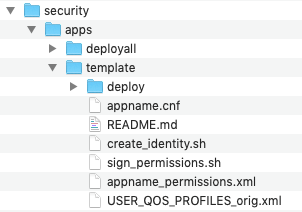
You are now your own Permissions CA and can sign permissions and domain governance files for use by DDS Secure applications. The following files have been added to your filesystem:

|  |  |  |
| --- | --- | --- |
| File | Location | Description |
| ca.csr | . | Certificate request used to create Permissions CA. Based on the Permissions CA private key. Can be deleted once the Permissions CA cert is created. |
| permissionscacert.pem | . | Signed Permissions CA certificate |
| permissionscakey.pem | ./private | Private key for your Permissions CA. Keep this private! Anyone that gets this key can impersonate your Permissions CA and sign permissions and governance documents |

# 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 two-step process: generate a private key and 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.

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



|  |  |
| --- | --- |
| ./deploy | Folder where keys, certs, and signed files for this app get generated. |
| appname.cnf | Sample OpenSSL configuration file. This will be renamed after running the create\_identity script. |
| README.md | Instructions for using the files and scripts in this directory |
| create\_identity.sh | Script file to do one-time creation of identity certs and keys for this application. |
| sign\_permissions.sh | Script to sign the permissions file for this app. Can be run multiple times as you make changes to permissions |
| appname\_permissions.xml | Sample permissions file. Gets renamed by create\_identity.sh |
| USER\_QOS\_PROFILES\_orig.xml | Sample QoS configuration file. There is also one in the deploy folder. |

To begin creating files for a new application, copy the template folder:

cd ~/security/apps

cp -R template myapp

cd myapp

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=test@rti.com

commonName=test

**These fields will need to match those in your permissions file.**

*Note: The steps below for creating identity artifacts for an application are captured in a script file in the template directory called “create\_identity.sh”. You can either enter them manually as shown below, or just edit and run the script file. See the README.md file for instructions on using the scripts.*

Execute the following command once for each application directory you create, substituting in the application name each time:

openssl genrsa -out ./private/**<app name>**key.pem 2048

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:

openssl req -config **<app name>**.cnf -new -key ./private/**<app name>**key.pem -out **<app name>**.csr

Now use the CSRs to generate identity certificates for each application:

openssl ca -config ../../ca/identity/openssl.cnf -days 365 -in **<app name>**.csr -out **<app name>**.pem

At the conclusion of this step you will have two files ready to be deployed with the application that uniquely identify it:

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

# 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’ve created the Domain Governance file, you sign it with your Identity CA (single CA installation) or your Permissions CA (dual CA installation) certificate.

[ps> run script sign\_gov.sh from the governance directory]

## Domain Governance File Overview

## 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\_range>

<min>50</min>

<max>250</max>

</id\_range>

</domains>

<allow\_unauthenticated\_participants>false</allow\_unauthenticated\_participants>

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

<!—See next section for topic rule configuration -->

</topic\_access\_rules>

</domain\_rule>

<domains> specifies either a single domain ID enclosed by a single <domain></domain> pair, or a range of domain IDs specified with <id\_range>, <min>, and <max> tags as shown. 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>

a main <domain\_access\_rules> section. This can contain one or more <domain\_rule> sections. Within each <domain\_rule> section there is a list of domains to apply the rules to, a handful of domain-wide settings, and <topic\_access\_rules> section containing one or more <topic\_rule> sections. Each of these is detailed below.

Domain-level settings

Topic access rules

## Signing the Domain Governance File

In the example below, we sign a Domain Governance file named “demo\_governance.xml” in a dual-CA installation:

openssl smime -sign -in /path/demo\_governance.xml -text -out /path/deployall/demo\_governance\_signed.p7s -signer /path\_xyz/permissionscacert.pem -inkey /path/private/permissionscakey.pem

(depending upon what directory you are in, relative paths to each of the files may be used)

This creates a signed copy of the Domain Governance file (demo\_governance\_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.

A “sign\_gov.sh” script is shipped with the examples to sign the governance file with the command above and place the signed file to the deployall directory.

# Create and Sign Permissions files

Note: Each App Permissions file <subject></subject> must match the information in the app\_name.cnf file – Country, state, email etc.

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. As noted in 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’ll 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. Here we sign a permissions file called DemoAppPermissions.xml:

(again below the paths to each file are omitted)

openssl smime -sign -in DemoAppPermissions.xml -text -out DemoAppPermissions\_signed.p7s -signer permissionscacert.pem -inkey ./private/permissionscakey.pem

This creates a signed copy of the Permissions file (DemoAppPermissions\_signed.p7s) that you deploy with the application. Each time you change the permissions file, you’ll need to sign the permissions file using the command above.

A “sign\_permissions.sh” script is shipped with the examples to sign the permissions file with the command above and place the signed file in the deploy directory for the application. See the README.md file for instructions on how to use it. Each time you make changes to the permissions file, re-run the script file to re-sign the permissions.

# Files to Deploy

From the examples above, you’ll deploy the following files:

Files common to all applications:

|  |  |  |
| --- | --- | --- |
| Step created | File | Description |
| 2 | cacert.pem | Identity CA certificate |
| 3 | permissionscacert.pem | Permissions CA certificate |
| 5 | Demo\_governance\_signed.p7s | Signed Domain Governance file |

Files specific to each application:

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

# OpenSSL command summary

## Creating Self-Signed CAs

Generate a self-signed 2048-bit RSA CA:

openssl genrsa -out ./private/cakey.pem 2048

openssl req -new -key ./private/cakey.pem -out ca.csr -config openssl.cnf

openssl x509 -req -days 3650 -in ca.csr -signkey ./private/cakey.pem -out cacert.pem

Keep the private key file (cakey.pem) private! You can distribute the CA certificate (cacert.pem).

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

openssl genrsa -out ./private/permissionscakey.pem

openssl req -new -key ./private/permissionscakey.pem -out ca.csr -config openssl.cnf

openssl x509 -req -days 3650 -in ca.csr -signkey ./private/permissionscakey.pem -out permissionscacert.pem

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

## Creating Signed Certificates for Each DDS Application

Change the appname for each application:

openssl genrsa -out ./private/appname\_key.pem 2048

openssl req -config appname.cnf -new -key ./private/appname\_key.pem -out appname.csr

openssl ca -config openssl.cnf -days 365 -in appname.csr -out appname.pem

## Signing Documents with SMIME

Sign a document using an existing Identity CA (single CA setup):

openssl smime -sign -in FileToSign.xml -text -out FileToSign\_signed.p7s -signer cacert.pem -inkey ./private/cakey.pem

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

openssl smime -sign -in FileToSign.xml -text -out FileToSign\_signed.p7s -signer permissionscacert.pem -inkey ./private/permissionscakey.pem

# Starting Shapes

Remember to start secure shapes from a window where you: export DDS\_SECURE\_DEMO\_HOME=path\_to\_top\_level\_security\_dir (e.g. $HOME/DDSexamples/security)

You may start a secure and unsecure 6.0.1 shapes demo. The secure demo may contain both the reader and writer USER\_QOS\_PROFILES.XML

TODO:

Explain the importance of subject names: how they match

Explain the Domain Governance file – use the templated file

How to test each setting:

Allow\_unauthenticated\_participants

File structure

Explanation of individual governance settings

Sections in a Permissions file

How to create permissions rules

File structure

Explanation of individual governance settings