RTI Security Plugins Getting Started

Modern C++

Version 7.7.0



Contents

1 Introduction to Security Plugins (RTI Security Plugins) 1
- 1.1 Key Features . . . 2
- 1.2 Paths Mentioned in Documentation . . . 4

2 DDS System Threats 6

3 Securing a DDS Domain 8
- 3.1 Securing a DomainParticipant . . . 10

4 Hands-On 1: Securing Connext DDS Applications 12
- 4.1 Generating a Connext DDS Project . . . 12
- 4.2 Adding Security Artifacts to Your Project . . . 14
- 4.3 Enabling Security in Your QoS Profiles . . . 14
- 4.4 Linking Your Applications Against Security Plugins and OpenSSL Libraries . . . 16
  - 4.4.1 Building the Application . . . 17
- 4.5 Running the Applications . . . 18
  - 4.5.1 Configuring the Environment in Both Command Prompts . . . 18
  - 4.5.2 Checking Communication . . . 18
- 4.6 Checking that Your Applications Communicate Securely . . . 20
  - 4.6.1 Verifying that Eavesdropping Attempts are Frustrated . . . 20
  - 4.6.2 Detecting Eavesdropping Attempts . . . 20
- 4.7 Further Exercises . . . 21
  - 4.7.1 Give Different Credentials to Each Application in Your System . . . 21
- 4.8 Troubleshooting . . . 23

5 Hands-On 2: Defining Your System's Security Requirements 25
- 5.1 Specifying the Security Requirements . . . 25
- 5.2 Composing a Governance Document with the Security Requirements . . . 28
- 5.3 Signing the Governance Document . . . 30
- 5.4 Updating the QoS Profiles in Your Project . . . 30
- 5.5 Checking that the Specified Security Rules Are Applied . . . 31
  - 5.5.1 Verifying Communication . . . 31
  - 5.5.2 Checking the New Security Rules . . . 32
- 5.6 Further Exercises . . . 33
  - 5.6.1 Protecting the Domain . . . 33
  - 5.6.2 Adding a Topic Rule for the PatientMonitoring Topic . . . 34
- 5.7 Troubleshooting . . . 35

6 Hands-On 3: Defining the DomainParticipant Permissions **36**
6.1 Granting Permissions to Your Secure Participants . . . 36
6.2 Binding the Permissions Document to Your DomainParticipants . . . 38
6.3 Signing the Permissions Documents . . . 40
6.4 Updating the QoS Profiles in Your Project . . . 40
6.5 Checking that the New Permissions Are Applied . . . 41
6.5.1 Communication Only Works in Domain 1 . . . 41
6.5.2 Alice Is Only Allowed to Publish Data . . . 44
6.6 Further Exercises . . . 46
6.6.1 Define Different Permissions for Each Application in Your System . . . 47
6.7 Troubleshooting . . . 47

7 Hands-On 4: Generating and Revoking Your Own Certificates Using OpenSSL **49**
7.1 Preliminary Steps . . . 50
7.1.1 Initialize the OpenSSL CA Database . . . 51
7.1.2 Limit the Access of the CA's Private Key . . . 51
7.2 Generating a New Identity CA . . . 52
7.2.1 Specifying the New Identity CA Certificate in QoS Profiles . . . 53
7.3 Generating Identity Certificates . . . 53
7.3.1 Specifying the New Identity Certificates to Your QoS Profiles . . . 54
7.4 Updating Permissions Documents with New Credentials . . . 55
7.5 Generating a New Permissions CA . . . 57
7.5.1 Specifying the New Permissions CA Certificate in QoS Profiles . . . 58
7.6 Signing the Governance and Permissions Documents . . . 58
7.6.1 Specifying the New Governance and Permissions Documents in Your QoS Profiles . . . 58
7.7 Updating the Subscriber's Configuration . . . 59
7.8 Revoking an Identity Certificate . . . 61
7.8.1 Specifying the New Certificate Revocation List in QoS Profiles . . . 62
7.9 Troubleshooting . . . 62

8 Hands-On 5: Checking that Your DDS Traffic Is Protected **64**
8.1 Disabling Security and Preparing Your Project for Traffic Capturing . . . 64
8.1.1 Analyzing RTPS Packets in Wireshark . . . 65
8.2 Encrypting the Serialized Payload . . . 68
8.2.1 Analyzing RTPS Packets in Wireshark . . . 69
8.3 Troubleshooting . . . 70

9 Next Steps **73**

10 Copyrights and Notices **74**

Chapter 1

Introduction to Security Plugins (RTI Security Plugins)

Prerequisites	• *RTI Connext®* installed, including SDK (see the RTI Connext Installation Guide) • Security Plugins installed (see the RTI Security Plugins Installation Guide) • Familiarity with *Connext* (i.e., you've completed Introduction to Publish/Subscribe) • Familiarity with *Connext* tools, such as *RTI Admin Console* • Familiarity with security concepts and techniques (digital certificates, public key infrastructure, private/public key pairs, authentication, encryption, etc.) • Familiarity with defining QoS profiles to *Connext* applications in XML format
Time to complete	2 hours
Concepts covered in this document	• Introduction to DDS Security and Security Plugins • Enabling Security Plugins in your *Connext* applications • Dynamic linking against Security Plugins and OpenSSL • Translating the security requirements of your system to a Governance Document • Publishing/Subscribing with different protection kinds (authentication, encryption, etc.) • Authenticating your applications with custom digital certificates and private keys • Defining privileges for your applications with Permissions Documents

Security Plugins allow you to address your databus security requirements in a granular and pluggable way. To support this, each of the Security Plugins covers a different aspect of security:

- **Authentication.** Provides the means to verify the identity of the application and/or user that invokes operations on DDS. Includes facilities to perform mutual authentication between *DomainParticipants*

and establish a shared secret.

- **Access Control.** Provides the means to enforce policy decisions on what DDS-related operations an authenticated entity can perform. For example, which *Domains* it can join, which *Topics* it can publish or subscribe to, etc.
- **Cryptography.** Implements (or interfaces with libraries that implement) all cryptographic operations including encryption, decryption, hashing, digital signatures, etc. This includes the means to derive keys from a shared secret.
- **Logging.** Supports auditing of all DDS security-relevant events.

The OMG DDS Security specification defines a set of builtin plugins for providing interoperable authentication, access control, cryptography, and a logging *Topic*. The Security Plugins are the *Connext* implementation of these OMG DDS Security builtin interoperability plugins. This way, the Security Plugins offer a DDS Security solution that can interoperate with DDS implementations from other vendors.

1.1 Key Features

- The OMG DDS Security specification decouples the different security aspects in a set of plugins:
  - **Authentication:** Ensures that DDS entities are authenticated.
  - **Access Control:** Enforces access control for *Domains*, *Topics*, etc.
  - **Cryptography:** Maintains data integrity and confidentiality.
  - **Logging:** Supports auditing of all DDS security-relevant events, allowing you to increase the system's visibility, which may help track and improve system's availability.
- The Security Plugins can potentially run over any transport, including the builtin UDP transport with multicast and TCP transport.
- Secure multicast support enables efficient and scalable distribution of data to many *Subscribers*.
- You can customize the Security Plugins to accommodate proprietary or FIPS 140-2 compliant cryptography solutions, take advantage of custom security hardware or change the behavior of the plugins in any number of ways. The Security Plugins SDK enables you to customize the Security Plugins to meet your system's security requirements.
- The OMG DDS Security specification addresses the security aspect of the communication in a one-to-many, friendly, data-centric way, enabling applications to define different security policies based on the nature of the shared data. This aligns with the decentralized nature of DDS and asserts its benefits:
  - No single point of failure
  - High performance and scalability
- The Security Plugins support all of the cryptographic algorithms specified by the OMG DDS Security specification. For more information about the supported algorithms, refer to Supported Cryptographic Algorithms in the *RTI Security Plugins User's Manual*.

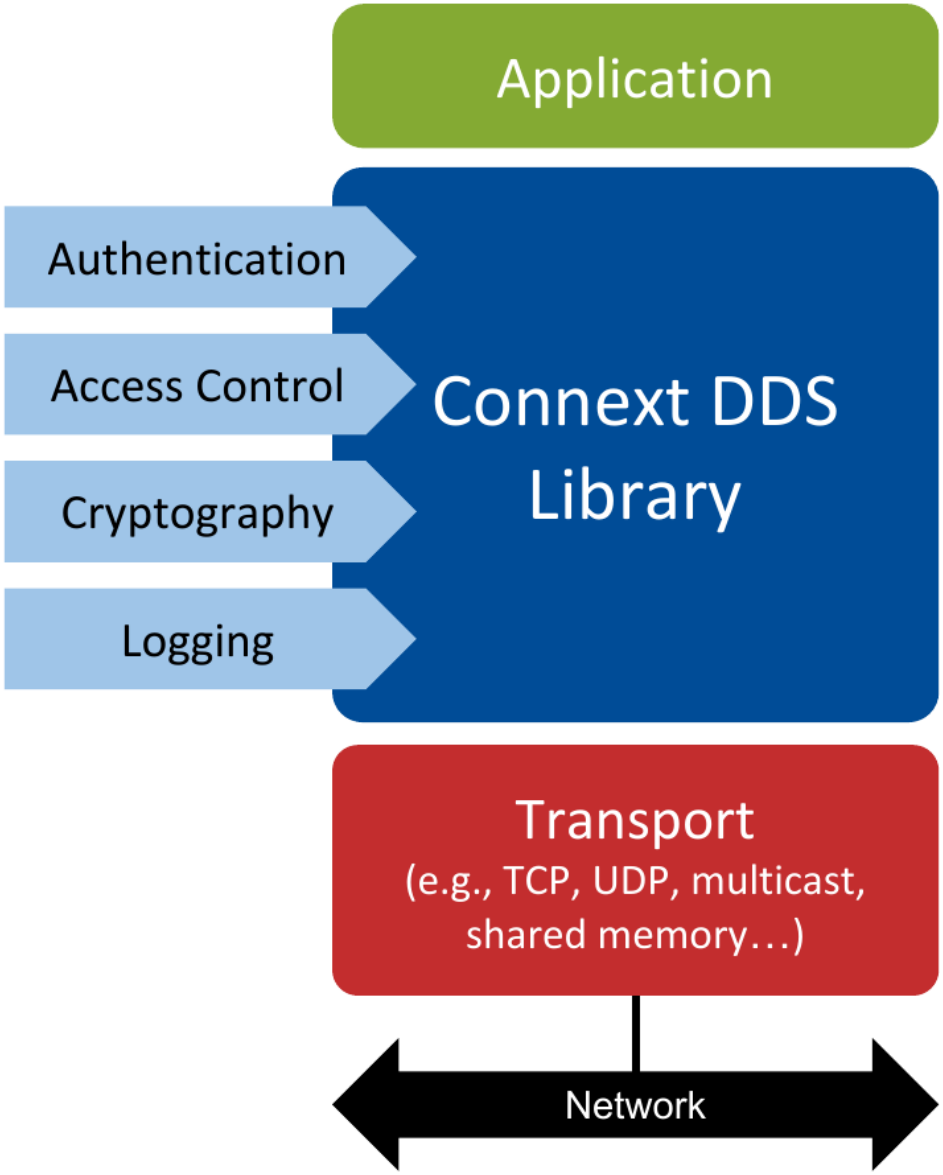


Figure 1.1: Architecture of an application using SECURITY PLUGINS. All the currently available security plugins (Authentication, Access Control, Cryptography and Logging) are implemented in a single nddssecurity library.

1.2 Paths Mentioned in Documentation

This documentation refers to the following directories, depending on your operating system:

Linux

- `$NDDSHOME` This refers to the installation directory for *Connext*.

  The default installation paths are:

  – Non-root user:

  `/home/<your user name>/rti_connext_dds-<version>`

  – Root user:

  `/opt/rti_connext_dds-<version>`

  `$NDDSHOME` is an environment variable set to the installation path.

- `<path to examples>` By default, examples are copied into your home directory the first time you run *RTI Launcher* or any script in `$NDDSHOME/bin`. This document refers to the location of the copied examples as `<path to examples>`.

  Wherever you see `<path to examples>`, replace it with the appropriate path.

  Default path to the examples:

  `/home/<your user name>/rti_workspace/<version>/examples`

macOS

- `$NDDSHOME` This refers to the installation directory for *Connext*.

  The default installation path is:

  `/Applications/rti_connext_dds-<version>`

  `$NDDSHOME` is an environment variable set to the installation path.

- `<path to examples>` By default, examples are copied into your home directory the first time you run *RTI Launcher* or any script in `$NDDSHOME/bin`. This document refers to the location of the copied examples as `<path to examples>`.

  Wherever you see `<path to examples>`, replace it with the appropriate path.

  Default path to the examples:

  `/Users/<your user name>/rti_workspace/<version>/examples`

Windows

- `%NDDSHOME%` This refers to the installation directory for *Connext*.

  The default installation paths are:

  – User without Administrator privileges:

  `<your home directory>\rti_connext_dds-<version>`

  - User with Administrator privileges:

    `"C:\Program Files\rti_connext_dds-<version>"`

  `%NDDSHOME%` is an environment variable set to the installation path.

Note: When using a command prompt to enter a command that includes the path `C:\Program Files` (or any directory name that has a space), enclose the path in quotation marks. For example: "`C:\Program Files\rti_connext_dds-<version>\bin\rtilauncher.bat`". Or if you have defined the `NDDSHOME` environment variable: `"%NDDSHOME%\bin\rtilauncher.bat"`.

- `<path to examples>` By default, examples are copied into your home directory the first time you run *RTI Launcher* or any script in `%NDDSHOME%/bin`. This document refers to the location of the copied examples as `<path to examples>`.

  Wherever you see `<path to examples>`, replace it with the appropriate path.

  Default path to the examples:

  `<your Windows documents folder>\rti_workspace\<version>\examples`

  Where `'your Windows documents folder'` depends on your version of Windows. For example, on Windows 10 systems, the folder is `C:\Users\<your user name>\Documents`.

Sometimes this documentation uses `<NDDSHOME>` to refer to the installation path. Whenever you see `<NDDSHOME>` used in a path, replace it with `$NDDSHOME` for Linux or macOS, with `%NDDSHOME%` for Windows, or with your installation path.

Chapter 2

DDS System Threats

One of the key values of *Connext* — and the OMG DDS standard — is *data-centricity*. Applications in the same domain can exchange data freely by just publishing and subscribing to the *Topics* they are interested in.

Suppose that in a hospital we have a patient monitoring device, Alice, that has a *DataWriter* publishing samples to a given *Topic T*, in a certain domain. A remote monitor in the control room, Bob, has a *DataReader* interested in *Topic T*. After Alice and Bob discover each other, they will start exchanging data samples for *Topic T*.

Both Alice and Bob are legitimate applications, so they should be able to communicate as they were designed to. In this case, nothing should prevent them from communicating. However, consider that a person from outside the hospital's organization has been able to connect to the hospital's network. This person has a device, Eve, with a *DataReader*, that isn't supposed to subscribe to *Topic T*. In a non-secure scenario, Eve and Alice *will* discover each other, then Alice will start sending data samples to Eve. Note that the original version of the DDS and RTPS standards do not define any mechanism to verify whether Eve is authorized[1] to subscribe to *Topic T*. Although eavesdropping is not always a problem on every *Topic* or every system, the medical data in *Topic T* is very sensitive and should be protected against eavesdropping. In other words, we should have a mechanism to guarantee that only authorized *DataReaders* are able to subscribe to *Topic T* and to make sense of data published to it.

A more severe problem comes when a malicious participant, Mallory, breaks into the network and joins the domain to publish random data to *Topic* T. Even worse, in a non-secure scenario she could easily perform tampering and/or replay attacks by subscribing to the *Topic* with a *DataReader*, then publishing modified data samples with a *DataWriter*. This could have severe consequences, such as nurses and doctors in the control room receiving false alarms regarding the vital signs of random patients, causing serious problems to the hospital.

Finally, these attacks could cross *DDS Domains* if an infrastructure service, such as *RTI Routing Service*, joins a domain being attacked by malicious *DomainParticipants*.

The diagram in Figure 2.1 depicts these attacks in a system with the following participants:

- **Alice**: a legitimate application publishing to *Topic T*, allowed to publish these samples.
- **Bob**: a legitimate application subscribing to *Topic T*, allowed to access that information.

[1] There are some measures that you can apply in order to not blindly trust everyone that joins the domain, such as the `accept_unknown_peers` setting in the DISCOVERY QosPolicy (see DISCOVERY QosPolicy (DDS Extension) in the Core Libraries User's Manual). However, these methods are not enough when your system is under a security attack.

- **Eve**: an eavesdropper trying to subscribe to *Topic T* without authorization - to perform unauthorized subscription (1).
- **Trudy**: an intruder trying to publish into the databus without authorization - to perform unauthorized publication (2).
- **Mallory**: a malicious insider (for instance, authorized to subscribe to data but not to publish) trying to perform tampering and replay (3).
- **Trent**: an Infrastructure Service that legitimately subscribes to and publishes data.

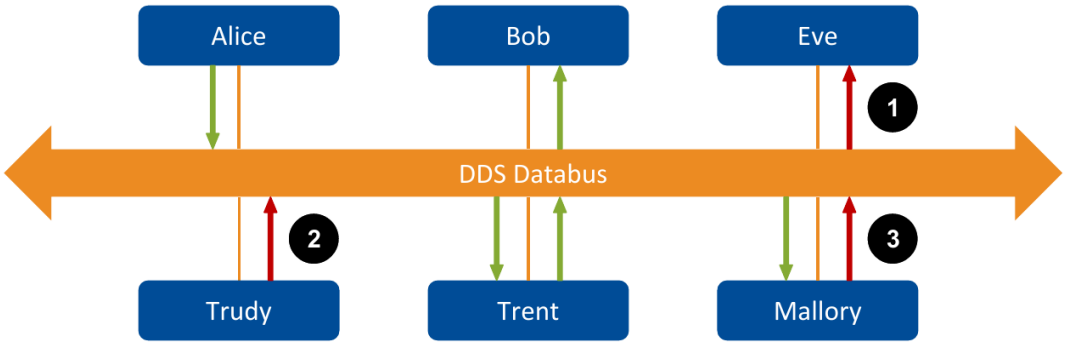


Figure 2.1: Threats affecting the OMG DDS standard in a non-secure scenario.

Given its data-centricity, traditional security solutions — mostly focused on securing a pipeline for byte exchange — do not fit OMG DDS well. Moreover, using a pipeline requires one-to-one sessions between peers, which doesn't allow multicast, so scalability is limited. In *Securing a DDS Domain*, we'll see how the SECURITY PLUGINS solve these problems.

Chapter 3

Securing a DDS Domain

In a DDS Secure system, a **Governance Document** defines the security requirements for communication. This file contains a mapping between *Domain* IDs and the security policies that *DomainParticipants* must follow to interact in that *Domain*. Some examples of those rules are:

Governance rule	Description	Possible values
`discovery_protection_kind`	The level of protection *DomainParticipants* must use for discovery	`NONE`, `SIGN`, `SIGN_WITH_ORIGIN_AUTHENTICATION`, `ENCRYPT` or `ENCRYPT_WITH_ORIGIN_AUTHENTICATION`
`allow_unauthenticated_participants`	Whether unauthenticated *DomainParticipants* may communicate within the *Domain*	`TRUE` or `FALSE`
`data_protection_kind`	The level of protection that should be used for data of individual *Topics* within that *Domain*	`NONE`, `SIGN`, `SIGN_WITH_ORIGIN_AUTHENTICATION`, `ENCRYPT` or `ENCRYPT_WITH_ORIGIN_AUTHENTICATION`

Going back to our example, the hospital can now protect the confidentiality of *Topic T* by setting its `data_protection_kind` to `ENCRYPT` in the *Domains* where it is published. With this, Eve will not be able to guess the vital signs of the patients, even if she has access to the messages being exchanged in the *Secure Domain*.

As you can see, the rules that compose the Governance Document specify how your system is protected. All the *DomainParticipants* in your secure system need to load the same[2] Governance Document, either by having

[2] If not the same, at least Governance Documents loaded by different *DomainParticipants* need to be compatible. For further

a copy of it, or by accessing a single Governance Document from a common location.

In addition to meeting the security requirements specified in the Governance Document, every *DomainParticipant* joining a *Secure Domain* must be associated with a **Permissions Document**. This Permissions Document contains a set of grants, which determine what the local participant is allowed to do in the *Domain*.

For example, Alice should only have permission to publish *Topic T*, since her only mission is to monitor patient vitals. On the other hand, Bob's role is to display everything related to patients' health conditions. Therefore, he should have permission to subscribe to any *Topic* related to patients' health, but he should not be allowed to publish any *Topics*. This step of creating Permissions Documents goes beyond many traditional methods of security, where an application that is allowed to communicate within a secure system is generally assumed to be safe, and may be able to access data that it should not. The use of Permissions Documents locks down the specific access that trusted applications have, so if one becomes compromised, the damage to the system is limited.[1]

Key Terms

A **Governance Document** defines how your system is protected, while **Permissions Documents** define who can access what.

To prevent Mallory from pretending to have permission to both publish and subscribe to *Topic T*, both Governance and Permissions Documents are signed by the **Permissions Certificate Authority (CA)**. The Permissions CA must be shared by all *DomainParticipants*[3], therefore any *DomainParticipant* trusting that Permissions CA can verify whether another *DomainParticipant* has the permissions it claims.

As you can see, Governance and Permissions Documents allow you to define the security requirements of your system in a data-centric way.

In *Hands-On 2: Defining Your System's Security Requirements*, you will define the security requirements of your secure *DDS Domain*. All the *DomainParticipants* joining your *Secure Domain* will have to meet these requirements in order to communicate. System-wide security requirements are completely defined by the following files:

- **Permissions CA certificate:** Shared by all the *DomainParticipants* in your secure system.[3] The Permissions CA certificate is used to verify that Permissions and Governance Documents are legitimate.
- **Identity CA certificate:** Shared by all the *DomainParticipants* in your secure system.[4] The Identity CA certificate is used to authenticate the remote *DomainParticipants*, by verifying that the Identity Certificates are legitimate (see *Securing a DomainParticipant*).
- **Governance Document:** Shared by all the *DomainParticipants* in your secure system.[Page 8, 2] The Governance Document specifies which *Domains* should be secured and how. It is signed by the Permissions CA.

information, see Governance Document in the *RTI Security Plugins User's Manual*.

[1] This approach is based on the Principle of Least Privilege. For further information, see Applying DDS Protection in the Security Plugins User's Manual.

[3] In systems where multiple Permissions Certificate Authorities may exist, you can use the `access_control.alternative_permissions_cas` property to specify alternative Permissions CA certificates. For further details, see Governance and Permissions in the *RTI Security Plugins User's Manual*.

[4] Depending on your use case, different *DomainParticipants* may trust a different Identity CA, for example, when several intermediate CAs exist in your PKI. For further details, see Alternative CAs in the *RTI Security Plugins User's Manual*.

Note: The Identity CA and Permissions CA may be the same, depending on your use case.

3.1 Securing a DomainParticipant

As introduced in the previous section, the Permissions Document defines what a specific *DomainParticipant* is allowed to do in a secure *DDS Domain*. However, Permissions Documents need to be exchanged to claim and verify permissions. This means that Mallory, who will not give up that easily, may intercept some Permissions Documents and pretend to be the legitimate holder of those permissions. To avoid this situation, *DomainParticipants* in a *Secure Domain* need to be mutually authenticated. To achieve this, every secure *DomainParticipant* has an **Identity Certificate** and a **Private Key**.[5] These documents are needed to perform mutual authentication and to establish shared secrets in a secure way, by using the Diffie-Hellman public-key protocol.

To prevent Mallory from forging her own identity, all the Identity Certificates in your system have to be signed by an **Identity CA**. As part of the authentication process, secure *DomainParticipants* will use the Identity CA certificate to validate the identity of discovered peers.

If a *DomainParticipant* cannot be authenticated, it won't be allowed to publish or subscribe to protected *Topics*. Depending on the policies defined in the Governance Document, it will be restricted to unprotected *Topics* or it won't be allowed to communicate at all (see `allow_unauthenticated_participants` in the Related Governance Attributes for Authentication) in the *RTI Security Plugins User's Manual*.

Because Mallory cannot forge her own identity, she will fail to authenticate, thus she will not be able to claim any permissions. That guarantees that Mallory will not be able to publish or subscribe to *Topic T*.

In *Hands-On 3: Defining the DomainParticipant Permissions*, you will provide an identity to your *DomainParticipants* and set the permissions for each of them. The steps in this hands-on exercise guarantee that if an attacker is able to compromise a trusted application in the system, the damage they can do is limited. The following files define the identity and permissions of a *DomainParticipant* entirely:

- **Identity Certificate** signed by the Identity CA. Other participants will request this certificate to verify the identity of the local participant.
- **Private Key**, only known to the local participant. It is needed to complete the authentication process, which provides a way of verifying the identity and setting a Shared Secret.
- **Permissions Document** signed by the Permissions CA. This document specifies what *Domains* and partitions[6] the local participant can join and what *Topics* it can read/write.

[5] Private keys need to be securely stored and securely accessed by the local application. The SECURITY PLUGINS do not provide a secure storage mechanism. You are responsible for storing your private keys in a secure place.

[6] Partitions are outside the scope of this document. For more information, see Partitions in the *RTI Security Plugins User's Manual* and the PARTITION QosPolicy in the Connext DDS Core Libraries User's Manual.

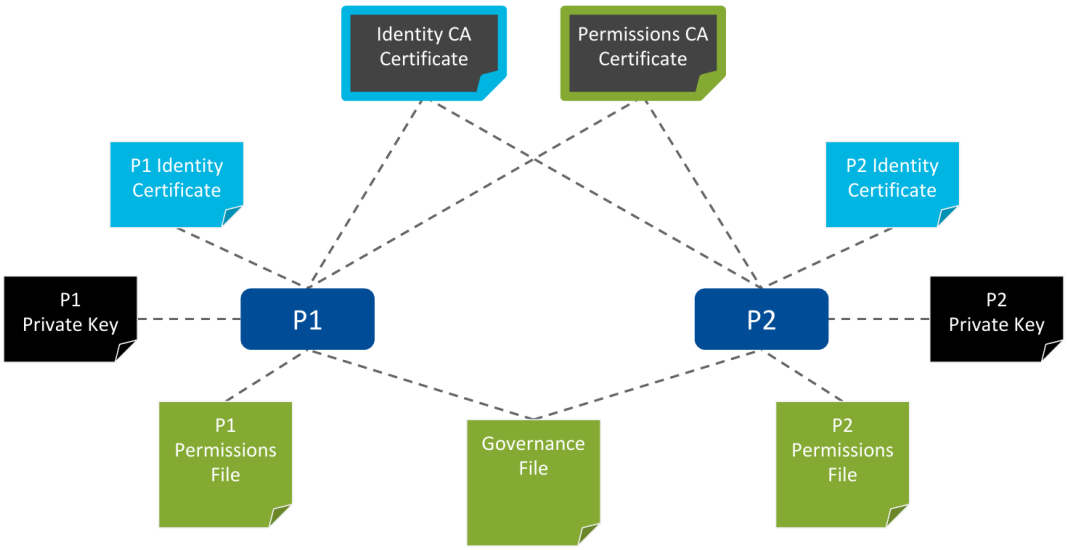


Figure 3.1: Two *DomainParticipants* (**P1** and **P2**) using the SECURITY PLUGINS, along with all the security artifacts needed for communication.

Chapter 4

Hands-On 1: Securing Connext DDS Applications

Suppose you are the cybersecurity expert in a company called Patient Monitoring Innovations (PMI). In this exercise, we will secure a *Connext* project created with *RTI Code Generator* (**`rtiddsgen`**).

Tip: For a brief introduction to *RTI Code Generator*, see Run Code Generator, in the Introduction to Publish/Subscribe. Full details are in the RTI Code Generator User's Manual.

As a first step, we will use some security artifacts provided with *Connext*. The applications in your project will use these artifacts to provide the required configuration and credentials to the Security Plugins, including the identity and permissions of the *DomainParticipants*. Then we will run the applications to check that we have communication. Finally, we will verify that eavesdropping is not possible with the configured level of security.

4.1 Generating a Connext DDS Project

1. Run **`rtisetenv_<architecture>`** in a new command prompt window, to avoid issues with paths and licensing.

   Where **`<architecture>`** depends on your target machine (where you will deploy your completed application). Architecture strings are listed in the RTI Connext DDS Core Libraries Platform Notes. Examples are **`x64Win64VS2017`** and **`x64Linux4gcc8.5.0`**.

   (See Set Up Environment Variables, in Introduction to Publish/Subscribe.)

2. Generate an example from an IDL file. You may want to use the following type definition for this example project:

   Listing 4.1: Sample contents of PatientMonitoring.idl

```
struct PatientMonitoring {
    string<128> patient_condition;
};
```

3. Put the IDL file in a directory called `patient_monitoring_project`.
4. Run *RTI Code Generator* (**`rtiddsgen`**) from that directory:

   Linux

```
$ cd patient_monitoring_project
$ rtiddsgen -example <architecture> -language C++11 PatientMonitoring.idl
```

   macOS

```
$ cd patient_monitoring_project
$ rtiddsgen -example <architecture> -language C++11 PatientMonitoring.idl
```

   Windows

```
> cd patient_monitoring_project
> rtiddsgen -example <architecture> -language C++11 -ppDisable␣
↪PatientMonitoring.idl
```

Note: We'll use `patient_monitoring_project` as the working directory. Unless otherwise indicated, we'll run all commands from this directory.

The generated example will be composed of the following files:

Table 4.1: Generated Files

Files	Description

`PatientMonitoring.cxx`
`PatientMonitoring.hpp`
`PatientMonitoringPlugin.cxx`
`PatientMonitoringPlugin.hpp`
	Support for your types in C++

`PatientMonitoring_publisher.cxx` `PatientMonitoring_subscriber.cxx`	Example publisher/subscriber application. Contains a *DomainParticipant* with a single *DataWriter*/*DataReader* for the first type defined in `PatientMonitoring.idl`.
Makefiles and Visual Studio® project files (for Windows applications)	Architecture-dependent build files
`USER_QOS_PROFILES.xml`	QoS profiles
`README_<architecture>.txt`	See this README for instructions on how to open and modify the files.

Our goal is to secure the generated publisher and subscriber applications (Alice and Bob, respectively, using the example from the previous sections).

4.2 Adding Security Artifacts to Your Project

We'll rely on some files from the `examples` directory in `rti_workspace` to define the security setup (see *Paths Mentioned in Documentation*). These files will provide the applications you'll build with the security configuration and credentials required by the SECURITY PLUGINS.

For convenience, copy these files into your project's top level:

Linux

```
$ cp -r <path to examples>/dds_security/* ./
```

Alternatively, copy/paste all the directories in `<path to examples>/dds_security/` to `patient_monitoring_project` (the working directory).

macOS

```
$ cp -r <path to examples>/dds_security/* ./
```

Alternatively, copy/paste all the directories in `<path to examples>/dds_security/` to `patient_monitoring_project` (the working directory).

Windows

```
> robocopy /E <path to examples>\dds_security\ .
```

Alternatively, copy/paste all the directories in `<path to examples>dds_security` to `:file:`patient_monitoring_project` (the working directory).

Note: These certificates and configuration XML files are provided only as examples and should not be used in production. These files are different between host platforms. If you are trying to communicate examples using different host platforms (e.g., Linux and Windows), you would need to copy over the files from one platform to the other so they are the same.

4.3 Enabling Security in Your QoS Profiles

Now modify the QoS profiles (XML) so that the applications will load the security libraries and the required security artifacts. To do so, define a profile named **Alice** in your `USER_QOS_PROFILES.xml` by replacing the existing default profile with the following sample profile:

Listing 4.2: Sample QoS profile with security enabled.

```
<qos_profile name="Alice" base_name="BuiltinQosLib::Generic.Security" is_
↪default_qos="true">
    <domain_participant_qos>
        <property>
            <value>
                <!-- Certificate Authorities -->
                <element>
```

```
                    <name>dds.sec.auth.identity_ca</name>
                    <value>file:./cert/ecdsa01/ca/ecdsa01RootCaCert.pem</
↪value>
                </element>
                <element>
                    <name>dds.sec.access.permissions_ca</name>
                    <value>file:./cert/ecdsa01/ca/ecdsa01RootCaCert.pem</
↪value>
                </element>
                <!-- Participant Public Certificate and Private Key -->
                <element>
                    <name>dds.sec.auth.identity_certificate</name>
                    <value>file:./cert/ecdsa01/identities/ecdsa01Peer01Cert.
↪pem</value>
                </element>
                <element>
                    <name>dds.sec.auth.private_key</name>
                    <value>file:./cert/ecdsa01/identities/ecdsa01Peer01Key.
↪pem</value>
                </element>
                <!-- Signed Governance and Permissions Documents -->
                <element>
                    <name>dds.sec.access.governance</name>
                    <value>file:./xml/signed/signed_Governance.p7s</value>
                </element>
                <element>
                    <name>dds.sec.access.permissions</name>
                    <value>file:./xml/signed/signed_PermissionsA.p7s</value>
                </element>
            </value>
        </property>
    </domain_participant_qos>
</qos_profile>
```

This profile inherits from the builtin `BuiltinQosLib::Generic.Security` profile. It tells your *DomainParticipant* to use the *Builtin Security Plugins* from the *RTI® Security Plugins*. (For further details, see Properties for Enabling Security in the *RTI Security Plugins User's Manual*.) Note that not indicating a Security Plugins library will result in a working but unsecure application.

The profile also configures the following properties:

- `dds.sec.auth.identity_ca` Configures the Identity CA certificate, used to verify that Identity Certificates from other *DomainParticipants* are legitimate. Usually, all the participants in your secure system will load the same Identity CA certificate.[1]
- `dds.sec.access.permissions_ca` Configures the Permissions CA certificate, used to verify that the Governance and Permissions Documents are legitimate. All the participants in your secure system must load the same Permissions CA certificate.[2]

[1] Depending on your use case, different *DomainParticipants* may trust a different Identity CA, for example, when several intermediate CAs exist in your PKI. For further details, see Alternative CAs in the *RTI Security Plugins User's Manual*.

[2] In systems where multiple Permissions Certificate Authorities may exist, you can use the `access_control.alternative_permissions_cas` property to specify alternative Permissions CA certificates. For further details, see Governance and Permissions in the *RTI Security Plugins User's Manual*.

- `dds.sec.auth.identity_certificate` Configures the Identity Certificate for the local *DomainParticipant*. Every secure participant should have its own unique Identity Certificate, signed by the Identity CA.
- `dds.sec.auth.private_key` Configures the Private Key for the local *DomainParticipant*, used during the authentication process. Every secure participant should have its own unique Private Key, only known to the local participant.
- `dds.sec.access.governance` Configures the Governance Document, which defines how your system is protected. All the participants in your secure system must load the same Governance Document[3], signed by the Permissions CA.
- `dds.sec.access.permissions` Configures the Permissions Document for the local *DomainParticipant*, which specifies what it is allowed to do in every *Secure Domain*. Every secure participant should have its own Permissions Document, signed by the Permissions CA.

Important: A participant that is not configured to use a Security Plugins library will omit any security-related properties, resulting in a working but unsecure *DomainParticipant*. Make sure your QoS profile inherits from the builtin `BuiltinQosLib::Generic.Security` profile or refer to Building and Running Security Plugins-Based Applications in the *RTI Security Plugins User's Manual*.

In later hands-on exercises, you will learn how to generate all the files listed above. For now, we'll use the example files you copied in the previous step.

4.4 Linking Your Applications Against Security Plugins and OpenSSL Libraries

So far, you have generated a *Connext* project and configured its QoS profiles to load the required security artifacts and enable the Security Plugins. Now we will build the applications, so you can see them in action. For convenience, we will link the applications dynamically to both *Connext* and the OpenSSL libraries (for other linking options, see Building and Running Security Plugins-Based Applications in the *RTI Security Plugins User's Manual*.

Note: We use `NDDSHOME` to refer to the path where the *Connext* bundle is installed in your system (in *Connext* 7.7.0, `NDDSHOME` is the path to rti_connext_dds-7.7.0). The installation process is described in the RTI Security Plugins Installation Guide[4].

---

[3] If not the same, at least Governance Documents loaded by different *DomainParticipants* need to be compatible. For further information, see Governance Document in the *RTI Security Plugins User's Manual*.

[4] In the free version (with `lm` in the bundle name), OpenSSL is installed automatically when you install the *Connext* host bundle. After installation, OpenSSL will be in `<installdir>/third_party/openssl-<version>`.

4.4.1 Building the Application

Linux

Use the generated Makefile with the `SHAREDLIB` variable set to `1` to force dynamic linking; set `DEBUG=1` to build in debug mode:

```
$ make -f makefile_PatientMonitoring_<architecture> SHAREDLIB=1 DEBUG=1
```

macOS

Use the generated Makefile with the `SHAREDLIB` variable set to `1` to force dynamic linking; set `DEBUG=1` to build in debug mode:

```
$ make -f makefile_PatientMonitoring_<architecture> SHAREDLIB=1 DEBUG=1
```

Windows

We will build the solution generated by *RTI Code Generator* with the Debug DLL configuration. You can do this by using the **`msbuild`** command-line tool provided with Visual Studio. To do this, open a Visual Studio command prompt and change to the `patient_monitoring_project` directory (`cd patient_monitoring_project`). Then build the Debug DLL solution, which is configured for dynamic linking:

```
> msbuild /property:Configuration="Debug DLL" PatientMonitoring-<architecture>
↪.sln
```

Alternatively, you can open `PatientMonitoring-<architecture>.sln` with Visual Studio, as shown in the following screenshot. If you choose this method, select the **Debug DLL** option in the *Solution Configurations* dropdown menu (1), then select **Build** for both **PatientMonitoring_publisher** and **PatientMonitoring_subscriber** (2).

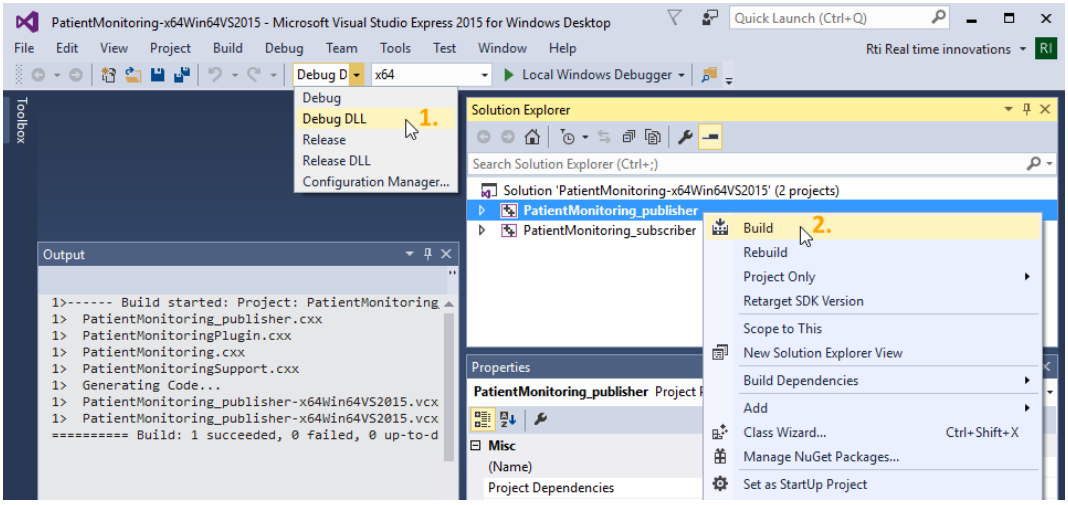


Figure 4.1: Building the application with Visual Studio 2015.

4.5 Running the Applications

It's time to see your secure publisher and subscriber working together! Open two command prompts and configure the environment in both to point to the location of the dynamic libraries, as explained below.

4.5.1 Configuring the Environment in Both Command Prompts

1. Configure the environment for *Connext* by running **`rtisetenv_<architecture>`** from the *Connext* installation path.

   For details, see Set Up Environment Variables, in Introduction to Publish/Subscribe.

   This will update your **PATH** to include the location of the *Connext* binaries.

2. Update the shared library environment variable to include the directory where the OpenSSL libraries reside. For this example, you may want to use OpenSSL debug libraries.

   Linux

   ```
   $ export LD_LIBRARY_PATH=$NDDSHOME/third_party/openssl-3.5.5/
   ↪<architecture>/debug/lib:$LD_LIBRARY_PATH
   ```

   macOS

   ```
   $ export DYLD_LIBRARY_PATH=$NDDSHOME/third_party/openssl-3.5.5/
   ↪<architecture>/debug/lib:$DYLD_LIBRARY_PATH
   ```

   Windows

   ```
   > set PATH=%NDDSHOME%\third_party\openssl-3.5.5\<architecture>\
   ↪debug\bin;%PATH%
   ```

   If you run the applications from within Visual Studio, you need to set up the environment before opening the solution in Visual Studio.

Tip: Make sure you set the environment in *both* command prompts.

4.5.2 Checking Communication

1. Run the publisher in one of the command prompts and the subscriber in the other.

   Do not provide any arguments to the applications, so that they communicate in *Domain 0*. This will be your *Secure Domain*.

2. You should see the message **Received data** in the subscriber's command prompt, which indicates successful communication:

   Linux

   Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

macOS

Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

Windows

Publisher:

```
> objs\<architecture>\PatientMonitoring_publisher.exe
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
> objs\<architecture>\PatientMonitoring_subscriber.exe
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

Congratulations! You have added Security Plugins to your applications.

4.6 Checking that Your Applications Communicate Securely

To check that communication is actually secure, now let's have a non-secure *Connext* application try to subscribe to the *PatientMonitoring Topic*. For convenience, we will use *RTI Admin Console* as the non-secure subscriber.

4.6.1 Verifying that Eavesdropping Attempts are Frustrated

1. Open *Admin Console* and join *Domain 0*.

   (For details on using *Admin Console*, see Viewing Your Data, in Introduction to Publish/Subscribe.)

2. Notice that your secure *DomainParticipants* do not show up in *Admin Console*.

   The security configuration of your *Secure Domains* requires *DomainParticipants* to authenticate in order to join the *Domain*. Any participant with wrong or no credentials will fail to authenticate and will be disallowed from communicating in the *Domain*. In this example, the Governance Document you copied into the project (and added to Alice's XML QoS configuration) protects the endpoint discovery with encryption. As a result, intruders and eavesdroppers will not have any information regarding the *Topics* in your secure system. We will show the configuration values to force authentication and to enable encryption in *Hands-On 2: Defining Your System's Security Requirements*.

4.6.2 Detecting Eavesdropping Attempts

When *Admin Console* runs in *Domain 0*, it sends discovery information to all other *DomainParticipants* in the same *Domain*. When your secure participants receive this information, they will require *Admin Console* to authenticate. However, *Admin Console* does not hold a valid credential and cannot authenticate, making discovery fail and preventing further communication.

You might want to detect that an unauthorized *DomainParticipant* tried to communicate in your secure system. To do this:

1. Modify your QoS profile to increase the verbosity of the Logging Plugin:

```
<qos_profile name="Alice" base_name="BuiltinQosLib::Generic.Security" is_
↪default_qos="true">
    <domain_participant_qos>
        <property>
            <value>
                <!-- Logging Plugin setup -->
                <element>
                    <name>com.rti.serv.secure.logging.verbosity</name>
                    <value>WARNING</value>
                </element>
                ...
```

2. Rerun your secure publisher or/and subscriber.

   A message like this should show up in its command prompt:

```
PRESParticipant_getRemoteParticipantInitialSecurityState:unauthenticated␣
↪remote participant 1018db2.74f9482c.2712840d denied
```

   This message indicates a failure during the authentication of a remote *DomainParticipant*. It will be logged every time an unauthenticated participant attempts to join a *Secure Domain* where `allow_unauthenticated_participants` is set to `FALSE`.

4.7 Further Exercises

So far, we have defined a single QoS profile named Alice to enable the Security Plugins. Since this profile has the attribute `is_default_qos=true`, any *Connext* application loading `USER_QOS_PROFILES.xml` will have the same security artifacts. This makes it convenient to load the same Governance Document and CA certificates in an example. However, all your applications will have the same identity and permissions, which is strongly discouraged. Please note that the authentication succeeds in this situation since the *DomainParticipants* still have valid credentials, i.e., the Identity CA has issued all the identities. Also note that the Permissions Document the publisher and the subscriber are loading gives them permission to publish and subscribe to any *Topic*, making communication possible.

4.7.1 Give Different Credentials to Each Application in Your System

1. To maximize security, different applications should always have different identities and permissions. Write a second QoS profile for your project in the same XML QoS file. You should end up with the following:

   - A profile called `Alice` for the publisher application
   - A profile called `Bob` for the subscriber application

Tip: You may want to have Bob's profile inherit from Alice's, with `base_name="Alice"`. This way, you don't need to specify the CA certificates and the Governance Document twice in `USER_QOS_PROFILES.xml`.

---

2. Modify Bob's identity and permissions.

   For example, use `ecdsa01Peer02Cert.pem`, `ecdsa01Peer02Key.pem`, and `signed_PermissionsB.p7s`.

   Note that `ecdsa01Peer02Key.pem` is password-protected, so use the `dds.sec.auth.password` property to specify the password in Bob's QoS profile:

```
<qos_profile name="Bob" base_name="Alice">
    <domain_participant_qos>
        <property>
            <value>
                <!-- Participant Public Certificate and Private Key -->
                <element>
                    <name>dds.sec.auth.identity_certificate</name>
                    <value>file:./cert/ecdsa01/identities/
↪ecdsa01Peer02Cert.pem</value>
                </element>
                <element>
                    <name>dds.sec.auth.private_key</name>
                    <value>file:./cert/ecdsa01/identities/
↪ecdsa01Peer02Key.pem</value>
                </element>
                <element>
                    <name>dds.sec.auth.password</name>
                    <value>VG9tQjEy</value>
                </element>
                <!-- Signed Permissions Document -->
                <element>
                    <name>dds.sec.access.permissions</name>
                    <value>file:./xml/signed/signed_PermissionsB.p7s</
↪value>
                </element>
            </value>
        </property>
    </domain_participant_qos>
</qos_profile>
```

3. Modify the applications' source code in `PatientMonitoring_publisher.cxx` and `PatientMonitoring_subscriber.cxx` to use these QoS profiles. The publisher's *DomainParticipant* should use the `Alice` QoS profile, and the subscriber's *DomainParticipant* should use the `Bob` QoS profile.

   To do so, use the default `QosProvider` to make your *DomainParticipants* load their corresponding profiles (see QosProvider in the Modern C++ API Reference). Note that you will need to include `ddscore.hpp`.

```
#include <dds/core/ddscore.hpp>
...

// Load Bob's QoS
dds::domain::qos::DomainParticipantQos bob_qos =
     dds::core::QosProvider::Default().participant_qos(
↪"PatientMonitoring_Library::Bob");
```

```
// Create a DomainParticipant with Bob's QoS
dds::domain::DomainParticipant participant(
     domain_id,
     bob_qos);
```

4. Rebuild your project and run your publisher and subscriber applications. You should see communication.

4.8 Troubleshooting

- When I try to run *Code Generator* (**rtiddsgen**), my system does not recognize or find the command.

  Make sure *Connext* is installed correctly.

  Also make sure you've correctly set the **PATH** environment variable (your **PATH** should include <NDDSHOME>/bin). You can run **rtisetenv_<architecture>** to add the location of the *Connext* binaries to your **PATH** (see Set Up Environment Variables, in Introduction to Publish/Subscribe).

- When I run *Code Generator* (**rtiddsgen**), I get this error:

  ```
  The preprocessor 'CL.EXE' cannot be found in your path.
  ```

  Make sure your toolchain's preprocessor is available. If you are using Visual Studio, make sure you are using a Visual Studio command prompt. Alternatively, run **rtiddsgen** with the **-ppDisable** option.

- When I run *Code Generator* (**rtiddsgen**), I get warnings stating:

  ```
  File exists and will not be overwritten
  ```

  Some files that would normally be generated already exist and will not be overwritten. These errors are expected.

- When I run the publisher/subscriber, I get this error:

  ```
  error while loading shared libraries: libnddscpp.so
  ```

  Make sure the shared library environment variable includes the directory where the *Connext* libraries reside. See *Configuring the Environment in Both Command Prompts* for a solution.

- When I run the publisher/subscriber, I get this error:

  ```
  !open library=libnddssecurity.so
  ```

  - Make sure the OpenSSL target bundle provided by RTI is installed correctly.[Page 16, 4]
  - Make sure the shared library environment variable includes the directory where OpenSSL libraries reside. See *Configuring the Environment in Both Command Prompts* for a solution.
  - Make sure your environment is pointing to the correct version of the OpenSSL libraries.

- When I run the publisher and subscriber, communication does not occur.

  - Make sure you run both applications from the `patient_monitoring_project` directory, so they load the same `USER_QOS_PROFILES.xml` file. Note: If the project file for your IDE was auto-generated and you are running from within your IDE, it should run from that directory automatically.
  - Make sure your applications are loading the right security artifacts or artifacts combinations (that is, you don't get a `!certificate verify fail` error).
  - Be careful with Bob's QoS profile depending on Alice's — you may be changing properties in Alice's profile and loading Bob's, or the opposite.

- When I run the publisher/subscriber, I get this error:

```
!certificate verify fail
```

  Make sure your applications are loading the right security artifacts or artifacts combinations.

Chapter 5

Hands-On 2: Defining Your System's Security Requirements

In this Hands-On, we will define the security requirements for your project, expressing them in the form of a **Governance Document**. We will sign this Governance Document with the provided Permissions CA. Lastly, we will tell your secure participants where to find the new Governance Document and we will see how the new security requirements are applied to your system.

Note: We will use the **OpenSSL CLI** to perform the security operations in the generation of the security artifacts. Make sure to include in the path your OpenSSL binary directory[8]. The installation process is described in the RTI Security Plugins Installation Guide.

Note that the Security Plugins do not depend on OpenSSL to generate these artifacts; you can use the security toolkit of your choice. With that said, we recommend using OpenSSL to make sure that the certificates are in the right format.

5.1 Specifying the Security Requirements

If you completed *Hands-On 1: Securing Connext DDS Applications*, you should have two applications using Security Plugins to communicate securely. But what does "securely" actually mean? What kind or what level of security is being applied? As mentioned in *Introduction to Security Plugins*, the answers to these questions are in the Governance Document, which defines the security rules that every *DomainParticipant* in your *Secure Domain* needs to follow. We will now focus on writing a Governance Document to specify your project's security requirements.

Governance Documents define two levels of rules that can be configured:

- **Domain-level rules**, which affect participants in the *Domain*;
- **Topic-level rules**, which affect *Endpoints* (*DataReaders* and *DataWriters*) for that *Topic*.

[8] In the free version (with `lm` in the bundle name), OpenSSL is installed automatically when you install the *Connext* host bundle. After installation, OpenSSL will be in `<installdir>/third_party/openssl-<version>`.

You will find a description of the currently available rules in the tables below. For more information, see these sections in the *RTI Security Plugins User's Manual*:

- Related Governance Rules for Authentication
- Related Governance Rules for Access Control
- Related Governance Rules for Cryptography

Table 5.1: Domain-Level Rules

Rule	Description	Possible values
`allow_unauthenticated_participants`	Determines if a secure *DomainParticipant* is allowed to match a participant that is not able to successfully complete the authentication process. By disallowing unauthenticated participants, we prevent them from publishing or subscribing to *Topics* in our *Secure Domain*.[1]	`TRUE` or `FALSE`
`enable_join_access_control`	Determines if the participant-level permissions configured in the Permissions Document are enforced for remote participants.	`TRUE` or `FALSE`
`discovery_protection_kind`	Configures the Discovery Protection, determining what level of protection is applied to the Builtin Secure Discovery Topics.	`ENCRYPT`, `ENCRYPT_WITH_ORIGIN_AUTHENTICATION`, `SIGN`, `SIGN_WITH_ORIGIN_AUTHENTICATION`, `NONE`
`liveliness_protection_kind`	Configures the Liveliness Protection, determining what level of protection is applied to the Builtin Secure Liveliness Topic.	`ENCRYPT`, `ENCRYPT_WITH_ORIGIN_AUTHENTICATION`, `SIGN`, `SIGN_WITH_ORIGIN_AUTHENTICATION`, `NONE`
`rtps_protection_kind`	Configures the RTPS Protection, determining what level of protection is applied to RTPS messages.	`ENCRYPT`, `ENCRYPT_WITH_ORIGIN_AUTHENTICATION`, `SIGN`, `SIGN_WITH_ORIGIN_AUTHENTICATION`, `NONE`
`rtps_psk_protection_kind`	Determines how to protect RTPS bootstrapping messages. Read section Domain-Level Rules in the *RTI Security Plugins User's Manual* for more information.	`ENCRYPT`, `SIGN`, `NONE`

continues on next page

Table 5.1 – continued from previous page

Rule	Description	Possible values
`allowed_security_algorithms`	Determines the cryptographic algorithms that are allowed in your system.	The values are a list of algorithms that depend on whether they are used for `digital_signature`, `digital_signature_identity_trust_chain`, `key_establishment`, or `symmetric_cipher`. For more information read section Domain-Level Rules in the *RTI Security Plugins User's Manual*.

Table 5.2: Topic-Level Rules

Rule	Description	Possible values
`enable_discovery_protection`	Determines if discovery information updates related to *Endpoints* from this *Topic* will be sent with the level of security defined in the `discovery_protection_kind` domain-level rule.	`TRUE` or `FALSE`
`enable_liveliness_protection`	Determines if liveliness updates related to *Endpoints* from this *Topic* will be sent with the level of security defined in the `liveliness_protection_kind` domain-level rule.	`TRUE` or `FALSE`
`enable_read_access_control`	Determines if endpoint-level permissions configured in the Permissions Document are enforced for local and remote *DataReaders*.	`TRUE` or `FALSE`
`enable_write_access_control`	Determines if endpoint-level permissions configured in the Permissions Document are enforced for local and remote *DataWriters*.	`TRUE` or `FALSE`
`metadata_protection_kind`	Configures the Submessage Protection, determining what level of protection is applied to RTPS submessages from *Endpoints* of the associated *Topic*.	`ENCRYPT`, `ENCRYPT_WITH_ORIGIN_AUTHENTICATION`, `SIGN`, `SIGN_WITH_ORIGIN_AUTHENTICATION`, `NONE`

continues on next page

[1] A system may allow unauthenticated participants as a way of combining older, unsecured applications with newer secure applications (not recommended, see Using Separate Domains for Secure and Unsecure Participants in the *RTI Security Plugins User's Manual*).

Table 5.2 – continued from previous page

Rule	Description	Possible values
`data_protection_kind`	Configures the Serialized Data Protection, determining what level of protection is applied to the serialized payload from *DataWriters* of the associated *Topic*.	`ENCRYPT`, `SIGN`, `NONE`

5.2 Composing a Governance Document with the Security Requirements

As the DDS Security expert at Patient Monitoring Innovations (PMI), you are going to specify the security requirements of your system in a file called `pmiGovernance.xml`.

Create `pmiGovernance.xml` in the `xml` directory (along with the XML files we copied from the *Connext* examples) and add the following content:

Listing 5.1: Sample Governance Document that applies to all the *Domains*. Different Topics will have a different kind of protection.

```
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://community.rti.com/schema/7.3.0/dds_
↪security_governance.xsd">
  <!-- Rules affecting different domains are defined under this tag -->
  <domain_access_rules>
    <domain_rule>
      <!-- 1. This determines when to apply this rule. In this case, any␣
↪domain -->
      <domains>
        <id_range>
          <min>0</min>
        </id_range>
      </domains>

      <!-- 2. The following fields determine behavior of
       DomainParticipants matching this rule -->
      <allow_unauthenticated_participants>TRUE</allow_unauthenticated_
↪participants>
      <enable_join_access_control>FALSE</enable_join_access_control>
      <discovery_protection_kind>ENCRYPT</discovery_protection_kind>
      <liveliness_protection_kind>ENCRYPT</liveliness_protection_kind>
      <rtps_protection_kind>NONE</rtps_protection_kind>

      <!-- 3. Rules affecting topics are defined under this tag -->
      <topic_access_rules>
        <!-- 3.1 Let's have a rule for all topics -->
        <topic_rule>
          <!-- This determines when to apply the rule -->
          <topic_expression>*</topic_expression>
```

```
          <!-- The following fields determine the behavior of
           topics/endpoints matching this rule -->
          <enable_discovery_protection>FALSE</enable_discovery_protection>
          <enable_liveliness_protection>FALSE</enable_liveliness_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>ENCRYPT</data_protection_kind>
        </topic_rule>

        <!-- 3.2 Later we will define other topic rules for specific topics --
↪>

      </topic_access_rules>
    </domain_rule>
  </domain_access_rules>
</dds>
```

Note: The following references to (1), (2), etc. correspond to comments in the above XML.

This Governance Document defines just one configuration, to be applied to any *Domain* in the system (1). Consequently, all the *Domains* and *Topics* in the system are protected in the same way. In particular, the following rules are defined (2):

Domain rule (`domain_rule`)	Value	Security implications
`allow_unauthenticated_participants`	`TRUE`	Non-authenticated participants are allowed to publish/subscribe unprotected *Topics*
`enable_join_access_control`	`FALSE`	Remote participant-level permissions are not checked
`discovery_protection_kind`	`ENCRYPT`	Endpoint Discovery will be protected with encryption for *Topics* setting `enable_discovery_protection` to `TRUE`[2]
`liveliness_protection_kind`	`ENCRYPT`	Liveliness assertions will be protected with encryption for *Topics* setting `enable_liveliness_assertion` to `TRUE`[3]
`rtps_protection_kind`	`NONE`	RTPS messages are sent without any additional protection (required to allow unauthenticated participants)

Then we can define different levels of protection depending on the *Topic* to protect (3). This Governance specifies a single rule that applies to every *Topic* and protects the user's data with encryption (3.1). We will define the protection of the *PatientMonitoring Topic* (3.2) as part of a later exercise.

[2] Enabling Discovery Protection has further implications, as described in discovery_protection_kind (domain_rule).
[3] Enabling Liveliness Protection has further implications, as described in discovery_protection_kind (domain_rule).

5.3 Signing the Governance Document

As mentioned in *Securing a DDS Domain*, both Governance and Permissions Documents must be signed by the Permissions CA. This way, all *DomainParticipants* trusting that Permissions CA can make sure that the Governance Document was not forged by an attacker.

We will use the provided Permissions CA's certificate and key to sign the Governance Document that we composed.[4]

Run the command below to create the signed Governance Document (with PKCS#7 format) named `xml/signed/signed_pmiGovernance.p7s`:

Linux

```
$ openssl cms -sign -in xml/pmiGovernance.xml -text -out xml/signed/signed_
↪pmiGovernance.p7s -signer cert/ecdsa01/ca/ecdsa01RootCaCert.pem -inkey cert/
↪ecdsa01/ca/private/ecdsa01RootCaKey.pem
```

macOS

```
$ openssl cms -sign -in xml/pmiGovernance.xml -text -out xml/signed/signed_
↪pmiGovernance.p7s -signer cert/ecdsa01/ca/ecdsa01RootCaCert.pem -inkey cert/
↪ecdsa01/ca/private/ecdsa01RootCaKey.pem
```

Windows

```
> openssl cms -sign -in xml\pmiGovernance.xml -text -out xml\signed\signed_
↪pmiGovernance.p7s -signer cert\ecdsa01\ca\ecdsa01RootCaCert.pem -inkey cert\
↪ecdsa01\ca\private\ecdsa01RootCaKey.pem
```

5.4 Updating the QoS Profiles in Your Project

Now we need to update `USER_QOS_PROFILES.xml` to make your *DomainParticipants* load the new Governance Document. Replace the value of the `dds.sec.access.governance` property as follows:

```
...
<!-- Signed Governance and Permissions Documents -->
<element>
    <name>dds.sec.access.governance</name>
    <value>file:./xml/signed/signed_pmiGovernance.p7s</value>
</element>
...
```

Here, the `file:` prefix means that the signed Governance Document will be loaded from the specified path in the file system. Note that the path is relative to the working directory from which you run your application (unless you specify an absolute path).

[4] In this example, we have control of the Permissions CA. This is not always the case and we may be required to send the Governance Document to an external entity to get it signed.

Another option is to set the value of the `dds.sec.access.governance` property to the contents of the Governance Document. You can do that by using the `data:,` prefix. This is useful when your application does not have access to a file system. For details, see DDS Security Properties for Configuring Access Control in the *RTI Security Plugins User's Manual*.

5.5 Checking that the Specified Security Rules Are Applied

It's time to see the changes in your security requirements! We will start by verifying that your *DomainParticipants* can communicate when they load the new Governance Document.

5.5.1 Verifying Communication

1. Run your publisher and subscriber as explained in *Running the Applications*.

   Note that building the applications is not required, but you still may have to set up your environment.

2. You should see the message "Received data" on the subscriber side, which indicates that it received samples from the publisher.

   Linux

   Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

   Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

   macOS

   Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
```

```
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

Windows

Publisher:

```
> objs\<architecture>\PatientMonitoring_publisher.exe
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
> objs\<architecture>\PatientMonitoring_subscriber.exe
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

5.5.2 Checking the New Security Rules

So far, your Governance Document protects the privacy of the user's data by encrypting the messages' payload. However, it does not protect discovery data, allowing unauthenticated participants to receive this information. To verify that these rules are applied, we'll try to subscribe to the *PatientMonitoring Topic* with *RTI Admin Console*.

1. Open *Admin Console* and join *Domain 0*.

   (For details on using *Admin Console*, see Viewing Your Data, in Introduction to Publish/Subscribe.)

Your secure participants should show up in the *DDS Logical View* window.

2. Right-click on the *Example PatientMonitoring Topic* and select *Subscribe…*.
3. A dialog will prompt you to select the data type; click *OK*.
4. Go to the *Data Visualization* perspective — by default, a dialog will give you the option to switch perspectives.
5. Notice that *Admin Console*'s subscriber is unable to receive any data samples, while your secure subscriber is receiving them.

At this point, you can be sure[5] that your applications are using the security rules you have defined. Congratulations!

5.6 Further Exercises

The Governance Document we defined allows unauthenticated participants to join the *Secure Domains* and to receive discovery data. This may be acceptable for some *Topics* or some systems. However, you may have more restrictive security requirements and want to prevent unauthenticated participants from communicating at all. Perhaps your security requirements may vary from one *Topic* to another. We will address these two issues in the following exercises.

Note: Defining the security requirements for a real system is not a trivial task. If you plan to deploy a secure system, your organization will need an in-house security expert to define the security requirements your system needs.

5.6.1 Protecting the Domain

Now, we want to disallow unauthenticated participants from performing any kind of communication in your *Secure Domains*.

To do so, modify your Governance Document to meet the following domain-level rules:

Domain rule (`domain_rule`)	Value	Security implications
`allow_unauthenticated_participants`	`FALSE`	Only authenticated participants are allowed in the system
`enable_join_access_control`	`TRUE`	Permissions are checked for any discovered *DomainParticipant*
`rtps_protection_kind`	`SIGN`	All RTPS messages in the system are signed

[5] In *Hands-On 5: Checking that Your DDS Traffic Is Protected* we will use Wireshark to verify that the messages from the publisher application are encrypted on the network.

Make sure the Permissions CA signs the modified Governance Document (see *Signing the Governance Document*); otherwise the changes will not be applied.

After applying this configuration, only authenticated and authorized participants will be allowed to join the system. Since RTPS messages are signed, only authenticated and authorized participants will be allowed to write messages to the system.

5.6.2 Adding a Topic Rule for the PatientMonitoring Topic

Your Governance may define different levels of protection, depending on the *Topic* to be protected.

Write a second *Topic* rule (`topic_rule`) to protect the *Example PatientMonitoring Topic* as follows:

Topic rule (`topic_rule`)	Value	Security implications
`enable_discovery_protection`	`TRUE`	Endpoint discovery data is protected (encrypted, as specified by `discovery_protection_kind`)
`enable_liveliness_protection`	`TRUE`	Liveliness assertions are protected[6] (encrypted, as specified by `liveliness_protection_kind`)
`enable_read_access_control`	`TRUE`	Enforce local endpoint-level permissions on locally created *DataReaders*; enforce remote endpoint-level permissions on remotely discovered *DataReaders*
`enable_write_access_control`	`TRUE`	Enforce local endpoint-level permissions on locally created *DataWriters*; enforce remote endpoint-level permissions on remotely discovered *DataWriters*
`metadata_protection_kind`	`ENCRYPT`	*DataWriters*' and *DataReaders*'s outgoing submessages are encrypted[7]
`data_protection_kind`	`ENCRYPT`	Payloads are encrypted

The Governance Document is parsed from top to bottom (for more information, see How the Governance Document is Interpreted in the *RTI Security Plugins User's Manual*. Therefore, you must add your new `topic_rule` before the wildcard rule that matches all topics (the one with a `topic_expression` value of `*`).

After applying the configuration, *PatientMonitoring* updates are exchanged encrypted, so an eavesdropper will not be able to have access to that data. Make sure the Permissions CA signs the modified Governance Document (see *Signing the Governance Document*); otherwise the changes will not be applied.

[6] The value of this attribute matters only if the *DataWriter*'s LIVELINESS QosPolicy is `AUTOMATIC` or `MANUAL_BY_PARTICIPANT`. See enable_liveliness_protection (topic_rule) in the *RTI Security Plugins User's Manual*.

[7] These submessages include, but are not limited to, DATA, HEARTBEAT, ACKNACK, and GAP. For more information, see Submessage Protection in the *RTI Security Plugins User's Manual*.

5.7 Troubleshooting

- When I run **`openssl cms`**, I get this error:

```
WARNING: can't open config file: <default openssl builtin path>/openssl.
↪cnf
```

  Set the environment variable `OPENSSL_CONF` to `./cert/openssl.cnf`.

Chapter 6

Hands-On 3: Defining the DomainParticipant Permissions

In this Hands-On, we will specify what the *DomainParticipants* in your project will be allowed to do within your secure system. We will achieve this by writing a Permissions Document for each of your participants.

Going back to our example, Alice will only have permission to publish the *PatientMonitoring* data, while Bob will be restricted to subscribe to patient-related *Topics*. We will associate each Permissions Document with its corresponding participant based on their identities, limiting the system damage if one of your applications becomes compromised. To put the Permissions Documents into effect, we will sign them with the provided Permissions CA. Finally, we will modify your secure participants' QoS profiles to point to the new Permissions Documents and see how the new permissions are applied.

Note: We will use the **OpenSSL CLI** to perform the security operations in the generation of the security artifacts. Make sure to include in the path your OpenSSL binary directory[3]. The installation process is described in the RTI Security Plugins Installation Guide.

Note that the Security Plugins do not depend on OpenSSL to generate these artifacts; you can use the security toolkit of your choice. With that said, we recommend using OpenSSL to make sure that the certificates are in the right format.

6.1 Granting Permissions to Your Secure Participants

As the DDS Security expert at Patient Monitoring Innovations (PMI), you are going to specify the permissions of every *DomainParticipant* in your system. You will define what *Topics* your publisher, Alice, can read/write by associating rules with the subject in her Identity Certificate.

Create a file called `pmiPermissionsAlice.xml` in the `xml` directory (along with the rest of XML files) and add the following content:

[3] In the free version (with `lm` in the bundle name), OpenSSL is installed automatically when you install the *Connext* host bundle. After installation, OpenSSL will be in `<installdir>/third_party/openssl-<version>`.

Listing 6.1: Sample Permissions Document configuring a single grant for *ParticipantAlice*

```
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://community.rti.com/schema/7.3.0/dds_
↪security_permissions.xsd">
  <permissions>
    <!-- Grants for a specific DomainParticipant will be grouped under this␣
↪tag -->
    <grant name="ParticipantAlice">
      <!-- 1. The rules below will apply to the DomainParticipant
       whose Identity certificate contains this subject name -->
      <subject_name>Alice's X.509 subject (see below)</subject_name>
      <!-- 2. Validity dates for this grant -->
      <validity>
        <!-- Format is CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm] in GMT -->
        <not_before>2019-10-31T13:00:00</not_before>
        <not_after>2029-10-31T13:00:00</not_after>
      </validity>

      <!-- 3. Allow this participant to publish the
       PatientMonitoring topic -->
      <allow_rule>
        <domains>
          <id>1</id>
        </domains>
        <publish>
          <topics>
            <topic>Example PatientMonitoring</topic>
          </topics>
        </publish>
      </allow_rule>

      <!-- 4. This participant will not be allowed to publish or
       subscribe to any other topic -->
      <default>DENY</default>
    </grant>
  </permissions>
</dds>
```

This Permissions Document configures a grant for *ParticipantAlice*, identified by its Identity Certificate (1). We will define the grant's subject name to make it apply to your publisher in *Binding the Permissions Document to Your DomainParticipants*. The validity of this grant is restricted in time (2). This grant has a single allow rule (3), so *ParticipantAlice* will only be allowed to:

- Join *Domain 1*
- Publish the *Topic* "Example PatientMonitoring" — only in *Domain 1*

ParticipantAlice will be denied attempts to perform any other action (4).

In addition to configuring *allow* rules, we can specify *deny* rules with the opposite effect. For details, see the

Permissions Document in the *RTI Security Plugins User's Manual*.

Note: Permissions Documents are always exchanged during authentication. The larger a Permissions Document is, the more network overhead it will cause. As such, we recommend that you keep separate Permissions Documents per identity (i.e., per *DomainParticipant*). For further details, see Choosing the Granularity of Your Permissions Documents for DomainParticipants in the *RTI Security Plugins User's Manual*.

6.2 Binding the Permissions Document to Your DomainParticipants

When a *DomainParticipant* loads a Permissions Document, it looks for a grant with a `subject_name` matching its identity. In other words, the `subject_name` identifies the *DomainParticipant* to which the grant's permissions apply. The contents of the `subject_name` tag should be the X.509 subject name for the *DomainParticipant*, as given in the **Subject** field of its Identity Certificate.

If you followed the steps in *Hands-On 1: Securing Connext DDS Applications*, Alice's Identity Certificate should be `cert/ecdsa01/identities/ecdsa01Peer01Cert.pem`. The subject of Alice's certificate does not match the `subject_name` section in grant *ParticipantAlice*. We will modify `pmiPermissionsAlice.xml` to make this grant apply to Alice.

1. Check the information from Alice's Identity Certificate (`ecdsa01Peer01Cert.pem`) with the following command:

   Linux

```
$ openssl x509 -in cert/ecdsa01/identities/ecdsa01Peer01Cert.pem -text -
↪noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7b:ba:b9:c9:2c:be:ee:b9:71:d0:62:4e:59:6b:de:89:3a:33:5c:ad
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=CA, L=Santa Clara, O=Real Time Innovations,␣
↪CN=RTI ECDSA01 (p256) ROOT CA, emailAddress=ecdsa01RootCa@rti.com
        Validity
            Not Before: Jan 28 21:26:58 2021 GMT
            Not After : Jan 27 21:26:58 2026 GMT
        Subject: C=US, ST=CA, O=Real Time Innovations,␣
↪emailAddress=ecdsa01Peer01@rti.com, CN=RTI ECDSA01 (p256) PEER01
        Subject Public Key Info:
        ...
```

   macOS

```
$ openssl x509 -in cert/ecdsa01/identities/ecdsa01Peer01Cert.pem -text -
↪noout
Certificate:
    Data:
        Version: 3 (0x2)
```

```
        Serial Number:
            7b:ba:b9:c9:2c:be:ee:b9:71:d0:62:4e:59:6b:de:89:3a:33:5c:ad
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=CA, L=Santa Clara, O=Real Time Innovations,␣
↪CN=RTI ECDSA01 (p256) ROOT CA, emailAddress=ecdsa01RootCa@rti.com
        Validity
            Not Before: Jan 28 21:26:58 2021 GMT
            Not After : Jan 27 21:26:58 2026 GMT
        Subject: C=US, ST=CA, O=Real Time Innovations,␣
↪emailAddress=ecdsa01Peer01@rti.com, CN=RTI ECDSA01 (p256) PEER01
        Subject Public Key Info:
        ...
```

Windows

```
> openssl x509 -in cert\ecdsa01\identities\ecdsa01Peer01Cert.pem -text -
↪noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7b:ba:b9:c9:2c:be:ee:b9:71:d0:62:4e:59:6b:de:89:3a:33:5c:ad
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=CA, L=Santa Clara, O=Real Time Innovations,␣
↪CN=RTI ECDSA01 (p256) ROOT CA, emailAddress=ecdsa01RootCa@rti.com
        Validity
            Not Before: Jan 28 21:26:58 2021 GMT
            Not After : Jan 27 21:26:58 2026 GMT
        Subject: C=US, ST=CA, O=Real Time Innovations,␣
↪emailAddress=ecdsa01Peer01@rti.com, CN=RTI ECDSA01 (p256) PEER01
        Subject Public Key Info:
        ...
```

2. Replace the `subject_name` in the Permissions Document (`pmiPermissionsAlice.xml`) with the **Subject** field of the Identity Certificate (`ecdsa01Peer01Cert.pem`).

3. You may also want to update the `validity` tag with the information from the Identity Certificate. Note that you are not required to have the same validity dates in the Permissions Document and the Identity Certificate (upon creation, your *DomainParticipant* will independently verify that the Identity Certificate and the grant in your Permissions Document are valid for the current date). If you decide to update the `validity` tag, pay attention to the date/time format.

This is the result of updating Alice's grant in `pmiPermissionsAlice.xml`:

```
...
<!-- Grants for a specific DomainParticipant will be grouped under␣
↪this tag -->
<grant name="ParticipantAlice">
    <!-- 1. The rules below will apply to the DomainParticipant
     whose identity certificate contains this subject name -->
    <subject_name>C=US, ST=CA, O=Real Time Innovations,␣
↪emailAddress=ecdsa01Peer01@rti.com, CN=RTI ECDSA01 (p256) PEER01</
↪subject_name>
```

```
    <!-- 2. Validity dates for this grant -->
    <validity>
    <!-- Format is CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm] in GMT -->
        <not_before>2019-11-15T20:24:34</not_before>
        <not_after>2024-11-13T20:24:34</not_after>
    </validity>
    ...
```

6.3 Signing the Permissions Documents

We will use the provided Permissions CA's certificate and key to sign the Permissions Documents that we composed.[1]

Run the command below to create the signed Permissions Document (with PKCS#7 format) named `xml/signed/signed_pmiPermissionsAlice.p7s`:

Linux

```
$ openssl cms -sign -in xml/pmiPermissionsAlice.xml -text -out xml/signed/
↪signed_pmiPermissionsAlice.p7s -signer cert/ecdsa01/ca/ecdsa01RootCaCert.
↪pem -inkey cert/ecdsa01/ca/private/ecdsa01RootCaKey.pem
```

macOS

```
$ openssl cms -sign -in xml/pmiPermissionsAlice.xml -text -out xml/signed/
↪signed_pmiPermissionsAlice.p7s -signer cert/ecdsa01/ca/ecdsa01RootCaCert.
↪pem -inkey cert/ecdsa01/ca/private/ecdsa01RootCaKey.pem
```

Windows

```
> openssl cms -sign -in xml\pmiPermissionsAlice.xml -text -out xml\signed\
↪signed_pmiPermissionsAlice.p7s -signer cert\ecdsa01\ca\ecdsa01RootCaCert.
↪pem -inkey cert\ecdsa01\ca\private\ecdsa01RootCaKey.pem
```

6.4 Updating the QoS Profiles in Your Project

Update `USER_QOS_PROFILES.xml` so that your *DomainParticipants* will load the new Permissions Document:

```
<qos_profile name="Alice" base_name="BuiltinQosLib::Generic.Security" is_
↪default_qos="true">
    <domain_participant_qos>
        <property>
            <value>
                ...
                <element>
```

[1] In this example, we have control of the Permissions CA. This is not always the case, and we may be required to send the Permissions Document to an external entity to get it signed.

```
                    <name>dds.sec.access.permissions</name>
                    <value>file:./xml/signed/signed_pmiPermissionsAlice.p7s</
↪value>
                </element>
                ...
```

Here again, the `file:` prefix means that the signed Permissions Document will be loaded from the specified path in the file system. Note that the path is relative to the working directory from which you run your application (unless you specify an absolute path). For more details, see DDS Security Properties for Configuring Access Control in the *RTI Security Plugins User's Manual*.

6.5 Checking that the New Permissions Are Applied

To verify that Alice is correctly loading her new permissions, we will do two tests. First, we will check that communication only works in *Domain 1*, as specified in the Permissions Document. Then we will make sure that Alice is only allowed to publish data.

6.5.1 Communication Only Works in Domain 1

1. Run your publisher and subscriber as explained in *Running the Applications*.

   Do not provide any arguments to the applications, so that they will try to communicate in *Domain 0*.

   If the publisher has loaded the new permissions, communication should not occur.

   Linux

   Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher
RTI_Security_AccessControl_create_participant:{"DDS:Security:LogTopicV2":
↪{"f":"10","s":"3","t":{"s":"1602772527","n":"484987999"},"h":"rti-10636
↪","i":"0.0.0.0","a":"RTI Secure DDS Application","p":"15137","k":
↪"33554496","x":[{"DDS":[{"domain_id":"<unknown>"},{"guid":"<unknown>"},
↪{"plugin_class":"DDS:Access:Permissions"},{"plugin_method":"RTI_
↪Security_AccessControl_create_participant"}]}],"m":"participant not␣
↪allowed: no rule found for the participant's domain id or domain tag;␣
↪default DENY"}}
DDS_DomainParticipantTrustPlugins_getLocalParticipantSecurityState:!
↪security function check_create_participant returned false
DDS_DomainParticipant_createI:!get local participant security state
DDS_DomainParticipantFactory_create_participant_disabledI:!create␣
↪participant
Exception in run_publisher_application(): Failed to create␣
↪DomainParticipant
```

   Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber
PatientMonitoring subscriber sleeping up to 1 sec...
```

```
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
```

macOS

Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher
RTI_Security_AccessControl_create_participant:{"DDS:Security:LogTopicV2":
 ↪{"f":"10","s":"3","t":{"s":"1602772527","n":"484987999"},"h":"rti-10636
 ↪","i":"0.0.0.0","a":"RTI Secure DDS Application","p":"15137","k":
 ↪"33554496","x":[{"DDS":[{"domain_id":"<unknown>"},{"guid":"<unknown>"},
 ↪{"plugin_class":"DDS:Access:Permissions"},{"plugin_method":"RTI_
 ↪Security_AccessControl_create_participant"}]}],"m":"participant not␣
 ↪allowed: no rule found for the participant's domain id or domain tag;␣
 ↪default DENY"}}
DDS_DomainParticipantTrustPlugins_getLocalParticipantSecurityState:!
 ↪security function check_create_participant returned false
DDS_DomainParticipant_createI:!get local participant security state
DDS_DomainParticipantFactory_create_participant_disabledI:!create␣
 ↪participant
Exception in run_publisher_application(): Failed to create␣
 ↪DomainParticipant
```

Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
```

Windows

Publisher:

```
> objs\<architecture>\PatientMonitoring_publisher.exe
RTI_Security_AccessControl_create_participant:{"DDS:Security:LogTopicV2":
 ↪{"f":"10","s":"3","t":{"s":"1602772527","n":"484987999"},"h":"rti-10636
 ↪","i":"0.0.0.0","a":"RTI Secure DDS Application","p":"15137","k":
 ↪"33554496","x":[{"DDS":[{"domain_id":"<unknown>"},{"guid":"<unknown>"},
 ↪{"plugin_class":"DDS:Access:Permissions"},{"plugin_method":"RTI_
 ↪Security_AccessControl_create_participant"}]}],"m":"participant not␣
 ↪allowed: no rule found for the participant's domain id or domain tag;␣
 ↪default DENY"}}
DDS_DomainParticipantTrustPlugins_getLocalParticipantSecurityState:!
 ↪security function check_create_participant returned false
DDS_DomainParticipant_createI:!get local participant security state
DDS_DomainParticipantFactory_create_participant_disabledI:!create␣
 ↪participant
Exception in run_publisher_application(): Failed to create␣
 ↪DomainParticipant
```

Subscriber:

```
> objs\<architecture>\PatientMonitoring_subscriber.exe
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
```

2. Run your publisher and subscriber using *Domain 1* (specified with the **-d** option in the command line).

   Now communication should succeed.

   Linux

   Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

   Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

   macOS

   Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

   Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
```

```
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

Windows

Publisher:

```
> objs\<architecture>\PatientMonitoring_publisher.exe -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
> objs\<architecture>\PatientMonitoring_subscriber.exe -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

6.5.2 Alice Is Only Allowed to Publish Data

We will now verify that a subscriber application with Alice's permissions is not able to receive data. This means that even if Alice becomes compromised, she can't listen for data she is not authorized to access.

1. Temporarily change your subscriber's QoS profile to load the same security artifacts as Alice, including her identity and permissions:

   ```
   <qos_profile name="Bob" base_name="Alice">
   </qos_profile>
   ```

   This means we are now configuring Bob with Alice's permissions.

2. Run your publisher and subscriber using *Domain 1*.

   Communication should not occur:

   Linux

   Publisher:

   ```
   $ ./objs/<architecture>/PatientMonitoring_publisher -d 1
   Writing PatientMonitoring, count 0
   Writing PatientMonitoring, count 1
   ```

```
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber -d 1
RTI_Security_AccessControl_check_create_datareader:{
 ↪"DDS:Security:LogTopicV2":{"f":"10","s":"3","t":{"s":"1603104580","n":
 ↪"357797999"},"h":"rti-10636","i":"0.0.0.0","a":"RTI Secure DDS␣
 ↪Application","p":"28945","k":"33554496","x":[{"DDS":[{"domain_id":"1"},
 ↪{"guid":"B3339B76.D20F948B.9A35A793.000001C1"},{"plugin_class":
 ↪"DDS:Access:Permissions"},{"plugin_method":"RTI_Security_AccessControl_
 ↪check_create_datareader"}]}],"m":"endpoint not allowed: no rule found;␣
 ↪default DENY"}}
DDS_DomainParticipantTrustPlugins_getLocalDataReaderSecurityState:!
 ↪security function check_create_datareader
DDS_DataReader_create_presentation_readerI:ERROR: Failed to get local␣
 ↪datareader security state
DDS_DataReader_createI:!create reader
DDS_Subscriber_create_datareader_disabledI:!create reader
Exception in run_subscriber_application(): Failed to create DataReader
```

macOS

Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber -d 1
RTI_Security_AccessControl_check_create_datareader:{
 ↪"DDS:Security:LogTopicV2":{"f":"10","s":"3","t":{"s":"1603104580","n":
 ↪"357797999"},"h":"rti-10636","i":"0.0.0.0","a":"RTI Secure DDS␣
 ↪Application","p":"28945","k":"33554496","x":[{"DDS":[{"domain_id":"1"},
 ↪{"guid":"B3339B76.D20F948B.9A35A793.000001C1"},{"plugin_class":
 ↪"DDS:Access:Permissions"},{"plugin_method":"RTI_Security_AccessControl_
 ↪check_create_datareader"}]}],"m":"endpoint not allowed: no rule found;␣
 ↪default DENY"}}
DDS_DomainParticipantTrustPlugins_getLocalDataReaderSecurityState:!
 ↪security function check_create_datareader
DDS_DataReader_create_presentation_readerI:ERROR: Failed to get local␣
 ↪datareader security state
DDS_DataReader_createI:!create reader
DDS_Subscriber_create_datareader_disabledI:!create reader
Exception in run_subscriber_application(): Failed to create DataReader
```

Windows

Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
> objs\<architecture>\PatientMonitoring_subscriber.exe -d 1
RTI_Security_AccessControl_check_create_datareader:{
 ↪"DDS:Security:LogTopicV2":{"f":"10","s":"3","t":{"s":"1603104580","n":
 ↪"357797999"},"h":"rti-10636","i":"0.0.0.0","a":"RTI Secure DDS␣
 ↪Application","p":"28945","k":"33554496","x":[{"DDS":[{"domain_id":"1"},
 ↪{"guid":"B3339B76.D20F948B.9A35A793.000001C1"},{"plugin_class":
 ↪"DDS:Access:Permissions"},{"plugin_method":"RTI_Security_AccessControl_
 ↪check_create_datareader"}]}],"m":"endpoint not allowed: no rule found;␣
 ↪default DENY"}}
DDS_DomainParticipantTrustPlugins_getLocalDataReaderSecurityState:!
 ↪security function check_create_datareader
DDS_DataReader_create_presentation_readerI:ERROR: Failed to get local␣
 ↪datareader security state
DDS_DataReader_createI:!create reader
DDS_Subscriber_create_datareader_disabledI:!create reader
Exception in run_subscriber_application(): Failed to create DataReader
```

This test requires your Governance Document to set `enable_read_access_control` to `TRUE` (see *Further Exercises* in Hands-On 2); otherwise, endpoint-level permissions related to *DataReaders* will not be checked and Bob will be able to receive the data. When adding new rules to your Governance Document, keep in mind (as we mentioned in *Adding a Topic Rule for the PatientMonitoring Topic*) that the Governance Document is processed from top to bottom, and that only the first matching rule is applied. Also, make sure the Permissions CA signs the modified Governance Document (see *Signing the Governance Document*); otherwise the changes will not be applied.

This exercise illustrates what happens if an application tries to do something it does not have permissions to do, by temporarily giving Bob (the subscriber) the wrong permissions.

Tip: Make sure you revert the changes to your subscriber's QoS profile before continuing the exercises.

6.6 Further Exercises

At this point, you have defined the permissions that will be applied to your publisher application, restricting it to publish *PatientMonitoring* data. However, we have not modified your subscriber's permissions.

6.6.1 Define Different Permissions for Each Application in Your System

To minimize the damage in case one of your secure applications is compromised, a *DomainParticipant* should only have permissions to perform the actions that are necessary for its legitimate purpose.[2]

1. Define the permissions for Bob in a file called `pmiPermissionsBob.xml`. This Permissions Document should contain a single grant named *ParticipantBob*, defining permissions to subscribe to any *Topic* containing the string `Patient` — only in *Domain 1*.

   You may want to use the following `allow_rule` as a reference:

   ```
   <allow_rule>
       <domains>
           <id>1</id>
       </domains>
       <subscribe>
           <topics>
               <topic>*Patient*</topic>
           </topics>
       </subscribe>
   </allow_rule>
   ```

   ParticipantBob should be denied any other actions.

2. Bind Bob's permissions to your subscriber application (see *Binding the Permissions Document to Your DomainParticipants*).

   If you completed the exercises in *Hands-On 1: Securing Connext DDS Applications*), Bob's identity certificate should be `cert/ecdsa01/identities/ecdsa01Peer02Cert.pem`. You will have to bind Bob's permissions to the subject of this certificate.

3. Make sure the Permissions CA signs the new Permissions Document and modify Bob's QoS profile to load it.

   After applying this configuration, your subscriber application, Bob, will only be allowed to subscribe to any *Topic* containing the string `Patient` in *Domain 1*, while your publisher application, Alice, will only be allowed to publish *Topic* "Example PatientMonitoring" in *Domain 1*.

6.7 Troubleshooting

- When I run the publisher/subscriber, I get this error:

  ```
  RTI_Security_PermissionsCfgFileParser_getGrantForSubjectName:XML file␣
  ↪doesn't contain a grant for subject name
  ```

  Make sure your *DomainParticipants* load the correct Identity Certificate and Permissions Document.

  Make sure the subject in your Permissions Document's grant matches the information in your participant's identity certificate (see *Binding the Permissions Document to Your DomainParticipants*). Pay attention to the format.

[2] This is known as the Principle of Least Privilege. For further information, see Applying DDS Protection in the Security Plugins User's Manual.

- When I run the publisher/subscriber, I get this error:

```
RTI_Security_AccessControl_create_participant:participant not allowed:␣
↪no rule found for the participant's domain id or domain tag; default␣
↪DENY
```

  Make sure you run your applications in the right *Domain*, by specifying the Domain ID with the **-d** option in the command line. In this example, we are using *Domain 1*.

- The subscriber is able to receive data, even if it does not have permissions to subscribe.

  Make sure your Governance Document sets `enable_read_access_control` `TRUE`. Otherwise, permissions will not be enforced on locally created or remotely discovered *DataReaders*.

  You may be using the Governance Document from *Composing a Governance Document with the Security Requirements*, with a specific topic rule appearing after the wildcard rule that applies to all topics. If your topic rule is more restrictive than the wildcard rule, it must be located first. The Governance Document is processed from top to bottom, and the first matching rule is applied.

- When I run the publisher/subscriber, I get an error I do not understand.

  See Logging Messages in the *RTI Security Plugins User's Manual*.

Chapter 7

Hands-On 4: Generating and Revoking Your Own Certificates Using OpenSSL

In this Hands-On, you will take control of your project's keys and certificates. This way, you will not need to rely on the example artifacts provided with *Connext* to secure your applications. We will start by generating a self-signed Identity CA, which will issue the Identity Certificates for Alice and Bob. This will require us to set up a minimal security infrastructure first. Then we will generate a new Permissions CA that will sign the Governance Document and all the Permissions Documents. After each certificate generation, we will refer to the modifications needed to make your project load the new artifacts. Finally, we will revoke one Identity Certificate and use a Certificate Revocation List (CRL) to avoid communicating with the revoked *DomainParticipant*.

We will show examples for ECDSA and RSA as the public-key algorithm to generate the certificates. Note that you can use any public-key algorithm listed in Supported Cryptographic Algorithms in the *RTI Security Plugins User's Manual*.

Note: We will use the **OpenSSL CLI** to perform the security operations in the generation of the security artifacts. Make sure to include in the path your OpenSSL binary directory[3]. The installation process is described in the RTI Security Plugins Installation Guide.

Note that the SECURITY PLUGINS do not depend on OpenSSL to generate these artifacts; you can use the security toolkit of your choice. With that said, we recommend using OpenSSL to make sure that the certificates are in the right format.

---

[3] In the free version (with `lm` in the bundle name), OpenSSL is installed automatically when you install the *Connext* host bundle. After installation, OpenSSL will be in `<installdir>/third_party/openssl-<version>`.

7.1 Preliminary Steps

Setting up a security infrastructure requires some preliminary configuration. We will cover a minimal setup here.

1. If you followed the steps in *Hands-On 1: Securing Connext DDS Applications*, you should have an OpenSSL configuration file named `cert/ecdsa01/ca/ecdsa01RootCa.cnf`[1]. Make two copies of this file and call them `pmiIdentityCa.cnf` and `pmiPermissionsCa.cnf`. To better organize your project, save these copies in a new directory called `cert/pmi/ca`:

   Linux

   ```
   $ cp cert/ecdsa01/ca/ecdsa01RootCa.cnf cert/pmi/ca/pmiIdentityCa.cnf
   $ cp cert/ecdsa01/ca/ecdsa01RootCa.cnf cert/pmi/ca/pmiPermissionsCa.cnf
   ```

   macOS

   ```
   $ cp cert/ecdsa01/ca/ecdsa01RootCa.cnf cert/pmi/ca/pmiIdentityCa.cnf
   $ cp cert/ecdsa01/ca/ecdsa01RootCa.cnf cert/pmi/ca/pmiPermissionsCa.cnf
   ```

   Windows

   ```
   > copy cert\ecdsa01\ca\ecdsa01RootCa.cnf cert\pmi\ca\pmiIdentityCa.cnf
           1 file(s) copied.
   > copy cert\ecdsa01\ca\ecdsa01RootCa.cnf cert\pmi\ca\pmiPermissionsCa.cnf
           1 file(s) copied.
   ```

   Hint: If you want to use RSA as your public-key algorithm, you may want to copy the homologous example files from the `cert/rsa01/` directory.

2. Modify `pmiIdentityCa.cnf` to redefine the `name` variable. Note that this configuration file uses this variable to derive some filenames, such as those used in the next section:

   ```
   ...
   # Variables defining this CA
   name = pmiIdentityCa                    # Name
   desc =                                  # Description
   ...
   ```

[1] Read the official documentation for more information on the OpenSSL configuration files.

7.1.1 Initialize the OpenSSL CA Database

When using a CA to perform an operation, OpenSSL relies on special database files to keep track of the issued certificates, serial numbers, revoked certificates, etc. We need to create these database files to be able to use the **`openssl x509 -req`** command:

Linux

```
$ mkdir cert/pmi/ca/database
$ touch cert/pmi/ca/database/pmiIdentityCaIndex
$ echo 01 > cert/pmi/ca/database/pmiIdentityCaSerial
```

macOS

```
$ mkdir cert/pmi/ca/database
$ touch cert/pmi/ca/database/pmiIdentityCaIndex
$ echo 01 > cert/pmi/ca/database/pmiIdentityCaSerial
```

Windows

```
> mkdir cert\pmi\ca\database
> type nul > cert\pmi\ca\database\pmiIdentityCaIndex
> echo 01> cert\pmi\ca\database\pmiIdentityCaSerial
```

7.1.2 Limit the Access of the CA's Private Key

It is also a good practice to store the CA's private key in a separate directory with more restrictive access rights, so only you can sign certificates.

Linux

```
$ mkdir cert/pmi/ca/private
$ chmod 700 cert/pmi/ca/private
```

macOS

```
$ mkdir cert/pmi/ca/private
$ chmod 700 cert/pmi/ca/private
```

Windows

```
> mkdir cert\pmi\ca\private

> icacls cert\pmi\ca\private /t /inheritance:d
processed file: cert\pmi\ca\private
Successfully processed 1 files; Failed processing 0 files

> icacls cert\pmi\ca\private /t /remove Administrator "Authenticated Users"␣
↪BUILTIN Everyone System Users
processed file: cert\pmi\ca\private
Successfully processed 1 files; Failed processing 0 files
```

```
> icacls cert\pmi\ca\private /grant %USERNAME%:F
processed file: cert\pmi\ca\private
Successfully processed 1 files; Failed processing 0 files
```

7.2 Generating a New Identity CA

1. Modify `cert/pmi/ca/pmiIdentityCa.cnf` and specify the fields in the `req_distinguished_name` section. This information will be incorporated into your certificate:

```
...
[ req_distinguished_name ]

countryName            = US
stateOrProvinceName    = CA
localityName           = Santa Clara
0.organizationName     = Patient Monitoring Innovations
commonName             = PMI Identity CA
emailAddress           = identityca@pmi.com
...
```

2. Use the OpenSSL CLI to generate a self-signed certificate using the Identity CA's configuration. Run the following command from the `cert/pmi` directory:

   ECDSA secp256r1

```
openssl req -nodes -x509 -days 1825 -text -sha256 -newkey ec -pkeyopt ec_
↪paramgen_curve:prime256v1 -keyout ca/private/pmiIdentityCaKey.pem -out␣
↪ca/pmiIdentityCaCert.pem -config ca/pmiIdentityCa.cnf
```

   ECDSA secp384r1

```
openssl req -nodes -x509 -days 1825 -text -sha384 -newkey ec -pkeyopt ec_
↪paramgen_curve:secp384r1 -keyout ca/private/pmiIdentityCaKey.pem -out␣
↪ca/pmiIdentityCaCert.pem -config ca/pmiIdentityCa.cnf
```

   RSA

```
openssl req -nodes -x509 -days 1825 -text -sha256 -newkey rsa:2048 -
↪keyout ca/private/pmiIdentityCaKey.pem -out ca/pmiIdentityCaCert.pem -
↪config ca/pmiIdentityCa.cnf
```

   This will produce a new private key, `pmiIdentityCaKey.pem` in the `cert/pmi/ca/private` directory, and a new certificate, `pmiIdentityCaCert.pem`, in the `cert/pmi/ca` directory. This certificate will be valid for 1825 days (5 years) starting today.

7.2.1 Specifying the New Identity CA Certificate in QoS Profiles

Modify `USER_QOS_PROFILES.xml` to make your *DomainParticipants* load the certificate of the new Identity CA:

```
...
<element>
    <name>dds.sec.auth.identity_ca</name>
    <value>file:./cert/pmi/ca/pmiIdentityCaCert.pem</value>
</element>
...
```

7.3 Generating Identity Certificates

As explained in *Introduction to Security Plugins*, Identity Certificates are verified against the Identity CA when authenticating remote *DomainParticipants*. Therefore, in the simplest scenario, it is the Identity CA that is responsible for issuing Identity Certificates.[2] We will create a certificate signing request (CSR) for Alice. Then we will use the new Identity CA to issue the certificate requested by the CSR.

1. Add the information you want to include in Alice's certificate in a file called `pmiAlice.cnf`. Save this file in a new directory called `cert/pmi/identities`. You may want to use the following contents as a reference:

   Listing 7.1: Sample contents of pmiAlice.cnf

   ```
   prompt              = no
   distinguished_name  = req_distinguished_name

   [ req_distinguished_name ]
   countryName           = US
   stateOrProvinceName   = CA
   localityName          = Santa Clara
   organizationName      = Patient Monitoring Innovations
   emailAddress          = alice@pmi.com
   commonName            = Alice
   ```

   You are free to modify any field except `countryName`, `stateOrProvinceName`, and `organizationName`. These fields must match the ones of the Identity CA; otherwise it will refuse to issue the requested certificate (note that a `commonName` is also required). These requirements are specified in `pmiIdentityCa.cnf`, in the `policy_match` section.

2. Generate Alice's key and CSR. Run the following command from the `cert/pmi` directory:

   ECDSA secp256r1

   ```
   openssl req -nodes -new -newkey ec -pkeyopt ec_paramgen_curve:prime256v1␣
   ↪-config identities/pmiAlice.cnf -keyout identities/pmiAliceKey.pem -
   ↪out identities/pmiAlice.csr
   ```

[2] Depending on your use case, the Identity Certificates may be issued by an intermediate CA in your PKI instead. For further information, see Public Key Infrastructure (PKI) in the *RTI Security Plugins User's Manual*.

ECDSA secp384r1

```
openssl req -nodes -new -newkey ec -pkeyopt ec_paramgen_curve:secp384r1 -
↪config identities/pmiAlice.cnf -keyout identities/pmiAliceKey.pem -out␣
↪identities/pmiAlice.csr
```

RSA

```
openssl req -nodes -new -newkey rsa:2048 -config identities/pmiAlice.cnf␣
↪-keyout identities/pmiAliceKey.pem -out identities/pmiAlice.csr
```

This will produce an RSA private key, `pmiAliceKey.pem`, and a CSR based on that key, `pmiAlice.csr`. Since CSRs have all the information and cryptographic material that a CA needs to issue a certificate, Alice's private key must never be known to anyone but her.

3. Use the new Identity CA's certificate and private key to issue Alice's Identity Certificate. Run the following command from the `cert/pmi` directory:

   ECDSA secp256r1

   ```
   openssl x509 -req -days 730 -text -CAserial ca/database/
   ↪pmiIdentityCaSerial -CA ca/pmiIdentityCaCert.pem -CAkey ca/private/
   ↪pmiIdentityCaKey.pem -in identities/pmiAlice.csr -out identities/
   ↪pmiAliceCert.pem
   ```

   ECDSA secp384r1

   ```
   openssl x509 -req -days 730 -sha384 -text -CAserial ca/database/
   ↪pmiIdentityCaSerial -CA ca/pmiIdentityCaCert.pem -CAkey ca/private/
   ↪pmiIdentityCaKey.pem -in identities/pmiAlice.csr -out identities/
   ↪pmiAliceCert.pem
   ```

   RSA

   ```
   openssl x509 -req -days 730 -text -CAserial ca/database/
   ↪pmiIdentityCaSerial -CA ca/pmiIdentityCaCert.pem -CAkey ca/private/
   ↪pmiIdentityCaKey.pem -in identities/pmiAlice.csr -out identities/
   ↪pmiAliceCert.pem
   ```

   The Identity CA will issue Alice's public certificate, `pmiAliceCert.pem`, which will be valid for 730 days (2 years) starting today.

7.3.1 Specifying the New Identity Certificates to Your QoS Profiles

Modify `USER_QOS_PROFILES.xml` to make your publisher application load the new pair of certificate and private key:

```
...
<!-- Participant Public Certificate and Private Key -->
<element>
    <name>dds.sec.auth.identity_certificate</name>
    <value>file:./cert/pmi/identities/pmiAliceCert.pem</value>
```

```
</element>
<element>
    <name>dds.sec.auth.private_key</name>
    <value>file:./cert/pmi/identities/pmiAliceKey.pem</value>
</element>
...
```

7.4 Updating Permissions Documents with New Credentials

As explained in *Binding the Permissions Document to Your DomainParticipants*, grants in a Permissions Document are bound to *DomainParticipant* identities. We will update Alice's Permissions Document with the same information we included in Alice's Identity Certificate.

1. To meet the format requirements of the Permissions Document, you may want to check your certificate's information with the following command:

   Linux

   ```
   $ openssl x509 -in cert/pmi/identities/pmiAliceCert.pem -text -noout
   Certificate:
       Data:
           Version: 1 (0x0)
           Serial Number: 1 (0x1)
       Signature Algorithm: ecdsa-with-SHA256
           Issuer: C=US, ST=CA, L=Santa Clara, O=Patient Monitoring␣
   ↪Innovations, CN=PMI Identity CA, emailAddress=identityca@pmi.com
           Validity
               Not Before: Feb  7 15:37:09 2021 GMT
               Not After : Feb  7 15:37:09 2023 GMT
           Subject: C=US, ST=CA, O=Patient Monitoring Innovations, CN=Alice,
   ↪ emailAddress=alice@pmi.com
           Subject Public Key Info:
           ...
   ```

   macOS

   ```
   $ openssl x509 -in cert/pmi/identities/pmiAliceCert.pem -text -noout
   Certificate:
       Data:
           Version: 1 (0x0)
           Serial Number: 1 (0x1)
       Signature Algorithm: ecdsa-with-SHA256
           Issuer: C=US, ST=CA, L=Santa Clara, O=Patient Monitoring␣
   ↪Innovations, CN=PMI Identity CA, emailAddress=identityca@pmi.com
           Validity
               Not Before: Feb  7 15:37:09 2021 GMT
               Not After : Feb  7 15:37:09 2023 GMT
           Subject: C=US, ST=CA, O=Patient Monitoring Innovations, CN=Alice,
   ↪ emailAddress=alice@pmi.com
           Subject Public Key Info:
           ...
   ```

Windows

```
> openssl x509 -in cert\pmi\identities\pmiAliceCert.pem -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 1 (0x1)
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, ST=CA, L=Santa Clara, O=Patient Monitoring␣
↪Innovations, CN=PMI Identity CA, emailAddress=identityca@pmi.com
        Validity
            Not Before: Feb  7 15:37:09 2021 GMT
            Not After : Feb  7 15:37:09 2023 GMT
        Subject: C=US, ST=CA, O=Patient Monitoring Innovations, CN=Alice,
↪ emailAddress=alice@pmi.com
        Subject Public Key Info:
        ...
```

2. Modify the `subject_name` in the Permissions Document, `xml/pmiPermissionsAlice.xml`, to match the certificate's subject.

3. You may also want to update the `validity` tag with the information from the Identity Certificate. Note that you are not required to have the same validity dates in the Permissions Document and the Identity Certificate (upon creation, your *DomainParticipant* will independently verify that the Identity Certificate and the grant in your Permissions Document are valid for the current date). If you decide to update the `validity` tag, pay attention to the date/time format.

```
...
<!-- Grants for a specific DomainParticipant will be grouped under this␣
↪tag -->
<grant name="ParticipantAlice">
    <!-- 1. The rules below will apply to the DomainParticipant
            whose identity certificate contains this subject name -->
    <subject_name>C=US, ST=CA, O=Patient Monitoring Innovations,␣
↪CN=Alice, emailAddress=alice@pmi.com</subject_name>
    <!-- 2. Validity dates for this grant -->
    <validity>
        <!-- Format is CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm] in GMT -->
        <not_before>2019-11-25T16:17:07</not_before>
        <not_after>2021-11-24T16:17:07</not_after>
    </validity>
    ...
```

7.5 Generating a New Permissions CA

Note:

- The Identity CA and Permissions CA may be the same, depending on your use case.
- This section is analogous to *Generating a New Identity CA*.

1. Modify `cert/pmi/ca/pmiPermissionsCa.cnf` and specify the fields under the `req_distinguished_name` section.

   This information will be incorporated into your certificate:

   ```
   ...
   [ req_distinguished_name ]

   countryName            = US
   stateOrProvinceName    = CA
   localityName           = Santa Clara
   0.organizationName     = Patient Monitoring Innovations
   commonName             = PMI Permissions CA
   emailAddress           = permissionsca@pmi.com
   ...
   ```

2. Use the OpenSSL CLI to generate a self-signed certificate using the Permissions CA's configuration. Run the following command from the `cert/pmi` directory:

   ECDSA secp256r1

   ```
   openssl req -nodes -x509 -days 1825 -text -sha256 -newkey ec -pkeyopt ec_
   ↪paramgen_curve:prime256v1 -keyout ca/private/pmiPermissionsCaKey.pem -
   ↪out ca/pmiPermissionsCaCert.pem -config ca/pmiPermissionsCa.cnf
   ```

   ECDSA secp384r1

   ```
   openssl req -nodes -x509 -days 1825 -text -sha384 -newkey ec -pkeyopt ec_
   ↪paramgen_curve:secp384r1 -keyout ca/private/pmiPermissionsCaKey.pem -
   ↪out ca/pmiPermissionsCaCert.pem -config ca/pmiPermissionsCa.cnf
   ```

   RSA

   ```
   openssl req -nodes -x509 -days 1825 -text -sha256 -newkey rsa:2048 -
   ↪keyout ca/private/pmiPermissionsCaKey.pem -out ca/pmiPermissionsCaCert.
   ↪pem -config ca/pmiPermissionsCa.cnf
   ```

   This will produce a new private key, `pmiPermissionsCaKey.pem` in the `cert/pmi/ca/private` directory, and a new certificate, `pmiPermissionsCaCert.pem`, in the `cert/pmi/ca` directory. This certificate will be valid for 1825 days (5 years) starting today.

7.5.1 Specifying the New Permissions CA Certificate in QoS Profiles

Modify `USER_QOS_PROFILES.xml` to make your *DomainParticipants* load the certificate of the new Permissions CA:

```
...
    <element>
        <name>dds.sec.access.permissions_ca</name>
        <value>file:./cert/pmi/ca/pmiPermissionsCaCert.pem</value>
    </element>
    ...
```

7.6 Signing the Governance and Permissions Documents

Note: This section was covered in *Signing the Governance Document* and *Signing the Permissions Documents*.

We will use the Permissions CA's certificate and key that we generated to sign the Governance and Permissions Documents that we composed in previous Hands-On sections.

1. Run the command below to create the signed Governance Document (with PKCS#7 format) named `xml/signed/pmiSigned_pmiGovernance.p7s`:

```
openssl cms -sign -in xml/pmiGovernance.xml -text -out xml/signed/
↪pmiSigned_pmiGovernance.p7s -signer cert/pmi/ca/pmiPermissionsCaCert.
↪pem -inkey cert/pmi/ca/private/pmiPermissionsCaKey.pem
```

2. Run the command below to create the signed Permissions Document (with PKCS#7 format) named `xml/signed/pmiSigned_pmiPermissionsAlice.p7s`:

```
openssl cms -sign -in xml/pmiPermissionsAlice.xml -text -out xml/signed/
↪pmiSigned_pmiPermissionsAlice.p7s -signer cert/pmi/ca/
↪pmiPermissionsCaCert.pem -inkey cert/pmi/ca/private/
↪pmiPermissionsCaKey.pem
```

7.6.1 Specifying the New Governance and Permissions Documents in Your QoS Profiles

Lastly, update `USER_QOS_PROFILES.xml` so that your *DomainParticipants* will load the Governance and Permissions Documents signed by your Permissions CA:

```
...
<!-- Signed Governance and Permissions Documents -->
<element>
    <name>dds.sec.access.governance</name>
    <value>file:./xml/signed/pmiSigned_pmiGovernance.p7s</value>
</element>
<element>
```

```
    <name>dds.sec.access.permissions</name>
    <value>file:./xml/signed/pmiSigned_pmiPermissionsAlice.p7s</value>
</element>
...
```

7.7 Updating the Subscriber's Configuration

At this point, you have control of the keys and certificates used in your project. Congratulations!

In the previous steps, we have focused on updating your publisher application, Alice, with a new identity and matching permissions. Now, we will update Bob's identity and permissions accordingly.

1. Create an identity for your subscriber application, Bob, as described in *Generating Identity Certificates*. After this step, Bob's QoS profile in `USER_QOS_PROFILES.xml` should load his new Identity Certificate, `pmiBob.pem`, and private key, `pmiBobKey.pem`.
2. Update Bob's Permissions Document, `pmiPermissionsBob.xml`, to match his new credentials (see *Updating Permissions Documents with New Credentials*).
3. Finally, sign Bob's Permissions Document with your new Permissions CA, as explained in *Signing the Governance and Permissions Documents*. After this step, Bob's QoS profile in `USER_QOS_PROFILES.xml` should load the signed version of his new Permissions Document, `pmiSigned_pmiPermissionsBob.p7s`.

Now, run your publisher and subscriber using *Domain 1* (specified with the **-d** option in the command line). You should see communication.

Linux

Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

macOS

Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
$ ./objs/<architecture>/PatientMonitoring_subscriber -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

Windows

Publisher:

```
> objs\<architecture>\PatientMonitoring_publisher.exe -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
> objs\<architecture>\PatientMonitoring_subscriber.exe -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: ]
PatientMonitoring subscriber sleeping up to 1 sec...
```

7.8 Revoking an Identity Certificate

At this point, Alice and Bob are communicating using Identity Certificates that you generated yourself. Congratulations!

Now, we will learn how to revoke Alice's Identity Certificate and how Bob can react to Alice's change in status.

1. To revoke Alice's Identity Certificate, run the following command from the `cert/pmi` directory:

   Linux

   ```
   $ openssl ca -config ca/pmiIdentityCa.cnf -batch -revoke identities/
   ↪pmiAliceCert.pem
   ```

   macOS

   ```
   $ openssl ca -config ca/pmiIdentityCa.cnf -batch -revoke identities/
   ↪pmiAliceCert.pem
   ```

   Windows

   ```
   > openssl ca -config ca\pmiIdentityCa.cnf -batch -revoke identities\
   ↪pmiAliceCert.pem
   ```

   You may repeat this command to revoke more certificates issued by the same CA.

   If you would like to generate a valid CRL without revoking any certificates, you may skip this step and proceed directly to the next step.

2. When you are done revoking certificates, you have to inform the *DomainParticipants* about the revoked certificates by generating a list of the revoked identities, which is called a Certificate Revocation List (CRL). To generate the list, run the following commands from the `cert/pmi` directory:

Linux

```
$ echo 01 > ca/database/pmiIdentityCaCrlNumber
$ openssl ca -config ca/pmiIdentityCa.cnf -batch -gencrl -out ca/pmi.crl
```

macOS

```
$ echo 01 > ca/database/pmiIdentityCaCrlNumber
$ openssl ca -config ca/pmiIdentityCa.cnf -batch -gencrl -out ca/pmi.crl
```

Windows

```
> echo 01> ca\database\pmiIdentityCaCrlNumber
> openssl ca -config ca\pmiIdentityCa.cnf -batch -gencrl -out ca\pmi.crl
```

7.8.1 Specifying the New Certificate Revocation List in QoS Profiles

Each *DomainParticipant* that cares about avoiding communication with revoked *DomainParticipants* needs to load this list. Let's assume Bob cares about avoiding communication with Alice. Modify `USER_QOS_PROFILES.xml` to make Bob's *DomainParticipant* load the new CRL:

```
...
<element>
    <name>dds.sec.auth.crl</name>
    <value>file:./cert/pmi/ca/pmi.crl</value>
</element>
...
```

Stop and restart the subscriber application so that Bob reads the updated QoS and loads the new CRL. Now, when Bob discovers Alice, Bob will print this error:

```
"X509_verify_cert returned 0 with error 23: certificate revoked
subject name: /C=US/ST=CA/L=Santa Clara/O=Patient Monitoring Innovations/
↪emailAddress=alice@pmi.com/CN=Alice
issuer name: /C=US/ST=CA/L=Santa Clara/O=Patient Monitoring Innovations/
↪CN=PMI Identity CA/emailAddress=identityca@pmi.com"
```

Communication will not occur between Alice and Bob.

Note: The `com.rti.serv.secure.authentication.crl` property is mutable, so you can change it at runtime. To do so, use the set_qos API or, if file tracker is enabled, directly change the contents of the file that the property points to. You can start with an empty CRL file, and then add entries to it as you revoke more certificates.

7.9 Troubleshooting

- When I run my subscriber, I get the following error:

  ```
  [CREATE Participant] RTI_Security_CertHelper_loadPrivateKey:private_key␣
  ↪is not encrypted, yet password is supplied. Aborting participant␣
  ↪creation due to inconsistent configuration.
  ```

  In previous Hands-On sections, your subscriber used the `ecdsa01Peer02Key.pem` private key, which is password protected. If you did not specify a password for `BobKey.pem`, make sure you remove the `dds.sec.auth.password` property from Bob's profile in `USER_QOS_PROFILES.xml`.

- How can I verify whether a CA correctly issued an Identity Certificate?

  ```
  openssl verify -CAfile <identityCACert> <peerIdentityCert>
  ```

- How can I verify the signed Governance and Permissions Documents?

```
openssl cms -verify -CAfile <permissionsCACert> -in <signedFile>
```

- I am using the Security Plugins with wolfSSL and see the following error when running either my publisher or my subscriber:

```
RTI_Security_CertHelper_loadCertsCrls:WolfSSL function PEM_X509_INFO_
↪read_bio failed with error: (error details not available)
RTI_Security_CertHelper_loadCertsCrls:Error loading certificates
```

  This might be caused by the certificates being in an incompatible format. We recommend generating the certificates using OpenSSL version 1.1.1k or above, which you can install using the process described in the RTI Security Plugins Installation Guide.

- When I attempt to create a *DomainParticipant*, I get the following error:

```
X509_verify_cert returned 0 with error 79: invalid CA certificate
```

  You may be using an X.509 v3 Identity CA that does not have `basicConstraints = CA:true`. Refer to The basicConstraints X.509 v3 extension in the *RTI Security Plugins User's Manual* for more details. In `pmiIdentityCa.cnf`, make sure the `x509_extensions = v3_ca` line is not commented out.

Note: When running `openssl ca` or `openssl x509`, OpenSSL version 3.2.0 and above generate X.509 v3 (as opposed to v1) certificates by default.

For further troubleshooting, see *Troubleshooting in Hands-On 3*.

Chapter 8

Hands-On 5: Checking that Your DDS Traffic Is Protected

In this Hands-On, you will learn about the effects that enabling DDS Security has on the wire. We will start by disabling security in our project and viewing the contents of RTPS packets with Wireshark. Then we will re-enable the protections defined in *Hands-On 2: Defining Your System's Security Requirements*. We will use Wireshark again to verify that messages from the publisher are encrypted.

Note: We will use Wireshark to capture and analyze the packets on the network. For instructions on getting and installing Wireshark, please refer to Building and Installing Wireshark in Wireshark's User's Guide. For instructions on enabling convenient coloring rules for the RTPS protocol, see How To configure Wireshark to show RTPS packets with specific colors from our Knowledge Base.

8.1 Disabling Security and Preparing Your Project for Traffic Capturing

In *Hands-On 1: Securing Connext DDS Applications* we defined a QoS profile named Alice to enable the Security Plugins. This profile inherits from the builtin profile, `BuiltinQosLib::Generic.Security`, which tells your *DomainParticipant* to enable the Security Plugins. (See Properties for Enabling Security) in the *RTI Security Plugins User's Manual*. We can instead use the `BuiltinQosLib::Generic.Common` profile to have an application that does not load the Security Plugins.

We will also need to force communication over UDPv4 to capture traffic using Wireshark. We need to do this because, by default, applications running on the same machine will communicate using shared memory (for more details, see How to capture traffic if my Connext DDS applications are communicating through shared memory from our Knowledge Base).

1. Modify `USER_QOS_PROFILES.xml` to load the `BuiltinQosLib::Generic.Common` profile and force communication over the UDPv4 transport protocol:

```
<qos_profile name="Alice" base_name="BuiltinQosLib::Generic.Common" is_
↪default_qos="true">
```

```
    <domain_participant_qos>
        <!-- Disable shared memory to capture UDP packets on Wireshark --
↪>
        <transport_builtin>
            <mask>UDPv4</mask>
        </transport_builtin>
        ...
```

Please note that this change affects both the publisher and subscriber applications.

2. Change your publisher's source code to send a message that we will read using Wireshark:

```
void run_publisher_application(unsigned int domain_id, unsigned int␣
↪sample_count)
{
    ...
    PatientMonitoring data;
    for (unsigned int samples_written = 0;
    !application::shutdown_requested && samples_written < sample_count;
    samples_written++) {
        // Modify the data to be written here
        data.patient_condition(
                std::string("{heart_rate: ")
                + std::to_string(100 + samples_written%10)
                + "}");
        ...
```

After this step, you will need to rebuild your project (see *Linking Your Applications Against Security Plugins and OpenSSL Libraries*).

8.1.1 Analyzing RTPS Packets in Wireshark

1. Open Wireshark and start capturing packets on the loopback interface[1]. Alternatively, you may want to use another interface if your publisher and subscriber applications are on different machines[2]. To filter out packets that are not related to your applications, apply the following display filter:

```
rtps.domain_id == 1
```

Note: In most cases, you want to start capturing packets before you start your DDS applications. This way, you will be able to analyze the communication from the beginning, including DDS Discovery traffic.

2. Run your publisher and subscriber as explained in *Running the Applications*. Specify *Domain 1* with the **-d** option in the command line. You should see communication:

[1] Recording on the loopback interface on a Windows system may require additional software (see How to capture traffic from the loopback interface using Wireshark in Windows from our Knowledge Base).

[2] The screenshots in this Hands-On show the publisher and subscriber running on two different machines (with different IP addresses). We have edited the *Resolved Name* in Wireshark to better identify the applications. If the publisher and subscriber run on the same machine, you can still differentiate them by looking at the UDP ports.

Linux

Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
$ ./objs/x64Linux3gcc5.4.0/PatientMonitoring_subscriber -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 101}]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 102}]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 103}]
PatientMonitoring subscriber sleeping up to 1 sec...
```

macOS

Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
$ ./objs/x64Linux3gcc5.4.0/PatientMonitoring_subscriber -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 101}]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 102}]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 103}]
PatientMonitoring subscriber sleeping up to 1 sec...
```

Windows

Publisher:

```
> objs\<architecture>\PatientMonitoring_publisher.exe -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
> objs\<architecture>\PatientMonitoring_subscriber.exe -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 101}]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 102}]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 103}]
PatientMonitoring subscriber sleeping up to 1 sec...
```

3. In the Wireshark capture, you should see a series of RTPS packets, as shown in the following screenshot. You can see the DDS data samples by looking at the *serializedData* field of DATA submessages. Since the current scenario does not load the SECURITY PLUGINS, security is completely disabled and the payload can be decoded as ASCII plaintext.

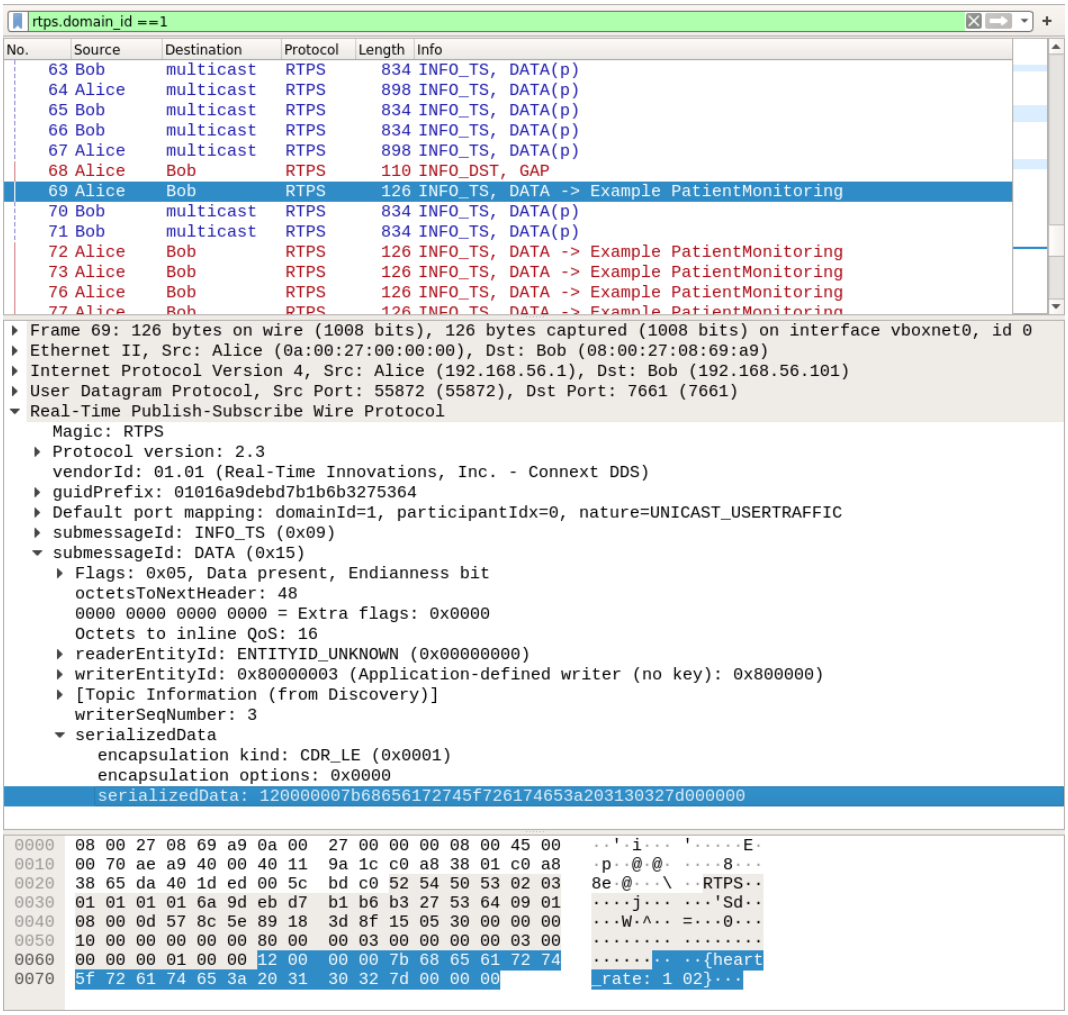


Figure 8.1: Wireshark capture of unprotected communication. The payload can be decoded as ASCII plaintext.

If we focus on the structure of captured packets, we can see that RTPS messages consist of a *header* and one or several *submessages*. In turn, every submessage has its *header*, *metadata*, and *contents*. In the previous screenshot, we can see a DATA submessage with some *Flags* in its header, then some metadata such as the *writerEntityId*, and finally, its serialized payload for the content. The following diagram outlines the structure of an unprotected RTPS packet.

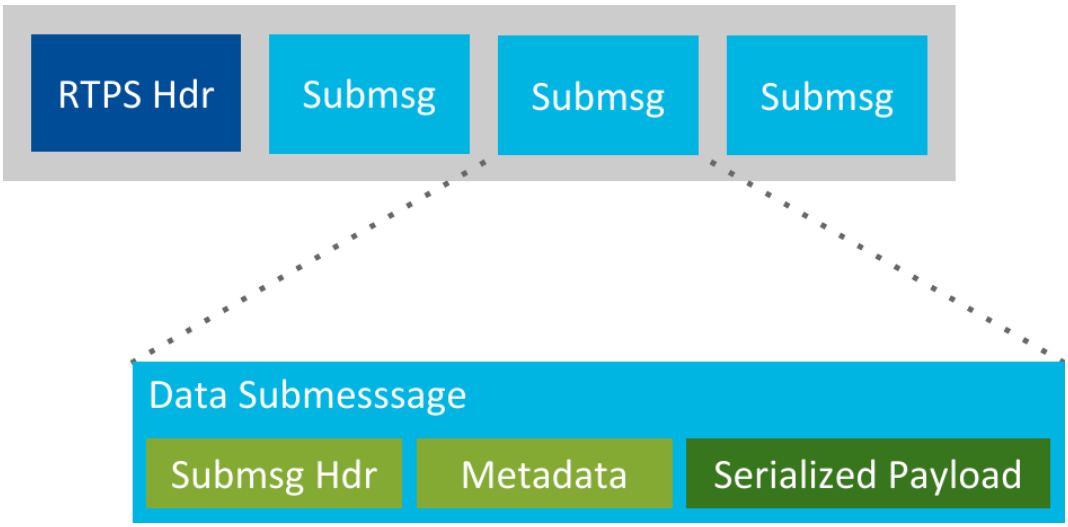


Figure 8.2: Structure of an unprotected RTPS packet.

When our application uses security, the contents of the RTPS packets will show some differences. For example, cryptographic metadata will be added around the protected parts, which may be encrypted depending on the configured protection. For further details, see Securing DDS Messages on The Wire in the *RTI Security Plugins User's Manual*.

8.2 Encrypting the Serialized Payload

In the previous section, you configured your *DomainParticipants* so they did not load the Security Plugins, making *PatientMonitoring* data vulnerable to eavesdropping and tampering attacks. In this section, we will analyze the effects that encrypting the payload has on the RTPS packets.

1. Modify `USER_QOS_PROFILES.xml` to load the `BuiltinQosLib::Generic.Security` profile:

```
<qos_profile name="Alice" base_name="BuiltinQosLib::Generic.Security" is_
↪default_qos="true">
    <domain_participant_qos>
    ...
```

   Please note that this change affects both the publisher and subscriber applications.

2. Make sure that this profile loads the Governance Document that we defined in *Hands-On 2: Defining Your System's Security Requirements*, which configures payload protection by setting the `data_protection_kind` to `ENCRYPT`.

8.2.1 Analyzing RTPS Packets in Wireshark

1. Rerun your publisher and subscriber as explained in *Running the Applications*. Specify *Domain 1* with the **-d** option in the command line. You should see communication:

   Linux

   Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

   Subscriber:

```
$ ./objs/x64Linux3gcc5.4.0/PatientMonitoring_subscriber -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 101}]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 102}]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 103}]
PatientMonitoring subscriber sleeping up to 1 sec...
```

   macOS

   Publisher:

```
$ ./objs/<architecture>/PatientMonitoring_publisher -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

   Subscriber:

```
$ ./objs/x64Linux3gcc5.4.0/PatientMonitoring_subscriber -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 101}]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 102}]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 103}]
PatientMonitoring subscriber sleeping up to 1 sec...
```

   Windows

   Publisher:

```
> objs\<architecture>\PatientMonitoring_publisher.exe -d 1
Writing PatientMonitoring, count 0
Writing PatientMonitoring, count 1
Writing PatientMonitoring, count 2
Writing PatientMonitoring, count 3
```

Subscriber:

```
> objs\<architecture>\PatientMonitoring_subscriber.exe -d 1
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 101}]
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 102}]
PatientMonitoring subscriber sleeping up to 1 sec...
PatientMonitoring subscriber sleeping up to 1 sec...
[patient_condition: {heart_rate: 103}]
PatientMonitoring subscriber sleeping up to 1 sec...
```

2. Go back to Wireshark and look at the *serializedData* field of DATA submessages. You can check that the payload is now encrypted. Therefore it cannot be decoded as ASCII plaintext and trying to do so results in garbled output, as seen here:

Notice that the length of the data field has increased. The reason is that the *serializedData* field now includes a header and a footer with cryptographic information. The *CryptoHeader* includes information such as an identifier for the key used to encrypt the payload. The *CryptoFooter* includes the generated message authentication code (MAC). This additional information allows the receiver to decrypt the message and verify its integrity. For further details, see Securing DDS Messages on The Wire in the *RTI Security Plugins User's Manual*.

You may also notice that the field named *encapsulation options* has changed to `0x0002`. Although a *CryptoHeader* is now at the beginning of the *serializedData*, Wireshark does not have enough information to parse it. In fact, Wireshark does not know whether or not the *serializedData* is encrypted. For this reason, it always displays the *encapsulation header* (*encapsulation kind* and *encapsulation options*) at the beginning of the *serializedData* field. In this case, the payload is encrypted and these four bytes correspond to the *transformation kind* field in the *CryptoHeader*, where `0x00000002` means that the AES-128 GCM transformation has been applied (the default for `ENCRYPT`).

The following diagram depicts the structure of an RTPS packet protected with payload encryption:

Congratulations! You have verified that your payload is safe from a potential eavesdropper in your network.

8.3 Troubleshooting

- My Wireshark captures are hard to read because message-type coloring is not active:

  Make sure that your Wireshark contains the coloring rules for the RTPS protocol (see How To configure Wireshark to show RTPS packets with specific colors from our Knowledge Base).

- When I run the publisher/subscriber, they don't communicate:

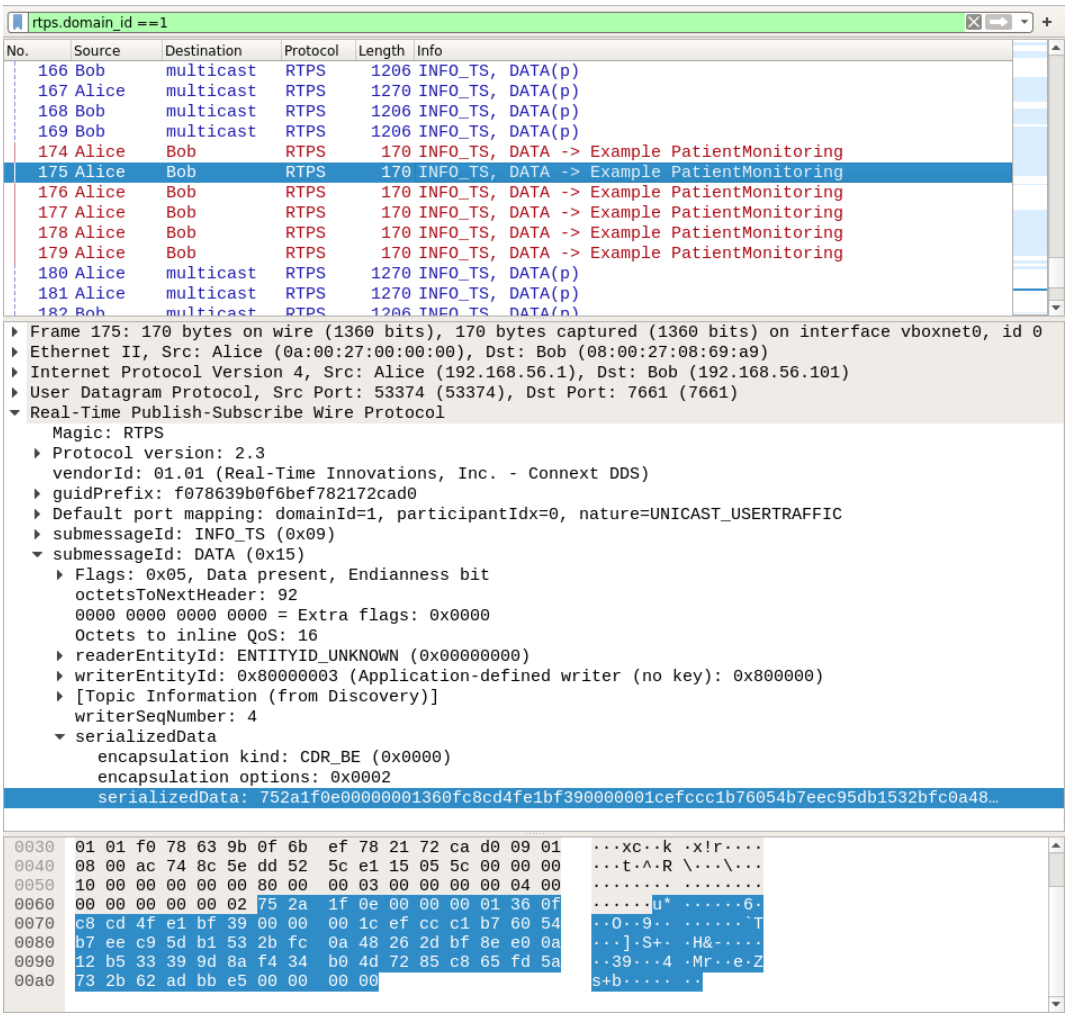


Figure 8.3: Wireshark capture of communication protected with `data_protection_kind` set to `ENCRYPT`. The payload is encrypted.

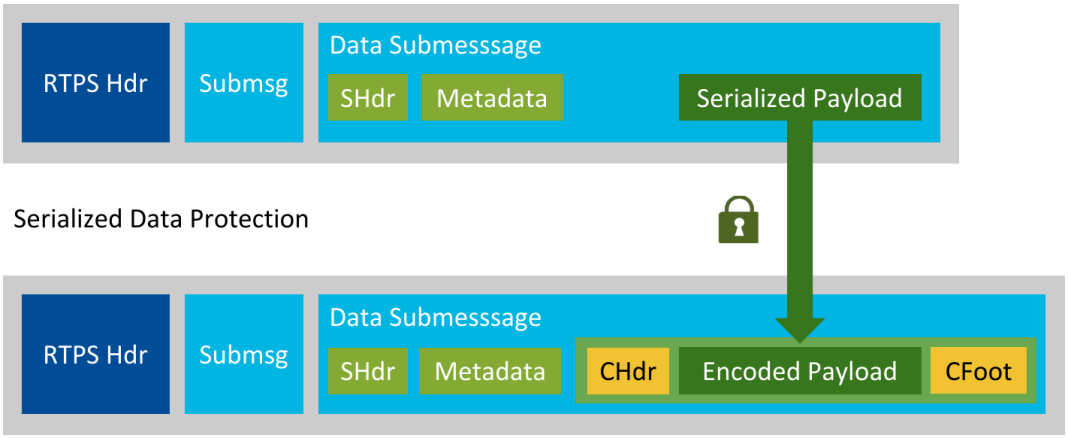


Figure 8.4: Structure of an RTPS packet with payload protection kinds, configured by the `<data_protection_kind>` rule.

Make sure that the Governance Document is the same for both applications (publisher and subscriber). Otherwise, communication won't work. Remember to sign the Governance Document after modifying it and to restart your applications so changes take effect (see *Signing the Governance Document*).

- When I run the publisher/subscriber, I get this error:

```
RTI_Security_AccessControl_create_participant:participant not allowed:␣
↪no rule found; default DENY
```

Make sure your applications are running in the allowed domain, in this case *Domain 1* (requirement from *Hands-On 3: Defining the DomainParticipant Permissions*).

- When I run the publisher/subscriber, I get this error:

```
RTI_Security_AccessControl_validate_remote_permissions:failed to␣
↪validate permissions
```

Make sure both your publisher and subscriber have permissions signed by the same Permissions CA.

- When I run the publisher/subscriber, I get this error:

```
RTI_Security_AccessControl_get_participant_sec_attributes:failed to␣
↪verify governance document signature
```

Make sure that the Governance Document has been signed by the right Permissions CA.

- I cannot capture traffic from the loopback interface on a Windows system:

Recording on the loopback interface on a Windows system may require additional software (see How to capture traffic from the loopback interface using Wireshark in Windows from our Knowledge Base).

Chapter 9

Next Steps

Congratulations! You have enabled Security Plugins in your *Connext* applications. In these exercises, you have protected your *DDS Domain* and sensitive *Topics* with a Governance Document specifying both domain-level and topic-level rules. You have also defined the permissions of your *DomainParticipants*, allowing them to access only the data and resources they need for their legitimate purpose. Finally, you have taken control of the security infrastructure by creating cryptographic identities for your secure *DomainParticipants*.

As you can see, enabling Security Plugins requires trivial or no changes to existing *Connext* applications. It just requires modifying the QoS profiles (usually an XML file) to point to the security artifacts and the Security Plugins library, and configuring some environment variables before running your applications. Nevertheless, keep in mind that enabling security always has an impact on the resources needed by your applications and their performance.

Now you're ready to start implementing. You may want to consult the RTI Security Plugins User's Manual as a reference. In particular, make sure to read these chapters:

- Building and Running Security Plugins-Based Applications
- Best Practices

Chapter 10

Copyrights and Notices

© 2017-2026 Real-Time Innovations, Inc. All rights reserved. April 2026

Trademarks

RTI, Real-Time Innovations, Connext, Connext Drive, NDDS, the RTI logo, 1RTI and the phrase, "Your Systems. Working as one." are registered trademarks, trademarks or service marks of Real-Time Innovations, Inc. All other trademarks belong to their respective owners.

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished solely under and subject to RTI's standard terms and conditions available at https://www.rti.com/terms and in accordance with your License Acknowledgement Certificate (LAC) and Maintenance and Support Certificate (MSC), except to the extent otherwise agreed to in writing by RTI.

Securing a distributed, embedded system is an exercise in user risk management. RTI expressly disclaims all security guarantees and/or warranties based on the names of its products, including Security Plugins and Security Plugins SDK *(RTI Security Plugins SDK)*. Visit https://www.rti.com/terms/ for complete product terms and an exclusive list of product warranties.

Third-Party Software

RTI software may contain independent, third-party software or code that are subject to third-party license terms and conditions, including open source license terms and conditions. Copies of applicable third-party licenses and notices are located at community.rti.com/documentation. IT IS YOUR RESPONSIBILITY TO ENSURE THAT YOUR USE OF THIRD-PARTY SOFTWARE COMPLIES WITH THE CORRESPONDING THIRD-PARTY LICENSE TERMS AND CONDITIONS.

Notices

Deprecations and Removals

Deprecated means that the item is still supported in this release, but will be removed in a future release. *Removed* means that the item is discontinued or no longer supported.

Any deprecations or removals noted in RTI's documentation serve as notice under the Real-Time Innovations, Inc., Maintenance Policy #4220 and/or any other agreements by and between RTI and customer regarding maintenance and support of RTI's software. RTI's current standard terms and support and maintenance policies are available at https://www.rti.com/terms.

Technical Support Real-Time Innovations, Inc. 232 E. Java Drive Sunnyvale, CA 94089 Phone: (408) 990-7444 Email: support@rti.com Website: https://support.rti.com/