RTI Security Plugins

Getting Started Guide

Version 5.3.0

Printed in U.S.A. First printing.
June 2017.

Trademarks

Copy and Use Restrictions

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 RTI Connext DDS Secure, RTI Security Plugins, and RTI Security Plugins SDK. Visit rti.com/terms for complete product terms and an exclusive list of product warranties.

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/

Contents

1 Introduction 1

2 Paths Mentioned in Documentation 3

3 Download Instructions 4

4 Installation Instructions 5

4.1 UNIX-Based Systems 5

4.2 Windows Systems 6

5 License Management 6

5.1 Installing the License File 6

5.2 Adding or Removing License Management 8

6 Restrictions when Using RTI Security Plugins 8

6.1 When to Set Security Parameters 8

6.2 Mixing Libraries Not Supported 8

7 Authentication 8

7.1 Configuration Properties Common to All Authentication Plugins 13

7.2 Re-Authentication 13

7.2.1 Supporting Re-Authentication in Custom Plugins 13

7.3 Protecting Participant Discovery 14

7.3.1 Supporting TrustedState in Custom Plugins 14

8 Access Control 14

8.1 Related Governance Attributes 16

8.1.1 Default Attributes 16

8.1.2 enable_join_access_control 16

8.1.3 enable_read/write_access_control 16

9 Cryptography 16

9.1 Related Governance Attributes 17

9.1.1 rtps_protection_kind 17

9.1.2 Other Protection Kinds 17

9.1.3 metadata_protection_kind 17

9.1.4 Endpoint Compatibility 18

9.1.5 discovery_protection_kind 18

9.1.6 enable_discovery_protection 18
9.1.7 enable_liveliness_protection 18
10 Logging 18
11 Support for OpenSSL Engines 22
12 Support for RTI Persistence Service 22
13 RTPS-HMAC-Only Mode 23
14 What's Different from the OMG Security Specification 24
14.1 Differences Affecting Builtin Plugins to be Addressed by Next DDS Security Specification 24
14.1.1 General 24
14.1.1.1 BuiltinTopicKey_t Type Definition 24
14.1.2 Authentication 24
14.1.2.1 SHA256 Applied to Derived Shared Secret 24
14.1.3 Cryptography 24
14.1.3.1 Secure Volatile Endpoints Use Submessage Protection 24
14.1.3.2 Secure Volatile Endpoints Transformation Kind 24
14.1.3.3 Additional Authenticated Data 24
14.1.4 Logging 25
14.1.4.1 Wrong Facility Value for Logging Plugin 25
14.2 Differences Affecting Builtin Plugins 25
14.2.1 General 25
14.2.1.1 Support for Infrastructure Services 25
14.2.1.2 Configuration 25
14.2.2 Access Control 25
14.2.2.1 check_remote_topic 25
14.2.2.2 Protection Kinds 25
14.2.2.3 Immutability of Publisher Partition QoS in Combination with Non-Volatile Durability Kind 25
14.2.3 Cryptography 26
14.2.3.1 Behavior when is_rtps_protected is Set to True 26
14.3 Differences Affecting Custom Plugins 26
14.3.1 Authentication 26
14.3.1.1 Revocation 26
14.3.2 Access Control 26
14.3.2.1 check_local_datawriter_register_instance 26
14.3.2.2 check_local_datawriter_dispose_instance 26
14.3.2.3 check_remote_datawriter_register_instance 26
14.3.2.4 check_remote_datawriter_dispose_instance 26
14.3.2.5 Revocation 26
14.3.2.6 PermissionsToken 26
14.3.3 Tagging 27
A Quick Reference: Governance File Settings 29

Welcome to RTI Security Plugins

1 Introduction

RTI® Security Plugins introduces a robust set of security capabilities, including authentication, encryption, access control and logging. Secure multicast support enables efficient and scalable distribution of data to many subscribers. Performance is also optimized by fine-grain control over the level of security applied to each data flow, such as whether encryption or just data integrity is required.

This release of *Security Plugins* includes partial support for the DDS Security specification from the Object Management Group (OMG)[1]. This support allows *DomainParticipants* to authenticate and authorize each other before initializing communication, and then encode and decode the communication traffic to achieve confidentiality, message authentication, and data integrity.

Specifically, these features are now supported:

- Authentication can now be done as part of the *RTI Connext® DDS* discovery process to ensure that DomainParticipants validate each other's identity.
- Access Control permissions checking can now be done as part of the *Connext DDS* discovery process to ensure that *DomainParticipants*, *DataWriters*, and *DataReaders* have the appropriate permissions to exist and match with each other. Domain governance can now be done during entity creation to ensure the right security attributes are applied to the right *DomainParticipants*, *DataWriters*, and *DataReaders*.
- Cryptographic operations can now be done as part of *Connext DDS* steady-state communication to ensure confidentiality, message authentication, and data integrity.
- Logging operations can now be done using the Logging Plugin. There are options to print the log messages to standard output or a file, distribute log messages over a DDS topic, and control the verbosity level of the log messages.
- The above features are supported in the RTI core middleware in the C, C++, Java, and .NET programming languages.

The following DDS Security features are *not* supported:

- Revocation of identities and permissions
- Data tagging
- Instance-level permissions checking

1. http://www.omg.org/spec/DDS-SECURITY/1.0/

For descriptions and examples of the security configuration in this release, please consult the **hello_security** examples under the **rti_workspace/*version*/examples/connext_dds/[c, cpp, java, csharp]** directory.

To use *Security Plugins*, you will need to create private keys, identity certificates, governance, and permission files, as well as signed versions for use in secure authenticated, authorized, and/or encrypted communications.

If you are new to the world of internet security, see this link:

- https://en.wikipedia.org/wiki/Public-key_cryptography

Fundamentally, if you want to deploy a secure system, your organization will need to have an in-house security expert. Just using *Security Plugins* is not sufficient. It is a tool that can build secure systems, but you do have to use it (configure it) to meet your requirements. If used incorrectly, systems deployed with *Security Plugins* may not meet the security requirements of a project.

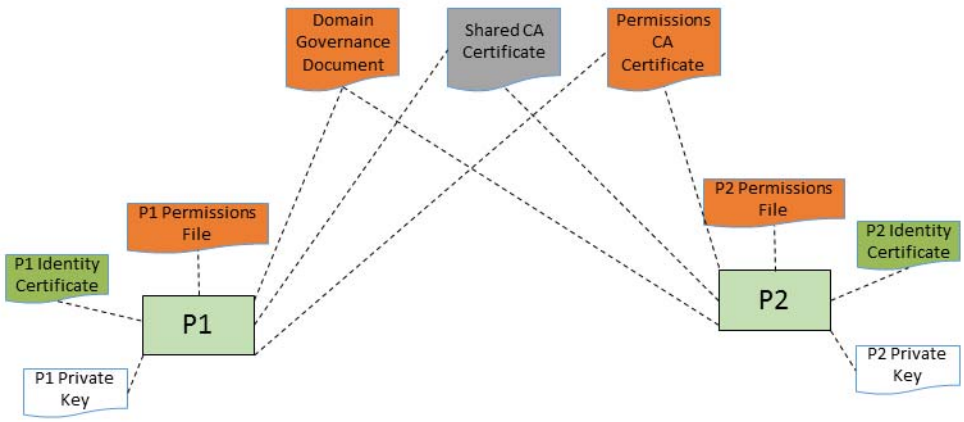
The *Security Plugins* bundle includes a set of builtin plugins that implement those defined by the DDS Security specification. It is also possible to implement new custom plugins by using the *Security Plugins SDK* bundle (for more information, please contact **support@rti.com**).

You should know that the *Security Plugins* use the same technology as most of the world's eCommerce, so if you have ever purchased something on the internet, the same technology protecting your purchase is used by *Security Plugins* to protect data exchange.

As an end user, you need to have the following files that an application using *Security Plugins* needs to communicate in a secure DDS domain:

- **Keys.** Each participant has a Private Key and Identity Certificate pair that is used in the authentication process.
- **Shared CA** has signed participant public keys. Participants must also have a copy of the CA certificate (also known as Identity Certificate Authority Certificate).
- **Permissions File** specifies what domains/partitions the *DomainParticipant* can join, what topics it can read/write, and what tags are associate with the readers/writers.
- **Domain Governance** specifies which domains should be secured and how.
- **Permissions CA** has a signed participant permission file, as well as the domain governance document. Participants must have a copy of the permissions CA certificate (also known as Permissions Authority Certificate).

Figure 1.1 **Configuring & Deploying DDS Security**



2 Paths Mentioned in Documentation

The documentation refers to:

- <NDDSHOME>

  This refers to the installation directory for *Connext DDS*.

  The default installation paths are:

  - Mac® OS X systems:

    /Applications/rti_connext_dds-*version*

  - UNIX™-based systems, non-*root* user:

    /home/*your user name*/rti_connext_dds-*version*

  - UNIX-based systems, *root* user:

    /opt/rti_connext_dds-*version*

  - Windows® systems, user without Administrator privileges:

    <your home directory>*\rti_connext_dds-*version

  - Windows systems, user with Administrator privileges:

    C:\Program Files\rti_connext_dds-*version* (for 64-bits machines) or
    C:\Program Files (x86)\rti_connext_dds-*version* (for 32-bit machines)

  You may also see $NDDSHOME or %NDDSHOME%, which refers to an environment variable set to the installation path.

  Wherever you see <NDDSHOME> used in a path, replace it with your installation path.

  Note for Windows Users: 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\rtiddsgen"
  ```

  or if you have defined the NDDSHOME environment variable:

  ```
  "%NDDSHOME%\bin\rtiddsgen"
  ```

- RTI Workspace directory, **rti_workspace**

  The RTI Workspace is where all configuration files for the applications and example files are located. All configuration files and examples are copied here the first time you run *RTI Launcher* or any script in **<NDDSHOME>/bin**. The default path to the RTI Workspace directory is:

  - Mac OS X systems:

    /Users/*your user name*/rti_workspace

  - UNIX-based systems:

    /home/*your user name*/rti_workspace

  - Windows systems:

    ***your Windows documents folder*\rti_workspace**

    Note: '*your Windows documents folder*' depends on your version of Windows.
    For example, on Windows 7, the folder is **C:\Users\your user name\Documents**; on Windows Server 2003, the folder is **C:\Documents and Settings\your user name\Documents**.

You can specify a different location for the **rti_workspace** directory. See the *RTI Core Libraries Getting Started Guide* for instructions.

❑ <path to examples>

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 these examples as **<path to examples>.** Wherever you see <path to examples>, replace it with the appropriate path.

By default, the examples are copied to **rti_workspace/*version*/examples**

So the paths are:

- Mac OS X systems:

  /Users/*your user name*/rti_workspace/*version*/examples

- UNIX-based systems:

  /home/*your user name*/rti_workspace/*version*/examples

- Windows systems:

  ***your Windows documents folder*\rti_workspace*version*\examples**

  Note: '*your Windows documents folder*' is described above.

You can specify that you do not want the examples copied to the workspace. See the *RTI Connext DDS Core Libraries Getting Started Guide* for instructions.

3 Download Instructions

Download *Security Plugins* from the RTI Support Portal, accessible from https://support.rti.com/.

Security Plugins also requires OpenSSL®, which is available from RTI's Support Portal, or you may obtain it from another source.

You will need your username and password to log into the portal; these are included in the letter confirming your purchase or evaluation copy. If you do not have this letter, please contact **license@rti.com**.

Once you have logged into the portal, select the **Downloads** link, then select the appropriate version of *Security Plugins* and OpenSSL for your platform.

If you need help with the download process, contact **support@rti.com**.

❑ **Security Plugins** can be downloaded in the following packages:

- **rti_security_plugins-5.3.0-eval-<*target architecture*>.rtipkg**
  includes the compiler-independent *Security Plugins* dependencies (documentation, headers, and the libraries used by RTI tools and services) for the host platform and the *Security Plugins* libraries you will link against for your target platform.

❑ **OpenSSL**

- OpenSSL distribution files for RTI tools and services follow the naming convention: **openssl-<*version*>-host-<*host platform*>.rtipkg**.
- OpenSSL distribution files to link against your application follow the naming convention: **openssl-<*version*>-target-<*target architecture*>.tar.gz** (or **.zip** on Windows systems).

The OpenSSL version number should be 1.0.2j or above (see the *RTI Security Plugins Release Notes* for the currently supported version). Architecture names are described in the *RTI Connext DDS Core Libraries Platform Notes*. For example:

- Bundle with distribution files for RTI tools and services:
  openssl-1.0.2j-host-x64Win64.rtipkg

- Bundle with distribution files to link against your application:
  openssl-1.0.2j-target-x64Win64VS2013.zip

4 Installation Instructions

You do not need administrator privileges. All directory locations are meant as examples only; adjust them to suit your site.

These instructions assume you are installing *Security Plugins* 5.3.0 and OpenSSL 1.0.2j. See the *RTI Security Plugins Release Notes* for the currently supported versions.

4.1 UNIX-Based Systems

1. Install the *RTI Connext DDS* host and target bundles, as described the *RTI Connext DDS Core Libraries Getting Started Guide*.
2. Install the *Security Plugins* package. Use the package installer, just as you did for the *RTI Connext DDS* target bundles in step 1.
   - **rti_security_plugins-5.3.0-eval-<*target architecture*>.rtipkg**

   (Where <*target architecture*> is one of the supported platforms, see the *RTI Connext DDS Core Libraries Platform Notes*).

   After installation, the security header files and libraries will be under **include/ndds/security** and **lib/<*target architecture*>**, respectively.
3. Install an OpenSSL host package from RTI: **openssl-1.0.2j-host-<*host platform*>.rtipkg**.
4. Install an OpenSSL target package from RTI: **openssl-1.0.2j-target-<*target architecture*>.tar.gz**.
   a. Make sure you have GNU's version of the tar utility, **gtar** (which handles long file names), and GNU's version of the unzip utility, **gunzip**.
   b. Move the downloaded OpenSSL distribution file to a directory of your choice, such as **/local/rti**, and change to that directory:

      ```
      > cd /local/rti
      ```

   c. Use **gunzip** to uncompress the OpenSSL file. (This is not the same as the OpenSSL host package in the previous step.) For example (your filename may be different):

      ```
      > gunzip openssl-1.0.2j-target-armv7aQNX6.6.0qcc_cpp4.7.3.tar.gz
      ```

   d. Use **gtar** to extract the distribution from the uncompressed file. For example:

      ```
      > gtar xvf openssl-1.0.2j-target-armv7aQNX6.6.0qcc_cpp4.7.3.tar
      ```

      This will extract files into **/local/rti/openssl-1.0.2j**.

   e. Include the resulting **/bin** directory in your PATH. For example, assuming you want to use the "release" version of the OpenSSL libraries (enter the command all on one line):

      ```
      > setenv PATH
      /local/rti/openssl-1.0.2j/armv7aQNX6.6.0qcc_cpp4.7.3/release/bin:$ {PATH}
      ```

   f. If linking dynamically, include the resulting **/lib** directory in your LD_LIBRARY_PATH. For example, assuming you want to use the "release" version of the OpenSSL libraries (enter the command all on one line):

      ```
      > setenv LD_LIBRARY_PATH
        /local/rti/openssl-1.0.2j/armv7aQNX6.6.0qcc_cpp4.7.3/release/lib:$PATH
      ```

   g. To verify your installation, enter:

      ```
      > openssl version
      ```

You should see a response similar to:

```
OpenSSL 1.0.2j
```

5. Your *Security Plugins* distribution may require a license. See License Management (Section 5).

4.2 Windows Systems

1. Install *RTI Connext DDS* host and target bundles on top of each other, as described the *RTI Connext DDS Core Libraries Getting Started Guide*.
2. Install the *Security Plugins* package. Use the package installer, just as you did for the *RTI Connext DDS* target bundles in step 1.

   - **rti_security_plugins-5.3.0-eval-<*target architecture*>.rtipkg**

   (Where <*target architecture*> is one of the supported platforms, see the *RTI Connext DDS Core Libraries Platform Notes*).

   After installation, the security header files and libraries will be under **include/ndds/security** and **lib/<target architecture>**, respectively.
3. Install an OpenSSL host package from RTI: **openssl-1.0.2j-host-<*host platform*>.rtipkg**.
4. Install an OpenSSL target package from RTI: **openssl-1.0.2j-target-<*target architecture*>.zip**.
   a. Right-click the distribution file and extract the contents in a directory of your choice.
   b. Add the resulting **bin** directory to your **Path** environment variable:

      c:\rti\openssl-1.0.2j\<*target architecture*>\release\bin

      (If you need help with this process, please see the *RTI Connext DDS Core Libraries Getting Started Guide*.)
   c. To verify your installation, open a command prompt and enter:

      ```
      > openssl version
      ```

      You should see a response similar to:

      ```
      OpenSSL 1.0.2j
      ```
5. Your *Security Plugins* distribution may require a license. See License Management (Section 5).

5 License Management

Most package types (Professional, Basic, and Evaluation) require a license file in order to run.

If your distribution requires a license file, you will receive one from RTI via email.

If you have more than one license file from RTI, you can concatenate them into one file.

A single license file can be used to run on any architecture and is not node-locked. You are not required to run a license server.

5.1 Installing the License File

Save the license file in any location of your choice; the locations checked by the plugin are listed below. You can also specify the location of your license file in *RTI Launcher's* **Installation** tab. Then *Launcher* can copy the license file to the installation directory or to the user workspace.

Each time your application starts, it will look for the license file in the following locations until it finds a valid license. (The properties are in the PropertyQosPolicy of the *DomainParticipant*.)

1. A property called **com.rti.serv.secure.license_string**. The value for this property can be set to the content of a license file. (This may be necessary if a file system is not supported on your platform.)

2. A property called **dds.license.license_string**. (Only if you have an evaluation version of *Connext DDS Professional*.)

   The above two **license_string** properties can be set to the content of a license file. (This may be necessary if a file system is not supported on your platform.) You can set the property either in source code or in an XML file.

   If the content of the license file is in XML, special characters for XML need to be escaped in the license string. Special characters include: quotation marks (") (replace with "), apostrophes (') (replace with '), greater-than (>) (replace with >), less-than (<) (replace with <), and ampersands (&) (replace with &).

   Example XML file:

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>dds.license.license_string</name>
                <value>contents of license file</value>
            </element>
        </value>
    </property>
</participant_qos>
```

3. A property called **com.rti.serv.secure.license_file**.

4. A property called **dds.license.license_file**. (Only if you have an evaluation version of *Connext DDS Professional*.)

   The above two **license_file** properties can be set to the location (full path and filename) of a license file. (This may be necessary if a default license location is not feasible and environment variables are not supported.) You can set the property either in source code or in an XML file.

   Example XML to set **dds.license.license_file**:

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>dds.license.license_file</name>
                <value>path to license file</value>
            </element>
        </value>
    </property>
</participant_qos>
```

5. In the location specified in the environment variable RTI_LICENSE_FILE, which you may set to point to the full path of the license file, including the filename.

   Note: When you run any of the scripts in the **<NDDSHOME>/bin** directory, this automatically sets the RTI_LICENSE_FILE environment variable (if it isn't already set) prior to calling the executable. It looks for the license file in two places: your **rti_workspace** directory and the installation directory (NDDSHOME). (See Paths Mentioned in Documentation (Section 2).)

6. If you are running any of the tools/services as executables via **NDDSHOME/bin/<executable script>** or through *Launcher*:
   a. In your **rti_workspace/<version>** directory, in a file called **rti_license.dat**.
   b. In your **rti_workspace** directory, in a file called **rti_license.dat**.
   c. In <NDDSHOME> (the *Connext DDS* installation directory), in a file called **rti_license.dat**.
7. If you are running your own application linked with *Connext DDS* libraries:
   a. In your current working directory, in a file called **rti_license.dat**.
   b. In <NDDSHOME> (the *Connext DDS* installation directory), in a file called **rti_license.dat**.

As *Connext DDS* attempts to locate and read your license file, you may (depending on the terms of the license) see a message with details about your license.

If the license file cannot be found or the license has expired, your application may be unable to initialize, depending on the terms of the license. If that is the case, your application's call to **DomainParticipantFactory.create_participant()** will return null, preventing communication.

If you have any problems with your license file, please email **support@rti.com**.

5.2 Adding or Removing License Management

If your license file changes—for example, you receive a new license for a longer term than your original license—you do not need to reinstall.

However, if you switch from a license-managed distribution of to one of the same version that does not require license management, or visa versa, RTI recommends that you first uninstall your original distribution before installing your new distribution. Doing so will prevent you from inadvertently using a mixture of libraries from multiple installations.

6 Restrictions when Using RTI Security Plugins

6.1 When to Set Security Parameters

In *RTI Connext DDS*, you must set the security-related participant properties (see Table 7.1) *before* you create a participant. You cannot create a participant without security and then call **DomainParticipant::set_qos()** with security properties, even if the participant has not yet been enabled.

6.2 Mixing Libraries Not Supported

Mixing static and dynamic RTI libraries (e.g., using RTI static core libraries and dynamic Security Plugins libraries) is not supported for user applications.

7 Authentication

Authentication is the process of making sure a *DomainParticipant* is who it claims to be. Loading any security plugins will configure the *DomainParticipant* to authenticate a newly discovered remote participant before initiating endpoint discovery with that participant. Authentication is done via a series of inter-participant challenge and response messages. These messages perform mutual authentication, so the end result is that this participant authenticates the remote participant and vice-versa. If this participant fails to authenticate the remote participant, the remote participant is ignored. Otherwise, this participant initiates endpoint discovery with the remote participant and communication resumes as normal.

Table 7.1 lists the properties that you can set for Authentication and enabling security in general. These properties are configured through the *DomainParticipant*'s PropertyQosPolicy.

Table 7.1 **Properties for Enabling Security and Configuring Authentication**

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
com.rti.serv.load_plugin *(note: this does not take a prefix)*	***Required*** The prefix name of the security plugin suite that will be loaded by *Connext DDS*. For example: **com.rti.serv.secure**. You will use this string as the prefix to the property names. Setting this value to non-NULL will also configure the *DomainParticipant* to attempt authentication with newly discovered remote participants. Note: you can load only one security plugin suite. Default: NULL unless using the **Generic.Security** builtin profile
library	***Only required if linking dynamically*** Must be set to the dynamic library that implements the security plugin suite. If using *Connext DDS*'s provided security plugin suite, you must set this value to **nddssecurity**. This library and the dependent OpenSSL libraries must be in your library search path (pointed to by the environment variable **LD_LIBRARY_PATH** on UNIX/Solaris systems, **Path** on Windows systems, **LIBPATH** on AIX systems, **DYLD_LIBRARY_PATH** on Mac OS systems). Default: NULL unless using **Generic.Security** builtin profile
create_function	***Only required if linking dynamically*** Must be set to the security plugin suite creation function that is implemented by the library. If using *Connext DDS*'s provided security plugin suite, you must set this value to **RTI_Security_PluginSuite_create**. Default: NULL unless using **Generic.Security** builtin profile
create_function_ptr	***Only required if linking statically*** Must be set to the security plugin suite creation function implemented by the library. If using *Connext DDS*'s provided security plugin suite, you must set this value to the stringified pointer value of **RTI_Security_PluginSuite_create**, as demonstrated in the **hello_security** examples. Note: you cannot set this value in an XML profile. You must set it in code. Default: NULL
authentication. shared_secret_algorithm	***Optional*** The algorithm used to establish a shared secret during authentication. The options are dh and ecdh for (Elliptic Curve) Diffie-Hellman. If two participants discover each other and they specify different values for this algorithm, the algorithm that is used is the one that belongs to the participant with the lower-valued participant_key. **Note: ecdh** does not work with static OpenSSL libraries when using Certicom Security Builder Engine. Default: **ecdh**

Table 7.1 **Properties for Enabling Security and Configuring Authentication**

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description

authentication.ca_file	***Required*** A string that specifies the fully-qualified path and name of the file containing Identity Certificate Authority certificates. The file should be in PEM format. This Identity Certificate Authority is used for signing authentication certificate files.
	OpenSSL should generate this file using commands such as the following. For an example **openssl.cnf** file, refer to the example **cert** folder: **rti_workspace/version/examples/dds_security/cert**. Note: You will need to modify this file to match your certificate folder structure and Identity Certificate Authority desired configuration.
	RSA:
	`% openssl genrsa -out cakey.pem 2048`
	`% openssl req -new -key cakey.pem -out ca.csr -config openssl.cnf`
	`% openssl x509 -req -days 3650 -in ca.csr -signkey cakey.pem -out cacert.pem`
	`% echo 01 > ca.srl`
	DSA:
	`% openssl dsaparam 2048 > dsaparam`
	`% openssl gendsa -out cakeydsa.pem dsaparam`
	`% openssl req -new -key cakeydsa.pem -out dsaca.csr -config openssldsa.cnf`
	`% openssl x509 -req -days 3650 -in dsaca.csr -signkey cakeydsa.pem -out cacertdsa.pem`
	ECDSA:
	`% openssl ecparam -name prime256v1 > ecdsaparam`
	`% openssl req -nodes -x509 -days 3650 -newkey ec:ecdsaparam -keyout cakeyECdsa.pem -out cacertECdsa.pem -config opensslECdsa.cnf`
	Note: When running the above commands, you may run into these OpenSSL warnings:
	❑ `WARNING: can't open config file: [default openssl built-inpath]/openssl.cnf`
	To resolve this, set the environmental variable OPENSSL_CONF with the path to the **openssl.cnf** file you are using.
	❑ `unable to write 'random state'`
	To resolve this, set the environmental variable RANDFILE with the path to a writable file.
	Two participants that want to securely communicate with each other must use the same Identity Certificate Authority.
	Default: NULL

Table 7.1 **Properties for Enabling Security and Configuring Authentication**

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description

authentication.crl_file	***Optional*** A string that specifies the fully qualified path and name of the file containing a Certificate Revocation List. The file should be in PEM format. This Certificate Revocation List keeps track of untrusted X.509 certificates.
	OpenSSL should generate this file using commands such as the following. For an example **opensslECdsa.cnf** file, refer to the example **cert** folder: **rti_workspace/version/examples/dds_security/cert**. Note: You will need to modify this file to match your certificate folder structure and Certificate Revocation List desired configuration:
	`% touch indexECdsa.txt`
	`% echo 01 > crlnumberECdsa`
	`% openssl ca -config opensslECdsa.cnf -batch -revoke peerRevokedECdsa.pem`
	`% openssl ca -config opensslECdsa.cnf -batch -gencrl -out democaECdsa.crl`
	In this example:
	❑ **crlnumberECdsa** is the database of revoked certificates. This file should match the **crlnumber** value in **opensslECdsa.cnf**.
	❑ **peerRevokedECdsa.pem** is the **certificate_file** of a revoked *DomainParticipant*.
	❑ **democaECdsa.crl** should be the value of the **crl_file** property.
	If **crl_file** is set to NULL, no CRL is checked, and all valid certificates will be considered trusted.
	If **crl_file** is set to an invalid CRL file, the *DomainParticipant* creation will fail.
	If **crl_file** is set to a valid CRL file, the CRL will be checked upon *DomainParticipant* creation and upon discovering other *DomainParticipants*. Creating a *DomainParticipant* with a revoked certificate will fail. If ParticipantA uses a certificate that does not appear in ParticipantA's CRL but does appear in ParticipantB's CRL, then ParticipantB will reject and ignore ParticipantA. Changes in the CRL will not be enforced until the *DomainParticipant* using the CRL is deleted and recreated.
	Default: NULL

authentication.private_key_file	***Required*** A string that specifies the fully-qualified path and name of the file containing a private key. The file should be in PEM format.
	After generating the **ca_file**, OpenSSL should generate this file using commands such as the following:
	RSA:
	`% openssl genrsa -out peer1key.pem 2048`
	DSA:
	`% openssl dsaparam 2048 > dsaparam`
	`% openssl gendsa -out peer1keydsa.pem dsaparam`
	ECDSA:
	`% openssl ecparam -name prime256v1 > ecdsaparam1`
	`% openssl req -nodes -new -newkey ec:ecdsaparam1 -config example1ECdsa.cnf -keyout peer1keyECdsa.pem -out peer1reqECdsa.pem`
	peer1reqECdsa.pem will be used to generate the certificate file. This property value should be set to **peer1keyECdsa.pem**.
	Default: NULL

Table 7.1 **Properties for Enabling Security and Configuring Authentication**

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description

authentication.certificate_file	***Required*** A string that specifies the fully-qualified path and name of the file containing an Identity Certificate. The file should be in PEM format. An Identity Certificate is required for secure communication.
	To generate this file, first generate the **ca_file** and **private_key_file**. Then create a blank **index.txt** file and a serial file whose contents are **01**. The names of these files will depend on the contents of the **openssl*.cnf** file. Then use OpenSSL to generate the certificate file using commands such as the following. For example **.cnf** files, refer to the example **cert** folder: **rti_workspace/version/examples/dds_security/cert**. Note: You will need to modify this file to match your certificate folder structure and Identity Certificate desired configuration:
	RSA:
	`% openssl req -config example1.cnf -new -key peer1key.pem -out user.csr`
	`% openssl ca -config openssl.cnf -days 365 -in user.csr -out peer1.pem`
	DSA:
	`% openssl req -config example1dsa.cnf -new -key peer1keydsa.pem -out dsauser.csr`
	`% openssl ca -config opensslдsa.cnf -days 365 -in dsauser.csr -out peer1dsa.pem`
	ECDSA:
	Generate **peer1reqECdsa.pem** using the instructions for private_key_file.
	`% openssl ca -batch -create_serial -config opensslECdsa.cnf -days 365 -in peer1reqECdsa.pem -out peer1ECdsa.pem`
	Notes:
	❑ **openssl((EC)dsa).cnf** must have the same **countryName**, **stateOrProvinceName**, and **localityName** as the example **.cnf** files.
	❑ Example **.cnf** files of different participants must have different commonNames.
	Default: NULL

1. Assuming you used 'com.rti.serv.secure' as the alias to load the plugin. If not, change the prefix to match the string used with com.rti.serv.load_plugins. This prefix must begin with 'com.rti.serv.'

7.1 Configuration Properties Common to All Authentication Plugins

Table 7.1 lists a set of properties that are not exclusive to the shipped Security Plugins, but that will affect any Authentication Plugin.

Table 7.1 **Properties for Configuring Authentication Common to Any Authentication Plugin**

Property Name (prefix with 'dds.participant.trust_plugins.')	Property Value Description
authentication_timeout.sec	***Optional*** Controls the maximum time in seconds that an ongoing authentication can remain without completing. After this timeout expires, the authentication process is cancelled, and associated resources are released. Default: 60 seconds
authentication_request_delay.sec	***Optional*** Controls the delay in seconds before sending an authentication_request to the remote participant. For more information, please see Re-Authentication (Section 7.2). Default: 5 seconds

7.2 Re-Authentication

The *Security Plugins* support securely re-authenticating remote Participants as an extension to the DDS Security specification. This is needed in scenarios where there is an asymmetric liveliness loss.

Asymmetric liveliness loss occurs between two Participants A and B when Participant A loses liveliness with B, and therefore cleans up all the associated state, while B still keeps the authenticated state. As B keeps an authenticated state from A, it will not accept new authentication messages from A. Without the ability to re-authenticate, asymmetric liveliness loss will lead to communication not recovering. The *Security Plugins* address this problem by including re-authentication capability as an extension to the RTI Security specification.

In *Security Plugins*, if Participant A that has not completed an ongoing authentication with a Participant B after an specific period, it will send an com.rti.sec.auth.request message that includes a nonce[1] to Participant B. This message will give a hint to Participant B that Participant A is pending Authentication with Participant B. This specific period is configured by the property **dds.participant.trust_plugins.authentication_request_delay.sec**, see Table 7.1, "Properties for Configuring Authentication Common to Any Authentication Plugin".

When Participant B receives a com.rti.sec.auth.request message, it will check if it already has a valid completed authentication with Participant A. If that is the case, that could mean that an asymmetric liveliness loss has occurred. In order to verify that the authentication request is legitimate, the two Participants will now conduct a whole Authentication process that includes the nonce received as part of the triggering com.rti.sec.auth.request. Only if this secondary authentication succeeds, the old state will be removed in Participant B and replaced with the new one, allowing for discovery to complete again and communication to recover. If this secondary authentication fails, no change will be made in Participant B and the old authenticated session will be kept.

Because the old authenticated state is kept until the new authentication has successfully completed, the *Security Plugins* re-authentication is robust against attackers trying to bring down an existing authentication.

7.2.1 Supporting Re-Authentication in Custom Plugins

To support re-authentication in plugins other than *Security Plugins*, the following APIs must be implemented by the custom plugin:

1. Nonce: an arbitrary number used only once in a cryptographic communication, used to avoid replay attacks.

- **begin_auth_request()**
- **process_auth_request()**

For more details, see the **RTI_SecurityPlugins_BuildableSourceCode_Instructions.txt** file included with *Security Plugins SDK*.

7.3 Protecting Participant Discovery

Participant discovery is sent through an unsecure channel. Consequently, additional mechanisms need to be put in place to make sure the received information comes from a legitimate participant. In *Security Plugins*, the mechanism for protecting the participant discovery information is known as TrustedState.

Security Plugins TrustedState is an RTI extension to the DDS Security Authentication specification that covers two limitations in the DDS Security Specification:

- Vulnerability in the protocol: The lack of a standardized mechanism for validating that the Participant Discovery information received by DDS actually matches the one authenticated.
- Participant Discovery Data is immutable after authentication. This prevents functionality such as updating IP addresses.

Security Plugins TrustedState is a digest of the participant discovery data, plus information that unambiguously identifies the current local participant state, plus information that unambiguously identifies the current authentication session. TrustedState is exchanged as part of the authentication process as a vendor extension. Once the authentication completes, involved participants will validate received participant discovery information against the received TrustedState. This way, participants can be sure that the received participant discovery comes from the authenticated participant.

In order to securely propagate participant discovery changes after authenticating the remote participant, the *Security Plugins* use the participant's identity private key to sign the participant discovery data plus some additional information identifying the local participant state (and which is consistent with the one serialized in the TrustedState). This signature is then serialized as a property in the participant discovery data. This way, other participants can validate that the update is legitimate by verifying the received participant discovery against the participant's public key.

7.3.1 Supporting TrustedState in Custom Plugins

To secure participant discovery updates through the TrustedState mechanism in plugins other than the *Security Plugins*, the following APIs must be implemented by the custom plugin:

- **set_local_participant_trusted_state()**
- **verify_remote_participant_trusted_state()**
- **get_max_signature_size()**
- **private_sign()**
- **verify_private_signature()**

For more information, please see the **RTI_SecurityPlugins_BuildableSourceCode_Instructions** file included in the *Security Plugins* SDK.

8 Access Control

Access Control consists of two components: governance and permissions checking. Governance is the process of configuring locally created *DomainParticipants, Topics, DataWriters,* and *DataReaders* to perform the right amount of security for the right use case. Permissions checking is the process of making sure locally created and remotely discovered entities are allowed to do what they want to do. Both governance and permissions checking are enforced by XML documents that are signed by a permissions certif-

icate authority that may or may not be the same as the identity certificate authority that signs identity certificates. The XSD definitions of these documents are in **$(NDDSHOME)/resource/schema/dds_security_governance.xsd** and **dds_security_permissions.xsd**.

Examples of these documents are in **rti_workspace/version/examples/dds_security/xml/**, see **Governance.xml** and **PermissionsA.xml**. Use these files just as a reference, you will need to update their content/create new files to match your system configuration (domains, topics, and used identity certificates) before signing them. To specify that you want to use these XML files, add the properties in Table 8.1, "Properties for Configuring Access Control" to the DDS_DomainParticipantQos property:

Table 8.1 **Properties for Configuring Access Control**

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
access_control. permissions_authority_file	***Required*** A string that specifies the fully-qualified path and name of the file containing Permissions Certificate Authority certificates. The file should be in PEM format. This Permissions Certificate Authority is used for signing access control governance and permissions XML files and verifying the signatures of those files. The Permissions Certificate Authority file may or may not be the same as the Identity Certificate Authority file, but both files are generated in the same way. See Table 7.1, "Properties for Enabling Security and Configuring Authentication" for the steps to generate this file. Two participants that want to securely communicate with each other must use the same Permissions Certificate Authority. Default: NULL
access_control. governance_file	***Required*** The signed file that specifies the level of security required per domain and per topic. To sign an XML document with a Permissions Certificate Authority, run the following OpenSSL command (enter this all on one line): `openssl smime -sign -in Governance.xml -text -out signed_Governance.p7s -signer cacert.pem -inkey cakey.pem` Then set this property value to **signed_Governance.p7s**. Default: NULL
access_control. permissions_file	***Required*** The signed file that specifies the access control permissions per domain and per topic. The <subject_name> element identifies the *DomainParticipant* to which the permissions apply. Each subject name can only appear in a single <permissions> section within the XML Permissions document. The contents of the <subject_name> element should be the X.509 subject name for the *DomainParticipant*, as given in the "Subject" field of its Identity Certificate. A <permissions> section with a subject name that does not match the subject name given in the corresponding Identity Certificate will be ignored. To sign an XML document with a Permissions Certificate Authority, run the following OpenSSL command (enter this all on one line): `openssl smime -sign -in PermissionsA.xml -text -out signed_PermissionsA.p7s -signer cacert.pem -inkey cakey.pem` Then set this property value to **signed_PermissionsA.p7s**. The signed permissions document only supports validity dates between 1970010100 and 2038011903. Any dates before 1970010100 will result in an error, and any dates after 2038011903 will be treated as 2038011903. Currently, Connext DDS will not work if the system time is after January 19th, 2038. Default: NULL

1. Assuming you used ‘com.rti.serv.secure’ as the alias to load the plugin. If not, change the prefix to match the string used with com.rti.serv.load_plugins. This prefix must begin with 'com.rti.serv.'

8.1 Related Governance Attributes

The Access Control governance attributes follow the DDS Security specification as much as possible. This section provides some clarifications that are not described in the specification.

8.1.1 Default Attributes

If no matching domain or topic rule is found, the default rules apply. The default rules have minimum security, so all attributes are FALSE or NONE except for **allow_unauthenticated_participants**.

8.1.2 enable_join_access_control

This attribute controls whether or not remote participant permissions are checked when a remote participant is discovered.

8.1.3 enable_read/write_access_control

These attributes control whether or not *DataReader* or *DataWriter* permissions are checked. If **enable_read_access_control** is TRUE for a given topic, the local permissions are enforced on locally created *DataReaders* of that topic, and the remote permissions are enforced on remotely discovered *DataReaders* of that topic. Similar logic applies to **enable_write_access_control** and *DataWriters*.

9 Cryptography

Cryptography is the process of making sure no adversaries can manipulate or eavesdrop on communication. To prevent manipulation of data, set the governance attribute **rtps_protection_kind** to SIGN. To prevent eavesdropping of data, set the governance attribute **metadata_protection_kind or data_protection_kind** to ENCRYPT.

The following properties in the DDS_DomainParticipantQos **property** configure Cryptography:

Table 9.1 **Property for Configuring Cryptography**

Property Name (prefix with ‘com.rti.serv.secure.’)[1]	Property Value Description
cryptography. max_blocks_per_session	***Optional*** The number of message blocks that can be encrypted with the same key material. Whenever the number of blocks exceeds this value, new key material is computed. The block size depends on the encryption algorithm. You can specify this value in decimal, octal, or hex. This value is an unsigned 64-bit integer. Default: 0xffffffffffffffff

Table 9.1 **Property for Configuring Cryptography**

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
cryptography. encryption_algorithm	***Optional*** The algorithm used for encrypting and decrypting data and metadata. The options are **aes-128-gcm**, **aes-192-gcm**, and **aes-256-gcm** ("gcm" is Galois/Counter Mode (GCM) authenticated encryption). The number indicates the number of bits in the key and the block. Participants are not required to set this property to the same value in order to communicate with each other. In the Domain Governance document, a "protection kind" set to ENCRYPT will use GCM, and a "protection kind" set to SIGN will use the GMAC variant of this algorithm. Default: **aes-128-gcm**
cryptography. max_receiver_specific_macs	***Optional*** The maximum number of receiver-specific Message Authentication Codes (MACs) that are appended to an encoded result. For example, if this value is 32, and the Participant is configured to protect both RTPS messages and submessages, there could be 32 receiver-specific MACs in the result of **encode_datawriter_submessage**, and there could be another 32 receiver-specific MACs in the result of **encode_rtps_message.** If there are more than 32 receivers, the receivers will be assigned one of the 32 possible MACs in a round-robin fashion. Note that in the case of **encode_datawriter_submessage,** all the readers belonging to the same participant will always be assigned the same receiver-specific MAC. Setting this value to 0 will completely disable receiver-specific MACs. Default: 0. Range: [0, 3275]

1. Assuming you used 'com.rti.serv.secure' as the alias to load the plugin. If not, change the prefix to match the string used with com.rti.serv.load_plugins. This prefix must begin with 'com.rti.serv.'

9.1 Related Governance Attributes

This section provides some clarifications about the Cryptography governance attributes that are either not described or described differently in the specification.

9.1.1 rtps_protection_kind

SIGN and NONE are the only supported values for **rtps_protection_kind**. Setting **rtps_protection_kind** = NONE will cause the *DomainParticipant* to accept both encoded and unencoded incoming RTPS messages.

Setting **rtps_protection_kind** = SIGN will cause the *DomainParticipant* to append a GMAC to outgoing RTPS messages and reject incoming RTPS messages that do not have such a GMAC.

9.1.2 Other Protection Kinds

ENCRYPT and NONE are the only supported values of other protection kinds besides **rtps_protection_kind**.

9.1.3 metadata_protection_kind

Since **metadata_protection_kind** controls the EndpointSecurityAttribute **is_submessage_protected**, and a submessage may consist of both metadata and data, **metadata_protection_kind** applies to both metadata and data. An endpoint will accept both encoded and unencoded incoming submessages regardless of the setting of **metadata_protection_kind**.

9.1.4 Endpoint Compatibility

A *DataWriter* with **metadata_protection_kind** = NONE and **data_protection_kind** = NONE is not compatible with a *DataReader* with **metadata_protection_kind** = ENCRYPT or **data_protection_kind** = ENCRYPT. A *DataReader* will not successfully receive samples from a *DataWriter* that has a different **data_protection_kind** setting from the *DataReader*.

9.1.5 discovery_protection_kind

discovery_protection_kind is partially supported. Currently supported values are NONE and ENCRYPT.

9.1.6 enable_discovery_protection

Indicates if the meta information for the entities matching the associated topic rule shall be sent using secure builtin topics or the regular builtin topics. This includes both builtin discovery topics and the service request channel (used for sending Topic Queries and Locator Reachability Response messages).

9.1.7 enable_liveliness_protection

This is a *Security Plugins* extension to the DDS Security specification. If set, it determines if the liveliness information for the entities matching the associated topic rule shall be sent using a secure liveliness topic or the regular liveliness topic. If not set, the configuration for liveliness will be determined by the value set for **enable_discovery_protection**.

10 Logging

Logging is the process of notifying the user of security events. This release supports printing log messages to the standard output, printing log messages to a file, distributing log messages over DDS, and adjusting the verbosity level of the log messages. By default, log messages are printed to the standard output, and the verbosity level of the log messages is WARNING_LEVEL.

The following properties in the DDS_DomainParticipantQos **property** configure Logging:

Table 10.1 **Properties for Configuring Logging**

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
logging.log_file	***Optional*** The file that log messages are printed to. Default: NULL
logging.log_level	***Optional*** The logging verbosity level. All log messages at and below the **log_level** setting will be logged. Possible values: ❑ 0: emergency ❑ 1: alert ❑ 2: critical ❑ 3 (default): error ❑ 4: warning ❑ 5: notice ❑ 6: informational ❑ 7: debug Default: 3 (error)

Table 10.1 **Properties for Configuring Logging**

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
logging.distribute.enable	***Optional*** Controls whether security-related log messages should be distributed over DDS. To subscribe to the log messages, run *rtiddsgen* on **resource/idl/builtin_logging_type.idl**. Create a *DataReader* of type **DDSSecurity::BuiltinLoggingType** and topic **DDS:Security:LogTopic**. The *DataReader* must be allowed to subscribe to this topic according to its *DomainParticipant's* permissions file. Boolean. Default: false.
logging.distribute.profile	***Optional*** QoS Library and QoS profile used to create logging-related entities (*Publisher*, *Topic* and *DataWriter*). Must be a string of the format **QosLibraryName::QosProfileName**. String. Default: empty string (uses default QoS profile).
logging.distribute.writer_history_depth	***Optional*** History depth (in samples) of the logging *DataWriter*. Integer. Default: 64.
logging.distribute.writer_timeout	***Optional*** Number of milliseconds to wait before giving up trying to write a log message. This property overwrites the **max_blocking_time** QoS of the logging *DataWriter*. Integer. Default: 5000 milliseconds.
logging.distribute.queue.size	***Optional*** Size of the logging thread queue, in bytes. Integer. Default: 50688.
logging.distribute.queue.message_count_max	***Optional*** Maximum number of log messages in the logging queue. Integer. Default: 64.
logging.distribute.queue.message_size_max	***Optional*** Maximum serialized size of a log message in the logging queue. Integer. Default: 792.
logging.distribute.thread.message_threshold	***Optional*** Number of bytes to preallocate for the logging message string in the logging thread, beyond which dynamic allocation will occur. Integer. Default: 256.
logging.distribute.thread.plugin_method_threshold	***Optional*** Number of bytes to preallocate for the plugin method string in the logging thread, beyond which dynamic allocation will occur. Integer. Default: 256.
logging.distribute.thread.message_threshold	***Optional*** Number of bytes to preallocate for the plugin class string in the logging thread, beyond which dynamic allocation will occur. Integer. Default: 256.

1. Assuming you used 'com.rti.serv.secure' as the alias to load the plugin. If not, change the prefix to match the string used with com.rti.serv.load_plugins. This prefix must begin with 'com.rti.serv.'

Table 10.1 lists security-related events and the log messages they generate.

Table 10.2 **Log Messages**

Event	Log Level	Message
Failed to allocate memory	EMERGENCY	insufficient memory
AllowUnauthenticatedParticipants = false, and discovered remote participant that is unauthenticable, i.e. has not enabled security	CRITICAL	unauthenticated remote participant [participant ID] denied
AllowUnauthenticatedParticipants = true, and discovered remote participant that is either unauthenticable or fails authentication	WARNING	allowing unauthenticated participant [participant ID]
Received invalid X509 certificate, from either remote or local participant	CRITICAL	failed to decode certificate
Couldn't verify certificate's signature against neither the certificate of the Identity Certificate Authority nor any alternative CAs	CRITICAL	failed to verify certificate
Certificate appears in Certificate Revocation List	CRITICAL	certificate revoked
Upon receiving HandshakeReplyMessageToken or HandshakeFinalMessageToken, couldn't verify challenge's signature against peer's certificate. Peer likely has mismatched private and public keys, so it’s an imposter.	CRITICAL	failed to verify challenge signature
Couldn't verify permissions or governance file signature against neither the certificate of the Permissions Authority nor any alternative permissions authorities	CRITICAL	!PKCS7_verify: document signature verification failed. Make sure document was signed by the right permissions authority.
Received signed permissions or governance document that is not an XML document	ALERT	received invalid signed [permissions or governance] document
Couldn't parse the permissions file for some reason, such as duplicate grants for the same subject name or no grant for the intended subject name	ALERT	failed to parse permissions file
Couldn't parse the governance file for some reason	ALERT	failed to parse governance file
Denied participant because there is a deny rule explicitly prohibiting the participant	CRITICAL	participant not allowed: deny rule found
Denied participant because there is no rule for the participant, and the default is to deny	CRITICAL	participant not allowed: no rule found; default DENY
Denied writer or reader because there is a deny rule explicitly prohibiting the writer or reader	CRITICAL	endpoint not allowed: deny rule found
Denied writer or reader because there is no rule for the writer or reader, and the default is to deny	CRITICAL	endpoint not allowed: no rule found; default DENY
Parsed publish/subscribe rule in permissions file that does not apply to the writer/reader because no topic expressions match the writer/reader's topic	WARNING	This publish/subscribe rule doesn't apply because none of the rule's topic expressions match the endpoint's topic name of [topic name]
Parsed publish/subscribe rule in permissions file that does not apply to the writer/reader because even though there's a matching topic expression, there are no matching partition expressions	WARNING	This publish/subscribe rule doesn't apply because none of the rule's partition expressions match with any of the endpoint's partitions
Received authenticated content that has been tampered with, i.e. EVP_DecryptFinal_ex failed because the GCM or GMAC tag verification failed	ALERT	DecryptFinal failed. Possible GCM authentication failure.

Table 10.2 **Log Messages**

Event	Log Level	Message
Received submessage encrypted with a key whose MasterKeyId hasn't yet been exchanged via CryptoToken	DEBUG	received submessage from an endpoint that discovered me but that I haven't discovered yet; dropping submessage hoping it will be repaired. It will not be repaired if the endpoint did not properly share its MasterKeyId in its CryptoToken
Writing a log message over the LogTopic fails due to insufficient logging queue size	LOCAL[1]	Failed to write log message of size = [message size] because the logging queue is full. Try to increase logging.distribute.queue. message_count_max, which is currently [message_count_max].

1. This log message can be viewed by configuring the verbosity of the NDDS_Config_Logger.

11 Support for OpenSSL Engines

RTI Security Plugins support the option of using an OpenSSL engine. The following property in the DDS_DomainParticipantQos **property** configures the usage of OpenSSL engines:

Table 11.1 **Properties for Configuring OpenSSL Engines**

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
openssl_engine	***Optional*** The dynamic library that implements an OpenSSL engine. If this property value is not set, then the RTI Security Plugins will use native OpenSSL code with its default engine. Otherwise, you must set this value to the filename, excluding the "lib" prefix and the file extension, of the dynamic library that implements the engine, and you must set your $LD_LIBRARY_PATH or %path% environment variable to include the dynamic library and any of its dependent libraries. Failure to load the engine, due to an incorrect $LD_LIBRARY_PATH or otherwise, will result in failure to create the *DomainParticipant*. The engine will perform all security operations, including encryption, HMAC, and Diffie-Hellman. The value of this property for the first *DomainParticipant* of the application will be the value for all other *DomainParticipants* in the application. Setting this property to a different value for subsequent *DomainParticipants* will not be effective. Default: not set

1. Assuming you used 'com.rti.serv.secure' as the alias to load the plugin. If not, change the prefix to match the string used with com.rti.serv.load_plugins. This prefix must begin with 'com.rti.serv.'

One example of an OpenSSL engine is Certicom Corp.'s *Security Builder Engine for OpenSSL*, which supports the architecture armv7aQNX6.6.0qcc_cpp4.7.3. Usage of Certicom requires their dynamically-loaded libraries (which RTI does not provide) and your LD_LIBRARY_PATH environment variable must include:

```
$RTI_OPENSSLHOME/release/lib/:$CERTICOM_SBENGINEHOME/tools/sb/sb-$(CERTICO-
MOS)/lib/:$CERTICOM_SBENGINEHOME/lib/$(CERTICOMOS)
```

where RTI_OPENSSLHOME is the **installation directory/armv7aQNX6.6.0qcc_cpp4.7.3** of the OpenSSL distributed by RTI, CERTICOM_SBENGINEHOME is the installation directory of Certicom *Security Builder Engine*, and CERTICOMOS is Certicom's architecture corresponding to RTI's armv7aQNX6.6.0qcc_cpp4.7.3, e.g. qnx6.5_armv7. The authentication.shared_secret_algorithm ecdsa-ecdh does not work with static OpenSSL libraries when enabling Certicom *Security Builder Engine.*

12 Support for RTI Persistence Service

RTI's security solution may be used in conjunction with *RTI Persistence Service*. To store persisted data encrypted, *Persistence Service* must use a configuration whose **participant_qos** includes security properties for 1) dynamically loading the security libraries and 2) using a Governance document that sets **data_protection_kind** to ENCRYPT for the desired topics (or * for all topics). The %PATH% or $LD_LIBRARY_PATH environment variable must include RTI and OpenSSL DLLs or libraries.

If *Persistence Service* stores encrypted data, it also stores the PRSTDataWriter's encryption key along with the rest of the writer's metadata. If *Persistence Service* shuts down and restarts with the same configuration, the new PRSTDataWriter will discard its normally random key and use the old PRSTDataWriter's key, which it securely exchanges with user *DataReaders* to allow them to correctly decrypt the data. Key rotation works seamlessly in this scenario because the stored encrypted data includes not only the payload but also the metadata necessary to decrypt it, including the **session_id** used to derive the session key from

the master key. When the encryption key is stored, it is stored encrypted. The key of this encryption is a function of an optional user-specified property, and the Cryptography Plugin implementation determines the encryption algorithm. In RTI's default plugin implementation, the encryption algorithm involves SHA-256 and AES-256-GCM.

Attempting to use an insecure *Persistence Service* to restore encrypted data or a secure *Persistence Service* to restore plain-text data will result in a graceful failure to create *Persistence Service*.

The following properties in the *Persistence Service* **participant_qos** or **persistence_group.datawriter_qos property** configure the *Persistence Service*'s usage of security:

Table 12.1 **Properties for Configuring Secure Persistence Service**

Property Name	Property Value Description
dds.data_writer. history.key_material_key	***Optional*** The basis of the key material used to encrypt the PRSTDataWriter's key material. This property may be specified in either the DomainParticipantQos or the DataWriterQos. Attempting to restore encrypted data using the wrong **key_material_key** will result in an informative log message and failure to create *Persistence Service*. Default: undisclosed non-NULL

13 RTPS-HMAC-Only Mode

The *Security Plugins* library includes an alternative set of "RTPS-HMAC-Only" plugins. These plugins allow RTPS messages to be signed with a user-provided HMAC key while disabling all other security features (authentication, access control and encryption). To set up the behavior of the RTPS-HMAC-Only mode, refer to Table 13.1, "Properties for Configuring HMAC-Only Mode".

Table 13.1 **Properties for Configuring HMAC-Only Mode**

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
hmac_only.enabled	***Optional*** Enables or disables the HMAC-only mode. Default: false
hmac_only.cryptography.key	***Required*** Sets the static HMAC key used to compute message signatures. The HMAC key can be either a plain text string or an arbitrary binary string. Empty keys (either string or binary) are not allowed. The maximum HMAC key size is bounded by the maximum property size, controlled by the DomainParticipant resource limit **participant_property_string_max_length**. • Plain text HMAC keys are case sensitive, and must start with the prefix **str:** (e.g.: str:Some secret key string) • Binary HMAC keys must be provided as a sequence of upper- or lower-case hexadecimal digits prefixed by **hex:** (e.g.: hex:1489a95de3873df5). Default: not set
hmac_only.cryptography. max_blocks_per_session	***Optional*** For signing RTPS messages, HMAC-only mode uses a key derived from the HMAC key and a sessionId that is serialized as part of the signed RTPS message representation. This property sets the number of message blocks that can be signed with the same sessionId. The current message block size is fixed at 32 bytes. Default: 0xffffffffffffffff)

1. Assuming you use 'com.rti.serv.secure' as the alias to load the plugin. If not, change the prefix to match the string used with com.rti.serv.load_plugins. This prefix must begin with 'com.rti.serv.'

14 What's Different from the OMG Security Specification

This section describes differences between *RTI Security Plugins* 5.3.0 and the latest OMG DDS Security specification (Version 1.0).

14.1 Differences Affecting Builtin Plugins to be Addressed by Next DDS Security Specification

14.1.1 General

14.1.1.1 BuiltinTopicKey_t Type Definition

Section 7.2 in the specification defines BuiltinTopicKey_t as 16 octets. However, this will be revised in the next DDS Security specification. In particular, BuiltinTopicKey_t will remain an array of unsigned longs and the DDS Security specification will use a new type DDS_GUID_t (16 byte octet array) instead. *Security Plugins* follow this convention.

14.1.2 Authentication

14.1.2.1 SHA256 Applied to Derived Shared Secret

Section 9.3.2.3.2 in the specification defines how to compute the shared secret. However, this will be revised in the next DDS Security specification. In particular, the next version of the DDS Security specification will state that, regardless of the key agreement algorithm, the SharedSecret (see Table 42) shall be computed as the SHA256 hash of the derived shared secret computed by the key agreement algorithm. *Security Plugins* follow this convention.

14.1.3 Cryptography

14.1.3.1 Secure Volatile Endpoints Use Submessage Protection

The current DDS Security Specification only protects the content (i.e., the keys) and not the metadata of the Secure Volatile Endpoints. However, this will be revised in the next DDS Security specification. In particular, the next version of the DDS Security specification will protect the whole Secure Volatile submessages. *Security Plugins* follow this convention.

14.1.3.2 Secure Volatile Endpoints Transformation Kind

Table 52 in the specification defines CRYPTO_TRANSFORMATION_KIND_AES128_GCM and CRYPTO_TRANSFORMATION_KIND_AES256_GCM as possible transformation kinds for the Secure Volatile Endpoints. However, this will be revised in the next DDS Security specification. In particular, the next version of the DDS Security specification will only use CRYPTO_TRANSFORMATION_KIND_AES256_GCM as protection kind. *Security Plugins* follow this convention.

14.1.3.3 Additional Authenticated Data

Sections 9.5.3.3.4.5 and 9.5.3.3.4.6 in the specification state that Additional Authenticated Data should be populated with some specific bytes. However, this will be revised in the next DDS Security specification. In particular, the next version of DDS Security specification will state that Additional Authenticated Data should be empty. *Security Plugins* follow this convention.

14.1.4 Logging

14.1.4.1 Wrong Facility Value for Logging Plugin

Section 9.6 in the specification defines 0x10 as the Facility value for Logging Plugin. However, this will be revised in the next DDS Security specification. In particular, the next version of DDS Security specification will define 0x0A (10) as the Facility value to use. *Security Plugins* follow this convention.

14.2 Differences Affecting Builtin Plugins

14.2.1 General

14.2.1.1 Support for Infrastructure Services

Section 7.1.1.4 in the specification describes the mechanism for preventing unauthorized access to data by infrastructure services. To support this capability, certain functions have an output parameter called **relay_only**. *Security Plugins* does not implement this mechanism.

14.2.1.2 Configuration

Tables 35 and 45 in the specification describe the properties used to configure the builtin plugins. *Security Plugins* support only a subset of these properties, and the properties have different names. For descriptions of the supported properties, see these tables in this document:

- Table 7.1, "Properties for Enabling Security and Configuring Authentication"
- Table 8.1, "Properties for Configuring Access Control"
- Table 9.1, "Property for Configuring Cryptography"
- Table 10.1, "Properties for Configuring Logging"
- Table 11.1, "Properties for Configuring OpenSSL Engines"
- Table 12.1, "Properties for Configuring Secure Persistence Service"

14.2.2 Access Control

14.2.2.1 check_remote_topic

Section 8.4.2.6.12 in the specification describes the **check_remote_topic()** operation. *RTI Security Plugins* do not implement this operation.

14.2.2.2 Protection Kinds

Section 9.4.1.2.1 in the specification describes the possible protection kinds as NONE, SIGN, and ENCRYPT. The rest of section 9.4.1.2 describes the various Domain Governance Document protection kind elements. In *Security Plugins*, **rtps_protection_kind** only supports NONE or SIGN; all other protection kinds only support NONE or ENCRYPT.

14.2.2.3 Immutability of Publisher Partition QoS in Combination with Non-Volatile Durability Kind

Section 7.3.5 in the specification states that for security reasons, the Publisher PartitionQos policy is immutable under certain circumstances. *Security Plugins* do not implement this constraint.

14.2.3 Cryptography

14.2.3.1 Behavior when is_rtps_protected is Set to True

The current DDS Security Specification states that RTPS protection should only be enforced for authenticated Participants. *Security Plugins* does not follow this convention; instead it enforces RTPS protection for all received RTPS messages, regardless of the source participant's authentication state. Consequently, if **is_rtps_protected** is set to true in the local participant, it will not accept RTPS messages from unauthenticated participants (being the only exception participant discovery messages), regardless of the value for **allow_unauthenticated_participants**.

14.3 Differences Affecting Custom Plugins

14.3.1 Authentication

14.3.1.1 Revocation

Section 8.3.2.10.1 in the specification describes the mechanism for revoking identities. *Security Plugins* do not implement this mechanism. This release supports looking up a certificate revocation list upon *DomainParticipant* creation and discovery.

14.3.2 Access Control

14.3.2.1 check_local_datawriter_register_instance

Section 8.4.2.6.7 in the specification describes the **check_local_datawriter_register_instance()** operation. *Security Plugins* do not implement this operation.

14.3.2.2 check_local_datawriter_dispose_instance

Section 8.4.2.6.8 in the specification describes the **check_local_datawriter_dispose_instance()** operation. *Security Plugins* do not implement this operation.

14.3.2.3 check_remote_datawriter_register_instance

Section 8.4.2.6.15 in the specification describes the **check_remote_datawriter_register_instance()** operation. *Security Plugins* do not implement this operation.

14.3.2.4 check_remote_datawriter_dispose_instance

Section 8.4.2.6.16 in the specification describes the **check_remote_datawriter_dispose_instance()** operation. *Security Plugins* do not implement this mechanism.

14.3.2.5 Revocation

Section 8.4.2.7.1 in the specification describes the mechanism for revoking permissions. *Security Plugins* do not implement this mechanism.

14.3.2.6 PermissionsToken

Table 10 in the specification mentions PermissionsToken as a new parameter in ParticipantBuiltinTopicData. *Security Plugins* 5.3.0 sends this parameter, but when receiving this parameter, it is not used in any Access Control functionality. The built-in Access Control plugin does not use PermissionsToken, so this issue only affects certain custom Access Control plugins.

14.3.3 Tagging

Section 8.7 in the specification defines the Data Tagging plugin. *Security Plugins* do not implement the Data Tagging plugin.

Appendix A Quick Reference: Governance File Settings

This table shows common security objectives and the Governance file settings necessary to achieve them. The highlighted cells indicate settings that increase security.

	Governance Parameter	Baseline	Enable Authentication and Access Control	MAC and Encrypt Discovery Data *	MAC Liveliness Messages (Protect Builtin Topic)	MAC and Encrypt Liveliness Messages [1] (Protect Builtin Topic)	MAC Data Messages	MAC and Encrypt Data Messages [1]	MAC Data and Metadata Messages	MAC and Encrypt Data and Metadata Messages [1]	MAC Entire RTPS packet (including header)	MAC Entire RTPS packet (including header) and encrypt-then-MAC data [1]	MAC Entire RTPS packet (including header) and encrypt-then-MAC data and metadata [1]	All Possible Protections [1]
Domain	allow_unauthenticated_participants	T	F	T	—	T	—	T	—	T	T	T	T	F
	enable_join_access_control	F	T	F	—	F	—	F	—	F	F	F	F	T
	discovery_protection_kind	N	N	E	—	N	—	N	—	N	N	N	N	E
	liveliness_protection_kind	N	N	N	—	E	—	N	—	N	N	N	N	E
	rtps_protection_kind	N	N	N	—	N	—	N	—	N	S	S	S	S
Topic	enable_discovery_protection	F	T	T	—	T [2]	—	F	—	F	F	F	F	T
	enable_read_access_control	F	T	F	—	F	—	F	—	F	F	F	F	T
	enable_write_access_control	F	T	F	—	F	—	F	—	F	F	F	F	T
	metadata_protection_kind	N	N	N	—	N	—	N	—	E	N	N	E	E
	data_protection_kind	N	N	N	—	N	—	E	—	N	N	E	N	N

1. Assumes that aes-gcm is the encryption algorithm
2. Alternatively, RTI Security Plugins enable_liveliness_protection extension can be enabled

Legend:

- T = TRUE
- F = FALSE
- N = NONE
- E = ENCRYPT
- S = SIGN
- — = unsupported