RTI Security Plugins

User's Manual

Version 6.0.1

Printed in U.S.A. First printing.
November 2019.

Trademarks

Copy and Use Restrictions

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

Chapter 1 Introduction

RTI® Security Plugins is 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 supported:

- Authentication can 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 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 be done as part of *Connext DDS* steady-state communication to ensure confidentiality, message authentication, and data integrity.
- Logging operations can be done using the Logging Plugin. There are options to print the log messages using NDDS_Config_Logger or an output file, distribute the log messages over a DDS topic, and control the verbosity level of the log messages.
- Data tagging can be done using the DataTagQosPolicy, and data tags can now be allowed or denied using the Permissions Document.

---

[1] http://www.omg.org/spec/DDS-SECURITY/1.1/

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
- Instance-level permissions checking

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, c++, java, cs]** 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**).

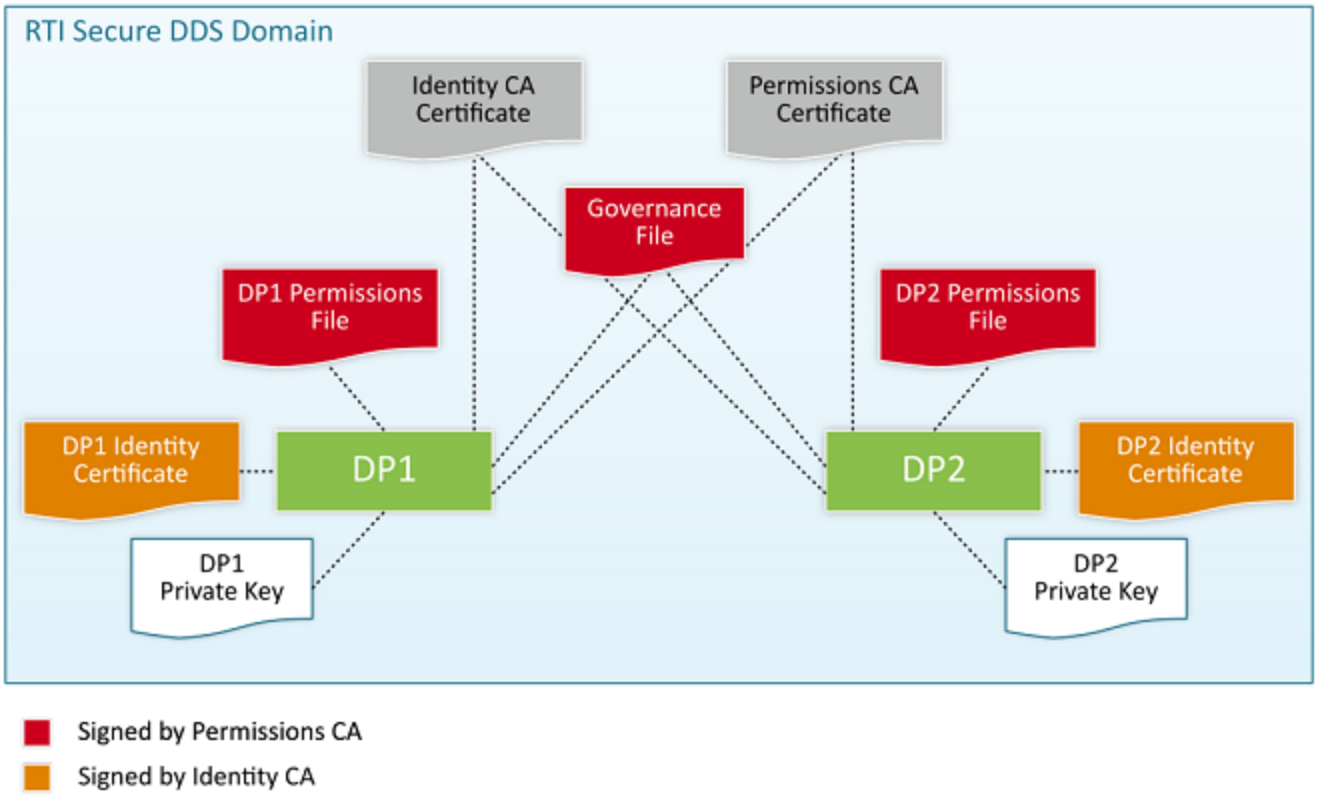
You should know that *Security Plugins* use the same technology as most of the world's eCommerce, so if you have ever purchased something on the internet, the same technology protecting your purchase is used by *Security Plugins* to protect data exchange.

As an end user, you need to have the following files, which 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: Artifacts required for RTI Security Plugins



Chapter 2 Paths Mentioned in Documentation

The documentation refers to:

- **<NDDSHOME>**

  This refers to the installation directory for *RTI® Connext® DDS*. The default installation paths are:

  - macOS® systems:
    /Applications/rti_connext_dds-6.0.1
  - UNIX-based systems, non-*root* user:
    /home/<*your user name*>/rti_connext_dds-6.0.1
  - UNIX-based systems, *root* user:
    /opt/rti_connext_dds-6.0.1
  - Windows® systems, user without Administrator privileges:
    <*your home directory*>\rti_connext_dds-6.0.1
  - Windows systems, user with Administrator privileges:
    C:\Program Files\rti_connext_dds-6.0.1

  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-6.0.1\bin\rtiddsgen"
```

Or if you have defined the **NDDSHOME** environment variable:

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

- ***<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:

  - macOS systems: **/Users/<*your user name*>/rti_workspace/6.0.1/examples**
  - UNIX-based systems: **/home/<*your user name*>/rti_workspace/6.0.1/examples**
  - Windows systems: ***<your Windows documents folder>*\rti_workspace\6.0.1\examples**

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

  Note: You can specify a different location for **rti_workspace**. You can also specify that you do not want the examples copied to the workspace. For details, see *Controlling Location for RTI Workspace and Copying of Examples* in the *RTI Connext DDS Installation Guide*.

Chapter 3 Restrictions when Using RTI Security Plugins

3.1 When to Set Security Parameters

In *Connext DDS*, you must set the security-related participant properties *before* you create a participant (see the tables in Authentication (Chapter 4 on page 11)). 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.

3.2 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; however, you can use *either* static or dynamic linking. The following sections explain what to be aware of and to avoid in the different scenarios.

The examples in this section are for Linux systems, but except for small differences in names, the same concepts apply to Windows and macOS systems as well.

Suppose you have a *Connext DDS*-based application **myApp**, and you want to use *Security Plugins* to protect the communication. The library dependency looks something like that in Figure 3.1: Library Dependency below.

Figure 3.1: Library Dependency

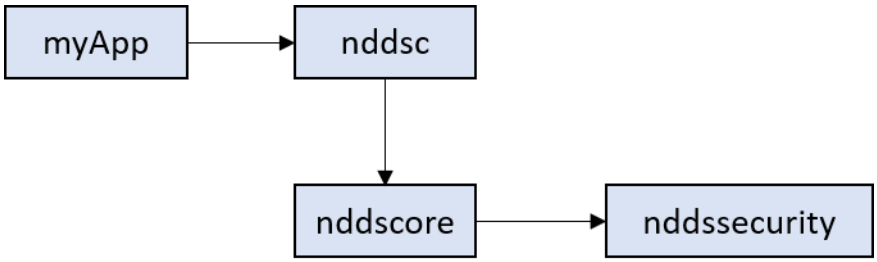
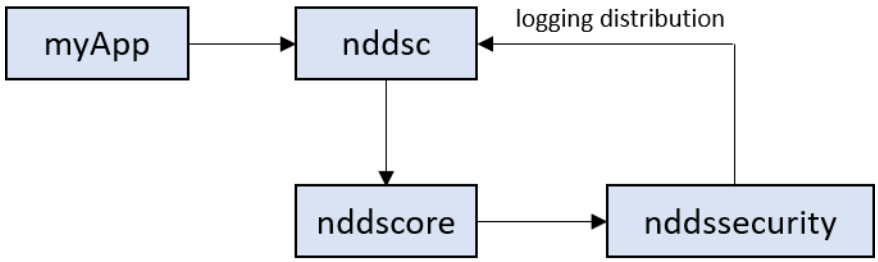


Figure 3.1: Library Dependency on the previous page is a simple and common situation, but in some cases you may end up with a circular dependency among the libraries. In particular, circular dependency may occur when you enable the *Security Plugins* logging distribution (see Logging (Chapter 7 on page 38)). In this case, *Security Plugins* performs calls to the *Connext DDS* core library, potentially creating a dangerous situation, as shown in Figure 3.2: Circular Library Dependency below.

Figure 3.2: Circular Library Dependency



3.2.1 Dynamic Linking

An easier and more flexible solution is to use dynamic linking. At run time, your application loads the *Connext DDS* libraries (on UNIX-based systems: **libnddsc.so**, **libnddscore.so**), and everything is controlled from the QoS defined in an XML file. To specify dynamic linking, use something like the following during the link phase of your application:

```
gcc -o myApp myApp.o -L$NDDSHOME/lib/$ARCH -lnddsc -lnddscore
```

Note: *Security Plugins* is *not* included in the list of required libraries, because it is dynamically loaded at run time from the participant QoS **com.rti.serv.secure.library**:

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>com.rti.serv.secure.library</name>
                <value>nddssecurity</value>
            </element>
        </value>
    </property>
</participant_qos>
```

Dynamic loading is particularly useful if, for example, you use your own security plugins library, because the library can be easily defined at run time through the QoS.

- When your application starts, the dynamic loader automatically loads the *Connext DDS* libraries (**libnddsc.so**, **libnddscore.so**) from your dynamic library search path.
- When the *DomainParticipant* is created and the QoS is set, the *Connext DDS* core libraries dynamically load the security library at run time. Because the *Security Plugins* library depends on

libnddscore.so, the dynamic loader knows that the library has already been loaded, and it automatically resolves the undefined symbols to use the currently loaded library.

- The security library then initializes and creates the Logging Plugin. The Logging Plugin initializes the DomainParticipantFactory (which is implemented in the *Connext DDS* core library) .

3.2.2 Static Linking

If you choose to statically link the RTI libraries, the mechanism for dynamic selecting and loading of *Security Plugins* is no longer available. Compared to dynamic linking, you need to pay attention to two things with static linking.

First, you need to include the *Security Plugins* library *and* the OpenSSL dependency libraries in the list of libraries needed during linking:

```
gcc -o myApp myApp.o -L$NDDSHOME/lib/$ARCH -lnddscz -lnddscorez -lnddssecurityz -lcryptoz -
lsslz
```

Second, in your code you need to manually tell *Connext DDS* the pointer to the function of the entry point of *Security Plugins* before you create the DomainParticipant, as shown in the following snippet:

```
#include "security/security_default.h"

 struct DDS_DomainParticipantQos participant_qos = DDS_DomainParticipantQos_INITIALIZER;
DDS_ReturnCode_t retcode;

retcode = DDS_DomainParticipantFactory_get_participant_qos_from_profile(
        DDS_TheParticipantFactory, &participant_qos, "AppQosLibrary::MyAppProfile",
profile);
if (retcode != DDS_RETCODE_OK) {
    // error: Unable to get default participant qos
    [...]
}

retcode = DDS_PropertyQosPolicyHelper_assert_pointer_property(
        &participant_qos.property,
        BUILTIN_PLUGIN_NAME ".create_function_ptr",
        RTI_Security_PluginSuite_create);
if (retcode != DDS_RETCODE_OK) {
    // error: Unable to assert create_function_ptr property
    [...]
}
/* Create the domain participant on domain ID 18 */
participant = DDS_DomainParticipantFactory_create_participant(
        DDS_TheParticipantFactory,
        18, /* Domain ID */
        &participant_qos, /* Qos */
        NULL, /* Listener */
        DDS_STATUS_MASK_NONE);
[...]
```

IMPORTANT: If you statically link *Security Plugins*, the QoS property **com.rti.serv.secure.library** will be silently ignored, if defined. *Security Plugins* is only set at compile time. There is no runtime selection.

The example above works for *Connext DDS* 6.0.0 and newer. For older versions, refer to the example code **examples/connext_dds/c/hello_security/src/HelloWorld_subscriber.c**.

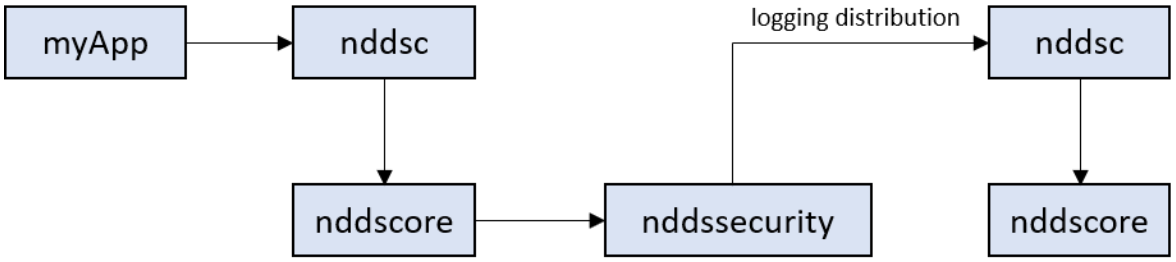
3.2.3 Mixed Linking

The last scenario is when you have your application statically linked with the *Connext DDS* library, but you dynamically select the *Security Plugins* library to load at run time.

> Mixing static and dynamic libraries is not valid, and you should avoid it.

You can easily end up in a mixed linking scenario without realizing the implications. For example, suppose you design your statically linked application without security in mind, then add security from the QoS. This scenario is not valid because when the runtime dynamic loader loads the *Security Plugins* library, it also loads a second copy of the *Connext DDS* core libraries in memory as shown by Figure 3.3: Mixed Library Linking below.

Figure 3.3: Mixed Library Linking



If you enable *Security Plugins* logging distribution, the moment the *Security Plugins* library is loaded, you will get an error message like the following:

```
DDS_DomainParticipantGlobals_initializeWorkerFactoryI:!Potential library mismatch.
This may happen if your application uses the static and the shared RTI core
libraries simultaneously.
For example, using the shared RTI Monitoring library
and linking statically with the RTI core libraries will cause this mismatch
```

If you don't have *Security Plugins* logging distribution enabled, your application might still work (because the detection of this condition is implemented in the initializer of the DDS DomainParticipantFactory), but **this configuration is not supported and you might end up with unexpected behavior at run time**.

3.3 Impact of Using Security Plugins

Enabling *Security Plugins* may affect the timing of *Connext DDS* discovery, causing your applications to behave slightly differently when starting your system. With *Security Plugins* enabled, two additional

processes need to happen before data is successfully exchanged between two applications: first, the two involved *DomainParticipants* need to complete authentication, which is a three-way handshake process; then, each one of the secured *DataWriters* and *DataReaders* need to exchange the key material for protecting the data. Endpoints need to exchange this key material so that protected (encrypted or signed) payloads and submessages can be decrypted and verified.

If protected data, such as user samples, arrive before the key material exchange is complete, this protected data is dropped by *Connext DDS*. Only samples exchanged over non-volatile, reliable channels (i.e., a channel with the Reliability QoS kind set to RELIABLE, and Durability QoS kind other than VOLATILE) will be resent if they're dropped due to incomplete key material exchange. Since key material exchange is required with *Security Plugins*, and it takes some additional time for this exchange to occur before endpoints begin accepting data, more data may be sent as repair traffic than in scenarios without *Security Plugins* enabled.

As a result, if your application starts writing data samples right after enabling the *DataWriter*, you may observe those initial samples to take longer to be received (if using RELIABLE reliability) or to not be received at all (if using BEST_EFFORT reliability), even if those samples were usually received when not using *Security Plugins*. (They could also have been lost even if not using security, if discovery was not completed at the time of writing the sample.)

Another consideration is that using the "Generic.Security" profile for enabling security also does some tuning to the reliability protocol parameters for endpoint discovery traffic. The goal of this tuning is to shorten discovery times; however, the configured parameters could be too aggressive for some systems, significantly increasing network traffic during the discovery phase. These parameters can be tuned down by explicitly configuring the following DiscoveryConfigQosPolicy's **publication_writer** and **subscription_writer** values: **fast_heartbeat_period** to 1 sec, **late_joiner_heartbeat_period** to 1 sec, and **max_heartbeat_retries** to 30.

Chapter 4 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 4.1 Properties for Enabling Security below, Table 4.2 DDS Security Properties for Configuring Authentication on page 13, and Table 4.3 RTI Security Plugins Properties for Configuring Authentication on page 16 list the properties that you can set for Authentication and enabling security in general. These properties are configured through the *DomainParticipant*'s PropertyQosPolicy.

Table 4.1 Properties for Enabling Security

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 some of 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

[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, followed by the '.' character.

Table 4.1 Properties for Enabling Security

Property Name (prefix with 'com.rti.serv.secure.') [1]	Property Value Description
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 macOS 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. Notes: • You cannot set this value in an XML profile. You must set it in code. • If this property is set to a value other than NULL, it will always take effect, even if create_function is also set to a value other than NULL. 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, followed by the '.' character.

Table 4.2 DDS Security Properties for Configuring Authentication

Property Name	Property Value Description

dds.sec.auth.identity_ca	***Required***
	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`
	DSA:
	`% openssl dsaparam 2048 > dsaparam`
	`% openssl gendsa -out cakeydsa.pem dsaparam`
	`% openssl req -new -key cakeydsa.pem -out dsaca.csr -config opensslдsa.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`
	If specifying the file name as the property value, this property value should be set to "file:" (no space after the colon), followed by the file name that appears after the "-out" parameter of the final openssl command (e.g., cacert.pem, cacertdsa.pem, or cacertECdsa.pem).
	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`
	Or:
	`unable to write 'random state'`
	To resolve the first issue, set the environmental variable OPENSSL_CONF with the path to the **openssl.cnf** file you are using. (See Installing OpenSSL in the Security Plugins Getting Started Guide.) To resolve the second issue, 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.
	The document should be in PEM format. You may specify either the file name or the document contents. If specifying the file name, the property value must have the prefix "file:" (no space after the colon), followed by the fully-qualified path and name of the file. If specifying the contents of the document, the property value must have the prefix "data:," (no space after the comma), followed by the contents inside the document. For example:
	`"data:,-----BEGIN CERTIFICATE-----\nabcdef\n-----END CERTIFICATE-----"`
	Note that the two "\n" characters are required.
	Default: NULL

Table 4.2 DDS Security Properties for Configuring Authentication

Property Name	Property Value Description

dds.sec.auth.private_key	***Required***
	The private key associated with the first certificate that appears in identity_certificate. After generating the **identity_ca**, 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**.
	The document should be in PEM format. You may specify either the file name or the document contents. If specifying the file name, the property value must have the prefix "file:" (no space after the colon), followed by the fully-qualified path and name of the file. If specifying the contents of the document, the property value must have the prefix "data:," (no space after the comma), followed by the contents inside the document. For example:
	`"data:,-----BEGIN PRIVATE KEY-----\nabcdef\n-----END PRIVATE KEY-----"`
	Note that the two “\n” characters are required.
	Default: NULL

Table 4.2 DDS Security Properties for Configuring Authentication

Property Name	Property Value Description

dds.sec.auth.identity_certificate	***Required***
	An Identity Certificate is required for secure communication.
	To generate this file, first generate the **identity_ca** and **private_key**. Then create a **serial** file whose contents are **01** and a blank **index.txt** file. 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.
	`% openssl ca -batch -create_serial -config openssl ECdsa.cnf \`
	`-days 365 -in peer1reqECdsa.pem -out peer1ECdsa.pem`
	If specifying the file name as the property value, this property value should be set to "file:" (no space after the colon), followed by the file name that appears after the "-out" parameter.
	Notes:
	• The above commands will generate an optional human-readable section above the mandatory section delimited by
	-----BEGIN CERTIFICATE-----
	-----END CERTIFICATE-----
	The human-readable section's "Subject" field may be copied to a **<subject_name>** element in the Permissions Document. But this section adds no functionality and is not protected in any way. It may be modified without changing the validity of the certificate. To generate a certificate without this section, add the command line parameter **-notext** to the "openssl ca" command.
	• **openssl((EC)dsa).cnf** must have the same **countryName**, **stateOrProvinceName**, and **localityName** as the example **.cnf** files.
	• For better security, it is recommended, but not required, that **.cnf** files of different participants have different commonNames
	• The document should be in PEM format. You may specify either the file name or the document contents. If specifying the file name, the property value must have the prefix "file:" (no space after the colon), followed by the fully qualified path and name of the file. If specifying the contents of the document, the property value must have the prefix "data:," (no space after the comma), followed by the contents inside the document. For example:
	"data:,-----BEGIN CERTIFICATE-----\nabcdef\n-----END CERTIFICATE-----"
	Note that the two "\n" characters are required.
	• You may put a chain of certificates in the Identity Certificate by concatenating individual certificates and specifying the concatenated result as a single file or string. See 4.1 Identity Certificate Chaining on page 18.
	Default: NULL

Table 4.2 DDS Security Properties for Configuring Authentication

Property Name	Property Value Description

dds.sec.auth.password	***Only required if private_key is encrypted***
	The password used to decrypt the private_key. The value of the password property is interpreted as the Base64 encoding of the symmetric key that will be used to decrypt the private_key. For example, if the private_key was encrypted using this command:
	`% openssl req -new -newkey ec:ecdsaparam2 -config example2ECdsa.cnf \`
	`-keyout peer2keyECdsa.pem -passout pass:MyPassword -out peer2reqECdsa.pem`
	you can obtain the Base64 encoding of MyPassword using these commands:
	`% echo MyPassword > foo`
	`% openssl base64 -e -in foo TXlQYXNzd29yZAo=`
	The value of the password property should be "TXlQYXNzd29yZAo=". If the private_key was not encrypted, then the password must be NULL.
	Default: NULL

Table 4.3 RTI Security Plugins Properties for Configuring Authentication

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
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**
authentication. alternative_ca_files	***Optional*** A comma-separated list of alternative Identity CA certificates. If the verification of a file fails with the main certificate (identity_ca/ca_file), verification will be retried with all of the corresponding alternative certificates. If none of the alternative certificates can be used to verify the file, the verification process will fail. If any of the alternative certificate files fail to be loaded, the DomainParticipant creation will fail. 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, followed by the '.' character.

Table 4.3 RTI Security Plugins Properties for Configuring Authentication

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

authentication.crl_file	***Optional***
	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.
	This property value may optionally contain "file:" (no space after the colon) as a prefix to the fully-qualified path and name of the file.
	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, followed by the '.' character.

Table 4.3 RTI Security Plugins Properties for Configuring Authentication

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
authentication. x509v3_extension_enforcement.key_usage	***Optional*** How to enforce the presence of the X.509 v3 extension keyUsage (see https://tools.ietf.org/html/rfc5280#section-4.2.1.3). This property has three possible values (case insensitive): • **auto**: The Security Plugins will not do anything special to enforce the presence of keyUsage in a certificate. Note this is the behavior in 6.0.0 and below. • **inherited**: The Security Plugins will enforce the presence of keyUsage if the certificate's parent has keyUsage. For example, if cacert.pem has keyUsage, cacert.pem signed peer1.pem, peer-1.pem does NOT have keyUsage, identity_ca is set to cacert.pem, and identity_certificate is set to peer1.pem, then participant creation will fail. The same applies if identity_certificate is set to the concatenation of peer1.pem and cacert.pem and cacert.pem is just an intermediate CA in a chained identity_certificate. • **force**: The Security Plugins will always enforce the presence of keyUsage. This is not the default, but it will force you to make sure that all of the certificates in the system have their extensions properly set. Default: **inherited**

4.1 Identity Certificate Chaining

In the **dds.sec.auth.identity_certificate** property (see Table 4.2 DDS Security Properties for Configuring Authentication on page 13), you may put a chain of certificates in the Identity Certificate by concatenating individual certificates and specifying the concatenated result as a single file or string. The Identity Certificate will be verified against the Identity CA using the following procedure; see Figure 4.1: Identity Certificate Chaining on the next page:

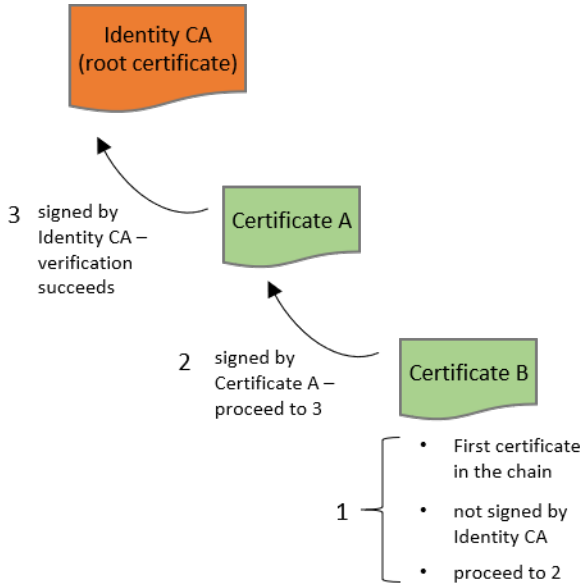
- The current certificate is the first certificate in the Identity Certificate chain.
- Perform the following steps up to and including the case when the current certificate is the last certificate in the Identity Certificate chain:
    - If the current certificate is signed by the Identity CA or any of the CAs in the list of **authentication.alternative_ca_files**, then the verification succeeds immediately.
    - Otherwise:
        - If a next certificate exists in the chain and the current certificate is signed by that

[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, followed by the '.' character.

next certificate, then the next certificate becomes the current certificate.

- Otherwise, verification fails immediately.

Figure 4.1: Identity Certificate Chaining



4.2 Related Governance Attributes

This section describes the Authentication attributes that appear in the Governance Document.

4.2.1 domain_rule

The attribute **allow_unauthenticated_participants** belongs inside a <domain_rule>.

This attribute may be TRUE or FALSE. It controls whether or not a remote DomainParticipant that either doesn't support security or fails the authentication handshake is allowed to proceed in trying to communicate with the local DomainParticipant. Communication with such a DomainParticipant will happen only on the topics for which all of the topic rules are set to FALSE or NONE.

If **allow_unauthenticated_participants** is set to TRUE, **rtps_protection_kind** must be set to NONE.

4.3 Fragmentation Support for the Authentication Topic

Security Plugins supports fragmenting Authentication (ParticipantStatelessMessage) built-in topic samples. This is useful in scenarios with a hard limit on the transport maximum message size.

This feature is enabled by default: fragmentation of Authentication built-in topic samples will be triggered when sending samples that exceed the **message_size_max** configured in the transports used by *Connext DDS*.

4.4 Configuration Properties Common to All Authentication Plugins

Table 4.4 Properties for Configuring Authentication Common to Any Authentication Plugin below lists a set of properties that are not exclusive to the shipped *Security Plugins*, but that will affect any Authentication Plugin.

Table 4.4 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. A *DomainParticipant* should set its own **authentication_timeout** to be shorter than the **participant_liveliness_lease_duration** (in the *DomainParticipant*'s DISCOVERY_CONFIG QosPolicy) of its peers. This restriction helps a temporarily disconnected peer to successfully reconnect with the *DomainParticipant*. Default: 60
authentication_request_delay.sec	***Optional*** Controls the delay in seconds before sending an authentication_request to the remote participant. For more information, please see 4.5 Re-Authentication below. Default: 5

4.5 Re-Authentication

The *Security Plugins* support securely re-authenticating remote Participants as described in 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 described in the DDS Security specification.

In *Security Plugins*, if Participant A has not completed an ongoing authentication with Participant B after a specific period, it will send a "*dds.sec.auth_request*" message (or "*com.rti.sec.auth.request*" message if the remote Participant is 5.3.x or older) 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

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

configured by the property **dds.participant.trust_plugins.authentication_request_delay.sec**, see Table 4.4 Properties for Configuring Authentication Common to Any Authentication Plugin on the previous page.

When Participant B receives a "*dds.sec.auth_request*" (or "*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 "*dds.sec.auth_request*" (or "*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.

4.6 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.

4.6.1 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.

Chapter 5 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 certificate 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 5.1 DDS Security Properties for Configuring Access Control on the next page and Table 5.2 RTI Security Plugins Properties for Configuring Access Control on page 25 to the DDS_DomainParticipantQos property.

Table 5.1 DDS Security Properties for Configuring Access Control

Property Name	Property Value Description
dds.sec.access.permissions_ca	***Required*** 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 the tables at the beginning of Chapter 4 Authentication on page 11 for the steps to generate this file. Two participants that want to securely communicate with each other must use the same Permissions Certificate Authority. The document should be in PEM format. You may specify either the file name or the document contents. If specifying the file name, the property value must have the prefix "file:" (no space after the colon), followed by the fully-qualified path and name of the file. If specifying the contents of the document, the property value must have the prefix "data:," (no space after the comma), followed by the contents inside the document. For example: `"data:,-----BEGIN CERTIFICATE-----\nabcdef\n-----END CERTIFICATE-----"` Note that the two "\n" characters are required. Default: NULL

dds.sec.access.governance	***Required***
	The signed document 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**.
	You may specify either the file name or the document contents. If specifying the file name, the property value must have the prefix "file:" (no space after the colon), followed by the fully-qualified path and name of the file. If specifying the contents of the document, the property value must have the prefix "data:," (no space after the comma), followed by the contents inside the document. For example:
	`"data:,MIME-Version: 1.0\nContent-Type:...boundary=\"---7236\"\n\n"`
	Note that for signed XML files, all whitespace characters (' ', '\r', '\n') are significant, and all quotes must be escaped by a backslash. The safest way to get the correct property value is to call the fread() function on the file and use the resulting buffer as the property value.
	Default: NULL

Table 5.1 DDS Security Properties for Configuring Access Control

Property Name	Property Value Description

dds.sec.access.permissions	***Required***
	The signed document 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.
	You may specify either the file name or the document contents. If specifying the file name, the property value must have the prefix "file:" (no space after the colon), followed by the fully-qualified path and name of the file. If specifying the contents of the document, the property value must have the prefix "data:," (no space after the comma), followed by the contents inside the document. For example:
	`"data:,MIME-Version: 1.0\nContent-Type:...boundary=\"---7236\"\n\n"`
	Note that for signed XML files, all whitespace characters (' ', '\r', '\n') are significant, and all quotes must be escaped by a backslash. The safest way to get the correct property value is to call the fread() function on the file and use the resulting buffer as the property value.
	Default: NULL

Table 5.2 RTI Security Plugins Properties for Configuring Access Control

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
access_control. alternative_permissions_authority_files	***Optional*** A comma-separated list of alternative Permissions CA certificates. If the verification of a file fails with the main certificate (permissions_ca/permissions_authority_file), verification will be retried with all of the corresponding alternative certificates. If none of the alternative certificates can be used to verify the file, the verification process will fail. If any of the alternative certificate files fail to be loaded, the DomainParticipant creation will fail. 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, followed by the '.' character.

Table 5.2 RTI Security Plugins Properties for Configuring Access Control

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
access_control. use_530_permissions_rules_precedence	***Optional*** How to deal with conflicting allow/deny rules in a Permissions Document. If TRUE, then the last rule will take precedence, which is consistent with Connext DDS 5.3.0 behavior. If FALSE, then the first rule will take precedence, which is consistent with the intended behavior of the DDS Security specification. Default: FALSE
access_control.use_530_logging_protection	***Optional*** How to set the value of <metadata_protection_kind> for the Builtin Logging Topic. If TRUE, then the value will be NONE, which is consistent with Connext 5.3.0 behavior. If FALSE, then the value will be SIGN, which is consistent with the behavior of the DDS Security specification. Default: FALSE
access_control.use_530_partitions	***Optional*** How to determine a match between a *DataWriter* or *DataReader's* partitions and an "allowed partitions" condition in a Permissions Document. If TRUE, then an entity is matched if it has at least one partition in the condition; this is consistent with Connext 5.3.0 behavior. If FALSE, then an entity is matched only if all of its partitions are in the condition; this is consistent with the behavior of the DDS Security specification. For example, if a *DataWriter* has partitions [A, B], and a Permissions Document allows partitions [B, C], then when use_530_partitions = TRUE, the *DataWriter* is allowed because B is allowed. When use_530_partitions = FALSE, the *DataWriter* is not allowed because A is not allowed. Default: FALSE

5.1 Specifying Domain IDs

Both the Governance Document and the Permissions Document require you to specify the applicable domain IDs using the <domains> tag. You may use this tag to specify individual domain IDs, domain ranges, open domain ranges, and combinations thereof.

Example: Individual domain IDs

```
<!-- Domains 0 and 1 -->
<domains>
    <id>0</id>
    <id>1</id>
</domains>
```

[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, followed by the '.' character.

Example: Domain Ranges

```
<!-- All domains between 3 and 10, inclusive -->
<domains>
    <id_range>
        <min>3</min>
        <max>10</min>
    </id_range>
</domains>
```

Example: Open Domain Ranges

```
<domains>
<!-- Domain 10 and above -->
    <id_range>
        <min>10</min>
    </id_range>
    <!-- Domains from 0 to 5, inclusive -->
    <id_range>
        <max>5</max>
    </id_range>
</domains>
```

5.2 Related Governance Attributes

This section describes the Access Control attributes that appear in the Governance Document.

5.2.1 domain_rule

The following attributes belong inside a <domain_rule>:

- **enable_join_access_control** may be TRUE or FALSE. It controls whether or not remote DomainParticipant permissions are checked when a remote DomainParticipant is discovered. Local DomainParticipant permissions are always checked using the local DomainParticipant's Permissions Document. There is no way to configure whether or not local DomainParticipant permissions are checked when a DomainParticipant is created.
- **topic_access_rules** contains one or more **topic_rule** attributes.

5.2.2 topic_rule

The following attributes belong inside a <topic_rule>:

- **enable_read_access_control**
- **enable_write_access_control**

These attributes may be TRUE or FALSE. They 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*.

5.2.3 No Matching Rule

If no matching domain or topic rule is found, the Entity creation will fail.

5.3 Permissions Document

The Permissions Document is an XML document containing the permissions of the DomainParticipant and binding them to the subject name of the DomainParticipant as defined in the Identity Certificate. The Permissions CA must sign the Permissions Document. This document contains a set of <grant> sections, each of which contains a <subject_name> section, a <validity> section, zero or more <allow_rule> sections, and zero or more <deny_rule> sections. This document is exchanged during authentication handshaking, so to conserve bandwidth, it is best to have this document contain exactly one <grant> section, which contains the subject name and rules for the DomainParticipant that is sending the document.

The following sections describe the elements within the <publish> and <subscribe> sections, which are inside <allow_rule> and <deny_rule> sections.

5.3.1 Topics

The <topics> element defines the DDS Topic names that must be matched for the rule to apply. Topic names may be given explicitly or by means of Topic name expressions. Each topic name or topic-name expression appears separately in a <topic> sub-element within the <topics> element. The Topic name expression syntax and matching shall use the syntax and rules of the POSIX fnmatch() function as specified in POSIX 1003.2-1992, Section B.6.

Example (appearing within an <allow_rule> and within a publish or subscribe action):

```
<topics>
    <topic>Square</topic>
    <topic>B*</topic>
</topics>
```

The above topic condition would match Topic “Square” and any topic that starts with a “B”.

5.3.2 Partitions

The *RTI Connext DDS Core Libraries User's Manual* describes the PartitionQosPolicy as a sequence of strings that belong to a Publisher or Subscriber. The Security Plugins Access Control plugin uses these partitions to determine whether or not a DataWriter or DataReader is allowed to exist according to the Permissions Document. Inside the Permissions Document, the <partitions> element may appear within a <publish> or <subscribe> element. <partitions> may contain one or more <partition> elements, each containing a string. For example:

```
<subscribe>
    <topics>
        <topic>Square</topic>
    </topics>
    <partitions>
        <partition>aPartition1</partition>
        <partition>aPartition2</partition>
        <partition>bPartition*</partition>
    </partitions>
</subscribe>
```

Note the asterisk in the third partition. POSIX fnmatch() matching is allowed for the <partition> element.

5.3.2.1 Allowed

If the <partitions> are under an <allow_rule>, then the <partitions> delimit an allowed partitions condition section. In order for an action (e.g., a publish action) to meet the allowed partitions condition, the set of the partitions associated with the DDS Entity performing the action (e.g., a *DataWriter* for a publish action) must be contained in the set of partitions defined by the allowed partitions condition section. If there is no <partitions> section, then the default "empty string" partition is assumed. This means that the allow action (e.g., publish action) would only allow publishing on the "empty string" partition.

Example (appearing within a <allow_rule> and within a <publish> action):

```
<partitions>
    <partition>A</partition>
    <partition>B</partition>
</partitions>
```

The above allowed partitions condition would be matched if the partitions associated with the DDS Entity performing the action (e.g., *DataWriter* for publish action) are a subset of the set [A, B] . So it would be OK to publish in partition A, in B, or in [A, B] but not in [A, B, C] (assuming the value of the property access_control.use_530_partitions is FALSE) or in the "empty string" partition.

5.3.2.2 Denied

If the <partitions> are under a <deny_rule>, then the <partitions> delimit a denied partitions condition section. For this condition to be met, the DDS Entity associated with the action (e.g., *DataWriter* for a publish action) must have a partition that matches one of the partitions explicitly listed in the denied partitions condition section. If there is no <partitions> section, then the "*" partition expression is assumed. This means that the deny action (e.g., deny publish action) would apply regardless of the partitions associated with the DDS Endpoint (e.g., *DataWriter* for a publish action).

Example (appearing within a <deny_rule> and within a <publish> action):

```
<partitions>
    <partition>A</partition>
    <partition>B</partition>
</partitions>
```

The above denied partitions condition would be matched if the partitions associated with the DDS Entity performing the action (e.g., *DataWriter* for a publish action) intersect the set [A, B]. So, it would be OK to publish in partition C or in the "empty string" partition, but not in partition A, in [A,B], or in [A, B, C].

5.3.2.3 Partitions Mutability

Security Plugins does not allow a Publisher to change the PartitionQosPolicy after the Publisher has been enabled if the Publisher contains any *DataWriter* that meets the following two criteria:

- The TopicSecurityAttributes for that *DataWriter* have **is_read_protected** (which corresponds to **enable_read_access_control** in the Governance Document) set to TRUE.
- The *DataWriter* has the DurabilityQos policy **kind** set to something other than VOLATILE.

When these two criteria are met, a *DataWriter* should send historical data only to *DataReaders* that were passing the topic access control rules at the time the historical data was generated. The rule about PartitionQos immutability enforces this behavior by conservatively preventing a *DataWriter* of a protected topic from sending historical data to *DataReaders* that were not matched before a PartitionQos change and that potentially could have failed to pass the topic access control rules.

5.3.3 Data Tags

The *RTI Connext DDS Core Libraries User's Manual* describes the DataTagQosPolicy as a sequence of (name, value) string pairs that belong to a *DataWriter* or *DataReader*. The *Security Plugins* Access Control plugin uses these tags to determine whether or not a *DataWriter* or *DataReader* is allowed to exist according to the Permissions Document. Inside the Permissions Document, the <data_tags> element may appear within a <publish> or <subscribe> element. <data_tags> may contain one or more <tag> elements, each containing a <name> and a <value> element. For example:

```
<subscribe>
    <topics>
        <topic>Sq*</topic>
    </topics>
    <data_tags>
        <tag>
            <name>Department</name>
            <value>Engineering</value>
        </tag>
        <tag>
            <name>Seniority</name>
            <value>Senior</value>
        </tag>
        <tag>
            <name>Title</name>
            <value>*Software*</value>
        </tag>
    </data_tags>
</subscribe>
```

Note the asterisk in the third tag's value. POSIX fnmatch() matching is allowed for the <value> element, but not for the <name> element.

5.3.3.1 Allowed

If the <data_tags> are under an <allow_rule>, then the <data_tags> delimit an allowed data tags condition section. In order for an action (e.g., a publish action) to meet the allowed data tags condition, the set of the data tags associated with the DDS Entity performing the action (e.g., a *DataWriter* for a publish action) must be contained in the set of data tags defined by the allowed data tags condition section. If there is no <data_tags> section, then the default empty set is assumed. This means that the allow action (e.g., publish action) would only allow publishing if there are no data tags associated with the DDS Endpoint (*DataWriter* for a publish action).

Example (appearing within a <allow_rule> and within a <publish> action):

```
<data_tags>
    <tag>
        <name>aTagName1</name>
        <value>aTagValue1</value>
    </tag>
</data_tags>
```

The above allowed data tags condition would be matched if the data tags associated with the DDS Entity performing the action (e.g., *DataWriter* for publish action) are a subset of the set [(aTagName1, aTagValue)]. So it would be OK to publish using a *DataWriter* with no associated data tags, or a *DataWriter* with a single tag with name "aTagName1" and value "aTagValue1".

5.3.3.2 Denied

If the <data_tags> are under a <deny_rule>, then the <data_tags> delimit a denied data tags condition section. For this condition to be met, the DDS Entity associated with the action (e.g., *DataWriter* for a publish action) must have a data tag name and value pair that matches one of the data tags explicitly listed in the denied data tags condition section. If there is no <data_tags> section, then the "set of all possible tags" set is assumed as default. This means that the deny action (e.g., deny publish action) would apply regardless of the data tags associated with the DDS Endpoint (e.g., *DataWriter* for a publish action).

Example (appearing within a <deny_rule> and within a <publish> action):

```
<data_tags>
    <tag>
        <name>aTagName1</name>
        <value>aTagValue1</value>
    </tag>
</data_tags>
```

The above denied data tags condition would be matched if the data tags associated with the DDS Entity performing the action (e.g., *DataWriter* for a publish action) intersect the set [(aTagName1, aTagValue1)]. So it would not deny publishing using a *DataWriter* with no associated data-tags, or a *DataWriter* with a

single tag with name "aTagName2", or a *DataWriter* with a single tag with name "aTagName1" and value "aTagValue2". But it would deny publishing using a *DataWriter* with two associated data tags [(aTagName1, aTagValue1), (aTagName2, aTagValue2)].

Chapter 6 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 **rtps_protection_kind** to ENCRYPT.

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

Table 6.1 RTI Security Plugins Properties 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
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. 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**

[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, followed by the '.' character.

Table 6.1 RTI Security Plugins Properties for Configuring Cryptography

Property Name (prefix with 'com.rti.serv.secure.') [1]	Property Value Description
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 with origin authentication, 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], excluding 1
cryptography.share_key_for_metadata_and_data_protection	***Optional*** Indicator of whether the metadata and data encoding operations share the same key material or use different keys. By default, *DataWriters* with both **metadata_protection_kind** and **data_protection_kind** set to a value other than NONE use the same key material for encoding both submessages and serialized data. To change this behavior, set this property to FALSE. Default: TRUE (they share key material)

6.1 Related Governance Attributes

This section describes the Cryptography attributes that appear in the Governance Document.

6.1.1 ProtectionKind

Attributes whose names end with **_protection_kind** share a type called ProtectionKind. The DDS Security specification lists five possible values of ProtectionKind, all of which are supported by *Security Plugins*.

- **NONE** indicates that no cryptographic transformation is applied.
- **SIGN** indicates that the cryptographic transformation is purely a Galois message authentication code (GMAC). No encryption is performed. The GMAC is placed after the content. If the receiver finds a missing or incorrect GMAC, the receiver will reject the content.

[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, followed by the '.' character.

- **ENCRYPT** indicates that the cryptographic transformation is an AES encryption followed by a GMAC computed on the ciphertext, also known as Galois/Counter Mode (GCM). The GMAC is placed after the content. If the receiver finds a missing or incorrect GMAC, the receiver will reject the content.
- **WITH_ORIGIN_AUTHENTICATION protection kinds**. There are two protection kinds that have WITH_ORIGIN_AUTHENTICATION in their names. WITH_ORIGIN_ AUTHENTICATION indicates that in addition to using the sender's key to generate a common GMAC, the sender generates receiver-specific GMACs using keys that are specific to individual sender-receiver pairs. The additional GMACs are placed after the common GMAC. They prove to the receiver that the sender originated the message, preventing other receivers from impersonating the sender. If the receiver finds a missing or incorrect common GMAC, the receiver will reject the content. If the receiver finds a missing or incorrect receiver-specific GMAC that was computed using its own receiver-specific key, the receiver will reject the content. WITH_ORIGIN_ AUTHENTICATION protection kinds are allowed only if the value of the property **cryptography.max_receiver_specific_macs** is greater than 1.

  The WITH_ORIGIN_AUTHENTICATION protection kinds are as follows:
  - **SIGN_WITH_ORIGIN_AUTHENTICATION** indicates that a common GMAC is performed on the content, and receiver-specific GMACs are performed on the common GMAC.
  - **ENCRYPT_WITH_ORIGIN_AUTHENTICATION** indicates that a GCM is performed on the content, and receiver-specific GMACs are performed on the GMAC of the GCM.

6.1.2 domain_rule

The following attributes belong inside a <domain_rule>.

- **rtps_protection_kind**. This ProtectionKind specifies how to protect a DomainParticipant's outgoing messages and what kind of protection is required of incoming messages. A message consists of an RTPS header and submessages, so a message is an envelope around submessages. If **allow_ unauthenticated_participants** is set to TRUE, **rtps_protection_kind** must be set to NONE. Setting **rtps_protection_kind** to NONE will cause the *DomainParticipant* to accept both protected and unprotected incoming RTPS messages. Setting **rtps_protection_kind** to something other than NONE will cause the *DomainParticipant* to reject incoming RTPS messages that have a missing or incorrect GMAC or GCM.
- **discovery_protection_kind**. This ProtectionKind specifies the **metadata_protection_kind** used for the secure builtin *DataWriter* and *DataReader* entities used for discovery, Topic Queries, and Locator Reachability Responses.
- **liveliness_protection_kind**. This ProtectionKind specifies the **metadata_protection_kind** used for the secure builtin *DataWriter* and *DataReader* entities used for liveliness.

6.1.3 topic_rule

The following attributes belong inside a <topic_rule>.

- **metadata_protection_kind**. This ProtectionKind specifies how to protect a *DataWriter*'s or *DataReader*'s outgoing submessages. These submessages include, but are not limited to, DATA, HEARTBEAT, ACKNACK, and GAP. A DATA submessage is an envelope around a serialized payload, so **metadata_protection_kind** affects data as well as metadata. One difference between **metadata_protection_kind** and **data_protection_kind** is that for **metadata_protection_kind**, the submessage protection takes effect immediately before sending out the content, so a protected submessage is re-protected when it is resent.
- **data_protection_kind**. This attribute may be NONE, SIGN, or ENCRYPT. It specifies how to protect a *DataWriter*'s serialized payload. The writer history stores the protected payload, so the protected payload is not re-protected when it is resent. Receiver-specific GMACs are never included in this protection, so the WITH_ORIGIN_AUTHENTICATION values are not allowed here.
- **enable_discovery_protection**. This attribute may be TRUE or FALSE. It specifies whether to use the secure or non-secure builtin endpoints for certain outgoing traffic related to this topic. Such traffic includes endpoint discovery messages and TopicQuery messages. **enable_discovery_protection** also specifies whether or not to reject non-secure incoming endpoint discovery messages related to this topic.
- **enable_liveliness_protection**. This attribute may be TRUE or FALSE. The value of this attribute matters only if the DataWriter LivelinessQosPolicy is AUTOMATIC_LIVELINESS_QOS or MANUAL_BY_PARTICIPANT_LIVELINESS_QOS. In either of these cases, **enable_liveliness_protection** specifies whether or not to use the secure builtin endpoints for exchanging liveliness messages for *DataWriters* of this topic.

6.2 Configuration Properties Common to All Cryptography Plugins

Table 6.2 Properties for Configuring Cryptography Common to Any Cryptography Plugin below lists a set of properties that are not exclusive to the shipped Security Plugins, but that will affect any Cryptography Plugin.

Table 6.2 Properties for Configuring Cryptography Common to Any Cryptography Plugin

Property Name	Property Value Description
dds.data_writer-.history.use_530_encoding_alignment	***Optional*** Determines whether or not to align a serialized payload to a 4-byte boundary before encoding it. If TRUE, then this alignment does not occur; this is consistent with Connext 5.3.0 behavior. If FALSE, then this alignment does occur; this is the only way to make the builtin Cryptography plugin work with **data_protection_kind** = SIGN. This property applies to the DataWriterQos. Default: FALSE

6.3 Reliability Behavior When MAC Verification Fails

When setting **data_protection_kind**, **metadata_protection_kind**, or **rtps_protection_kind** to a value other than NONE, the *DataReader* may reject a sample due to MAC verification (for example, if the sample is tampered or replayed). When this happens, the *DataReader* does not deliver the sample to the application, and the sample is lost. If the ReliabilityQosPolicy is configured with DDS_RELIABLE_RELIABILITY_QOS, however, the *DataWriter* can still repair the lost sample.

Note that depending on the level of protection, a tampered/replayed sample may be rejected at different levels:

- If **metadata_protection_kind** or **rtps_protection_kind** is a value other than NONE, the sample will be rejected before reaching the *DataReader* queue.
- If metadata and rtps protection checks passed, and **data_protection_kind** is set to a value other than NONE, the sample will be rejected by the *DataReader* queue.

6.4 Enabling Asynchronous Publishing for the Key Exchange Topic

Security Plugins supports fragmenting Key Exchange (ParticipantSecureVolatileMessageSecure) built-in topic samples. This is useful in scenarios with a hard limit on the transport maximum message size. Key Exchange is a reliable topic; therefore, enabling fragmentation requires changing the publish mode to asynchronous publishing. For more information about how to configure the Key Exchange topic publish mode, see Table 6.3 DDS_DiscoveryConfigQosPolicy fields Affecting Key Exchange Topic below.

Table 6.3 DDS_DiscoveryConfigQosPolicy fields Affecting Key Exchange Topic

Type	Field Name	Description
PUBLISH_MODE QosPolicy (DDSExtension) (Section 6.5.18 of *RTI Connext DDS Core Libraries User's Manual*)	secure_volatile_writer_publish_mode	Determines whether the Key Exchange built-in subscription *DataWriter* publishes data synchronously or asynchronously, and how.

Chapter 7 Logging

Security Plugins uses its own Logging Plugin to notify you of security events. This Logging Plugin supports the following logging methods:

- Using *Connext DDS*'s own builtin logging system to send security messages
- Printing security log messages to a file
- Distributing security log messages over DDS

By default, log messages are processed by the *Connext DDS* builtin logging system. You can instead print log messages to a file by setting **logging.log_file** and/or distribute log messages over DDS by setting **logging.distribute.enable properties**. You can also adjust the verbosity level of the log messages with **logging.verbosity**. See Table 7.1 RTI Security Plugins Properties for Configuring Logging.

If you are distributing log messages over DDS, the messages' log levels are used as the severity values of the BuiltinLoggingType. If you are using the *Connext DDS* builtin logging system (which is the default logging method), the messages' log levels are mapped to the values shown in Table 7.3 Mapping between Logging Plugin and Connext DDS Builtin Logging System.

See Controlling Messages from Connext DDS, in the RTI Connext DDS Core Libraries User's Manual for more information on the *Connext DDS* builtin logging system.

7.1 *Connext DDS* Builtin Logging System

When security logging is configured to be written to *Connext DDS*'s own builtin logging system (also known as NDDS_Config_Logger), you will see messages that look something like the following:

```
[1541757196.822050]RTI_Security_Authentication_validate_local_identity:successfully
validated local identity
[1541757196.822290]RTI_Security_AccessControl_get_permissions_credential_
token:successfully got permissions token
[1541757196.825604]RTI_Security_AccessControl_validate_local_permissions:successfully
```

```
validated local permissions
[1541757196.828948]RTI_Security_AccessControl_get_participant_sec_attributes:successfully got
participant security attributes
```

The messages may be preceded by additional information, such as timestamps or GUIDs, depending on the *Connext DDS* builtin logging system format configuration and the message itself. (See Format of Logged Messages, in the RTI Connext DDS Core Libraries User's Manual.)

To write a log to the *Connext DDS* builtin logging system, none of the following should be enabled; if they are, the log won't be shown in the *Connext DDS* builtin logging system:

- The output file property (**com.rti.serv.secure.logging.log_file**)
- The distributed log property (**com.rti.serv.secure.logging.distribute.enable**)

If the above options *aren't* enabled, the log is written to the *Connext DDS* builtin logging system, which is the default option.

7.2 Log File

When security logging is configured to be written to an output log file, you will see messages that look something like the following:

```
[1541757196.822050]RTI_Security_Authentication_validate_local_identity:successfully validated
local identity
[1541757196.822290]RTI_Security_AccessControl_get_permissions_credential_token:successfully got
permissions token
[1541757196.825604]RTI_Security_AccessControl_validate_local_permissions:successfully validated
local permissions
[1541757196.828948]RTI_Security_AccessControl_get_participant_sec_attributes:successfully got
participant security attributes
```

The messages may be preceded by additional information, such as timestamps or GUIDs, depending on the message.

To write the log to an output file, set the following property within the **<participant_qos>** (see Configuring QoS with XML, in the RTI Connext DDS Core Libraries User's Manual):

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>com.rti.serv.secure.logging.log_file</name>
                <value>log.txt</value>
            </element>
        </value>
    </property>
</participant_qos>
```

7.3 Distributed over DDS

When the messages are distributed over DDS, you get more information than if you use the *Connext DDS* builtin logging system or a log file. Furthermore, the information follows the DDS Security specification. For example, here is how the messages look in *RTI DDS Spy* using the **-printSample** option if you use logging distribution over DDS:

```
1543498637.929123  d +M  DCB9C740    DDS:Security:LogTopic
DDSSecurity::BuiltinLoggingType
facility: 10
severity: DEBUG_LEVEL
timestamp:
   sec: 1543498637
   nanosec: 928903998
hostname: "localhost"
hostip: "0.0.0.0"
appname: "RTI Secure DDS Application"
procid: "9654"
msgid: "security"
message: "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."
structured_data:
   [0]:
      key: "DDS"
      pairs:
         [0]:
            name: "guid"
            value: "dcb9c740.7ecf85eb.42aa8349.1c1"
         [1]:
            name: "domain_id"
            value: "25"
         [2]:
            name: "plugin_method"
            value: "RTI_Security_Cryptography_preprocess_secure_submsg"
         [3]:
            name: "plugin_class"
            value: "Cryptography"
```

You can find the BuiltinLoggingType in **<NDDSHOME>/resource/idl/builtin_logging_ type.idl**. (See Chapter 2 Paths Mentioned in Documentation on page 4.)

IMPLEMENTATION NOTES:

- The **DDS:Security:LogTopic** (described in 7.3.2 Using a Custom Subscriber on the next page) is published by the same *DomainParticipant* that uses *Security Plugins* to communicate securely. Therefore, this topic is published in a secured domain, and the Governance and Permissions files that apply to that *DomainParticipant* also apply to this log topic. For this reason, the Permissions file must allow the **DDS:Security:LogTopic** to be published.
- *Security Plugins* publishes log messages using the QoS profile specified by the property **com.rti.serv.secure.logging.distribute.profile**. If this property is not set, *Security Plugins* will use the same QoS profile as the *DomainParticipant* that is loading *Security Plugins*.
- If you specify a custom profile in the property **com.rti.serv.secure.logging.distribute.profile**, only *Publisher*, *Topic*, and *DataWriter* QoS will be used from the specified profile. This is because the *DataWriter* that distributes the log belongs to the same *DomainParticipant* as the one instantiating *Security Plugins*. As a result, you can use the custom profile for the logger if, for example, you want to change the logger's reliability, but you cannot use it to change the logger's identity (private key and certificate).

7.3.1 Setting the Properties

To distribute the log messages over DDS, set the following property within the **<participant_qos>** (see Configuring QoS with XML, in the RTI Connext DDS Core Libraries User's Manual):

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>com.rti.serv.secure.logging.distribute.enable</name>
                <value>true</value>
            </element>
        </value>
    </property>
</participant_qos>
```

The granularity of the log messages depends on the verbosity level set for the security log. You can set the verbosity level as follows, DEBUG being the highest verbosity level (see 7.4 Logging Properties and Messages on page 43):

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>com.rti.serv.secure.logging.verbosity</name>
                <value>DEBUG</value>
            </element>
        </value>
    </property>
</participant_qos>
```

7.3.2 Using a Custom Subscriber

To get the security logging information that is being distributed over DDS, you will need to create an application that subscribes to the secure log topic:

- Topic name: **DDS:Security:LogTopic**
- Type name: **DDSSecurity::BuiltinLoggingType**

To create a custom subscriber, you can start from the IDL containing the definition of the **DDSSecurity::BuiltinLoggingType**, which can be found here: **<NDDSHOME>/resource/idl/builtin_logging_type.idl**. (See Chapter 2 Paths Mentioned in Documentation on page 4.)

With the IDL, you can run *rtiddsgen* to generate type support code and example code. For example, the following command will generate all the files you need to access the **DDS:Security:LogTopic** topic from your *Connext DDS* application:

```
rtiddsgen -language C -example x64Linux3gcc4.8.2 -unboundedSupport builtin_logging_type.idl
```

IMPORTANT: You need to use the command-line option **-unboundedSupport** because **DDSSecurity::BuiltinLoggingType** contains strings without a specified limit. By default, *rtiddsgen* will limit unbounded strings to 255 characters, unless you specify the **-unboundedSupport** option. Without this option, the type support code will not be compatible with the type used by *Security Plugins*, and you won't be able to receive any messages. For more information about the use of unbounded support with builtin types, see Managing Memory for Builtin Types, in the RTI Connext DDS Core Libraries User's Manual.

Since the log messages are distributed in a secured domain, your custom subscriber needs an identity (private key and certificate). In addition, it needs to have permissions to subscribe to **DDS:Security:LogTopic**; this permission needs to be included in the Permissions file.

7.4 Logging Properties and Messages

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

Table 7.1 RTI Security Plugins Properties for Configuring Logging

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
applies to log file method only	
logging.log_file	***Optional*** The file that log messages are printed to. If you specify a file in this property, *Security Plugins* will print to this file instead of to the *Connext DDS* builtin logging system or via DDS distribution. Default: NULL
applies to all logging methods	
logging.log_level (deprecated)	***Optional*** The logging verbosity level. This setting applies to all methods of security logging (the *Connext DDS* builtin logging system, a log file, and DDS distribution). All log messages at and below the **logging.log_level** setting will be logged. This property can take any of the following values: • 0: emergency • 1: alert • 2: critical • 3 (default): error • 4: warning • 5: notice • 6: informational • 7: debug Notes: • This property is deprecated and may stop working in a future release. Use **logging.verbosity** instead. • This property is incompatible with **logging.verbosity** (i.e. they cannot be used simultaneously).

[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, followed by the '.' character.

Table 7.1 RTI Security Plugins Properties for Configuring Logging

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
logging.verbosity	***Optional*** The logging verbosity level. This setting applies to all methods of security logging (the *Connext DDS* builtin logging system, a log file, and DDS distribution). All log messages at and below the verbosity level will be logged. This property can take any of the following values (case sensitive): • SILENT (least verbose) • EMERGENCY • ALERT • CRITICAL • ERROR (default) • WARNING • NOTICE • INFORMATIONAL • DEBUG (most verbose) This property is incompatible with **logging.log_level** (i.e., they cannot be used simultaneously).
applies to logging distribution over DDS method only	
logging.distribute.enable	***Optional*** Controls whether security-related log messages should be distributed over a DDS *DataWriter*. If enable is true, then the Logging Plugin will create a *Publisher* and *DataWriter* within the same *DomainParticipant* that is setting this property. There is no option to use a separate *DomainParticipant* or to share a *DataWriter* among multiple *DomainParticipants*. 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. To subscribe to the log messages, run **rtiddsgen** on **resource/idl/builtin_logging_type.idl**. For example: `rtiddsgen -language C -example x64Linux3gcc4.8.2 -unboundedSupport builtin_logging_type.idl` **IMPORTANT:** You must use the command-line option **-unboundedSupport**. See 7.3.2 Using a Custom Subscriber on page 42 for more information. Boolean. Default: FALSE

[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, followed by the '.' character.

Table 7.1 RTI Security Plugins Properties for Configuring Logging

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

[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, followed by the '.' character.

Table 7.1 RTI Security Plugins Properties for Configuring Logging

Property Name (prefix with 'com.rti.serv.secure.')[1]	Property Value Description
log-ging.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
log-ging.dis-tribute.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

Table 7.2 Log Messages lists security-related events and the log messages they generate.

Table 7.2 Log Messages

Event	Log Level	Message
Failed to allocate memory	EMERGENCY	insufficient memory
allow_unauthenticated_participants = FALSE, and discovered remote participant that is unauthenticable, i.e. has not enabled security	WARNING	unauthenticated remote participant [participant ID] denied
allow_unauthenticated_participants = 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	ERROR	failed to decode certificate
Couldn't verify certificate's signature against neither the certificate of the Identity Certificate Authority nor any alternative CAs	ERROR	failed to verify certificate
Certificate appears in Certificate Revocation List	ERROR[2]	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.	ERROR	failed to verify challenge signature

[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, followed by the '.' character.

[2]This log message is logged using *Connext DDS*'s own builtin logging system, regardless of the logging method used.

Table 7.2 Log Messages

Event	Log Level	Message
Another participant has lost liveliness with this one before authentication was completed and is trying to re-authenticate itself (see 4.5 Re-Authentication on page 20). A new authentication should restore communication once the ongoing authentication times out. Or: Network traffic corruption during authentication. Or: Misbehaving remote participant during authentication.	NOTICE	RTI_Security_Authentication_process_handshake:received unexpected handshake message, probably from a participant that lost liveliness with this one before ongoing authentication completed. Once current authentication times out, communication should be restored.
RTI_Security_Authentication_process_handshake failed (see previous row).	STATUS_REMOTE[1]	DDS_DomainParticipantTrustPlugins_forwardProcessHandshake:!security function process_handshake returned VALIDATION_FAILED
Couldn't verify permissions or governance file signature against either the certificate of the Permissions Authority or any alternative permissions authorities	ERROR	Document signature verification failed. Make sure document was signed by the right permissions authority.
Received signed permissions document that is not an XML document	ALERT	received invalid signed permissions document
Received signed governance document that is not an XML document	ERROR	received invalid signed 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	ERROR	participant not allowed: deny rule found
Denied participant because there is no rule for the participant's domain ID, and the default is to deny	ERROR	participant not allowed: no rule found for the participant's domainId; default DENY
Denied writer or reader because there is a deny rule explicitly prohibiting the writer or reader	ERROR	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	ERROR	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	DEBUG	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	DEBUG	This publish/subscribe rule doesn't apply because none of the rule's partition expressions match with any of the endpoint's partitions

[1]This log message is logged using *Connext DDS*'s own builtin logging system, regardless of the logging method used.

Table 7.2 Log Messages

Event	Log Level	Message
Another participant, which is not using is_rtps_protected = true or is not using security at all, has sent this one an unprotected RTPS message that is not a participant announcement, handshake message, or key exchange message. This participant is using is_rtps_protected = true, so it drops the message. Or: Network traffic corruption after key exchange. Or: Misbehaving remote participant after key exchange.	STATUS_ REMOTE[1]	MIGInterpreter_parse:received unencoded rtps message. Unacceptable due to is_rtps_protected = true
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.
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	STATUS_ LOCAL[2]	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].
Parsed publish/subscribe <allow_rule> in permissions file that does not apply to the writer/reader because even though there's a matching topicexpression, the partition expressions in the QoS are not a subset of the ones in the <allow_rule>	DEBUG	This publish/subscribe rule doesn't apply because endpoint's partitions are not a subset of the rule's partition expressions

Log messages generated by the Logging Plugin in *Security Plugins* and those generated by the *Connext DDS* builtin logging system (which is the default method) have a different structure. For example, the Logging Plugin includes information about the process ID (procid) and the host (hostname, hostip). This information is lost when using the *Connext DDS* builtin logging system. The log level values used by these logging systems are also different. Table 7.3 Mapping between Logging Plugin and Connext DDS Builtin Logging System shows the mapping between Logging Plugin log values and the *Connext DDS* builtin logging system log values. For example, messages marked as "emergency," "alert," "critical," or "error" in *Security Plugins* are translated to "error" when using the builtin logging system.

[1]This log message is logged using *Connext DDS*'s own builtin logging system, regardless of the logging method used.

[2]This log message is logged using *Connext DDS*'s own builtin logging system, regardless of the logging method used.

Table 7.3 Mapping between Logging Plugin and *Connext DDS* Builtin Logging System

Logging Plugin (Security Plugins) log level values[1]	*Connext DDS* builtin logging system log level values[2]
DDS_LOGGING_EMERGENCY_LEVEL DDS_LOGGING_ALERT_LEVEL DDS_LOGGING_CRITICAL_LEVEL DDS_LOGGING_ERROR_LEVEL	NDDS_CONFIG_LOG_LEVEL_ERROR
DDS_LOGGING_WARNING_LEVEL	NDDS_CONFIG_LOG_LEVEL_WARNING
DDS_LOGGING_NOTICE_LEVEL	NDDS_CONFIG_LOG_LEVEL_STATUS_LOCAL
DDS_LOGGING_INFORMATIONAL_LEVEL	NDDS_CONFIG_LOG_LEVEL_STATUS_REMOTE
DDS_LOGGING_INFORMATIONAL_DEBUG	NDDS_CONFIG_LOG_LEVEL_DEBUG

[1]These values correspond to the values listed in **logging.verbosity**.

[2]These values are described in Controlling Messages from Connext DDS, in the RTI Connext DDS Core Libraries User's Manual.

Chapter 8 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 8.1 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, followed by the '.' character.

Table 8.1 Properties for Configuring OpenSSL Engines

Property Name (prefix with 'com.rti.serv.secure.') [1]	Property Value Description
authentication.keyform	***Optional*** The format of the private key specified by dds.sec.auth.private_key. The value of this property can be one of the following: • pem: The key is in PEM format and will be loaded as a file or a string, depending on its "file:" or "data:," prefix. • engine: The key is an array of bytes and will be loaded by the engine specified by openssl_engine. Default: pem

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-$(CERTICOMOS)/lib/:$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 arm-v7aQNX6.6.0qcc_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.*

8.1 Support for Engine Control Commands

You may use the DDS_DomainParticipantQos property to invoke a sequence of **ENGINE_ctrl_cmd_string()** function calls (see also https://www.openssl.org/docs/man1.1.0/man3/ENGINE_ctrl_cmd_string.html, "Issuing control commands to an ENGINE"). The Qos Profile XML format of the properties is the following, assuming that the prefix is **com.rti.serv.secure**:

```
<participant_qos>
    <property>
        <value>
            <element>
                <name>com.rti.serv.secure.openssl_engine</name>
```

[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, followed by the '.' character.

```
                <value>[engineName]</value>
            </element>
            <element>
                <name>com.rti.serv.secure.openssl_engine.[engineName].[cmdName1]</name>
                <value>[arg1]</value>
            </element>
            <element>
                <name>com.rti.serv.secure.openssl_engine.[engineName].[cmdName2]</name>
               <value>[arg2]</value>
            </element>
            ...
        </value>
    </property>
</participant_qos>
```

The above properties will result in the following code being called:

```
ENGINE_ctrl_cmd_string(engineIdentifiedBy[engineName], [cmdName1], [arg1], 0);
ENGINE_ctrl_cmd_string(engineIdentifiedBy[engineName], [cmdName2], [arg2], 0);
```

Chapter 9 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 the output of a derivation function whose input is an optional user-specified property, and the Cryptography Plugin implementation determines both the key derivation function and the encryption algorithm.

In RTI's default plugin implementation, the key derivation function involves PBKDF2 (Password-Based Key Derivation Function) with SHA-512 (Secure Hash Algorithm with a 512-bit hash value) and a random salt, and the encryption algorithm involves AES-256-GCM. The key derivation function derives both the key and the IV (Initialization Vector) used in the encryption. *Persistence Service* stores the random salt along with the PRSTDataWriter's encrypted key.

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* use of security:

Table 9.1 Properties for Configuring Secure Persistence Service

Property Name	Property Value Description
dds.data_writer. history.key_material_key	***Required*** 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 a non-existent or incorrect **key_material_key** will result in an informative log message and failure to create *Persistence Service*. You may specify either the file name or the document contents: • If specifying the file name, the property value may optionally have the prefix "**file:**" (no space after the colon), followed by the fully qualified path and name of the file. • If specifying the contents of the document, the property value must have the prefix "**data:,**" (no space after the comma), followed by the contents inside the document. For example: "**data:,myPassword**". The length of the key material contents may not exceed 2,147,483,647. Default: NULL

Chapter 10 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 10.1 Properties for Configuring HMAC-Only Mode below.

Table 10.1 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

[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, followed by the '.' character.

Table 10.1 Properties for Configuring HMAC-Only Mode

Property Name (prefix with 'com.rti.serv.secure.') [1]	Property Value Description
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). To compute the actual key that the RTPS-HMAC-Only plugins use, the plugins compute a SHA256 hash over the contents of a buffer containing the user-provided HMAC key, plus a random session-id, plus some non-disclosed strings. Consequently, passing a user-provided HMAC key longer than 32 bytes does not provide any benefit with respect to the security of the key. As such, we recommend using a full entropy 32-byte HMAC key for maximum security. 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, followed by the '.' character.

Chapter 11 What's Different from the OMG Security Specification

This section describes differences between *Security Plugins* 6.0.1 and the OMG DDS Security specification (Version 1.1).

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

11.1.1 Acess Control

11.1.1.1 Mutability of Publisher PartitionQosPolicy

Section 7.3.5 in the specification defines the kind of *DataWriters* that a Publisher must contain in order for its PartitionQosPolicy to be immutable. These *DataWriters* meet the following two criteria:

1. The *DataWriter* either encrypts the SerializedPayload submessage element or encrypts the Data or DataFrag submessage elements.
2. The *DataWriter* has the DurabilityQosPolicy kind set to something other than VOLATILE.

The next version of the specification will change the first criterion to be the following:

1. The TopicSecurityAttributes for that *DataWriter* have **is_read_protected** set to TRUE.

The second criterion still applies. *Security Plugins* uses this new criteria to determine PartitionQosPolicy immutability. (**is_read_protected** corresponds to <enable_read_access_control> in the Governance Document.)

11.2 Differences Affecting Builtin Plugins

11.2.1 General

11.2.1.1 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**. While *Security Plugins* functions include this additional parameter, *Security Plugins* does not implement this mechanism, and the parameter is currently not used or populated.

11.3 Differences Affecting Custom Plugins

11.3.1 Authentication

11.3.1.1 Revocation

Section 8.3.2.12 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.

11.3.2 Access Control

11.3.2.1 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.

11.3.2.2 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.

11.3.2.3 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.

11.3.2.4 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.

11.3.2.5 check_local_datawriter_match / check_local_datareader_match

When calling **check_local_datawriter_match** / **check_local_datareader_match**, the subscription_data and publication_data parameters are not set.

11.3.2.6 Revocation

Section 8.4.2.10 in the specification describes the mechanism for revoking permissions. *Security Plugins* do not implement this mechanism.

11.3.2.7 PermissionsToken

Table 10 in the specification mentions PermissionsToken as a new parameter in ParticipantBuiltinTopicData. *Security Plugins* 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.

11.3.2.8 check_remote_topic

Section 8.4.2.9.12 in the specification describes the **check_remote_topic()** operation, which receives a TopicBuiltinTopicData as an input. Instead of invoking this operation when the remote DomainParticipant creates a certain Topic, *Connext DDS* invokes this operation when discovering the first *DataWriter* or *DataReader* belonging to that Topic-DomainParticipant combination.

This distinction matters if the implementation of **check_remote_topic()** considers any of the QosPolicies within the TopicBuiltinTopicData structure. (The builtin plugins do not consider these QosPolicies.) For example, if Participant B creates two *DataReaders* of the same topic, Participant A will call **check_remote_topic()** only when it discovers the first *DataReader*. If the second *DataReader*'s DeadLineQosPolicy matches that of Participant B's TopicQos and the first *DataReader*'s DeadlineQosPolicy does not match, then **check_remote_topic()** will receive the wrong DeadlineQosPolicy as part of the input TopicBuiltinTopicData. This problem would occur only if **check_remote_topic()** considers the DeadlineQosPolicy when deciding whether to return TRUE or FALSE.

Appendix A Quick Reference: Governance File Settings

The following tables show common security objectives and the Governance file settings necessary to achieve them. The highlighted cells indicate settings that increase security.

Table A.1 Domains

Security Objective	Governance Parameter				
	allow_ unauthenticated_ join	enable_ join_ access_ control	discovery_ protection_ kind	liveliness_ protection_ kind	rtps_ protection_ kind
Baseline	T	F	N	N	N
Enable authentication & access control	F	T	N	N	N
Encrypt-then-MAC discovery data	T	F	E	N	N
MAC liveliness messages (protect builtin topic)	T	F	N	S	N
MAC entire RTPS packet (including header)	T	F	N	N	S
MAC entire RTPS packet (including header), protecting against tampering and replay by an authorized subscriber	T	F	N	N	SOA
MAC entire RTPS packet (including header) and encrypt-then-MAC data	T	F	N	N	S
MAC entire RTPS packet (including header) and encrypt-then-MAC data and metadata	T	F	N	N	S
All possible participant-level protections	F	T	N	N	EOA

Table A.2 Topics

Security Objective	Governance Parameter					
	enable_ discovery_ protection	enable_ liveliness_ protection	enable_read_ access_ control	enable_write_ access_control	metadata_ protection_ kind	data_ protection_ kind
Baseline	F	F	F	F	N	N
Enable authentication & access control	F	F	T	T	N	N
Encrypt-then-MAC discovery data	T	F	F	F	N	N
MAC liveliness messages (protect builtin topic)	F	T	F	F	N	N
MAC entire RTPS packet (including header)	F	F	F	F	N	N
MAC entire RTPS packet (including header), protecting against tampering and replay by an authorized subscriber	F	F	F	F	N	N
MAC entire RTPS packet (including header) and encrypt-then-MAC data	F	F	F	F	N	E
MAC entire RTPS packet (including header) and encrypt-then-MAC data and metadata	F	F	F	F	E	N
All possible participant-level protections	F	F	T	T	N	N

Legend:

T = TRUE
F = FALSE
N = NONE
E = ENCRYPT
EOA = ENCRYPT_WITH_ ORIGIN_ AUTHENTICATION
S = SIGN
SOA = SIGN_WITH_ORIGIN_AUTHENTICATION