<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Building with Security Enabled</strong></td>
<td>3</td>
</tr>
<tr>
<td>Windows</td>
<td>3</td>
</tr>
<tr>
<td>Using Microsoft vcpkg</td>
<td>3</td>
</tr>
<tr>
<td>Manual Build</td>
<td>4</td>
</tr>
<tr>
<td>Linux</td>
<td>5</td>
</tr>
<tr>
<td>macOS</td>
<td>5</td>
</tr>
<tr>
<td><strong>Architecture of the DDS Security Specification</strong></td>
<td>6</td>
</tr>
<tr>
<td><strong>Terms and Background Info</strong></td>
<td>6</td>
</tr>
<tr>
<td><strong>Required DDS Security Artifacts</strong></td>
<td>6</td>
</tr>
<tr>
<td>Per-Domain Artifacts</td>
<td>6</td>
</tr>
<tr>
<td>Per-Participant Artifacts</td>
<td>6</td>
</tr>
<tr>
<td><strong>Required OpenDDS Configuration</strong></td>
<td>7</td>
</tr>
<tr>
<td>DDS Security Configuration via PropertyQosPolicy</td>
<td>7</td>
</tr>
<tr>
<td>PropertyQosPolicy Example Code</td>
<td>7</td>
</tr>
<tr>
<td>Identity Certificates and Certificate Authorities</td>
<td>8</td>
</tr>
<tr>
<td>Identity, Permissions, and Subject Names</td>
<td>9</td>
</tr>
<tr>
<td>Examples in the OpenDDS Source Code Repository</td>
<td>9</td>
</tr>
<tr>
<td>Using OpenSSL Utilities for OpenDDS</td>
<td>9</td>
</tr>
<tr>
<td>Notes on OpenSSL CA Configuration</td>
<td>9</td>
</tr>
<tr>
<td>Creating Self-Signed Certificate Authorities</td>
<td>10</td>
</tr>
<tr>
<td>Creating Signed Certificates with an Existing CA</td>
<td>10</td>
</tr>
<tr>
<td>Signing Documents with SMIME</td>
<td>10</td>
</tr>
<tr>
<td><strong>Domain Governance Document</strong></td>
<td>10</td>
</tr>
<tr>
<td>Global Governance Model</td>
<td>10</td>
</tr>
<tr>
<td>Key Governance Elements</td>
<td>11</td>
</tr>
<tr>
<td>Domain List</td>
<td>11</td>
</tr>
<tr>
<td>Governance Configuration Types</td>
<td>11</td>
</tr>
<tr>
<td>Boolean</td>
<td>11</td>
</tr>
<tr>
<td>ProtectionKind</td>
<td>11</td>
</tr>
<tr>
<td>BasicProtectionKind</td>
<td>11</td>
</tr>
<tr>
<td>FnmatchExpression</td>
<td>11</td>
</tr>
<tr>
<td>Domain Rule Configuration Options</td>
<td>11</td>
</tr>
<tr>
<td>&lt;allow_unauthenticated_participants&gt; : Boolean</td>
<td>11</td>
</tr>
<tr>
<td>&lt;enable_join_access_control&gt; : Boolean</td>
<td>12</td>
</tr>
<tr>
<td>&lt;discovery_protection_kind&gt; : ProtectionKind</td>
<td>12</td>
</tr>
<tr>
<td>&lt;liveliness_protection_kind&gt; : ProtectionKind</td>
<td>12</td>
</tr>
<tr>
<td>Notes on OpenSSL CA Configuration</td>
<td>9</td>
</tr>
<tr>
<td>Creating Self-Signed Certificate Authorities</td>
<td>10</td>
</tr>
<tr>
<td>Creating Signed Certificates with an Existing CA</td>
<td>10</td>
</tr>
<tr>
<td>Signing Documents with SMIME</td>
<td>10</td>
</tr>
</tbody>
</table>
<rtps_protection_kind> : ProtectionKind 12

Topic Rule Configuration Options 12
  <topic_expression> : FnmatchExpression 12
  <enable_discovery_protection> : Boolean 12
  <enable_read_access_control> : Boolean 12
  <enable_write_access_control> : Boolean 12
  <metadata_protection_kind> : ProtectionKind 13
  <data_protection_kind> : BasicProtectionKind 13

Governance XML Example 13

Participant Permissions Document 14
  Key Permissions Elements 14
    Grants 14
    Subject Name 14
    Validity 14
    Allow / Deny Rules 15
    Default Rule 15
    Domain List 15
    Publish / Subscribe / Relay Rules (PSR rules) 15
    Topic List 15
    Partition List 16
    Data Tags List 16

Permissions XML Example 16

Implementation Status 18
Building with Security Enabled

OpenDDS 3.13 includes a beta implementation of the OMG DDS Security 1.1 specification. Building OpenDDS with security enabled requires the following dependencies:

1. Xerces-C++ v3
2. OpenSSL v1.0.2+ or v1.1 (1.1 is preferred)
3. Google Test v1.8 (only required if building OpenDDS tests)
   ○ If you are using OpenDDS from a git repository, Google Test is provided as a git submodule. Make sure to enable submodules with the --recursive option to git clone.
4. CMake (required if building OpenDDS tests and building Google Test and other dependencies from source).

General notes on using OpenDDS configure script with DDS Security:

● DDS Security is disabled by default, enable it with --security
● OpenDDS tests are enabled by default, disable them with --no-tests
   ○ Doing so skips the Google Test and CMake dependencies
   ○ If tests are enabled, the configure script can run CMake and build Google Test

Platform-specific details are in the following subsections.

Windows

Using Microsoft vcpkg

Microsoft vcpkg is a “C++ Library Manager for Windows, Linux, and macOS” which helps developers build/install dependencies. Although it is cross-platform, this guide only discusses vcpkg in terms of Windows.

As of this writing, vcpkg is only supported by Visual Studio 2015 Update 3 and Visual Studio 2017; other versions of Visual Studio should skip down below to the manual setup instructions.

1. If OpenDDS tests will be built, Install CMake or put the one that comes with Visual Studio on the PATH (see Common7\IDE\CommonExtensions\Microsoft\CMake).
2. Head over to https://github.com/Microsoft/vcpkg and follow the instructions to obtain vcpkg by cloning the repository and bootstrapping it.
3. Search for the openssl and xerces packages to ensure the proper version is available. As of this writing, only OpenSSL v1.0.2 is available (which is supported so this should be fine—but keep an eye out for updates).
   >vcpkg search openssl
   ...
   openssl-windows  1.0.2o  OpenSSL is an open source...
>vcpkg search xerces

... xerces-c 3.1.4-3  Xerces-C++ is a XML parser...

... Fetch and build the dependencies; by default, vcpkg targets x86 so be sure and force the x64 target if required by specifying it when invoking vcpkg install, as shown here:
> vcpkg install openssl:x64-windows xerces-c:x64-windows

5. If a debug build is desired, a symbolic link to the include directory needs to be created under the installed tree within the vcpkg hierarchy. For instance, since x64-windows was the target used in the previous step, the associated debug libraries are contained within vcpkg\installed\x64-windows\debug. This location does not contain an include directory though (which OpenDDS will look for), so create a link to the parents include directory:
> cd installed\x64-windows\debug
> mklink /J include ..\include

6. Configure OpenDDS by explicitly passing the openssl and xerces3 switches. As a convenience, it can be helpful to set an environment variable to store the path since it is the same location for both dependencies. In this case, the same debug directory mentioned above is used.
> set VCPKG_INSTALL=c:\path\to\vcpkg\installed\x64-windows\debug
> configure --security --openssl=%VCPKG_INSTALL% --xerces3=%VCPKG_INSTALL%

7. Compile with msbuild (or by opening the solution file in Visual Studio and building from there).
> msbuild /m DDS_TAOv2_all.sln

Manual Build

Note: for all of the build steps listed here, please double check that each package targets the same architecture (either 32-bit or 64-bit but not both) by compiling all dependencies within the same type of Developer Command Prompt.

Compiling OpenSSL (for further details: openssl wiki):
1. Install Perl and add it to the Path environment variable. For this guide, ActiveState is used.
2. Install Netwide Assembler (NASM). Click through the latest stable release and there is a win32 and win64 directory containing executable installers. The installer does not seem to update the Path environment variable, so a manual entry (%LOCALAPPDATA%\bin\NASM) was necessary in our case.
3. Download the required version of OpenSSL by cloning the repository.
4. Open a Developer Command Prompt (32-bit or 64-bit depending on the desired target architecture) and change into the freshly cloned openssl directory.
5. Run the configure script and specify a required architecture (perl Configure VC-WIN32 or perl Configure VC-WIN64A).
6. Run nmake.
7. Run nmake install.
Note: if the default OpenSSL location is desired, which will be searched by OpenDDS, open the Developer Command Prompt as an administrator before running the install. It will write to “C:\Program Files” or “C:\Program Files (x86)” depending on the architecture.

Compiling Xerces 3 (for further details: build instructions):
1. Download/extract the Xerces source files.
2. Create a cmake build directory and change into it (from within the Xerces source tree).
   >mkdir build
   >cd build
3. Run cmake with the appropriate generator. In this case Visual Studio 2017 with 64-bit is being used so:
   >cmake -G “Visual Studio 15 2017 Win64” ..
4. Run cmake again with the build switch and install target (this should be done in an administrator command-prompt to install in the default location as mentioned above).
   >cmake --build . --target install

Configuring and Building OpenDDS:
1. Change into the OpenDDS root folder and run configure with security enabled.
   a. If the default location was used for OpenSSL and Xerces, configure should automatically find the dependencies:
      >configure --security
   b. If a different location was used (assuming environment variables NEW_SSL_ROOT and NEW_XERCES_ROOT point to their respective library directories):
      >configure --security --openssl=%NEW_SSL_ROOT% --xerces3=%NEW_XERCES_ROOT%
2. Compile with msbuild (or by opening the solution file in Visual Studio and building from there).
   >msbuild /m DDS_TAOv2_all.sln

Linux

Xerces-C++ and OpenSSL may be installed using the system package manager, or built from source. If using the system package manager (that is, headers can be found under /usr/include), invoke the configure script with the “--security” option. If Xerces-C++ and/or OpenSSL are built from source or installed in a custom location, also provide the “--xerces=/foo” and “--openssl=/bar” command line options.

macOS

Xerces-C++ and OpenSSL may be installed using homebrew or another developer-focused package manager, or built from source. The instructions above for Linux also apply to macOS but the package manager will not install directly in /usr so make sure to specify the library locations to the configure script.
Architecture of the DDS Security Specification

The DDS Security specification defines plugin APIs for Authentication, Access Control, and Cryptographic operations. These APIs provide a level of abstraction for DDS implementations as well as allowing for future extensibility and version control. Additionally, the specification defines Built-In implementations of each of these plugins, which allows for a baseline of functionality and interoperability between DDS implementations. OpenDDS implements these Built-In plugins, and this document assumes that the Built-In plugins are being used. Developers using OpenDDS may also implement their own custom plugins, but those efforts are well beyond the scope of this document.

Terms and Background Info

DDS Security uses current industry standards and best-practices in security. As such, this document makes use of several security concepts which may warrant additional research by OpenDDS users:

- **Public Key Cryptography (including Private Keys)**
  - RSA
  - Elliptic Curve Cryptography
- **Public Key Certificates**
  - Certificate Authority (CA)
  - X.509 (and PEM format)
- **Signed Documents**

Required DDS Security Artifacts

**Per-Domain Artifacts**

These are shared by all participants within the secured DDS Domain:

- Identity CA
- Permissions CA (may be same as Identity CA)
- Governance Document
  - Signed by Permissions CA using its private key

**Per-Participant Artifacts**

These are specific to the individual Domain Participants within the DDS Domain:

- Identity Certificate and its Private Key
  - Issued by Identity CA (or a CA that it authorized to act on its behalf)
- Permissions Document
  - Contains a "subject name" which matches the participant certificate’s Subject
Signed by Permissions CA using its private key

Required OpenDDS Configuration

The following configuration steps are required to enable OpenDDS Security features:

1. Enable RTPS Discovery and the RTPS-UDP Transport; because DDS Security only works with these configurations, both must be specified for any security-enabled participant.

2. Enable OpenDDS security-features, which can be done two ways:
   a. Via API: “TheServiceParticipant->set_security(true);” or

DDS Security Configuration via PropertyQosPolicy

When the application creates a DomainParticipant object, the DomainParticipantQos passed to the create_participant() method now contains a PropertyQosPolicy object which has a sequence of name-value pairs. The following properties must be included to enable security. Except where noted, these values take the form of a URI starting with either the scheme “file:” followed by a filesystem path (absolute or relative) or the scheme “data:” followed by the literal data.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>dds.sec.auth.identity_ca</td>
<td>Certificate PEM file</td>
<td>Can be the same as permissions_ca</td>
</tr>
<tr>
<td>dds.sec.access.permissions_ca</td>
<td>Certificate PEM file</td>
<td>Can be the same as identity_ca</td>
</tr>
<tr>
<td>dds.sec.access.governance</td>
<td>Signed XML (.p7s)</td>
<td>Signed by permissions_ca</td>
</tr>
<tr>
<td>dds.sec.auth.identity_certificate</td>
<td>Certificate PEM file</td>
<td>Signed by identity_ca</td>
</tr>
<tr>
<td>dds.sec.auth.private_key</td>
<td>Private Key PEM file</td>
<td>Private key for identity_certificate</td>
</tr>
<tr>
<td>dds.sec.auth.password</td>
<td>Private Key Password (not a URI)</td>
<td>Optional, Base64 encoded</td>
</tr>
<tr>
<td>dds.sec.access.permissions</td>
<td>Signed XML (.p7s)</td>
<td>Signed by permissions_ca</td>
</tr>
</tbody>
</table>

PropertyQosPolicy Example Code

Below is an example of code that sets the DDS Participant QoS’s PropertyQoSPolicy in order to configure DDS Security.

```c
// DDS Security artifact file locations
const char auth_ca_file[] = "file:identity_ca_cert.pem";
const char perm_ca_file[] = "file:permissions_ca_cert.pem";
const char id_cert_file[] = "file:test_participant_01_cert.pem";
const char id_key_file[] = "file:test_participant_01_private_key.pem";
const char governance_file[] = "file:governance_signed.p7s";
```
Identity Certificates and Certificate Authorities

All certificate inputs to OpenDDS, including self-signed CA certificates, are expected to be an X.509 v3 certificate in PEM format for either a 2048-bit RSA key or else a 256-bit Elliptic Curve key for the prime256v1 curve.
Identity, Permissions, and Subject Names

The “subject_name” element for a signed permissions XML document must match the “Subject:” field provided by the accompanying Identity Certificate which is transmitted during participant discovery, authentication, and authorization. This ensures that the permissions granted by the Permissions CA do, in fact, correspond to the identity provided.

Examples in the OpenDDS Source Code Repository

C++ application that configures security QoS policies via command-line parameters:
https://github.com/objectcomputing/OpenDDS/blob/master/tests/DCPS/Messenger/publisher.cpp

Identity CA (along with private key):

Permissions CA (along with private key):
https://github.com/objectcomputing/OpenDDS/blob/master/tests/security/certs/permissions/permissions_ca_cert.pem

Participant Identity Certificate (along with private key):
https://github.com/objectcomputing/OpenDDS/blob/master/tests/security/certs/identity/test_participant_01_cert.pem

Governance XML Document (alongside signed document):
https://github.com/objectcomputing/OpenDDS/blob/master/tests/DCPS/Messenger/governance.xml

Permissions XML Document (alongside signed document):
https://github.com/objectcomputing/OpenDDS/blob/master/tests/DCPS/Messenger/permissions_1.xml

Using OpenSSL Utilities for OpenDDS

Notes on OpenSSL CA Configuration

To generate certificates using the openssl command, a configuration file gets passed into the ca option (see below for examples). Before proceeding, it may be helpful to review OpenSSL’s manpages to get help with the file format. In particular, configuration file format and ca command’s documentation and configuration file options.
An example OpenSSL CA-Config file used in OpenDDS testing can be found here: https://github.com/objectcomputing/OpenDDS/blob/master/tests/security/certs/identity/identity_ca_openssl.cnf

Creating Self-Signed Certificate Authorities

Generate a self-signed 2048-bit RSA CA:
openssl genrsa -out ca_key.pem 2048
openssl req -config openssl.cnf -new -key ca_key.pem -out ca.csr
openssl x509 -req -days 3650 -in ca.csr -signkey ca_key.pem -out ca_cert.pem

Generate self-signed 256-bit Elliptic Curve CA:
openssl ecparam -name prime256v1 -genkey -out ca_key.pem
openssl req -config openssl.cnf -new -key ca_key.pem -out ca.csr
openssl x509 -req -days 3650 -in ca.csr -signkey ca_key.pem -out ca_cert.pem

Creating Signed Certificates with an Existing CA

Generate a signed 2048-bit RSA certificate:
openssl genrsa -out cert_1_key.pem 2048
openssl req -new -key cert_1_key.pem -out cert_1.csr
openssl ca -config openssl.cnf -days 3650 -in cert_1.csr -out cert_1.pem

Generate a signed 256-bit Elliptic Curve certificate:
openssl ecparam -name prime256v1 -genkey -out cert_2_key.pem
openssl req -new -key cert_2_key.pem -out cert_2.csr
openssl ca -config openssl.cnf -days 3650 -in cert_2.csr -out cert_2.pem

Signing Documents with SMIME

Sign a document using existing CA & CA private key:
openssl smime -sign -in doc.xml -text -out doc_signed.p7s -signer ca_cert.pem -inkey ca_private_key.pem

Domain Governance Document

The signed governance document is used by the DDS Security built-in access control plugin in order to determine both per-domain and per-topic security configuration options for specific domains. For full details regarding the content of the governance document, see the OMG DDS Security specification section 9.4.1.2.

Global Governance Model

It's worth noting that the DDS Security Model expects the governance document to be globally shared by all participants making use of the relevant domains described within the governance document.
Even if this is not the case, the local participant will verify incoming authentication and access control requests as if the remote participant shared the same governance document and accept or reject the requests accordingly.

**Key Governance Elements**

**Domain List**
A list of domain ids and/or domain id ranges of domains impacted by the current domain rule.

**Governance Configuration Types**
The following types and values are used in configuring both per-domain and per-topic security configuration options. We summarize them here to simplify discussion of the configuration options where they’re used, found below.

**Boolean**
A boolean value indicating whether a configuration option is enabled or not. Recognized values are: 
{true, or false}

**ProtectionKind**
The method used to protect domain data (message signatures or message encryption) along with the ability to include origin authentication for either protection kind. Currently, OpenDDS doesn’t implement origin authentication. So while these options are recognized, the underlying configuration is unsupported. Recognized values are: {NONE, SIGN, ENCRYPT, SIGN_WITH_ORIGIN_AUTHENTICATION, or ENCRYPT_WITH_ORIGIN_AUTHENTICATION}

**BasicProtectionKind**
The method used to protect domain data (message signatures or message encryption). Recognized values are: {NONE, SIGN, or ENCRYPT}

**FnmatchExpression**
A wildcard-capable string used to match topic names. Recognized values will conform to POSIX fnmatch() function as specified in POSIX 1003.2-1992, Section B.6.

**Domain Rule Configuration Options**
The following XML elements are used to configure domain participant behaviors:

<allow_unauthenticated_participants> : Boolean
A boolean value which determines whether to allow unauthenticated participants for the current domain rule.
<enable_join_access_control> : Boolean
A boolean value which determines whether to enforce domain access controls for authenticated participants.

<discovery_protection_kind> : ProtectionKind
The discovery protection element specifies the protection kind used for the built-in DataWriter(s) and DataReader(s) used for secure endpoint discovery messages.

<liveliness_protection_kind> : ProtectionKind
The liveliness protection element specifies the protection kind used for the built-in DataWriter and DataReader used for secure liveliness messages.

<rtps_protection_kind> : ProtectionKind
Indicate the desired level of protection for the whole RTPS message. Currently, OpenDDS doesn’t implement whole-message protection operations, so the only supported “RTPS protection kind” is NONE. It is worth noting that very little RTPS data exists outside the “metadata protection” envelope (see topic rule configuration options), and so for most use cases topic-level “data protection” or “metadata protection” can be combined with discovery protection and/or liveliness protection in order to secure domain data adequately.

Topic Rule Configuration Options
The following XML elements are used to configure topic endpoint behaviors:

<topic_expression> : FnmatchExpression
A wildcard-capable string used to match topic names. See description above. A “default” rule to catch all previously unmatched topics can be made with: <topic_expression>*</topic_expression>

<enable_discovery_protection> : Boolean
Enables the use of secure discovery protections for matching user topic announcements.

<enable_read_access_control> : Boolean
Enables the use of access control protections for matching user topic DataReaders.

<enable_write_access_control> : Boolean
Enables the use of access control protections for matching user topic DataWriters.
<metadata_protection_kind> : ProtectionKind
Specifies the protection kind used for the RTPS SubMessages sent by any DataWriter and DataReader whose associated Topic name matches the rule's topic expression.

<data_protection_kind> : BasicProtectionKind
Specifies the basic protection kind used for the RTPS SerializedPayload SubMessage element sent by any DataWriter whose associated Topic name matches the rule's topic expression.

Governance XML Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   <domain_access_rules>
      <domain_rule>
         <domains>
            <id>0</id>
            <id_range>
               <min>10</min>
               <max>20</max>
            </id_range>
         </domains>
         <allow_unauthenticated_participants>FALSE</allow_unauthenticated_participants>
         <enable_join_access_control>TRUE</enable_join_access_control>
         <rtps_protection_kind>SIGN</rtps_protection_kind>
         <discovery_protection_kind>ENCRYPT</discovery_protection_kind>
         <liveliness_protection_kind>SIGN</liveliness_protection_kind>
      </domain_rule>
      <topic_access_rules>
         <topic_rule>
            <topic_expression>Square*</topic_expression>
            <enable_discovery_protection>TRUE</enable_discovery_protection>
            <enable_read_access_control>TRUE</enable_read_access_control>
            <enable_write_access_control>TRUE</enable_write_access_control>
            <metadata_protection_kind>ENCRYPT</metadata_protection_kind>
            <data_protection_kind>ENCRYPT</data_protection_kind>
         </topic_rule>
         <topic_rule>
            <topic_expression>Circle</topic_expression>
            <enable_discovery_protection>TRUE</enable_discovery_protection>
            <enable_read_access_control>FALSE</enable_read_access_control>
            <enable_write_access_control>TRUE</enable_write_access_control>
            <metadata_protection_kind>ENCRYPT</metadata_protection_kind>
            <data_protection_kind>ENCRYPT</data_protection_kind>
         </topic_rule>
         <topic_rule>
            <topic_expression>Triangle</topic_expression>
            <enable_discovery_protection>FALSE</enable_discovery_protection>
            <enable_read_access_control>FALSE</enable_read_access_control>
            <enable_write_access_control>TRUE</enable_write_access_control>
            <metadata_protection_kind>ENCRYPT</metadata_protection_kind>
            <data_protection_kind>ENCRYPT</data_protection_kind>
         </topic_rule>
      </topic_access_rules>
   </domain_access_rules>
</dds>
```
Participant Permissions Document

The signed permissions document is used by the DDS Security built-in access control plugin in order to determine participant permissions to join domains and to create endpoints for reading, writing, and relaying domain data. For full details regarding the content of the permissions document, see the OMG DDS Security specification section 9.4.1.3.

Key Permissions Elements

Grants
Each permissions file consists of one or more permissions grants. Each grant bestows access control privileges to a single subject name for a limited validity period.

Subject Name
Each grant’s subject name is intended to match against a corresponding identity certificate’s “subject” field. In order for permissions checks to successfully validate for both local and remote participants, the supplied identity certificate subject name must match the subject name of one of the grants included in the permissions file.

Validity
Each grant’s validity section contains a start date and an end date to indicate the period of time during which the grant is valid.
Allow / Deny Rules

Grants will contain one or more allow / deny rules to indicate which privileges are being applied. When verifying that a particular operation is allowed by the supplied grant, rules are checked in the order they appear in the file. If the domain, partition, and (when implemented) data tags for an applicable topic rule match the operation being verified, the rule is applied (either allow or deny). Otherwise, the next rule is considered. Special Note: If a grant contains any allow rule that matches a given domain (even one with no publish / subscribe / relay rules), the grant may be used to join a domain with join access controls enabled.

Default Rule

The default rule is the rule applied if none of the grant’s allow rules or deny rules match the incoming operation to be verified.

Domain List

Every allow or deny rule must contain a list of domain ids to which it applies. The syntax is the same as the domain list found in the governance document.

Publish / Subscribe / Relay Rules (PSR rules)

Every allow or deny rule may optionally contain a list of publish, subscribe, or relay rules bestowing privileges to publish, subscribe, or relay data (respectively). Each rule applies to a collection of topics in a set of partitions with a particular set of data tags. As such, each rules must then meet these three conditions (topics, partitions, and (when implemented) data tags) in order to apply to a given operation. These conditions are governed by their relevant subsection, but the exact meaning and default values will vary depending on the both the PSR type (publish, subscribe, relay) as well as whether this is an allow rule or a deny rule. Each condition is summarized below, but please refer to the OMG DDS Security specification for full details. OpenDDS does not currently support relay-only behavior and consequently ignores allow and deny relay rules for both local and remote entities. Additionally, OpenDDS does not currently support data tags, and so the data tag condition applied is always the “default” behavior described below.

Topic List

The list of topics and/or topic expressions for which a rule applies. Topic names and expressions are matched using POSIX fnmatch() rules and syntax. If the triggering operation matches any of the topics listed, the topic condition is met. The topic section must always be present for a PSR rule, so there is no default behavior.
Partition List

The partitions list contains the set of partition names for which the parent PSR rule applies. Similarly to topics, partition names and expressions are matched using POSIX fnmatch() rules and syntax. For “allow” PSR rules, the DDS entity of the associated triggering operation must be using a strict subset of the partitions listed for the rule to apply. When no partition list is given for an “allow” PSR rule, the “empty string” partition is used as the default value. For “deny” PSR rules, the rule will apply if the associated DDS entity is using any of the partitions listed. When no partition list is given for a “deny” PSR rule, the wildcard expression “*” is used as the default value.

Data Tags List

Data tags are an optional part of the DDS Security specification and are not currently implemented by OpenDDS. If they were implemented, the condition criteria for data tags would be similar to partitions. For “allow” PSR rules, the DDS entity of the associated triggering operation must be using a strict subset of the data tags listed for the rule to apply. When no data tag list is given for an “allow” PSR rule, the empty set of data tags is used as the default value. For “deny” PSR rules, the rule will apply if the associated DDS entity is using any of the data tags listed. When no data tag list is given for a “deny” PSR rule, the set of “all possible tags” is used as the default value.

Permissions XML Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <permissions>
    <grant name="ShapesPermission">
      <subject_name>emailAddress=cto@acme.com, CN=DDS Shapes Demo, OU=CTO Office, O=ACME Inc., L=Sunnyvale, ST=CA, C=US</subject_name>
      <validity>
        <!-- Format is CCYY-MM-DDThh:mm:ss[+|-]hh:mm The time zone may be specified as Z (UTC) or (+|-)hh:mm. Time zones that aren't specified are considered UTC. -->
        <not_before>2015-10-26T00:00:00</not_before>
        <not_after>2020-10-26T22:45:30</not_after>
      </validity>
      <allow_rule>
        <domains>
          <id>0</id>
        </domains>
      </allow_rule>
      <deny_rule>
        <domains>
        </domains>
      </deny_rule>
    </grant>
  </permissions>
</dds>
```
<id>0</id>
</domains>
<publish>
<topics>
  <topic>Circle1</topic>
</topics>
</publish>
<publish>
<topics>
  <topic>Square</topic>
</topics>
<partitions>
  <partition>A_partition</partition>
</partitions>
</publish>
<subscribe>
<topics>
  <topic>Square1</topic>
</topics>
</subscribe>
<subscribe>
<topics>
  <topic>Tr*</topic>
</topics>
<partitions>
  <partition>P1*</partition>
</partitions>
</subscribe>
</deny_rule>
<default>DENY</default>
</grant>
</permissions>
</dds>
Implementation Status

This beta release of DDS Security features in OpenDDS doesn't implement:

- Optional parts of the DDS Security v1.1 specification
  - Ability to write a custom plugin in C or in Java (C++ is supported)
  - Logging Plugin support
  - Built-in Logging Plugin
  - Data Tagging
- Use of RTPS KeyHash for encrypted messages
- Immutability of Publisher’s Partition QoS (see OMG Issue DDSSEC12-49)
- Transport-level hardening against malformed input
- Use of multiple plugin configurations (with different Domain Participants)
- Authentication and Access Control listener support (revoking access based on expiration)
- CRL (RFC 5280) and OCSP (RFC 2560) support
- Certain plugin operations not used by built-in plugins may not be invoked by middleware
- Origin Authentication
- Full-message encryption (sub-message and payload encryption is supported)
- PKCS#11 for certificates, keys, passwords
- Relay as a permissions “action” (Publish and Subscribe are supported)
- Legacy matching behavior of permissions based on Partition QoS (9.4.1.3.2.3.1.4 in spec)
- 128-bit AES keys (256-bit is supported)
- Configurability of Built-In Crypto’s key reuse (within the DataWriter) and blocks-per-session
- Signing (without encrypting) at the payload level, see OMG Issue DDSSEC12-59